Fakultät für Physik und Astronomie
Ruprecht-Karls-Universität Heidelberg

Diplomarbeit

im Studiengang Physik
vorgelegt von
Jan-Peter Loock
aus Bonn
2006

Evaluierung eines Floating Gate Analogspeichers für Neuronale Netze im Single-Poly UMC 180nm CMOS-Prozess

Die Diplomarbeit wurde von Jan-Peter Loock ausgeführt am
Kirchhoff-Institut für Physik
unter der Betreuung von
Prof. Karlheinz Meier

Evaluierung eines Floating Gate Analogspeichers für Neuronale Netze im Single-Poly UMC 180nm CMOS-Prozess:
In der vorliegenden Arbeit wird ein Floating Gate Transistor im Single-Poly UMC $0.18\mu m$ CMOS-Prozess entwickelt, der zur nichtflüchtigen Speicherung von analogen Spannungswerten in einem hardwarebasierten neuronalen Netz eingesetzt werden kann. Ein solches Netz besteht im Wesentlichen aus analogen Neuronen- und Synapsenschaltungen, in denen verschiedene Parameter gespeichert werden müssen. Aufgrund der großen Anzahl der Parameter muss diese Speicherung möglichst platz- und stromsparend, sowie im Idealfall nichtflüchtig erfolgen. Daher bietet sich die Verwendung eines Floating Gate Transistor an, dessen Betrieb jedoch die Kenntniss der auftretenden Tunnelströme und deren Modellierung in Schaltkreissimulationssoftware voraussetzt. Im ersten Teil der Arbeit wird ein Tunnelstrommodell entwickelt, das als Grundlage für den Entwurf einer Speicherzelle, basierend auf einem Floating Gate Transistor, dient. Nachfolgend wird die Implementierung verschiedener Testzellen auf einem Mikrochip in einem Standardprozess, der zur Herstellung der neuronalen Netzwerkchips verwendet werden kann, beschrieben. Die Vermessung des Mikrochips im zweiten Teil der Arbeit ermöglicht die Verifikation der Funktionsweise der Zelle und die Anpassung des entwickelten Tunnelstrommodells im verwendeten Herstellungsprozess, so dass eine vollständige Simulation dieses Floating Gate Transistors, eingebettet in Analogschaltungen, durchgeführt werden kann.

Evaluation of a Floating Gate Memory Cell in Single-Poly UMC 180nm CMOS-Process for Implementation in Neural Networks:
The presented thesis describes the development of a floating gate transistor in the Single-Poly UMC $0.18\mu m$ CMOS-Process, which will be used for non-volatile storage of analog voltages in a hardware-based neural network. This type of network consists of analog neuron and synapse circuits, for which a huge number of configurable parameters must be stored. Due to the great number of parameters a low-power, space-saving and non-volatile storage solution is necessary. Therefore, the use of floating gate transistors is advantageous. To operate these transistors, the behaviour of the tunneling currents must be known, allowing compact modeling and simulation in circuit simulation software. The first part of the thesis contains the developement of a tunneling model, which provides a basis for a memory cell design based on floating gate transistors. Subsequently the implementation of different test cells in a microchip, produced in the standard production process which is also used for the neural network chips, is described. The measurements of the microchip in the second part provide the verification of the storage functionality and the adaptation of the theoretical tunneling models. Thus, a complete simulation of the used floating gate transistor embedded in analog circuits is possible.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Motivation zur Entwicklung eines nichtflüchtigen Analogspeichers zur Implementierung in ein hardwarebasiertes neuronales Netz entstammt der Notwendigkeit in einem Netzwerk, bestehend aus analogen Synapsen- und Neuronenschaltungen, eine Vielzahl von Analogwerten zu speichern.
Um das Verhalten eines solchen Netzes zu untersuchen, müssen komplexe und großskalige Strukturen aufgebaut werden, die Millionen von Synapsen und Tausende von Neuronen enthalten. Erst durch die Verwendung von intergrierten Schaltkreisen auf Mikrochips können solche Strukturen realisiert werden. Ein aktuelles Beispiel stellt hierbei der in der Forschungsgruppe Electron Visions entwickelte zeitkontinuierlich arbeitende Spikey Chip dar [9]. Im Rahmen des FACETS-Projektes[1] sollen zukünftig mehrere solcher Chips mittels *Wafer Scale Integration* zuerst auf einem Wafer und anschließend über mehrere übereinander angeordnete Wafer miteinander verbunden werden.
Die Neuronen und Synapsen werden durch analoge Schaltungen simuliert und enthalten jeweils mehrere Spannungs- bzw. Stromwerte, die variabel sind und dauerhaft gespeichert werden müssen. Nichtflüchige Speicherung bedeutet, dass die Spannungen auch nach Abschalten der Stromersorgung erhalten bleiben und kein regelmäßiges Auffrischen der gespeicherten Werte nötig ist. Aufgrund der großen Anzahl sind klein dimensionierte und stromsparende Speicherzellen notwendig.

Verschiedene Spannungswerte sind hierbei zu berücksichtigen:

Zum einen gibt es Analogwerte, die dem biologischen Model entstammen. Dabei handelt es sich um Synapsen- und Neuronenparameter, wie beispielsweise Zeitkonstante, Umkehrpotentiale und Synapsengewichte [11]. Je nach Anforderung an die Geschwindigkeit, mit der ein Wert geändert werden soll, oder an die Langlebigkeit, die ein gespeicherter Wert aufweisen muss, können diese Pa-

[1] Fast Analog Computing with Emergent Transient States: Ein von der Europäischen Union unterstütztes Projekt, Grant no. IST-2004-15879

rameter auf einem Netzwerkchip auf unterschiedliche Art und Weise gespeichert werden.
Für sich schnell verändernde Werte, wie beispielsweise das Synapsengewicht ist eine Kapazität, die auf den gewünschten Spannungswert aufgeladen und anschließend isoliert wird, ausreichend. Doch führen Leckströme zum Entladen der Kapazität im mikrosekunden Bereich, so dass entweder nur kurze Speicherzeiten realisiert werden können oder die gespeicherten Werten regelmäßig wieder korrigiert werden müssen. Der Vorteil dieser Speichermethode liegt in der großen Schreibgeschwindigkeit und der einfachen Implementierung.
Zeitkonstanten und Umkehrpotentiale hingegen ändern sich im Vergleich zum Synapsengewicht nur relativ langsam und müssen folglich langlebiger gespeichert werden. Hierfür wird bislang ein zusammemgesetzter statischer Speicher aus DAC's[2] und Digitalspeicher[3] verwendet[9]. Diese beiden Komponenten weisen zusammen einen großen Platz- und Stromverbrauch auf und sind daher für den Einsatz in einem großskaligen integrierten Netzwerk von Nachteil.

Zum anderen gibt es Konfigurationsvariablen, die die zweite große Gruppe zu speichernder Analogwerte auf einem neuronalen Netzwerkchip bilden. Dazu gehören Kalibrations-, Referenz- und Biasspannungen. Diese Werte sollen dem System möglichst permanent und konstant zur Verfügung stehen. Insbesondere wäre eine Speicherung der Netzwerkkonfiguration auch nach dem Abschalten der Stromversorgung wünschenswert.

Um die Anforderungen der Nichtflüchtigkeit, des geringen Stromverbrauchs und der kleinskaligen Struktur zu erfüllen, bietet sich die Nutzung von Floating Gate Transistoren als Speicher an [3]. Floating Gate Transistoren werden bislang jedoch nur als Digitalspeicher in Form von kommerziellen Flashspeichern (beispielsweise USB-Speichersticks) in speziellen dual-poly Herstellungsprozessen eingesetzt. Es ist aber möglich, diese Tranistoren auch in analoge Schaltungen zu implementieren[10].
Die Evaluierung der Floating Gate Transistoren im Standard Single-Poly $0.18\mu m$-Prozess für die Anwendung in neuronalen Netzen und zum Abspeichern von analogen Werten ohne die Anwendung von DAC's und statischen SRAM-Speichern ist Ziel der vorliegenden Arbeit.
Dazu wird zunächst ein Tunnelstrommodell entwickelt, dass die Grundlage für den Entwurf einer Floating Gate Speicherzelle legt, die in verschiedenen Variationen auf einem Testchip implementiert wird. Die Vermessung des Mikrochips ermöglicht die Verifikation der Funktionsweise der Zellen und die Anpassung des theoretischen Modells, so dass abschließend durch einen Vergleich zwischen den gemessenen Daten und einer Computersimulation der Speicherzelle die Modellie-

[2]Digital to Analog Converter
[3]SRAM

rung des Floating Gate Transistors demonstriert werden kann.

Die vorliegende Arbeit ist folgendermaßen aufgebaut:
Nn Kapitel 2 wird der grundlegende Aufbau und die Funktionsweise eines Floating Gate Transistors diskutiert. Ein einfaches theoretisches Konzept beschreibt die Transportmechanismen zum Laden und Entladen.
Hierauf aufbauend wird in Kapitel 3 die Speicherzelle und deren Beschaltung erläutert.
In Kapitel 4 und 5 werden der entwickelte Testchip mit Schaltbild und Layout, sowie die Testplatine und der Testaufbau vorgestellt.
Nachfolgend wird die Aufnahme und die Auswertung der Messdaten, sowie deren Analyse dokumentiert.
Das 7. Kapitel setzt sich kritisch mit den gewonnenen Daten auseinander und beinhaltet die Schlußfolgerung.

Kapitel 2

Grundlagen

In diesem Kapitel werden zunächst die Grundlagen eines Floating Gate Transistors vorgestellt und dessen Lade- und Entlademechanismen mit Hilfe von Tunnelströmen diskutiert .

2.1 Der Floating Gate Transistor

2.1.1 Der Floating Gate Transistor in dual-poly Technologie

Der erste Floating Gate Transistor, der als nichtflüchtiger Speicher verwendet wurde, ist erstmals 1967 von Kahng und Sze vorgestellt worden.[1]

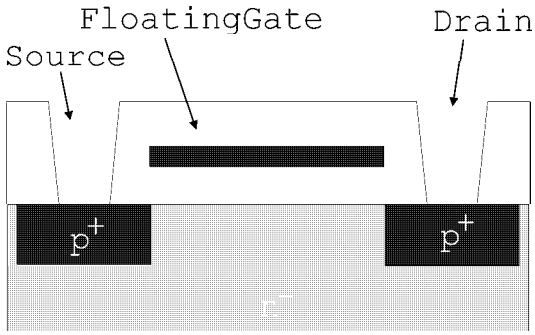

Abbildung 2.1: Floating Gate Transistor mit vollständig isoliertem Gate

Es handelt sich dabei um einen Standard MOS-Transistor, dessen Gate vollständig isoliert ist und dadurch ein Floating Gate bildet [1] (Abb. 2.1), d.h. das Gate liegt auf keinem fest definierten Potential, sondern die Spannung wird allein durch die Koppelkapazitäten bezüglich Drain, Source und Bulk auf der einen Seite und der auf dem Floating Gate befindlichen Ladung auf der anderen Seite bestimmt. Durch quantenmechanische Tunnelprozesse kann, bei entsprechender Beschaltung, Ladung durch die isolierende Siliziumdioxidschicht zwischen Bulk und Gate

auf das Floating Gate gelangen und den Zustand des Transistors verändern. So kann beispielsweise eine digitale Eins für einen leitenden Zustand (ausgebildeter Kanal im Transistor) und eine Null für einen nichtleitenden Zustand (kein ausgebildeter Kanal) dauerhaft gespeichert werden. Der Nachteil dieses Transistors ist die kurze Speicherzeit, da der Ladungstransfer allein durch Tunnelströme ein sehr dünnes Oxid voraussetzt, wodurch ein Leck-Tunnelsrom über das Gateoxid begünstigt wird.
1971 entwickelten Forham und Betchkowsky den FAMOS[1] Transistor, der mit einem dickeren Oxid ausgestattet werden konnte, da ein neuer Prozess zum Ladungstransfer über das SiO_2 entdeckt wurde. Dieser Vorgang wird Avalanche Injection genannt und in den folgenden Kapiteln erklärt. Ein solcher Speichertransistor wird als EPROM-Speicher bezeichnet und kann einmal beschrieben nur mit Hilfe von UV-Strahlung wieder gelöscht werden.
Die entscheidende Neuerung wurde 1972 von Tarui, durch erstmaliges Einbringen eines zweiten Gates in das Siliziumdioxid (Abb.2.2) eingeführt. Das neue Gate bildet das sogenannte Control Gate, welches für den Lade- und Entladevorgang entsprechend beschaltet werden kann. Das vollständig isolierte Gate bildet weiterhin das Floating Gate, auf welchem die transferierte Ladung dauerhaft gespeichert wird.

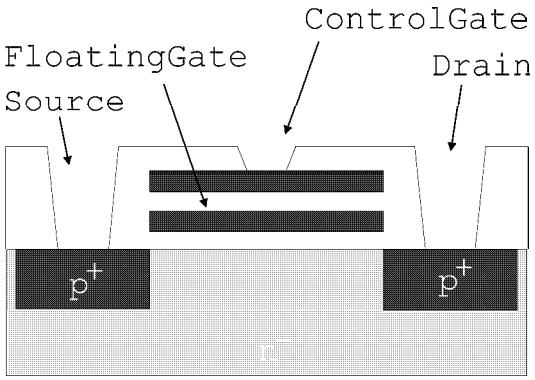

Abbildung 2.2: Floating Gate Transistor mit zusätzlichem Control Gate

Der Vorteil dieses zweiten Control Gates liegt darin, daß man den Transistor durch entsprechende elektrische Beschaltung mit Hilfe von Tunnelströmen, auch ohne UV-Einstrahlung, wieder entladen kann. Folglich wird dieser Transistor als EEPROM [2] bezeichnet.
In heutigen Standard Digitalspeichern (Flash-Speicher) werden EEPROMs im dual-poly Prozessen realisiert, d.h. beide Gates bestehen aus hochdotiertem Polysilizium [1].

[1] floating-gate avalanche injection metal-oxide-semiconductor
[2] Electrically Erasable Programmable Read Only Memory

2.1.2 Realisation des Floating Gate Transistors in single-poly Technologie

Da die neuronalen Netze im Single-Poly UMC 180nm CMOS Prozess realisiert werden, muss der Floating Gate Transistor ebenfalls in diesen Prozess implementiert werden können. In der single-poly Technologie steht jedoch kein zweites Polysilizium, das als zusätzliches Gate fungieren könnte, zur Verfügung.
Um dennoch einen Floating Gate Transistor nach oben beschriebenem Vorbild als Speicherzelle zu verwenden, wird er durch die folgende Ersatzschaltung (Abbildung 2.3) realisiert [2].

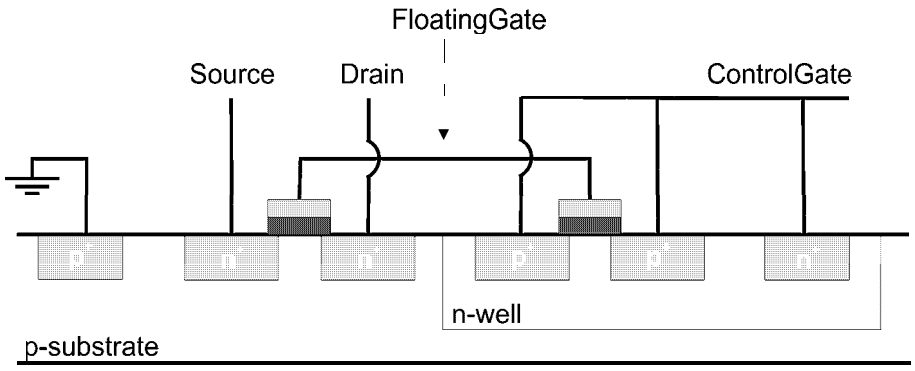

Abbildung 2.3: Realisierung eines Floating Gate Transistors in single-poly Technologie

Die abgebildete Schaltung zeigt einen nMOS-Transistor mit Source- und Drainanschluss, der im Weiteren auch als Auslesetransistor bezeichnet werden wird und einen pMOS Transistor in einer n-Well. Die beiden Gates sind fest verbunden und bilden das vollständig isolierte Floating Gate. Die Source- und Drainknoten des pMOS-Transistors (Tunneltransistors) sind mit dem Bulkanschluss der n-Well kurzgeschlossen und bilden das Control Gate. Der pMOS Transistor fungiert lediglich als Koppelkapazität zwischen Control Gate und Floating Gate. Der nMOS Transistor hingegen stellt einen Standardtransistor dar, dessen Auslesezustand durch die bekannten Transistorkennlinien in Abhängigkeit der nun nicht mehr direkt zugänglichen Gatespannung bestimmt ist.

2.1.3 Grundgleichungen des Floating Gate Transistors

Um die Floating Gate Spannung zu berechnen, müssen die Kapazitäten C_D, C_{CG}, C_B und C_S in Abbildung 2.4 berücksichtigt werden [3].

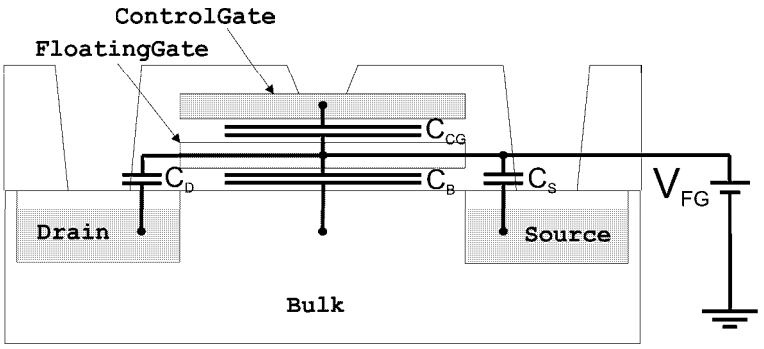

Abbildung 2.4: Schematischer Aufbau einer EEPROM Speicherzelle mit eingezeichneten Kapazitäten.

Ist auf dem Floating Gate keine Ladung gespeichert, kann man $Q = 0$ setzen. Die Ladung Q berechnet sich aufgrund der Beziehung $Q = C \cdot U$ wie folgt :

$$Q = 0 = C_{CG}(V_{FG} - V_{CG}) + C_S(V_{FG} - V_S) + C_D(V_{FG} - V_D) + C_B(V_{FG} - V_B).$$

Hierbei sind V_{FG} die Floating Gate Spannung, sowie V_S, V_D und V_B die Source, Drain und Control Gate Potentiale. C_{CG}, C_S, C_D und C_B sind die jeweiligen Kapazitäten gegenüber dem Floating Gate.
Durch Einführung der Gesamtkapazität $C_T = C_{CG} + C_S + C_D + C_B$ und des Koppelkapazitätskoeffizienten $\alpha_i = \frac{C_i}{C_T}$ mit $i = (CG, S, D, B)$ ergibt sich:

$$V_{FG} = \alpha_{CG}V_{CG}\alpha_D V_D \alpha_S V_S \alpha_B V_B$$

Liegen Source und Bulk während des Lesevorgangs auf Masse, so vereinfacht sich die Gleichung:

$$V_{FG} = \alpha_{CG}(V_{CG} + f \cdot V_{DS}) \text{ mit } f = \frac{\alpha_D}{\alpha_{CG}} = \frac{C_D}{C_{CG}}$$

Nun können die Standardgleichungen für einen konventionellen MOS-Transistor [4] verwendet werden. Es müssen lediglich die MOS-Gatespannung V_{GS} durch die Floating Gate Spannung V_{FG} ersetzt und transistorspezifische Paramter wie Schwellspannung V_{th} und Transkonduktanz β durch Werte abhängig von der Control Gate Spannung ausgetauscht werden:

$$V_{th}^{FG} = \alpha_{CG} \cdot V_{th}^{CG} \text{ und } \beta^{FG} = \frac{\beta^{CG}}{\alpha_{CG}}$$

Die Standardgleichung zur Berechnung des Drainstroms eines MOSFET-Transistors ist die Sah-Gleichung[3]:

[3]Eine detalierte Herleitung ist in [4] zu finden

$i_D = \beta^{CG}\left[(V_{GS} - V_{th}^{CG})V_{DS} - \frac{V_{DS}^2}{2}\right]$ für $V_{GS} \geq V_{th}$ und $V_{DS} \leq (V_{GS} - V_{th})$

Für den Drainstrom eines Floating Gate Transistors ergibt sich mit den oben angeführten Betrachtungen folgende Beziehung:

$$\Longrightarrow i_D = \beta^{FG}\left[(V_{FG} - V_{th}^{FG})V_{DS} - \frac{V_{DS}^2}{2}\right]$$
$$\Longleftrightarrow i_D = \frac{\beta^{CG}}{\alpha_{CG}}\left[(\alpha_{CG}(V_{CG} + f \cdot V_{DS}) - \alpha_{CG}V_{th}^{FG})V_{DS} - \frac{V_{DS}^2}{2}\right]$$
$$\Longleftrightarrow i_D = \beta^{CG}\left[(V_{CG} - V_{th}^{CG})V_{DS} - (f - \frac{1}{2\alpha_{CG}})V_{DS}^2\right]$$

für $V_{DS} \leq (V_{FG} - \alpha_{CG}V_{th}^{CG}) \Longleftrightarrow V_{DS} \leq \alpha_{CG}(V_{CG} + fV_{DS} - V_{th}^{CG})$

Ist auf dem Floating Gate die Ladung Q gespeichert ($Q \neq 0$), bleiben die obigen Annahmen richtig. Bei der Berechnung von V_{FG} und V_{th}^{CG} muss nun die zusätzliche Ladung Q berücksichtigt werden:

$V_{FG} = \alpha_{CG}V_{CG}\alpha_D V_D \alpha_S V_S \alpha_B V_B + \frac{Q}{C_T}$
$V_{th}^{CG} = V_{th0}^{CG} - \frac{Q}{C_{CG}}$ mit Schwellspannung V_{th0} für $Q = 0$.

Setzt man diese Erweiterungen in die Sah-Gleichung für den Drainstrom ein, so ergibt sich für einen geldadenen Floating Gate Transistor folgende Beziehung:

$$i_D = \beta^{CG}\left[(V_{CG} - V_{th0}^{CG} + \frac{Q}{C_{CG}})V_{DS} - (f - \frac{1}{2\alpha_{CG}})V_{DS}^2\right]$$

für $V_{DS} \leq \alpha_{CG}(V_{CG} + fV_{DS} - V_{th0}^{CG} + \frac{Q}{C_{CG}})$

Zur Berechnung des Drainstroms müssen die Koppelkapazitätskoeffizienten α_i bestimmt werden. Da das Floating Gate jedoch vollständig isoliert und damit von außen nicht zugänglich ist, können die Koppelkapazitäten nicht direkt aus Messungen gewonnen werden. Die übliche Methode zur Bestimmung des wichtigsten Koppelkapazitätskoeffizienten α_{CG} ist die Berechnung aus dem Verhältnis zwischen Schwellspannung und Transkonduktanz bei der Vermessung eines Dummy Floating Gate Transistors und einer Floating Gate Speicherzelle. Bei kommerziellen Speicherzellen ist dies die übliche Verfahrenweise. Allerdings sind dabei eine Vielzahl von Messungen möglich und es werden identische Dimensionen der verschiedenen Transistoren (matching) angenommen, die jedoch immer herstellungsbedingten Schwankungen unterlegen sind. Folglich ist die Genauigkeit dieser Methode begrenzt.

2.2 Physikalische Lade- und Entlademechanismen

2.2.1 Fowler-Nordheim Tunneling

Der quantenmechanische Tunneleffekt ermöglicht es einem Teilchen, mit endlicher Wahrscheinlichkeit eine Potentialbarriere zu durchdringen, die energetisch höher ist als die kinetische Energie dieses Teilchens. In der klassischen Mechanik würde das Teilchen an der Barriere vollkommen reflektiert und die Überwindung der Barriere wäre verboten. Die Theorie zum Tunnnelprozess geht aus der Lösung der Schrödinger Gleichung hervor [8].
Das Bänderdiagramm (Abb. 2.5) zeigt die Energieniveaus eines MOSFET-Transistors, d.h die Energiebänder zwischen dem stark n-dotierten Polysilizium des Gates, der Isolationsschicht und dem p-dotierten Substrat.

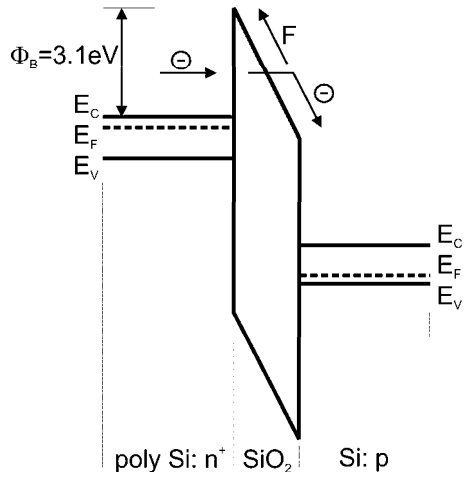

Abbildung 2.5: Energieniveaus in einem pMOS Transistor bei Anlegen einer äußeren Spannung und der Fowler-Nordheim Tunnelstrom durch die trapezförmige SiO_2 Potentialbarriere.

Die Tunnelwahrscheinlichkeit der Elektronen hängt zum einen von der Höhe, Breite und Form der Tunnelbarriere, sowie zum anderen von der Verteilung der besetzten Zustände im Festkörper ab [3]. Verwendet man ein Freies Elektronengas zur Berechnung der Energieverteilung der Elektronen und die Wentzel-Kramers-Brillouin Näherung zur Berechnung der Tunnelwahrscheinlichkeit, so ergibt sich für die Fowler-Nordheim Tunnelstromdichte (FNT):

$$J_{FN} = \frac{\pi kTC_{FN}}{\sin(\pi kTC_{FN})} A_{FN} A' F_{OX}^2 \cdot \exp(-\frac{B_{FN}}{F_{OX}})$$

$$A_{FN} = \frac{q^3 m_{Si}}{16\Pi^2 \hbar m_{OX} \Phi_0}$$

$$B_{FN} = \frac{4\sqrt{2m_{OX}\Phi_0^3}}{3q\hbar}$$
$$C_{FN} = \frac{2\sqrt{2m_{OX}\Phi_0}}{q\hbar F_{OX}}$$

wobei:
F_{OX}: Elektrisches Feld im Oxid, T: Temperatur, q: Elementarladung, m_{Si}: Effektive Masse eines Elektrons in $Si0_2$, $\hbar$: Planck'sches Wirkungsquantum, Φ_0: Höhe der Potentialbarriere

Der Faktor $\frac{\pi kTC_{FN}}{\sin(\pi kTC_{FN})}$ beschreibt die Temperaturabhängigkeit des Tunnelstroms und wird in der Literatur aufgrund der relativ schlechten Übereinstimmung mit gemessenen Daten meist vernachlässigt. In der weiteren Betrachtung wird er gleich eins gesetzt.
A_{FN}, B_{FN}, C_{FN} sind Koeffizienten, die aus physikalischen Konstanten berechnet werden können. In der Praxis wird A_{FN} jedoch auch als Fitting-Parameter verwendet, um die Abweichungen zwischen Theorie und gemessenen Daten zu minimieren. Erklärt wird die Abweichung durch herstellungsbedingte Unterschiede im Halbleiter, wie beispielsweise Abweichungen an der Si/SiO_2-Grenzschicht, die zu unterschiedlichen Formen der Potentialbarriere führen können, analytisch aber nicht vorhersagbar sind. Um zwischen dem theoretisch berechneten Faktor A und dem gefitteten Wert A unterscheiden zu können, wird A' als zusätzlicher Fitfaktor eingeführt und in der Theorie zunächst gleich eins gesetzt: $A' = 1$.

Das elektrische Feld durch das Oxid in Abhängigkeit der angelegten Spannung wird wie folgt berechnet:

$$F_{OX} = \frac{V - \Psi_C - \Psi_A - V_{FB}}{t_{OX}}$$

wobei:
t_{OX}: Oxiddicke, V_{FB}: Flatband Spannung, $\Psi_{C/A}$: Kontaktspannung an der Anode oder Kathode, V: Spannung über das Oxid

Die Flatband Spannung wird durch folgende Gleichung bestimmt:

$$V_{FB} = \Phi_{MS} - \frac{Q_{PA}}{C_{OX}}$$

wobei:
Φ_{MS}: Austrittsarbeitsdifferenz, Q_{PA}: Parasitäre Ladung am Si/SiO_2-Übergang, C_{OX}: Oxikapazität

Zur Berechnung der Kontaktspannungen unterscheidet man grundsätzlich zwischen drei Zuständen in denen sich der Halbleiter befinden kann: (i) Akkumula-

tion, (ii) Inversion und (iii) Verarmung.

(i) Ist der Halbleiter in Akkumulation, so kann als Näherung für die Kontakspannungen:

$$\Psi_{C/A} = 0$$

angenommen werden. Je höher die Dotierung des Anoden-/Kathodenmaterials, desto besser ist diese Näherung.

(ii) Im Verarmungszustand lässt sich $\Psi_{C/A}$ wie folgt aus dem Gausschen Gesetz am Si/SiO_2-Übergang abschätzen:

$$\Psi_{C/A} = \frac{\epsilon_{OX}^2}{2qN_{C/A}\epsilon_{Si}} F_{OX}^2$$

$\epsilon_{Si/OX}$*:Permeabilitätszahl von* Si/SiO_2, $N_{C/A}$ *Dotierung des Kathoden- oder Anodenmaterials*

(iii) Bei Inversion ergibt sich die Kontaktspannung aus folgender Beziehung:

$$\Psi_{C/A} = 2\Psi_{Fermi,p/n} + 6\Psi_{thermisch}$$

$$\Psi_{thermisch} = \frac{k_B T}{q}$$

$\Psi_{Fermi,p} = \Psi_{thermisch} \ln \frac{N_{C/A}}{n_i}$ für einen p-dotierten Halbleiter

$\Psi_{Fermi,n} = \Psi_{thermisch} \ln \frac{n_i}{N_{C/A}}$ für einen n-dotierten Halbleiter

k_B*: Boltzmann Konstante,* n_i*: intrinsische Ladungsträgerkonzentration*

2.2.2 Direct Tunneling

Wird zwischen der Kathode und Anode des Si/SiO_2-Übergangs eine große Spannung angelegt, so verformt sich die näherungsweise rechteckige Potentialbarriere

durch die Verschiebung der Energieniveaus in eine trapezförmige Barriere. Die auftretende Tunnelstromdichte wurde im vorangegangenen Abschnitt abgeschätzt (Fowler-Nordheim Tunnelstrom).
Wird die Potentialdifferenz kleiner, so ist die Form der Barriere ein Rechteck und das FNT-Modell ist nicht mehr korrekt (Abbildung 2.6). Man spricht in diesem Fall von Direct Tunneling.

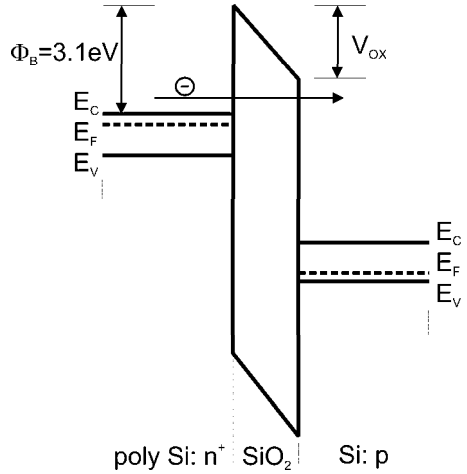

Abbildung 2.6: Direct Tunneling durch die SiO_2 Potentialbarriere in einem pMOS

Zur verbesserten Modellierung des Tunnelstroms bei kleineren Spannungen ist die Einführung eines Vorfaktors zur FNT-Tunnelstromdichte erforderlich [5].

$$J = J_{FN} * (1 - \sqrt{\frac{\Phi_0 - V_{OX}}{\Phi_0}})^{-2} \cdot \exp\left(\frac{B_{FN}}{F_{OX}} \cdot (\frac{\Phi_0 - V_{OX}}{\Phi_0})^{3/2}\right)$$

Die Lebensdauer der gespeicherten Ladung auf dem Floating Gate Transistor hängt von der Stärke des Direct Tunneling Prozesses ab, da dieser Strom zu einer langsamen Entladung des Floating Gates führt. Die Stromstärke hängt sehr stark von der Oxiddicke der verwendeten Transistoren ab. Deshalb wird durch diesen Prozess eine untere Schranke bezüglich der geometrischen Größe des verwendeten Floating Gate Transistors, je nach gewünschter Lebensdauer vorgegeben.

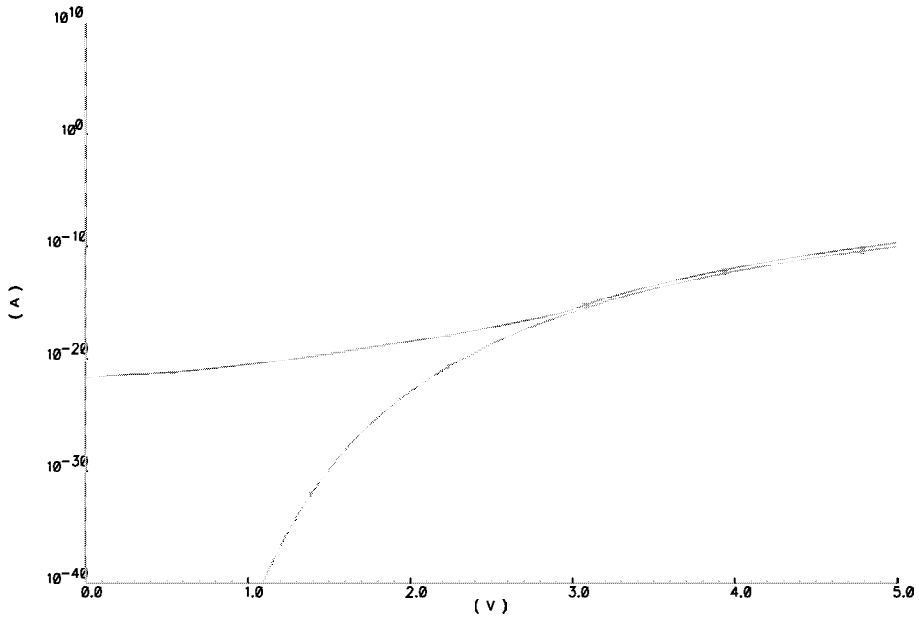

Abbildung 2.7: Schematischer Vergleich zwischen Fowler-Nordheim Tunnelstrom mit Direct Tunneling J (grün) und Fowler-Nordheim Tunnelstrom J_{FN} (blau) ohne Direct Tunneling bei einer Oxiddicke von 4.2nm.

Müsste nur der Fowler-Nordheim Tunnelstrom durch die trapezförmige Barriere berücksichtigt werden (blau), wäre der Leckstrom bei kleinen Spannungen zu vernachlässigen. Da der Anteil des Tunnelstroms durch die rechteckige Barriere (blau) bei kleineren Spannung wächst, dominiert er den Leckstrom bei kleinen Spannungen.

2.2.3 Channel Hot Electron Injection

Channel Hot Electron Injection (CHE) ist der am häufigsten verwendete Mechanismus in Flash-Speichern, um Ladung auf das Floating Gate zu transferieren [3]. Ein Elektron, das sich in einem MOSFET durch den ausgebildeten Kanal bewegt, wird durch das vertikale Feld beschleunigt und gewinnt an kinetischer Energie. Gleichzeitig verliert es aber auch Energie durch Stöße mit den akustischen und optischen Phononen im Kristallgitter des Halbleiters. Bis zu einer Feldstärke von ungefähr $100\frac{kV}{cm}$ handelt es sich dabei um ein quasistatisches Gleichgewicht, d.h. das Elektron gibt im Mittel genau soviel Energie an die Gitteratome ab, wie es durch das elektrische Feld gewinnt. Oberhalb dieser Feldstärken beginnt die kinetische Energie zu wachsen (Hot Electrons). Wird die Energie größer als die Höhe der Potentialbarriere, ist das Elektron in der Lage die SiO_2-Schicht zu überwinden und das Polysilizium des Floating Gates zu erreichen, wenn es in Richtung des Gates gestreut wird (Abb. 2.8).

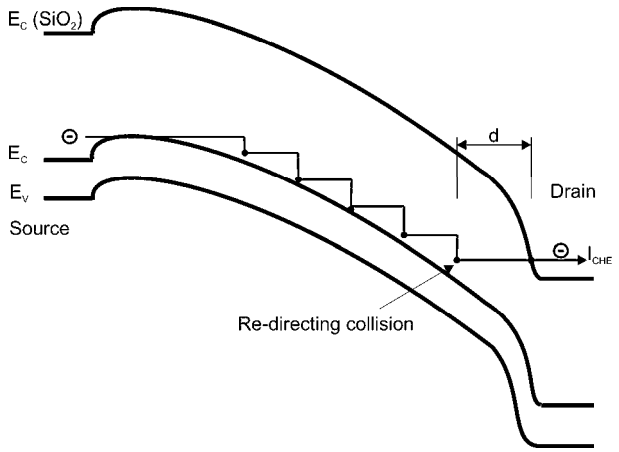

Abbildung 2.8: Schematischer Weg der Hot Electrons in einem MOSFET-Transistor [3]

Zur Modellierung von Hot Electrons wird häufig das 'lucky electron model' verwendet. Dieses Modell basiert auf der Annahme, dass ein Elektron eine freie Wegstrecke d zurücklegen muss, um vom lateralen elektrischen Feld E_M (wird über den Kanal als konstant angenommen) genügend beschleunigt zu werden, damit die Potentialbarriere der Höhe Φ_0 überwunden werden kann. Die Wahrscheinlichkeit P_0 dafür berechnet sich mit Hilfe der mittleren freien Weglänge von Hot Electrons λ wie folgt:

$$P_0 = \exp -\frac{d}{\lambda} \text{ mit } d = \frac{\Phi_0}{qF_{lateral}}$$

Daraus berechnet sich der 'Channel Hot Electron'- Strom:

$$I_{CHE} = C \cdot I_{DS} \cdot exp(-\frac{\Phi_0}{q \cdot E_M \lambda})$$

C ist eine Konstante, die aus Messungen bestimmt werden muss. E_M ist der Maximalwert des lateralen elektrischen Feldes am Drainknoten:

$$E_M = \frac{V_{DS} - V_{DS,SAT}}{\Delta L}$$

$$\Delta L = K_L t_{OX}^{1/3} X_J^{1/2}$$

$$V_{DS,SAT} = \frac{(V_G - V_T) L_E E_{SAT}}{V_G - V_T + L_E E_{SAT}}$$

Das schematische Verhalten des CHE-Stroms in einem nMOS-Transistor (Länge:

180nm, Breite: 240nm) mit einer Source-Drain Spannung von 2.5 V gegenüber dem Substrat in Abhängigkeit der Gatespannung ist in Abbildung 2.9 zu sehen. Bei einer Gatespannung von ungefähr 2 Volt geht der Strom in eine Art Sättigung. Ab dieser Spannung ist der Drainstrom maximal. Ebenso ist die Attraktivität des Gates für Elektronen, die in diese Richtung gestreut werden und genügend Energie zur Überwindung der Potentialbarriere haben, auf einem hohen Niveau, so dass der Großteil dieser Elektronen auf das Gate treffen. Durch weitere Erhöhung des Gatepotentials werden aber keine zusätzlichen Elektronen mehr gestreut (der Drainstrom ist beim Maximum) und der CHE-Strom nimmt nicht weiter zu:

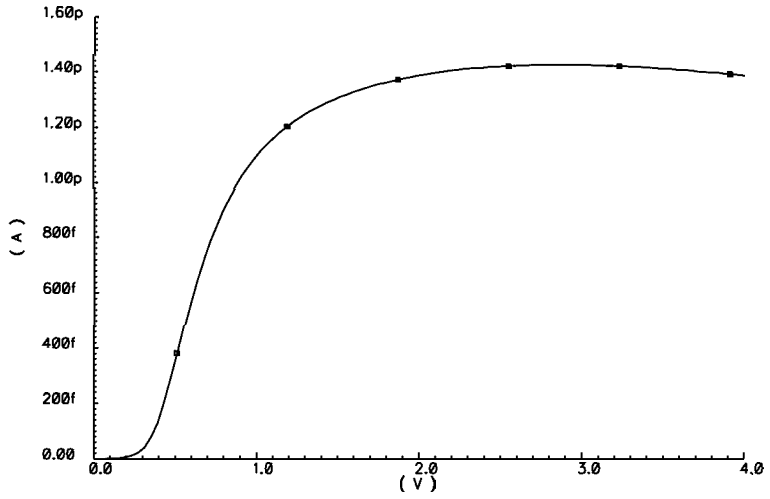

Abbildung 2.9.: Schematisches Verhalten des Hot Electron Stroms in Abhängigkeit der Gatespannung.

Kapitel 3

Implementierung des Floating Gate Transistors

In diesem Kapitel wird das Design und die Beschaltung des Floating Gate Transistors für den Einsatz als Speicherzelle vorgestellt und erklärt.

3.1 Funktionsweise des Floating Gate Transistors

Um einen Floating Gate Transistor als Speichermedium in neuronalen Netzen verwenden zu können, ist man auf die korrekte Modellierung der Tunnelströme in geeingneten Modellen angewiesen. Zur Einbettung der Speicherzellen in analoge Schaltungen müssen sie in Simulationsprogrammen wie bspw. Spectre von der Firma Cadence simuliert und getestet werden können.
Die in Kaptiel 2 erwähnten Tunnelströme sind nicht in den normalen Spice-Tranistormodellen enthalten. Zur Einbindung dieser Ströme bieten sich VerilogA[1] Elemente an, die als strom- und/oder spannungsgesteuerte Stromquellen fungieren und in den Schaltplan eingebaut werden. In den Elementen sind die theoretischen Überlegungen zu den Tunnelströmen in analytischer Form enthalten.
Es gibt nun verschiedene Möglichkeiten, den Floating Gate Transistor zu beschalten, sowie das Größenverhältnis der Transistoren zu wählen, um Tunnelströme zum Laden bzw. Entladen über den nMOS oder den pMOS zu induzieren [2].

Für die Anwendung als Speicherzelle werden die folgenden drei Phasen unterschieden:

- Laden durch Tunnelströme

[1]Programmiersprache, die eine Untermenge von Verilog-AMS der Firma Cadence darstellt und zur Beschreibung analoger Komponenten und Systeme genutzt werden kann

- Entladen durch Tunnelströme

- Entlade durch Hot-Electron Injection

3.1.1 Laden des Floating Gates durch Tunnelströme

Schaltet man das Control Gate auf ein hohes Potential V_{CG} gegenüber den auf Masse gelegten Drain- und Sourceknoten V_{Drain}, V_{Source} (Abb. 3.1), so folgt das Floating Gate Potential V_{FG} durch die kapazitive Kopplung über C_{CG} dem Floating Gate auf eine höhere Spannung, da es keinen festen Bezugspunkt hat. Die sich einstellende Spannung des Floating Gates hängt nun von dem Kapazitätsverhältnissen am pMOS und am nMOS ab, d.h. zwischen C_{CG} auf der einen Seite und C_{Drain}, C_{Source} und C_{Bulk} auf der anderen. Dieses Kapazizätsverhältnis ist durch das Design der Transistoren, also Länge und Breite des Auslese- als auch des Tunneltransistors, frei wählbar. Da man nun im Lademechanismus einen möglichst großen Tunnelstrom über das Gateoxid des pMOS induzieren möchte, ist es notwendig $C_{nMOS} > C_{pMOS}$ zu wählen, da so die Kopplung über C_{CG} geringer ist und eine kleinere Spannung am Control Gate zu einer größeren Potentialdifferenz am Tunneltransistor führt. Zusätzlich exisitert ein Entladestrom über das Oxid des nMOS, der dem Aufladen entgegenwirkt. Um diesen Strom so klein wie möglich zu halten, ist ebenfalls das beschriebene Kapazitätsverhältnis günstig.

C_{nMOS} größer als C_{pMOS} heißt:

$\Rightarrow$ kleinere ControlGate Spannungen zur Zündung des Tunnelstroms nötig

$\Rightarrow$ größere Tunnelströme $\Rightarrow$ größere Beschreibgeschwindigkeit

Die Elektronen, die im Ladezustand von dem Floating Gate auf das Control Gate tunneln, bewirken nun eine zunehmend positive Spannung des Floating Gates. Mit dieser steigenden Spannung sinkt die Potentialdifferenz am Gateoxid des pMOS und der Tunnelstrom nimmt langsam ab, bis er zum Erliegen kommt. Schaltet man anschließend das ControlGate ab, so koppelt das Floating Gate nicht mehr zum Ausgangszustand zurück, sondern bleibt auf einem höheren Potential. Dieser Zustand bleibt solange gespeichert, bis er durch die nachfolgende Beschaltung gelöscht wird oder langsam durch Leckströme zerfällt.

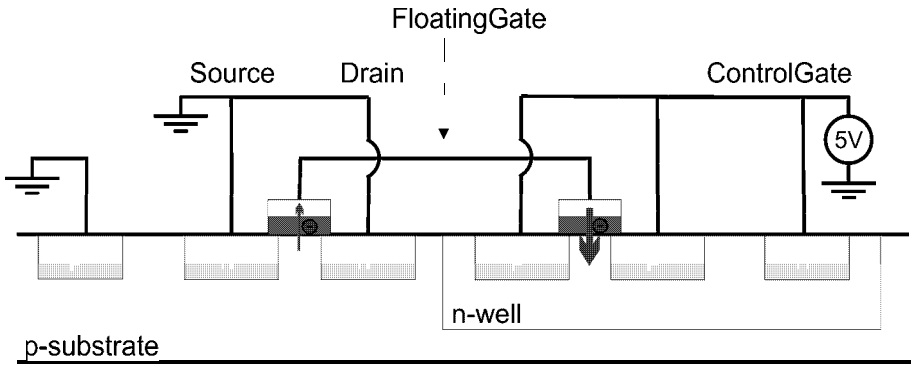

Abbildung 3.1: Beschaltung der Zelle zum Laden des Floating Gates mit Hilfe von Fowler-Nordheim und Direct Tunneling

Das Mitkoppeln des Floating Gates mit dem Control Gate hat den Effekt, dass man als Ladespannung V_{CG} eine Spannung weit höher als die theoretische Durchbruchsspannung der Transistoren nutzen kann, da nur die Spannungsdifferenz zwischen Control- und Floating Gate diese Spannung nicht überschreiten darf. Bei den in diesem Prozess verwendeten 1.8 Volt Transistoren liegt die angegebene Durchbruchsspannung bei 4.2 Volt.
Als Lade- und Entladespannungen werden Pulse von 5 Volt verwendet, so liegt die maximal auftretende Oxidspannung bei den gewählten Verhältnissen bei ungefähr 3.8 bis 4 Volt.

3.1.2 Entladen des Floating Gates durch Tunnelströme

Zum Entladen des Floating Gates bieten sich grundsätzlich zwei Prozesse an. Zum einen wieder Fowler-Nordheim und Direct Tunneling diesmal allerdings in die entgegengesetzte Richtung und zum anderen Hot Electron Injection.
Es sei nun das Floating Gate auf einem positiven Potential bezüglich Masse und das Control Gate aus Masse, so kann durch Beschaltung von Drain und Source auf ein ebenfalls positives und weitaus höheres Potential durch die kapazitive Kopplung über C_{nMOS} die Floating Gate Spannung weiter erhöht werden (Abb. 2.11). Je größer die kapazitive Kopplung, desto höher ist V_{FG} gegenüber V_{CG}. Folglich stellt sich eine Potentialdifferenz über das SiO_2 des pMOS sein und ein Tunnelstrom, diesmal vom Floating Gate zum Control Gate, setzt ein. Auch hier erhöht also ein Kapazitätsverhältnis $C_{nMOS} > C_{pMOS}$ den Tunnelstrom und die o.g. Annahme ist somit auch für den Löschvorgang mit Hilfe der Tunnelströme richtig:

C_{nMOS} größer als C_{pMOS} heißt:

$\Rightarrow$ kleinere Drain-/Sourcespannungen zur Zündung des Tunnelstroms nötig

$\Rightarrow$ größere Tunnelströme $\Rightarrow$ größere Löschgeschwindigkeit

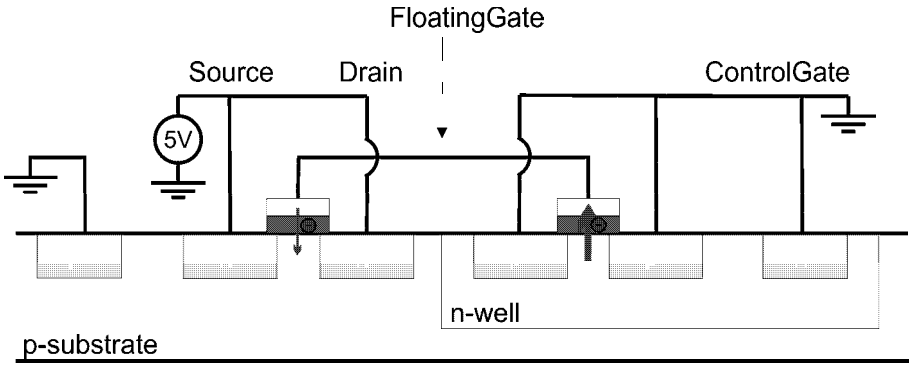

Abbildung 3.2: Beschaltung der Zelle zum Entladen des Floating Gates mit Hilfe von Fowler-Nordheim und Direct Tunneling

Die Abnahme der Floating Gate Spannung durch den Tunnelstrom bewirkt wieder eine Abnahme der Potentialdifferenz über die Tunnelbarriere und der Prozess des Tunnelns schaltet sich automatisch ab. Wird nun Source und Drain wieder auf Masse gelegt, so stellt sich das Potential des Floating Gates, je nach Dimensionen, auf einen Wert unterhalb des Ausgangswertes ein.

3.1.3 Entladen des Floating Gates durch Hot Electron Injection

Zum Entladen des Floating Gates durch Hot Electron Injection bietet sich der nMOS-Transistor an, da nun Source und Drain unterschiedlich geschaltet werden müssen (Abb. 2.12).

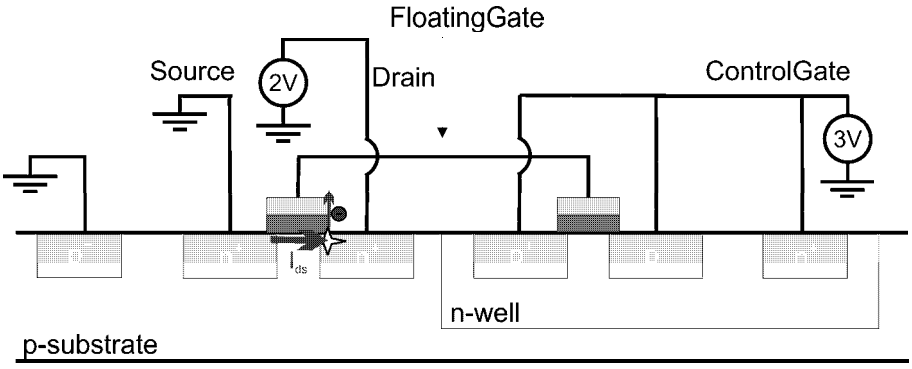

Abbildung 3.3: Beschaltung der Zelle zum Entladen des Floating Gates mit Hilfe von Hot Electron Injection

Die Voraussetzung zur Zündung von CHE ist eine Feldstärke längs des Kanals, die ca. $100\frac{KV}{cm}$ übersteigt (siehe Kapitel 2). Bei einem Transistor minimaler Größe im UMC $0.18\mu m$ Prozess (180nm Länge) würde dies einer Spannung von 1.8 Volt entsprechen. Je länger der Transistor, desto größer ist die benötigte Spannung. Diese Spannung ist umso leichter zu induzieren, je kleiner die Breite des Transistors ist, da sonst ein großer Strom durch den Kanal, der durch die positive Floating Gate Spannung gegenüber des geerdeten Substrats ausgebildet ist, fließt. Ein großer Drainstrom würde die Spannung entweder an Transmission Gates (einfache Transistoren als Schalter) vor der Speicherzelle abfallen lassen oder die Potentialdifferenz über den Kanal würde bei ausreichender Dimensionierung der Transmission Gates sehr viel Energie benötigen. Für diesen Prozess ist es daher von Vorteil, einen möglichst kurzen und schmalen nMOS Transistor einzusetzen.
Zusätzlich kann man durch Anlegen einer positiven Spannung am Control Gate V_{CG} durch die kapazitive Kopplung die Floating Gate Spannung V_{FG} erhöhen und somit die Ausbildung des Kanals im nMOS bei kleinen gespeicherten Werten sicherstellen. Dabei wird die Attraktivität des Floating Gate Potentials für Elektronen, die durch das Oxid in Richtung Floating Gate gestreut werden, erhöht. Fließt nun ein CHE Strom, wird er die positive Ausgangsspannung des Floating Gates schnell kompensieren. Dies wird zur Rückbildung des Kanals im nMOS und schließlich zum Erliegen des Hot-Electron Stroms führen. Der Vorgang schaltet sich folglich automatisch ab.

3.2 VerilogA Modelle

Grundsätzlich benötigt man zur Modellierung der Floating Gate Speicherzelle in Abbildung 2.4 in Spectre Simulationen drei VerilogA-Elemente. Eines zur Simulation des Direct- und Fowler-Nordheim-Tunnelstroms über das Gateoxdid des pMOS-Transistors und ein Weiteres zur Simulation der beiden Tunnelströme über das SiO_2 des nMOS-Transistors. Für Hot Electron Injection im nMOS Transistor wird das dritte VerilogA Element benötigt. Für den pMOS Transistor ist dieser Mechanismus bei der obigen Beschaltung nicht vorgesehen, andernfalls müsste auch dort ein weiteres Element eingefügt werden.
Der Schaltplan für das Floating Gate ist in Abbildung 3.4 zu sehen. Die Widerstände sind für die Schaltung unwichtig. Der Obere ($R_2 = 10^{45}\Omega$ dient zur Findung eines Anfangszustandes für die Simulation der Schaltung und der Untere $R_1 = 1n\Omega$ zur Drainstrommessung für das Element CHEcurrent18. Dazu wird vor und nach dem Widerstand die Spannung vd1 und vd2 gemessen.

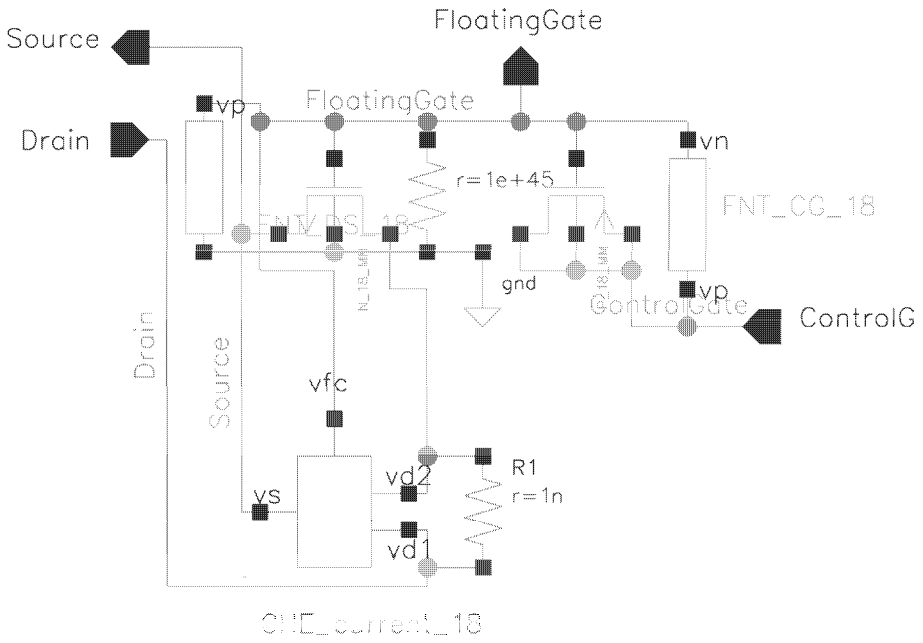

Abbildung 3.4: Ersatzschaltbild des Floating Gate Transistors mit VerilogA Tunnelstrom und Hot Electron Strom Modellen zur Simulation in Spectre

Die VerilogA-Skripte beeinhalten die in Kapitel 2 beschriebenen theoretischen Modelle für Fowler Nordheim und Direct Tunneling, sowie für Hot Electron Injection. Die ensprechenden Quellcodes sind in Kapitel 7 und im Anhang A einzusehen.

3.3 Floating Gate Transistor mit zusätzlichem Lasttransistor

In Kapitel 3.1 wurde bereits erwähnt, dass für die Tunnelströme ein Kapazitätsverhältnis $C_{nMOS} > C_{pMOS}$ benötigt wird, um mit kleinen Spannungen größere Lade- und Entladegeschwindigkeiten zu erzielen. Bei Hot Electron Injection ist hingegen ein kurzer und schmaler Auslesetransistor notwendig, welches unvereinbar mit einer großen Kapazität am nMOS ist.
Dieses Problem kann man durch Einfügen eines weiteren nMOS Transistors lösen, dessen Source- und Drainknoten mit dem Sourceknoten des Auslesetransistors kurzgeschlossen sind und der lediglich als kapazitiver Lasttransistor fungiert.
Vorteilhafter wäre ein pMOS Transistor, dessen Source-, Drain- und Bulkanschlüsse analog zum Control Gate kurzgeschlossen wären und eine Art zweites Control Gate bilden würden. Durch die große kapazititve Kopplung zum Floating Gate könnte dessen Potential mit dem zweiten Control Gate in die gewünschten Richtungen verschoben werden. Der pMOS würde im Layout jedoch eine dritte n-Well auf eigenem Potential bedeuten und aufgrund der vorgeschriebenen großen

Abstände zwischen zwei n-Wells von $2\mu m$ [2], das Design der Speicherzelle sehr stark vergößern.
Der eingefügte nMOS-Lasttransistor bewirkt beim Tunnelmechanismus im Ladezustand eine geringe Mitkopplung des Floating Gates mit dem Control Gate und eine große Kopplung beim Entladen mit V_{Source}. Er fungiert folglich als Vergrößerung der Kapazität C_{Source}. Gleichzeitig kann der Auslesetransistor in minimaler Größe dimensioniert werden, um Hot Electron Current bei niedriger Schwellspannung und geringem Energieaufwand zu ermöglichen und einen großen Ausgangswiderstand der Zelle zu gewährleisten.
Der Schaltplan dieser Zelle mit drei Transistoren ist in Abbildung 3.5 zu sehen:

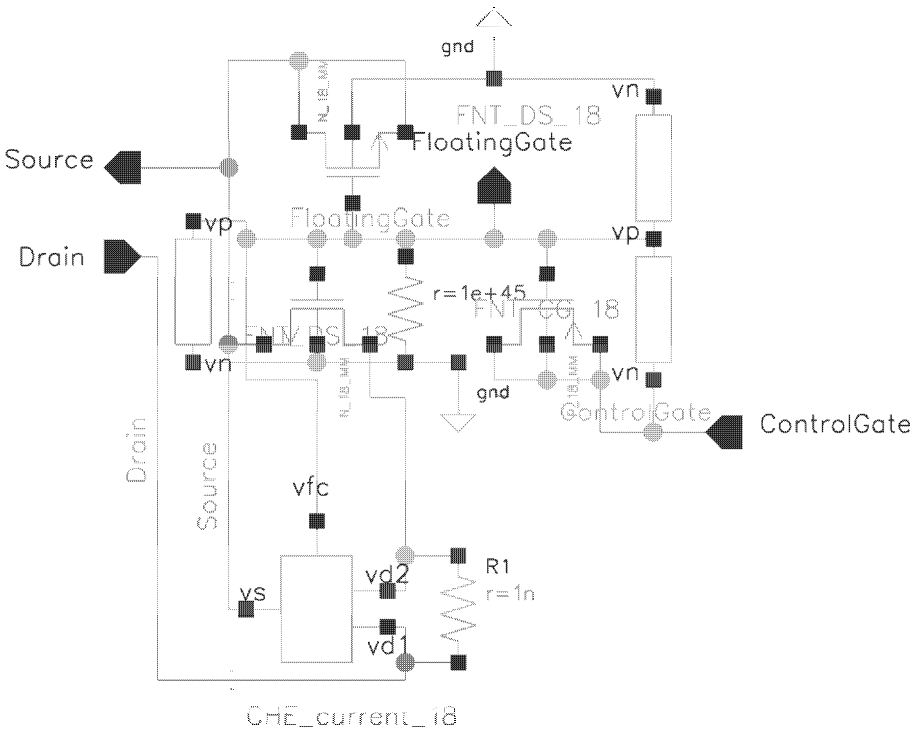

Abbildung 3.5: Ersatzschaltbild des Floating Gate Transistors mit zusätzlichem Lasttransistor

Zuätzlich muss nun auch Fowler Nordheim und Direct Tunneling durch den Lasttransistor berücksichtigt werden. Aus diesem Grund ist im Schaltplan ein weiteres VerilogA Tunnelstromelement zu finden.

[2] vorgeschrieben in den Design Rules von UMC

Kapitel 4

Aufbau der Speicherzellen und des Testchips

Zur Verifikation und Vermessung der Überlegungen aus Kapitel 2 und 3 wird ein Testchip HDFG10 entworfen. Der Schaltplan und die Funktionsweise des Chips wird in diesem Kapitel behandelt.

4.1 Schaltplan und Layout des Testchips

Der Chip zum Testen der Floating Gate Transistoren und zum Vermessen, sowie Anpassen der Lade- und Entlademechanismen besteht grundsätzlich aus einer großen Matrix (128x100) verschiedener Speicherzellen, die die oben gezeigten Floating Gate Transistoren in unterschiedlichen Größenverhältnissen zwischen den nMOS und pMOS Transistoren enthält (Abb. 4.0). In einer Spalte befinden sich jeweils 128 identische Zellen, während in den unterschiedlichen Spalten entweder die Standard Floating Gate Zelle aus Kapitel 3.1 mit einem nMOS und einem pMOS oder die Konfiguration aus Kapitel 3.3 mit einem zusätzlichen nMOS Lasttransistor in verschiedenen Kapazitätsverhältnissen zu finden sind. Eine Auflistung der 100 verschiedenen Zelltypen der Spalten befindet sich im Anhang B4.

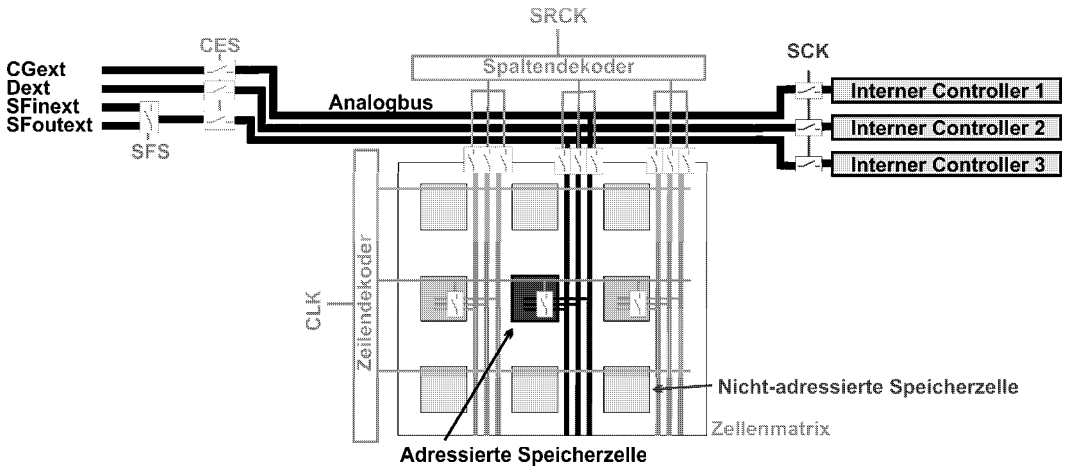

Abbildung 4.0: Schematischer Aufbau des Testchips. Die Signalbezeichnungen sind im Text erklärt.

Zur Adressierung einer Speicherzelle in der Matrix benötigt man einen Spalten- und einen Zeilendecoder und eine Einheit (IO-switch), um zwischen interner Steuerung und äußerer Steuerung schalten zu können, da der Chip zusätzlich noch einen internen Controller zum automatischen Beschreiben und Löschen einer Zelle enthält.
Der interne Controller funktioniert nach folgendem Prinzip. Auf die Steuerleitungen werden je nach gewünschtem Vorgang Lade- (5 Volt Puls auf das Control Gate, Drain und Source sind geerdet) oder Entladepulse (5 Volt Pulse auf Drain und Source, Control Gate ist geerdet) gegeben. Zwischen den Pulsen wird Drain auf 1.8 Volt gesetzt und die Zelle ausgelesen. Die Auslesespannung wird mit einer extern einstellbaren Referenzspannung verglichen und entschieden, ob die Zelle noch weitere Lade- oder Entladepulse zum Erreichen des gewünschten Wertes benötigt. Ist der Referenzwert erreicht, wird die Routine abgebrochen. Der Entwurf und das Design dieses Controllers sind jedoch nicht Teil der vorliegende Arbeit.
Des Weiteren braucht man Pads, um äußere Signale an den Chip anlegen zu können, die mit ESD-Strukturen zum Schutz vor kurzzeitiger Überspannung ausgestattet sind und die Levelshifter für die Digitallogik auf dem Chip enthalten. Die Levelshifter übersetzt zwischen 1.8 Volt Digitallogik des Chips und 5 Volt Digitallogik der Platine.
Im Folgenden ist der Chip dargestellt. Die erste Abbildung zeigt die höchste Hierachieebene in den Schaltplänen, in der die verschiedenen Komponenten Speicherzellenmatrix, Spaltendekoder, Zeilendekoder, IO-Switch und interne Controller zu sehen sind. Abbildung 4.2 zeigt das vollständige Design des Chips und die Abbildung 4.3 ein Foto des produzierten und in ein JLCC44-Package gebondeten Testchips.

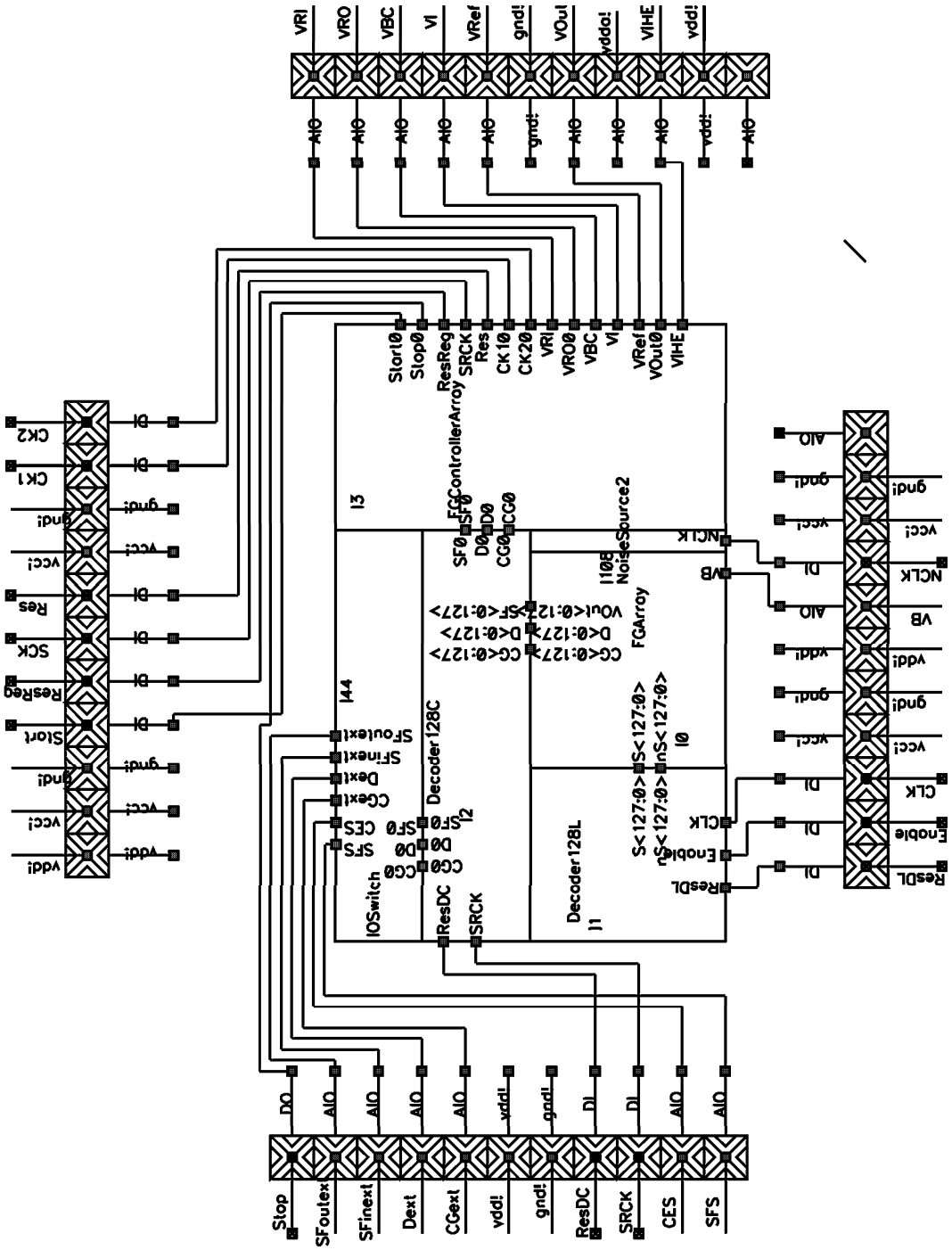

Abbildung 4.1: Schaltplan des Testchips auf oberster Hierachieebene

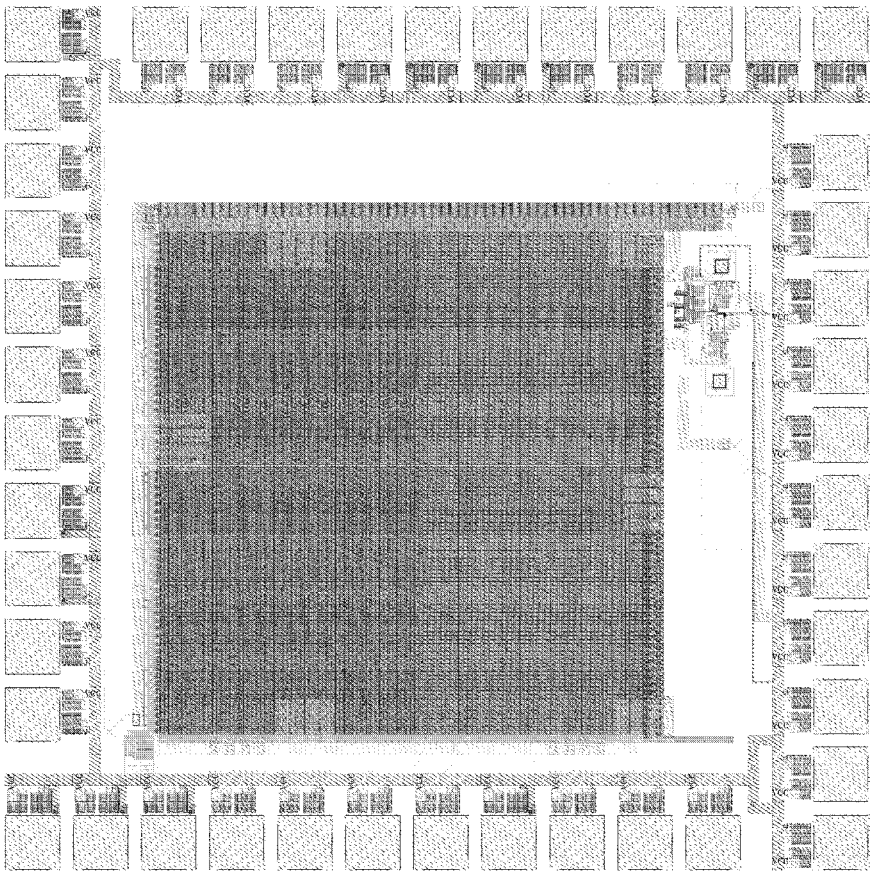

Abbildung 4.2: Layout des Testchips

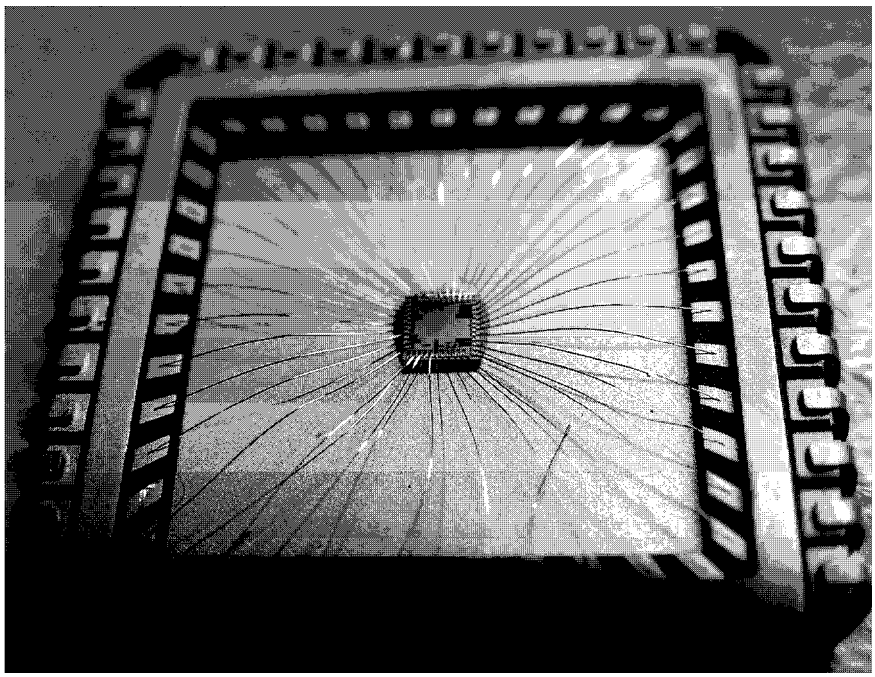

Abbildung 4.3: Foto des gebondeten Testchips

4.1.1 Speicherzelle

Jede Speicherzelle (Abbildung 4.4) enthält einen Floating Gate Transistor entweder mit oder ohne zusätzlichen Lasttransistor (siehe Kapitel 3). Daneben sind Transmission Gates angeordnet, die aus einem 3.3 Volt nMOS und einem 3.3 Volt pMOS bestehen, deren Gates von dem Zeilendekoder gesteuert werden. Diese Transistoren haben eine Durchbruchspannung von 7 Volt und sind folglich in der Lage 5 Volt Signale zu schalten. Der Zeilendecoder schaltet im adressierten Zustand die Leitung S (Select) auf eins und nS (not select) auf null. So sind die Steuerleitungen Drain, Source und Control Gate des Floating Gate Transistors angeschlossen. Im nicht-adressierten Zustand werden die Select-Leitungen des Dekoders invers geschaltet und die Steuerleitungen werden geerdet, um ein kapazitives Koppeln mit benachbarten Zellen zu verhindern.
Die Ausgangsleitung V_{OUT} wird über einen Sourcefolger (3.3 Volt Transistor, Breite: 440nm, Länge: 340nm, Biasspannung: 700mVolt) zum Auslesen der Floating Gate Spannung geschaltet und der Arbeitspunkt des Sourcefolgers durch die Biasspannung V_{VB} gesteuert.

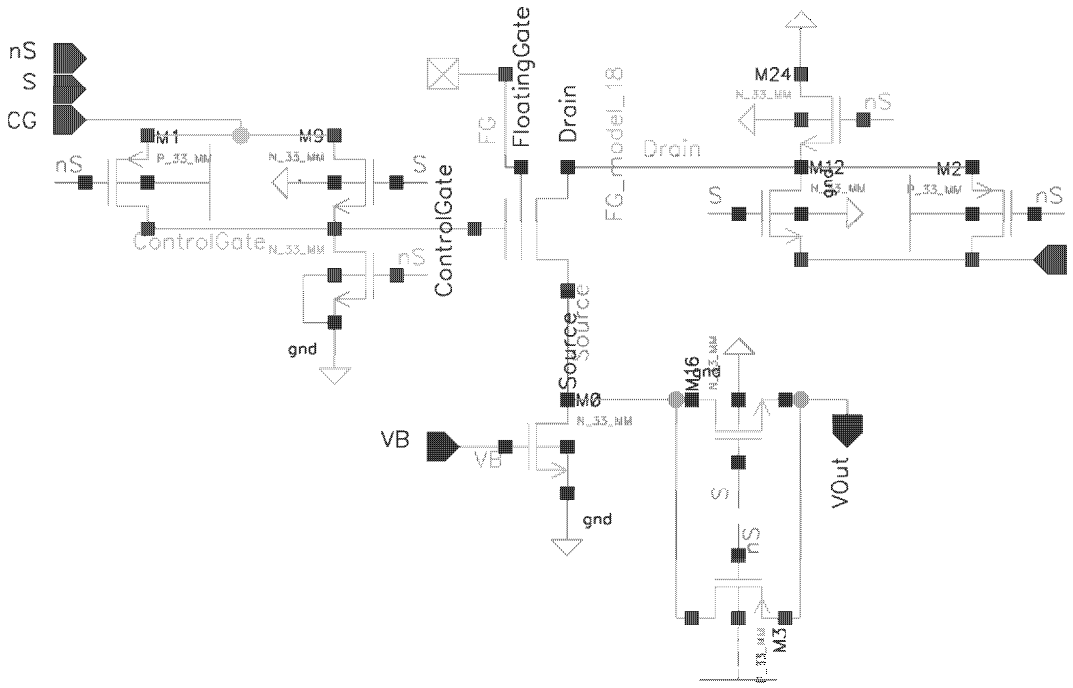

Abbildung 4.4: Schaltplan einer Speicherzelle mit Floating Gate Transistor

Das Layout dieser Zellen ist in den folgenden Abbildungen zu sehen:

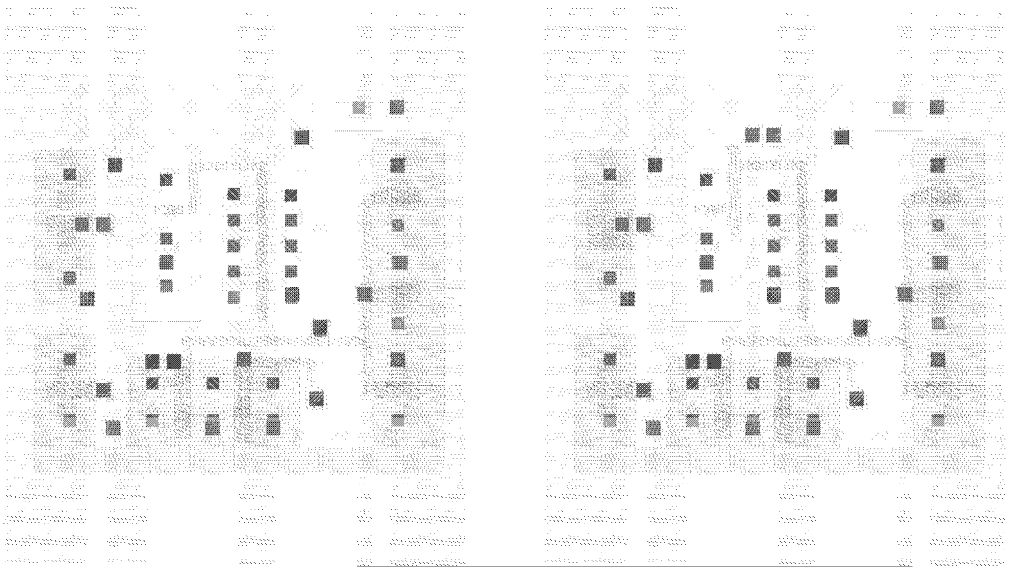

Abbildung 4.5: Layout einer Speicherzelle bestehend aus einem Floating Gate Transistor mit zwei bzw. drei Transistoren

4.1.2 Zeilendekoder/Spaltendekoder/IO-Switch

Der Zeilendekoder (Abbildung 4.6) ist ein 7-bit Dekoder, der durch die Eingabe einer 7-stelligen Binärzahl, die über eine externe Clock (CLK) eingeben wird, eine von 128 Selectleitungen einschaltet [7]. Die Selectleitungen schalten die Transmission Gates der Speicherzellen einer Zeile. Dadurch werden die drei Knoten Drain, Source und Control Gate der Speicherzelle an die vertikal durch die Speichermatrix verlaufenden Steuerleitungen angeschlossen.
Die Clock (CLK) geht zunächst auf einen Dekodercontroller, der aus dem Signal 7 Adressleitungen Adr(6:0) kodiert. Das erste Element des Dekoders dekodiert die ersten beiden Adressleitungen und schaltet durch fest verbundene AND-Gatter eine von vier Leitungen A(3:0). Die darauffolgende Stufe schaltet wiederum aus zwei Adressleitungen eine von 4 Leitungen DB(9:6) und verbindet über feste NOR-Gatter die vier vorherigen Ausgänge A(3:0) zu 8 Leitungen B(15:0), von denen wiederum nur eine adressiert ist. Die beiden nachfolgenden Stufen machen mit den weiteren 3 Adressleitungen daraus 128 Selectleitungen S(127:0), von der immer nur eine geschaltet ist. Ein Levelshifter transferiert die 1.8 Volt Signale auf 5 Volt Signale für die Transmission Gates, die zur Schaltung der großen Tunnelspannungsdifferenzen mit 5 Volt Versorgungsspannung (VCC) arbeiten. Zusätzlich gibt es eine Enable Leitung, die zur Durchschaltung der adressierten Selectleitung auf eins liegen muß, damit während den Schaltvorgängen keine Zellen kurzzeitig adressiert werden können. Ebenso ist ein Resetsignal (ResDL) implementiert, um den Dekoder zurückzusetzen.

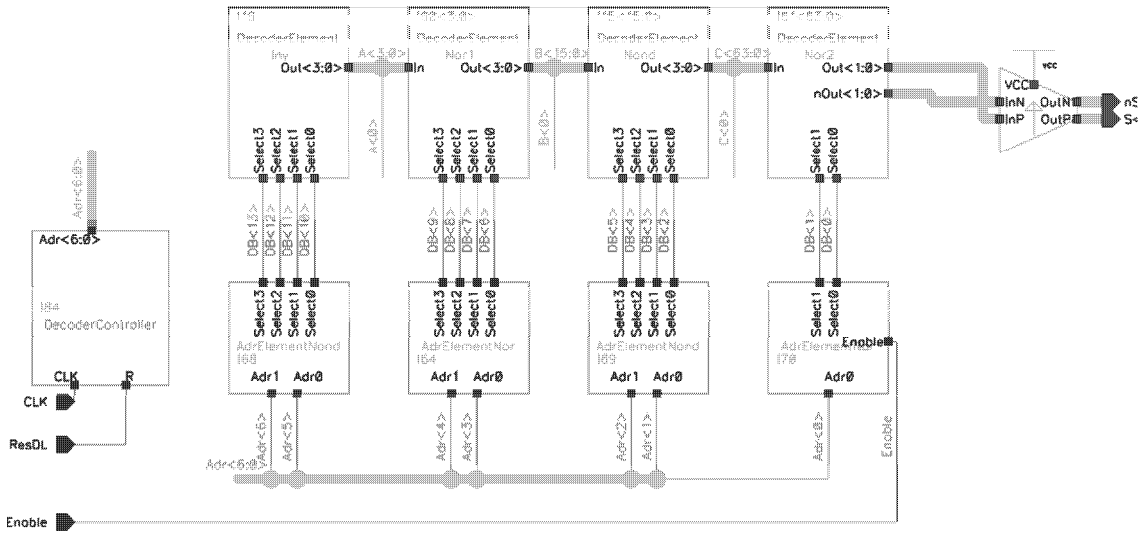

Abbildung 4.6: Schaltplan des Zeilendecoders

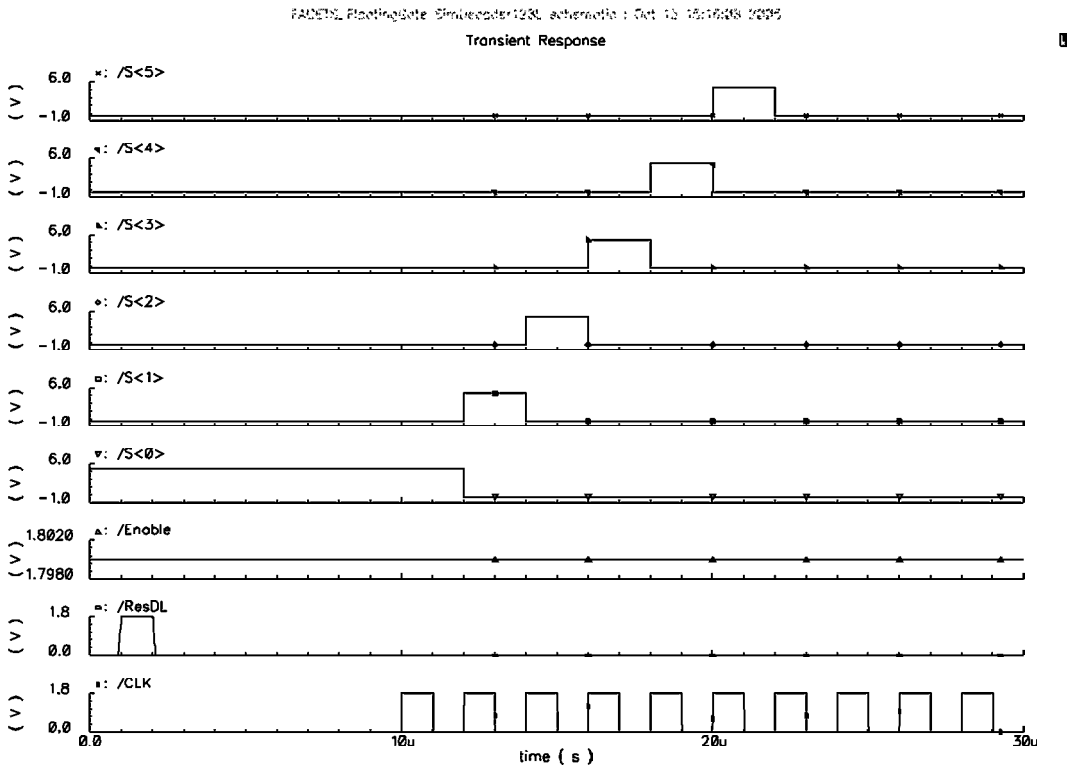

Abbildung 4.7: Simulation des Zeilendecoders. Jede Clock schaltet eine neue Adressleitung auf eins und alle anderen Leitungen auf null. Das ResetSignal ResDL setzt den Dekoder zurück.

Um die Steuerleitungen Drain D(0:127), Source SF(0:127) und Control Gate CG(0:127) einer Spalte auf den Analogbus (D0, SF0, CG0), der wahlweise zu den externen Anschlüssen oder zu dem internen Controller führt, zu schalten, befinden sich am oberen Ende der Steuerleitungen zusätzliche Transmission Gates, die von einem einfachen ringförmig geschalteten Schieberegister (Abb. 4.8), das im Kreis verschaltet ist, adressiert werden S(127:0) [7]. Das Schieberegister wird von einer externen Clock (SRCK) gesteuert und schaltet je nach Anzahl eingegangener Pulse eine von 128 Selectleitungen SR(127:0) auf 1.8 Volt. Das adressierte Transmission Gate verbindet die entsprechenden Steuerleitungen mit dem Analogbus. Alle anderen Transmission Gates erhalten von den Schieberegistern ein null Volt Signal. Um Übersprechen von benachbarten Spalten zu verhindern, werden die Steuerleitungen von nicht-adressierten Spalten auf Masse gezogen. Zusätzlich gibt es noch eine Reset Leitung (ResDC) die alle Schieberegister zurücksetzt und die erste Spalte adressiert. Hinter den Ausgängen der Register sind Levelshifter angeordnet, die aus den digitalen 1.8 Volt Signalen 5 Volt Signale machen, da die Zellen mit 5 Volt Versorgungsspannung (VCC) betrieben werden und der Digitalteil nur mit 1.8 Volt Versorgungsspannung (VDD) arbeitet. Eine Simulation des Spaltendekoders ist in Abbildung 4.9 zu sehen.

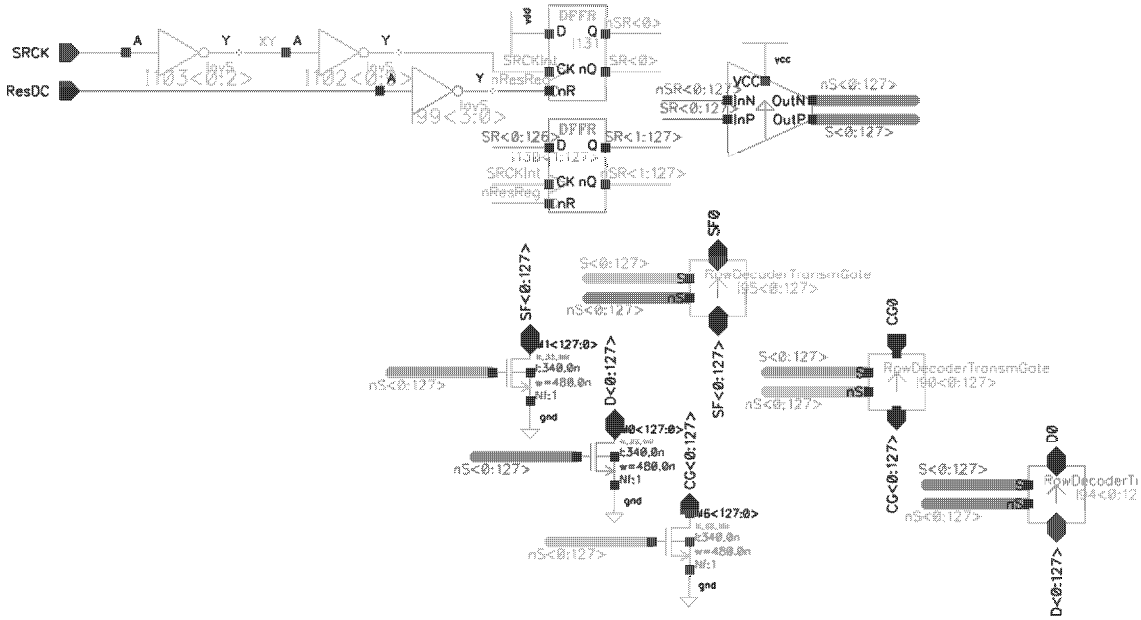

Abbildung 4.8: Schaltplan des Spaltendecoders

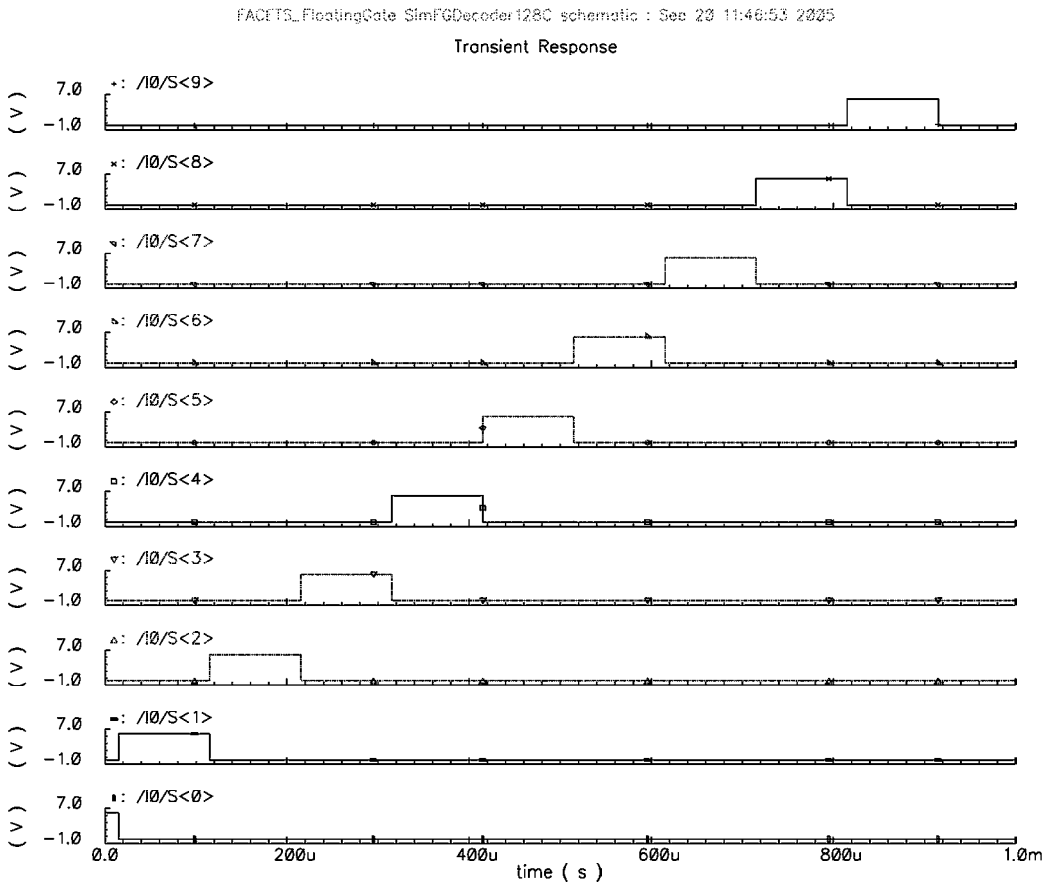

Abbildung 4.9: Simulation des Spaltendecoders. Jede Clock schaltet eine neue Adressleitung auf eins und alle anderen Leitungen auf null.

Um den analogen Bus (SF0, CG0, D0) wahlweise an die externen Pads zu schalten oder an den internen Controller geschaltet zu lassen, sind weitere Transmission Gates eingefügt, die über das digitale 5 Volt Signal CES geschaltet werden. Die Leitung am Sourcefolger SF0 wird während des Entladevorgangs sowohl als Eingang, wenn die 5 Volt Entladepulse auf den Sourceknoten des Floating Gate Transistors gegeben werden, oder auch als Ausgang, während des Auslesezyklus, verwendet. Hierfür existieren zwei Leitungen SFinext und SFoutext, die von einem Digitalsignal SFS über einen Multiplexer geschaltet werden. Das Schaltbild ist in Abbildung 4.10 zu sehen.

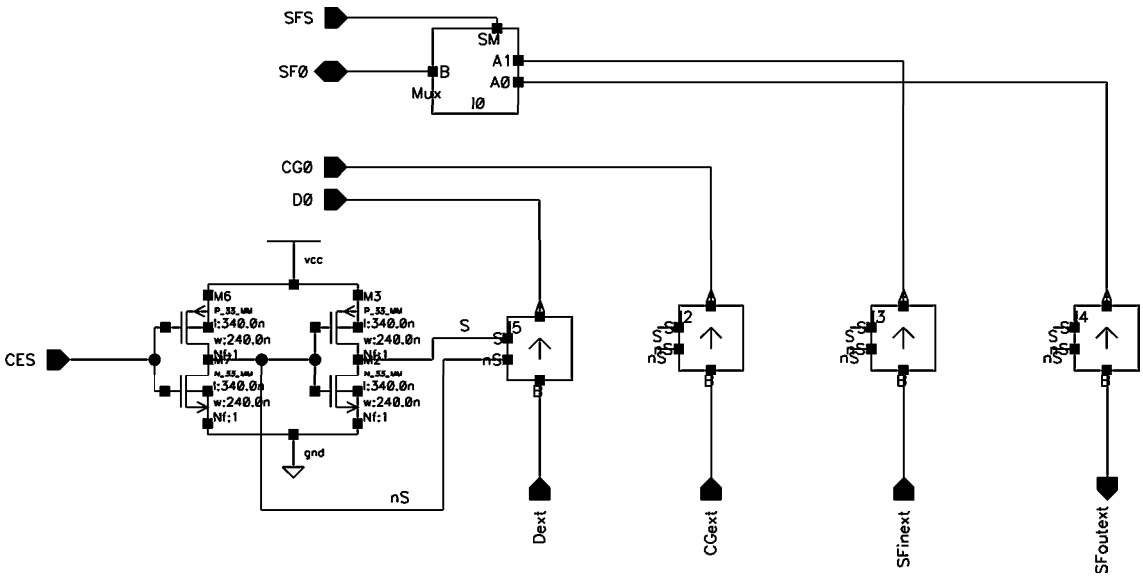

Abbildung 4.10: Schaltplan zur Steuerung des Analogbusses

Kapitel 5

Beschreibung des Testaufbaus

Der Testchip wird in einem Sockel auf einer Platine betrieben. Eine Labview 12bit Ni-DAQ Karte in einem PC ist über einen SCSI-Stecker mit der Platine verbunden und kann analoge und digitale Signale zwischen Chip und PC übertragen. Alle Signale am Ausgang der Karte werden mit Hilfe eines LabView Programms erzeugt und die Eingänge aufgezeichnet. Die Clocks für den internen Controller des Chips generiert ein programmierbarer Mikrocontroller auf der Platine und ein externes Netzgerät ist für die Stromversorgung verantwortlich.

5.1 Platinenaufbau

Die Testplatine besteht aus zwei Spannungsreglern, die die Versorgungspannung $V_{dd} = 1.8$ Volt für die Digitallogik des Chips und die über ein Potentiometer veränderbare Versorgungsspannung $V_{cc} = 5$ Volt für den Analogteil generieren. Ein weiterer Spannungsregler erzeugt zusammen mit vier weiteren Potentiometern die einstellbaren Biasspannungen $VB = 0.7$ Volt, $VIHE$, VI und VBC, wobei die letzten drei Spannungen von dem internen Controller benötigt werden. Die analogen Ausgangssignale vom Chip gehen über zwei zweikanalige Operationsverstärker, die als Impedanzwandler beschaltet sind, über den SCSI-Stecker auf die Analogeingänge der Ni-DAQ Karte (die Steckerbelegung ist im Anhang B2 zu finden). Die Digitalsignale werden über $330k\Omega$ Widerstände gebuffert und direkt auf die Schnittstelle geleitet. Der vollständige Schaltplan der Testplatine ist in Abbildung 5.1 und ein Foto von Vorder- und Rückseite der bestückten Platine in Abbildung 5.2 dargestellt.

Abbildung 5.1: Schaltplan der Platine

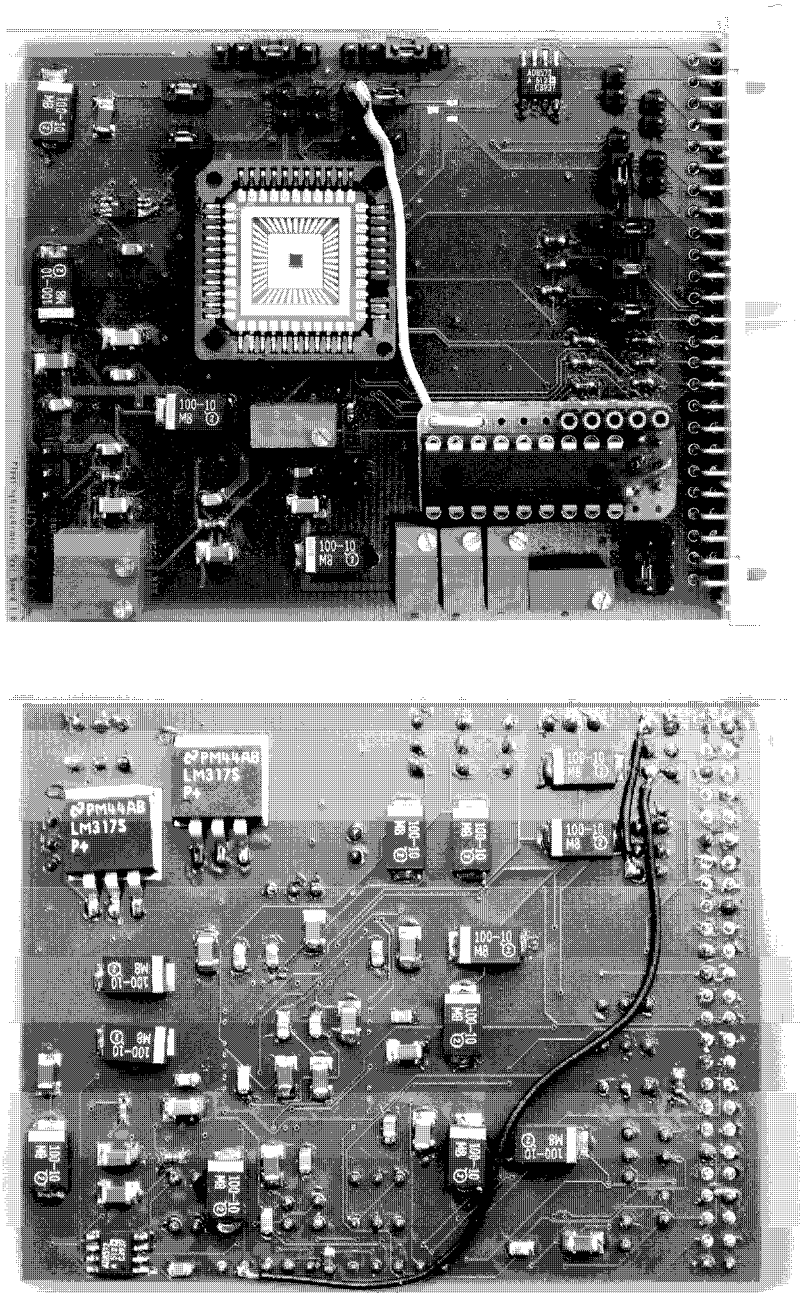

Abbildung 5.2: Fotos der Platine mit Testchip

5.2 LabView-Programme zur Datenaufnahme

Die LabView-Programme haben die Aufgabe, durch das entsprechende Beschalten der Steuerleitungen die Speicherzellen zu adressieren, den internen Controller

zu aktivieren, eine Referenzspannung vorzugeben und die Daten des Analogbusses in eine Datei zu schreiben. Aufgezeichnet wird das Drain, Control Gate und V_{OUT} Potential. Alternativ können externe Spannungen an die Pads des Analogbusses angelegt und so die Zellen beschrieben oder gelöscht werden.
Grundsätzlich gibt es drei Programme: Jeweils eines zum Aufzeichnen des Lade- und des Löschvorgangs, sowie ein weiteres Programm zum dauerhaften Auslesen der adressierten Zelle, um Leckströme durch Messen des Spannungsabfalls von V_{OUT} in Abhängigkeit der Zeit zu bestimmen.
Das Programm zum Laden der Zelle schaltet zunächst die Pads des Chips und das Enable Signal des Zeilendekoders aus, um anschließend Zeilendekoder, Spaltendekoder und den Dekoder zum Auswählen des richtigen internen Controllers[1] zurückzusetzen. Nun werden auf den Zeilendekoder und den Spaltendekoder jeweils soviele Pulse gegeben, bis die gewünschte Zelle adressiert und an den analogen Bus angeschlossen ist.
Anschließend wird die Zelle zunächst über den internen Controller auf den gewünschten Startwert (bspw. 0.4 Volt) geladen oder entladen. Dazu muss der Controllerdekoder zunächst über ein Reset Signal zurückgesetzt werden und bekommt anschließend die richtige Anzahl Pulse über SES für den jeweiligen Controller zugesendet. Danach wird Enable und SFS zum Einschalten der Pads auf Eins gesetzt und ein Startpuls an den Controller geschickt, der daraufhin seine Arbeitsroutine durchführt. Das LabView Programm wartet, bis bei Erreichen der gewünschten Startspannung ein Stopsignal vom Chip zurückgesendet wird.
Nun wird die Referenzspannung angelegt und der Controller erneut zum Laden an den Analogbus geschaltet. Das Programm sendet jetzt wieder ein Startsignal und beginnt die Spannungswerte auf dem Analogbus in eine ASCII-Datei zu schreiben. Da der Chip schneller beginnt, die Zelle aufzuladen als das LabView Programm die Aufzeichnungsroutine startet, wird das Startsignal durch ein Skript in dem Mikrocontroller auf dem Tochterboard der Platine verzögert. Zusätzlich generiert der Mikrocontroller die Clocks für den internen Controller. Die Samplerate und Anzahl der Samples beim Aufzeichnen ist frei wählbar. Aufgezeichnet wird die Zeit, die Drain, V_{OUT} und Control Gate Spannung. Nachdem die Aufzeichnung beendet ist, wird ein Puls zum Zeilendekoder gesendet, so dass die nächste Zeile dieser Spalte adressiert und das Programm erneut gestartet wird. Auf diese Weise werden nacheinander alle Zellen einer Spalte automatisch vermessen.
Während den Digitalschaltvorgängen wird Enable auf Null gesetzt, damit kurze Spannungsspitzen auf dem Analogbus nicht die Zelle zerstören. Zum Auslesen der Zelle wird die Drainspannung V_{Drain} auf 1.8 Volt geschaltet.

Das Entladeprogramm funktiert nach dem gleichen Prinzip, nur invers. D.h. am Anfang wird die adressierte Zelle auf einen höheren Wert geladen (bspw. 1.5 Volt)

[1] Es ist jeweils ein interner Controller zum Laden und Entladen durch Tunnelströme, sowie einer zum Entladen mit Hot Electrons implementiert

und der Entladevorgang wird entsprechend aufgezeichnet.

Die Routine zum Messen der Leckströme im adressierten Zustand lädt die gewünschte Zelle nach obigen Prinzip auf und schreibt anschließend einfach die Auslesespannung V_{OUT} in Abhängikeit der Zeit in eine ASCII-Datei, mit einer sehr niedrigen Samplerate. Danach wird die nächste Zelle adressiert und die Routine startet erneut.

Kapitel 6

Auswertung und Darstellung der Messdaten

Dieses Kapitel zeigt und erläutert die aufgezeichneten Messdaten. Zur Anpassung der Tunnelstrommodelle müssen die Daten mit Hilfe von C-Programmen weiterverarbeitet werden.

6.1 Vermessung der Standardspeicherzelle

6.1.1 Ladevorgang der Standardzelle

Im Folgenden wird der Aufladevorgang einer in Kapitel 3.1.1 beschriebenen Standard Floating Gate Speicherzelle mit einem nMOS (Länge: 210nm, Breite: 2480nm) und einem pMOS (Länge: 180nm, Breite: 240nm) gezeigt, die von 0.4 auf 1.8 Volt Auslesespannung geladen wird.
Die Rohdaten, die von dem LabView-Programm (siehe Kapitel 5.2) zur Datenaufzeichnung in eine ASCII-Datei geschrieben werden, sind in Abbildung 6.1 zu sehen. Durch die blauen Messpunkte wird die Ausgangsspannung des Sourcefolgers dargestellt, die mit der Zeit zunimmt und nach ungefähr 3.6 Sekunden den Maximalwert erreicht. Aufgeladen wird das Floating Gate durch 5 Volt Pulse auf dem Control Gate (gelbe Messpunkte). Zwischen den Ladepulsen wird die Spannung am Drainknoten (rote Punkte) auf 1.8 Volt geschaltet, um die Sourcefolgerspannung auslesen zu können und den Zielwert mit dem aktuellen Wert zu vergleichen. Während des Lesevorgangs wird das Control Gate geerdet.
Zur Visualisierung der einzelnen Schaltvorgänge ist der Bereich von 1 bis 1.05 Sekunden zusätzlich stark vergrößert abgebildet. Die Geschwindigkeit des gesamten Ladevorgangs ist wie in Kapitel 3.1.1 beschrieben von dem Kapazitätsverhältnis zwischen nMOS und pMOS abhängig.

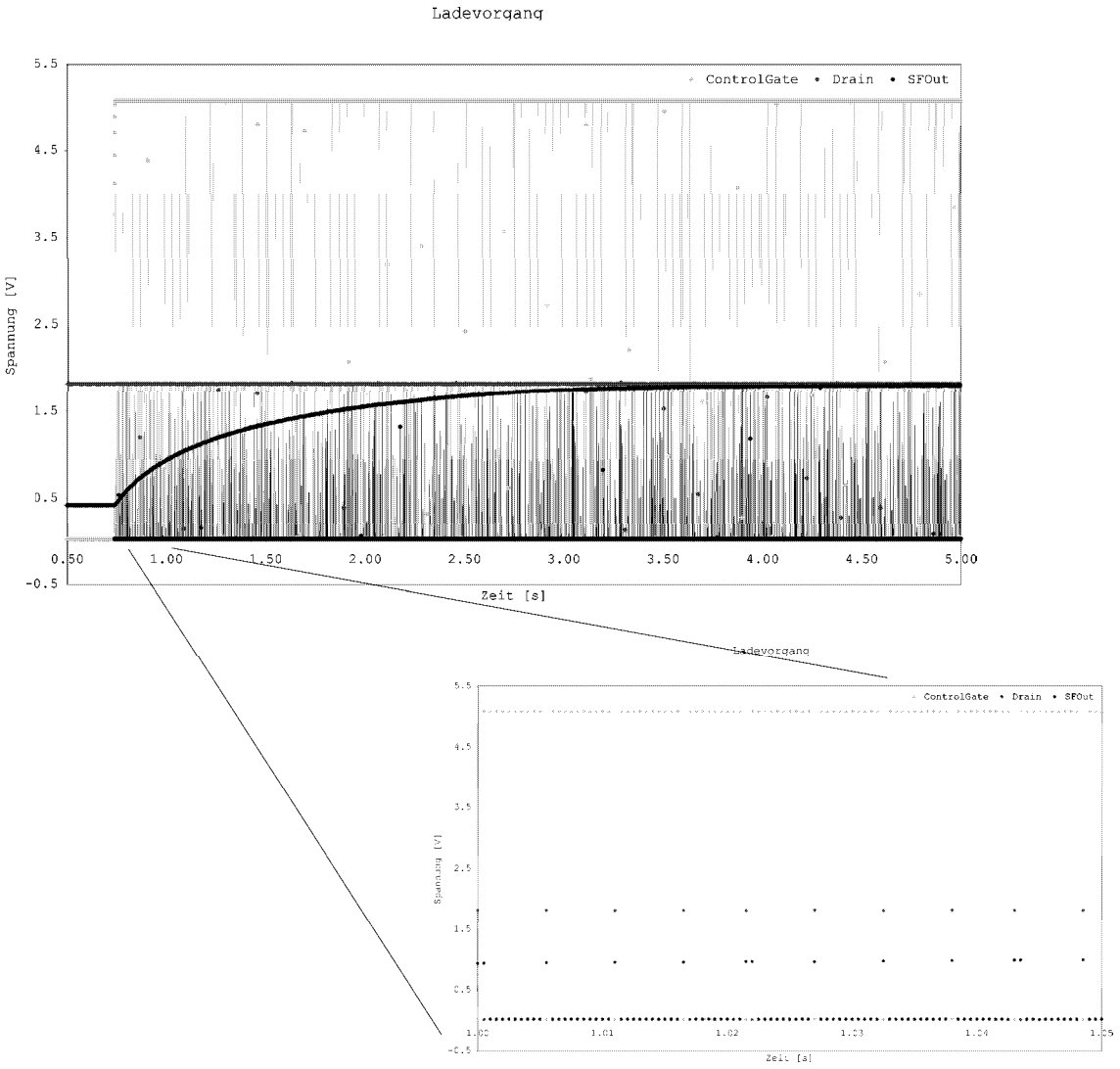

Abbildung 6.1: Rohdaten einer Standardspeicherzelle beim Laden von 0.4 auf 1.8 Volt. Zu sehen sind die abwechselnden Lade- ($V_{CG} = 5$ Volt) und Lesepulse ($V_{Drain} = 1.8$ Volt), sowie die ansteigende Spannung V_{OUT} am Sourcefolger.

Zur Berechnung der Tunnelströme durch das SiO_2 mit Hilfe der ansteigenden zeitabhängigen Spannungswerte am Sourcefolger müssen die Daten zunächst umgerechnet werden. Hierfür wurde jeweils für den Lade- und Entladevorgang ein Datenauswertungsprogramm in C geschrieben. Ausführlich wird die Datenaufbereitung anhand des Ladevorgangs beschrieben, die durch das Programm charge.c erfolgt. Alle Programme sind im Anhang B5 zu finden.

Die eigentliche Auswertung beginnt in Zeile 290 mit dem Einlesen der Daten aus der von LabView angelegten ASCII-Datei. Da die Spannungsänderung des

Floating Gates in Abhängigkeit der Zeit benötigt wird, muss zuerst die Sourcefolger Ausgangsspannung V_{OUT} in die Floating Gate Spannung V_{FG1} umgerechnet werden. Dies ist auf analytischem Weg nicht möglich, da die Koppelkapazitätskoeffizienten (siehe Kapitel 2.1.3) unbekannt sind. Folglich wird eine Simulation der gesamten Speicherzelle durchgeführt (Abb. 6.2) und mit Hilfe der in Spectre vorhandenen Transistormodelle die Floating Gate Spannung in Abhängigkeit der Source, Drain und Control Gate Spannung bestimmt. Die Floating Gate Spannung wird durch eine gepulste Konstantstromquelle im Schaltplan sukzessive von 0 auf 5 Volt erhöht.
Die Daten für die im nachfolgenden durchgeführten Approximationen sind aus dieser Simulation gewonnen. Für die korrekte Wiedergabe der Kapazitäten wurde zuvor eine Extraktion von parasitären Koppelkapazitäten aus dem Layout der Zelle mit Hilfe des Programms Calibre [1] durchgeführt. Die Kapazitäten sind als PCAPS in das Schaltbild eingefügt.

[1]Programm zur Verifikation von Design Rules von der Firma Mentor

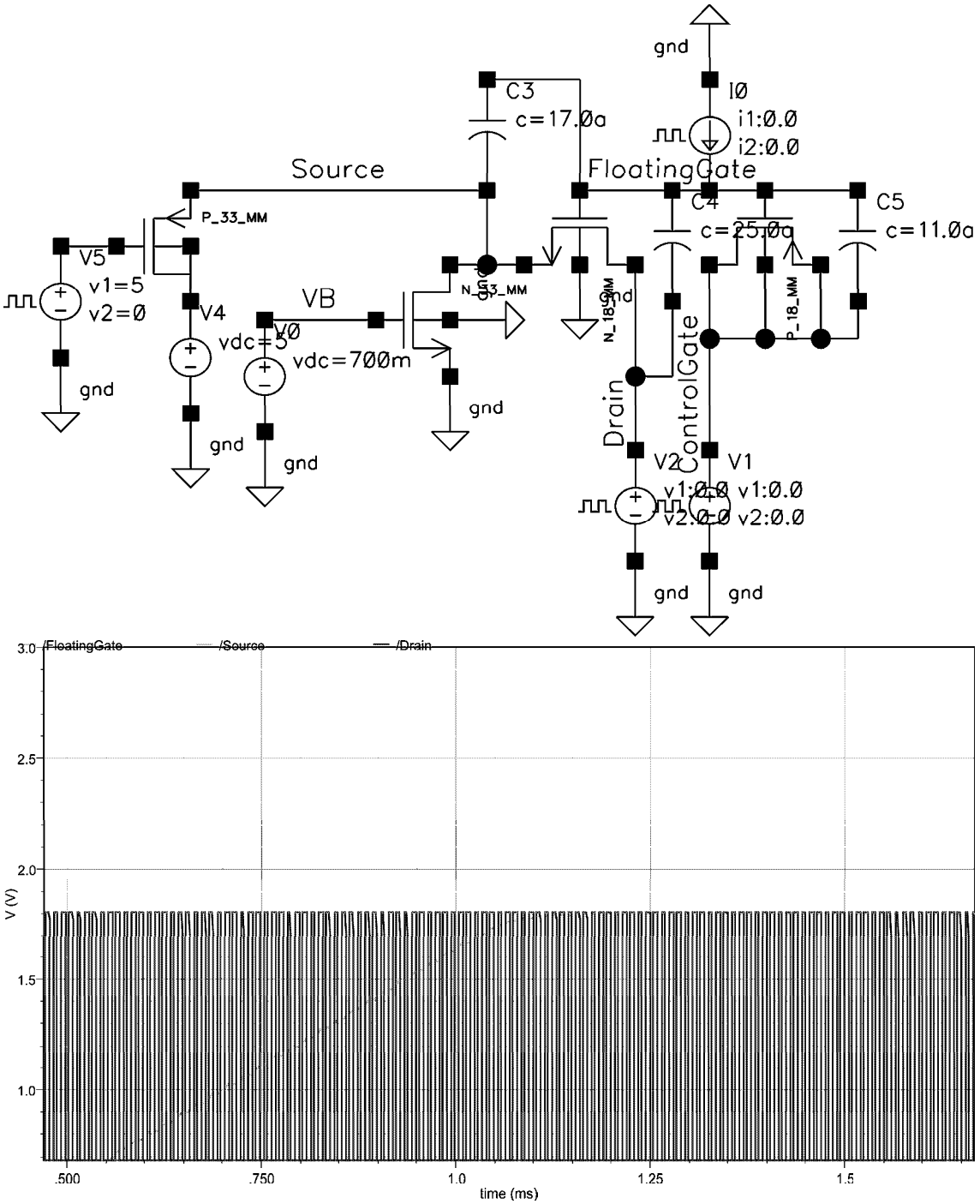

Abbildung 6.2: Schaltbild der simulierten Zelle mit eingefügten Koppelkapazitäten und erzielten Simulationsergebnissen. In blau gezeichnet ist die Floating Gate Spannung, in grün die Sourcespannung und die Drainspannung in violett.

Zur Berechnung der Floating Gate Spannung mittels der gemessenen Sourcefol-

gerspannung, wird der lineare Zusammenhang zwischen den beiden Größen mit Hilfe der Simulation approximiert. Der Fit ist in Abbildung 6.3 zu sehen und ist im Auswertungsprogramm in Zeile 295 eingefügt:

$$V_{FG1} = 1.1474 \cdot V_{OUT} + 0.4043$$

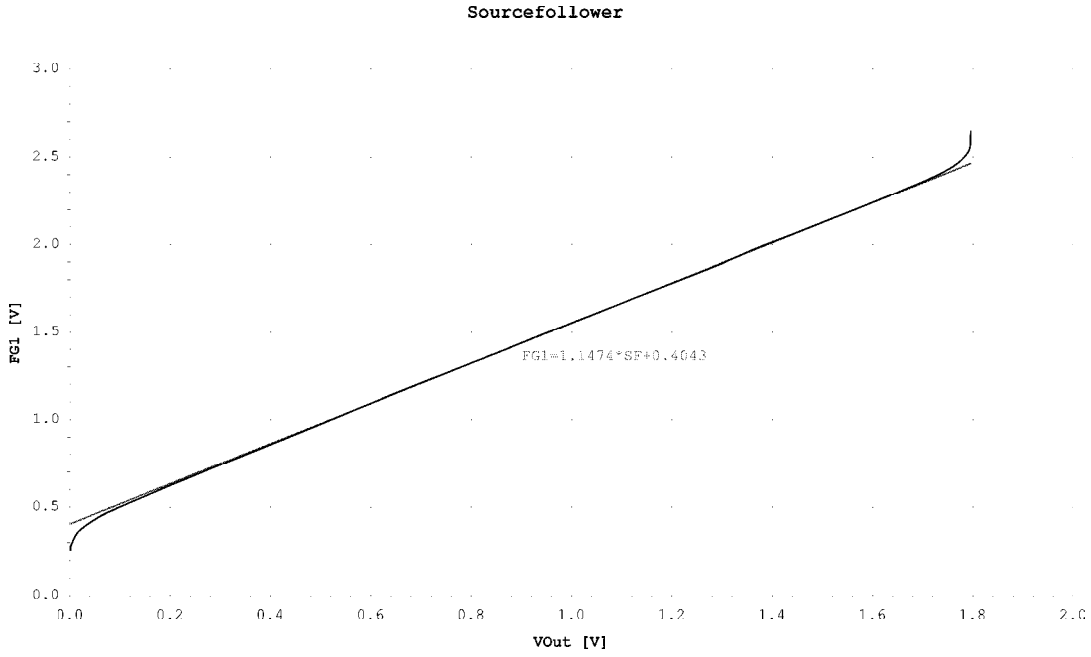

Abbildung 6.3: Aufgetragen und linear approximiert ist die simulierte Floating Gate Spannung gegen die Sourcefolgerspannung.

Das Problem ist nun, dass V_{FG1} im Auslesezustand errechnet wurde, d.h. bei $V_{Drain} = 1.8$ Volt. Zur Berechnung der Spannung über das Tunneloxid muss jedoch die Floating Gate Spannung V_{FG2} im Ladezustand, d.h. bei $V_{ControlGate} = 5$ Volt und $V_{Drain} = 0$ Volt, bekannt sein, da das Floating Gate sowohl mit Drain, als auch mit der Control Gate Spannung mitkoppelt. Die Kapazitäten der Transistoren sind spannungsabhängig. Daher wurde zum Rückschluss auf die Spannungsmitkopplung zwischen $V_{Drain} = 1.8$ Volt, $V_{CG} = 0$ Volt und $V_{Drain} = 0$ Volt, $V_{CG} = 5$ Volt in Abhängigkeit der Floating Gate Spannung ebenfalls ein Fit der oben durchgeführten Spectre Simulation verwendet (Abb. 6.2).

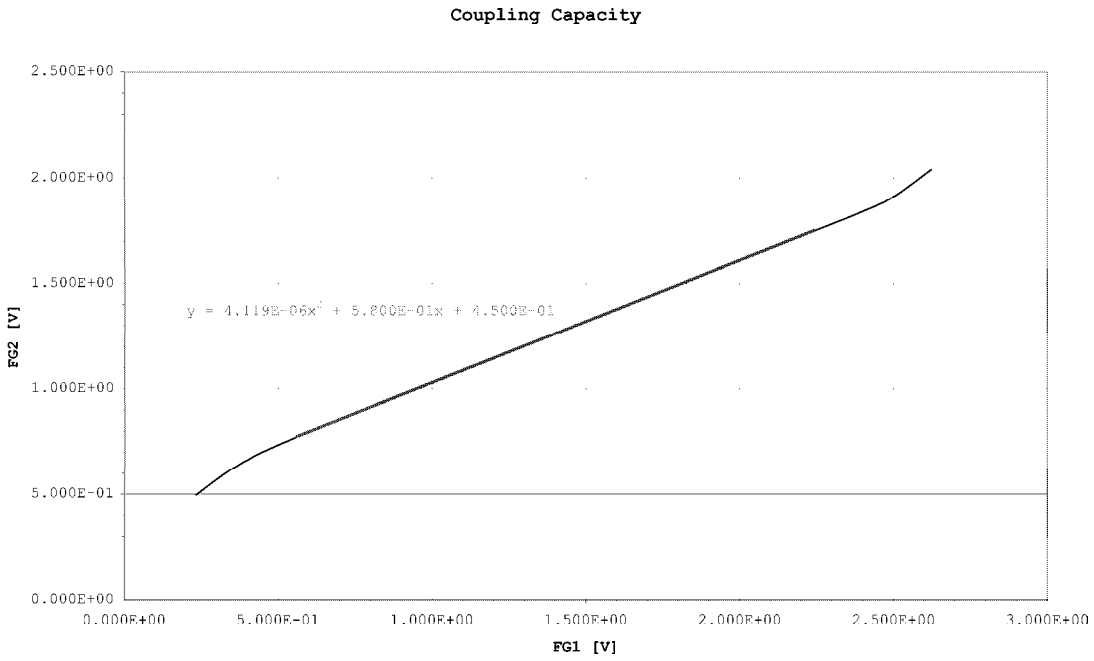

Abbildung 6.4: Dargestellt ist die Floating Gate Spannung während der Ladepulse in Abhängigkeit der Floating Gate Spannung während der Auslesepulse und die lineare Approximation im Arbeitsbereich.

Dieser in Abbildung 6.4 durchgeführte Fit wird in Zeile 296 verwendet:

$$V_{FG2} = 4.119E-06 \cdot V_{FG1}^2 + 5.8E-01 \cdot V_{FG1} + 4.5E-1$$

In Zeile 298 wird von dem Ladepuls auf dem Control Gate (in dieser Konfiguration immer gleich 5 Volt) die Floating Gate Spannung subtrahiert und so die Potentialdifferenz durch das Tunneloxid am pMOS Transistor berechnet.
Zur Angabe eines Ladestroms muss die steigende Floating Gate Spannung mit Hilfe der Kapazität in Ladung umgerechnet werden: $Q_{FG} = C_{FG} \cdot V_{FG}$. Da die Kapazität C_{FG} als Summe der Kapazitäten der beiden Transistoren und parasitärer Kapazitäten auch von der Floating Gate Spannung abhängt, wird zur Abschätzung dieser Größe wieder die Approximation einer Spectre basierten Simulation verwendet (Abb. 6.2). Simuliert wird eine Floating Gate Zelle, die mit einem konstanten Strom $I_{const} = 10^{-11}A$ aufgeladen wird. Aus der Ladezeit in Abhängikeit der Ladespannung lässt sich durch die folgende Beziehung die Gesamtkapazität abschätzen:

$$C_{FG} = \frac{Q_{FG}}{U_{FG}} \Rightarrow C = \frac{I_{const} \cdot t(V_{FG})}{U_{FG}}.$$

Die Approximation der Größe $t(V_{FG})$ ist in Abbildung 6.5 zu sehen und in Zeile 297 eingefügt:

$$C = \frac{1E-11}{V_{FG2}} \cdot (2.11513E-04 \cdot V_{FG2} + 3.31773E-05)$$

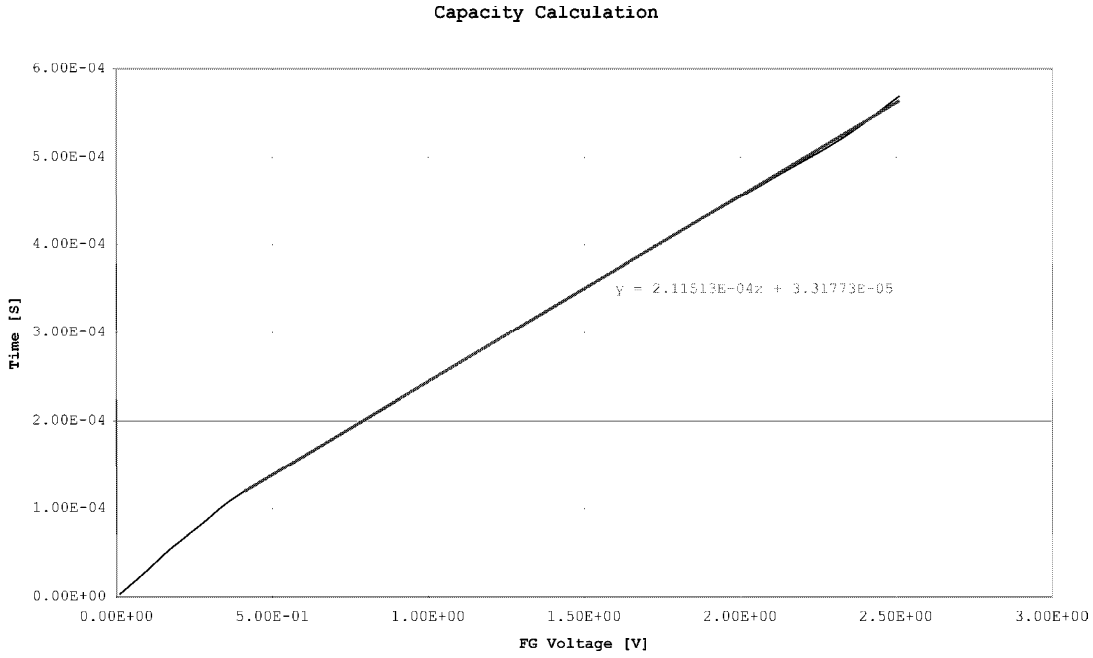

Abbildung 6.5: Ladezeit des Floating Gates in Abhängigkeit der Floating Gate Spannung bei einem konstanten Ladestrom von 10pA zur Berechnung der Floating Gate Gesamtkapazität.

Da mit einer konstanten Samplerate aufgezeichnet wird, ist der Zeitschritt zwischen zwei aufeinanderfolgenden Spannungswerten $\frac{1}{samplerate}$ (Zeile 248, 310). Aufgrund der kurzen Lesezyklen zwischen den Ladevorgängen muss die absolute Zeitskala durch das Verhältnis zwischen Lese- und Ladezeit korrigiert werden. Dieses Zeitverhältnis wird durch die Clocks des Mikrocontrollers bestimmt und beträgt 0.899 (Zeile 365, 383).
Nun könnte durch die Beziehung $I_{tunnel} = \frac{\Delta Q}{\Delta t \cdot 0.899}$ der Tunnelstrom auf/vom Floating Gate allein durch die Veränderung der Spannung in Abhängigkeit der Zeit mit der Berechnung der Zeit- und Spannungsdifferenzen in Zeile 364 und 382 bestimmt werden.

Hierbei tritt jedoch folgendes Problem auf:

Die Auflösung der Ni-DAQ-Karte beträgt 12bit für ein Intervall von -12 bis +12 Volt und ist folglich zu grob, um die minimalen Spannungsänderungen zwischen zwei Lesezyklen richtig aufzulösen. Die berechneten Spannungsdifferenzen zwischen zwei Messwerten sind quantisiert und liegen häufig im Bereich der minimalen Auflösung $5 \cdot 10^{-3}$ oder einem Vielfachen davon. Folglich ergeben sich auch quantisierte Ströme. Eine Lösung dieses Problems besteht darin, die Größe $Q(t)$ zunächst durch ein Polynom $Q_{FG} = \sum_{i=0}^{n} a_i t^i$ zu approximieren. Dazu wird in das Programm eine Funktion zum Ausführen von Polynomfits beliebiger Ordnung eingefügt (Zeile 17 bis 204) und in Zeile 434 ausgeführt.

Die erste Ableitung des Polynoms ergibt den gesuchten Strom $I(t) = \frac{dQ(t)}{dt}$ (Zeile 455).
Da letztendlich der Strom in Abhängigkeit der Oxidspannung V_{OX} und nicht in Abhändigkeit der Zeit gesucht ist und eine feste Beziehung zwischen Zeit und Oxidspannung besteht, kann der berechnete zeitabhängige Tunnelstrom zusammen mit der zugehörigen zeitabhängigen Oxidspannung ausgegeben werden (Zeile 463). Das Programm springt anschließend in den Ausgangszustand zurück und bearbeitet den nächsten Datensatz.

Da die Zellen untereinander in der Ladegeschwindigkeit sehr stark abweichen, ist eine Mittelung über viele Zellen notwendig. Dazu wurde ein eigenes C-Programm fitting.c (Anhang B5) entworfen, welches den Mittelwert der Tunnelströme einer bestimmten Oxidspannung und dessen Standardabweichung über alle Zellen berechnet.
In der folgenden Abbildung 6.6 sind die Tunnelströme beim Laden von 420 verschiedenen Zellen von vier Testchips und der Fit der Messdaten abgebildet. Im Diagramm sind einzelne Ausreißer von Messpunkten zu erkennen, die auf fehlerhafte Zellen (bspw. bereits durchgebrochene Oxidbarrieren) oder Fehler bei der Durchführung des Polynomfits zuruckzuführen sind, aber bei der großen Menge an Datenpunkten nicht ins Gewicht fallen.

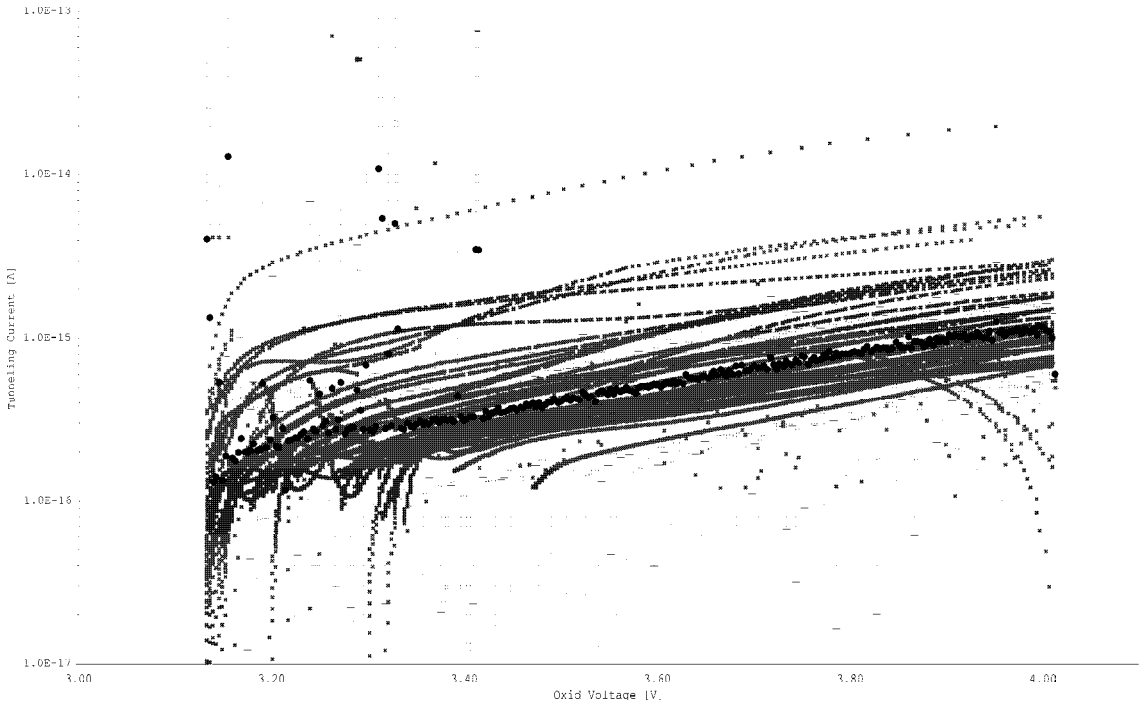

Abbildung 6.6: Ladeströme von 420 Standardspeicherzellen vier verschiedener Testchips in Abhängigkeit der Oxidspannung.

6.1.2 Entladevorgang der Standardzelle

Der Entladevorgang einer in Kapitel 3.1 beschriebenen Standard Floating Gate Zelle mit einem nMOS (Länge:210nm, Breite: 2480nm) und einem pMOS (Länge:180nm, Breite:240nm) von 1.5 auf 0.3 Volt Auslesespannung wird analog zum Ladevorgang ausgewertet. In Abbildung 6.7 sind die aufgezeichneten Rohdaten einer Zelle zu sehen.

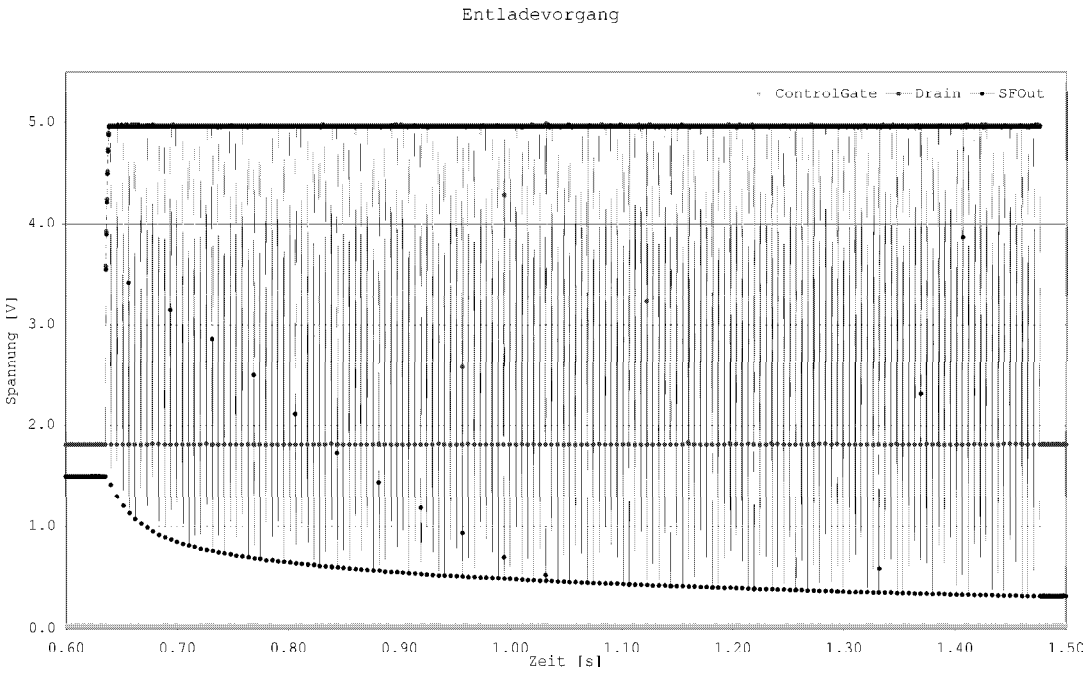

Abbildung 6.7: Entladen einer Standard Speicherzelle von 1.5 auf 0.3 Volt Auslesespannung (blau). Die Control Gate Spannung wird durch gelbe, die Drainspannung durch rote Linien dargestellt.

Die blauen Messpunkte zeigen wieder die Ausgangsspannung des Sourcefolgers, die nun mit der Zeit abnimmt und nach ungefähr 0.8 Sekunden den Minimalwert erreicht. Somit ist der Entladevorgang deutlich schneller als der Ladevorgang, der für diese Spannungsdifferenz ungefähr 1.9 Sekunden dauert.
Beim Entladen werden auf den Drainknoten (rot), der im Auslesezustand auf 1.8 Volt liegt, 5 Volt Pulse geschaltet. Gleichzeitig wird der Sourceknoten (blau), der in den Auslesezyklen als Ausgang fungiert, während der Entladezyklen ebenfalls auf 5 Volt geschaltet. Die Control Gate Spannung (gelb) liegt dauerhaft auf Masse. Erreicht die Sourcespannung im Auslesezyklus den Zielwert von 0.3 Volt, wird die Messung abgebrochen.

Zur Umrechnung der zeitabhängigen Spannungsänderung in spannungsabhängige Tunnelströme wird wieder ein C-Programm discharge.c verwendet (siehe Anhang B5), das analog zum Ladeprogramm charge.c funktioniert. Die approximierten Größen können ebenfalls verwendet werden, da es sich um die gleichen Zellen mit gleichen Kapazitätsverhältnissen handelt.

Allerdings müssen folgende Modifikationen vorgenommen werden:
Erstens ist die Oxidspannung am Tunneltransistor hier gleich der Floating Gate

Spannung, da das Control Gate auf null Volt liegt (Zeile 281). Die errechneten Ladungsdifferenzen müssen zusätzlich mit dem Faktor -1 multipliziert werden, da die Ladung beim Entladevorgang mit der Zeit abnimmt (Zeile 322, 340). Des Weiteren ist die Koppelkapazitätsberechnung $FG2(FG1)$ nun nicht mehr richtig, da die Source- und Drainknoten zwischen Auslesezustand und 5 Volt Ladeszustand geschaltet werden, während das ControlGate konstant auf Masse bleibt. Deshalb ist wieder eine Spectre Simulation mit dem in Abbildung 6.2 gezeigten Schaltplan und entsprechender Beschaltung (Kapitel 3.1.2) durchgeführt worden. Aus dem Simulationsergebnis (Abbildung 6.8) wurde mit Hilfe einer Approximation (Abbildung 6.9) die Funktion $FG2 = FG2(FG1)$ berechnet. Diese Funktion ist in Zeile 280 im Auswertungsprogramm eingefügt:

$$V_{FG2} = 1.2647E{-}01 \cdot V_{FG1}^3 - 4.7997E{-}01 \cdot V_{FG1}^2 + 1.2841E{+}00 \cdot V_{FG1} + 2.2760E{+}00$$

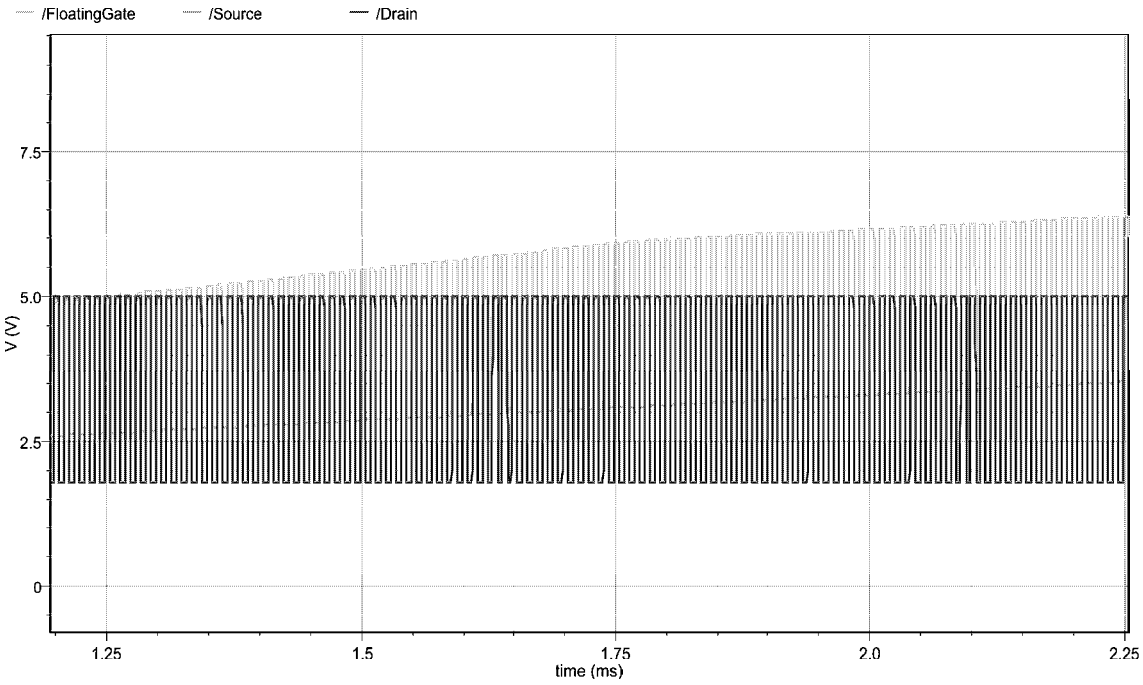

Abbildung 6.8: Simulationsergebnis der Standardzelle mit Entladebeschaltung. Gezeichnet ist in Blau die Floating Gate Spannung, in Grün die Sourcespannung und die Drainspannung in Violett.

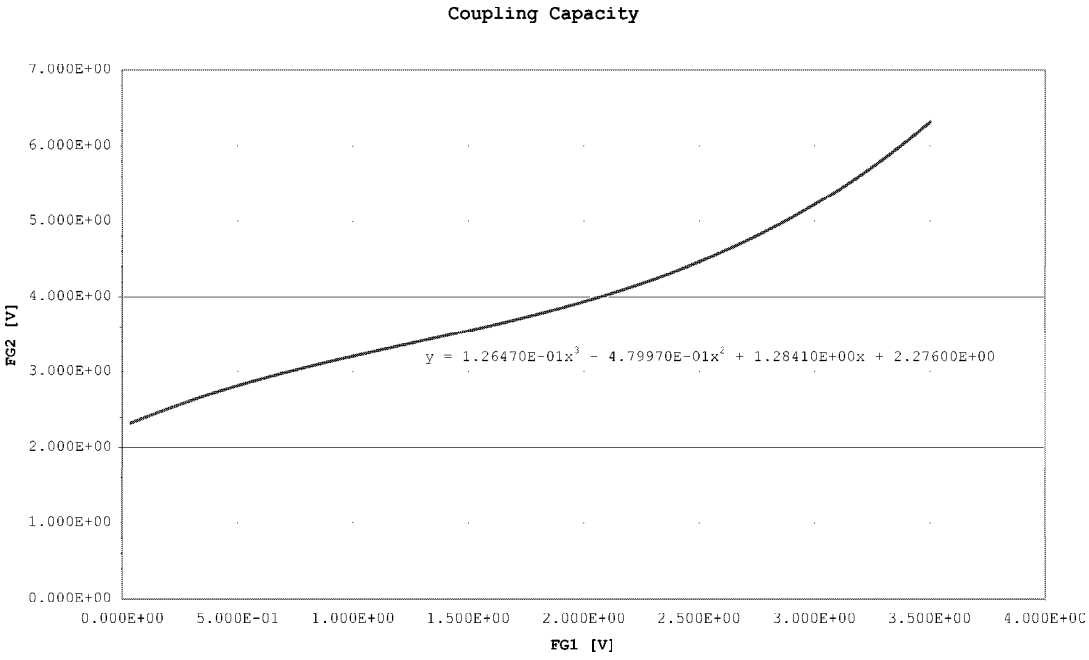

Abbildung 6.9: Dargestellt ist die Floating Gate Spannung während der Entladepulse in Abhängigkeit der Floating Gate Spannung während der Auslesepulse und die lineare Approximation im Arbeitsbereich.

Die folgende Abbildung zeigt die Tunnelströme beim Entladen von 420 verschiedenen Zellen von vier Testchips:

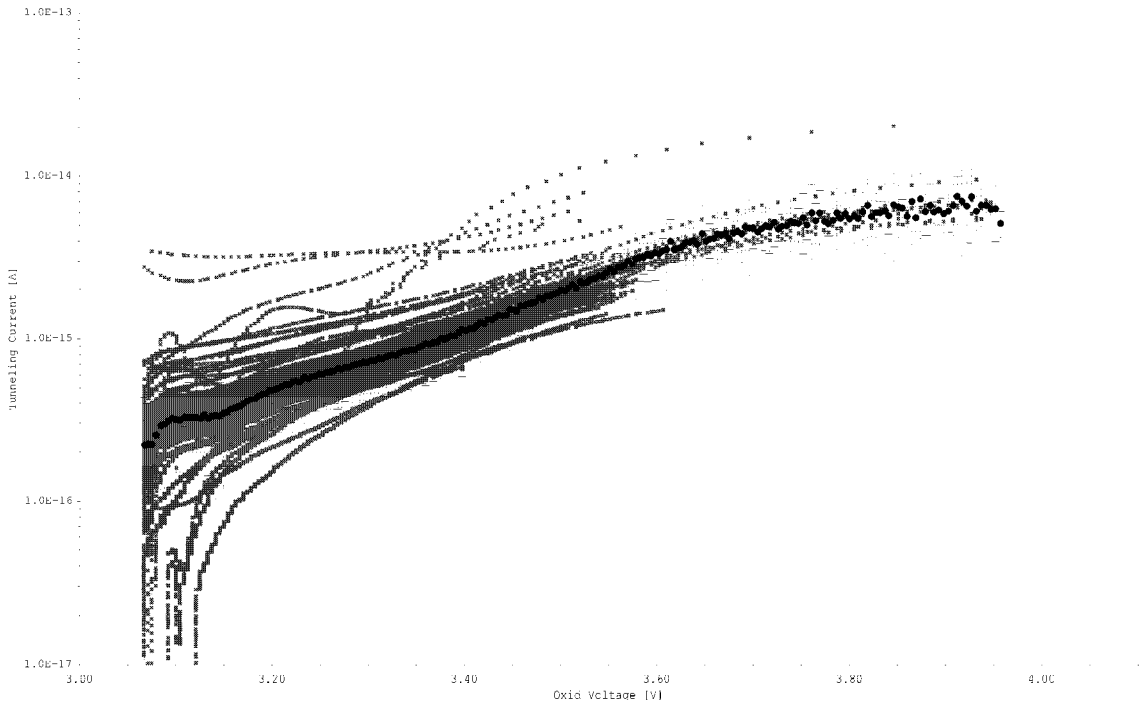

Abbildung 6.10: Entladeströme von 420 Standardspeicherzellen vier verschiedener Testchips in Abhängigkeit der Oxidspannung.

6.1.3 Leckstrom

Tunnelströme treten nicht nur während der Lade- und Entladevorgänge auf, sondern auch im adressierten Auslesezustand. Dadurch verringert sich die Auslesespannung einer beschriebenen Speicherzelle in Abhängigkeit der Auslesezeit. Auch dieser Prozess muss durch das Tunnelstrommodel für den Entladevorgang beschrieben werden können. Allerdings sind die Oxidspannungen deutlich geringer als im Entladevorgang. In Abbildung 6.11 sieht man schwarz eingezeichnet die mittlere Zerfallskurve beim dauerhaften Auslesen von 50 verschiedenen Zellen. Die Mittelung über die gemessenen Zellen wurde analog zum Lade- bzw. Entladevorgang mit Hilfe eines C-Programms lfitting.c durchgeführt (Anhang B5). Dabei wurde für jeden Zeitpunkt der einfache Mittelwert der noch verbleibenden Spannung errechnet und die Standardabweichung bestimmt.

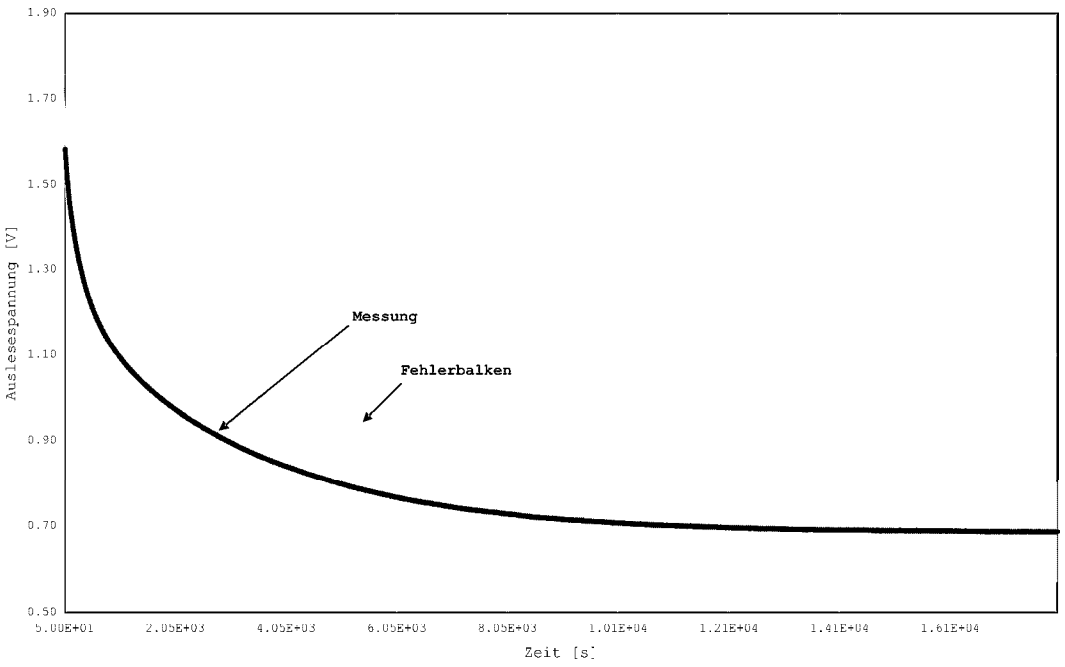

Abbildung 6.11: Aufzeichnung des Auslesespannungsverhaltens einer auf 1.7 Volt geladenen adressierten Standard-Speicherzelle mit der Zeit (schwarz) und die Spectre Simulation der Zelle mit Hilfe der VerilogA Modelle (rot).

6.2 Vermessung der Standardzelle mit Lasttransistor

Der Lade- und Entladevorgang der Speicherzelle mit drei Transistoren (siehe Kapitel 3.3) arbeitet analog zur Funktionsweise der oben beschriebenen Standardzelle. Beim Schreibvorgang werden 5 Volt Pulse auf das Control Gate gegeben, während Drain und Source geerdet sind. Umgekehrt wird beim Löschvorgang das Control Gate auf Masse gelegt und die 5 Volt Pulse auf Drain und Source gegeben. In dieser Konfiguration werden zusätzlich die Drain- und Sourceknoten des nMOS-Lasttransistors (Länge: 180nm, Breite: 2480nm) mit dem Sourceanschluss des Auslesetransistors (Länge: 180nm, Breite: 240nm) kurzgeschlossen. Die Beschaltung des Tunneltransistors (Länge: 180nm, Breite: 240nm) ist unverändert. Die nachfolgenden Abbildungen 6.12, 6.13 zeigen das Lade- bzw. Entladeverhalten einer Zelle. Die Auslesespannung ist durch blaue, die Drainspannung durch rote und das Control Gate durch gelbe Linien dargestellt:

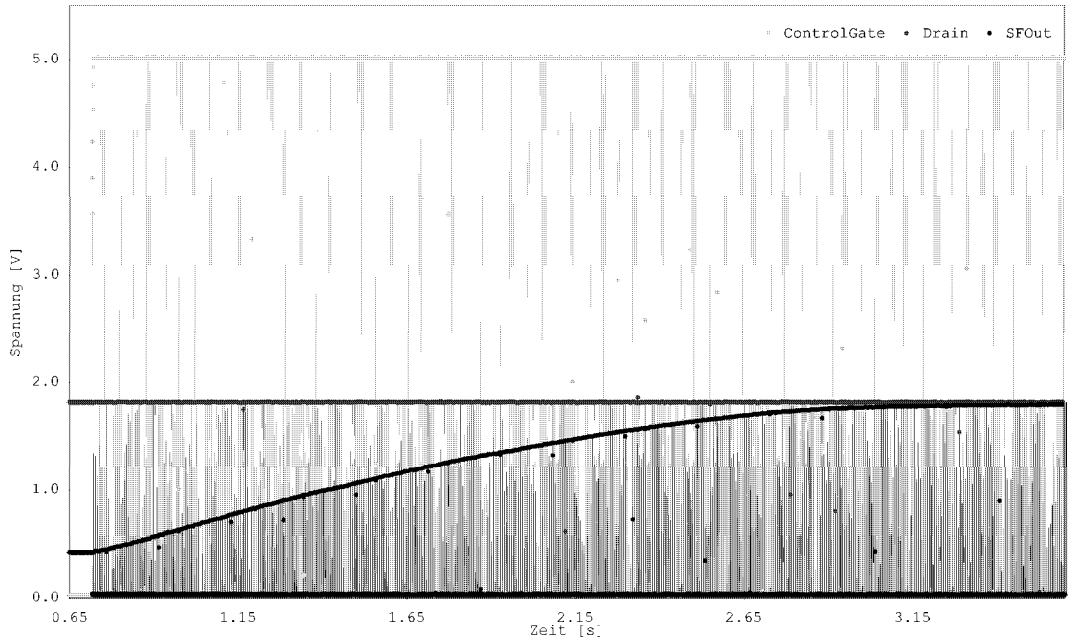

Abbildung 6.12: Rohdaten einer Speicherzelle mit zusätzlichem Lasttransistor beim Laden von 0.4 auf 1.8 Volt Auslesespannung.

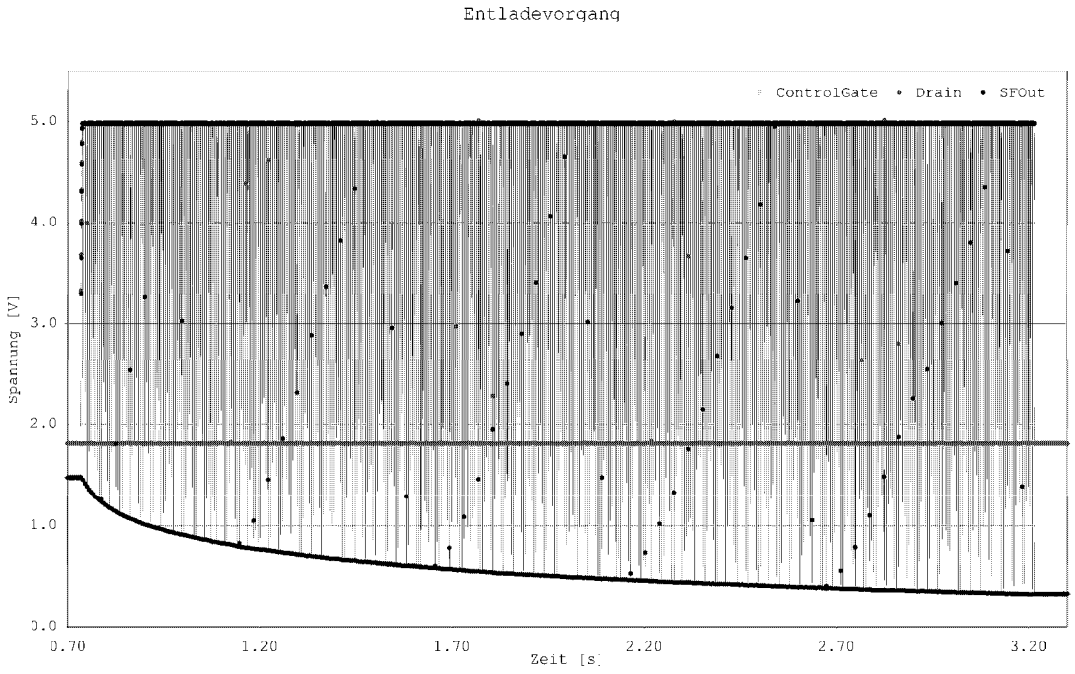

Abbildung 6.13: Rohdaten einer Speicherzelle mit zusätzlichem Lasttransistor beim Entladen von 1.5 auf 0.3 Volt Auslesespannung.

Um diese zeitabhängigen Spannungswerte in Tunnelströme umzurechnen, werden wieder C-Programme verwendet, die analog zu denen in Kapitel 6.1.1 und 6.1.2 arbeiten. Allerdings haben sich die Kapazitätsverhältnisse der Zelle verändert. Deswegen müssen die Näherungen in den Zeilen 291, 292 und 293 durch Approximation einer neuen Spectre Simulation aktualisiert werden. Die entsprechenden Fitkurven sind in den Abbildungen 6.14 a-d zu sehen und in die entsprechenden Zeilen des Auswertungsprogramms eingesetzt. Die kompletten Programme sind im Anhang B5 zu finden.

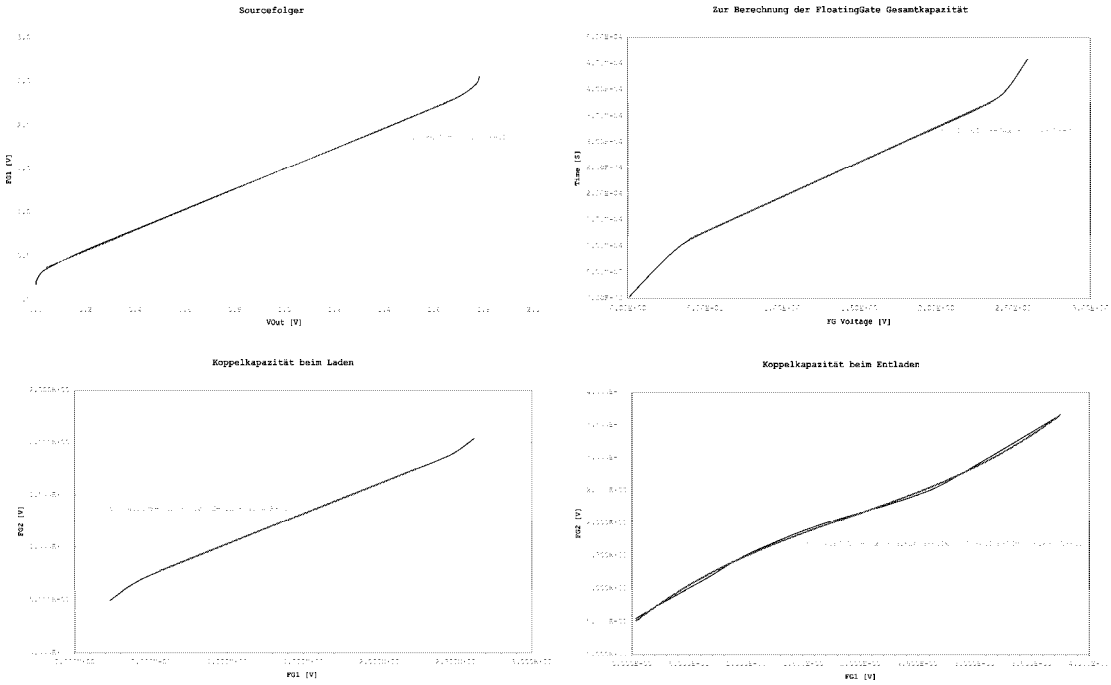

Abbildung 6.14: Approximationen aus der Simulation der Tranistorzelle mit zusätzlichem Lasttransistor und parasitären Koppelkapazitäten.

6.3 Vermessung von Hot Electron Injection

Bei keiner Zelle ist es gelungen, Hot Electron Injection zu zünden, um eine einmal beschriebene Zelle zu entladen. Auch mit Spannungen, die weit über den in der oben beschriebenen Theorie (Kapitel 2.2.3) angegebenen Werten lagen, hat sich kein Entladestrom eingestellt. Dies wurde mit Spannungen bis zu $V_{Drain} = 5$ Volt und $V_{CG} = 5$ Volt getestet.

6.4 Vergleich unterschiedlich dimensionierter Zellen

Die Schreib- und Löschgeschwindigkeit der Zellen hängt nach Kapitel 3.1 vom Verhältnis der Kapazitäten C_{nMOS} zu C_{pMOS} ab. Um dies zu verifizieren sind im Folgenden die gemessenen Ladevorgänge verschieden dimensionierter Zellen aufgezeichnet. Dabei wurden die Daten von jeweils 20 Zellen mit Hilfe des im Anhang B5 abgedruckten Programms mfitting.c gemittelt.

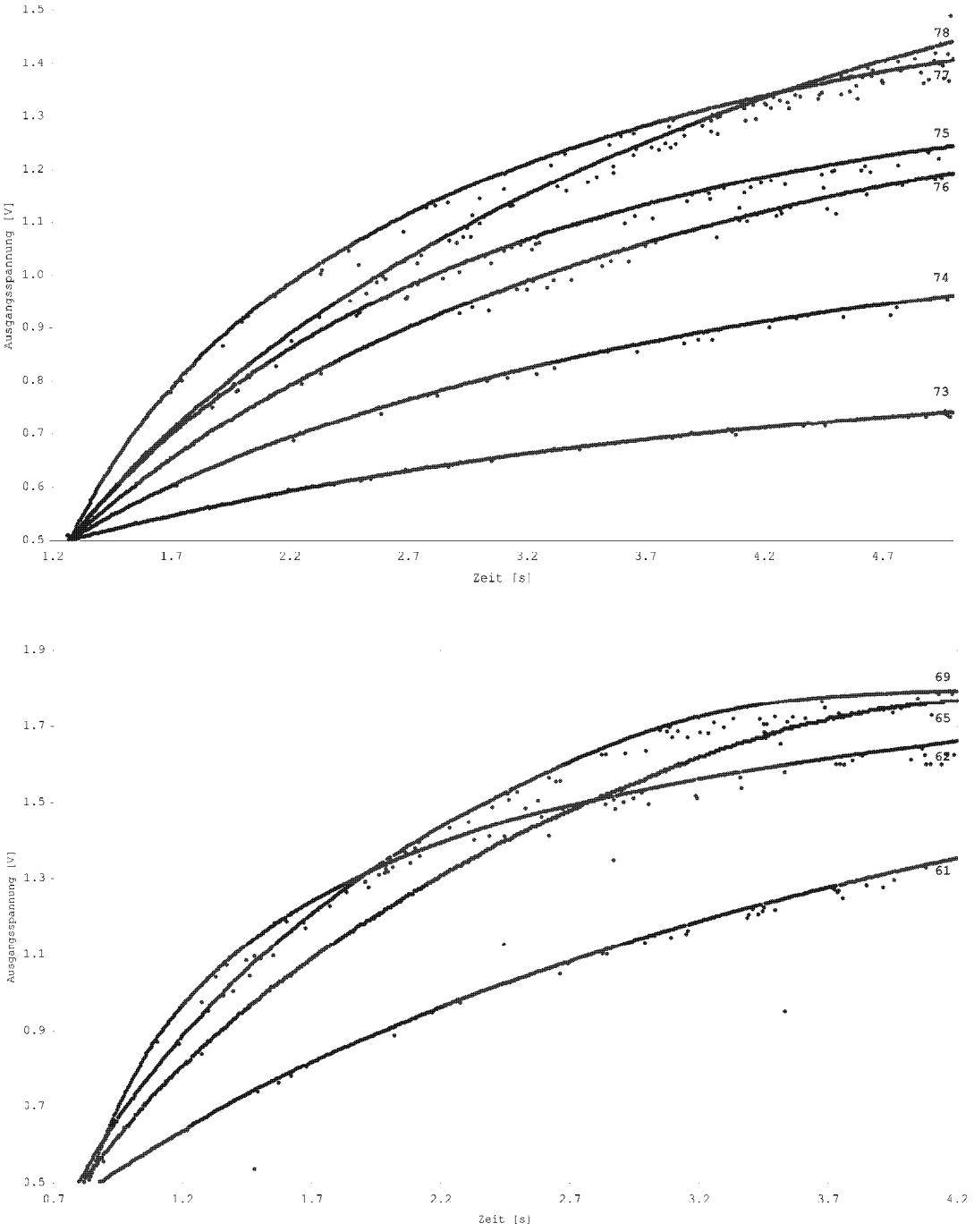

Abbildung 6.15: Vergleich der Ladeströme unterschiedlich dimensionierter Speicherzellen. Es wurden jeweils 20 Zellen vermessen.

Man sieht deutliche Abweichungen (Abb. 6.15) zwischen den Zellen zum einen in der Ladezeit und zum anderen im Verlauf der Kurven. Die nMOS Transistoren der Zellen Nr. 61, 62, 65, 69 haben die Dimensionen (Länge x Breite): (180 x 2480)nm, (210 x 2480)nm, (300 x 2480)nm, (420 x 2480)nm. Der pMOS Transistor ist bei allen Zellen gleicher Dimension (180 x 240)nm. Die Höhe der Auslesespannung nach 3.5 Sekunden entspricht der erwarteten Verteilung: Je höher die Kapazität des nMOS, desto schneller ist der Ladevorgang.

Bei den Zellen der Nummer 73, 74, 75, 76, 77, 78 hat der nMOS folgene Maße (Länge x Breite):
(180 x 1000)nm, (180 x 1250)nm, (180 x 1500)nm, (180 x 1750)nm, (180 x 2000)nm, (180 x 1250)nm. In Abbildung 6.15 ist zu sehen, dass wieder die Zellen größerer nMOS Kapazität, bis auf die Ausnahme bei den Zellen 75 und 76, schneller geladen sind. Auch die Zelle 77 zeigt bis zu einer Ladezeit von 3 Sekunden zunächst einen höheren Ladestrom als Zelle 78. Die Abweichungen sind durch herstellungsbedingte Schwankungen in den Dimensionierungen der Floating Gate Transistoren, als auch in den Dimensionierungen des Sourcefoglers zu erklären. Ebenso können relative Schwankungen in der Oxiddicke auftreten, die bei der exponentiellen Abhängigkeit der Tunnelwahrscheinlichkeit von dieser Größe große Auswirkungen auf die Schreibgeschwindigkeit haben. Mögliche Fehlerquellen sind im folgenden Kapitel ausführlich erläutert.

Kapitel 7

Ergebnis

Im Folgenden ist die Interpretation der gewonnenen Messdaten und die damit verbundene Modifikation der Tunnelstrommodelle zu finden.

7.1 Funktionsweise der Speicherzellen

Die Speicherzellen in den nominellen Größenverhältnissen arbeiten wie erwartet. Ausgenommen davon ist der Hot Electron Strom, der bei keiner Zelle gezündet werden konnte (Kapitel 6.3).
Es ist möglich, Zellen auf einen gewünschten Wert zwischen 0.3 und 1.8 Volt zu beschreiben und wieder zu löschen. Dies dauert im Mittel wenige Sekunden. Das Ausbleiben der CHE ist aufgrund des in beide Richtungen funktionierenden Fowler Nordheim und Direct Tunneling Stroms für den Einsatz der Zelle als Speichermedium unwichtig. Jedoch ist der schnelle Abfall der gespeicherten Werte auf dem Floating Gate durch einen Leckstrom im Auslesezustand so groß, dass nur Speicherzeiten von maximal millisekunden realisierbar sind. Dies gilt insbesondere für gespeicherte Werte, die größer als ein Volt sind.
Die Zellen in denen das Kapazitätsverhältnis $\frac{\sum C_{nmos}}{C_{pmos}}$ kleiner ist als in der Standardzelle, verhalten sich genauso wie diese, sie sind jedoch, wie erwartet, wesentlich langsamer. Für die Anwendung als Speicherzelle sollte dieses Verhältnis also unbedingt maximal sein.
Der Anteil der Zellen, die sich im Bereich zwischen 0.4 und 1.5 Volt überhaupt nicht beschreiben lassen, liegt in etwa bei 0.41 Prozent, d.h. 7 von 1700 Zellen. Das kann aber auch eine Folge des zur Auslese verwendeten Sourcefolgers sein, bei dem die relative fertigungsspezifische Schwankung durch die minimale Dimensionierung groß ist und folglich die Ausgangsspannung variiert.
Die Übereinstimmung der aufgenommenen Tunnelströme mit Referenzwerten in Abbildung 7.1 ist und erstaunlich gut und wird im nachfolgenden Kapitel erläutert. Zur Anpassung des theoretischen Modells aus dem Grundlagenkapitel musste jedoch der Vorfaktor A', die Barrierenhöhe Φ_0 und die effektive Masse

der Elektronen im Oxid m_{ox} angepasst werden.

7.2 Anpassung der Tunnelstrommodelle

Die gemittelten Daten der Lade- und Entladevorgänge von über 840 Standard Speicherzellen sind in der Abbildung 7.1 zu sehen.
Grundsätzlich lassen sich die Messwerte jedoch nicht wie erwartet vollständig durch das Fowler Nordheim Tunnelstrommodell erklären, da die Steigung sowohl für den Lade-, als auch für den Entladevorgang um ein Vielfaches zu niedrig ist. Es wurde bereits erwähnt, dass dieses semi-empirische Modell durch die Approximation des Vorfaktors A, der effektiven Elektronenmasse m_{ox} im Siliziumdioxid und der Höhe der Si/SiO_2-Potentialbarriere Φ_0 durch Messdaten angepasst werden muss. Doch auch durch diese Anpassung kann das Model mit physikalisch sinnvollen Parametern die Messdaten nicht beschreiben. Dass die Messdaten grob in den richtigen Größenordnungen liegen müssen, ergibt sich aus dem Vergleich mit den in Abbildung 7.1 eingezeichneten Referenzwerten (gelbe Punkte) für einen Tunnelprozess mit 4.1nm Oxiddicke [6]. Bei diesen Literaturwerten ist jedoch nicht bekannt, in welchem Zustand sich der Halbleiter (Inversion, Akkumulation) während des Tunnelns befindet und wie er dotiert ist. Dass diese Werte oberhalb der in Kapitel 6 ermittelten Datensätze liegen, ist auf die geringere Oxiddicke zurückzuführen. Die größte Schwierigkeit, bei der Approximation der Messdaten durch das Fowler Nordheim Model, stellt die Steigung, der gegen eine logarithmische Y-Achse gezeichneten Exponentialfunkton, dar. Diese ist bei der gemessenen und bei der in der Literatur angegeben Messkurve vergleichbar.

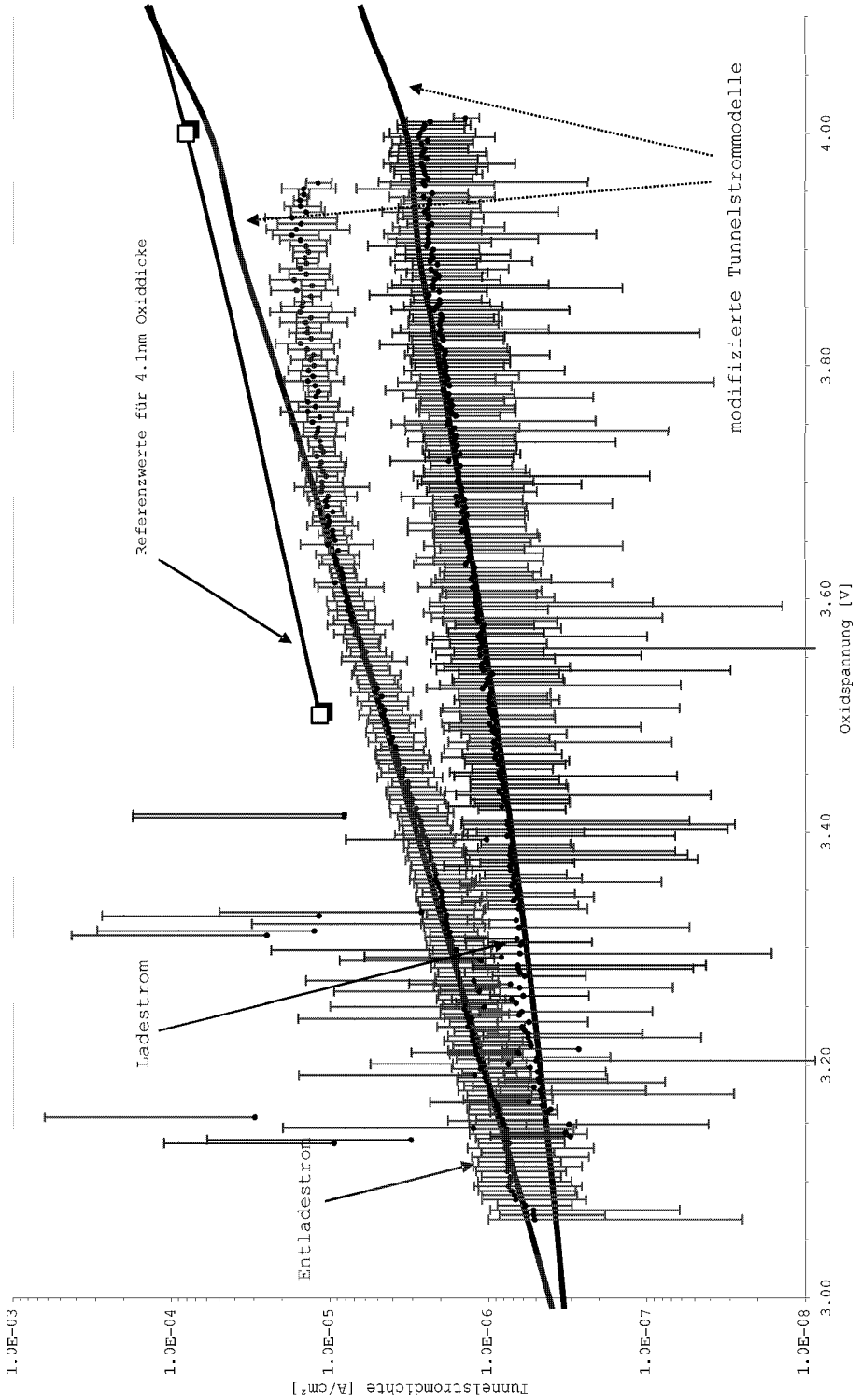

Abbildung 7.1: Gemittelte Lade- (blau) und Entladeströme (rot), sowie die an die Daten angepassten theoretischen Tunnelstromodelle.

Nur durch die Einbeziehung des Direct Tunneling Modells aus Kapitel 2.2.2 ergibt sich in der Theorie eine Steigung in der Größenordnung von den gemessenen Daten und den Referenzwerten. Um die das Modell mit den gemessenen Daten in bestmögliche Deckung zu bringen, sind folgende Approximationen nötig: Die effektive Elektronenmasse wird in der Referenzliteratur mit Werten zwischen 1.7E-31 kg und 3.6E-31 kg [3] angegeben. In den oben gezeigten Modellen wurde die effektive Masse mit 2.7E-31 kg angenommen, da dieser Werte die beste Übereinstimmung mit den Messdaten zeigt und im Intervall der in der Literatur genannten Werte liegt.
Ebenso wurde die Barrierenhöhe Φ_0 an die Messdaten angepasst und auf den Wert Φ_0=4eV festgesetzt. Grundsätzlich ist die Größe für eine klassiche Si/SiO_2-Barriere sehr gut bekannt und liegt bei 3.12eV. In modernen Halbleiterprozessen sind die Gates jedoch aus polykristallinem Silizium hergestellt und folglich muss die Barrierenhöhe neu angepasst werden. In der Praxis geschieht dies ebenfalls durch die Approximation von Messdaten. Weiterhin beinhaltet das beschriebene Fowler-Nordheim Tunnelmodel nur Tunnelströme aus dem Leitungsband. Bei kleiner werdenden Oxidbarrieren (um die $t_{OX} = 3.6nm$) müssen ebenso Tunnelströme aus dem Valenzband berücksichtig werden, die eine Barriere von $\Phi_0 = 4.2eV$ zu überwinden haben [6]. Die Oxiddicke im verwendeten Prozess beträgt 4.1nm. Also kann ein Tunnelprozess aus dem Valenzband vermutlich nicht mehr unberücksichtigt gelassen werden. Zusätzlich gibt es die Möglichkeit, dass Löcher aus dem Valenzband des Silizums über die Barriere in umgekehrter Richtung tunneln. Die Barrierenhöhe würde in diesem Fall 4.5eV betragen. Doch auch in diesen Bereichen ist durch die Verwendung von polykristallinem Silizum für die Gates eine Approximation durch die Messdaten unausweichlich.

Mit diesen Modifkationen erzielt man mit Fitfaktoren

$$A'_{laden} = 3.9E-02, A'_{entladen} = 1.38E+02$$

die beste Übereinstimmung mit den Messdaten.
Die berechneten Vorfaktoren (Kapitel 2.2.1) betragen

$$A_{FN} = \frac{q^3 m_{Si}}{16\Pi^2 \hbar m_{OX} \Phi_0} = 2.4649E-07$$
$$B_{FN} = \frac{4\sqrt{2m_{OX}\Phi_0^3}}{3q\hbar} = 2.9747E10.$$

Für eine nMOS-Kapazität mit einem p^+-dotierten Polysilizium beträgt die Flatband-Spannung $V_{FB} = 0.14eV$.

Zur Berechnung der Kontaktspannung muss zwischen Lade- und Entladevorgang unterschieden werden:

Ladevorgang:
Der pMOS-Transistor ist während des Ladevorgangs, d.h. $V_{FloatingGate} = (0.5 - 3.5)V$ und $V_{ControlGate} = 5V$, in Inversion. Folglich wird die Bulk-Oxid Kontaktspannung der Anode Ψ_A nach Kapitel 2.2.1 (iii) berechnet durch

$$\Psi_A = 2\Psi_{Fermi,p/n} + 6\Psi_{thermisch} \text{ mit } \Psi_{thermisch} = \frac{k_B T}{q}$$

und für die n-dotierte Well

$$\Psi_{Fermi,n} = \Psi_{thermisch} \ln \frac{n_i}{N_A}$$

Entscheidend ist die Dotierung an der Oberfläche, weswegen die Kanaldotierung für N_A eingesetzt wird $N_A = N_{Kanal} = 3.7E17\frac{1}{cm^3}$.
Mit einer intrinsischen Ladungsträgerdichte $n_i = 1.14e10\frac{1}{cm^3}$ ergibt sich für die Kontaktspannung der Anode:

$$\Psi_A = -0.722V$$

Das Polysilizium ist sehr hoch dotiert ($N_{poly} = 1E23\frac{1}{cm^3}$). Also kann die Kontaktspannung der Kathode gemäß Kapitel 2.2.1 als Näherung gleich null gesetzt werden:

$$\Psi_C = 0V$$

Entladevorgang:
Ebenso verhält es sich im Entladezustand ($V_{FloatingGate} = (1-3.5)V$ und $V_{ControlGate} = 0V$). Der Halbleiter befindet sich in Akkumulation und so wird (Kapitel 2.2.1 (i))

$$\Psi_C = 0V$$

angenommen. Auch in diesem Fall ist die Kontaktspannung des Polysilizium aus oben genanntem Grund gleich Null:

$$\Psi_A = 0 \text{ Volt.}$$

Da die vorangegangenen Approximationen sowohl für den Lade-, Entlade- als auch für den Leckstrom im Auslesevorgang in unterschiedlicher Konfiguration gelten müssen, ist der Spielraum der Parameteranpassung sehr begrenzt, d.h. es gibt keine andere physikalisch plausible Näherung der Faktoren aus Kapitel 2, die zu einer Übereinstimmung der Theorie mit den Messungen führen würde.

So liegt die Theorie, bis auf eine Abweichung zwischen 3.8 und 3.9 Volt beim Entladevorgang im Arbeitsbereich des Floating Gate Speichers, gut über den Messdaten und deutlich in den 68 Prozent Fehlerbalken. Die Abweichung im oberen Bereich der Messdaten ist vermutlich auf den Polynomfit Q(t) zurückzuführen. Dieser liegt bei großen Ladungsdifferenzen pro Zeit am unteren Rand der Messdaten und führt dadurch zu größeren Abweichungen in der Ableitung. Diese Abweichungen treten bei großen Oxidspannungen auf und sorgen für einen geringeren Strom. Jedoch ist es aufgrund der Vielzahl der Datensätze nicht sinnvoll, für jeden einzelnen Satz einen eigenen Fit und dessen Kontrolle durchzuführen.

Die starken Abweichungen zwischen den Zellen, die zu der großen Varianz führen, sind auf verschiedene Faktoren zurückzuführen. Zum einen kann die Struktur der Grenzfläche zwischen Si und SiO_2 auf der einen Seite und dem Polysilizium und dem SiO_2 auf der anderen Seite relativ zur geringen Oxiddicke von 4.2nm stark schwanken. Dadurch treten Dickenschwankungen im Oxid auf und es kommt, aufgrund der exponentiellen Abhängigkeit der Tunnelwahrscheinlichkeit von der Dicke, zu starken Stromschwankungen bei diesen Zellen. Außerdem bleiben in solchen sich ausbildenden Tunnelkanälen Elektronen im Oxid stecken und beeinträchtigen die Isolierfähigkeit der $Si/SiO2$-Schicht [3]. Dieselben Effekte können auch bei dem nMOS-Tranistor auftreten und eine erhöhtes Direct Tunneling in entgegengesetzte Richtung bewirken.
Zum anderen birgt die Angabe der Tunnelfläche eine weitere Unsicherheit. Es ist schwierig abzuschätzen, wie weit das Gate über die dotierten Drain- und Sourcebereiche reicht und dort noch weitere Tunnelströme induziert, da die geometrische Anordnung stark vom Herstellungsprozess abhängt.
Des Weiteren sind die Prozessparameter, wie beispielsweise die Dotierung, nicht genau bekannt, da diese vom Hersteller UMC geschützt und nicht weitergegeben werden. Die Parameter wie Dotierung und Flatband-Spannung sind nur in grober Näherung angegeben oder geschätzt.
Eine weitere Quelle für Abweichungen stellt der Sourcefolger dar, der in minimaler Dimensionierung implementiert ist und so herstellungsbedingte große relative Schwankungen in Breite und Länge auftreten können, die die Ausgangskennlinie verändert.

Simuliert man nun die gesamte Speicherzelle, die das oben angepasste Tunnelmodell enthält, so verhält sich die Zelle und deren Ausgangsspannungen analog zu den vemessenen Zellen auf dem Testchip (Abb. 7.2, 7.3). Die VerilogA Modelle ermöglichen damit eine komplette Spectre-Simulation des Standard Floating Gate Transistors mit allen auftretenden Tunnelströmen unabhängig davon, in welcher Schaltung der Transistor realisiert ist. Damit ist das Hauptziel der Diplomarbeit erreicht und die Speicherzelle könnte in ein neuronale Netz eingebaut und simuliert dort werden.

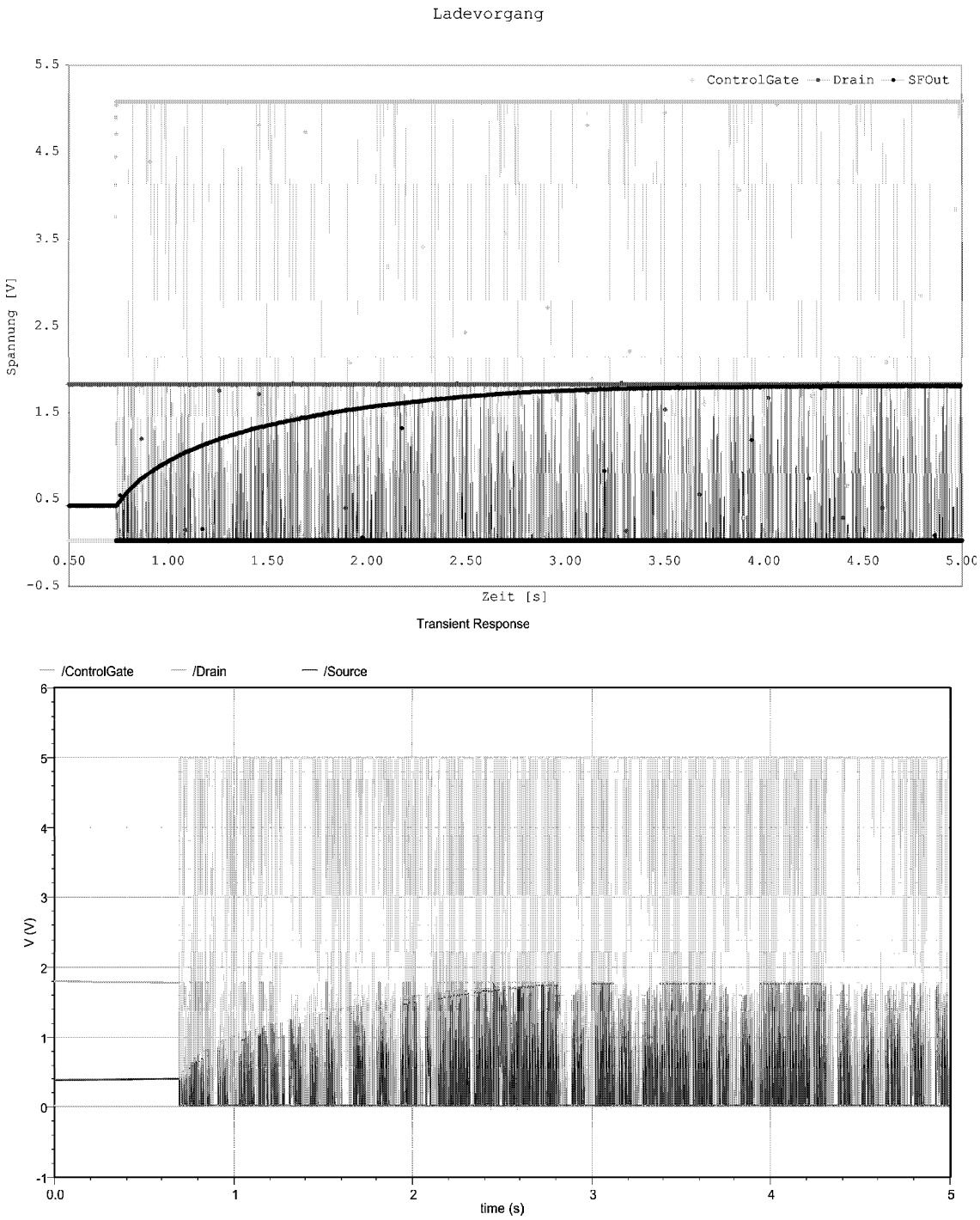

Abbildung 7.2: Vergleich zwischen gemessener (oben) und simulierter (unten) Ausgangsspannung im Ladevorgang.

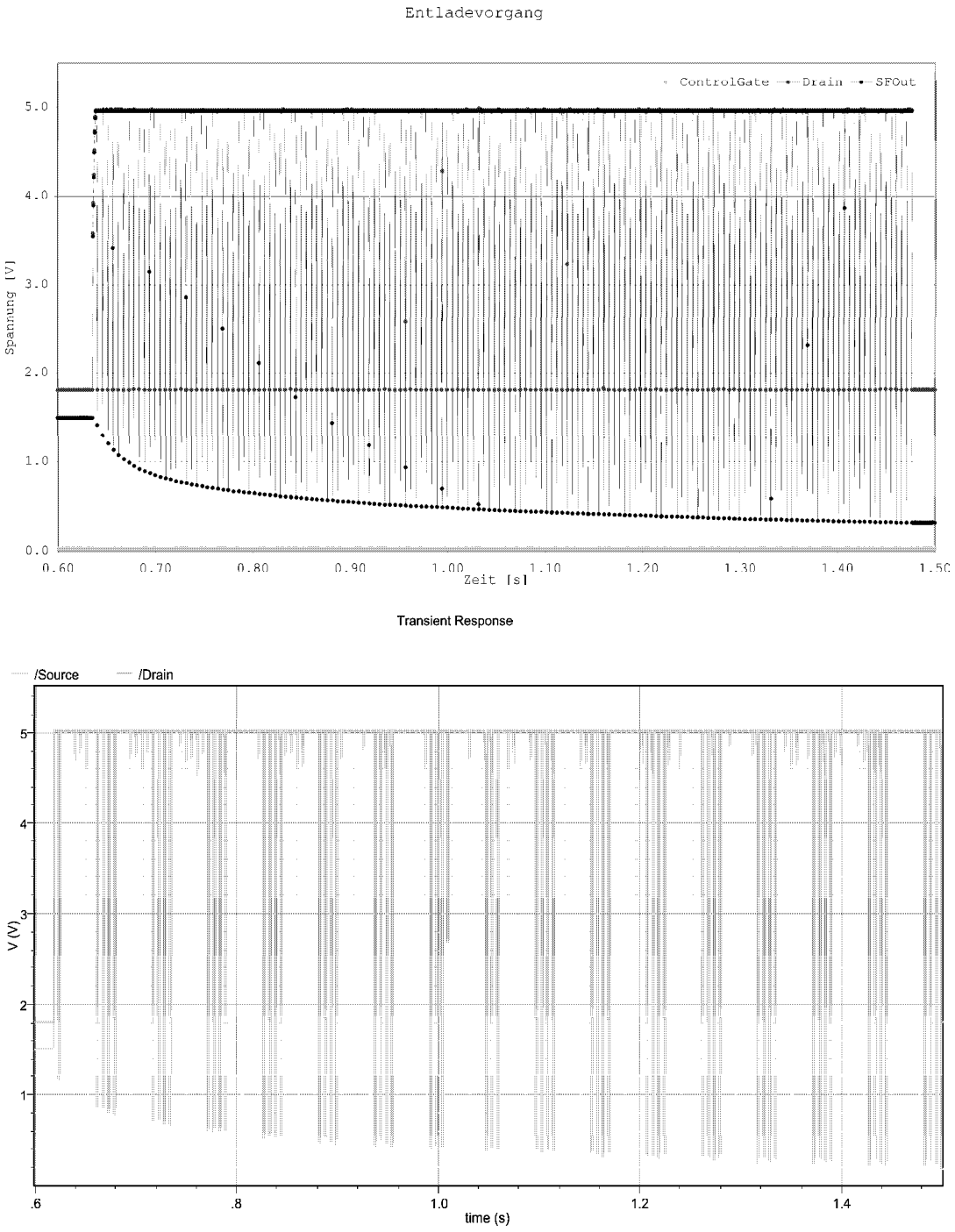

Abbildung 7.3: Vergleich zwischen gemessener (oben) und simulierter (unten) Ausgangsspannung im Entladevorgang.

7.2.1 Verifikation des Tunnelstrommodells durch eine Leckstrom Simulation

Wird die Schaltung der vermessenen Standardzelle simuliert (bspw. in Spectre) und das oben beschriebene, angepasste Tunnelstrommodell genutzt, so kann man einen Abfall der Auslesespannung beobachten (rote Linie in Abbildung 7.4), die dann nur eine geringe Abweichung von den gemessenen Daten (schwarze Linie) aufweist (Kapitel 6.1.3). Besonders wenn man die großen Fehler (grau) der Messungen betrachtet, ist die Übereinstimmung innerhalb der 1σ Fehlertoleranz akzeptabel.

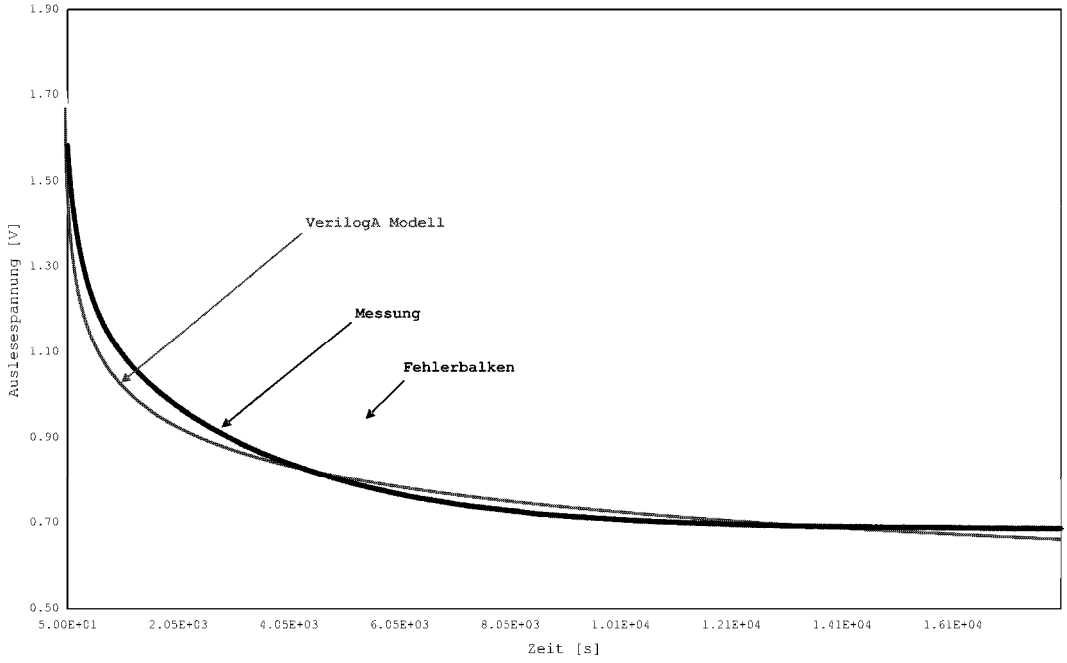

Abbildung 7.4: Aufzeichnung des Auslesespannungsverhaltens einer auf 1.7 Volt geladenen adressierten Standard-Speicherzelle mit der Zeit (schwarz) und die Spectre Simulation der Zelle mit Hilfe der VerilogA Modelle (rot).

Rechnet man nun die Spannungsänderung in Abhängigkeit der Zeit in einen Tunnelstrom, abhängig von der Oxidspannung nach dem gleichen Verfahren wie beim Entladevorgang in Kapitel 6.1.2, nur ohne die Koppelkapazitätberechnung (da es keine Lese- oder Ladepulse gibt) aus, ergibt sich folgende Grafik (Abbildung 7.5):

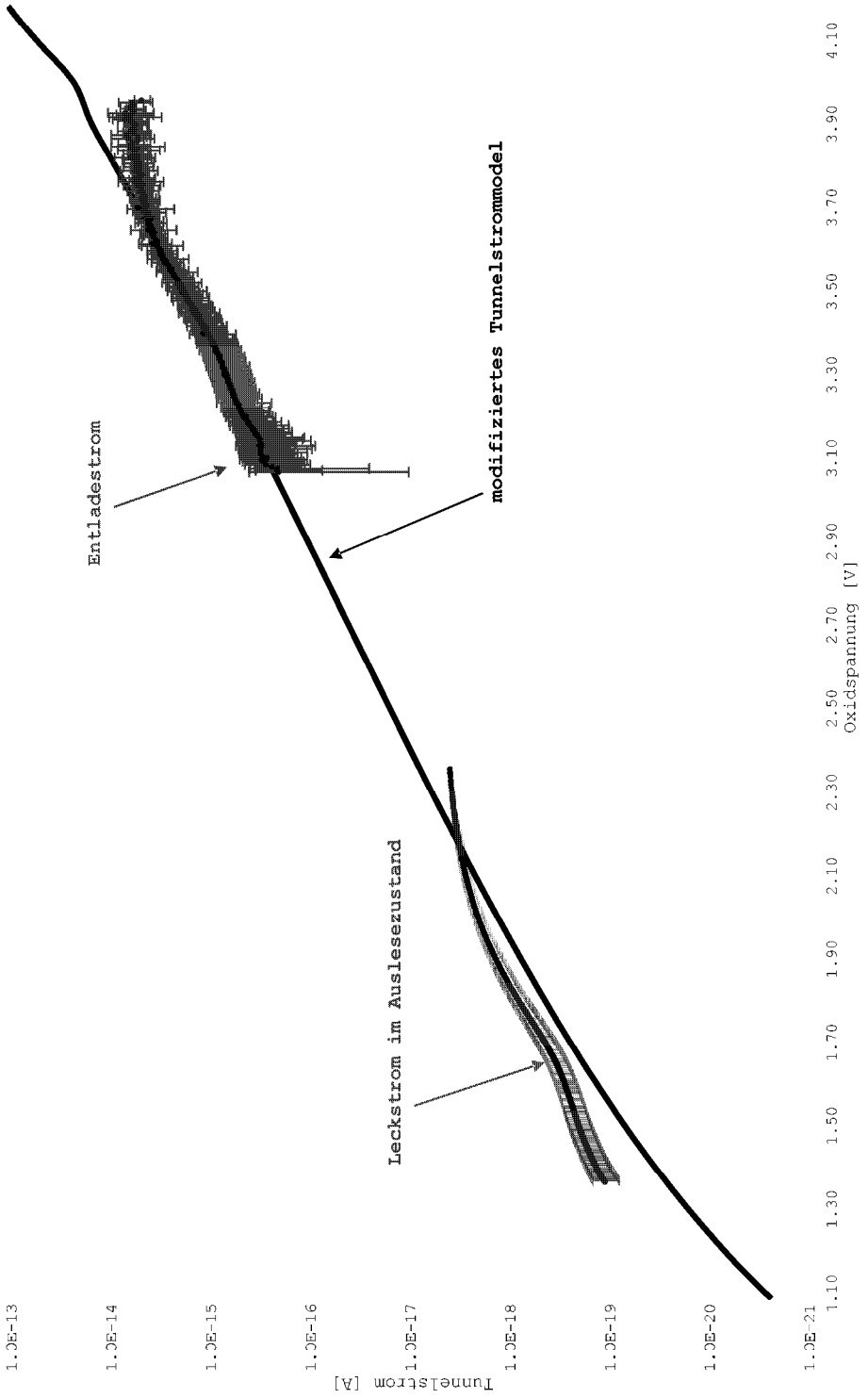

Abbildung 7.5: Aufzeichnung des Entladestroms und des Leckstroms im Lesezustand (rot). Das angepasste theoretische Tunnelstrommodel ist schwarz gezeichnet.

Das Tunnelstrommodell liegt in der gleichen Größenordnung und zeigt eine ähnliche Abhängigkeit von der Oxidspannung, wie die in Tunnelströme umgerechneten Messdaten. Im oberen Bereich des Auslesezustands (2.0 - 2.3 Volt) weicht die Steigung der Messpunkte stärker vom Model ab, dies hängt vermutlich mit der in diesem Bereich schlechteren Approximation des Polynomfits der Ladung gegen die Zeit zusammen (siehe Kapitel 6.1.2). Dass die Messdaten insgesamt oberhalb der Theorie liegen, lässt sich durch einen in der Realität zusätzlich vorhandenen kleinen Leckstrom im nMOS-Auslesetransistor erklären, den das eingezeichnete pMOS-Tunnelstrommodel nicht berücksichtigt. Im nMOS ist die Spannungdifferenz zwischen Floating Gate und Drain bzw. des Kanals im Bereich 0.5 bis 1 Volt. Bei dieser Oxidspannung treten bereits Leckströme durch Direct Tunneling auf, die die Stromdifferenz erklären könnten.

7.2.2 Verifikation des Modells mit Hilfe der Lasttransistorzelle

Vergleicht man die gemessenen Daten der Standardzelle (Kapitel 3.1) und der Standardzelle mit zusätzlichem Lasttransistor (Kapitel 3.3), so kann die Datenauswertung mit Hilfe der Spectre Simulationen verifiziert werden, da für den pMOS Transistor dieser Zelle die gleichen Ergebnisse zu erwarten sind.
Das Ergebnis der Datenauswertung aus Kapitel 6.2 ist in Abbildung 7.6 zu sehen. Die gemessenen Kurven liegen sowohl für den Lade- als auch für den Entladestrom unterhalb der Standardzellendaten, aber noch innerhalb der Fehlerbalken. Grundsätzlich ist zu sehen, dass der Weg der Datenaufnahme und -auswertung auch für einen anderen Zelltyp vergleichbare Ergebnisse liefert. Jedoch sind die Simulation in Spectre und auch die Approximation für den Sourcefolger, die Koppelkapazitätsberechnung, die Gesamtkapazitätsberechnung aus den Simulationsergebnissen und auch der Polynomfit der Ladung gegen die Zeit potentielle Fehlerquellen, aus denen vermutlich die Abweichung der Daten hervorgeht.
Zusätzlich zu den Fehlern, die bereits bei der Standardzelle diskutiert worden sind, ist bei diesem Zelltyp eine größere Abweichung bei der Bestimmung der Floating Gate Spannung zu erwarten, da der Auslesetransistor nun die minimale Größe (Länge:180nm, Breite:240nm) hat und dadurch herstellungsbedingte Schwankungen in der Länge und Breite des Transistors relativ zur Gesamtgröße wesentlich größer sind. Zum anderen ist die Treiberstärke dieses Transistors deutlich geringer und es besteht die Möglichkeit, dass Spannungsabfälle in der Ausgangsleitung auf dem Weg zur Ni-DAQ-Karte auftreten.
Physikalisch sollte es keinen Grund geben, dass Tunnelströme in dieser Zelle kleiner ausfallen, als in der Standardzelle, da der Tunneltransistor in beiden Fällen von gleicher Dimension ist. Jedoch ist ein größerer nMOS Tunnelstrom, der dem pMOS-Tunnelstrom entgegewirkt, in dieser Zelle möglich, da es nun zwei nMOS-Tranistoren gibt. Doch sollte dieser Effekt bei den auftretenden Oxidspannungen

am nMOS erwartungsgemäß vernachlässigbar sein.

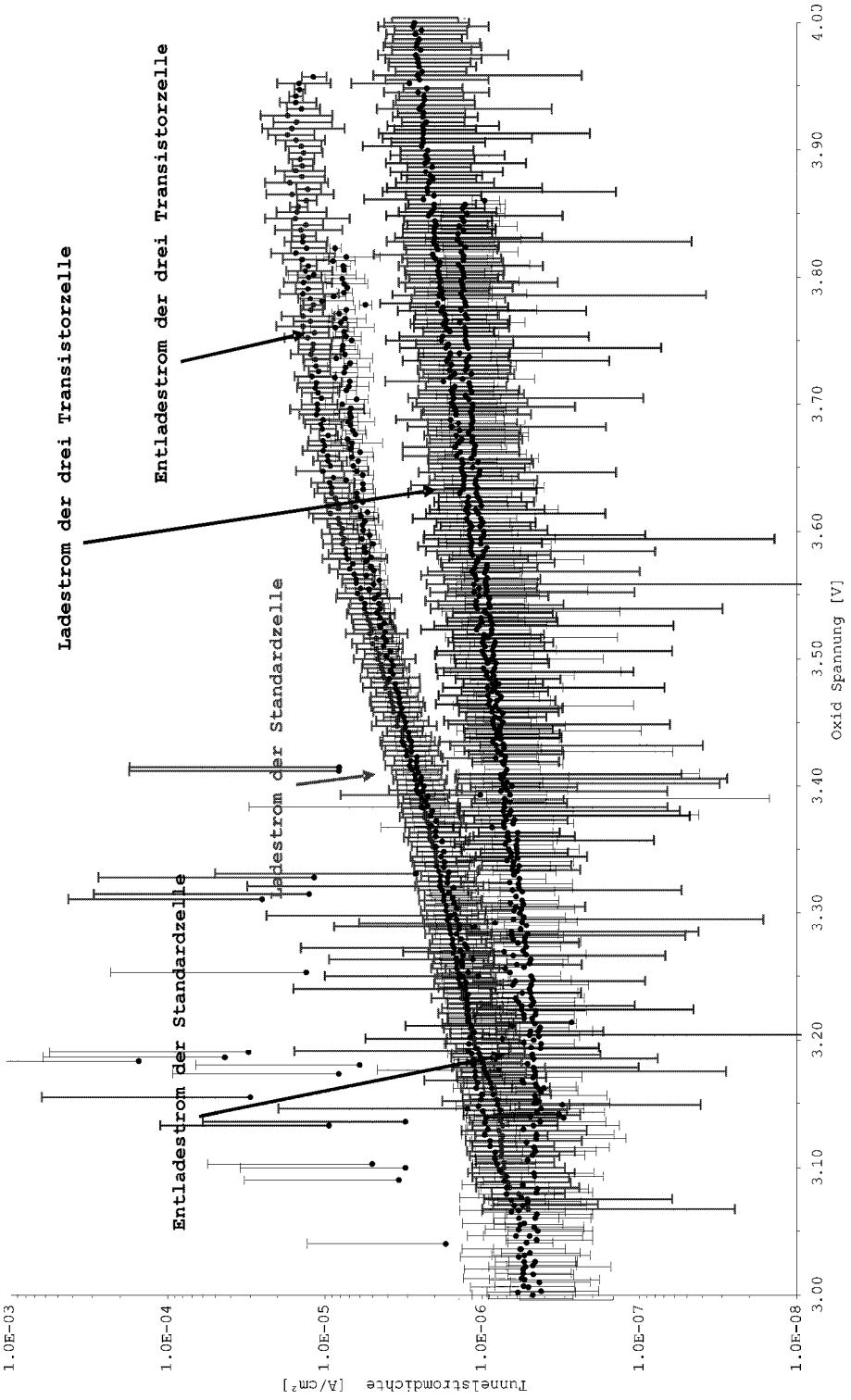

Abbildung 7.6: Vergleich der Lade- und Entladetunnelströme der Standardzelle (blau/rot) und der Speicherzelle mit zusätzlichem Lasttransistor (schwarz).

Eine Messung von Tunnelströmen größer dimensionierter pMOS Transistoren in der Zellenmatrix (Anhang B4) ist leider nicht möglich, da sich keine Speicherzelle der implementierten Größenordnungen beschreiben oder löschen lässt, die einen Tunneltransistor größer als Länge:180nm, Breite:240nm enthält. Um eine solche Zelle zu betreiben, müsste die Gesamtkapazität der nMOS-Tranistoren noch deutlich größer sein. Doch aufgrund der gewünschten minimalen Größe der Speicherzelle und der Tatsache, dass der pMOS nur als Koppelkapazität und nicht als Transistor fungiert, würde eine Vergrößerung des Tunneltransistors nur mit Nachteilen verbunden sein (bspw. größerer Platzverbrauch, Notwendigkeit höherer Lade- und Entladespannungen, geringere Speicherzeit (höhere Leckströme durch größere Tunnelfläche), etc.).
Folglich wird in einer Speicherzelle, betrieben nach dem obigen Prinzip (Kapitel 3), immer ein pMOS in minimaler Größe implementiert werden.

7.3 Implementierung der Tunnelmodelle in VerilogA

7.3.1 Sourcecode für Fowler Nordheim und Direct Tunneling

Zur Simulation des Schreib- und Löschvorgangs der Speicherzelle in Spectre, müssen die auftretenden Tunnelströme gemäß Abschnitt 7.2 durch VerilogA-Modelle modelliert werden.

Das mit Hilfe der Messungen angepasste theoretische Model aus Kapitel 2 ist im folgenden VerilogA Quellcode für den pMOS-Transistor implementiert:

```
1 VerilogA for FloatingGate, TUNNELINGCG18, veriloga
2
3 `include constants.h
4 `include discipline.h
5
6 module TUNNELING (vp, vn);
7
8 inout vp, vn;
9 electrical vp, vn;
10
11 parameter real tox = 42e-10; // oxid thickness [m]
12 parameter real temp = 293; // temperature [K]
13 parameter real nch = 3.7e17; // channel doping concentration [1/qcm]
14 parameter real length = 180e-9;
```

```
15 parameter real width = 240e-9;
16 parameter real k = 1.380658e-23; // Boltzmann [J/K]
17 parameter real q = 1.60217733e-19; // electron charge [C]
18 parameter real pi = 3.1415926535;
19 parameter real msi = 1.730784e-31; // electron effective mass in silicon [Kg]
20 parameter real mox = 2.7e-31; // electron effective mass in oxid layer [Kg]
21 parameter real h = 1.05457266e-34; // Planck constant [Js]
22 parameter real phi0 = 6.4e-19; // oxid barrier [J]
23 parameter real phib = 4; // oxid barrier [V]
24 parameter real vfb = 0.14; // flatband voltage [eV]
25 parameter real AC = 3.9E-2; // fitting paramter charge
26 parameter real AD = 1.38E2; // fitting paramter dscharge
27 parameter real ni = 1.14e10; // silicon intrinsic carrier concentration [1/qcm]
28
29 real A, B, vox, voxd, psi, it, fox;
30
31 analog begin
32
33 A = q*q*q*msi/(16*pi*pi*h*mox*phi0);
34 B = 4*sqrt(2*mox*phi0*phi0*phi0)/(3*q*h);
35 vox = V(vp) - V(vn);
36
37 if (vox > 2.3) begin // charging
38
39 psi = 2*k*temp/q*(ln(ni/nch)+3);
40 fox = (vox-vfb-psi)/tox;
41
42 if (vox < phib) it = AC*A*length*width*fox*fox*exp(-B/fox)*
43 1/((1-sqrt(1-vox/phib))*(1-sqrt(1-vox/phib)))*
44 exp(B/fox*sqrt((1-vox/phib)*(1-vox/phib)*(1-vox/phib)));// DirectTunneling
45 else it = AC*A*length*width*fox*fox*exp(-B/fox); // FNTunneling
46
47 end
48
49 if (vox < -0.5) begin // discharging
50
51 voxd = -1*vox;
52 fox = (voxd-vfb)/tox;
53 if (voxd < phib) it = -1*AD*A*length*width*fox*fox*exp(-B/fox)*
54 1/((1-sqrt(1-voxd/phib))*(1-sqrt(1-voxd/phib)))*
55 exp(B/fox*sqrt((1-voxd/phib)*(1-voxd/phib)*(1-voxd/phib)));// DirectTunneling
56 else it = -1*AD*A*length*width*fox*fox*exp(-B/fox); // FNTunneling
57
```

```
58 end
59
60 I(vp) <+ -1*it;
61 I(vn) <+ it;
62 end
63 endmodule
```

Der erste Teil der Programme dient lediglich der Definition von Konstanten. Die Größen V(vn) und V(vp) sind die Spannungen auf beiden Seiten der Tunnelbarriere und I(vp), I(vn) die jeweiligen Ausgangsströme. Nachdem die Vorfaktoren A und B, sowie die Oxidspannung vox in Zeile 33 bis 35 berechnet werden, modellieren die nachfolgenden Zeilen für positiven Werte von vox den Ladetunnelstrom und für negative Werte den Entladetunnelstrom. Erreicht die Oxidspannung die Barrierenhöhe Φ_0, so geht jeweils Direct Tunneling in reines Fowler-Nordheim Tunneling über (Zeile 42, 53). In Zeile 60 und 61 wird der berechnete Strom ausgegeben.

Das VerilogA-Programm für den nMOS-Transistor ist analog zum obigen Quellcode aufgebaut. Nur die Flatband Voltage, die Dimensionierung des Transistors und die Berechnung des Fermipotentials müssen angepasst werden: Die Flatband-Spannung einer pMOS-Kapazität mit n^+-dotiertem Polysilizium beträgt $V_{FB} = -0.14$ Volt. Bei einem p-dotierten Substrat befindet sich nun der p-dotierte Halbleiter während des Lade- und des Entladevorgangs in Inversion, da der Bulkanschluss in beiden Fällen auf Massepotential liegt. D.h. die Kontaktspannung berechnet sich in beiden Fällen gemäß Kapitel 2.2.1 (iii) für einen p-dotierten Halbleiter zu $\Psi_C = 1.02$ Volt. Für das Polysilizium wird sie aufgrund der hohen Dotierung in beiden Vorgängen wieder gleich Null gesetzt $\Psi_A = 0$ Volt.

```
1 VerilogA for FloatingGate, TUNNELINGDS18, veriloga
2
3 `include constants.h
4 `include discipline.h
5
6 module TUNNELING (vp, vn);
7
8 inout vp, vn;
9 electrical vp, vn;
10
11 parameter real tox = 42e-10; // oxid thickness [m]
12 parameter real temp = 293; // temperature [K]
13 parameter real nch = 3.7e17; // channel doping concentration [1/qcm]
```

```
14 parameter real length = 210e-9;
15 parameter real width = 2480e-9;
16 parameter real k = 1.380658e-23; // Boltzmann [J/K]
17 parameter real q = 1.60217733e-19; // electron charge [C]
18 parameter real pi = 3.1415926535;
19 parameter real msi = 1.730784e-31; // electron effective mass in silicon [Kg]
20 parameter real mox = 2.7e-31; // electron effective mass in oxid layer [Kg]
21 parameter real h = 1.05457266e-34; // Planck constant [Js]
22 parameter real phi0 = 6.4e-19; // oxid barrier [J]
23 parameter real phib = 4; // oxid barrier [V]
24 parameter real vfb = -0.14; // flatband voltage [eV]
25 parameter real AC = 3.9E-2; // fitting paramter charge
26 parameter real AD = 1.38E2; // fitting paramter dscharge
27 parameter real ni = 1.14e10; // silicon intrinsic carrier concentration [1/qm]
28
29 real A, B, vox, voxd, psi, it, fox;
30
31 analog begin
32
33 A = q*q*q*msi/(16*pi*pi*h*mox*phi0);
34 B = 4*sqrt(2*mox*phi0*phi0*phi0)/(3*q*h);
35 vox = V(vp) - V(vn);
36
37 if (vox > 2.3) begin // charging
38
39 psi = 2*k*temp/q*(ln(nch/ni)+3); 40 fox = (vox-vfb-psi)/tox;
41
42 if (vox < phib) it = AC*A*length*width*fox*fox*exp(-B/fox)*
43 1/((1-sqrt(1-vox/phib))*(1-sqrt(1-vox/phib)))*
44 exp(B/fox*sqrt((1-vox/phib)*(1-vox/phib)*(1-vox/phib))); // DirectTunneling
45 else it = AC*A*length*width*fox*fox*exp(-B/fox); // FNTunneling
46
47 end
48
49 if (vox < -0.5) begin // discharging
50
51 voxd = -1*vox;psi = 2*k*temp/q*(ln(nch/ni)+3);
52 fox = (voxd-vfb-psi)/tox;

53 if (voxd < phib) it = AD*A*length*width*fox*fox*exp(-B/fox)*
54 1/((1-sqrt(1-voxd/phib))*(1-sqrt(1-voxd/phib)))*
55 exp(B/fox*sqrt((1-voxd/phib)*(1-voxd/phib)*(1-voxd/phib))); // DirectTunneling
56 else it = AD*A*length*width*fox*fox*exp(-B/fox); // FNTunneling
```

```
57
58 end
59
60 I(vp) <+ -1*it;
61 I(vn) <+ it;
62 end
63 endmodule
```

Die Simulation der beschriebenen Modelle für beide Transistortypen ergibt folgende Tunnelströme (Abb. 7.8):

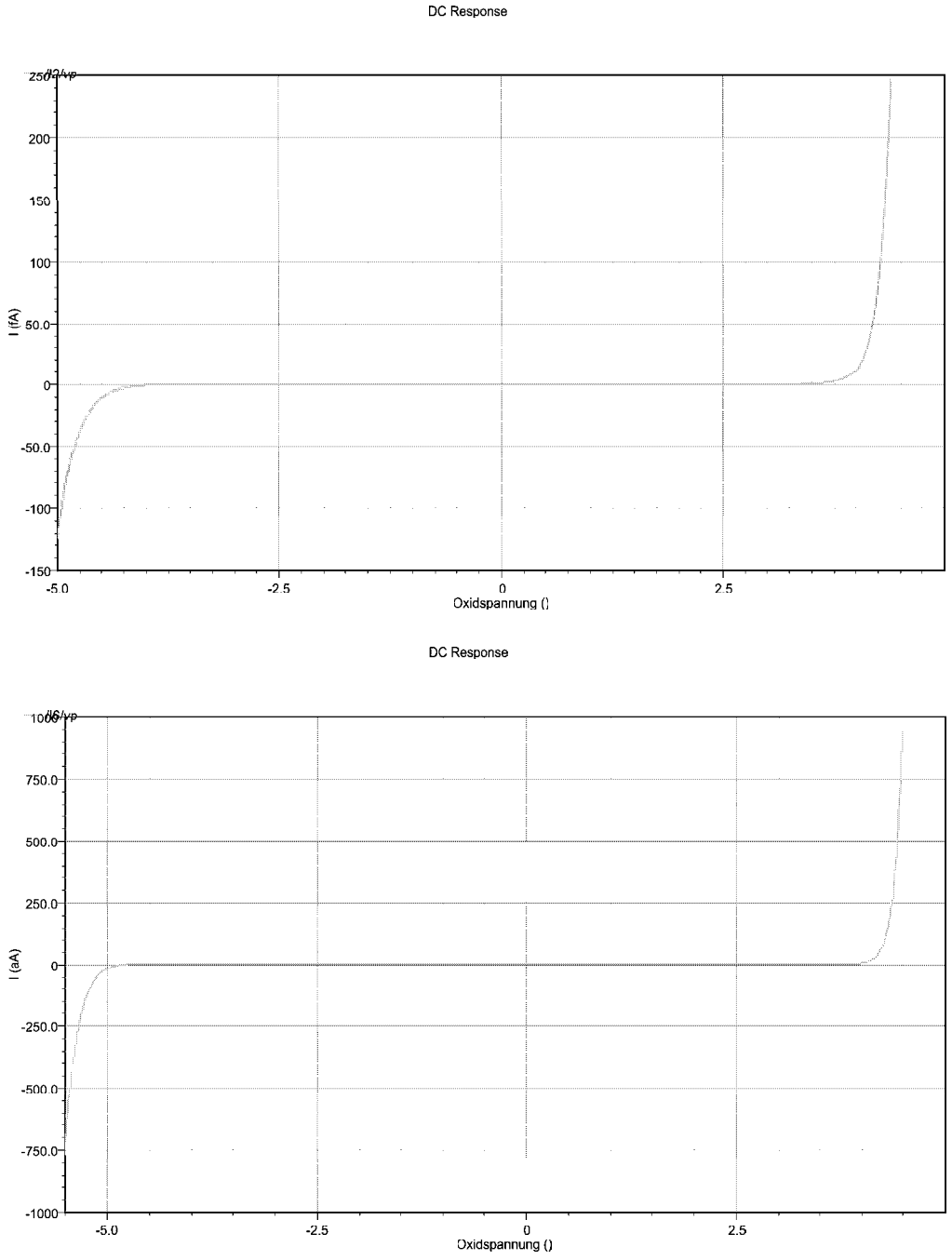

Abbildung 7.8: Simulation der modelierten Tunnelströme für einen pMOS- (grün) und einen nMOS-Transistor (blau) beim Be- (negative Oxidspannung) und Entladen (positive Oxidspannung).

7.3.2 Sourcecode für Hot-Electron Injection

Da der Hot-Elctron Strom nicht gezündet werden konnte, ist die Theorie aus Kapitel 2.2.3 zumindest für einen nMOS-Transistor der in dem Speicherarray verwendeten verschiedenen Dimensionen nicht richtig. Der aus der Theorie zunächst entwickelte Sourcecode zur Simulierung des CHE-Stroms wird der Vollständigkeit halber im Anhang A angegeben.

Kapitel 8

Zusammenfassung

Die nominelle Standardspeicherzelle arbeitet wie erwartet. Sie kann in wenigen Sekunden im Bereich von 0.3 bis 1.8 Volt durch Tunnelströme beschrieben und gelöscht werden. Damit wurde die prinzipielle Funktionsweise der Floating Gate Speicherzelle im Single-Poly UMC 0.18μm CMOS-Prozess gezeigt. Die Vermessung der auftretenden Ströme hat die Anpassing der Tunnelstrommodelle und deren Einbindung in VerilogA-Elemente ermöglicht. So ist es möglich den Floating Gate Transistor mit den auftretenden Tunnelströmen in Simulationssoftware (bspw. Spectre) vollständig zu simulieren.
Der Hot Electron Injection Mechanismus konnte bei keiner Zelle beobachtet werden.
Eine Nutzung als nichtflüchtiger Langzeitspeicher in neuronalen Netzen ist jedoch nicht möglich, da aufgrund des großen Leckstroms die gespeicherten Spannungswerte nur im Millisekunden Bereich gehalten werden.
Die Zelle ist im Vergleich zu Digitalspeichern mit Analog-Digital-Konvertern jedoch platz- und stromsparender.
Um eine längere Speicherzeit der Zellen zu erzielen, muss eine größere Oxiddicke verwendet werden. Im gleichen UMC 0.18μm - Prozess gibt es einen 3.3 Volt Transistor, der mit einer Oxiddicke von 7nm arbeitet. Dieser Transistor wurde auf dem Testchip zum Schalten der 5 Volt Signale in den Transmission Gates genutzt.
Setzt man die Floating Gate Zelle aus diesen Transistoren zusammen, so wird der Leckstrom um Größenordnungen kleiner ausfallen und die Speicherzeit in gleichem Maße steigen. Vermutlich würde dies eine Langzeitspeicherung mit Lebensdauern, vergleichbar der digitalen und kommerziell eingesetzten Flashspeicher, die ebenfalls in der Größenordnung von 7nm Oxiddicke arbeiten, ermöglichen.
Allerdings müssen in diesem Fall voraussichtlich Programmierspannungen von bis zu 10 Volt geschaltet werden, die eine komplexere Steuerschaltung (bspw. mit mehreren Triplewells) erfordern. Die Übereinstimmung der Theorie mit den Messungen sollte bei dickerem Oxid ebenfalls besser sein, da die Modelle für Fowler Nordheim Tunneling ursprünglich für diese Oxiddicken entwickelt wurden und

die relative Abweichung an den Si/SiO_2-Oberflächen geringer ist.
Für die Weiterentwicklung dieser Speicherzelle sind die vorgenommenen Messungen jedoch sehr aufschlussreich, da bei Verwendung von 3.3 Volt Transistoren im gleichen Herstellungsprozess lediglich die Oxiddicke in der Theorie korrigiert werden muss.

Anhang A

VerilogA Sourcecode für Hot-Electron-Injection

Der Sourcecode zur Simulierung des Hot-Electron Stroms sieht wie folgt aus:

```
1 VerilogA for FloatingGate, CHEcurrent18, veriloga
2 `include constants.h
3 `include discipline.h
4 module CHEcurrent18 (vfg, vs, vd1, vd2);
6 parameter real length = 180e-9;
7 parameter real width = 240e-9;
8 parameter real xj = 1.6e-7; // junction depth [m]
9 parameter real phi0 = 4.64e-19; // oxid barrier height [J]
10 parameter real q = 1.6e-19; // electron charge [C]
11 parameter real tox = 42e-10; // oxid thickness [m]
12 parameter real lambda = 8.5e-9; // mean free path of hot electrons
13 parameter real fhot = 4e6; // fields strength for hot electrons [V/m]
14 parameter real kl = 1.02e-1; // constant [m**1/6]
15 parameter real vto = 0.3075; // threshold voltage
16 parameter real fstart = 1e7; // Minimum field strength for hot electrons[V/m]
17
18 input vs, vd1, vd2;
19 inout vfg;
20
21 electrical vs, vd1, vd2, vfg;
22
23 real iche, ids, dl, em, vdcrit, vd, vf, a, vstart;
24
25 analog begin
26
27 ids = (V(vd1)-V(vd2))/1e-9;
```

```
28 vd = V(vd2)-V(vs);
29 vf = V(vfg);
30 dl=kl*sqrt(xj)*exp(0.333*ln(tox));
31 a=phi0*dl/(q*lambda);
32 vdcrit = ((vf-vto)*fhot*length)/(vf-vto+fhot*length);
33 vstart = fstart*length;
34
35 if (vd > vdcrit + 0.01 AND vd > vstart AND vf > 0) begin
36 iche=ids*exp(-a/(vd-(((vf-vto)*fhot*length)/(vf-vto+fhot*length)))); // Model nach Pavan
37
38 // Diorio-Model: iche=ids*1.3e-5*exp(-155.75/(((vd-vf)+0.7)*((vd-vf)+0.7)+(V(vd2))));
39
40 end else iche = 0;
41
42 I(vfg) <+ iche;
43
44 end
45 endmodule
```

Als Eingang für dieses Programm dienen die Spannungen V(vd1),V(vd2), V(vs) und V(vcg) für Drain, Source und Floating Gate Spannung. Nachdem die Schwellspannung längs des Kanals zur Zündung von Hot-Electrons berechnet (Zeile 2) und überschritten (Zeile 8) wurde, bestimmt der Ausdruck in Zeile 10 den Drain-Source Strom, indem die Spannung vor V(vd1) und nach V(vd2) dem Widerstand R1 in Abbildung 3.1 abgegriffen wird. In Zeile 11 wird der Strom iche auf das Floating Gate gemäß Kapitel 2.2.3 berechnet.

Anhang B

Testplatine

B.1 Pad Liste

1	VDD	Stromversorgung für den Digitalteil
2	GND	Ground
3	ResDC	Reset für Spaltendekoder
4	SRCK	Clock für Spaltendekoder
5	CES	Pads ein-/auschalten
6	SFS	Schalter SFinext/SFoutext
7	ResDL	Reset für Zeilendekoder
8	Enable	Zeilendekoderadresse einschalten
9	CLK	Clock für Zeilendekoder
10	VCC	Stromversorgung für den Analogteil
11	GND	Ground
12	VDD	Stromversorgung für den Digitalteil
13	VB	Biasspannung für den Sourcefolger
14	NCLK	Noiseclock
15	VCC	Stromversorgung für den Analogteil
16	GND	Ground
17		keine Belegung
18		keine Belegung
19	VDD	Stromversorgung für den Digitalteil
20	VIHE	Biasspannung für den internen Controller
21	VDD	Stromversorgung für den Digitalteil
22	VOut	Ausgangsspannung des internen Controllers

23	GND	Ground
24	VRef	Referenzspannung für den internen Controller
25	VI	Biasspannung für den internen Controller
26	VBC	Biasspannung für den internen Controller
27	VRO	Biasspannung für den internen Controller
28	VRI	Biasspannung für den internen Controller
29	CK2	Clock für den internen Controller
30	CK1	Clock für den internen Controller
31	GND	Ground
32	VCC	Stromversorgung für den Analogteil
33	Res	Rücksetzen des internen Controllers
34	SCK	Controllerdekoder
35	ResReg	Zurücksetzen des Controllerdekoders
36	Start	Startsignal für den internen Controller
37	GND	Ground
38	VCC	Stromversorgung für den Analogteil
39	VDD	Stromversorgung für den Digitalteil
40	Stop	Stopsignal des internen Controllers
41	SFoutext	Sourcefolgerausgangsspannung
42	SFinext	Eingang für Sourcespannung
43	Dext	Drainspannung
44	CGext	ControlGateSpannung

B.2 SCSI-Steckerbelegung

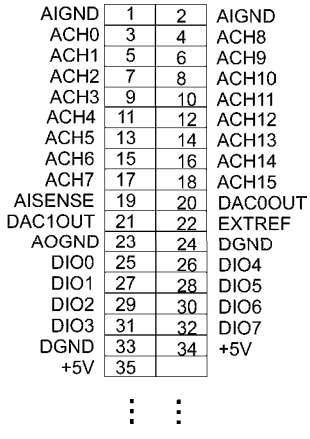

Abbildung B.2.1: Pinbelegung der ersten 35 Pins des 68-Pin-SCSI-Steckers.

DIO0	ResDC,ResDL,RES
DIO1	SCK
DIO2	CES
DIO3	Enable
DIO4	ResReg
DIO5	Start
DIO6	CLK
DIO7	SRCK
DAC0OUT	VRef
DAC1OUT	nicht belegt
ACH0	SFoutextout
ACH1	Stop
ACH2	Voutout
ACH3	CGextout
ACH4	Dextout
ACH5	SFoutext
ACH6	VRI
ACH7	Vout
ACH8	CGext
ACH9	Dext

B.3 Bestückungsliste der Testplatine

1x	SCSI Stecker
1x	Sockel PLCC44
8x	1x2 Jumper
7x	1x3 Jumper
2x	2x3 Jumper
2x	1x5 Jumper
2x	LM317 Spannungsregler
1x	LP2989LV Spannungsregler (1.8V)
2x	MAX4252 OPAMP (2 Kanal), 2 Kanal Operationsverstärker
1x	NMOS Transistor
7x	$10k\Omega$ Potentiometer
2x	$10nF$ Kapazität
17x	$100nF$ Kapazität
21x	$22\ \mu F$ Kapazität
2x	$100\mu F$ Kapazität
1x	$330k\Omega$ Widerstand
13x	$10k\Omega$ Widerstand
2x	$2.2k\Omega$ Widerstand

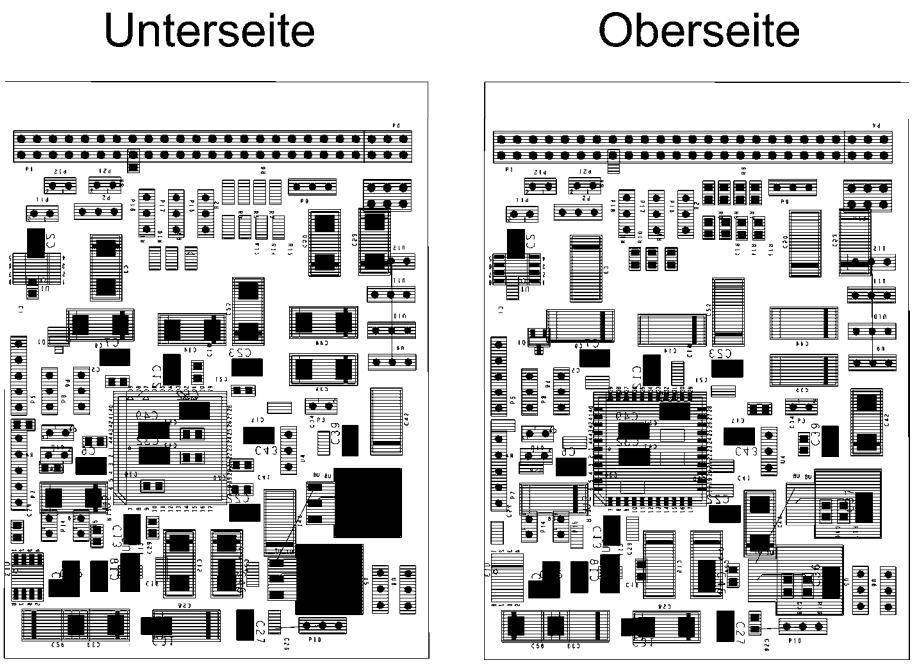

Abbildung B.3.1: Bestückungsplan der Platine. Die Elemente der jeweiligen Seite sind schwarz gezeichnet.

B.4 Dimensionen der Speicherzellen

	Auslesetransistor nMOS		Lasttransistor nMOS		Tunneltransistor pMOS	
Spalte	länge [nm]	breite [nm]	länge [nm]	breite [nm]	länge [nm]	breite [nm]
0	180	240	180	***250***	180	240
1	180	240	180	***500***	180	240
2	180	240	180	***750***	180	240
3	180	240	180	***1000***	180	240
4	180	240	180	***1250***	180	240
5	180	240	180	***1500***	180	240
6	180	240	180	***1750***	180	240
7	180	240	180	***2000***	180	240
8	180	240	180	***2250***	180	240
9	180	240	180	***2480***	180	240
10	***200***	240	180	2480	180	240
11	***220***	240	180	2480	180	240
12	***240***	240	180	2480	180	240
13	***260***	240	180	2480	180	240
14	***280***	240	180	2480	180	240
15	***300***	240	180	2480	180	240
16	***320***	240	180	2480	180	240
17	***340***	240	180	2480	180	240
18	***360***	240	180	2480	180	240
19	***380***	240	180	2480	180	240
20	***400***	240	180	2480	180	240
21	180	240	360	2480	***260***	240
22	180	240	360	2480	***360***	240
23	180	240	360	2480	***460***	240
24	180	240	520	2480	***260***	420
25	180	240	520	2480	***360***	420
26	180	240	520	2480	***460***	420
27	180	***240***	180	2480	180	240
28	180	***340***	180	2480	180	240
29	180	***440***	180	2480	180	240
30	200	***240***	180	2480	180	240
31	200	***340***	180	2480	180	240
32	200	***440***	180	2480	180	240
33	220	***240***	180	2480	180	240
34	220	***340***	180	2480	180	240
35	220	***440***	180	2480	180	240
36	240	***240***	180	2480	180	240
37	240	***340***	180	2480	180	240
38	240	***440***	180	2480	180	240
39	200	240	180	***250***	270	240
40	200	240	180	***500***	270	240
41	200	240	180	***750***	270	240
42	200	240	180	***1000***	270	240
43	200	240	180	***1250***	270	240
44	200	240	180	***1500***	270	240
45	200	240	180	***1750***	270	240
46	200	240	180	***2000***	270	240
47	200	240	180	***2250***	270	240
48	200	240	180	***2480***	270	240
49	220	240	180	***250***	360	240
50	220	240	180	***500***	360	240
51	220	240	180	***750***	360	240
52	220	240	180	***1000***	360	240
53	220	240	180	***1250***	360	240
54	220	240	180	***1500***	360	240
55	220	240	180	***1750***	360	240
56	220	240	180	***2000***	360	240
57	220	240	180	***2250***	360	240
58	220	240	180	***2480***	360	240
59	400	400	520	2480	460	420
60	***210***	2480			180	240
61	***180***	2480			180	240
62	***210***	2480			180	240
63	***240***	2480			180	240
64	***270***	2480			180	240
65	***300***	2480			180	240
66	***330***	2480			180	240
67	***360***	2480			180	240
68	***390***	2480			180	240
69	***420***	2480			180	240

70	180	***250***			180	240
71	180	***500***			180	240
72	180	***750***			180	240
73	180	***1000***			180	240
74	180	***1250***			180	240
75	180	***1500***			180	240
76	180	***1750***			180	240
77	180	***2000***			180	240
78	180	***2250***			180	240
79	180	***2480***			180	240
80	360	***250***			360	480
81	360	***500***			360	480
82	360	***750***			360	480
83	360	***1000***			360	480
84	360	***1250***			360	480
85	360	***1500***			360	480
86	360	***1750***			360	480
87	360	***2000***			360	480
88	360	***2250***			360	480
89	360	***2480***			360	480
90	360	***250***			570	480
91	360	***500***			570	480
92	360	***750***			570	480
93	360	***1000***			570	480
94	360	***1250***			570	480
95	360	***1500***			570	480
96	360	***1750***			570	480
97	360	***2000***			570	480
98	360	***2250***			570	480
99	360	***2480***			570	480
100	180	240	180	2480	180	240
101	210	2480			180	240
102	180	240	180	2480	180	240
103	210	2480			180	240
104	200	260	200	2480	200	260
105	200	260	200	2480	240	300
106	200	260	200	2480	280	340
107	230	2480			200	260
108	230	2480			240	300
109	230	2480			280	340
110	200	260	1000	5000	1000	480
111	200	260	2000	5000	2000	480
112	1000	5000			1000	480
113	2000	5000			2000	480

B.5 C-Programme zur Datenauswertung

Die Programme zur Datenauswertung sind in folgender Reihenfolge eingefügt:

charge.c
discharge.c
3VPcharge.c
3VPdischarge.c
fitting.c
lfitting.c
mfitting.c

charge.c

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#undef  abs
#define abs(x) (((x)<0.0)?(-(x)):(x))
#undef  max
#define max(x,y) (((x)>(y))?((x)):(y))
#undef  min
#define min(x,y) (((x)<(y))?((x)):(y))

FILE *datas;
int counter, fitcount,
savei0,savei1,savei2,redundant,vdii,vdif,f,kc,workc1,workcc;
double
E[],timex1,timex2,voxx1,voxx2,qx1,qx2,TIMEI[2500],QI[2500],VOXI[2500],timediff
,IF[2500],VOXF[2500],FEHLER[2500],IFQ[2500];
double
MFEHLER,cnr,vdi,obercounter[2500],vmax,vmin,QST[2500],IFF[2500],VOXFF[2500];

static void simeq(int n, double A[], double Y[], double X[]);

static int data_set(double *y, double *x){
        char * pEnd;
        double vox1,q1,vfq1,time1,vox2,q2,vfq2,time2,time,q,vox;
        char svox[100], sq[100], svfq[100], stime[100];

        double xx;
        double yy;

        if (fitcount == 0) {
                xx = TIMEI[savei0];
                yy = QI[savei0];
                savei0=savei0+1;
                if (savei0 >= counter) return 0;
        }
        if (fitcount == 1) {
                xx = VOXF[savei1];
                yy = IF[savei1];
                savei1=savei1+1;
                if (savei1 >= vdii) return 0;
        }
        if (fitcount == 2) {
                xx = VOXFF[savei2];
                yy = IFF[savei2];
                savei2=savei2+1;
                if (savei2 >= vdii) return 0;
        }
        *x = xx;
        *y = yy;

        return 1;
}

static void fit_pn(int n, double A[], double Y[], double C[]){
        int i, j, k;
        double x, y, t;
        double pwr[30]; /* at least 2n */

        for(i=0; i<n; i++)
        {
                for(j=0; j<n; j++)
                {
                        A[i*n+j] = 0.0;
                }
                Y[i] = 0.0;
        }
        while(data_set(&y, &x))  {
                        pwr[0] = 1.0;
                        for(i=1; i<2*n; i++) pwr[i] = pwr[i-1]*x;
```

```
                                charge.c
                        for(i=0; i<n; i++)
                        {
                                for(j=0; j<n; j++)
                                {
                                        A[i*n+j] = A[i*n+j] + pwr[i]*pwr[j];
                                }
                                Y[i] = Y[i] + y*pwr[i];
                        }
                }
        simeq(n, A, Y, C);
        for(i=0; i<n; i++); /*printf("C[%d]=%q \n", i, C[i]);*/
}

static void check_pn(int n, double C[],
        double *rms_err, double *avg_err, double *max_err){
        double x, y, ya, diff;
        double sumsq = 0.0;
        double sum = 0.0;
        double maxe = 0.0;
        double xmin, xmax, ymin, ymax, xbad, ybad;
        int i, k, imax;

        k = 0;
        while(data_set(&y, &x))  {
                        if(k==0)    {
                                        xmin=x;
                                        xmax=x;
                                        ymin=y;
                                        ymax=y;
                                        imax=0;
                                        xbad=x;
                                        ybad=y;
                                }
                        if(x>xmax) xmax=x;
                        if(x<xmin) xmin=x;
                        if(y>ymax) ymax=y;
                        if(y<ymin) ymin=y;
                        k++;
                        ya = C[n-1]*x;
                        for(i=n-2; i>0; i--)
                        {
                                ya = (C[i]+ya)*x;
                        }
                        ya = ya + C[0];
                        diff = abs(y-ya);
                        if(diff>maxe)    {
                                        maxe=diff;
                                        imax=k;
                                        xbad=x;
                                        ybad=y;
                                }
                        sum = sum + diff;
                        sumsq = sumsq + diff*diff;
                }
        *max_err = maxe;
        *avq_err = sum/(double)k;
        *rms_err = sqrt(sumsq/(double)k);
}

static void simeq(int n, double A[], double Y[], double X[]){

        double *B;
        int *ROW;
        int HOLD , I_PIVOT;
        double PIVOT;
        double ABS_PIVOT;
        int i,j,k,m;

        B = (double *)calloc((n+1)*(n+1), sizeof(double));
        ROW = (int *)calloc(n, sizeof(int));
```

```
        m = n+1;

        for(i=0; i<n; i++){
                for(j=0; j<n; j++){
                        B[i*m+j] = A[i*n+j];
                }
                B[i*m+n] = Y[i];
        }

        for(k=0; k<n; k++){
                ROW[k] = k;
        }


        for(k=0; k<n; k++){


                PIVOT = B[ROW[k]*m+k];
                ABS_PIVOT = abs(PIVOT);
                I_PIVOT = k;
                for(i=k; i<n; i++){
                        if( abs(B[ROW[i]*m+k]) > ABS_PIVOT){
                                I_PIVOT = i;
                                PIVOT = B[ROW[i]*m+k];
                                ABS_PIVOT = abs ( PIVOT );
                        }
                }


                HOLD = ROW[k];
                ROW[k] = ROW[I_PIVOT];
                ROW[I_PIVOT] = HOLD;


                if( ABS_PIVOT < 1.0E-10 ){
                        for(j=k+1; j<n+1; j++){
                                B[ROW[k]*m+j] = 0.0;
                        }
                        redundant=1;
                        if (fitcount == 2) redundant=2;
                }
                else{


                        for(j=k+1; j<n+1; j++){
                                B[ROW[k]*m+j] = B[ROW[k]*m+j] / B[ROW[k]*m+k];
                        }


                        for(i=0; i<n; i++){
                                if( i != k){
                                        for(j=k+1; j<n+1; j++){
                                                B[ROW[i]*m+j] = B[ROW[i]*m+j]
- B[ROW[i]*m+k] * B[ROW[k]*m+j];
                                        }
                                }
                        }
                }

        }


        for(i=0; i<n; i++){
                X[i] = B[ROW[i]*m+n];
        }
        free(B);
        free(ROW);
}
```

charge.c

```
int main(int argc, char *argv[]){
        FILE *input, *output, *result, *data;
        int i,
zeile,spalte,a=1,samplenumber,b,cellnumber,c=1,ca,results,startpt,cc,vsun,cell
,FEHLERCOUNT,threshold;
        char
in[10],*s,scanrate[100],load[100],inputfile[100],outputfile[100],startw[100],e
ndw[100],qd1[100],vd1[100],td1[100],qd2[100],vd2[100],td2[100],vf1[100],vf2[10
0];
        char * pEnd;
        char * pEnt;
        char * pEntt;
        char * pEnttt;
        char * pEntttt;
        double
SF,FG1,FG2,CH,Q,dbl,time,times,globaltime,r1,r2,t1,t2,Vox,I,Q2,Q1,dbscan,times
tep,period;
        double
dbnumber,CG,startwert,endwert,dtd1,dtd2,dvd1,dvd2,dqd1,dqd2,Dtime1,Dtime2,dvf1
,dvf2;
        double
startdvd,startdqd,startdvf,startdtd,diffdqd1,diffdqd2,diffdtd1,diffdtd2;
        double
C1,C2,C3,C4,C5,C0,C6,A1,A2,A3,A4,A5,A0,A6,vinitial,vend,ir,tr,vr,vmaxi,vmini;
        int n,ctr,vctr,rd;
        double A[2500];
        double C[50];
        double Y[50];
        double rms_err, avg_err, max_err;
        workcl=0;
        printf("\ninputfile:");
        scanf("%s",inputfile);
        printf("outputfile:");
        scanf("%s",outputfile);
        printf("number of measured cells:");
        scanf("%d",&cellnumber);
        printf("threshold:");
        scanf("%d",&threshold);
        printf("number of samples:");
        scanf("%d",&samplenumber);
        printf("scan rate (scans per sec):");
        scanf("%s",scanrate);
        printf("starting voltage:");
        scanf("%s",startw);
        printf("ending voltage:");
        scanf("%s",endw);
        startwert = strtod (startw, &pEnttt);
        endwert = strtod (endw, &pEntttt);
        startwert = startwert + 0.05;
        endwert = endwert - 0.02;
        dbscan = strtod (scanrate, &pEnt);
        CG = 5;
        timestep = 1/dbscan;
        rd=0;
        cell=0;
        period = samplenumber/dbscan;
        time=0;
        globaltime=0;
        b=1;
        i=1;
        cc=cellnumber;
        c=1;
        a=1;
        cnr=1;
        result = fopen(outputfile,"w");
        input = fopen(inputfile, "r");
        vmax=10;
        vmin=1;
        if (input == NULL) {
```

```
                                charge.c
                printf("\nfile not found\n");
                return(-1);
        }
        else
            {

                do{
                        zeile=samplenumber-1;
                        time=0;
                        b=1;
                        i=1;
                        c=1;
                        a=1,ca=0;
                        globaltime=0,times=0;
                        output = fopen("data.dat", "w");
                        data = fopen("dat.dat", "w");

                        do /* Zeilendurchgang */
                        {
                                zeile=zeile - 1;
                                spalte=7;

                                do /* Spaltendurchgang */
                                {
                                        spalte=spalte - 1;
                                        fscanf(input,"%s",in);
                                        dbl = strtod (in, &pEnd);

                                        if (spalte == 4){
                                                SF=dbl;
                                                FG1=1.1474*SF+0.4043;

FG2=0.00000411864*FG1*FG1+0.58*FG1+0.45;

CH=1E-11/FG2*(2.11513E-04*FG2+3.31773E-05);
                                                Vox=5-FG2;
                                                Q=CH*FG2;
                                                times=time;
                                        }
                                }
                                while (spalte);
                                if (SF >= startwert)
        {
                                                if (SF <= endwert)
                                        {

fprintf(data,"%E\t %E\t %E\t %E\n",time,Q,Vox,FG2);
                                                                b=b+1;
                                                        }
                                        }
                                time=time+timestep;
                                globaltime=globaltime+timestep;
                                ca=0;
                                if (time >= period) {
                                        time=0;
                                }
                        }
                        while (zeile);

                        SF=0;
                        dbl=0;
                        FG1=0;
                        FG2=0;
                        Vox=0,CH=0;
                        Q=0;
                        fclose(data);
                        data=fopen("dat.dat","r");
                        results=b/2;
                        i=1;
                        counter=0;

                                Seite 5
```

charge.c

```
startpt=0;
dtd1=0;
dqd1=0;
dvd1=0;
dvf1=0;
dtd2=0;
dqd2=0;
dvd2=0;
dvf2=0;
Dtime1=0;
Dtime2=0;
diffdqd1=0;
diffdqd1=0;
startdqd=0;
diffdtd1=0;
diffdtd1=0;
startdtd=0;

do {
        fscanf(data,"%s %s %s %s",td1,qd1,vd1,vf1);
        dtd1 = strtod (td1, &pEnd);
        dqd1 = strtod (qd1, &pEnd);
        dvd1 = strtod (vd1, &pEnd);
        dvf1 = strtod (vf1, &pEnd);
        Dtime1 = dtd1-dtd2;
        if (Dtime1 < 0) startpt = 0;
        if (Dtime1 > 0.003){
                if (startpt == 0) {
                        startpt=1;
                        startdvd=dvd1;
                        startdqd=dqd1;
                        startdvf=dvf1;
                        startdtd=dtd1;
                }
                diffdqd1=dqd1-startdqd;
                diffdtd1=(dtd1-startdtd)*0.899;
                if (diffdtd1 > 0) if (diffdqd1 > 0) if
(sqrt((dqd1-dqd2)*(dqd1-dqd2)) < 3E-1) {
                        VOXI[counter] = dvd1;
                        QI[counter]=diffdqd1;
                        TIMEI[counter]=diffdtd1;
                        fprintf(output,"%E\t %E\t %E\t
%E\t %d\n",dvd1,diffdqd1,dvf1,diffdtd1,counter); //3E-16
                        counter=counter+1;
                }
        }
        fscanf(data,"%s %s %s %s",td2,qd2,vd2, vf2);
        dtd2 = strtod (td2, &pEnd);
        dqd2 = strtod (qd2, &pEnd);
        dvd2 = strtod (vd2, &pEnd);
        dvf2 = strtod (vf2, &pEnd);
        Dtime2 = dtd2 - dtd1;
        if (Dtime2 < 0) startpt = 0;
        if (Dtime2 > 0.003)     {
                diffdqd2=dqd2-startdqd;
                diffdtd2=(dtd2-startdtd)*0.899;
                if (diffdtd2 > 0) if (diffdqd2 >0) if
(sqrt((dqd2-dqd1)*(dqd2-dqd1)) < 3E-1) {
                        VOXI[counter] = dvd2;
                        QI[counter]=diffdqd2;
                        TIMEI[counter]=diffdtd2;
                        fprintf(output,"%E\t %E\t %E\t
%E\t %d\n",dvd2,diffdqd2, dvf2,diffdtd2,counter);
                        counter=counter+1;
                }
        }
        results=results-1;
        if (results < 0) i=0;
}
while (i);
```

```
                                charge.c
                        dtd1=0;
                        dqd1=0;
                        dvf1=0;
                        dtd2=0;
                        dqd2=0;
                        dvd2=0;
                        dvf2=0;
                        Dtime1=0;
                        Dtime2=0;
                        diffdqd1=0;
                        diffdqd2=0;
                        startdqd=0;
                        diffdtd1=0;
                        diffdtd2=0;
                        startdtd=0;
                        counter=counter-1;

                        fclose(data);
                        fclose(output);

                        savei0=0;
                        savei1=0;
                        savei2=0;
                        redundant=0;

                        if (VOXI[1] < vmax) vmax = VOXI[1];
                        if (VOXI[counter-1] > vmin) vmin = VOXI[counter-1];

                        vinitial=2.3;
                        vend=3.5;
                        vdii=(vend-vinitial)/0.001;
                        kc = 0;
                        workcc=0;
                        f=0;

                        timediff=0;
                        fitcount=0;
                        n=7;
                        fit_pn(n, A, Y, C);
                        savei0=0;
                        savei1=0;
                        savei2=0;
                        check_pn(n, C, &rms_err, &avq_err, &max_err);
                        C0=C[0];
                        C1=C[1];
                        C2=C[2];
                        C3=C[3];
                        C4=C[4];
                        C5=C[5];
                        C6=C[6];

                        cnr=cnr+1;
                        vctr=1;
                        f=0;
                        vr=vinitial;
                        if (counter > threshold)                        {
                                        do
                                            {
                                                tr=TIMEI[f];

ir=6*C6*tr*tr*tr*tr*tr+5*C5*tr*tr*tr*tr+4*C4*tr*tr*tr+3*C3*tr*tr+2*C2*tr+C1;
                                                if (ir > 0)
                {
                                                                        if (workcc==0)
{

workcl=workcl+1;

workcc=1;
                                                                        }

                                Seite 7
```

```
                                        charge.c
                                                                IF[f]=ir;

VOXF[f]=VOXI[f];

fprintf(result,"%E\t%E\n",VOXF[f],IF[f]);
                                                                }
                                                        else vctr=0;
                                                        f=f+1;
                                                        if (f == counter) vctr=0;
                                                }
                                                while(vctr);
                                        }
                                cc=cc-1;
                        }

                        while(cc);
                }
                fclose(input);
                fclose(result);
                return 0;
}
```

discharge.c

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#undef  abs
#define abs(x) (((x)<0.0)?(-(x)):(x))
#undef  max
#define max(x,y) (((x)>(y))?((x)):(y))
#undef  min
#define min(x,y) (((x)<(y))?((x)):(y))

FILE *datas;
int counter, fitcount,
savei0,savei1,savei2,redundant,vdii,vdif,f,kc,workc1,workcc;
double
E[],timex1,timex2,voxx1,voxx2,qx1,qx2,TIMEI[2500],QI[2500],VOXI[2500],timediff
,IF[2500],VOXF[2500],FEHLER[2500],IFQ[2500];
double
MFEHLER,cnr,vdi,obercounter[2500],vmax,vmin,QST[2500],IFF[2500],VOXFF[2500];

static void simeq(int n, double A[], double Y[], double X[]);

static int data_set(double *y, double *x)
{
  char * pEnd;
  double vox1,q1,vfg1,time1,vox2,q2,vfg2,time2,time,q,vox;
  char svox[100], sq[100], svfg[100], stime[100];
  double xx;
  double yy;

if (fitcount == 0) {xx = TIMEI[savei0];
                                        yy = QI[savei0];
                                        savei0=savei0+1;
                                        if (savei0 >= counter) return 0;
                                        }
if (fitcount == 1) {xx = VOXF[savei1];
                                        yy = IF[savei1];
                                        savei1=savei1+1;
                                        if (savei1 >= vdii) return 0;
                                        }
if (fitcount == 2) {xx = VOXFF[savei2];
                                        yy = IFF[savei2];
                                        savei2=savei2+1;
                                        if (savei2 >= vdii) return 0;
                                        }

  *x = xx;
  *y = yy;


  return 1;
}


static void fit_pn(int n, double A[], double Y[], double C[])
{
  int i, j, k;
  double x, y, t;
  double pwr[30]; /* at least 2n */

  for(i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
    {
      A[i*n+j] = 0.0;
    }
    Y[i] = 0.0;
  }
  while(data_set(&y, &x))
  {
```

```
    pwr[0] = 1.0;
    for(i=1; i<2*n; i++) pwr[i] = pwr[i-1]*x;
    for(i=0; i<n; i++)
    {
      for(j=0; j<n; j++)
      {
        A[i*n+j] = A[i*n+j] + pwr[i]*pwr[j];
      }
      Y[i] = Y[i] + y*pwr[i];
    }
  }
  simeq(n, A, Y, C);
  for(i=0; i<n; i++); /*printf("C[%d]=%g \n", i, C[i]);*/
}



static void check_pn(int n, double C[],
                     double *rms_err, double *avq_err, double *max_err)
{
  double x, y, ya, diff;
  double sumsq = 0.0;
  double sum = 0.0;
  double maxe = 0.0;
  double xmin, xmax, ymin, ymax, xbad, ybad;
  int i, k, imax;

  k = 0;
  while(data_set(&y, &x))
  {
    if(k==0)
    {
      xmin=x;
      xmax=x;
      ymin=y;
      ymax=y;
      imax=0;
      xbad=x;
      ybad=y;
    }
    if(x>xmax) xmax=x;
    if(x<xmin) xmin=x;
    if(y>ymax) ymax=y;
    if(y<ymin) ymin=y;
    k++;
    ya = C[n-1]*x;
    for(i=n-2; i>0; i--)
    {
      ya = (C[i]+ya)*x;
    }
    ya = ya + C[0];
    diff = abs(y-ya);
    if(diff>maxe)
    {
      maxe=diff;
      imax=k;
      xbad=x;
      ybad=y;
    }
    sum = sum + diff;
    sumsq = sumsq + diff*diff;
  }
  *max_err = maxe;
  *avq_err = sum/(double)k;
  *rms_err = sqrt(sumsq/(double)k);
 }

static void simeq(int n, double A[], double Y[], double X[])
{
```

discharge.c

```
double *B;
int *ROW;
int HOLD , I_PIVOT;
double PIVOT;
double ABS_PIVOT;
int i,j,k,m;

B = (double *)calloc((n+1)*(n+1), sizeof(double));
ROW = (int *)calloc(n, sizeof(int));
m = n+1;


for(i=0; i<n; i++){
  for(j=0; j<n; j++){
    B[i*m+j] = A[i*n+j];
  }
  B[i*m+n] = Y[i];
}

for(k=0; k<n; k++){
  ROW[k] = k;
}


for(k=0; k<n; k++){


  PIVOT = B[ROW[k]*m+k];
  ABS_PIVOT = abs(PIVOT);
  I_PIVOT = k;
  for(i=k; i<n; i++){
    if( abs(B[ROW[i]*m+k]) > ABS_PIVOT){
      I_PIVOT = i;
      PIVOT = B[ROW[i]*m+k];
      ABS_PIVOT = abs ( PIVOT );
    }
  }

  HOLD = ROW[k];
  ROW[k] = ROW[I_PIVOT];
  ROW[I_PIVOT] = HOLD;


  if( ABS_PIVOT < 1.0E-10 ){
    for(j=k+1; j<n+1; j++){
      B[ROW[k]*m+j] = 0.0;
    }
            redundant=1; if (fitcount == 2) redundant=2;
                  }
  else{


    for(j=k+1; j<n+1; j++){
      B[ROW[k]*m+j] = B[ROW[k]*m+j] / B[ROW[k]*m+k];
    }


    for(i=0; i<n; i++){
      if( i != k){
        for(j=k+1; j<n+1; j++){
          B[ROW[i]*m+j] = B[ROW[i]*m+j] - B[ROW[i]*m+k] * B[ROW[k]*m+j];
        }
      }
    }
  }

}
```

discharge.c

```
    for(i=0; i<n; i++){
      X[i] = B[ROW[i]*m+n];
    }
    free(B);
    free(ROW);
}

int main(int argc, char *argv[])
{
                FILE *input, *output, *result, *data;
                int i,
zeile,spalte,a=1,samplenumber,b,cellnumber,c=1,ca,results,startpt,cc,vsun,cell
,FEHLERCOUNT,threshold;
                char
in[10],*s,scanrate[100],load[100],inputfile[100],outputfile[100],startw[100],e
ndw[100],qd1[100],vd1[100],td1[100],qd2[100],vd2[100],td2[100],vf1[100],vf2[10
0];
                char * pEnd;
                char * pEnt;
                char * pEntt;
                char * pEnttt;
                char * pEntttt;
                double
SF,FG1,FG2,CH,Q,db1,time,times,globaltime,r1,r2,t1,t2,Vox,I,Q2,Q1,dbscan,times
tep,period,SFSAVE;
                double
dbnumber,CG,startwert,endwert,dtd1,dtd2,dvd1,dvd2,dqd1,dqd2,Dtime1,Dtime2,dvf1
,dvf2;
                double
startdvd,startdqd,startdvf,startdtd,diffdqd1,diffdqd2,diffdtd1,diffdtd2;
                double C1,C2,C3,C4,C5,C0,C6,C7,
A1,A2,A3,A4,A5,A0,A6,vinitial,vend,ir,tr,vr,vmaxi,vmini;
                int n,ctr,vctr,rd;
                double A[2500];
                double C[50];
                double Y[50];
                double rms_err, avq_err, max_err;
                workcl=0;
                printf("number of measured cells:");
                scanf("%d",&cellnumber);
                printf("number of samples:");
                scanf("%d",&samplenumber);
                printf("scan rate (scans per sec):");
                scanf("%s",scanrate);
                printf("starting voltage:");
                scanf("%s",startw);
                printf("ending voltage:");
                scanf("%s",endw);
                startwert = strtod (startw, &pEnttt);
                endwert = strtod (endw, &pEntttt);
                startwert = startwert - 0.05;
                endwert = endwert + 0.05;
                dbscan = strtod (scanrate, &pEnt);
                CG = 5; SFSAVE=10000;
                timestep = 1/dbscan; rd=0; cell=0;
                period = samplenumber/dbscan;
                time=0;globaltime=0;b=1;i=1;
                cc=cellnumber; c=1;a=1;cnr=1;
                result = fopen("results.txt","w");
                input = fopen("data.txt", "r");
                vmax=10; vmin=1;
                if (input == NULL) {printf("\nfile not found\n");return(-1);}
                else
                        {

do{
zeile=samplenumber;time=0;b=1;i=1;c=1;a=1,ca=0;globaltime=0;SFSAVE=10000;
                output = fopen("data.dat", "w");
                data = fopen("dat.dat", "w");
```

```
                                  discharge.c
                                do /* Zeilendurchgang */
                                {
                                        zeile=zeile - 1;
                                        spalte=7;

                                                do /* Spaltendurchgang */
                                                {
                                                spalte=spalte - 1;
                                                fscanf(input,"%s",in);
                                                dbl = strtod (in, &pEnd);

                                                if (spalte == 4)
                                                        {
                                                        SF=dbl;
                                                        FG1=1.1474*SF+0.4043;

FG2=1.2647E-01*FG1*FG1*FG1-4.7997E-01*FG1*FG1+1.2841E+00*FG1+2.2760E+00;
                                                        Vox=FG2; // Oxyd
Voltage pmos (CG=0)

CH=1E-11/FG2*(2.11513E-04*FG2+3.31773E-05);
                                                        Q=CH*FG2;

                                                        }
                                                }
                                                while (spalte);
                                if (SF <= startwert)
                                        {if (SF >= endwert)
                                                {if (SF <= SFSAVE)

                                                        {fprintf(data,"%E\t
%E\t %E\t %E\n",time,Q,FG2,SF);b=b+1; SFSAVE=SF;}
                                                }
                                        }
                                time=time+timestep;
globaltime=globaltime+timestep; ca=0;
                                if (time >= period) {time=0;}
                                }
                                while (zeile);

                SF=0; dbl=0;FG1=0;FG2=0;Vox=0,CH=0;Q=0;
                fclose(data);
                data=fopen("dat.dat","r");
                results=b/2;
                i=1;
                counter=0;
                startpt=0;

dtd1=0;dqd1=0;dvd1=0;dvf1=0;dtd2=0;dqd2=0;dvd2=0;dvf2=0;Dtime1=0;Dtime2=0;

diffdqd1=0;diffdqd1=0;startdqd=0;diffdtd1=0;diffdtd1=0;startdtd=0;

                        do
                        {
                                        fscanf(data,"%s %s %s
%s",td1,qd1,vd1,vf1);
                                        dtd1 = strtod (td1, &pEnd);
                                        dqd1 = strtod (qd1, &pEnd);
                                        dvd1 = strtod (vd1, &pEnd);
                                        dvf1 = strtod (vf1, &pEnd);
                                        Dtime1 = dtd1-dtd2;
                                        if (Dtime1 < 0) startpt = 0;
                                        if (Dtime1 > 0.003)
                                                {
                                                if (startpt == 0) {startpt=1;
startdvd=dvd1; startdqd=dqd1; startdvf=dvf1; startdtd=dtd1;}
                                                diffdqd1=-1*(dqd1-startdqd);

diffdtd1=(dtd1-startdtd)*0.899;
                                                if (diffdtd1 > 0) if (diffdqd1
```

```
                                        discharge.c
> 0) if (sqrt((dqd1-dqd2)*(dqd1-dqd2)) < 3E-16)
                                                        {
                                                        VOXI[counter] = dvd1;
QI[counter]=diffdqd1; TIMEI[counter]=diffdtd1;
                                                        fprintf(output,"%E\t
%E\t %E\t %E\t %d\n",dvd1,diffdqd1,dvf1,diffdtd1,counter);
                                                        counter=counter+1;
                                                        }
                                                }
                                        fscanf(data,"%s %s %s %s",td2,qd2,vd2,
vf2);
                                        dtd2 = strtod (td2, &pEnd);
                                        dqd2 = strtod (qd2, &pEnd);
                                        dvd2 = strtod (vd2, &pEnd);
                                        dvf2 = strtod (vf2, &pEnd);
                                        Dtime2 = dtd2 - dtd1;
                                        if (Dtime2 < 0) startpt = 0;
                                        if (Dtime2 > 0.003)
                                                {
                                                diffdqd2=-1*(dqd2-startdqd);

diffdtd2=(dtd2-startdtd)*0.899;
                                                if (diffdtd2 > 0) if (diffdqd2
> 0) if (sqrt((dqd2-dqd1)*(dqd2-dqd1)) < 3E-16)
                                                        {
                                                        VOXI[counter] = dvd2;
QI[counter]=diffdqd2; TIMEI[counter]=diffdtd2;
                                                        fprintf(output,"%E\t
%E\t %E\t %E\t %d\n",dvd2,diffdqd2, dvf2,diffdtd2,counter);
                                                        counter=counter+1;
                                                        }
                                                }
                                        results=results-1; if (results < 0)
i=0;
                        }
                        while (i);

dtd1=0;dqd1=0;dvf1=0;dtd2=0;dqd2=0;dvd2=0;dvf2=0;Dtime1=0;Dtime2=0;

diffdqd1=0;diffdqd2=0;startdqd=0;diffdtd1=0;diffdtd2=0;startdtd=0;
counter=counter-1;

fclose(data);
fclose(output);

savei0=0;
savei1=0;
savei2=0;
redundant=0;
if (VOXI[1] < vmax) vmax = VOXI[1];
if (VOXI[counter-1] > vmin) vmin = VOXI[counter-1];

vinitial=2.3; vend=3.5; vdii=(vend-vinitial)/0.001; kc = 0; workcc=0;
f=0;

        timediff=0;
        fitcount=0; n=8;
        fit_pn(n, A, Y, C);
        savei0=0; savei1=0; savei2=0;
        check_pn(n, C, &rms_err, &avg_err, &max_err);
        C0=C[0];C1=C[1];C2=C[2];C3=C[3];C4=C[4];C5=C[5];C6=C[6];C7=C[7];

                        cnr=cnr+1;
                        vctr=1; f=0; vr=vinitial;
                        if (counter > 50)
                        {
                                do
                                {
                                tr=TIMEI[f];
```

```
                                        discharge.c
ir=7*C7*tr*tr*tr*tr*tr*tr+6*C6*tr*tr*tr*tr*tr+5*C5*tr*tr*tr*tr+4*C4*tr*tr*tr+3
*C3*tr*tr+2*C2*tr+C1;
                                    if (ir > 0)
                                            {
                                            if (workcc==0) {workcl=workcl+1;
workcc=1;}
                                            IF[f]=ir; VOXF[f]=VOXI[f];
fprintf(result,"%E\t%E\n",VOXF[f],IF[f]);
                                            }
                                    else vctr=0;
                                    f=f+1;
                                    if (f == counter) vctr=0;
                                    }
                                    while(vctr);
                            }

cc=cc-1;
}

while(cc);
}
fclose(input);
fclose(result);
return 0;
}
```

3vpcharge.c

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>


#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#undef  abs
#define abs(x) (((x)<0.0)?(-(x)):(x))
#undef  max
#define max(x,y) (((x)>(y))?((x)):(y))
#undef  min
#define min(x,y) (((x)<(y))?((x)):(y))

FILE *datas;
int counter, fitcount,
savei0,savei1,savei2,redundant,vdii,vdif,f,kc,workc1,workcc;
double
E[],timex1,timex2,voxx1,voxx2,qx1,qx2,TIMEI[2500],QI[2500],VOXI[2500],timediff
,IF[2500],VOXF[2500],FEHLER[2500],IFQ[2500];
double
MFEHLER,cnr,vdi,obercounter[2500],vmax,vmin,QST[2500],IFF[2500],VOXFF[2500];


static void simeq(int n, double A[], double Y[], double X[]);

static int data_set(double *y, double *x)
{
  char * pEnd;
  double vox1,q1,vfg1,time1,vox2,q2,vfg2,time2,time,q,vox;
  char svox[100], sq[100], svfg[100], stime[100];
  double xx;
  double yy;

if (fitcount == 0) {xx = TIMEI[savei0];
                                        yy = QI[savei0];
                                        savei0=savei0+1;
                                        if (savei0 >= counter) return 0;
                                        }
if (fitcount == 1) {xx = VOXF[savei1];
                                        yy = IF[savei1];
                                        savei1=savei1+1;
                                        if (savei1 >= vdii) return 0;
                                        }
if (fitcount == 2) {xx = VOXFF[savei2];
                                        yy = IFF[savei2];
                                        savei2=savei2+1;
                                        if (savei2 >= vdii) return 0;
                                        }

  *x = xx;
  *y = yy;
  return 1;
}


static void fit_pn(int n, double A[], double Y[], double C[])
{
  int i, j, k;
  double x, y, t;
  double pwr[30]; /* at least 2n */

  for(i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
    {
      A[i*n+j] = 0.0;
```

```
    }
    Y[i] = 0.0;
  }
  while(data_set(&y, &x))
  {
    pwr[0] = 1.0;
    for(i=1; i<2*n; i++) pwr[i] = pwr[i-1]*x;
    for(i=0; i<n; i++)
    {
      for(j=0; j<n; j++)
      {
        A[i*n+j] = A[i*n+j] + pwr[i]*pwr[j];
      }
      Y[i] = Y[i] + y*pwr[i];
    }
  }
  simeq(n, A, Y, C);
  for(i=0; i<n; i++); /*printf("C[%d]=%g \n", i, C[i]);*/
}

static void check_pn(int n, double C[],
                     double *rms_err, double *avg_err, double *max_err)
{
  double x, y, ya, diff;
  double sumsq = 0.0;
  double sum = 0.0;
  double maxe = 0.0;
  double xmin, xmax, ymin, ymax, xbad, ybad;
  int i, k, imax;

  k = 0;
  while(data_set(&y, &x))
  {
    if(k==0)
    {
      xmin=x;
      xmax=x;
      ymin=y;
      ymax=y;
      imax=0;
      xbad=x;
      ybad=y;
    }
    if(x>xmax) xmax=x;
    if(x<xmin) xmin=x;
    if(y>ymax) ymax=y;
    if(y<ymin) ymin=y;
    k++;
    ya = C[n-1]*x;
    for(i=n-2; i>0; i--)
    {
      ya = (C[i]+ya)*x;
    }
    ya = ya + C[0];
    diff = abs(y-ya);
    if(diff>maxe)
    {
      maxe=diff;
      imax=k;
      xbad=x;
      ybad=y;
    }
    sum = sum + diff;
    sumsq = sumsq + diff*diff;
  }

  *max_err = maxe;
  *avg_err = sum/(double)k;
  *rms_err = sqrt(sumsq/(double)k);
}
```

3vpcharge.c

```
static void simeq(int n, double A[], double Y[], double X[])
{


    double *B;
    int *ROW;
    int HOLD , I_PIVOT;
    double PIVOT;
    double ABS_PIVOT;
    int i,j,k,m;

    B = (double *)calloc((n+1)*(n+1), sizeof(double));
    ROW = (int *)calloc(n, sizeof(int));
    m = n+1;


    for(i=0; i<n; i++){
      for(j=0; j<n; j++){
        B[i*m+j] = A[i*n+j];
      }
      B[i*m+n] = Y[i];
    }

    for(k=0; k<n; k++){
      ROW[k] = k;
    }


    for(k=0; k<n; k++){


      PIVOT = B[ROW[k]*m+k];
      ABS_PIVOT = abs(PIVOT);
      I_PIVOT = k;
      for(i=k; i<n; i++){
        if( abs(B[ROW[i]*m+k]) > ABS_PIVOT){
          I_PIVOT = i;
          PIVOT = B[ROW[i]*m+k];
          ABS_PIVOT = abs ( PIVOT );
        }
      }


      HOLD = ROW[k];
      ROW[k] = ROW[I_PIVOT];
      ROW[I_PIVOT] = HOLD;


      if( ABS_PIVOT < 1.0E-10 ){
        for(j=k+1; j<n+1; j++){
          B[ROW[k]*m+j] = 0.0;
        }
                redundant=1; if (fitcount == 2) redundant=2;
                      }
      else{


        for(j=k+1; j<n+1; j++){
          B[ROW[k]*m+j] = B[ROW[k]*m+j] / B[ROW[k]*m+k];
        }


        for(i=0; i<n; i++){
          if( i != k){
            for(j=k+1; j<n+1; j++){
              B[ROW[i]*m+j] = B[ROW[i]*m+j] - B[ROW[i]*m+k] * B[ROW[k]*m+j];
            }
          }
        }
```

```
      }

    }


    for(i=0; i<n; i++){
      X[i] = B[ROW[i]*m+n];
    }
    free(B);
    free(ROW);
}



int main(int argc, char *argv[])
{
                FILE *input, *output, *result, *data;
                int i,
zeile,spalte,a=1,samplenumber,b,cellnumber,c=1,ca,results,startpt,cc,vsun,cell
,FEHLERCOUNT,threshold;
                char
in[10],*s,scanrate[100],load[100],inputfile[100],outputfile[100],startw[100],e
ndw[100],qd1[100],vd1[100],td1[100],qd2[100],vd2[100],td2[100],vf1[100],vf2[10
0];
                char * pEnd;
                char * pEnt;
                char * pEntt;
                char * pEnttt;
                char * pEntttt;
                double
SF,FG1,FG2,CH,Q,dbl,time,times,globaltime,r1,r2,t1,t2,Vox,I,Q2,Q1,dbscan,times
tep,period;
                double
dbnumber,CG,startwert,endwert,dtd1,dtd2,dvd1,dvd2,dqd1,dqd2,Dtime1,Dtime2,dvf1
,dvf2;
                double
startdvd,startdqd,startdvf,startdtd,diffdqd1,diffdqd2,diffdtd1,diffdtd2;
                double
C1,C2,C3,C4,C5,C0,C6,A1,A2,A3,A4,A5,A0,A6,vinitial,vend,ir,tr,vr,vmaxi,vmini;
                int n,ctr,vctr,rd;
                double A[2500];
                double C[50];
                double Y[50];
                double rms_err, avq_err, max_err;
                workcl=0;
                printf("\ninputfile:");
                scanf("%s",inputfile);
                printf("outputfile:");
                scanf("%s",outputfile);
                printf("number of measured cells:");
                scanf("%d",&cellnumber);
                printf("threshold:");
                scanf("%d",&threshold);
                printf("number of samples:");
                scanf("%d",&samplenumber);
                printf("scan rate (scans per sec):");
                scanf("%s",scanrate);
                printf("starting voltage:");
                scanf("%s",startw);
                printf("ending voltage:");
                scanf("%s",endw);
                startwert = strtod (startw, &pEnttt);
                endwert = strtod (endw, &pEntttt);
                startwert = startwert + 0.05;
                endwert = endwert - 0.02;
                dbscan = strtod (scanrate, &pEnt);
                CG = 5;
                timestep = 1/dbscan; rd=0; cell=0;
                period = samplenumber/dbscan;
                time=0;globaltime=0;b=1;i=1;
```

```
                                3vpcharge.c
                cc=cellnumber; c=1;a=1;cnr=1;
                result = fopen(outputfile,"w");
                input = fopen(inputfile, "r");
                vmax=10; vmin=1;
                if (input == NULL) {printf("\nfile not found\n");return(-1);}
                else
                        {

do{
zeile=samplenumber-1;time=0;b=1;i=1;c=1;a=1,ca=0;globaltime=0,times=0;
                output = fopen("data.dat", "w");
                data = fopen("dat.dat", "w");

                                do /* Zeilendurchgang */
                                {
                                        zeile=zeile - 1;
                                        spalte=7;

                                                do /* Spaltendurchgang */
                                                {
                                                spalte=spalte - 1;
                                                fscanf(input,"%s",in);
                                                dbl = strtod (in, &pEnd);

                                                if (spalte == 4){
                                                        SF=dbl;
                                                        .1474*SF+0.4043;
                                                        FG1=1.178*SF+0.3224;

FG2=0.00000411864*FG1*FG1+0.58*FG1+0.64234;

CH=1E-11/FG2*(1.34476E-04*FG2+5.94897E-05);
                                                        Vox=5-FG2;
                                                        Q=CH*FG2;
                                                        times=time;}
                                                }
                                                while (spalte);
                                if (SF >= startwert)
                                        {if (SF <= endwert)
                                                {fprintf(data,"%E\t
%E\t %E\t %E\n",time,Q,Vox,FG2);b=b+1;}
                                        }
                                time=time+timestep;
globaltime=globaltime+timestep; ca=0;
                                if (time >= period) {time=0;}
                                }
                                while (zeile);

                SF=0; dbl=0;FG1=0;FG2=0;Vox=0,CH=0;Q=0;
                fclose(data);
                data=fopen("dat.dat","r");
                results=b/2;
                i=1;
                counter=0;
                startpt=0;

dtd1=0;dqd1=0;dvd1=0;dvf1=0;dtd2=0;dqd2=0;dvd2=0;dvf2=0;Dtime1=0;Dtime2=0;

diffdqd1=0;diffdqd1=0;startdqd=0;diffdtd1=0;diffdtd1=0;startdtd=0;

                        do {
                                        fscanf(data,"%s %s %s
%s",td1,qd1,vd1,vf1);
                                        dtd1 = strtod (td1, &pEnd);
                                        dqd1 = strtod (qd1, &pEnd);
                                        dvd1 = strtod (vd1, &pEnd);
                                        dvf1 = strtod (vf1, &pEnd);
                                        Dtime1 = dtd1-dtd2;
                                        if (Dtime1 < 0) startpt = 0;
                                        if (Dtime1 > 0.003){
```

```
                                                        if (startpt == 0)
{startpt=1; startdvd=dvd1; startdqd=dqd1; startdvf=dvf1; startdtd=dtd1;}

diffdqd1=dqd1-startdqd;

diffdtd1=(dtd1-startdtd)*0.899;
                                                        if (diffdtd1 > 0) if
(diffdqd1 > 0) if (sqrt((dqd1-dqd2)*(dqd1-dqd2)) < 3E-1) { VOXI[counter] =
dvd1; QI[counter]=diffdqd1; TIMEI[counter]=diffdtd1;
                                                        fprintf(output,"%E\t
%E\t %E\t %E\t %d\n",dvd1,diffdqd1,dvf1,diffdtd1,counter);
                                                        counter=counter+1;}
                                                        }
                                                fscanf(data,"%s %s %s %s",td2,qd2,vd2,
vf2);
                                                dtd2 = strtod (td2, &pEnd);
                                                dqd2 = strtod (qd2, &pEnd);
                                                dvd2 = strtod (vd2, &pEnd);
                                                dvf2 = strtod (vf2, &pEnd);
                                                Dtime2 = dtd2 - dtd1;
                                                if (Dtime2 < 0) startpt = 0;
                                                if (Dtime2 > 0.003)     {

diffdqd2=dqd2-startdqd;

diffdtd2=(dtd2-startdtd)*0.899;
                                                        if (diffdtd2 > 0) if
(diffdqd2 >0) if (sqrt((dqd2-dqd1)*(dqd2-dqd1)) < 3E-1) {VOXI[counter] = dvd2;
QI[counter]=diffdqd2; TIMEI[counter]=diffdtd2;
                                                        fprintf(output,"%E\t
%E\t %E\t %E\t %d\n",dvd2,diffdqd2, dvf2,diffdtd2,counter);
                                                        counter=counter+1;}
                                                        }
                                                results=results-1; if (results < 0)
i=0;}
                                while (i);

dtd1=0;dqd1=0;dvf1=0;dtd2=0;dqd2=0;dvd2=0;dvf2=0;Dtime1=0;Dtime2=0;

diffdqd1=0;diffdqd2=0;startdqd=0;diffdtd1=0;diffdtd2=0;startdtd=0;
counter=counter-1;

fclose(data);
fclose(output);

savei0=0;
savei1=0;
savei2=0;
redundant=0;

if (VOXI[1] < vmax) vmax = VOXI[1];
if (VOXI[counter-1] > vmin) vmin = VOXI[counter-1];

vinitial=2.3; vend=3.5; vdii=(vend-vinitial)/0.001; kc = 0; workcc=0;
f=0;

        timediff=0;
        fitcount=0; n=7;
        fit_pn(n, A, Y, C);
        savei0=0; savei1=0; savei2=0;
        check_pn(n, C, &rms_err, &avq_err, &max_err);
        //printf("rms_err=%g, avg_err=%g, max_err=%g \n\n",rms_err, avg_err,
max_err);
        C0=C[0];C1=C[1];C2=C[2];C3=C[3];C4=C[4];C5=C[5];C6=C[6];

                        cnr=cnr+1;
                        vctr=1; f=0; vr=vinitial;
                        if (counter > threshold)
                        {
                                do
```

```
                                    3vpcharge.c
                                   {
                                   tr=TIMEI[f];

ir=6*C6*tr*tr*tr*tr*tr+5*C5*tr*tr*tr*tr+4*C4*tr*tr*tr+3*C3*tr*tr+2*C2*tr+C1;
                                   if (ir > 0)
                                          {
                                          if (workcc==0) {workcl=workcl+1;
workcc=1;}
                                          IF[f]=ir; VOXF[f]=VOXI[f];
fprintf(result,"%E\t%E\n",VOXF[f],IF[f]);
                                          }
                                   else vctr=0;
                                   f=f+1;
                                   if (f == counter) vctr=0;
                                   }
                                   while(vctr);
                            }
cc=cc-1;
}

while(cc);
}
fclose(input);
fclose(result);
return 0;
}
```

3vpdischarge.c

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#undef  abs
#define abs(x) (((x)<0.0)?(-(x)):(x))
#undef  max
#define max(x,y) (((x)>(y))?((x)):(y))
#undef  min
#define min(x,y) (((x)<(y))?((x)):(y))

FILE *datas;
int counter, fitcount,
savei0,savei1,savei2,redundant,vdii,vdif,f,kc,workc1,workcc;
double
E[],timex1,timex2,voxx1,voxx2,qx1,qx2,TIMEI[2500],QI[2500],VOXI[2500],timediff
,IF[2500],VOXF[2500],FEHLER[2500],IFQ[2500];
double
MFEHLER,cnr,vdi,obercounter[2500],vmax,vmin,QST[2500],IFF[2500],VOXFF[2500];

static void simeq(int n, double A[], double Y[], double X[]);

static int data_set(double *y, double *x)
{
  char * pEnd;
  double vox1,q1,vfg1,time1,vox2,q2,vfg2,time2,time,q,vox;
  char svox[100], sq[100], svfg[100], stime[100];



  double xx;
  double yy;



if (fitcount == 0) {xx = TIMEI[savei0];
                                        yy = QI[savei0];
                                        savei0=savei0+1;
                                        if (savei0 >= counter) return 0;
                                        }
if (fitcount == 1) {xx = VOXF[savei1];
                                        yy = IF[savei1];
                                        savei1=savei1+1;
                                        if (savei1 >= vdii) return 0;
                                        }
if (fitcount == 2) {xx = VOXFF[savei2];
                                        yy = IFF[savei2];
                                        savei2=savei2+1;
                                        if (savei2 >= vdii) return 0;
                                        }

  *x = xx;
  *y = yy;



  return 1;
}


static void fit_pn(int n, double A[], double Y[], double C[])
{
  int i, j, k;
  double x, y, t;
  double pwr[30]; /* at least 2n */

  for(i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
    {
```

```
      A[i*n+j] = 0.0;
    }
    Y[i] = 0.0;
  }
  while(data_set(&y, &x))
  {
    pwr[0] = 1.0;
    for(i=1; i<2*n; i++) pwr[i] = pwr[i-1]*x;
    for(i=0; i<n; i++)
    {
      for(j=0; j<n; j++)
      {
        A[i*n+j] = A[i*n+j] + pwr[i]*pwr[j];
      }
      Y[i] = Y[i] + y*pwr[i];
    }
  }
  simeq(n, A, Y, C);
  for(i=0; i<n; i++); /*printf("C[%d]=%g \n", i, C[i]);*/
}

static void check_pn(int n, double C[],
                     double *rms_err, double *avg_err, double *max_err)
{
  double x, y, ya, diff;
  double sumsq = 0.0;
  double sum = 0.0;
  double maxe = 0.0;
  double xmin, xmax, ymin, ymax, xbad, ybad;
  int i, k, imax;

  k = 0;
  while(data_set(&y, &x))
  {
    if(k==0)
    {
      xmin=x;
      xmax=x;
      ymin=y;
      ymax=y;
      imax=0;
      xbad=x;
      ybad=y;
    }
    if(x>xmax) xmax=x;
    if(x<xmin) xmin=x;
    if(y>ymax) ymax=y;
    if(y<ymin) ymin=y;
    k++;
    ya = C[n-1]*x;
    for(i=n-2; i>0; i--)
    {
      ya = (C[i]+ya)*x;
    }
    ya = ya + C[0];
    diff = abs(y-ya);
    if(diff>maxe)
    {
      maxe=diff;
      imax=k;
      xbad=x;
      ybad=y;
    }
    sum = sum + diff;
    sumsq = sumsq + diff*diff;
  }
  /*printf("check_pn k=%d, xmin=%g, xmax=%g, ymin=%g, ymax=%g \n",
          k, xmin, xmax, ymin, ymax);*/
  *max_err = maxe;
  *avg_err = sum/(double)k;
```

```
  *rms_err = sqrt(sumsq/(double)k);
  /*printf("max=%g at %d, xbad=%g, ybad=%g\n", maxe, imax, xbad, ybad);*/
}


static void simeq(int n, double A[], double Y[], double X[])
{


    double *B;
    int *ROW;
    int HOLD , I_PIVOT;
    double PIVOT;
    double ABS_PIVOT;
    int i,j,k,m;

    B = (double *)calloc((n+1)*(n+1), sizeof(double));
    ROW = (int *)calloc(n, sizeof(int));
    m = n+1;


    for(i=0; i<n; i++){
      for(j=0; j<n; j++){
        B[i*m+j] = A[i*n+j];
      }
      B[i*m+n] = Y[i];
    }

    for(k=0; k<n; k++){
      ROW[k] = k;
    }


    for(k=0; k<n; k++){


      PIVOT = B[ROW[k]*m+k];
      ABS_PIVOT = abs(PIVOT);
      I_PIVOT = k;
      for(i=k; i<n; i++){
        if( abs(B[ROW[i]*m+k]) > ABS_PIVOT){
          I_PIVOT = i;
          PIVOT = B[ROW[i]*m+k];
          ABS_PIVOT = abs ( PIVOT );
        }
      }


      HOLD = ROW[k];
      ROW[k] = ROW[I_PIVOT];
      ROW[I_PIVOT] = HOLD;


      if( ABS_PIVOT < 1.0E-10 ){
        for(j=k+1; j<n+1; j++){
          B[ROW[k]*m+j] = 0.0;
        }
        //printf("redundant row (singular) %d \n", ROW[k]);
                redundant=1; if (fitcount == 2) redundant=2;
                      }
      else{


        for(j=k+1; j<n+1; j++){
          B[ROW[k]*m+j] = B[ROW[k]*m+j] / B[ROW[k]*m+k];
        }


        for(i=0; i<n; i++){
```

```
          if( i != k){
            for(j=k+1; j<n+1; j++){
              B[ROW[i]*m+j] = B[ROW[i]*m+j] - B[ROW[i]*m+k] * B[ROW[k]*m+j];
            }
          }
        }
      }

    }

    for(i=0; i<n; i++){
      X[i] = B[ROW[i]*m+n];
    }
    free(B);
    free(ROW);
}


int main(int argc, char *argv[])
{
                FILE *input, *output, *result, *data;
                int i,
zeile,spalte,a=1,samplenumber,b,cellnumber,c=1,ca,results,startpt,cc,vsun,cell
,FEHLERCOUNT,threshold;
                char
in[10],*s,scanrate[100],load[100],inputfile[100],outputfile[100],startw[100],e
ndw[100],qd1[100],vd1[100],td1[100],qd2[100],vd2[100],td2[100],vf1[100],vf2[10
0];
                char * pEnd;
                char * pEnt;
                char * pEntt;
                char * pEnttt;
                char * pEntttt;
                double
SF,FG1,FG2,CH,Q,dbl,time,times,globaltime,r1,r2,t1,t2,Vox,I,Q2,Q1,dbscan,times
tep,period,SFSAVE;
                double
dbnumber,CG,startwert,endwert,dtd1,dtd2,dvd1,dvd2,dqd1,dqd2,Dtime1,Dtime2,dvf1
,dvf2;
                double
startdvd,startdqd,startdvf,startdtd,diffdqd1,diffdqd2,diffdtd1,diffdtd2;
                double
C1,C2,C3,C4,C5,C0,C6,A1,A2,A3,A4,A5,A0,A6,vinitial,vend,ir,tr,vr,vmaxi,vmini;
                int n,ctr,vctr,rd;
                double A[2500];
                double C[50];
                double Y[50];
                double rms_err, avg_err, max_err;
                workcl=0;

                printf("number of measured cells:");
                scanf("%d",&cellnumber);
                printf("number of samples:");
                scanf("%d",&samplenumber);
                printf("scan rate (scans per sec):");
                scanf("%s",scanrate);
                printf("starting voltage:");
                scanf("%s",startw);
                printf("ending voltage:");
                scanf("%s",endw);
                startwert = strtod (startw, &pEnttt);
                endwert = strtod (endw, &pEntttt);
                startwert = startwert - 0.05;
                endwert = endwert + 0.05;
                dbscan = strtod (scanrate, &pEnt);
                CG = 5; SFSAVE=10000;
                timestep = 1/dbscan; rd=0; cell=0;
                period = samplenumber/dbscan;
                time=0;globaltime=0;b=1;i=1;
```

```
                                3vpdischarge.c
                cc=cellnumber; c=1;a=1;cnr=1;
                result = fopen("results.txt","w");
                input = fopen("data.txt", "r");
                vmax=10; vmin=1;
                if (input == NULL) {printf("\nfile not found\n");return(-1);}
                else
                        {

do{
zeile=samplenumber;time=0;b=1;i=1;c=1;a=1,ca=0;globaltime=0;SFSAVE=10000;
                output = fopen("data.dat", "w");
                data = fopen("dat.dat", "w");

                                do /* Zeilendurchgang */
                                {
                                        zeile=zeile - 1;
                                        spalte=7;

                                                do /* Spaltendurchgang */
                                                {
                                                spalte=spalte - 1;
                                                fscanf(input,"%s",in);
                                                dbl = strtod (in, &pEnd);

                                                if (spalte == 4)
                                                        {
                                                        SF=dbl;
                                                        FG1=1.178*SF+0.3224;

FG2=6.8405E-02*FG1*FG1*FG1-3.88648E-01*FG1*FG1+1.34517*FG1+2.14953;
                                                        Vox=FG2;

CH=1E-11/FG2*(1.34476E-4*FG2+5.94897E-05);
                                                        Q=CH*FG2;

                                                        }
                                                }
                                                while (spalte);
                                if (SF <= startwert)
                                        {if (SF >= endwert)
                                                {if (SF <= SFSAVE)

                                                        {fprintf(data,"%E\t
%E\t %E\t %E\n",time,Q,FG2,SF);b=b+1; SFSAVE=SF;}
                                                }
                                        }
                                time=time+timestep;
globaltime=globaltime+timestep; ca=0;
                                if (time >= period) {time=0;}
                                }
                                while (zeile);

                SF=0; dbl=0;FG1=0;FG2=0;Vox=0,CH=0;Q=0;
                fclose(data);
                data=fopen("dat.dat","r");
                results=b/2;
                i=1;
                counter=0;
                startpt=0;

dtd1=0;dqd1=0;dvd1=0;dvf1=0;dtd2=0;dqd2=0;dvd2=0;dvf2=0;Dtime1=0;Dtime2=0;

diffdqd1=0;diffdqd1=0;startdqd=0;diffdtd1=0;diffdtd1=0;startdtd=0;

                        do
                        {
                                        fscanf(data,"%s %s %s
%s",td1,qd1,vd1,vf1);
                                        dtd1 = strtod (td1, &pEnd);
                                        dqd1 = strtod (qd1, &pEnd);
```

```
                                        3vpdischarge.c
                                                dvd1 = strtod (vd1, &pEnd);
                                                dvf1 = strtod (vf1, &pEnd);
                                                Dtime1 = dtd1-dtd2;
                                                if (Dtime1 < 0) startpt = 0;
                                                if (Dtime1 > 0.003)
                                                        {
                                                        if (startpt == 0) {startpt=1;
startdvd=dvd1; startdqd=dqd1; startdvf=dvf1; startdtd=dtd1;}
                                                        diffdqd1=-1*(dqd1-startdqd);

diffdtd1=(dtd1-startdtd)*0.899;
                                                        if (diffdtd1 > 0) if (diffdqd1
> 0) if (sqrt((dqd1-dqd2)*(dqd1-dqd2)) < 3E-16)
                                                                {
                                                                VOXI[counter] = dvd1;
QI[counter]=diffdqd1; TIMEI[counter]=diffdtd1;
                                                                fprintf(output,"%E\t
%E\t %E\t %E\t %d\n",dvd1,diffdqd1,dvf1,diffdtd1,counter);
                                                                counter=counter+1;
                                                                }
                                                        }
                                                fscanf(data,"%s %s %s %s",td2,qd2,vd2,
vf2);
                                                dtd2 = strtod (td2, &pEnd);
                                                dqd2 = strtod (qd2, &pEnd);
                                                dvd2 = strtod (vd2, &pEnd);
                                                dvf2 = strtod (vf2, &pEnd);
                                                Dtime2 = dtd2 - dtd1;
                                                if (Dtime2 < 0) startpt = 0;
                                                if (Dtime2 > 0.003)
                                                        {
                                                        diffdqd2=-1*(dqd2-startdqd);

diffdtd2=(dtd2-startdtd)*0.899;
                                                        if (diffdtd2 > 0) if (diffdqd2
> 0) if (sqrt((dqd2-dqd1)*(dqd2-dqd1)) < 3E-16)
                                                                {
                                                                VOXI[counter] = dvd2;
QI[counter]=diffdqd2; TIMEI[counter]=diffdtd2;
                                                                fprintf(output,"%E\t
%E\t %E\t %E\t %d\n",dvd2,diffdqd2, dvf2,diffdtd2,counter);
                                                                counter=counter+1;
                                                                }
                                                        }
                                                results=results-1; if (results < 0)
i=0;
                                        }
                                        while (i);

dtd1=0;dqd1=0;dvf1=0;dtd2=0;dqd2=0;dvd2=0;dvf2=0;Dtime1=0;Dtime2=0;

diffdqd1=0;diffdqd2=0;startdqd=0;diffdtd1=0;diffdtd2=0;startdtd=0;
counter=counter-1;

fclose(data);
fclose(output);

savei0=0;
savei1=0;
savei2=0;
redundant=0;

if (VOXI[1] < vmax) vmax = VOXI[1];
if (VOXI[counter-1] > vmin) vmin = VOXI[counter-1];

vinitial=2.3; vend=3.5; vdii=(vend-vinitial)/0.001; kc = 0; workcc=0;
f=0;

        timediff=0;
        fitcount=0; n=7;
```

```
                                3vpdischarge.c
        fit_pn(n, A, Y, C);
        savei0=0; savei1=0; savei2=0;
        check_pn(n, C, &rms_err, &avg_err, &max_err);
        C0=C[0];C1=C[1];C2=C[2];C3=C[3];C4=C[4];C5=C[5];C6=C[6];

                        cnr=cnr+1;
                        vctr=1; f=0; vr=vinitial;
                        if (counter > 50)
                        {
                                do
                                {
                                tr=TIMEI[f];

ir=6*C6*tr*tr*tr*tr*tr+5*C5*tr*tr*tr*tr+4*C4*tr*tr*tr+3*C3*tr*tr+2*C2*tr+C1;
                                if (ir > 0)
                                        {
                                        if (workcc==0) {workcl=workcl+1;
workcc=1;}
                                        IF[f]=ir; VOXF[f]=VOXI[f];
fprintf(result,"%E\t%E\n",VOXF[f],IF[f]);
                                        }
                                else vctr=0;
                                f=f+1;
                                if (f == counter) vctr=0;
                                }
                                while(vctr);
                        }
cc=cc-1;
}

while(cc);}
fclose(input);
fclose(result);
return 0;
}
```

fitting.c

```
/* --- The following code comes from C:\lcc\lib\wizard\textmode.tpl. */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>


int counter,vv,memcounter,cc,kc;
char vox[100],ifn[100];
double
v,i,vsum[2500],isum[2500],vcount,c[2500],vout[2500],iout[2500],stddev[2500],qi
sum[2500];
char *pEnd;

int main(int argc, char *argv[])
{

vv=1;
counter=0;

do {

vsum[counter]=0;isum[counter]=0;qisum[counter]=0;c[counter]=0;vout[counter]=0;
iout[counter]=0;counter=counter+1;
        if (counter=120000) vv=0;
        }
while(vv);

FILE *input, *output;
output=fopen("output.txt","w");
counter=0;
vcount=2.7;
vv=1;
cc=1;
kc=0;

do {
        input=fopen("results.txt","r");

        do {
                fscanf(input,"%s %s",vox,ifn); v = strtod (vox, &pEnd); i =
strtod (ifn, &pEnd);
                if (v > vcount-0.0005)  if(v < vcount+0.0005)

{c[counter]=c[counter]+1;isum[counter]=isum[counter]+i;vsum[counter]=vsum[coun
ter]+vcount;qisum[counter]=qisum[counter]+i*i;}

                kc=kc+1;if (kc>120000) cc=0;
           }
        while(cc);

        kc=0;cc=1;
        v=0; i=0;
        fclose(input);
        printf("VOX: %E\n",vcount);
        vcount=vcount+0.001;
        counter=counter+1;
        if (vcount>4.5) vv=0;

        }
while (vv);

memcounter=counter;
counter=0;
vv=1;

do {
        iout[counter]=isum[counter]/c[counter];
        vout[counter]=vsum[counter]/c[counter];
```

fitting.c

```
stddev[counter]=sqrt(1/(c[counter]-1)*(qisum[counter]-c[counter]*iout[counter]
*iout[counter]));
                if (iout[counter] != 0) if (vout[counter] != 0)
fprintf(output,"%E\t %E\t %E\n",vout[counter],iout[counter],stddev[counter]);
        counter=counter+1;
        if (counter == memcounter) vv=0;
        }
while (vv);
fclose(input);
fclose(output);

return 0;
}
```

lfitting.c

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>


int a,counter,vv,ccc,memcounter,cc,kc;
char vi[100],i1[100],i2[100],i3[100],i4[100],i5[100],i6[100];
double v,t,vsum[100000],vout[100000],
time[100000],qvsum[100000],c[100000],stddev[100000];
char *pEnd;

int main(int argc, char *argv[])
{

vv=1;
counter=0;

do {
        time[counter]=0;
vsum[counter]=0;stddev[counter]=0;vout[counter]=0;c[counter]=0;qvsum[counter]=
0;counter=counter+1;
        if (counter=100000) vv=0;
        }
while(vv);

FILE *input, *output;
output=fopen("loutput.txt","w");

a=0;
counter=0;
t=0;
kc=0;
cc=1;ccc=0;
input=fopen("data.txt","r");

        do {
                fscanf(input,"%s%s%s%s%s%s%s",i1,i2,vi,i3,i4,i5,i6); v =
strtod (vi, &pEnd);
                        if (a=3)
                                {
                                c[counter]=c[counter]+1;
                                vsum[counter]=vsum[counter]+v;
                                qvsum[counter]=qvsum[counter]+v*v;
                                time[counter]=t;
                                counter=counter+1; a=0;
                                }
                a=a+1;
                t=t+2;
                ccc=ccc+1;
                if (ccc>=9000) {counter=0;t=0;ccc=0;}
                kc=kc+1;if (kc>=225000) cc=0;
           }
        while(cc);

counter=0;
vv=1;

do {
        vout[counter]=vsum[counter]/c[counter];

stddev[counter]=sqrt(1/(c[counter]-1)*(qvsum[counter]-c[counter]*vout[counter]
*vout[counter]));
        fprintf(output,"%E\t %E\t
%E\n",time[counter],vout[counter],stddev[counter]);
        if (stddev[counter]==0) vv=0;
        counter=counter+1;
        }
while (vv);
fclose(input);
```

```
fclose(output);

return 0;
}
```

mfitting.c

```
/* --- The following code comes from C:\lcc\lib\wizard\textmode.tpl. */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>


int counter,vv,ccc,memcounter,cc,kc;
char vi[100],i1[100],i2[100],i3[100],i4[100],i5[100],i6[100];
double v,t,vsum[100000],vout[100000],
time[100000],qvsum[100000],c[100000],stddev[100000];
char *pEnd;

int main(int argc, char *argv[])
{

vv=1;
counter=0;

do {
        time[counter]=0;
vsum[counter]=0;stddev[counter]=0;vout[counter]=0;c[counter]=0;qvsum[counter]=
0;counter=counter+1;
        if (counter=100000) vv=0;
        }
while(vv);

FILE *input, *output;
output=fopen("moutput.txt","w");

counter=0;
t=0;
kc=0;
cc=1;ccc=0;
input=fopen("data.txt","r");

        do {
                fscanf(input,"%s%s%s%s%s%s%s",i1,i2,vi,i3,i4,i5,i6); v =
strtod (vi, &pEnd);
                if (v > 0.3) if (v < 1.6)
                        {
                                c[counter]=c[counter]+1;
                                vsum[counter]=vsum[counter]+v;
                                qvsum[counter]=qvsum[counter]+v*v;
                                time[counter]=t;
                                counter=counter+1;
                        }
                t=t+(0.0005);
                ccc=ccc+1;
                if (ccc>=10000) {counter=0;t=0;ccc=0;}
                kc=kc+1;if (kc>=200000) cc=0;
           }
        while(cc);

counter=0;
vv=1;

do {
        vout[counter]=vsum[counter]/c[counter];

stddev[counter]=sqrt(1/(c[counter]-1)*(qvsum[counter]-c[counter]*vout[counter]
*vout[counter]));
        fprintf(output,"%E\t %E\t
%E\n",time[counter],vout[counter],stddev[counter]);
        if (stddev[counter]==0) vv=0;
        counter=counter+1;
        }
while (vv);
fclose(input);
fclose(output);
```

```
return 0;
}
```

Literaturverzeichnis

[1] Kwok K. NG. *Complete Guide to Semiconductor Devices, John Wiley and Sons, 2002*

[2] Katsuhiko Ohsaki. *A Single Poly EEPROM Cell Structure for Use in Standard CMOS Processes, IEEE Journal of Solid-State Circuits, Vol.29, No.3, March 1994*

[3] Paolo Pavan. *Floating Gate Devices: Operation and Compact Modeling, Kluwer Academic Publishers, 2004*

[4] Phillip E. Allen. *CMOS Analog Circuit Design, Oxford University Press, 1987*

[5] K.F. Schuegraf. *Ultra-thin Silicon Dioxide Leakage Current and Scaling Limit, Symposium on VLSI Technology Digest of Technical Papers, 1992*

[6] Wen-Chin Lee. *Modeling CMOS Tunneling Currents Through Ultrathin Gate Oxid Due to Conduction- and Valence-Band Electron and Hole Tunneling, IEEE Transactions on Electron Devices, Vol.48, No.7, July 2001*

[7] André Srowig. *Trajectory Sensor and Readout Electronics of a Cosmic Dust Telescope*

[8] F. Schwabl. *Quantenmechanik, Springer-Verlag, 3.Auflage*

[9] J. Schemmel. *A New VLSI Model of Neural Microcircuits Including Spike Time Dependent Plasticity, IEEE Press, pp. 1711-1716, 2004*

[10] P. Hasler. *A CMOS Programmable Analog Memory-Cell Array Using Floating-Gate Circuits, IEEE Transactions on Circuits and Systems, Vol.48, No.1, 2001*

[11] R. Rojas. *Theorie der neuronalen Netze, Springer-Verlag, 4.Auflage*

[12] R. Geiger. *VLSI Design Techniques for Analog and Digital Circuits, McGraw-Hill, 1990*

Danksagung

Mein Dank gilt allen, die zur Durchführung dieser Diplomarbeit beigetragen haben. Ganz besonders möchte ich danken:

Herrn Prof. Dr. Karlheinz Meier für die Bereitstellung dieses interessanten Themengebietes und für die freundliche Aufnahme in seine Forschungsgruppe

Herrn Prof. Dr. Norbert Herrmann für die Übernahme der Zweitkorrektur

Dr. Johannes Schemmel für die umfassende, intensive und freundliche Betreuung

Dr. André Srowig für die unzählbaren Gespräche und Diskussionen, sowie für die vielen Hilfestellungen

Andreas Grübl für seine enorme Hilfsbereitschaft bei nahezu allen technischen Fragen und Problemen

Der gesamten Electronic Visions Gruppe für die freundliche und herzliche Arbeitsatmosphäre

Meinen Eltern für die umfassende Unterstützung während meines gesamten Studiums

Meiner Schwester Meike für die vielen Aufmunterungen und für die Korrektur der Arbeit

Meiner zukünftigen Ehefrau Julia Raberg einfach für alles.

Erklärung:
Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe.

Heidelberg, den ....