Softwaregestütztes Reverse-Engineering von Logik-Gattern in integrierten Schaltkreisen

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Autor: Martin Schobert
Gutachter: Prof. Dr. rer. nat. Jens-Peter Redlich und Dipl.-Inf. Henryk Plötz,
Humboldt-Universität zu Berlin
Dr. Karsten Nohl, Security Research Labs GmbH, Berlin
Abgabe: Juni 2011

Für Mila und Anett.

Zusammenfassung

Die Rekonstruktion von Logik in integrierten Schaltkreisen ist notwendig, um Aussagen über hardwarebasierte Sicherheitssysteme treffen zu können. Dieses Reverse-Engineering von Chips ist zwar aufwendig – die Arbeitsschritte sind jedoch sehr gradlinig. Um den Aufwand zu beherrschen, bedarf es lediglich geeigneter Werkzeuge. In dieser Diplomarbeit werden Verfahren für die Schaltkreisrekonstruktion vorgestellt. Des Weiteren wird die Software Degate beschrieben, die den Prozess des Reverse-Engineerings maßgeblich unterstützt. Als Eingabedaten dienen Bilder mehrschichtiger Schaltkreise. Ausgabedaten sind Netzlisten in einer Hardwarebeschreibungssprache.

Degate ist die erste freie Software, welche die wesentlichen Arbeitsschritte des Reverse-Engineerings abbildet. Degate wurde bereits in mehreren Projekten zur Rekonstruktion sicherheitsrelevanter Funktionen eingesetzt.

Dokumentenversionen

Diese Version enthält verglichen mit der abgegebenen Diplomarbeit kleinere Änderungen. Die Art der Änderungen sind in der folgenden Tabelle dokumentiert.

Datum	**Beschreibung**
06.06.2011	Druckversion/Abgabeversion
03.07.2011	Layoutkorrektur: Leerseite eingefügt
12.09.2011	Korrektur einiger Rechtschreibfehler, Aufnahme einer Versionshistorie

Erklärung zur Urheberschaft

Hiermit erkläre ich, dass ich die vorliegende Arbeit allein und nur unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 3. Juni 2011.

Martin Schobert

Lizenzbedingungen

Akronymverzeichnis

API	Application Programming Interface
AR	Acceptence Rate, Erkennungsrate
ASIC	Application-Specific Integrated Circuit
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CAD	Computer-Aided Design
CIF	Common Intermediate Format
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DECT	Digital Enhanced Cordless Telecommunications
DPA	Differential Power Analysis
DRAM	Dynamic Random Access Memory
DRC	Design Rule Checking
DSC	DECT Standard Cipher
EDIF	Electronic Design Interchange Format
ERC	Electrical Rule Checking
FAQ	Frequently Asked Questions
FAR	False Acceptence Rate, Fehlerkennungsrate
FET	Feldeffekttransistor
FIB	Focused Ion Beam
FPGA	Field-Programmable Gate Array
GDS	Graphic Database System
GDSII	GDS II Stream Format
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HDL	Hardware Description Language

HSM	Hardware Security Module
HTML	Hypertext Markup-Language
IC	Integrated Circuit
ID	Identifizierer
IEEE	Institute of Electrical and Electronics Engineers
IT	Informationstechnologie
IV	Initialisierungsvektor
LFSR	Linear Feedback Shift Register
MOS	Metal-Oxide Semiconductor
PDF	Portable Document Format
RAM	Random Access Memory
RC	Rule Checking
RFID	Radio Frequency Identification
RGB	Rot, Grün, Blau
RGBA	Rot, Grün, Blau und Alpha
ROID	Remote Object ID
ROM	Read-Only Memory
RPC	Remote Procedure Calls
RTL	Register Transfer Level
SoC	System on a Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static Random Access Memory
TCL	Tool Command Language
URI	Uniform Resource Identifier
VCD	Value Change Dump
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits
VLSI	Very-Large-Scale Integration
XML	Extensible Markup Language
XML-RPC	XML-basierte Remote Procedure Calls
XNOR	Exklusiv-Oder mit anschließender Negation
XOR	Exklusiv-Oder

Inhaltsverzeichnis

Kapitel 1

Einleitung

Viele Sicherheitsfunktionen in IT-Systemen sind hardwarebasiert. Häufig werden Chipkarten und RFID-Transponder eingesetzt, z. B. um Bezahl- und Zugangskontrollsysteme zu realisieren. Diese verwenden geheime Schlüssel und kryptografische Verfahren, um sich beispielsweise gegenüber einem Dienst zu authentisieren [Fin02, S. 225ff]. Die Chipkarte speichert die Schlüssel und implementiert die kryptografischen Verfahren in Soft- oder Hardware und schützt diese Informationen vor unbefugtem Zugriff. Nicht selten sind die eingesetzten Sicherheitsverfahren in Chips proprietär, also nicht öffentlich bekannt und entziehen sich damit einer direkten Analyse. Um dennoch Aussagen über deren Sicherheit zu treffen, müssen Chipkarten auf Hardwareebene analysiert werden.

Die Chipkarten werden in der Regel an Kunden ausgegeben und sind damit zunächst nicht mehr im Kontrollbereich des Systembetreibers. Dadurch ist es für einen potentiellen Angreifer einfach, Muster von Sicherheitschips zu erlangen, diese zu analysieren und die Sicherheitsfunktionen ggf. zu umgehen. Je nach Anwendungszweck gibt es häufig kein Risiko, bei der missbräuchlichen Nutzung entdeckt zu werden. Insbesondere im Bereich des Pay-TVs führt dies regelmäßig zu Schattenmärkten mit gefälschten Decoderkarten.

Wenn eine Sicherheitsannahme darin besteht, dass die Rekonstruktion von integrierten Schaltkreisen teuer und in hohem Maße anspruchsvoll sei, dann ist das Reverse-Engineering von Chips eine Möglichkeit, die Gültigkeit dieser Annahme zu testen. Verschiedene Forschungsergebnisse zeigen, dass sich Schaltungen zu geringen Kosten zurückgewinnen lassen und die Sicherheitsannahme nicht statthaft ist. Unter anderem

führte die Anwendung dieser Reverse-Engineering-Methoden auf integrierte Schaltkreise in den letzten Jahren zu einigen Publikationen über zuvor geheimgehaltenen Verschlüsselungsverfahren. Durch die Aufklärung dieser Verfahren ließen sich jeweils sofort Angriffsmöglichkeiten entwickeln [Noh+08] [NP07].

Chipanalysen werden seit Anbeginn der Halbleiterindustrie betrieben, um Schaltkreise nachzubauen, Patentverletzungen festzustellen und Technologien konkurrierender Unternehmen zu beobachten. Schaltkreisrekonstruktionen zum Zweck der öffentlichen Sicherheitsevaluation werden bisher nur vereinzelt durchgeführt. Drei Faktoren sind dafür relevant: Wissen, Hardware und Software. Die Methoden zur Rekonstruktion von Schaltkreisen lassen sich in wenigen Wochen erlernen. Die Anschaffungskosten für Geräte betragen etwa 1.000 bis 10.000 Euro und entfallen, da die Geräte an vielen Forschungseinrichtungen bereits vorhanden sind. Die Software nimmt eine Schlüsselrolle ein, denn kommerzielle Lösungen sind nicht kostengünstig lizensierbar und die Entwicklung eigener Werkzeuge beansprucht viel Zeit. Das Ziel dieser Diplomarbeit ist, für das Softwareproblem Abhilfe zu schaffen.

1.1 Motivation

Rekonstruktion sicherheitsrelevanter Verfahren und Implementationen

Aussagen über die Sicherheit unbekannter Verfahren bzw. unbekannter Implementationen gängiger sicherheitsrelevanter Verfahren sind schwierig bis unmöglich. Für eine Sicherheitsanalyse ist daher die Rekonstruktion der Algorithmen notwendig. Insbesondere proprietäre Verschlüsselungsverfahren sind oft geeignete Kandidaten für Angriffe.

Die Geheimhaltung eines kryptografischen Verfahrens sagt zwar prinzipiell nichts über die Widerstandsfähigkeit gegen Angriffe aus, in der Vergangenheit gab es jedoch zahlreiche Probleme mit proprietären Verfahren. Diese sind darin begründet, dass – wenn überhaupt – nur wenige Experten die Verfahren evaluieren konnten, die Ergebnisse aber nicht öffentlich sind. Die Sicherheit eines Verfahrens lässt sich nicht beweisen. Es lässt sich nur zeigen, mit welchem Aufwand und welcher Methode ein kryptografischer Algorithmus nach dem Stand der Forschung gebrochen werden kann.

In einigen Fällen besteht sogar die Notwendigkeit zum Einsatz proprietärer Verschlüsselungs- bzw. Verschleierungsverfahren in integrierten Schaltkreisen. Um beispielsweise Speicherinhalte von Smartcards gegen optisches oder elektronisches Auslesen zu schützen, werden für sicherheitsrelevante Anwendungen die Speicherinhalte verschlüsselt, etwa für Hardware Security Modules (HSM). Wäre das Verfahren und gegebenfalls der Schlüssel zur Entschlüsselung bekannt, ließe sich der Speicherinhalt als Klartext einfacher rekonstruieren [RE08, S. 747].

Neben Sicherheitsanalysen kann die Enthüllung von Verschlüsselungsverfahren dazu dienen, diese Verfahren in Open-Source-Software zu implementieren. Einige Verfahren, z. B. A5/1, der DECT Standard Cipher (DSC) und der GPRS Encyption Algorithm (Version 1 und 2), die auf der Luftschnittstelle in funkbasierten Kommunikationssystemen verwendet werden, sind bzw. waren der Öffentlichkeit unbekannt. Es besteht prinzipiell die Möglichkeit, diese Algorithmen im Rahmen von Lizenzvereinbarungen zu nutzen. Open-Source-Projekte profitieren nicht davon, da die Geheimhaltung dieser Verfahren im Widerspruch zum Open-Source-Gedanken steht. Wenn die Verfahren bekannt und Teile davon nicht (mehr) durch Patente geschützt sind, können die Verfahren in Open-Source-Projekten Verwendung finden. Ein weiteres Beispiel ist das Verschleierungsverfahren von Legic prime, das mittlerweile in Form von Programmcode in das Projekt Proxmark eingeflossen ist. Dabei handelt es sich um eine offene Hardware zum Umgang mit RFID-Transpondern und -Lesegeräten. Der Code für das Verschleierungsverfahren ermöglicht, dass mit der Proxmark-Hardware Transponder vom Typ Legic prime ausgelesen und theoretisch auch emuliert werden können.

Die Kenntnis von Konstruktionsmerkmalen kryptografischer Verfahren kann für weiterführende Angriffe von Bedeutung sein. Insbesondere bei hardwarebasierten Verschlüsselungsverfahren können Informationen über deren Aufbau Angriffe mittels differentieller Stromverbrauchsanalyse (DPA) vereinfachen. Der Stromverbrauch bei DPA-Angriffen wird häufig auf der Basis des Hamminggewichts bzw. der Hammingdistanz modelliert. Präzisere Modelle reduzieren nicht nur die Menge an Messreihen, die für DPA-Angriffe notwendig sind, in einigen Fällen ermöglichen sie erst einen Angriff (vgl. [MOP07, S. 131ff]).

Entwicklung freier Werkzeuge

Um Algorithmen aus integrierten Schaltkreisen mittels Reverse-Engineering zurückzugewinnen, bedarf es – aufgrund der Komplexität der Schaltkreise – geeigneter Software. Dafür existieren prinzipiell Programme (siehe Abschnitt 1.4). Diese sind aber entweder nicht öffentlich zugänglich oder deren Lizenzkosten bewegen sich in einem Rahmen, der für Privatpersonen, kleinere mittelständische IT-Sicherheitsfirmen und universitäre Forschungsgruppen unerschwinglich ist.

Für das Reverse-Engineering von Halbleiterstrukturen gibt es bisher keine Open-Source-Werkzeuge. Kostenlose Software begünstigt jedoch deren Verbreitung und senkt damit Einstiegshürden. Ferner ermöglicht freie Software, dass die Werkzeuge von Dritten angepasst und verbessert werden können.

1.2 Aufgabenstellung

Es sollen Verfahren zum Reverse-Engineering von digitaler Logik in integrierten Schaltkreisen entwickelt und in einer Software implementiert werden, welche den Reverse-Engineering-Prozess unterstützt. Gegenstand der Analysen sind auf Standardzellen aufbauende Anwendungspezifische Schaltkreise (ASICs). Aus diesen sollen hauptsächlich Verschlüsselungsverfahren softwaregestützt gezielt rekonstruiert werden.

Als Ausgangsmaterial dienen Bilddaten der einzelnen Schichten eines Chips. Das Ergebnis der Rekonstruktion ist eine Verhaltensbeschreibung der Standardzellen und eine Netzliste, welche die Struktur der Schaltkreise darstellt.

Das Reverse-Engineering ist defacto nur als semi-automatischer Prozess möglich. Nicht alle Arbeitsschritte sind automatisierbar. Es soll jedoch ein möglichst großer Automatisierungsgrad erreicht werden, wenn dies im Verhältnis zum Nutzen steht.

Komplexere integrierte Schaltkreise enthalten nie ausschließlich digitale Schaltungen, sondern auch partiell analoge Komponenten. Wenn dieser analoge Anteil für die Schaltung bedeutend ist, werden derartige Realisierungen als „mixed signal“ bezeichnet. Beim Reverse-Engineering werfen diese analogen Schaltungen Fragen auf, die ohne detaillierte Kenntnisse der Technologie und der physikalischen Modelle kaum beantwortet werden können. Ein Beispiel dafür ist die Kapazität eines Kondensators, welche vom Abstand der potentialtragenden Elemente und vom Dielektrikum abhängt. Da

der Fokus dieser Arbeit digitale Schaltkreise ist, sollen analoge Komponenten hier nicht berücksichtigt werden.

1.3 Beitrag dieser Arbeit

Diese Diplomarbeit beschreibt Verfahren zur Rekonstruktion von Schaltkreisen in Chips. Die Arbeitsschritte sind in einer Software abgebildet. Die mittel- und langfristigen Beiträge sind:

- Gestaltung eines planbaren Prozesses: Durch die Werkzeugunterstützung wird die Schaltkreisrekonstruktion zu einem gradlinigen, planbaren Prozess. Eine Software unterstützt nicht nur bei der Abwicklung automatischer Prozesse, sondern auch bei der Verwaltung von Metadaten. Ferner lassen sich der Zeitaufwand und damit die Kosten für die Funktionsrekonstruktion anhand vergleichbar umfangreicher Reverse-Engineering-Projekte abschätzen.
- Bereitstellung eines niederschwelligen Zugangs: Nachvollziehbare, softwaregestützte Arbeitsschritte senken die Einstiegshürden für die Schaltkreisrekonstruktion. Mit dem Veröffentlichen der Software unter einer Open-Source-Lizenz werden zusätzlich finanzielle Hürden gesenkt.
- Öffentliche Evaluation kryptografischer Verfahren: In der Praxis eingesezte Verfahren, die sich einer Evaluation bisher durch Geheimhaltung entzogen haben, lassen sich mit den hier dargestellten Methoden rekonstruieren. Dadurch erhalten auch Kryptografen Zugang zu neuem Forschungsmaterial.

1.4 Existierende Software

Die bekanntesten Firmen, die kommerziell Reverse-Engineering durchführen sind Chipworks Inc., Taeus, UBM TechInsights, Materials Analysis Technology Inc. (MaTek) und SiVenture [Kum00]. Diese Firmen haben eigene Softwarelösungen oder lizensieren Software, die den Reverse-Engineering-Prozess unterstützt. Über drei Softwarepakete findet man Informationen: ICWorks der Firma Chipworks, Matrix der Firma UBM TechInsights und Werkzeuge der Firma Cellixsoft. Die Informationslage ist jeweils dürftig. Es ist jedoch erkennbar, dass die Softwaresysteme für das Reverse-Engineering eingesetzt werden.

Das Softwarepaket Matrix wurde ursprünglich von der chinesischen Firma Sanguine Microelectronics Corporation entwickelt. United Business Media Ltd. kaufte sie 2008 und integrierte sie in die Firma Semiconductor Insights. Dadurch entstand die UBM TechInsights [Tec08]. Verschiedene Reverse-Engineering-Firmen lizensieren die Software Matrix, u. a. Taeus.

Wenn Analyseergebnisse an Kunden übergeben werden, bekommt der Kunde eine Software, mit der er durch die Bilddaten und Schaltpläne navigieren kann. Diese Aufgabe übernehmen die Werkzeuge Matrix1 und Matrix Reader. Die Software verwaltet Kreuzreferenzen zwischen Bildern bzw. Layout und Schaltplänen, so dass diese Ansichten verglichen werden können.

Für die Auswertung der Bilddaten kommen die Werkzeuge Matrix Analyzer und Matrix Prisma zum Einsatz. Mit diesen lassen sich Netzlisten und Schaltpläne rekonstruieren. Analysen können mittels TCL-Skripten automatisiert werden. Ergebnisse lassen sich als GDSII, CIF, EDIF, SPICE, Verilog, VHDL und PDF exportieren [Yao+05].

Die hierarchische Organisation von Schaltungen ist ein Hilfsmittel bei der Vorwärtsentwicklung von Chips. Bei der Umsetzung auf Halbleiterstrukturen geht diese Hierarchieinformation vollständig verloren. Ansätze sind gelegentlich zu erkennen, wenn funktional eigenständige Komponenten in separaten Bereichen auf dem Chip platziert sind. Dies gilt etwa häufig für Mikroprozessoren, die als System-on-a-Chip verwendet werden. Wenn Schaltkreise aus Netzlisten rekonstruiert werden, dann sind die Schaltpläne hierarchielos. Matrix Analyzer unterstützt bei der Rückgewinnung von Designhierarchien. Leider gibt es keine öffentlichen Informationen, welche Ansätze die Software dafür nutzt.

Die kanadische Firma Chipworks Inc. hat eigene Software für das Reverse-Engineering und die Präsentation von Ergebnissen entwickelt. Ähnlich wie bei der Matrix-Produktreihe wird an den Kunden eine funktional eingeschränkte Version ausgeliefert, die zum Navigieren durch die Bilddaten und zum Verfolgen von Signalpfaden dient, Schaltpläne anzeigen kann und Kreuzreferenzen zu den Bilddaten verwaltet. Die Software nennt sich ICWorks Browser[1].

Der ICWorks Arranger ist Chipworks Werkzeug, um hierarchische Schaltpläne ausgehend von „ausgerollten“ Netzlisten zu erstellen. Erfahrene Analysten führen diesen

[1] Chipworks ICWorks Browser, `http://www.chipworks.com/en/technical-competitive-analysis/resources/reerse-engineering-software/137-icworks-browser`, besucht am 23. Mai 2011

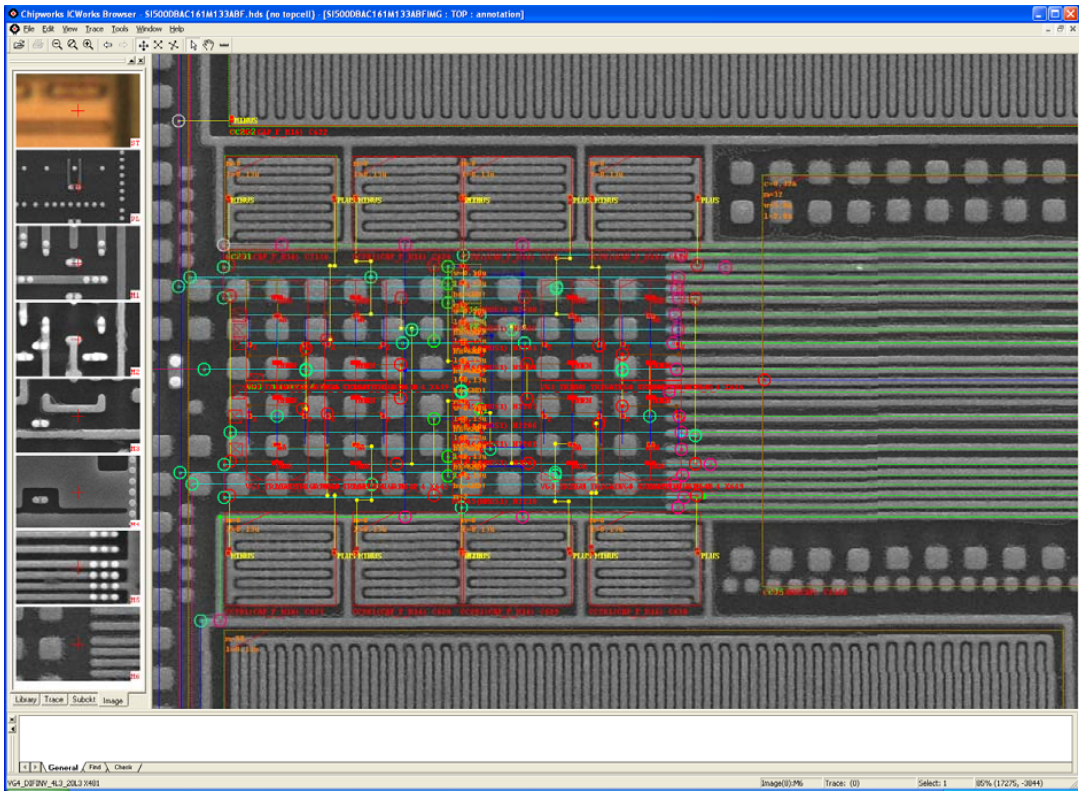

Abbildung 1.1: Screenshot der Software ICWorks Browser (Quelle: [Tor09])

Arbeitsschritt für ausgewählte Teile des gesamten Schaltplans durch. Die Software findet „einfache Strukturen“ [Tor09] automatisch. Identische Teilschaltkreise findet der ICWorks Arranger ebenfalls automatisch und organisiert sie entsprechend [TJ09]. Chipworks Software ist für die Analyse von analogen und digitalen Schaltungen ausgelegt.

Die chinesische Firma Cellixsoft bietet Analysedienstleistungen an und entwickelt Reverse-Engineering-Software[2] für integrierte Schaltkreise. Diese Software kann lizensiert werden. Nach Eigenangaben wurden mehr als 4000 Softwarelizenzen ausgegeben. Die Produktpalette umfasst drei Werkzeugpakete. Das Softwarepaket Filmshop dient dem Mikroskopieren, dem Zusammenfügen von mikroskopierten Bildern und deren Nachbearbeitung. Das Paket ChipLogic enthält Programme für die Bilderkennung und zur Netzlistenextraktion, sowie zum Bearbeiten von Layoutdaten. Des Weiteren enthält das Paket Werkzeuge für kollaboratives Arbeiten und zum optischen Auslesen

[2]Beschreibung der Softwareprodukte,
`http://cellixsoft.com/cellixen/Product.htm`, besucht am 23. Mai 2011

von Masken-ROMs. Das Paket Hierux System bietet Programme zur Hierarchisierung von Schaltungen, da die extrahierten Netzlisten zunächst hierarchielos sind. Ebenso wie bei den Matrix-Tools und den Werkzeugen von Chipworks gibt es für die Weitergabe von Analyseergebnissen funktional eingeschränkte Versionen, die ausschließlich zum Ansehen von Bilddaten und Schaltplänen dienen.

Die litauische Firma Radiolinija UAB [3] betreibt neben der Entwicklung von „Spyphones“ und Diagnosewerkzeugen für den Kfz-Bereich das Reverse-Engineering von Chips[4]. Für Zwecke der Schaltkreisrekonstruktion hat diese Firma wahrscheinlich eigene Software entwickelt, um ausgehend von mehrschichtigem Bildmaterial Netzlisten zu rekonstruieren. Für den Kunden werden SPICE-, VHDL-, Verilog- und EDIF-Daten exportiert. Zum Auslieferungsgegenstand gehören auch hierarchische Schaltpläne. Im Zusammenhang mit Radiolinija ist bekannt, dass dort ein Werkzeug mit dem Namen „RETool“ eingesetzt wird. Es ließ sich bis zuletzt nicht klären, welchen Funktionsumfang diese Software aufweist. Aus einer englischsprachigen Zusammenfassung der Dissertation von Giedrius Masalskis, einem Mitarbeiter von Radiolinija, ergibt sich, dass er Bilderkennungsverfahren in diese Software implementiert hat [Mas10]. Im Rahmen der Dissertation hat Masalskis seit 2008 mehrfach zu Bildverarbeitungsproblemen bei der Layoutrekonstruktion veröffentlicht, z. B. in [MN08] und [MN10].

1.5 Ähnliche Projekte

Neben diesen vier real existierenden Softwaresystemen gibt es Veröffentlichungen, die Versuche erkennen lassen, Algorithmen und Software für das Reverse-Engineering von Schaltkreisen ausgehend von Bildaufnahmen zu entwickeln. Ein Beispiel dafür ist ein System namens Antistrofeas, welches in einem Spezifikations-Paper von 1991 vorgestellt wurde und u. a. Mikroskopaufnahmen verarbeiten soll, um VLSI-Schaltkreise zu rekonstruieren [BR91]. Einem Paper von 2002 nach dienen nunmehr farbige CAD-Daten als Eingabedaten [Bou+02]. Darüber hinaus ist nichts über das System bekannt. Eine Nachfrage per E-Mail blieb bislang unbeantwortet.

Das junge Projekt visual6502.org verfolgt das Ziel, in historischen Homecomputern verwendete Chips soweit zu rekonstruieren, dass deren Schaltvorgänge grafisch

[3] `http://www.radiolinija.lt/`, besucht am 23. Mai 2011
[4] `http://www.semiresearch.com`, besucht am 23. Mai 2011

darstellbar werden. Das Projekt begann mit dem CPU-Typ 6502 der Firma MOS Technology. Diese dient im „Homecomputer C64“ als Prozessor. Sie wurde mit einem GUI-Tool manuell auf der Ebene von Polygonen reverse-engineered. Die dabei entstehenden Layoutdaten wurden in eine Netzliste überführt, die wiederum von einem Simulator verarbeitet wird. Die Dauer der Rekonstruktion ist in der Projekt-FAQ mit sechs Wochen angegeben. Der Grund, warum die Rekonstruktion manuell erfolgte, wird in der FAQ ebenfalls genannt. Die automatische Analyse führte zu vielen Fehlern, die bei der manuellen Rekonstuktion nicht auftreten und deren Behebung mit höherem Aufwand verbunden wäre als die manuelle Rekonstruktion[5]. Ein wesentlicher weiterer Grund für den manuellen Ansatz ist, dass ältere CPUs nicht auf einem Standardzellendesign basieren. Standardzelleninstanzen können mittels Bilderkennungsalgorithmen einfacher detektiert werden.

1.6 Projektwebseiten

Alle Informationen über die Software Degate sind auf der Webseite `http://degate.org/` abrufbar. Dies umfaßt die API-Bescheibung der libdegate, Anleitungen zum Übersetzen des Codes, Testprojektdaten, die FAQ und eine Literaturliste über gesammelte Veröffentlichungen, die sich mit Chip-Reverse-Engineering auseinandersetzen oder für die Entwicklung entsprechender Software interessant sind.

Der Sourcecode der Software Degate ist vollständig öffentlich. Der Sourcecode wird auf der Entwicklerplattform `github.com` unter `http://github.com/nitram2342/degate/` gespeichert.

[5] Visual Transistor-level Simulation of the 6502 CPU and other chips, `http://visual6502.org`

Kapitel 2

Aufbau digitaler integrierter Schaltkreise auf CMOS-Basis

Im Folgenden wird der Aufbau digitaler Logik auf der Basis von CMOS in integrierten Schaltkreisen beschrieben, soweit es für das Verständnis dieser Arbeit notwendig ist. Eine detaillierte Beschreibung ist der Fachliteratur zu entnehmen, beispielsweise bei Weste und Harris [WH05]. Wesentliche Punkte für das Reverse-Engineering wurden in der vorangegangen Studienarbeit behandelt [Sch10].

Elektronische Schaltungen, die auf einem gemeinsamen Substrat in kompakter Form aufgebaut sind, bezeichnet man als integrierte Schaltkreise. Das Substrat ist nicht nur die Grundfläche, auf der die Schaltkreiselemente aufgebracht werden, sondern auch aktiver Teil der Schaltung. Als Substratmaterial wird fast ausschließlich Silizium verwendet. Aus hochreinem Silizium werden Einkristalle gezüchtet. Diese werden in Scheiben geschnitten und nachbearbeitet. Dadurch entstehen die sogenannten Wafer.

2.1 Schichtweiser Aufbau

Integrierte Schaltkreise sind mehrschichtig aufgebaut. Die einzelnen Schichten entstehen dadurch, dass auf dem Substrat beispielsweise mittels Oxidation, Ionenimplantation, chemischer Gasphasenabscheidung und Kathodenzerstäubung (Sputtern) Materialien auf- bzw. eingebracht werden. Um ausgewählte Flächen bearbeiten zu können, werden sie maskiert: Eine fotoempfindliche Schicht wird aufgetragen und mittels chrombedeckter Quarzglasmasken selektiv belichtet. Mit zunehmender Inte-

grationsdichte und feiner werdenden Strukturen werden kleinere Wellenlängen des Lichts benutzt. Man verwendet ultraviolette bzw. extrem ultraviolette Strahlung. Entwicklerlösung löst die fotoempfindliche Schicht an unbelichteten bzw. an belichteten Stellen auf, abhängig davon, ob negativer oder positiver Fotolack verwendet wird. Stellen mit zurückgebliebenem Fotolack schützen darunter liegende Strukturen für einen Bearbeitungsschritt. Der Fotolack wird vor dem Aufbau weiterer Schichten chemisch entfernt. Diese Art der Bearbeitung wird als Fotolithografie bezeichnet. [WH05, S. 23ff, 114ff]

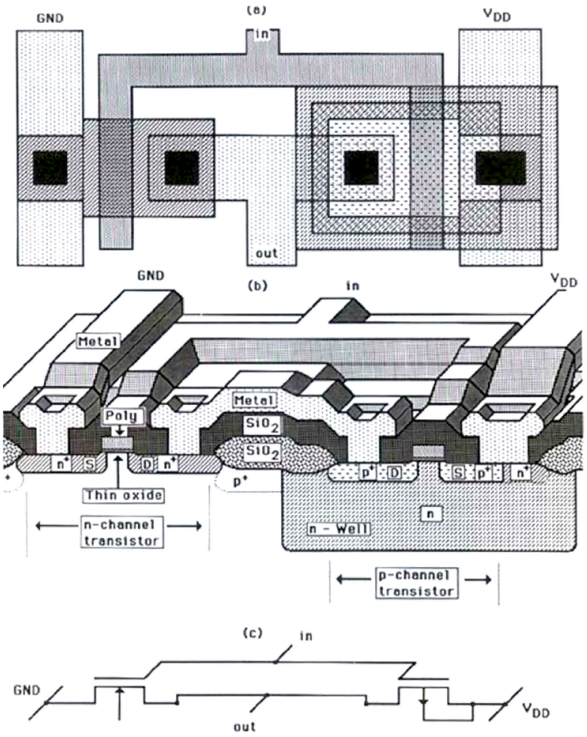

Abbildung 2.1: CMOS-Inverter: (a) Schematische Darstellung der Draufsicht als Stick-Diagramm, (b) schematischer Querschnitt der Halbleiterstrukturen, (c) Schaltbild (Quelle: [Mal87, S. 191])

Abbildung 2.1 (b) zeigt schematisch das Ergebnis der Bearbeitungsschritte. Die untersten Schichten formen die Transistoren.[1] Bei den Transistoren handelt es sich um Feldeffekttransistoren (FET). Die p- und n-dotierten Diffusionsbereiche[2] der Tran-

[1] Grundsätzlich müssen zwar Transistoren nicht auf den untersten Schichten eines integrierten Schaltkreises aufgebaut werden, in allen in Serienproduktion gefertigten Chips ist dies jedoch zutreffend (vgl. [Bal04]).

[2] Die Bereiche werden aus historischen Gründen als *diffusion areas* bezeichnet, selbst wenn andere

sistoren sind direkt in das Substrat eingearbeitet. Über diesen Diffusionsbereichen liegen die Leiterbahnen, welche die Anschlüsse der Feldeffekttranistoren bilden – für Source, Gate und Drain. Früher bestanden diese Leiterbahnen aus aufgedampftem Metall. Seit den 1970er Jahren kommt dotiertes polykristallines Silizium zum Einsatz. Der Gate-Anschluss der Transistoren ist gegenüber dem Substrat mittels einer Oxidschicht elektrisch isoliert, denn das Gate soll lediglich zur Ausbildung eines elektrischen Feldes beitragen. Aufgrund der verwendeten isolierenden Oxidschicht werden derartige Strukturen als Metalloxidhalbleiter (MOS) bezeichnet.

CMOS (engl. *Complementary MOS*) ist die führende Halbleitertechnologie für den Aufbau von Prozessoren und Speichern (DRAM und SRAM). In der CMOS-Halbleitertechnik werden n-Kanal- und p-Kanal-FETs verwendet. Beide Transistortypen werden paarweise in Schaltungen eingesetzt und sind komplementär geschaltet.

Die „Strukturgröße“ eines Halbleiterprozesses gibt die Abmessung der kleinsten Schaltungsstrukturen an. Der Begriff ist nicht eindeutig definiert. Im Zusammenhang mit Feldeffekttransistoren wird häufig die Länge der kleinsten Gate-Anschlüsse als Strukturgröße aufgefasst.

2.2 Standardzellenbibliotheken

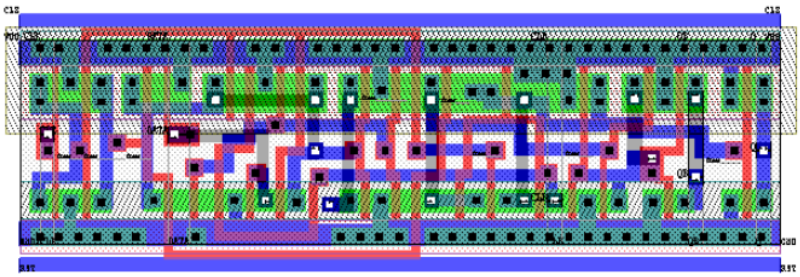

Abbildung 2.2: Layout eines D-Flipflops (Quelle: [Tan99])

Die Anordnung der Transistoren auf dem Substrat ist nicht beliebig. Vielmehr folgt sie einem Schema, welches mit möglichen Abweichungen in allen CMOS-Schaltungen in Chips zu finden ist. Für die Umsetzung digitaler Schaltungen werden sogenannte Standardzellen verwendet. Diese Standardzellen sind manuell entworfene, sorgfältig

Verfahren zum Einbringen von Fremdatomen verwendet werden, z. B. Ionenimplantation.

simulierte und praktisch erprobte Bausteine, die einfache Funktionen darstellen. Diese Standardzellen sind in technologiespezifischen Zellenbibliotheken hinterlegt. Beispielsweise enthält die Bibliothek in [Tan99] für einen 0.35 μm-Prozess 48 verschiedene Zellentypen. Flipflops sind mit bis zu 20-30 Transistoren in vielen Standardzellenbibliotheken die komplexesten Elemente (vgl. Abbildung 2.2).

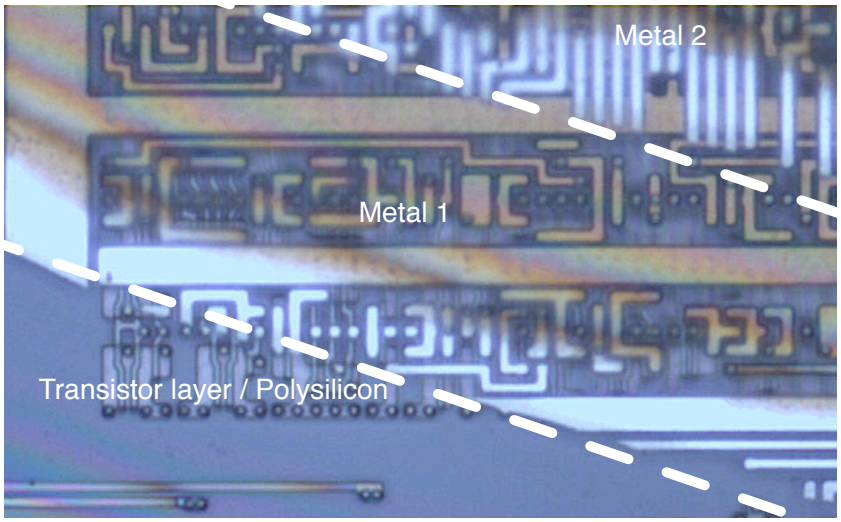

Abbildung 2.3: Schnitt durch verschiedene Ebenen eines Chips.

In der vorliegenden Arbeit werden überwiegend die Begriffe Standardzellentyp und Standardzelleninstanz genutzt. Eine Standardzelleninstanz ist dabei eine auf dem Chip platzierte Standardzelle. Der Begriff Gatter bzw. die englische Form „gate“ sind ungeeignete Begriffe, da sie in verschiedenen Kontexten unterschiedliche Bedeutungen haben. So bezeichnet ein Gate einen Anschluss an einem Feldeffekttransistor. Die Realisierung elementarer Logikoperationen wird häufig als Gatter bezeichnet (z. B. NAND-Gatter), welche wiederum Bestandteil komplexerer Standardzellen sind. Nicht zuletzt werden Standardzellen Gatter genannt. Aus historischen Gründen wird der Begriff Gate in der Software Degate verwendet, sowohl im Programmcode als auch in der grafischen Benutzeroberfläche. Sie haben dort die Bedeutung von Standardzellentypen bzw. -instanzen.

Von Hand optimierte komplexere Schaltungen sind eine Ausnahmeerscheinung. Manuelle Optimierungen werden, wenn überhaupt, nur für kritische Datenpfade vorgenommen und wenn der Aufwand in Relation zum Nutzen steht. I. d. R. wird eine Schaltung bei der Synthese auf die Standardzellen aus der jeweiligen Bibliothek abgebildet.

Die Standardzelleninstanzen werden in der Designphase zuerst auf der Chipfläche platziert – fast ausschließlich aneinandergereiht in Zeilen oder Spalten (vgl. Abbildung 2.4). Leiterbahnen, welche die Zellen mit Spannung versorgen, sind in das Layout der Standardzellen integriert. Sie verlaufen als Potentialschienen an den gegenüberliegenden Kanten der Zellen. In Abbildung 2.3 führen die breiten horizontal verlaufenden Leiterbahnen die festen Potentiale für V_{SS} (Masse) und V_{DD} und liefern die Versorgungsspannung für die platzierten Zellen.

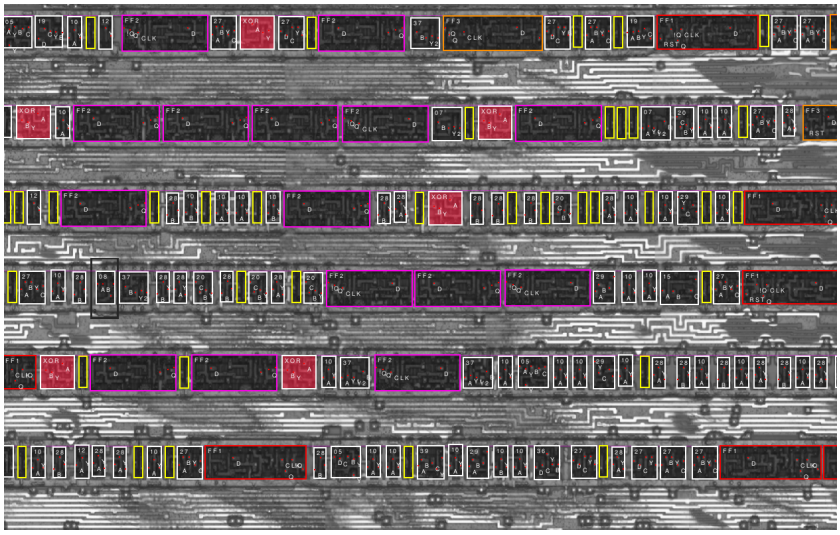

Abbildung 2.4: In Zeilen angeordnete Standardzelleninstanzen

Steht die Position der Gatter fest, können bei der Synthese die Leiterbahnen verlegt werden. Wenn zwischen den platzierten Standardzellen Freiräume gelassen wurden, ist eine Leiterbahnenführung durch diese Räume möglich. Da Leiterbahnen kreuzungsfrei verlegt werden müssen, sind dafür fast immer mehrere Ebenen eines Chips notwendig. Vergleichbar mit dem Layout mehrschichtiger Platinen, gibt es pro Ebene häufig eine Hauptrichtung für Leiterbahnen. Als Nomenklatur für leiterbahnführende Ebenen eines Chips hat sich die Bezeichnung Mi etabliert, wobei i die Nummer der Chipebene bezeichnet und M angibt, dass die Ebene durch metallische Elemente gebildet wird. Entsprechend wird der erste Layer, der die Transistoren verschaltet als M1 bezeichnet.

Im Rahmen vorangeganger Reverse-Engineering-Prozesse haben sich Bezeichnungen für Ebenen von Chips ergeben, die keine strengen Fachbegriffe darstellen, jedoch aus Reverse-Engineering-Sicht sinnvoll sind und in der hier präsentierten Software verschiedentlich verwendet werden. Zum einen ist das der Begriff Transistorebene,

der aus der Perspektive der Herstellung das Substrat, das isolierende Halbleiteroxid und die Ebene des polykristallinen Siliziums umfasst. Zum anderen ist dies der Begriff Logikebene. Diese Ebene wird der Fachliteratur als M1 bezeichnet. Der Begriff Logikebene ist dadurch motiviert, dass diese Chipebene die Verdrahtung der Transistoren bildet, und dadurch erst die funktionale Zusammenfassung zu Standardzellen realisiert. Diese Ebene ist insbesondere als Eingabedatum für den Algorithmus geeignet, der Instanzen von Standarzellen findet (siehe Kapitel 7). Innerhalb der Software werden alle sonstigen leiterbahnführenden Ebenen M_i mit $i \geq 2$ als Metall-Layer bezeichnet.

Kapitel 3

Der Reverse-Engineering-Prozess

Dieses Kapitel beschreibt die für die Rekonstruktion digitaler Logik in integrierten Schaltkreisen notwendigen Schritte. In einer Studienarbeit wurden die technischen Hintergründe bereits dargestellt (siehe [Sch10]). Das Reverse-Engineering umfasst demnach folgende Schritte:

- Vorbereitungsphase
- Das Entkapseln von Chips
- Erstellen von Fotos der einzelnen Chipebenen
- Das Ermitteln von Standardzellenkandidaten
- Finden von Standardzellenplatzierungen
- Die Rekonstruktion der Standardzellenfunktion
- Nachvollziehen der Verschaltung
- ggf. die Visualisierung als Schaltplan
- Verifikation der Ergebnisse

3.1 Vorbereitungsphase

Die Vorbereitungsphase umfasst hauptsächlich Recherchen über den Chip, also das Sammeln und Auswerten von Datenblättern und sonstigen Informationen, welche einen Einblick in die Funktionsweise des Chips ermöglichen. Dabei ist zu unterschei-

den, ob ein konkreter Chip-Typ zu analysieren ist oder ein Anwendungsfall, etwa ein konkretes Micropayment-System, für welches eventuell verschiedene Chip-Realisierungen existieren.

Wenn die Analyse eines Anwendungsfalles im Vordergrund steht, bestünde die Möglichkeit, das Chip-Reverse-Engineering zu vermeiden. Da bei Angriffen zunächst die schwächsten Glieder evaluiert werden, könnte es z. B. sein, dass ein Firmware- oder Protokoll-Reverse-Engineering schneller zum Ziel führt. Es kann ebenfalls sein, dass verschiedene Hardwarerealisierungen für eine Anwendung verwendet werden. In diesem Fall ist es sinnvoll, eine Realisierung als Chip für einen Angriff auszuwählen, bei der die geringsten Reverse-Engineering-Hürden zu erwarten sind. Eine technische Hürde stellt die Strukturgröße der Halbleiterprozesse dar. Diese ist für die Kosten des Reverse-Engineering relevant, denn die Strukturgröße ist entscheidend dafür, ob noch ein Lichtmikroskop eingesetzt werden kann oder ob „teure“ abbildende Verfahren zum Einsatz kommen müssen.

Als Reverse-Engineering-Hürden sind Verfahren zu verstehen, die gezielt als Maßnahmen gegen das Reverse-Engineering implementiert werden. Dabei kann für ältere Chips angenommen werden, dass Sicherheitsmaßnahmen nicht vorhanden oder nicht besonders ausgearbeitet sind. Eine dieser Maßnahmen ist beispielsweise das Bus-Scrambling, bei dem die Bedeutung einzelner Leiterbahnen eines Busses durch ungewöhnliches Routing verschleiert wird. Eine andere Maßnahme wäre die Platzierung funktionsloser Zellen, die sich im Abbild nicht von echten Zellen unterscheiden, beim Reverse-Engineering jedoch irreführend sind.

Im Rahmen der Vorbereitung sollte ebenfalls geprüft werden, inwiefern öffentlich zugängliche Dokumentation zu einem Chip existiert. So geben Blockschaltbilder einen Überblick über schalttechnische Komponenten, die auf einem Chip zu finden sind. Dies hilft, die Funktionsweise eines Chips auf hoher Abstraktionsebene nachzuvollziehen. Für das Aufklären von Verschlüsselungsverfahren muss ein Chip nie komplett untersucht werden. Kenntnisse über den Aufbau eines Chips helfen beim Finden von relevanten Stellen, die später gezielt analysiert werden.

Das hier beschriebene Reverse-Engineering-Verfahren ist eine statische Analyse. Dies bedeutet, dass sich damit nur physisch vorhandene Schaltkreiskomponenten rekonstruieren lassen. Komplexere Schaltkreise benutzen häufig eingebettete Mikrokontroller, die zusammen mit der Firmware das Ansteuern von Komponenten übernehmen. Diese firmwaregestütze Ansteuerung lässt sich mit einer rein statischen Analyse der

Schaltkreise nicht nachvollziehen. Insbesondere für die Rekonstruktion von Verschlüsselungsverfahren zwecks Nachimplementierung kann dies jedoch relevant sein, da die Initialisierung i. d. R. durch Software gesteuert wird.

3.2 Entkapselung

Die Befreiung des Halbleiterplättchens aus dem Gehäusematerial ist der erste Schritt beim Reverse-Engineering. Ein gängiges Material für Chipgehäuse sind Epoxide, die mechanisch, chemisch und physikalisch stabil sind. In der Präparationstechnik werden zum chemischen Entkapseln überwiegend konzentrierte Säuren verwendet, z. B. Salpeter- und Schwefelsäure. Unter nicht laborkonformen Arbeitsbedingungen ist die Entkapselung mittels Abietinsäure geeignet, etwa in Form von Kolophonium, bei der die Epoxidanteile im Chipgehäuse infolge längeren Kochens zerstört werden.

Durch die Entkapselung erhält man den reinen „die“, der frei von Kunststoffen ist und an dem in der Regel keine Bonddrähte und sonstige externe Metallteile hängen. Das Halbleiterplättchen wird anschließend unter Hilfe von Aceton im Ultraschallbad gereinigt.

3.3 Bildgewinnung

Um die Schichten eines Chips auswerten zu können, müssen sie einzeln freigelegt werden. In der Halbleiterindustrie verwendet man dafür nasschemische Ätzverfahren, bei denen praktisch immer Flusssäure zum Einsatz kommt [Bec98, S. 32ff]. Alternativ kann man trockene Verfahren anwenden, z. B. eine Kombination aus Plasmaätzen und Polieren, da sich damit Material gezielter abtragen lässt. Das kostengünstigste Verfahren ist das rein mechanische Entfernen mittels Polieren der Chipoberfläche.

Die Chipebenen werden Schicht für Schicht unter dem Mikroskop begutachtet. Für die spätere Analyse werden Aufnahmen angefertigt. Wenn der Chip in einer Strukturgrößen über 200 nm gefertigt ist, kann man ein Lichtmikroskop mit Kameramodul verwenden (vgl. [Sko05, S. 80]). Für feinere Strukturen benötigt man hochauflösende Geräte, beispielsweise ein Rasterelektronenmikroskop. Eine Strukturgröße von 200 nm stellt kein festes Limit dar. So ist die Ebene M1 eines Chips aus einem 180 nm-Prozess beispielsweise noch gut zu erkennen. Jedoch bedarf es dann für die Abbildung darunterliegender Ebenen hochauflösender Geräte.

Die Chipoberfläche wird meistens stückweise abgebildet, so dass man für eine Ebene mehrere teilweise überlappende Bildkacheln erhält. Diese Bildkacheln werden mittels sogenannter Stitching-Software zu einem großen Bild zusammengesetzt. Ist das Bild in sich gedreht, so dass Leiterbahnen nicht genau waagerecht bzw. senkrecht verlaufen sollte die Rotation mit einem Grafikbearbeitungsprogramm korrigiert werden. Für die automatische Bildauswertung kann es vorteilhaft sein, das Bildmaterial in Bezug auf Helligkeit, Kontrast und Sättigung zu normalisieren. Alle gängigen Bildbearbeitungsprogramme bieten Funktionen an, um die Parameter dafür automatisch zu bestimmen und die Bilddaten per Knopfdruck anzupassen. Weitere Nachbearbeitungsschritte, beispielsweise Entrauschung, sind für die Analyse mittels der Software Degate nicht erforderlich.

Wenn mehrere Ebenen des Chips fotografisch aufgenommen werden, müssen die Bilder in Übereinstimmung gebracht werden. Der Skalierungsfaktor der Ebenen sollte durch die Mikroskopeinstellung gleich sein. Die Rotation wurde zuvor manuell korrigiert. Dadurch ist lediglich eine Translation nötig, die durch Bildbearbeitungsprogramme, welche zumeist mit mehreren Layern umgehen können, angewendet werden kann.

3.4 Ermitteln von Standardzellenkandidaten

In Kapitel 2 wurde beschrieben, dass gängige Schaltkreisdesigns auf Standardzellen basieren. Beim Reverse-Engineering kann man sich diese Eigenschaft zunutze machen. Zunächst ermittelt man das Layout von Standardzellen. Dazu ist die Schicht M1 geeignet, in welcher fast ausschliesslich die interne Verschaltung der Transistoren der Standardzellen realisiert ist. Diese Leiterbahnenmuster auf M1 sind meistens sehr spezifisch und bilden eine Art „Fingerabdruck“ (vgl. Abbildung 3.1).

Der Schritt der Kandidatenermittlung ließe sich algorithmisch behandeln. Der Aufwand für ein manuelles Ermitteln ist jedoch gering. Des Weiteren ist beim manuellen Ermitteln das Auftreten von Fehlern so gut wie ausgeschlossen, womit Fehlerfortpflanzung folglich unterbunden wird.

Man kann ein mehrfaches Auftreten markanter Merkmale an verschiedenen Stellen schnell erkennen und durch Vergleich maximal gleiche Regionen ermitteln. So ist beispielsweise die labyrinthartige Leiterbahnstruktur in Abbildung 3.1 an mehreren Stellen zu finden. Durch Vergleich der Muster ist ebenfalls deren Begrenzung feststell-

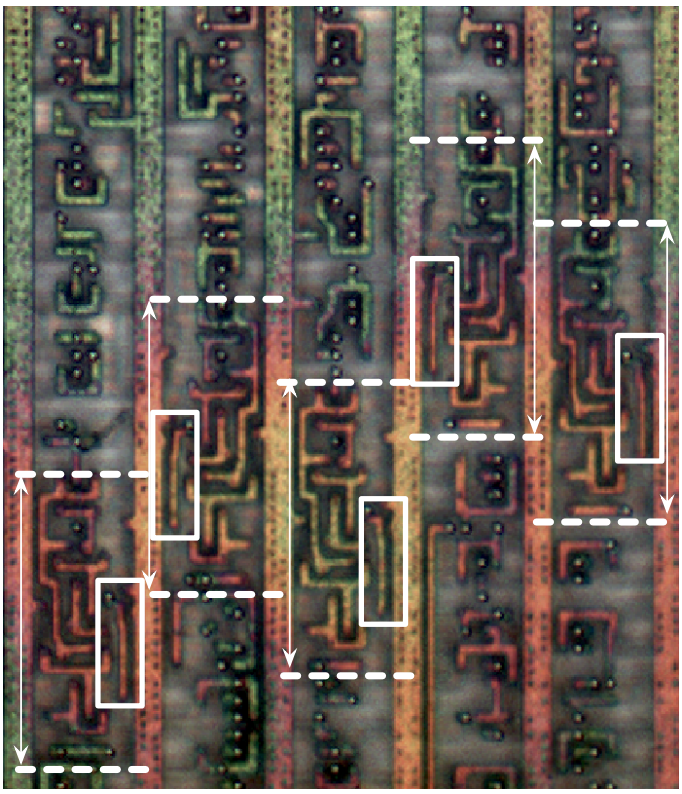

Abbildung 3.1: Konstante Bereiche sind Kandidaten für einen Standardzellentyp. Die Begrenzung ist durch die gestrichelten Linien dargestellt. Die Rechtecke markieren ein Element des Musters.

bar. Diese konstanten Bereiche sind sichere Kandidaten für eine Standardzelle. Sollte das Verdrahtungsmuster für eine Standardzelle auf dem Layer M1 keine markanten Merkmale aufweisen, etwa bei sehr kleinen Zellen, muss der Transistorlayer zum Vergleich herangezogen werden.

3.5 Finden von Standardzellenplatzierungen

Zuerst sollten große Kandidaten für Standardzellen identifiziert werden. In einem weiteren Schritt werden dann die Positionen der Standardzelleninstanzen ermittelt. Dieses Suchproblem und die zugrundeliegende Algorithmik wird in Kapitel 7 behandelt.

Das Ermitteln von Standardzellen und das Finden von Platzierungen wird iterativ so lange durchgeführt, bis keine freien Stellen auf der Chip-Oberfläche mehr vorhanden sind.

3.6 Ermitteln von Standardzellenfunktionen

Ausgehend von den Transistoren, deren Verdrahtung in der Ebene M1 und in wenigen Fällen teilweise in der Ebene M2, lässt sich die Schaltfunktion einer Standardzelle rekonstruieren. Dazu gehört, dass die festen Potentiale V_{DD} für Versorgungsspannung und V_{SS} für Masse identifiziert wurden. Dies lässt sich anhand der Gate-Längen der Transistoren bestimmen. Der Querschnitt unter dem Gateanschluss von Feldeffekttransistoren des p-Kanal-Typs ist bei Serienschaltungen größer dimensioniert als Gate-Längen der korrespondierenden n-Kanal-Transistoren, denn die Beweglichkeit der modellhaften Löcher ist geringer als die der Elektronen. Bei Parallelschaltungen von Transistoren werden Gate-Längen kleiner dimensionert (siehe Abbildung 3.2).

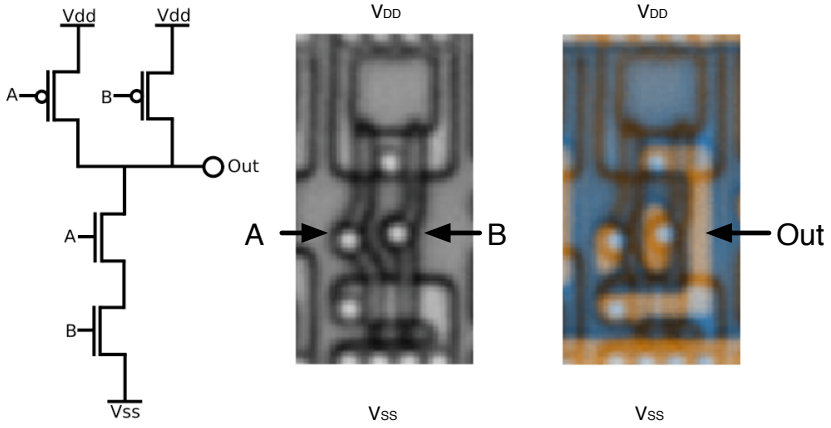

Abbildung 3.2: CMOS-NAND (Mifare Classic). Links ist der Schaltplan abgebildet. In der Mitte ist der Transistorlayer zu sehen. Rechts ist die Überlagerung des Transistorlayers mit der Verschaltung auf M1 dargestellt.

Die Eingänge einer Standardzelle sind stets Gate-Anschlüsse von Transistoren. Die Umkehrung, dass mit einer Schnittstelle einer Standardzelle verbundene Gate-Anschlüsse Eingänge sind, gilt nicht. Die verbleibenden Anschlüsse sind Ausgänge. Im Falle von Tristate-Logik sind Anschlüsse abhängig vom Schaltzustand auch galvanisch entkoppelt. Dies ist anhand der Schaltung nachvollziehbar.

Um die Schaltung in eine logische Formel zu überführen, kann man sich grundsätzlich die logischen Werte an den Ausgängen überlegen, wenn man alle möglichen Belegungen der Eingänge und gegebenfalls einen inneren Zustand (z. B. bei Flipflops) in Betracht zieht. Bei CMOS-Technologie ist das Ablesen der Schaltungsfunktion dadurch vereinfacht, dass man aufgrund der komplementären Verschaltung jeweils

nur das Pull-up- oder das Pull-down-Netzwerk betrachten muss. Dabei ist zu überlegen, unter welcher Bedingung ein Ausgang das HIGH-Signal führt. Beispielsweise entspricht der Ausgang des NANDs in Abbildung 3.2 dem Wert HIGH, wenn weder A noch B positives Potential führen.

Wenn das Layout einer Standardzelle einschließlich der Informationen über Potentiale und elektrische Verbindungen maschinenlesbar zur Verfügung stünde, könnte die Schaltfunktion auch automatisch ermittelt werden, z. B. über den Zwischenschritt einer Spice- oder Verilog-Simulation. Es wäre möglich, eine GUI-Anwendung zu entwickeln, mit der ein Standardzellenlayout ausgehend von Bilddaten mittels Nachzeichnen etwa in Verilog-Code überführt werden könnte. Das dadurch entstehende Modell einer Standardzelle ließe sich direkt für die Verhaltenssimulation benutzen. Da die Komplexität von Standardzellen zumeist sehr gering ist, bestand bisher kein konkreter Bedarf an dieser Entwicklung. Der Vorteil wäre jedoch, dass die Überführung in eine Verhaltensbeschreibung kaum Interpretationsspielraum eröffnet, damit (nahezu) fehlerfrei wäre und ohne Reverse-Engineering-Übung schnell umgesetzt werden könnte.

3.7 Verfolgen der Leiterbahnen

Sind die Standardzellentypen und deren Platzierung bekannt, können die Leiterbahnen verfolgt werden, welche die Ein- und Ausgänge miteinander verbinden. Die Leiterbahnen verlaufen in der Regel über mehrere Ebenen des Chips. Dabei sind die Leiterbahnen verschiedener Ebenen mittels Durchkontaktierungen elektrisch verbunden.

In vielen Fällen muss dieser Schritt manuell durchgeführt werden. Eine erhebliche Beschleunigung des Reverse-Engineering-Prozesses kann durch Automatisierung erzielt werden. Kapitel 8 behandelt diese Problemstellung.

3.8 Rekonstruktion des Schaltplans

Sind alle Schaltelemente und deren Verdrahtung bekannt, d. h. es gibt eine mehr oder weniger vollständige Netzliste, kann der Schaltplan extrahiert werden. Im Gegensatz zur Netzliste ist der Schaltplan dazu gedacht, von Menschen gelesen zu werden. Bei der Extraktion werden deshalb auch Transformationen durchgeführt, die von

Teilen der Netzliste abstrahieren. Es ist beispielsweise üblich, Komplexität dadurch zu verringern, dass Komponenten als Blockschaltbilder gezeichnet werden. In Kapitel 9 wird die Problematik behandelt.

3.9 Verifikation der Ergebnisse und Fehlererkennung

Es ist zu erwarten, dass beim Reverse-Engineering von Chips mit dem zuvor beschriebenen Verfahren Fehler in den Ergebnissen auftreten. Fehler sind grundsätzlich problematisch. Nur in seltenen Fällen wären Fehler tolerabel. Folglich bedarf es geeigneter Maßnahmen, um die Korrektheit von rekonstruierten Teilen zu prüfen. Diese Prüfung sollte in den verschiedenen Phasen des Reverse-Engineerings erfolgen, um Folgefehler zu vermeiden.

Beim vorwärtsgerichteten Entwurf von Schaltkreisen werden Komponenten einzeln getestet. Dabei ist durch Simulation prüfbar, ob eine Realisierung mit einem Modell für die jeweiligen Testfälle übereinstimmt. Beim Reverse-Engineering ist das Modell einzelner Komponenten bzw. der gesamten Schaltung zunächst unbekannt und deshalb nicht testbar. Im Einzelfall muss deshalb überlegt werden, welche Möglichkeiten es gibt, das Ergebnis einer Schaltkreisrekonstruktion zu prüfen.

Im Fall von Verschlüsselungsverfahren ist es sinnvoll, Beispiele für Klartextnachrichten und korrespondierende chiffrierte Daten bei bekanntem Initialisierungsvektor zu erlangen. Man kann damit testen, ob der ermittelte Algorithmus den gleichen Chiffretext produziert. Dies entspricht dem üblichen Vorgehen, einen kryptografischen Algorithmus auf korrekte Implementation zu testen, indem geeignete Testvektoren auf ihn anwendet werden und das Resultat mit bekannten Ergebnissen verglichen wird.

Ein Highlevel-Test ist unverzichtbar, um die Existenz von Fehlern im rekonstruierten Schaltkreis nachzuweisen. Dieser Test hilft jedoch nicht, Fehler auszuschließen bzw. Fehler zu lokalisieren. Aufgrund der Komplexität von Schaltkreisen würde ein Highlevel-Test ohne Prüfung von Zwischenergebnissen wahrscheinlich fehlschlagen. Die Prüfung von Zwischenergebnissen der einzelnen Reverse-Engineering-Phasen ist daher notwendig. Da die einzelnen Phasen des Reverse-Engineerings aufeinander aufbauen und auf Zwischenergebnisse in späteren Phasen zurückgegriffen wird, werden Fehler in der Regel gefunden.

Die Tabelle 3.1 gibt einen Überblick über einzelne Arbeitsschritte, die Wahrschein-

Arbeitsschritt	Fehler-wahrscheinlichkeit	Erkennungs-wahrscheinlichkeit
Manuelle Standardzellenidentifizierung	gering	hoch
Manuelle Funktionsrekonstruktion von Standardzellen (kombinatorische Logik)	gering	hoch
Manuelle Funktionsrekonstruktion von Standardzellen (sequentielle Logik)	mittel	mittel
Finden von Standardzellenplatzierungen	gering-mittel	hoch
Manuelles Nachvollziehen von Leiterbahnen	gering	hoch
Automatisches Nachvollziehen von Leiterbahnen	hoch	mittel

Tabelle 3.1: Arbeitsschritte und Fehlerwahrscheinlichekiten

lichkeit, dass bei dem jeweiligen Arbeitsschritt Fehler auftreten und die Wahrscheinlichkeit, dass derartige Fehler erkannt werden. Diese Erkennungswahrscheinlichkeit bedeutet ein zeitnahes Entdecken und Lokalisieren des Fehlers, nicht (nur) eine Existenzfeststellung. Ansätze und konkrete Maßnahmen für die Erkennung und Behandlung von Fehlern werden in den jeweiligen Abschnitten behandelt.

Viele Fehler können mittels Peer-Review entdeckt werden. Dies ist ein gängiges Verfahren und wird ebenfalls im kommerziellen Reverse-Engineering-Bereich, z. B. für die Rekonstruktion von Netzlisten, eingesetzt. Dabei werden die Ergebnisse zweier Teams miteinander verglichen, um Unstimmigkeiten zu finden. Ein auf Peer-Review basierender Ansatz lässt sich grundsätzlich auf alle Phasen des Reverse-Engineerings anwenden.

Bei der manuellen Auswertung gibt es Situationen, in denen beispielsweise der Verlauf von Leiterbahnen nicht eindeutig nachzuvollziehen ist. Diese Stellen sind als potentielles Problem ersichtlich und können markiert werden. Wenn bei der weiteren Analyse Fehler auftreten, können diese potentiell problematischen Stellen zur Einschränkung möglicher Lösungen herangezogen werden.

Schwieriger ist die Erkennung von Fehlern bei einer überwiegend automatisierten Bildauswertung. Einige Fehler lassen sich manuell erkennen. Dies wird beispielsweise

in Abschnitt 7.4 für Standardzellen beschrieben. Fehlerhaft erkannte oder nicht erkannte Leiterbahnen und Durchkontaktierungen können mittels Rule Checks aufgespürt werden. Anhang A.6 widmet sich dieser Problematik.

Kapitel 4

Projektidee

Im vorangegangen Kapitel wurden die Schritte des Reverse-Engineerings beschrieben. Demnach ist die Rekonstruktion von Schaltungsteilen in hohem Maße ein manueller Prozess, der von der automatisierten Lösung einiger Aufgaben erheblich profitiert. So sind diverse Suchprobleme sinnvollerweise algorithmisch zu behandeln.

Vor etwa zehn bis zwanzig Jahren war es eine gängige Methode, Fotos der Chipoberfläche aufzunehmen, aneinanderzukleben und anhand dieser Fotos Schaltpläne nachzuzeichnen [Tor09]. Mit fortschreitender Entwicklung der Technik, insbesondere der Speicherkapazitäten, wurde es zunehmend möglich, das Bildmaterial im Computer zu verwalten. Nach Angaben eines Chipkarten-Hackers[1], war es vor ca. zehn Jahren bei einem Unternehmen[2], welches die Sicherheit moderner PayTV-Systeme „evaluierte“, Stand der Technik, das Bildmaterial von Chips mit der Bildbearbeitungssoftware Photoshop zu verwalten und mit dieser ebenfalls Schaltkreisrekonstruktionen durchzuführen.

Bei dem Ende 2007 veröffentlichen Ergebnissen des Reverse-Engineering vom Verschlüsselungsverfahrens Crypto 1 (Mifare Classic) diente die Bildbearbeitungssoftware Gimp zur Navigation durch mehrschichtiges Bildmaterial, wobei Metadaten, etwa Verbindungsinformationen separat mit einem Spreadsheet-Programm verwaltet wurden und Matlab-Skripte beim Wiederfinden von Standardzellen halfen [KNP08] [Noh+08]. Jan Krißler, einer der Beteiligten am Mifare-Hack, erklärte in einem Gespräch, dass „diese manuelle Reverse-Engineering-Methode funktionierte, weil die

[1] Der Name ist dem Autor bekannt.
[2] ebenfalls bekannt

Komplexität des Mifare-Classic-Chips gerade noch handhabbar war. Wäre der Chip sehr viel komplexer gewesen, hätte das Projekt kein Ende gefunden.“ Die Idee für die Software Degate ergab sich aus diesem Projekt.

4.1 Benutzungskonzept

Ein Bildbearbeitungsprogramm ist zwar für die Betrachtung von (mehrschichtigen) Bildern geeignet. Es stößt jedoch an seine Grenzen, wenn es um den Umgang mit Objekten geht. Dafür wären allgemein Vektorgrafikprogramme oder spezieller CAD-Programme geeignet. Diese sind wiederum für den Umgang mit großen pixelbasierten Bildmengen ungeeignet und wenig domänenspezifisch.

Die Software Degate schließt diese Lücke. Ausgehend von den Erfahrungen des Reverse-Engineering von Crypto 1 entstand der Wunsch, anfallende Metainformationen in einer Software zu verwalten. Im Kern ist die Software Degate ein Grafikbetrachtungsprogramm für mehrschichtiges Bildmaterial, ergänzt um Datenmodelle und Funktionen, um mit den Bildobjekten zu arbeiten. Ferner stehen Bilderkennungsfunktionen zur Verfügung, um Bildmaterial automatisiert auszuwerten.

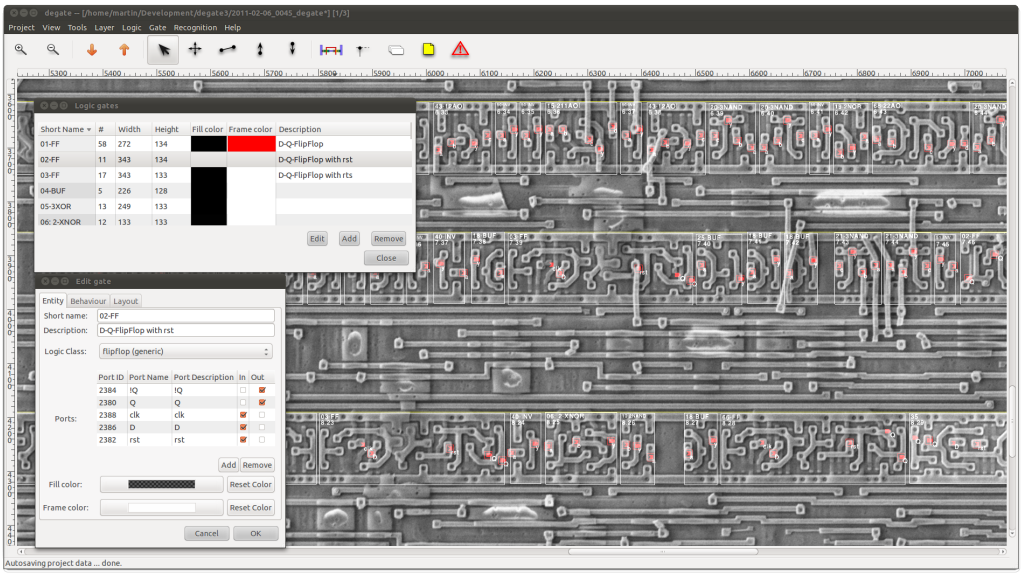

Abbildung 4.1: Bildschirmfoto der Software Degate.

Der Schwerpunkt bei der Entwicklung von Degate liegt demnach nicht darin, ein eigenes Grafikbearbeitungsprogramm oder ein CAD-Programm nachzuprogrammie-

ren. Vielmehr steht das Abbilden von Reverse-Engineering-Schritten im Computer unter Zuhilfenahme einiger Ideen von Grafik- bzw. CAD-Programmen im Vordergrund. Funktionen zum Springen zwischen den einzelnen Ebenen, zum Zoomen und Verschieben des sichtbaren Bereichs dienen der Navigation. Manuelles Einzeichnen von Standardzellen, Durchkontaktierungen und Leiterbahnen sind Elemente der Interaktion. Für einen Überblick über die Nutzung der Software sei auf Anhang A verwiesen.

4.2 Machbarkeitsstudie

Nach dem Reverse-Engineering von Crypto 1 wurde ein Prototyp entwickelt, der Kernfunktionen bereits enthielt. Diese Machbarkeitsstudie zeigte, dass mit dem Prototypen Schaltfunktionen einfacher zu rekonstruieren sind, denn Metainformationen mussten nun nicht mehr separat erfasst und verwaltet werden. Vielmehr entstanden sie beim Arbeiten mit dem Bildmaterial und lagen somit für weitere Bearbeitungsschritte direkt vor.

Bei der Entwicklung des Prototypen gab es keine formalen Anforderungen. Die Arbeitsschritte des Reverse-Engineerings waren im Wesentlichen klar. Somit gab es naheliegende Ideen, welche einzelnen Schritte in einer Software abzubilden wären. Allerdings waren mögliche Lösungen für Detailprobleme nicht erprobt. Es gab kein bekanntes und zugängliches Programm, dass zum Reverse-Engineering von Chips geeignet wäre, dass als Vorbild zum Nachprogrammieren hätte dienen können.

Der Prototyp entstand hauptsächlich aus einer spontanen Motivation heraus, Teile des Reverse-Engineerings zu automatisieren und Metadaten zu verwalten, damit dies nicht in Spreadsheet-Programmen erfolgen muss. Dabei gab es zunächst keinen Anspruch, den gesamten Rekonstruktionsprozess softwaregestützt zu begleiten.

An vielen Stellen war der Prototyp vom Design her nahe am zu lösenden Problem orientiert. Sämtliche Datenstrukturen des Prototypens wurden als C-Strukturen modelliert (z. B. Durchkontaktierungen und Leiterbahnen), d. h. es wurden gruppierte Datenfelder ohne Kapselung verwendet. Mit fortschreitender Entwicklung wurden diese Strukturen komplexer, so dass der Einbau von Erweiterungen Änderungen an vielen Stellen im Programmcode erforderte, denn diese Strukturen wurden als untypisierter Zeiger an Funktionen übergeben. Es war Aufgabe der jeweiligen Funktion zu erkennen, welchen Typ der Zeiger repräsentiert und entsprechend mit den

Daten zu operieren. Zwar ist es auch in der Sprache C möglich, Objekte sauber zu modellieren. In dem Prototypen war das Softwaredesign jedoch nicht dafür ausgelegt, die zunehmende Komplexität mittels Abstraktion zu bewältigen. Folglich sprachen Softwaredesigngründe für eine Abkehr vom Prototypen. Andererseits war der Prototyp aus Benutzerperspektive geeignet, um konkrete Reverse-Engineering-Projekte umzusetzen. Dadurch ließ sich die Benutzbarkeit von Ideen praktisch testen.

Neben den Softwaredesignproblemen zeigten sich folgende Schwachpunkte des Prototypen: Die Bilddaten für eine einzelne Ebene des Chips wurden mittels der Betriebssystemfunktion `mmap()` in den Speicher eingeblendet. Dabei waren die unkomprimierten Bilddaten in einer einzelnen Datei organisiert. Auf Rechnersystemen mit einem Adressraum der Größe 32 Bit führte dies zu Engpässen bei der maximalen Menge linear adressierbaren Speichers. Dadurch konnten lediglich Bilder mit einer Kantenlänge von maximal 20.000 Pixel verarbeitet werden. Als ebenfalls problematisch erwies sich die Entscheidung, extrahierte Daten (Gatter, Netzlisten, Leiterbahnen, etc.) in einem ASN.1-spezifizierten Dateiformat BER-kodiert zu persistieren. Zwar ermöglichte dies ein kompaktes und plattformunabhängiges Speicherformat, jedoch sind derart kodierte Daten selbst nach einer Aufbereitung kaum menschenlesbar. In der Testphase trat ein Fall ein, in dem die Daten Inkonsistenzen aufwiesen. Diese konnte mit vertretbarem Aufwand nicht repariert werden, so dass auf eine Sicherungskopie zurückgegriffen werden musste.

Im Rahmen der Diplomarbeit wurden die Codebasis überarbeitet. Sämtliche grundlegenden Funktionen wurden als Bibliothek `libdegate` in C++ neu entworfen und implementiert. Der Programmcode für die interaktive grafische Oberfläche wurde in wesentlichen Teilen überarbeitet.

4.3 Plattform und Bibliotheken

Sowohl für den Prototypen als auch für die neu geschriebene Version von Degate gab es keine Vorgaben hinsichtlich Programmiersprache und GUI-Umgebung. Aus praktischen Erwägungen lag die Entwicklung für unixartige Betriebssysteme nahe. Für die GUI-Bibliotheken bedeutet dies im Wesentlichen eine Eingrenzung auf GTK+ bzw. Qt, also jene Frameworks, welche die Grundlage für die Desktopumgebungen Gnome und KDE darstellen. Bereits für den Prototypen wurde die auf C++ basie-

rende Abstraktion gtkmm[3] verwendet. Diese Bibliothek ist einfacher zu verwenden als GTK+, denn GUI-Widgets sind ein klassisches Problemfeld der Objektorientierung. Die Option zur Portierung der Software auf das Betriebssystem Windows sollte grundsätzlich erhalten werden, wenngleich die Portierung kein konkretes Ziel ist. Windows-Portierungen der Bibliotheken GTK+ und gtkmm existieren.

Vor dem Nachprogrammieren des Prototypen wurde geprüft, welche Bibliotheken und Frameworks genutzt werden können, um den Aufwand der Programmierung zu verringern. Insbesondere wurde recherchiert, welche CAD-Frameworks bzw. Canvas-Bibliotheken geeignet wären. Auch wenn es CAD-Software auf der Basis von GTK bzw. QT gibt, schien keines der gängigen Programme geeignet zu sein, um die GUI-Funktionalität mit vertretbarem Aufwand zu extrahieren, um sie dann in eigener Software wiederzuverwenden. Zeitweise wurde die Bibliothek Goocanvasmm[4] evaluiert. Goocanvasmm ist eine C++-Abstraktion für die Bibliothek GooCanvas[5]. Es stellte sich jedoch heraus, dass deren API an vielen Stellen zu unflexibel ist. Mangels geeigneter Alternativen fiel die Entscheidung letztendlich zugunsten einer OpenGL-basierten Selbstentwicklung aus.

Ähnliche Überlegungen ergaben sich für die Verwaltung des Bildmaterials und deren computergestützter Auswertung. Dafür könnten Bibliotheken benutzt werden, die sowohl mit beliebig großen Bilddatenmengen umgehen könnten als auch Funktionen zur Bildverarbeitung bereitstellen. Vom Funktionsumfang mächtige Bibliotheken wie Open Computer Vision Library[6] (OpenCV) oder das Insight Segmentation and Registration Toolkit[7] (ITK) können nicht mit beliebig großen Datenmengen umgehen. Für ITK ist dies in einer zukünftigen Version geplant. ITK implementiert sehr viele Bildverarbeitungsfunktionen und ist für Open-Source-Software vorbildlich dokumentiert. Die Bibliothek Libvips[8] kann große Bilddatenmengen verarbeiten und stellt allgemeine Funktionen zur Verfügung, die für Signalverarbeitungszwecke geeignet sind.

Aus folgenden Gründen wurde in die Software Degate ein eigenes Bildformat und eine eigene API für die Behandlung von Bilddaten implementiert: Es handelt sich zunächst überwiegend um einen Container für Bilddaten und einige Primitiven

[3] C++ Interfaces for GTK+ and GNOME, `http://www.gtkmm.org/`, besucht am 24. Februar 2011
[4] `http://developer.gnome.org/goocanvasmm/`, besucht am 24. Februar 2011
[5] `http://live.gnome.org/GooCanvas`, besucht am 24. Februar 2011
[6] `http://sourceforge.net/projects/opencvlibrary/`, besucht am 20. Februar 2011
[7] `http://www.itk.org/`, besucht am 20. Februar 2011
[8] `http://www.vips.ecs.soton.ac.uk/index.php?title=Libvips`, besucht am 20. Februar 2011

für den Zugriff. Der Implementierungsaufwand dafür ist gering und es besteht die Möglichkeit, die API nach eigenen Wünschen zu gestalten. Leistungsfähigkeit ist im Wesentlichen nur für das Rendern der Bilddaten relevant. Da dies bei OpenGL mit Hardwarebeschleunigung in der Grafikkarte erfolgt, beschränkt sich der Bedarf an Leistungsfähigkeit auf einen schnellen Zugriff, um Bildteile blockweise in den Textur-Speicher der Grafikkarte zu laden.

Degates Zugriffsprimitiven für Bilddaten werden von „komplexeren“ Verarbeitungsalgorithmen verwendet, hauptsächlich Faltungsoperationen, Bildskalierung, Normalisierung und Farbraumkonvertierungen. Viele Bildverarbeitungsschritte sind nicht zeitkritisch und bedürfen daher keiner speziellen Optimierung. Dies bedeutet nicht, das diese Funktionen langsam wären. Es besteht in vielen Fällen lediglich keine Notwendigkeit für spezielle Optimierungen, insbesondere dann nicht, wenn nachfolgende Bildanalyseoperationen mehrere Minuten dauern.

Um den Entwicklungsaufwand für komplexere Signalverarbeitungsalgorithmen zu vermeiden, besteht trotz eigenes Speicherformats die Möglichkeit, Bilddaten in andere Bildformate bzw. Datentypen zu transformieren und eine der oben genannten Bibliotheken zu verwenden. Der Mehraufwand beschränkt sich dafür letztendlich auf eine Kopieroperation, die moderne Computer performant ausführen.

Für sonstige Zwecke, wo sich die Nutzung von Bibliotheken anbietet, werden sie genutzt. Beispielsweise wird die Bibliothek libxml++[9] für das Lesen und Schreiben von XML-Dateien verwendet. Um Bilder zu importieren bzw. zu exportieren werden die Bibliotheken libtiff[10] und libjpeg[11] genutzt. Im Prototypen fanden die Bibliotheken des Universalwerkzeugs ImageMagick[12] Verwendung, um Bilder in beliebigen Formaten im- bzw. exportieren zu können. Allerdings stellte dies für Benutzer der Software regelmäßig eine Hürde beim Kompilieren und Linken des Programmcodes dar, denn die dafür zu installierenden Pakete sind auf verschiedenen Betriebssystemen bzw. Systemversionen unterschiedlich benannt. Des Weiteren besteht keine Notwendigkeit eine Vielzahl von Grafikformaten zu unterstützen, wenn in der Praxis nur ein oder zwei Formate verwendet werden.

[9] `http://libxmlplusplus.sourceforge.net/`, besucht am 20. Februar 2011
[10] LibTIFF – TIFF Library and Utilities, `http://www.libtiff.org/`, besucht am 20. Februar 2011
[11] Independent JPEG Group, `http://www.ijg.org/`, besucht am 20. Februar 2011
[12] ImageMagick: Convert, Edit, Or Compose Bitmap Images, `http://www.imagemagick.org/`, besucht am 20. Februar 2011

Zum Archivieren von Projektdaten wird die Bibliothek libzip[13] benutzt. Es hat sich als zweckmäßig erwiesen, Exporte von Reverse-Engineering-Projekten mit eindeutigen Verzeichnis- und Archivnamen zu versehen und in einer einzelnen Datei abzulegen. Daten werden regelmäßig zwischen Projektbeteiligten ausgetauscht und zu Backupzwecken gespeichert. Projektdaten können zwar mittels Kommandozeile in ein Archiv überführt werden, einen niederschwelligen Zugang zu dieser Funktion aus Degate heraus anzubieten, führt zu klaren Dateibenennungen und damit zu mehr Übersicht beim Arbeiten mit verschiedenen Versionen eines Degate-Projektes.

Für die Nutzung von OpenGL dient die Bibliothek gtkglextmm als Schnittstelle. Die Darstellung von Text ist in OpenGL nicht spezifiziert. In Degate werden Glyphen deshalb mittels der Bibliothek Freetype in Bitmapdaten gerastert und als Texturen angezeigt.

[13] C library for reading, creating, and modifying zip archives, `http://www.nih.at/libzip/`, besucht am 20. Februar 2011

Kapitel 5

Logik-Modell

In der Software Degate wird als Logikmodell die Gesamtheit aller Objekte und deren Beziehungen, die schaltungstechnische Elemente des Chips abbilden, bezeichnet. In erster Linie gehören dazu Standardzelleninstanzen, Leiterbahnen und Durchkontaktierungen, die real auf einem Chip vorhanden sind. Ferner gibt es Hilfskonstrukte, die ebenfalls zum Logikmodell gehören. Beispielsweise sind dies Netze, mit denen elektrische Verbindungen in der Software modelliert werden, und Standardzellentypen. Darüber hinaus gehören zum Logikmodell Objekte, die in der realen Welt keine direkte oder indirekte Entsprechung haben, jedoch Informationen darstellen, die beim Reverse-Engineering auftreten. Dazu zählen beispielsweise Annotationen. Bilddaten sind zwar mit dem Logikmodell verknüpft, gehören jedoch nicht dazu.

Degate trifft für die Speicherung des Logikmodells die Annahme, dass der Hauptspeicher groß genug ist, um alle Objekte des Logikmodells darin vorzuhalten. Diese Annahme ist für alle realistischen Reverse-Engineering-Vorhaben statthaft. Bei bisherigen RE-Projekten mit der Software Degate umfasste das Logikmodell weniger als 10.000 Objekte. Die geschätzte obere Schranke für deren gesamten Speicherbedarf beträgt 10 MB RAM.

5.1 Überblick über zentrale Entitäten

Wire, Via, GatePort sind C++-Klassen, die in der libdegate implementiert sind. Deren reale Entsprechungen auf dem Chip haben die Eigenschaft, dass sie elektrisch verbindbar sind. In der libdegate wird dies modelliert, in dem diese Klassen von

ConnectedLogicModelObject erben (vgl. Abbildung 5.1). Dadurch erhalten sie die Fähigkeit, zu einem Netz zu gehören.

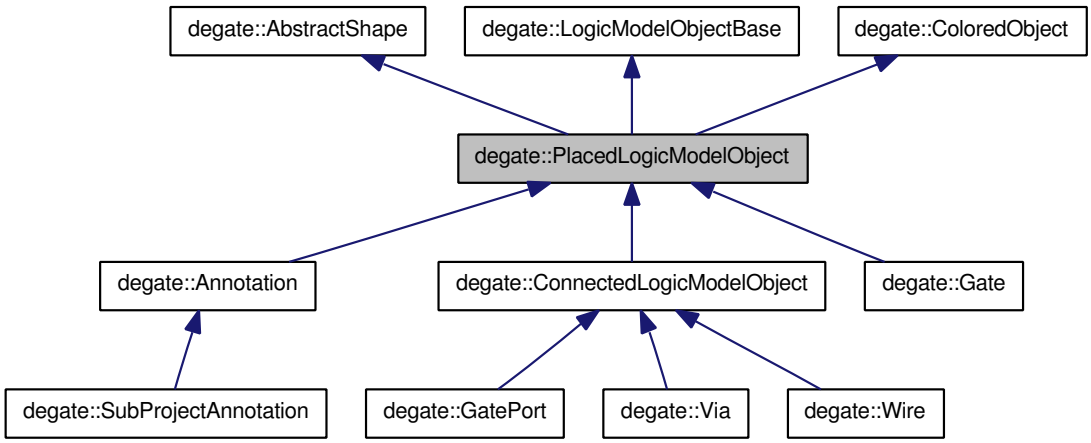

Abbildung 5.1: Vererbungsrelationen der Klasse PlacedLogicModelObject

ConnectedLogicModelObject ist eine Spezialisierung von PlacedLogicModelObject. Mit dieser Klasse wird modelliert, dass Objekte des Logikmodells in einer zweidimensionalen Ebene verortet werden können.

Für das zweidimensionale Indexieren implementiert libdegate eine Baumstruktur, die einen modifizierten Quadtree darstellt. Das Verfahren wird in Abschnitt 5.5 beschrieben. Damit ein Objekt mittels des Quadtrees indexiert werden kann, muss das Objekt sein minimal umgebendes Rechteck (engl. „bounding box“) angeben können. Zum Lesen des minimal umgebenden Rechtecks sind in AbstractShape Methoden festgelegt, die in den Klassen für geometrische Objekte, d. h. Kreise, Linien und Rechtecke, implementiert werden. Die Klasse PlacedLogicModelObject erbt von AbstractShape, damit platzierbare Logikmodellobjekte ebenfalls ihr minimal umgebendes Rechteck angeben können.

Die Klasse LogicModelObjectBase stellt Eigenschaften und Methoden bereit, die alle Logikmodellobjekte haben sollen. Derzeit sind dies Namen, Beschreibungen und IDs. Zudem sollen platzierbare Objekte farblich dargestellt werden. Dazu erbt PlacedLogicModelObject von ColoredObject, so dass Farben für Rahmen und Hintergrundfarbe festlegbar sind.

Abbildung 5.2 zeigt exemplarisch die Vererbungshierachie aus der Sicht der Klasse Wire. Eine Leiterbahn in Degate ist eine Linie. Wire erbt deshalb von der Klasse Line. Die Klasse Line stellt Methoden und Eigenschaften bereit, um die geometrische Infor-

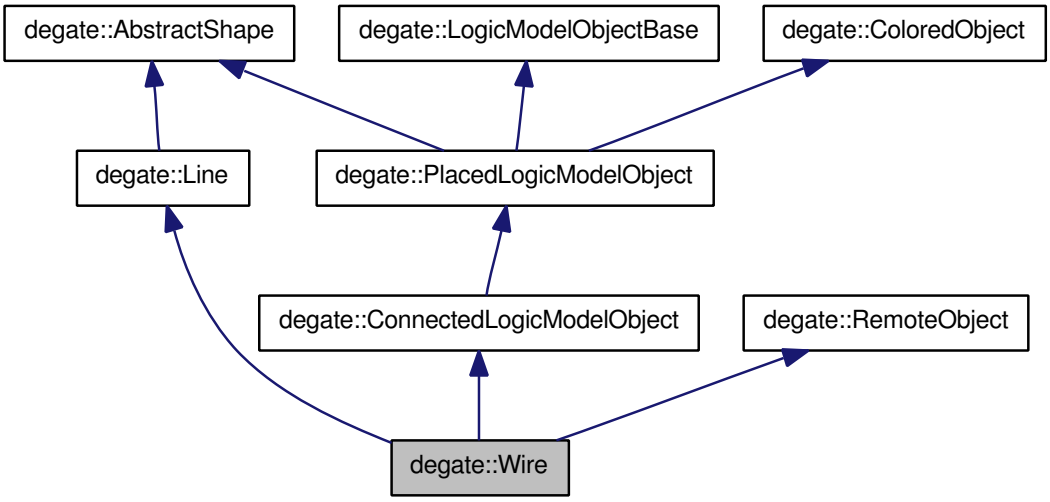

Abbildung 5.2: Vererbungsgraph für die Klasse Wire

mationen zu speichern und Zugriff darauf zu ermöglichen. Durchkontaktierungen und Standardzellenanschlüsse haben eine runde Form und erben entsprechend von der Klasse Circle. Hingegen erben Objekte mit rechteckiger Form, etwa Standardzellen und Annotationen, von Rectangle. Annotationen und Standardzellen gehören zu den nicht elektrisch verbindbaren Logikmodellobjekten. Sie erben daher nicht von ConnectedLogicModelObject, sondern von PlacedLogicModelObject.

Mittels der Klasse RemoteObject können Funktionen geerbt werden, mit denen ein Datenabgleich von lokalen und entfernten Objekten auf dem Server möglich wird.

5.2 Standardzellen und Ports

Die Klassennamen GateTemplate und GateTemplatePort sind historisch bedingte Bezeichner für Standardzellentypen und Anschlusstypen. Im Unterschied zu einer platzierten Standardzelle (Klasse Gate), beschreibt ein Standardzellentyp allgemeine Informationen, etwa Funktionsbeschreibungen in VHDL und Verilog, welche Anschlüsse allgemein zur Verfügung stehen (Klasse GateTemplatePort) und speichert Bildinformationen, die zum Finden von Standardzelleninstanzen genutzt werden.

Platzierte Standardzellen speichern, welchen Typs sie sind. Softwaretechnisch ist dies mit Shared Pointern auf einen Standardzellentyp umgesetzt. Jede platzierte Standardzelle verfügt über eine Menge von platzierten Anschlüssen (vgl. Abbildung 5.3).

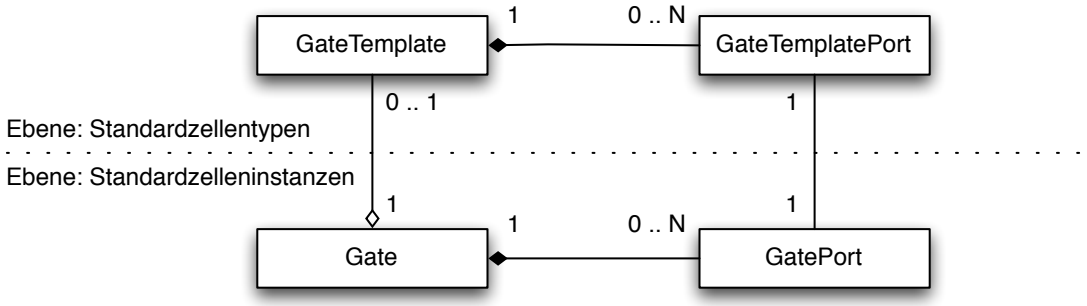

Abbildung 5.3: Beziehungsdiagramm für Standardzellentypen und Standardzelleninstanzen sowie Porttypen und platzierten Ports

Entsprechend hat ein Standardzellentyp eine Menge von Porttypen. Platzierte Ports verweisen auf einen Porttypen. Der Hintergrund für diese Modellierung besteht darin, dass Porttypen nicht elektrisch verbindbar sind, d. h. sie erben nicht von ConnectedLogicModelObject, da sie lediglich allgemeine Eigenschaften beschreiben, die alle Ports des jeweiligen Typs teilen. Hingegen hat ein platzierter Port die Eigenschaft, zu einem Netz zu gehören. In der Logikmodellimplementation wird sichergestellt, dass für jeden Porttypen eines Standardzellentyps ein korrespondierenes Objekt der Klasse GatePort instanziert ist und in diesem ein Shared Pointer auf den Porttypen verweist.

5.3 Netze

Elektrische Verbindungen werden in Degate als Netze modelliert. Ein Netz ist ein Container für elektrisch verbindbare Objekte. D. h. alle Objekte aus diesem Container führen das gleiche Potential. Demnach gilt: Zwei Objekte sind elektrisch verbunden, genau dann wenn sie zum selben Netz gehören.

Objekte, die von Typ ConnectedLogicModelObject sind, speichern, zu welchem Netz sie gehören und tragen sich automatisch aus dem Netz aus, wenn ihr Destruktor aufgerufen wird. Um zwei Objekte zu verbinden, werden beide Netze zu einem neuen Netz verschmolzen und in beide Objekte eingetragen. Die Umkehroperation ist die Isolation. Dabei trägt sich das Objekt aus dem Netz-Objekt aus und entfernt seinen eigenen Verweis zum Netz.

5.4 Ebenen und die Klasse LogicModel

Da integrierte Schaltkreise aus mehreren Ebenen bestehen, werden diese in der libdegate als Klasse Layer ebenfalls modelliert. Jede Ebene verwaltet z. B. das Bildmaterial, das der Ebene zugewiesen ist bzw. den Quadtree, in dem alle Objekte eines Layers indexiert sind. Für den Zugriff auf gespeicherte Objekte reicht die Klasse Layer Iteratoren aus der Quadtree-Implementation durch.

Die Klasse LogicModel repräsentiert das gesamte Logikmodell, d. h. alle Ebenen, Objekte, die Standardzellenbibliothek und Netze. Die Klasse stellt Methoden bereit, um Logikmodellobjekte einzutragen bzw. zu löschen. Diese Operationen müssen an einer zentralen Stelle erfolgen, weil das Ein- und Austragen Aktualisierungen notwendig macht. Beim Eintragen von Objekten stellt das Logikmodell sicher, dass Objekte eine eindeutige ID haben bzw. generiert eine neue ID, falls keine festgelegt ist. Ferner stellt es sicher, dass für platzierte Standardzellen Ports erzeugt werden und diese Ports jeweils eine valide Objekt-ID tragen. Das Logikmodell indexiert Objekt-IDs, so dass beispielsweise ein Objektpointer aus dem Logikmodell anhand einer ID geladen werden kann. Die Logikmodell-API stellt Funktionen bereit, um über Objekte eines Typs zu iterieren, z. B. über alle Durchkontaktierungen aller Layer.

5.5 Zweidimensionales Indexieren von Objekten

Zur zweidimensionalen Indexierung von Objekten verwendet Degate eine modifizierte Quadtree-Datenstruktur. Ein Quadtree teilt eine Ebene in vier gleichgroße Teilbereiche: links oben, rechts oben, links unten und rechts unten. Diese Teilung ist rekursiv, d. h. dass jedes dieser vier Teile in weitere vier Bereiche unterteilt wird. Die Rekursion stoppt, wenn ein bestimmter Unterteilungsgrad erreicht ist. Diese Aufteilung erzeugt einen Baum, in dem geometrische Objekte enstpechend ihrer Position verortet werden können. Jede Aufteilung stellt einen Knoten im Baum dar.

In vielen Quadtree-Implementationen werden Daten in den Blatt-Knoten des Baumes referenziert. Die Implementation in Degate ist dahingehend modifiziert, dass Objekte nicht nur in Blättern, sondern in beliebigen Knoten eingehangen werden können. Die Entscheidung, auf welchem Knoten ein Objekt im Quadtree referenziert wird, hängt vom minimal umgebenden Rechteck des zu speichernden Objektes ab. Ein Objekt wird in dem Knoten gespeichert, dessen Umgebungsrechteck das kleinste ist, in dem

das minimal umgebende Rechteck des Objektes Platz findet. Objekte mit großer Ausdehnung werden näher zur Wurzel des Baumes hin gespeichert. Entsprechend werden kleinere Objekte in tieferen Schichten referenziert. Mittels eines Iterators sind Bereichsanfragen möglich. Mit diesen kann man über alle Objekte iterieren, deren minimal umgebendes Rechteck mit dem Anfrageausschnitt überlappt.

5.6 Speicherformat des Logik-Modells und der Gatterbibliothek

Eine der Erfahrungen aus der Prototypenphase besteht darin, dass die Lesbarkeit der gespeicherten Logikmodelldaten wichtiger als eine speicherplatzsparende Repräsentation ist. Dies ist z. B. sinnvoll, um Fehler in der Software, insbesondere im Logikmodell, zu finden. Für die Speicherung von Logikmodelldaten wird daher XML verwendet. Die Verwendung von XML hat ferner den Vorteil, dass das Dateiformat mit wenig Programmcode von externen Werkzeugen gelesen und verarbeitet werden kann. Beispielsweise gibt es derzeit ein Perl-Skript, mit dem ein gespeichertes Logikmodell mit einem Referenzmodell verglichen werden kann, um Erkennungsraten und Fehlerkennungsraten zu bestimmen (vgl. Abschnitt 7.3). Es existiert ein weiteres Perl-Skript, mit dem eine Übersicht über Standardzellentypen im HTML-Format generiert werden kann. Darüber hinaus bilden die XML-Daten eine Schnittstelle, um Werkzeuge für die Analyse von Netzlisten bzw. deren grafischer Darstellung anzubinden.

In Degate werden Projektdaten, Logikmodelldaten und Standardzellenbibliotheken in separate Dateien gespeichert. Die Standardzellentypen werden in der Datei *gate_library.xml* hinterlegt. Diese ist exemplarisch in Listing 5.1 dargestellt. Unterhalb des XML-Tags „gate“ wird jeweils ein Standardzellentyp beschrieben. Zu beachten ist, dass der Wert des XML-Attributes „type-id“ die ID des Standardzellentyps angibt, nicht die ID einer platzierten Standardzelle. Entsprechend geben die Attribute „id“ des XML-Tags „port“ die IDs des Schnittstellentypes an und nicht die IDs der Ports von platzierten Zellen.

Listing 5.1: Beispielinhalt einer gespeicherten Standardzellenbibliothek

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
  <gate-library>
3   <gate-templates>
```

```
5     <gate type-id="2343" name="NOR4" description="27" logic-class="NOR"
          fill-color="#7FA0A0A0" frame-color="#FFFFB300"
7         width="46" height="70">

9       <images>
          <image layer-type="logic" image="2343_logic.tif"/>
11        <image layer-type="transistor" image="2343_transistor.tif"/>
        </images>
13
        <ports>
15        <port id="2347" name="y" description="y" type="out" x="43" y="36"/>
          <port id="2351" name="c" description="c" type="in" x="19" y="34"/>
17        <port id="2345" name="b" description="b" type="in" x="11" y="35"/>
          <port id="2353" name="a" description="a" type="in" x="3" y="36"/>
19        <port id="2349" name="d" description="d" type="in" x="36" y="37"/>
        </ports>
21
        <implementations>
23        <implementation type="vhdl" file="2343.vhdl"/>
        </implementations>
25    </gate>

27  </gate-templates>
</gate-library>
```

Der Quelltext in 5.2 zeigt ein dazu passendes Beispiel eines gespeicherten Logikmodells einer *lmodel.xml*. Im Logikmodell werden sämtliche platzierten Objekte eingebunden, jeweils in eigenen Sektionen. Platzierte Standardzellen sind unterhalb von „gates“ zu finden. Im Beispielquelltext existiert eine platzierte Zelle mit der ID 2342, deren Typ 2343 ist, d. h. dem in der Standardzellenbibliothek hinterlegten NOR mit vier Eingängen. Die angegebenen Ports haben ebenfalls eigene IDs. Das XML-Attribut „type-id“ gibt auch hier wieder den Typ des Ports an. Für Annotationen, Leiterbahnen und Durchkontaktierungen gibt es eigene Sektionen in der XML-Datei. Exemplarisch ist lediglich eine Durchkontaktierung angegeben. Leiterbahnen und Annotationen sind im Beispiel nicht enthalten.

Listing 5.2: Beispielinhalt eines gespeicherten Logikmodells

```
<?xml version="1.0" encoding="ISO-8859-1"?>
2 <logic-model>

4   <gates>
      <gate id="2342" name="" description=""
6         layer="1" orientation="normal"
          min-x="1264" min-y="275" max-x="1310" max-y="345"
8         type-id="2343">
        <port id="2344" name="b" type-id="2345"/>
10      <port id="2346" name="y" type-id="2347"/>
        <port id="2348" name="d" type-id="2349"/>
```

```
12        <port id="2350" name="c" type-id="2351"/>
          <port id="2352" name="a" type-id="2353"/>
14      </gate>
    </gates>
16
    <vias>
18      <via id="2807" name="" description="" layer="2" diameter="4"
            x="14" y="164" fill-color="#00000000"
20          frame-color="#00000000" direction="up" remote-id="0"/>
    </vias>
22
    <wires/>
24  <annotations/>

26  <nets>
      <net id="2483">
28      <connection object-id="2811"/>
        <connection object-id="2813"/>
30      <connection object-id="2814"/>
      </net>
32  </nets>

34 </logic-model>
```

Das Beispiel enthält die Beschreibung eines Netzes. Netze haben, wie alle Objekte des Logikmodells, eine ID. Hier ist dies die ID 2483. Das Netz verbindet drei Objekte – die mit der angegebenen ID.

Kapitel 6

Verwaltung von Bilddaten

Degate trifft für die Speicherung von Daten die Annahmen, dass der Hauptspeicher groß genug ist, um alle Objekte des Logikmodells vorzuhalten. Hingegen sind Bilddaten regelmäßig in größerem Umfang vorhanden als RAM zur Verfügung steht. Folglich bedarf es geeigneter Stragegien, um mit großen Bilddatenmengen umzugehen. Große Bilddaten werden als Kacheln gespeichert. Dabei werden einzelne Kacheln im Hauptspeicher zwischengespeichert. Kleine Bilder werden vollständig im RAM gehalten.

6.1 Bildkacheln

Die Software Degate muss mit großen Mengen gerasterter Bilddaten umgehen können. Es ist möglich, dass die abfotografierten Chipflächen in Bildgrößen von 100.000 x 100.000 Punkten resultieren. Besonders wenn für das Abfahren der Chips unter dem Mikroskop ein elektronisch steuerbarer XY-Tisch existiert und das Stitching der Bilder automatisiert erfolgt, gibt es keinen Grund, auf umfangreiche Aufnahmen von hochaufgelösten Bildern zu verzichten.

Bei 100.000 x 100.000 Bildpunkten pro unkomprimiertem Bild (Graustufen) ergeben sich etwa zehn Gigabyte Daten, wenn ein Bildpunkt ein Byte Speicher belegt. Da die Schaltkreise mehrschichtig aufgebaut sind und entsprechend mehrfach die Bilddaten anfallen, ergeben sich enorme Speichermengen. Zeitkritisch ist das Rendern der Bilddaten, denn die Applikation soll interaktiv benutzbar sein. Je größer die Datenmenge ist, desto größer wäre der Rechenaufwand bzw. der IO-Overhead beim Rendern.

Meist wird man kleine Bilddatenmengen verarbeiten, etwa Bildbereiche der Größe 20.000 x 20.000 Punkte. Angesichts des Preisverfalls auf dem Speichermarkt wäre es kaum ein Problem, einem Computer diese Speichermenge als RAM zur Verfügung zustellen. Die Menge an RAM soll jedoch nicht einschränken bis zu welcher Größe Bilder verarbeitet werden können.

Große Bilder werden deshalb in Bildkacheln zerlegt. Um ein schnelles Rendern zu ermöglichen, werden ferner sogenannte Bildpyramiden verwaltet. Eine Bildpyramide ist ein Satz von Bildern, welche in mehreren vorberechneten Skalierungen vorhanden sind. Dies reduziert den Rechenaufwand und Datentransfer beim Rendern erheblich. Es muss lediglich ein Bild mit einer geeigneten Vorskalierung gewählt werden. Tatsächlich sind die vorskalierten Bilder ebenfalls in Kacheln aufgeteilt.[1]

Dadurch ergeben sich eine Reihe von Vorteilen bei lediglich einem theoretischen Nachteil. Die Verzögerung des Renderns ist unabhängig von der Zoom-Stufe. Es müssen keine großen Datenmengen im RAM gehalten werden. Das Skalieren von großen Bildern muss nicht zur Laufzeit erfolgen. Es müssen lediglich vorskalierte Bilddaten an die tatsächliche Skalierung angepasst werden. Da der Renderer auf OpenGL basiert, kann dies bei geeigneter Systemkonfiguration in der Grafikkarte erfolgen. Selbst bei sofwarebasiertem OpenGL ist dies schnell genug. Die Kacheln können direkt als OpenGL-Texturen geladen werden. In OpenGL gibt es Grenzen für Texturgrößen. Durch den kachelbasierten Ansatz werden diese Größen nicht überschritten.

Der theoretische Nachteil besteht darin, dass der kachelbasierte Ansatz einen Mehraufwand erzwingt: Degate implementiert einen eigenen C++-Typen für Bilddaten, der Primitiven für den Zugriff bereitstellt. Diese Primitiven sind zunächst die Methoden `get_pixel()` und `set_pixel()`, mit denen Bilddaten pixelweise gelesen bzw. geschrieben werden. Der Zugriff auf einen Bildpunkt erfordert, dass die Kachel bestimmt wird, in welcher der Bildpunkt liegt. Dieser Mehraufwand ist bei jedem Zugriff notwendig. Deshalb verwendet Degate zur Kompensation dieses Mehraufwandes verschiedene Strategien für einen schnellen Zugriff.

Für jeden Zugriff auf einen Bildpunkt wird zunächst geprüft, ob die korrespondierende Bildkachel bereits geladen ist. Bildkacheln in Degate haben Kantenabmessungen, die Zweierpotenzen entsprechen. Mittels Shift-Operationen wird die Kachelnum-

[1] Vorskalierte Bildkacheln werden z. B. bei der Suche nach Standardzelleninstanzen verwendet, um den Berechnungsaufwand für die Kreuzkorrelation zu verringern.

mer (i, j) bestimmt. Entspricht die Kachelnummer der zuletzt geladenen, wird diese weiterbenutzt. Für die meisten Bildpunktzugriffe reduziert sich der Aufwand somit auf zwei Shift- und zwei Vergleichsoperationen.

Von einem besonderen Zugriff macht der OpenGL-Renderer Gebrauch. Dieser lädt aus Effizienzgründen komplette Bildkacheln.

6.2 Caches

Kachelbasierte Bilder in Degate haben einen lokalen Cache. Entspricht die Kachelnummer nicht der der zuletzte verwendeten, wird geprüft, ob im lokalen Cache die angefragte Bildkachel bereits geladen ist. Ist dies zutreffend, wird diese verwendet. Ist die Bildkachel noch nicht geladen, wird sie im Speicher eingeblendet.

Alle lokalen Caches werden von einer übergeordneten Cache-Verwaltung gesteuert. Für die Software Degate kann festgelegt werden, wieviel RAM zum Zwischenspeichern von Bildkacheln verwendet werden darf. Die Cache-Verwaltung kontrolliert dabei, ob für die Menge aller Bildkacheln diese Speichergrenze überschritten wird. Lokaler Cache und die übergeordnete Verwaltung stehen dabei in Verbindung. Der lokale Cache alloziert Speicher über die Verwaltung. Stellt die Verwaltung fest, dass die Gesamtspeichermenge aller Bildkacheln das Limit erreicht, werden Bildkacheln aus dem Speicher entfernt, welche längere Zeit nicht verwendet wurden.

Die Umgebungsvariable `DEGATE_CACHE_SIZE` legt diese Gesamtlimit in der Einheit Megabyte fest. Diese Speichergrenze sollte so dimensioniert werden, dass entsprechend der maximalen Bildbreite alle Kacheln einer Zeile Platz im Cache finden. Die Kantenabmessung der Bildkacheln wird im Programmcode definiert und ist festgelegt auf 1024 Pixel. Das Bildmaterial der einzelnen Chipebenen ist im RBGA-Format. Eine einzelne Bildkachel benötgt damit 4 MB Speicher. Hat das Bildmaterial in Degate eine horizontale Abmessung von z. B. 8000 Pixel, werden vier Kacheln benötigt, um eine Zeile im Cache halten zu können. Dies entspricht einer empfohlenen Cachegröße von 16 MB.

6.3 Datentypen für Bilddaten

Der allgemeine Datentyp in Degate für Bilddaten ist die Klasse `Image<PixelPolicy, StoragePolicy>`. Diese Klasse trägt zwei Template-Parameter. Beide Parameter sind Klassen, die aufgrund ihrer Verwendung Policy-Klassen genannt werden. Diese Policy-Klassen beinflussen das Verhalten der Klasse Image von außen (vgl. [Ale01, S. 7ff]).

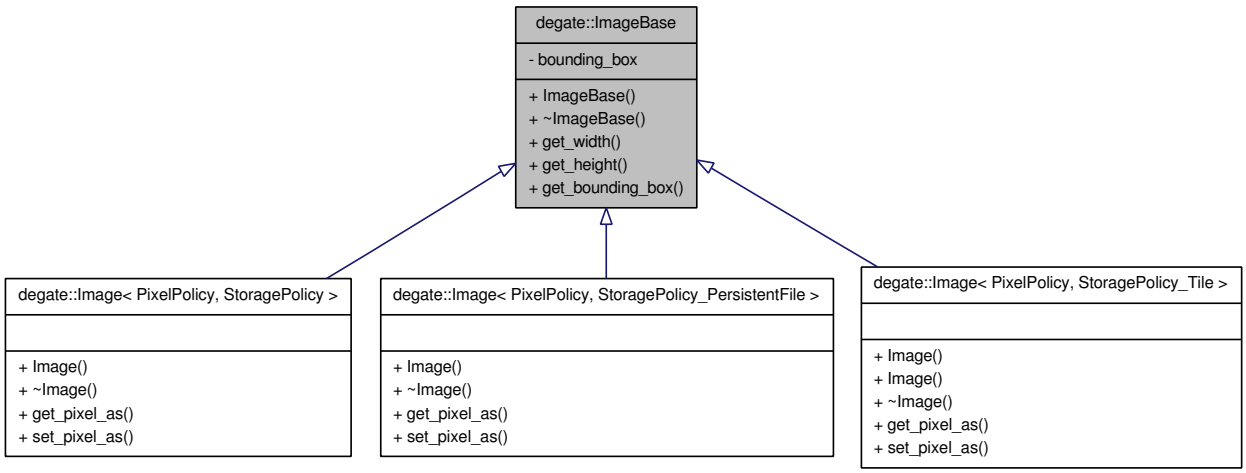

Abbildung 6.1: Vererbungsgraph für die Klasse Image

Der erste Templateparameter gibt den Pixel-Typen an. Dafür sind drei verschiedene Typen vorgesehen: RGBA-Werte, in denen die Anteile für rot, grün, blau und den Alpha-Kanal getrennt vorliegen, Graustufen, die als Bytewert repräsentiert werden und Gleitkommazahlen. Letztere Pixeltypen können ebenfalls Graustufen darstellen oder für gänzlich andere Zwecke verwendet werden, beispielsweise für Korrelationswerte.

Mit dem zweiten Templateparameter kann festgelegt werden, wie die Bilddaten gespeichert werden sollen. Degate bietet dafür Policy-Klassen, mit denen Bilddaten im Hauptspeicher gehalten werden können. Darüber hinaus steht eine Policy-Klasse zur Verfügung, mit der es möglich ist, Bilddaten in einer einzelnen Datei zu speichern. Dafür gibt es zwei Spezialisierungen, und zwar für die temporäre und für die permanente Speicherung. Im Unterschied zur permanenten Speicherung wird bei temporärer Plattenplatz wieder freigegeben, wenn das Bild nicht mehr benötigt wird. Bei der dateibasierten Speicherung werden Bilddaten mittels der Betriebsystemfunkti-

on `mmap()` in den Speicher eingeblendet. Diese Form der Speicherung wird von einer dritten Policy-Klasse verwendet, und zwar derjenigen, die den kachelbasierten Ansatz implementiert.

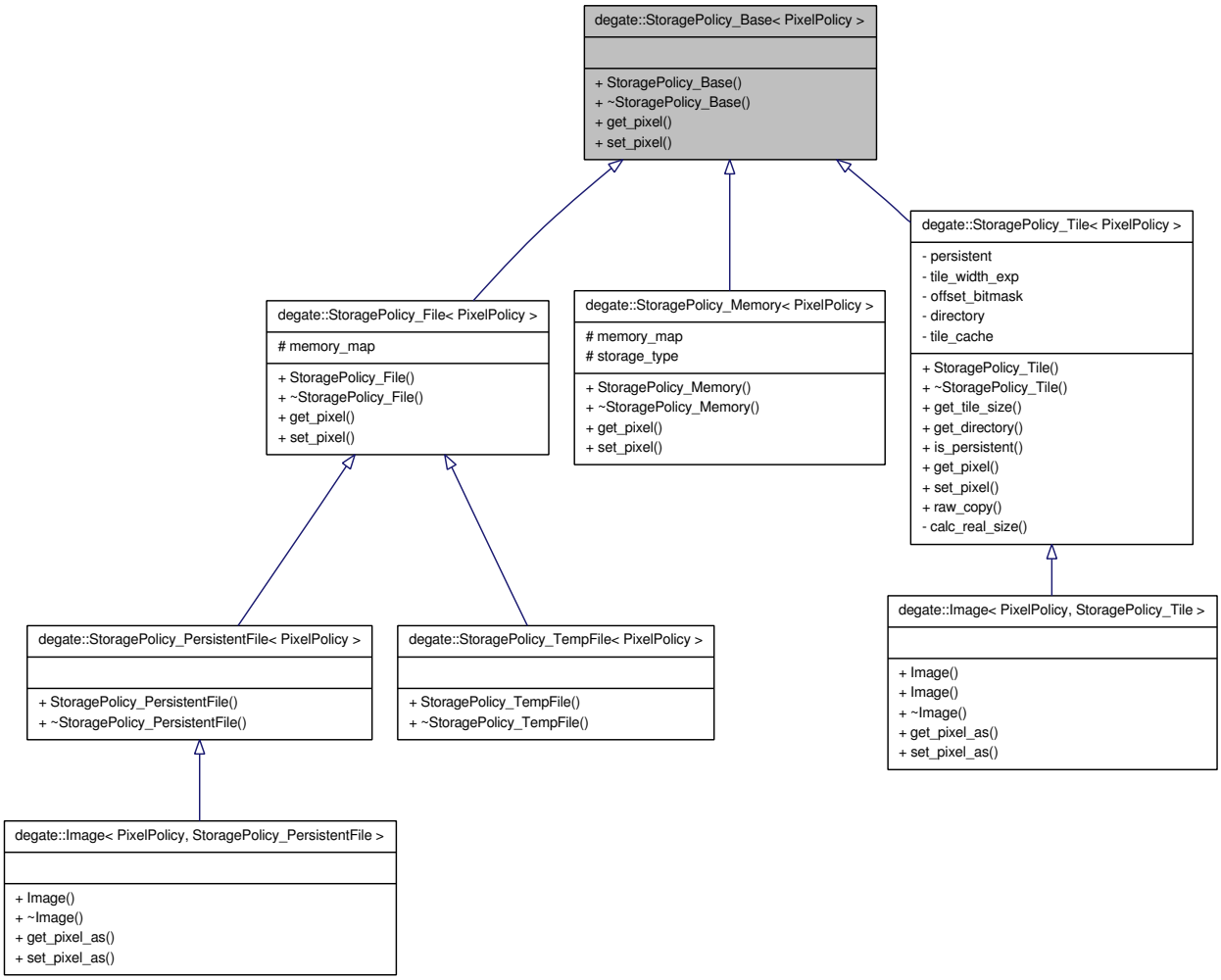

Abbildung 6.2: Vererbungsgraph für die StoragePolicy-Klassen. Die Image-Klassen erben u. a. von StoragePolicy.

Mittels dieses Ansatzes können Bilddatentypen für verschiedene Pixeltypen bei verschiedenen Speicherformen zur Kompilierungszeit generiert werden. Der Compiler erzeugt Programmcode, der ohne zusätzlichen Overhead auskommt. Die Festlegung des Bilddatentypen erfolgt konzeptuell während des Programmierens. Beispielsweise wird man für große Datenmengen einen kachelbasierten Typen wählen. Wenn reelle Werte für jeden Bildpunkt verwaltet werden müssen, wird man z. B. die Klasse `PixelPolicy_GS_DOUBLE` mit dem Kachelmodus kombinieren.

Für elementare Zugriffe stehen die Primitiven `get_pixel()` und `put_pixel()` zur Verfügung. Für die implizite Konvertierung von Bilddaten können die Funktionen `get_pixel_as<PixelType>()` und `set_pixel_as<PixelType>()` genutzt werden. Der Template-Parameter legt dabei den Zieldatentyp für die Konvertierung fest. Datenkonvertierungen sind z. B. dann sinnvoll, wenn etwa RBGA-Bilddaten gelesen und in intensitätsabbildende Graustufenwerte konvertiert werden müssen. Der Code für die Umwandlung muss nicht selbst aufgerufen werden. Die Konvertierung erfolgt implizit und nur wenn das Zielformat sich vom Quellformat unterscheidet.

Kapitel 7

Automatische Erkennung von Standardzelleninstanzen

Eine automatisierte Objekterkennung soll dabei helfen, die Komplexität des Reverse-Engineerings handhabbar zu machen. Dazu sind in Degate Funktionen zum Finden von Standardzelleninstanzen integriert. In diesem Kapitel wird der verwendete Algorithmus erläutert.

Das Bild einer Standardzelleninstanz kann als Schablone verwendet werden, die über einen Bildausschnitt verschoben wird. Für jede Verschiebung lässt sich die Übereinstimmung der Schablone mit dem jeweiligen Ausschnitt aus dem Hintergrundbild bestimmen. Dieses Schema wird allgemein als „Template Matching“ bezeichnet. Die normalisierte Kreuzkorrelation ist dabei ein gängiges Maß für die Übereinstimmung.

Die normalisierte Kreuzkorrelation einer Schablone t zu einem Bildausschnitt f an einem Eckpunkt (x_0, y_0) berechnet sich zu:

$$c(x_0, y_0) = \frac{1}{n-1} \sum_{x,y} \frac{(f(x+x_0, y+y_0) - \bar{f}) * (t(x,y) - \bar{t})}{\sigma_f \sigma_t} \tag{7.1}$$

Dabei bezeichnen σ_f und σ_t die Standardabweichungen und $\bar{f}$ und $\bar{t}$ die Mittelwerte des Bildausschnitts bzw. der Schablone. Mittels Subtraktion der Mittelwerte und Division durch die Standardabweichungen werden die Korrelationswerte gegenüber Helligkeitsschwankungen unempfindlich. Insbesondere bei Auflichtmikroskopie muss die Beleuchtung nicht homogen sein.

Um Bilder von Gatterinstanzen zu finden, kann die Schablone t Pixel für Pixel über das Hintergrundbild verschoben und jeweils die Korrelation zum Bildausschnitt f berechnet werden. Das Ergebnis ist eine Matrix mit Korrelationswerten. Stellen mit maximalen Korrelationswerten entsprechen möglichen Gatterplatzierungen. Dieses Verfahren ist rechenintensiv und kann an mehreren Stellen optimiert werden. In der Software Degate ist eine Variante der „schnellen normalisierten Kreuzkorrelation“ implementiert [BH01].

7.1 Optimierungen

Wenn Bildmaterial in Degate importiert wird, generiert die Software mittels Interpolation herunterskalierte Versionen des Bildes. Diese werden für die Darstellung von Übersichtsbildern verwendet. Sie können ebenfalls für die Korrelationsberechnung eingesetzt werden, denn ein um den Faktor i verkleinertes Bild verringert den Aufwand zur Berechnung einer Korrelation um den Faktor i^2 und schränkt ebenfalls den Suchraum um den Faktor i^2 ein. Eine Suche auf skalierten Bilddaten verringert zwar den Berechnungsaufwand. Da aber weniger Daten für die Korrelationsbestimmung herangezogen werden, vergrößert sich das zum Korrelationskoeffizienten gehörige Konfidenzintervall und damit die Wahrscheinlichkeit von Fehlerkennungen. Daher ist diese Optimierung nur für große Suchmuster sinnvoll.

Die größte Optimierung der Gattersuche ergibt sich aus der Einschränkung des Suchraumes. Dabei kann die Eigenschaft ausgenutzt werden, dass Gatter auf dem Substrat nicht zufällig angeordnet werden, sondern in Reihen oder Spalten. Die Software Degate unterstützt Gitterlinien, die der Benutzer in regelmäßigen bzw. unregelmäßigen Abständen sowohl horizontal als auch vertikal anordnen kann. Sind die Gitterlinien entlang der Gatterplatzierung verlegt, profitiert davon das Suchverfahren, da nunmehr nur die Schablone entlang der Gitterlinien verschoben wird.

Die Suche kann entlang von Gitterlinien, auf dem gesamten Bild bzw. in Teilbereichen stattfinden, auf dem Transistor-Layer oder auf der Chipebene M1. Ferner ist es möglich, die Suche für mehrere Gattertypen als Stapelverarbeitung durchzuführen. Die Erkennung funktioniert auch, wenn Gatter gegenüber dem Referenztyp vertikal und/oder horizontal gespiegelt sind.

Die Schablone t muss nicht Pixel für Pixel verschoben werden. Die Korrelation der Schablone mit dem Bildausschnitt an einer Stelle ist ähnlich der Korrelation für

benachbarte Stellen. Aufgrund dieser Stetigkeitsannahme kann die Schablone um mehrere Pixel verschoben werden. Dadurch werden weitere Korrelationsberechnungen gespart. Für die Gattererkennung in Degate kann die maximale Schrittweite festgelegt werden. Die Schrittweite ist adaptiv, d. h. in Bereichen mit geringer Korrelation ist die Schrittweite größer als in Bereichen mit hohen Korrelationswerten.

Für das Finden der genauen Gatterposition benutzt Degate ein Bergsteigeverfahren (engl. *hill climbing*). Ein Schwellwert t_{hc} legt fest, dass für Korrelationswerte $c(x_0, y_0) \geq t_{hc}$ das Bergsteigen an der Stelle (x_0, y_0) gestartet werden soll. Das Bergsteigen berechnet die Korrelation der Schablone mit allen acht Bildausschnitten, die um die Stelle (x_0, y_0) herum beginnen. Die Stelle (x', y') mit der höchsten Korrelation dient als neuer Startpunkt für den nächsten Schritt des Bergsteigeverfahrens. Das Bergsteigen erfolgt iterativ, bis keine höheren Korrelationswerte mehr erreicht werden, d. h. ein lokales Maximum gefunden wird. Die lokalen Maxima werden in einer Liste gespeichert. Diese Liste wird in einem letzten Schritt anhand des Korrelationswertes sortiert und ausgewertet. An den Stellen mit maximalem Korrelationswert werden die jeweilig gefundenen Gatter platziert. Ein zweiter Schwellwert legt dafür einen Mindestkorrelationswert fest.

Die Erkennungsrate ließe sich erhöhen, wenn ein kombiniertes Template-Matching auf allen merkmalstragenden Ebenen eines Chips, d. h. auf dem Transistorlayer und auf Metal 1, erfolgen würde. Dieser Ansatz ist in Degate bisher nicht implementiert. Selten tritt die Situation auf, dass die Layoutdaten zweier Standardzellen für eine Ebene des Chips nahezu identisch sind, sich sie Funktionen jedoch unterscheiden (siehe Abbildung 7.1). Dies führt bei der automatisierten Suche zu Fehlern, die mit einem kombinierten Template-Matching verhindert werden könnten.

7.2 Wahl der Schablone

In Degate ist es möglich, ein Gatterbild als Schablone auszuwählen, welches für die Suche nach Platzierungen verwendet wird. Die Wahl des Gatterbildes beeinflusst Erkennungs- sowie Fehlerkennungsraten. Je „schlechter“ eine Schablone die Menge der platzierten Gatter repräsentiert, desto geringer ist die Erkennungsrate. Der Benutzer wird ein Gatterbild wählen, das ein geeigneter Repräsentant ist. Die Wahl ist jedoch subjektiv und die Güte kann qualitativ nicht bestimmt werden, da die meisten Gatterinstanzen noch nicht gefunden wurden. Mitunter ist die Chipoberfläche nicht

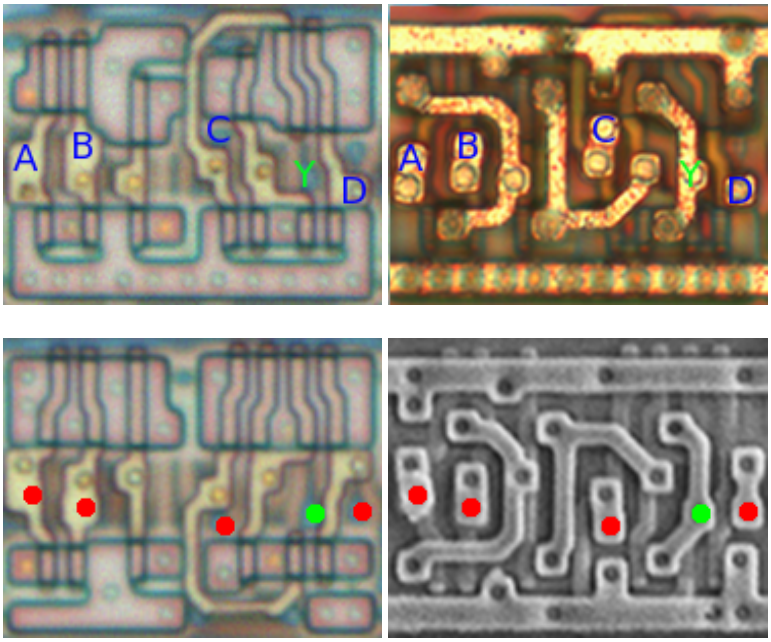

Abbildung 7.1: Zwei Standardzellen mit unterschiedlicher Funktion und nahezu identischem Layout auf der Ebene Metal 2 (gespiegelt). Die Schicht Metal 2 der oberen Zeile wurde mittels optischer Mikroskopie aufgenommen. Die Aufnahme unten rechts erfolgte durch ein Rasterelektronenmikroskop. In den Abbildungen sind die p-Kanal-FETs jeweils im oberen Teil des Bildes. (Chip: Megamos)

exakt parallel zu den Schichten eines Chips poliert, so dass großflächige Bereiche auftreten, in denen eine Chipebene stärker poliert ist. Es kann dann sinnvoll sein, für den stärker polierten Bereich ein anderes Gatterbild als Schablone zu verwenden, d. h. einen lokal besser geeigneten Repräsentanten zu wählen. In Degate gibt es diesbezüglich keine Festlegungen, wann welche Schablone verwendet werden soll. Für jeden Standardzellentyp gibt es jeweils ein aktuell festgelegtes Bild für die Transistorschicht, Metal 1 und Metal 2, wenn Bildmaterial für die jeweiligen Schichten in einem Degate-Projekt hinterlegt sind. Es ist dem Benutzer überlassen, geeignete Gatterbilder auszuwählen.

Degate bietet die Möglichkeit, Mischbilder zu ermitteln, die dann als Schablone verwendbar sind: Die Suche nach Standardzellentypen und entsprechenden Instanzen ist ein iterativer Arbeitsschritt. Für jeden Typ werden anfänglich zwei oder drei Positionen beobachtbar sein, an denen Gatter des Typs platziert sind. Für eine der Positionen kann der Standardzellentyp definiert werden. Implizit wird das zugrunde-

liegende Gatterbild als Schablone ausgewählt. Der Benutzer kann die aufgefundenen Gatterinstanzen markieren und Degate anweisen, ein Mischbild zu erzeugen und dies als neues Referenzbild zu hinterlegen. Das Mischbild ist ein pixel- und farbkanalweise arithmetisch gemitteltes Bild. Die Funktion kann auch für eine Auswahl von Gatter verschiedener Typen und verschiedener Spiegelungen angewendet werden. Degate ermittelt dann, welche Gattertypen von der Aktualisierung betroffen sind und berechnet für jeden Typ das neue Referenzbild.

7.3 Evaluation

Um zu bestimmen, ob und welchen Einfluss die Verwendung von Mischbildern und Filtermethoden auf die Erkennungs- bzw. Fehlerkennungsrate hat, wurden verschiedene Experimente durchgeführt. Bei diesen wurden Standardzelleninstanzen in einem Referenzprojekt gesucht. Als Referenzprojekt diente der Logikbereich eines Legic-Chips. Für diesen Logik-Bereich sind 604 plazierte Standardzellen bekannt und manuell geprüft worden (vgl. Anhang B). Die Treffer wurden mit den Referenzdaten verglichen[1]. Das Hintergrundbild wurde für die Experimente in einigen Durchläufen gefiltert und in anderen Fällen unbearbeitet gelassen, um den Einfluss von Filteroperationen vergleichen zu können. Somit gibt es drei verschiedenen Modi für den Hintergrund:

- Filterung des Hintergrundbildes mittels Gaussfilters (5x5, $\sigma = 1.1$)
- Filterung des Hintergrundbildes mittels Median-Filter (3x3)
- Keine Behandlung des Hintergrundbildes

Für die Standardzellensuche wurde sowohl ein manuell ausgewähltes Bild eines jeweiligen Standardzellentyps verwendet als auch ein jeweiliges Mischbild, das aus allen vorhandenen Zellinstanzen des gleichen Typs erzeugt wurde. Für diese insgesamt sechs Experimente wurden Messreihen aufgenommen. Dafür wurde der Schwellwert für die Akzeptanz von Standardzellen t_{det} zwischen 0,3 und 0,9 in Schritten von 0,1 verändert. Der Schwellwert t_{det} legt fest, dass eine Schablone mit dem korrespondierenden Ausschnitt aus dem Hintergrundbild mindestens mit t_{det} korrelieren muss, damit die entsprechende Stelle als Standardzellenplatzierung gewertet wird. Wenn

[1] Im git-Repository gibt es für den maschinellen Vergleich ein Perl-Skript unter *tools/cell_matching_evaluation/*.

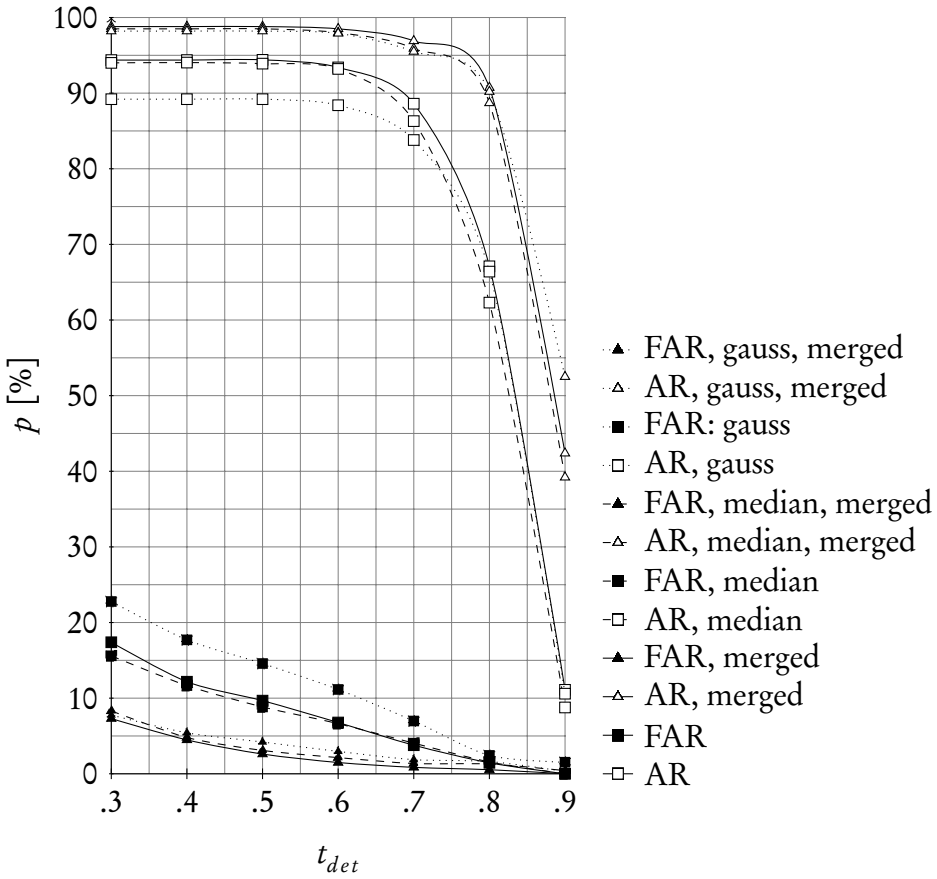

Abbildung 7.2: Darstellung der Erkennungsraten (oben) und Fehlerkennungsraten (im unteren Teil) in Abhängigkeit vom Korrelationsschwellwert

zwei verschiedene Schablonen höhere Korrelationswerte als t_{det} liefern, wird diejenige mit dem höheren Korrelationswert akzeptiert.

Die Zusammenfassung aller Bildinstanzen zu einer Schablone erfolgte aus praktischen Erwägungen. Zum einen entfällt dadurch die subjektive Auswahl geeigneter Standardzellenbilder, zum anderen reduzierte dies die Anzahl von Experimenten gegenüber zufällig auszuwählenden Teilmengen von Standardzelleninstanzen. In der Summe wurden dadurch nur 42 vollständige Suchläufe durchgeführt. Die Ergebnisse sind in der Grafik 7.2 zusammengefasst.

Auf der x-Achse sind die Schwellwerte für die Detektion von Standardzelleninstanzen aufgetragen, d. h. die Mindestkorrelationswerte. Auf der y-Achse sind die Erkennungsraten und Falscherkennungsraten eingetragen. Hier sind die Erkennungsraten (AR = *acceptance rate*) höher als die Falscherkennungsraten (FAR = *false acceptance rate*), weshalb die Messwerte für die Fehlerkennung im unteren Teil und die Messpunkte für

die Erkennungsraten im oberen Teil des Diagrams abgebildet sind. Die Messwerte der einzelnen Messreihen wurden zur besseren Lesbarkeit mit Bezierkurven verbunden.

Die als unausgefüllte Dreiecke dargestellten Messwerte bedeuten, dass die Standardzellenbilder durch Mittelwertgewinnung erzeugt wurden. Unausgefüllte Quadrate stellen Messreihen dar, bei denen einzelne Bilder als Schablonen verwendet wurden. Die jeweils ausgefüllten Symbole stehen für die Fehlerkennungsraten: Ausgefüllte Dreiecke sind Fehlerkennungsraten von Mischbildern, ausgefüllte Quadrate stehen für Einzelbilder. Die Linien der Bezierkurven stellen den Modus der Hintergrundbildbearbeitung dar: Durchgezogenen Linien weisen auf einen unbearbeiteten Hintergrund hin. Gestrichelte Linien geben eine Medianfilterung an. Gepunktete Linien bedeuten eine Gaussfilterung des Hintergrundbildes.

Anhand des Diagramms ist erkennbar, dass der Einfluss des Gauss- bzw. des Medianfilters auf die Erkennungsraten gering ist und partiell Erkennungsraten sogar senkt. Die Verwendung von Mischbildern wirkt sich dagegen positiv auf die Erkennungsraten aus. Gleichzeitig verringert sich die Fehlerkennungsrate. So beträgt für einen Schwellwert von 0,7 die Erkennungsrate 96,9 % bei einer Fehlerkennungsrate von 0,9 %. Die Bildqualität des Referenzprojektes ist eher gering, so dass bei besserer Qualität noch höhere Erkennungsrraten zu erwarten wären.

7.4 Behandlung von Fehlern

Die automatisierte Analyse von Bilddaten ist immer mit Fehlern behaftet. Es gibt keine Möglichkeit, das Auftreten von Fehlern automatisiert festzustellen. Derartige Verfahren wären allenfalls heuristisch und würden wiederum Fehlentscheidungen treffen. Um falsch erkannte Standardzellenplatzierungen zu erkennen, kann der Benutzer eine optische Kontrolle durchführen. Rule Checks erlauben, weitere Fehler zu finden.

Fälschlich erkannte Standardzellen ließen sich durch manuellen Bildvergleich von erkannter und realer Standardzelle gegebenfalls unter Zuhilfenahme eines Differenzbildes ermitteln. Für eine Entscheidung, ob eine Standardzellenplatzierung korrekt ist, bräuchte ein Mensch wahrscheinlich höchstens eine Sekunde. Dieses Verfahren kann deshalb als akzeptabel erachtet werden, wenn der Zeitaufwand für die Prüfung eine Stunde nicht überschreitet. Dies entspräche etwa 3600 platzierter Standardzellen – weit mehr als die Gesamtanzahl aller Standardzellenplatzierungen bei bisherigen

Schaltkreisrekonstruktionen mit Degate. Diese Funktion ist bisher nicht implementiert.

7.5 Alternative Verfahren

Fumihiko Saitoh schlägt ein Verfahren zum Template-Matching mittels Kanten-Spin-Korrelation vor [Sai05]. Ein Eingangsbild wird mit einem Prewitt- oder Sobeloperator gefaltet, wodurch ein Kantenbild entsteht. Für jeden Pixel im Kantenbild wird innerhalb einer Umgebung, z. B. 3x3 Pixeln, der Winkel der Kantennormalen bestimmt. Horizontal benachbarte Pixel werden dann hinsichtlich ihrer Winkel verglichen. Ist dem rechten Pixel ein größerer Winkel zugeordnet als der Winkel des betrachteten Pixels, wird dem betrachteten Pixel der Wert 1 zugeordnet, anderenfalls der Wert 0.

Dieser „Fingerabdruck“ wird sowohl für eine Schablone als auch für ein Bild ermittelt. Das Verfahren vergleicht dann pixelweise die Winkel und bildet das Verhältnis aus übereinstimmenden Winkeln zur Anzahl aller Winkel. Für nicht korrelierende Bilder ist dieses Verhältnis 0,5. Für maximal positiv korrelierende Bilder ist das Verhältnis 1. Anhand des Verhältnisses kann dann entschieden werden, ob eine Schablone und ein Bild als übereinstimmend betrachtet werden.

Kapitel 8

Automatisiertes Erkennen von Leiterbahnen und Durchkontaktierungen

In diesem Kapitel wird der Einfluss mikroskopischer Verfahren auf die Bildqualität behandelt. Die Bildqualität bei Verwendung von günstiger Ausrüstung ist geringer als bei Verwendung höherwertiger Geräte. Degate bietet deshalb Verfahren, mit denen Bilddaten manuell und automatisch ausgewertet werden können. Das Nachvollziehen elektrischer Verbindungen ist der zeitaufwendigste Arbeitsschritt, da er bisher vollständig manuell durchgeführt wurde. Etwa zwei Drittel der Zeit eines Reverse-Engineering entfallen auf das Nachvollziehen der elektrischen Verbindungen. Demnach wirkt sich jede Verbesserung in besonderem Maße positiv auf die Dauer des Reverse-Engineerings aus. Ein wesentliches Mittel zur Beschleunigung ist daher die automatische Erkennung von Leiternbahnen und Durchkontaktierungen. Ein Verfahren hierfür wird im Folgenden vorgestellt.

8.1 Nachvollziehen elektrischer Verbindungen

Abhängig davon, welche Geräte zur Aufnahme von Bildmaterial der Chipebenen verwendet werden, ist die Qualität des Bildmaterials verschieden und dadurch mehr oder weniger für eine automatisierte Auswertung von Leiterbahnen und Durchkontaktierungen geeignet. In der Software Degate wird diesem Umstand insofern Rechnung getragen, als es verschiedene Optionen zum Nachvollziehen der elektrischen Verbindungen gibt. Es gibt Modi, mit denen qualitativ ungünstiges Bildmaterial für ein

Reverse-Engineering verwendet werden kann. Der Nachteil besteht jedoch darin, dass sich der Grad der manuellen Bearbeitung erhöht. Durch „crowd sourcing“ mittels mehrer Projektteilnehmer lässt sich das manuelle Bearbeiten verteilen und dadurch parallelisieren (siehe Kapitel 10).

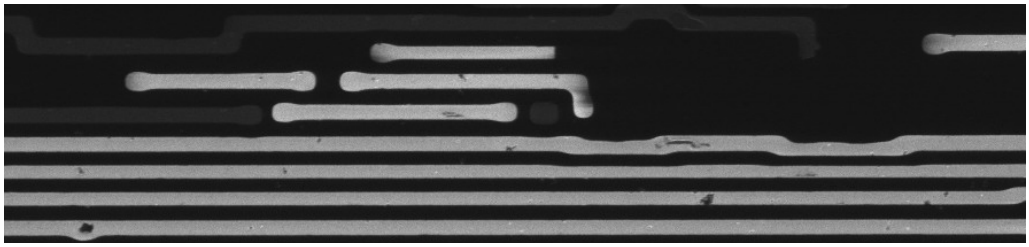

Abbildung 8.1: Aufnahmen von Leiterbahnen mittels FIB (Foto: Christopher Tarnovsky).

Hochwertiges Bildmaterial kann automatisch ausgewertet werden. Dies ist beispielsweise möglich, wenn Bilddaten mittels Ionenfeinstrahlanlage (vgl. Abbildung A.9) oder mittels eines Rasterelektronenmikroskops (vgl. Abbildung 8.2) aufgenommen werden. Die emittierten Elektronen dringen nicht in die Oberfläche des Chips ein und bilden somit nur diese ab. Dadurch entstandene Bilder weisen einen maximalen Kontrast auf – Leiterbahnen sind weiß und der Hintergrund ist schwarz.

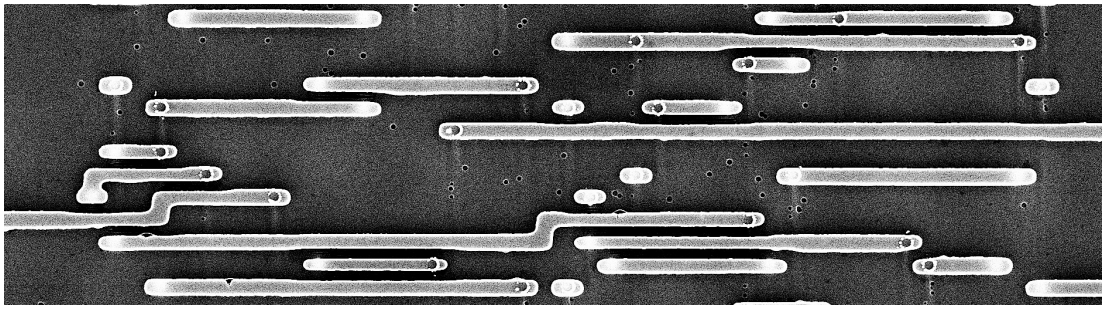

Abbildung 8.2: Aufnahmen von Leiterbahnen mittels Rasterelektronenmikroskop.

Bei der Auflichtmikroskopie werden tieferliegende Ebenen meistens mit abgebildet, denn der Chip besteht in großen Teilen aus durchsichtigem Siliziumdioxid. Dies führt bei der Kantenerkennung zu Problemen, denn die durchscheinenden Ebenen weisen ebenfalls Kanten auf (vgl. Abbildung 8.3). Der Effekt kann durch hohe Vergrößerungen abgeschwächt werden, denn dadurch verkleinert sich der Stellbereich, in dem die oberste Ebene fokussiert ist. Entfernter liegene Ebenen sind dann unscharf und weisen

schwächere Kanten auf, die durch den Einsatz von Median-Filtern entfernt werden können. Bei bisherigen Reverse-Engineering-Prozeduren wurde aus verschiedenen Gründen darauf verzichtet: Mangels eines motorisierten X-Y-Tisches mussten Bildaufnahmen vollständig manuell aufgenommen werden. Jede Verdopplung der optischen Vergrößerung bedeutet demnach, dass sich die Anzahl der aufzunehmenden Bildkacheln vervierfacht. Dabei ist eine Überlappung der Bildkacheln nicht einbezogen, die für das automatisierte Zusammensetzen notwendig ist.

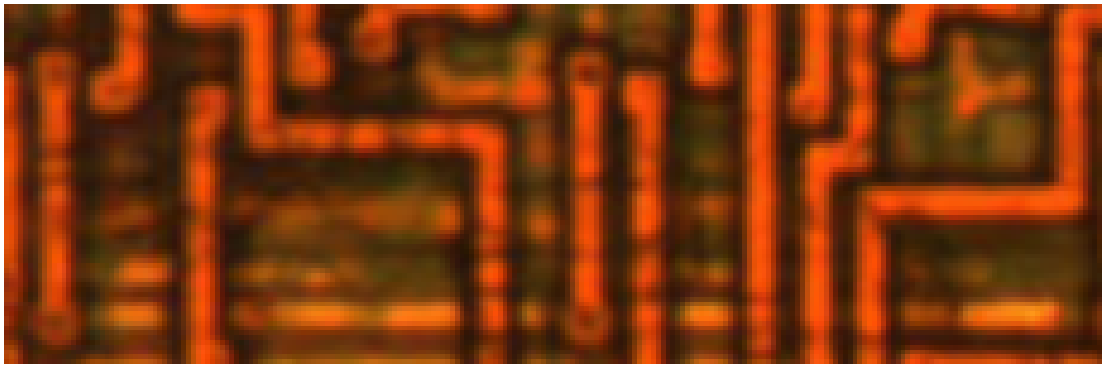

Abbildung 8.3: Aufnahmen von Leiterbahnen mittels Auflichtmikroskop (Chip: Legic, nicht planarisiert).

Eine Besonderheit bilden Chips, deren Ebenen nicht planarisiert sind. Dies trifft vor allem auf ältere Chips zu. Planarisiert bedeutet, dass eine Ebene im Chip überall die gleiche Schichtstärke hat. Bei nichtplanarisierten Chips sind Ebenen nicht überall gleich stark, insbesondere an Stellen, an denen Leiterbahnen verlaufen. Bei der mikroskopischen Abbildung enstehen dadurch Abschattungen (ebenfalls Abbildung 8.3), die ein Kantenfilter in Kanten überführt.

Ein ähnliches Problem ergibt sich für konfokalmikroskopisch aufgenommene Bilder. Konfokalmikroskope können alle Schichten in der gleichen Schärfe abbilden (Abbildung 8.4, oben). Für die manuelle Inspektion ist dies ein Vorteil, da die Verbindungen ebenenübergreifend besser dargestellt werden. Für ein algorithmisches Auswerten ist diese Darstellung ungeeignet, da in den Bildern Kanten von darunterliegenden Schichten zu sehen sind. Für die Bilderkennung sollten jedoch einzelne Schichten fokussieren werden (Abbildung 8.4, unten).

In Situationen, in denen Leiterbahnverläufe nicht maschinell ausgewertet werden können, müssen Signalverläufe manuell nachvollzogen werden. Degate bietet dafür mehrere Optionen an. Seit der Prototypenphase gibt es die Möglichkeit, mehrere Ports

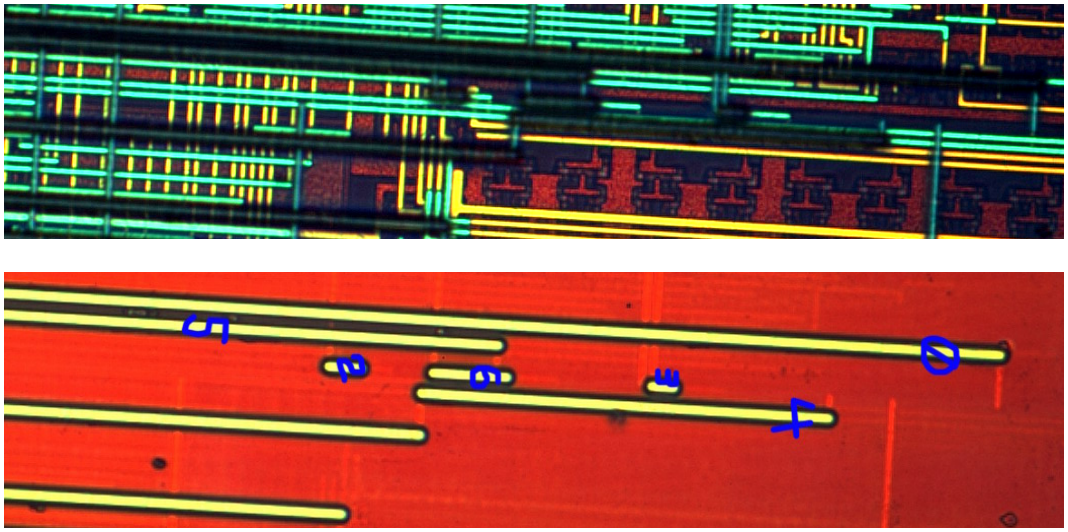

Abbildung 8.4: Aufnahmen von Leiterbahnen mittels Konfokalmikroskop mit verschiedener Fokussierung (Fotos: Christopher Tarnovsky).

von Standardzelleninstanzen zu selektieren und diese derart elektrisch zu verbinden, dass diese Ports einem Netz zugeordnet werden. Im Logik-Modell gelten diese Ports dann als elektrisch verbunden, ohne dass der Verlauf von Leiterbahnen in der Software gespeichert wird. Dies erfordert dennoch, dass deren Verlauf manuell verfolgt wird. Der Nachteil besteht darin, dass zunächst nicht erkennbar ist, ob zwei Objekte bereits elektrisch verbunden wurden.

Die zweite Option besteht darin, dass der Verlauf von Leiterbahnen und Durchkontaktierungen nachgezeichnet wird. Dazu gibt es in der Werkzeugpalette der GUI entsprechende Schaltelemente. Mit dem „Wire-Tool“ können mit der mittleren Maustaste ebenfalls Durchkontaktierungen eingetragen werden. Bei gleichzeitigem Drücken der Umschalttaste lassen sich Durchkontaktierungen zu einer höheren Chipebene festlegen. Ohne Drücken der Umschalttaste sind die Durchkontaktierungen zum unteren Layer gerichtet.

Wenn das Nachvollziehen von elektrischen Verbindungen mit dem Eintragen von Leiterbahnen und Durchkontaktierungen erfolgt und diese Objekte im Logik-Modell gespeichert werden, können die Positionen dieser Objekte zu einem Server übertragen werden. Auf dem Server gespeicherte Änderungen können in das lokale Projekt übernommen werden. Mit diesen (noch sehr experimentellen) Funktionen ist es möglich, mit mehreren Projektteilnehmern kollaborativ Leiterbahen zu verfolgen. Über Funktionsweise und Einschränkungen gibt Kapitel 10 Auskunft.

Stehen die Positionen der Leiterbahnen und Durchkontaktierungen fest, können sie elektrisch verbunden werden. In der aktuellen Version von Degate sind dies zwei separate Arbeitsschritte. Zunächst werden sich berührende geometrische Objekte miteinander elektrisch verbunden. In einem zweiten Schritt werden Verbindungen zwischen Chipebenen hergestellt. Diese Funktionen lassen sich ebenfalls auf ausgewählte Bereiche anwenden. Im Prototypen wurden Objekte beim Eintragen in das Logikmodell automatisch elektrisch verbunden, sobald geometrische Überschneidungen gegeben sind. Die Diskussion, welches Verhalten sinvoll ist, ist mangels praktischer Anwendung noch offen. Möglicherweise wird sich dieses Funktionsmerkmal in einer zukünftigen Version ändern.

8.2 Voruntersuchungen und Probleme

Anhand des Vergleiches von verschiedenem Bildmaterial einzelner Reverse-Engineering-Projekte ergeben sich eine Reihe von Eigenschaften, die für die automatische Auswertung grundsätzlich genutzt werden könnten. Allerdings lassen sich auch zahlreiche Probleme erkennen, die zunächst erläutert werden sollen. Ein wesentliches Problem ist die Überlagerung von Bilddaten benachbarter Schichten.

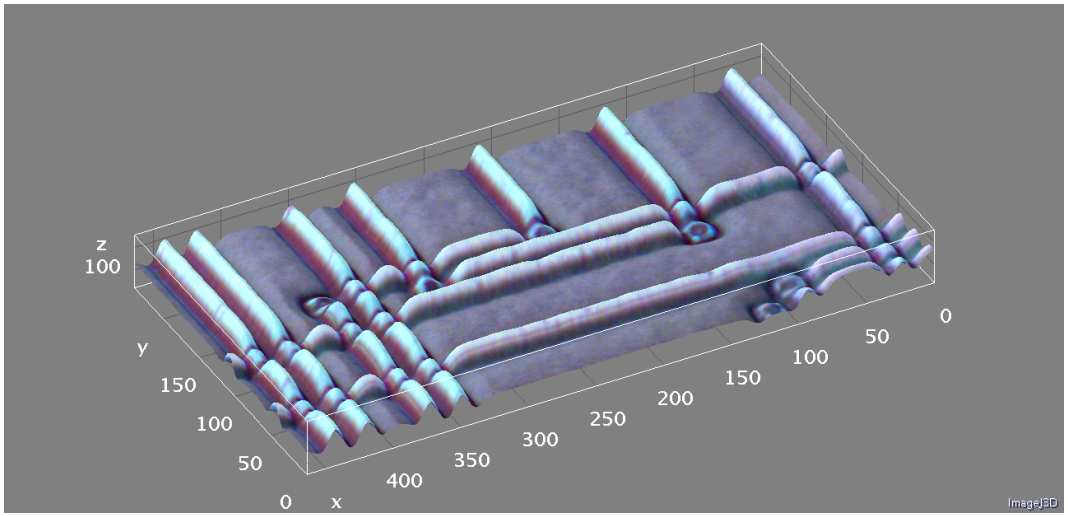

Abbildung 8.5: Höhenprofil der Leiterbahnen eines Megamos-Chips. Das Ausgangsbild wurde mittels Auflichtmikroskopie erstellt. Im Bild überlagern sich Leiterbahnverläufe zweier benachbarter Chip-Ebenen. Die z-Achse gibt die Intensität des Pixelwertes an.

Abbildung 8.5 zeigt exemplarisch einen Bildausschnitt der fotografisch aufgenommenen Chipoberfläche in einer Höhenprofildarstellung. Die darunterliegende Ebene ist ebenfalls sichtbar. Aus dieser Darstellung ergibt sich, dass die Helligkeitswerte der darunterliegenden Leiterbahnen ähnlich hohe Werte erreichen wie die Leiterbahnen auf der oberen Schicht. Ferner zeigt die Profilansicht, dass die Helligkeitswerte an Stellen abgeschwächt sind, an denen sich Leiterbahnen „kreuzen“, selbst wenn die Leiterbahnen auf unterschiedlichen Ebenen verlaufen. Verfahren, mit denen sich die Leiterbahnverläufe detektieren ließen, basieren überwiegend auf Kantenerkennungsfiltern. Stellen, an denen die Helligkeitswerte absinken, erzeugen neue Kanten. Dies würde in Leiterbahnfragmenten resultieren.

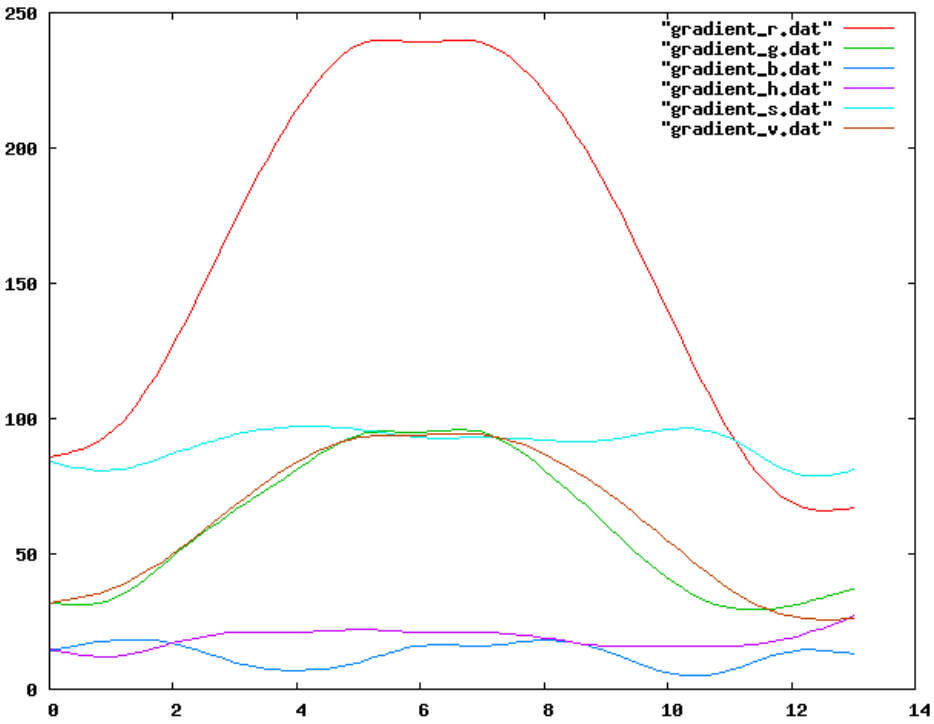

Abbildung 8.6: Höhenprofil einer Leiterbahn aufgeteilt nach Farbkanal im RGB- und HSV-Farbraum (Chip: Legic). Die x-Achse gibt Pixelabstände an. Entlang der y-Achse ist der Wert des Farbkanals abgebildet. Die diskreten Messwerte wurden zugunsten einer bessere Darstellung verbunden.

Im Vorfeld wurden einige Eigenschaften des Bildmaterials daraufhin untersucht, inwiefern sich diese für die automatisierte Leiterbahnerkennung nutzen ließen. Zu diesen Eigenschaften zählen:

- Helligkeits- bzw. Farbwerte entlang eines Schnitts orthogonal zur Verlaufsrichtung einer Leiterbahn entsprechen häufig einer Art „Höhenprofil“ (vgl. Abbildung 8.6). Es wäre möglich, dieses Profil auszuwerten. Allerdings würde auch dieses Verfahren an den oben beschriebenen Stellen scheitern, an denen sich Leiterbahnen (verschiedener Chip-Ebenen) kreuzen.
- Leiterbahnen unterscheiden sich oft in ihren Helligkeits- bzw. Farbwerten von tiefer gelegenen Objekten. Die Unterschiede können marginal ausfallen (siehe Abbildung 8.5) oder in einzelnen Farbkanälen erhebliche Unterschiede aufweisen (siehe Abbildung 8.7). Dieses Verfahren ist grundsätzlich geeignet, um Hintergrundbereiche vom Vordergrund zu trennen. So ist z. B. je nach Farbwert eine Wahrscheinlichkeitssaussage möglich, auf welcher Chipebene ein zu einer Leiterbahn gehörender Pixel liegen könnte. Allerdings gibt es Farbwerte, für die keine korrekte Zuordnung möglich ist. Dies betrifft wieder Stellen, an denen Leiterbahnenverläufe kreuzen. In einigen Fällen ist der Abstand von Leiterbahnen so gering, dass Pixel im Zwischenraum Farbwerte aufweisen, die ähnlich denen der Leiterbahn sind.
- Leiterbahnen folgen häufig einem Raster. Dieses ist nicht in jedem Fall regulär.
- Leiterbahnen auf einer Ebene des Chips haben eine Hauptorientierungsrichtung. Um Kreuzungsprobleme zu vermeiden, haben Leiterbahnen benachbarter Ebenen häufig orthogonale Orientierungen. Diese Eigenschaft kann in einigen Fällen ausgenutzt werden, um im Bildhintergrund erkannte Leiterbahnen herauszufiltern. Allerdings ist dies als eindeutiges Kriterium nicht geeignet.
- Eckstücke können scharfe Kanten aufweisen, abgerundet sein oder aus diagonalen Segmenten bestehen. In Ausnahmefällen basieren sie auf Winkeln, die kein Vielfaches von 45° betragen. Dies spräche gegen einen Ansatz, Leiterbahnen dadurch zu erkennen, dass ein Algorithmus entlang von Rasterlinien operiert und Pixel auswertet.

Keines der genannten Merkmale wäre als alleiniges Kriterium für eine Leiterbahnerkennung geeignet. Es ist fast auszuschließen, dass es ein einfaches Verfahren gibt, mit dem beliebiges Bildmaterial mit hohen Erkennungsraten und geringer Fehlerkennung auswertbar ist. In Degate ist ein Verfahren für die Erkennung von Leiterbahnen implementiert, dass sich auf einen allgemeinen Fall beschränkt. Es kann in Grenzen auch mit verrauschten Bilddaten umgehen. Spezialisierte Erkennungsverfahren können über den Skript-Mechanismus angebunden werden (siehe Abschnitt 8.4).

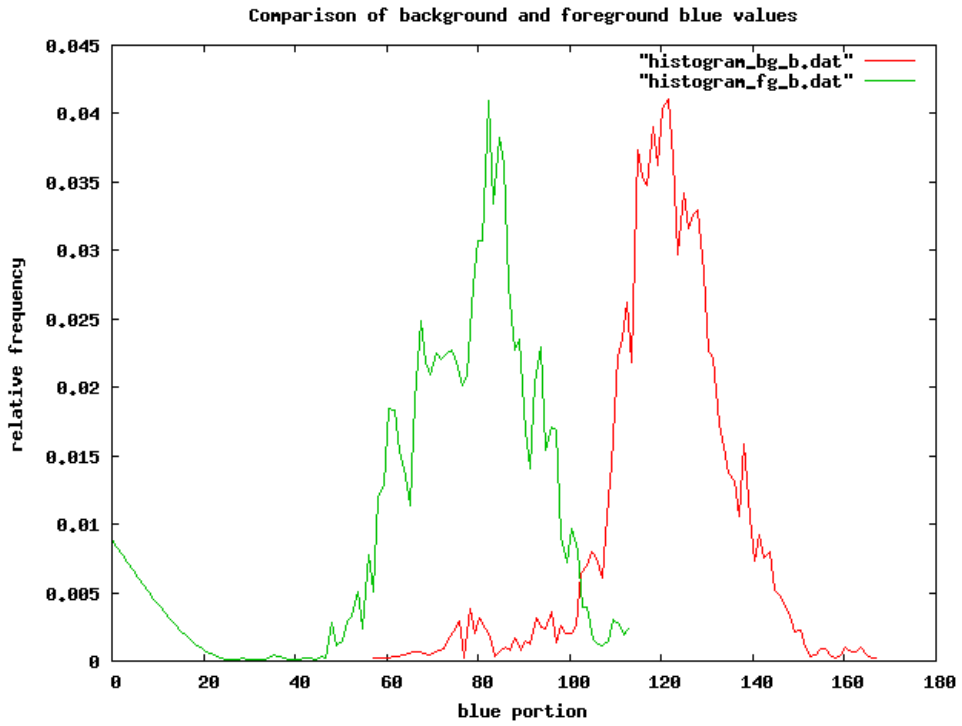

Abbildung 8.7: Histogramm: Blaukanalanteile von Leiterbahnen im Vorder- und Hintergrund (Chip: DECT, konfokalmikroskopische Aufnahme). Die diskreten Messwerte wurde zugunsten einer bessere Darstellung verbunden.

8.3 Algorithmus für die Leiterbahnerkennung

Der Algorithmus zur Erkennung von Leiterbahnen versucht, die zwei parallelen Kanten einer Leiterbahn zu finden. Dies hat sich als stabiler Ansatz erwiesen. Der Algorithmus besteht aus vier Schritten: die Vorverarbeitung der Bilddaten, die Anwendung eines Kantenfilters, die Analyse des Kantenbildes und die Transformation in Vektordaten. Als Eingabedatum dient ein Bild im RGBA-Modus. Ausgabedaten sind Leiterbahnen in vektorisierter Form. Der zu analysierende Bereich wird mit der Maus in der GUI selektiert und das Verfahren wird über das Menü gestartet. In einem Dialog können dann die Parameter eingestellt werden (siehe Abbildung 8.8) In diesem Dialog werden vier Parameter abgefragt. Der wichtige Parameter ist der Leiterbahndurchmesser: Der Algorithmus detektiert zunächst Kanten. Jede Leiterbahn weist zwei Kanten auf. Beide Kanten werden vom Verfahren betrachtet, um letztendlich eine vektorielle Beschreibung der Leiterbahn abzuleiten. Dazu muss das

Verfahren wissen, in welchem Abstand die Kanten zu erwarten sind. Dies entspricht dem Leiterbahndurchmesser. Der initiale Wert für den Durchmesser wird aus den Projekteinstellungen übernommen. Der Wert ist entscheidend für die Qualität der Erkennung. Er muss jedoch nicht pixelgenau angegeben werden.

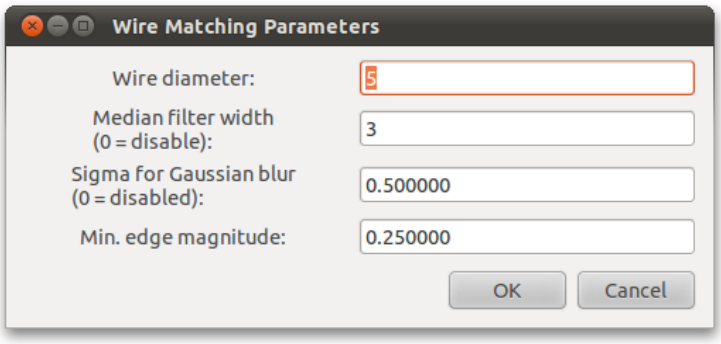

Abbildung 8.8: Dialogfenster zur Einstellung von Parametern für die Leiterbahnerkennung

Der erste Schritt des Algorithmus ist die Vorverarbeitung der Bildaten. Diese liegen im RGBA-Format vor und müssen in ein Graustufenbild umgewandelt werden. Der Algorithmus wendet auf das Graustufenbild ein Medianfilter an. Ein Medianfilter ist zur Entfernung von Rauschen geeignet. Der Vorteil gegenüber anderer Filter besteht in seiner Kantenerhaltung. Das Graustufenbild wird dann auf das Intervall $[0, 1]$ normalisert. Anschließend kann ein Gauss-Filter auf die Bilddaten angewendet werden. Die Anwendung dieses Filters ist i. d. R. nicht notwendig. Es ist lediglich für Experimente vorgesehen. Mittels Angabe des Wertes `0` im Parameter-Dialog können Median- und Gauss-Filter wahlweise deaktiviert werden.

Das vorverarbeitete Bild wird im nächsten Schritt mit Hilfe eines Sobel-Filterkerns gefaltet und in ein Kantenbild überführt (vgl. Abbildung 8.9 (b)). Jedes Leiterbahnsegment besteht aus zwei Kanten, wobei eine Kante lediglich stärkere Änderung der Intensität benachbarter Pixel darstellt. An den benachbarten Punkten mit höchstem Intensitätsunterschied treten im Kantenbild Maxima bzw. Minima auf. Mittels eines horizontalen und eines vertikalen Scans über das Kantenbild werden diese Maxima und Minima detektiert. Zu jedem Maximum müsste es ein Minimum geben, da eine Leiterbahn zwei Kanten aufweist. Bei einem Scan wird ermittelt, in welchem Abstand ein Maximum und Minimum liegen. In einem neuen Bild wird in die Mitte zwischen den Extremen ein Punkt eingetragen, wenn der Abstand etwa dem Leiterbahndurch-

messer entspricht und der Betrag beider Extremwerte überhalb eines Schwellwertes liegt. Dieser Schwellwert ist der vierte Parameter im Dialogfenster.

Unter Benutzung der beiden Extremwerte ist es möglich, zum einen die Mitte zwischen den Kantenverläufen zu finden und zum anderen zu entscheiden, zu welcher Leiterbahn eine Kante gehört. Dies ist beispielsweise relevant, wenn mehrere Leiterbahnen und dadurch mehrere Kanten parallel verlaufen. Wenn bei den Scans an einigen Stellen keine passenden Kantenmerkmale gefunden werden, z. B. weil eine der Kanten durch Bildstörungen kaum ausgeprägt ist, ist dies kein Problem. In dem Verfahren existieren zwei Schritte, die kleinere Fehler reparieren. Eine Korrekturmöglichkeit besteht in dem anschließenden Schritt der morphologischen Bearbeitung (vgl. [Bly+93]). Weitere Fehler werden durch die Vektorisierung korrigiert.

Das Ergebnis der Scanvorgänge ist ein transformiertes Bild, in dem Punktmengen Leiterbahnsegmente repräsentieren. Dieses Bild ist ähnlich zu dem in Abbildung 8.9 (c). Um kleinere Lücken in den Punktmengen zu schließen und Kanten zu glätten wird eine Close-Operation auf das Bild ausgeführt. Dabei handelt es sich um ein morphologisches Makro, welches aus einer Dilatations- und einer anschließenden Erosions-Operation besteht. Bei der Dilatation wird ein Punkt gesetzt, wenn wenigstens ein Punkt in einer 3x3-Umgebung um den betrachteteten Pixel gesetzt ist. Bei der Erosion wird ein Punkt gelöscht, wenn in der betrachteten 3x3-Umgebung kleiner gleich drei Pixel gesetzt sind. Diese 3x3-Umgebung entspricht dem sogenannten strukturierenden Element. Die Größe ist in Degate derzeit fix. [Erh08, S. 163ff]

Nach der Close-Operation wird das Bild skelettiert. Mit diesem Schritt sollen mehrere Pixel starke Kanten so ausgedünnt werden, dass die Kanten durch feine „Linien“ dargestellt werden. In Degate ist dazu der Algorithmus von Zhang und Suen implementiert, der „einer der besten Algorithmen“ [Erh08, S. 202] für die Skelettierung sei. Letztendlich ist der Algorithmus einfach zu implementieren und erfüllt in Degate seinen Zweck.

Bei der Transformation des pixelbasierten Bildes in Leiterbahnsegmente kommt ein Verfahren zum Einsatz, das Ideen von Dimitri Lagunovsky und Sergey Ablameyko aufgreift [LA99] [LA97].[1] Das Verfahren besteht darin, das pixelbasierte Bild zunächst in lineare Primitiven zu zerlegen. Eine lineare Primitive ist eine Menge von horizon-

[1] Die beiden Autoren haben ihren Publikationen nach zu urteilen an einem System gearbeitet, um Bilder von Chips automatisch auszuwerten. Dafür sprechen insbesondere die Publikationen [LAK96] und [LAK98]. Es war nicht in Erfahrung zu bringen, was aus diesem Projekt wurde und ob der Hauptanwendungszweck im Reverse-Engineering besteht oder mehr im Themenbereich VLSI-Fehlerdetektion.

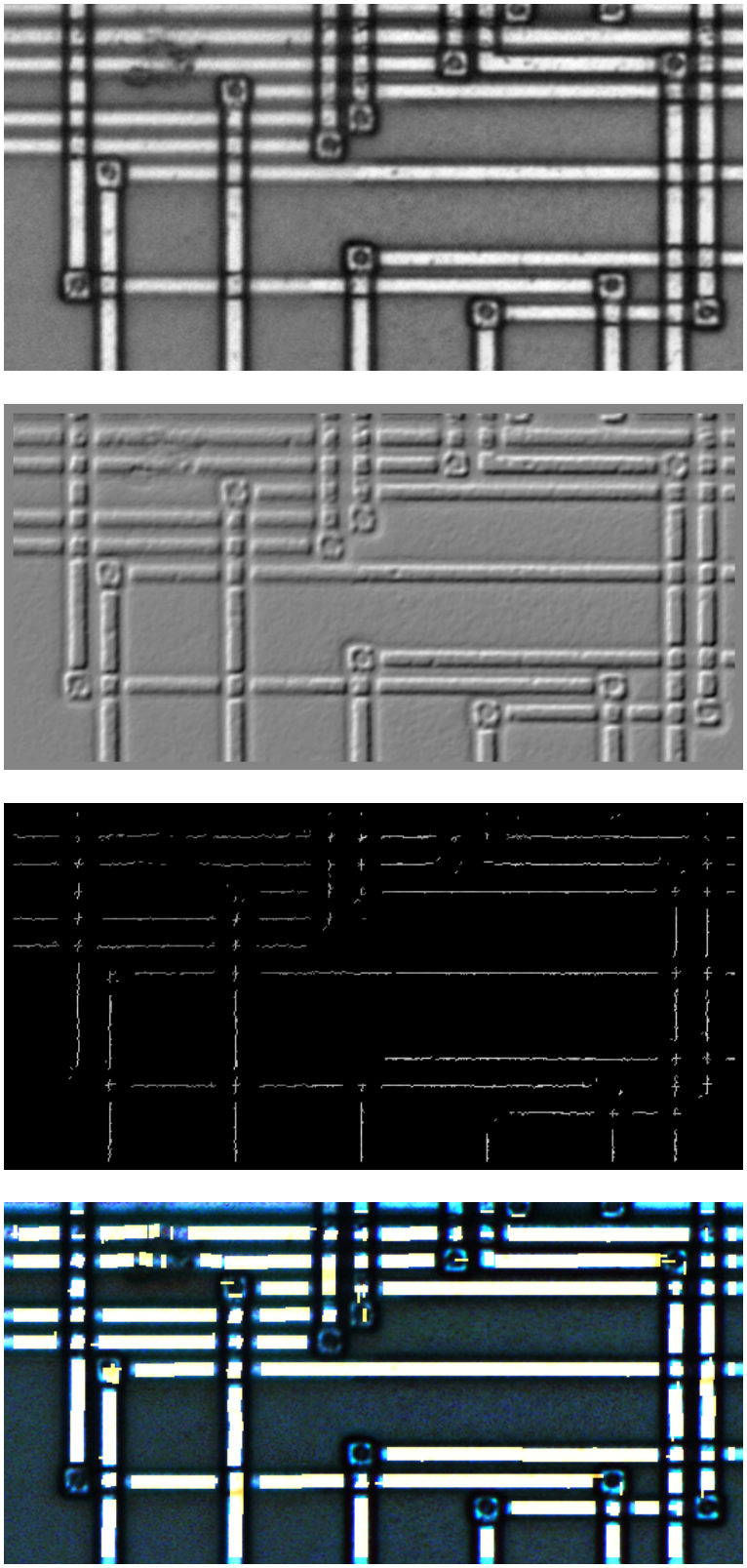

Abbildung 8.9: Einzelne Schritte bei der Erkennung von Leiterbahnen. Von oben nach unten: (a) Eingangsbild, (b) Kantenbild, (c) Ergebnis der morphologischen Close-Operation, (d) eingezeichnete Leiterbahnen (weiß, Hintergrund abgedunkelt)

tal oder vertikal benachbarten Pixeln. Diese werden mittels eines Scan-Vorganges ermittelt. Lineare Primitiven haben Start- und Endkoordinaten und implizit eine Orientierung (horizontal bzw. vertikal). In einem weiteren Schritt werden lineare Primitiven zu Liniensegmenten zusammengefügt. Dazu muss die Bedingung gelten, dass zwei lineare Primitiven die gleiche Orientierung aufweisen und der Abstand beider Primitiven sowohl in Orientierungsrichtung als auch orthogonal zu Orientierungsrichtung bestimmte Schwellwerte nicht überschreitet. Ein iterativer Prozess überführt somit die linearen Primitiven in Liniensegmente, die dann als Leiterbahnstücke in das Logikmodell eingetragen werden (vgl. Abbildung 8.9 (d)).

8.4 Anbindung von Skripten

Es besteht die Möglichkeit, eigene Skripte für die Erkennung von Leiterbahnen und Durchkontaktierungen aus Degate heraus aufzurufen. Dies dient dem Zweck, Prototypen für die Erkennung zu testen. Die Funktion wird aus der GUI heraus gestartet. Dies öffnet ein Dialogfenster, in dem ein Programmname abgefragt wird. Das Programm wird dann mit folgenden Parametern aufgerufen:

```
command --image <image-file> --results <result-file>
        --start-x <min-x> --start-y <min-y>
        --width <width> --height <height>
```

Der Parameter `image-file` spezifiziert das Bild mit den Abmessungen `width` und `height`, auf dem die Erkennung erfolgen soll. Dies ist entweder die gesamte Chipebene oder ein Ausschnitt, je nachdem ob der Benutzer einen speziellen Bereich mit der Maus ausgewählt hat. Die Parameter `min-x` und `min-y` bezeichnen die Position der linken oberen Ecke des Bildbereiches innerhalb der Chipebenen. Das Programm schreibt die Analyseergebnisse in eine Textdatei, die mittels `result-file` übergeben wird. Das Format der Ergebnisdatei sei exemplarisch wie folgt beschrieben:

Listing 8.1: Beispielinhalt einer Datei mit Analyseergebnissen

```
#
# Sample file for matching results.
#

# Wire: (format: "wire" x1 y1 x2 y2 diameter)

wire 10 23 100 23 5
```

```
8
  # Via: (format: "via" x y diameter direction)
10
  via 2 3 5 up
12 via 42 23 5 down
```

Die Ergebnisdatei wird von Degate gelesen und interpretiert. Entsprechende Objekte werden in das Logikmodell eingetragen. Eine Angabe der Chipebene ist nicht vorgesehen, da in Degate die Information existiert, auf welche Chipebene sich die Angaben beziehen.

8.5 Erkennung von Durchkontaktierungen

Durchkontaktierungen verbinden Leiterbahnen verschiedener Chipebenen miteinander. Um eine vollständige Netzliste zu erhalten, müssen die Durchkontaktierungen im Bildmaterial erkannt und in das Logikmodell eingetragen werden. Im Logikmodell ist eine Durchkontaktierung mit einer Richtungsinformation ausgestattet. Dies bedeutet, dass eine Durchkontaktierung auf einer Schicht des Chips platziert ist und entweder eine Verbindung mit der darüber- oder der darunterliegenden Chipebene herstellt. Degate unterstützt die manuelle Eintragung und die automatische Erkennung von Durchkontaktierungen.

Um eine „gültige“ Durchkontaktierung in Degate zu erstellen, müssen zwei Durchkontaktierungen für die zu verbindenden Schichten eingetragen werden. Die manuelle Platzierung und die automatische Erkennung tragen lediglich eine Durchkontaktierung auf der aktuell bearbeiteten Chipebene ein. In Abschnitt 8.1 wurde beschrieben, dass das Eintragen von Leiterbahnen, deren elektrisches Verbinden innerhalb sowie zwischen Chipebenen separate Arbeitsschritte sind. Für diese Interlayer-Verbindungen prüft Degate, ob auf zwei benachbarten Ebenen[2] gegensätzlich orientierte Durchkontaktierungen platziert sind. Ist dies der Fall, stellt Degate eine elektrische Verbindung her. Diese Symmetrieanforderung erlaubt es, fehlerhaft platzierte Durchkontaktierungen einfacher zu erkennen[3], denn die Wahrscheinlichkeit zweier fehlplatzierter Durchkontaktierungen auf benachbarten Ebenen an der gleichen Stelle mit gegenseitiger Orientierung ist vergleichsweise gering.

[2] Im „Layer Configuration“-Dialog deaktivierte Ebenen werden dabei ignoriert.
[3] Für manuell platzierte Durchkontaktierungen könnte automatisch das Gegenstück auf der benachbarten Chipebene eingetragen werden. Diese Funktion ist jedoch noch nicht in Degate integriert.

Bei der automatischen Erkennung von Durchkontaktierungen treten ähnliche Probleme auf wie bei der Leiterbahnerkennung. Die Bildqualität hat sogar einen größeren Einfluss auf die Erkennungsrate. Bildstörungen wirken sich stärker auf die Erkennung der kreisförmigen Objekte aus, während die Leiterbahnerkennung unempfindlich gegenüber kleinen Störungen ist, da genügend weitere Merkmale vorhanden sind.

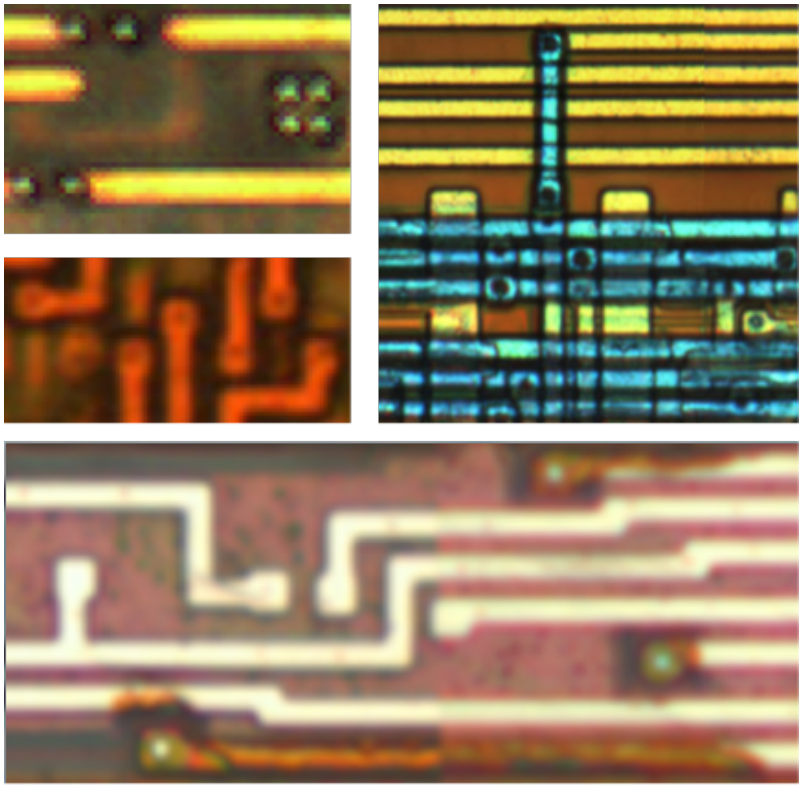

Abbildung 8.10: Beispiele für Durchkontaktierungen

Abbildung 8.10 zeigt einige Bildbeispiele verschiedener Qualitätsstufen. In fast allen Fällen sind Durchkontaktierungen anhand heller bzw. dunkler Punkte feststellbar. Diese gehen oft einher mit Leiterbahnverdickungen. Naturgemäß weisen Enden einer Leiterbahn Durchkontaktierungen auf, wenngleich bei verschiedenen Reverse-Engineering-Projekten auch „offene“ Leiterbahnenden gefunden wurden. Durchkontaktierungen können auch nicht benachbarte Ebenen verbinden. Dies ist in Abbildung 8.10 oben links zu sehen. Durchkontaktierungen zu höheren Schichten sind zumeist als helle Punkte wahrnehmbar. Dies ist darin begründet, dass sie beim Polieren der Chipoberfläche zuerst angegriffen werden. Da sie aus Metall bestehen, reflektieren sie dann bei Auflichtmikroskopie das Licht stärker als die noch nicht polierten Leiterbahnen. Entsprechend weisen dunkle Punkte auf Durchkontaktierungen zu tiefer

gelegenen Schichten hin. Mitunter treten Stellen auf, an denen Durchkontaktierungen infolge des Polierens zerstört wurden. Dies ist in Abbildung 8.10 unten dargestellt.

Ein naheliegender Ansatz für die Erkennung von Durchkontaktierungen ist die Detektion der punktförmigen Objekte im Bildmaterial. Grundsätzlich können auch die Leiterbahnverdickungen festgestellt werden, jedoch ist deren Form oft vom Chip und vom weiteren Verlauf der Leiterbahn abhängig. Eine Schwierigkeit besteht darin, diese theoretischen Möglichkeiten als GUI-Funktion umzusetzen. Dazu sind drei Fälle zu unterscheiden. Zu einem wäre es möglich, eine Erkennung ausschließlich auf in Textform zu erfassenden Parametern aufzubauen. Dies wäre beispielsweise mit einer Kreiserkennung auf der Basis einer generalisierten Hough-Transformation möglich, für die als relevanten Parameter ggf. der Radius der Durchkontaktierungen in Textform abzufragen wäre[4]. Zum anderen kann eine Erkennung anhand von Positivbeispielen erfolgen. In Degate könnten dazu Durchkontaktierungen von Hand eingezeichnet werden, die dann als Referenz dienen. In einer dritten Form ließen sich ferner Negativbeispiele aufnehmen. Für bestimmte Klassifikationsverfahren, etwa neuronale Netze [5] und „Support Vector Machines“, sind Negativbeispiele für Trainingszwecke unablässlich. Jedoch müssten diese Negativbeispiele in der GUI verwaltet werden und es müsste dem Benutzer vermittelbar sein, was geeignete Negativbeispiele sind, um Klassifikatoren zu trainieren. Daher wurde dieser dritte Ansatz aus Praktikablitätsgründen verworfen. Entsprechende Überlegungen gelten gleichsam für die Erkennung von Leiterbahnen.

In einem frühen Prototypen von Degate war ein Ansatz für die Erkennung von Durchkontaktierungen implementiert, der die oben beschriebenen hellen Punkte auf dem Bildmaterial von RFID-Transponder des Typs „Mifare classic“ erkennt. Das Verfahren bestand darin, das Bildmaterial zunächst in Graustufen umzurechnen, zu filtern und das Bild mittels eines Schwellwertes zu binarisieren. Ein modifizierter Flood-fill-Algorithmus bestimmte minimal umgebende Rechtecke von den Flächen, deren Helligkeitswerte über dem Schwellwert lagen. Wenn die Größe des minimal umgebenden Rechtecks in der Größenordnung einer Durchkontaktierung lag, wurde eine in die Chipebenene eingezeichnet. Die Erkennungsrate betrug innerhalb der Testdaten weit über 90 %, insbesondere weil das Bildmaterial eine hohe Qualität hatte.

[4] Eine Erkennung von Durchkontaktierungen mittels der Verallgemeinerung der Hough-Transformation wurde im Rahmen dieser Diplomarbeit ausprobiert und mangels halbwegs sinnvoller Resultate verworfen.

[5] Eine Erkennung von Chipstrukturen auf der Basis von neuronalen Netzen wird in [SV02] beschrieben.

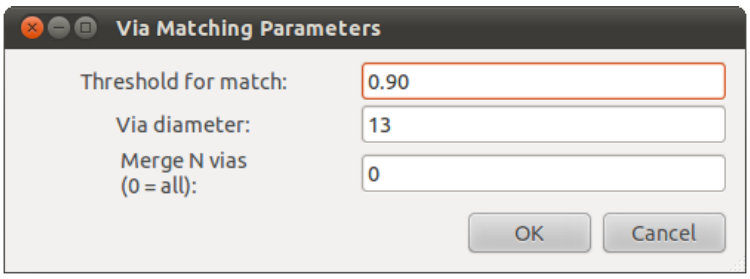

Abbildung 8.11: Dialogfenster zum Einstellen von Parametern für das Erkennen von Durchkontaktierungen.

In der aktuellen Version von Degate basiert die Erkennung von Durchkontaktierungen auf der normalisierten Kreuzkorrelation (vgl. Kapitel 7). Dies erfordert wenigstens ein Positivbeispiel in Form einer manuell platzierten Durchkontaktierung. Degate unterscheidet hier ebenfalls zwischen nach oben und nach unten gerichteten Durchkontaktierungen. Ähnlich wie bei der Standardzellenerkennung können mehrere Bilder von Durchkontaktierungen zu einer Schablone zusammengerechnet werden. Ein Dialog fragt nach der maximalen Anzahl zu mischender Bilder (siehe Abbildung 8.11). Der Durchmesser der Durchkontaktierungen darf dabei abweichen, denn für das Mischbild werden die Positionen der Zentren in Übereinstimmung gebracht. Die Auswahl einzelner Durchkontaktierungen ist zufällig. Sie müssen lediglich die gleiche Richtung (Verbindung zur darunter- oder darüberliegenden Chipebene) haben und auf der zu durchsuchenden Chipebene liegen. Die Größe von Durchkontaktierungen ist im Allgemeinen abhängig von der Chipebene. Auf höheren Schichten sind Durchkontaktierungen häufig größer. Da sich die Auswahl auf die aktuellen Ebene bezieht, wird dieser Umstand berücksichtigt.

Für die Benutzung der automatischen Erkennung ist es günstig, zunächst jeweils nur eine Durchkontaktierung pro Orientierung zu setzen und auf einem kleinen Teil der Chipebene weitere suchen zu lassen. Zwar können initial mehrere Durchkontaktierungen platziert werden. Deren Mischbild wird jedoch einen höheren Fehler in Bezug auf Deckungsgleichheit aufweisen.

In [MN08] beschreibt Giedrius Masalskis sowohl der Ansatz der „blob detection“ als auch ein auf Kreuzkorrelation aufsetzendes Verfahren und vergleicht die Erkennungsraten für vier verschiedene Testbilder. Beide Verfahren nutzen mediangefilterte Eingabebilder. Er kommt zu dem Ergebnis, dass eine simple Erkennung heller Punkte mittels Schwellwertverfahren zu hohen Erkennungsraten führt. Lediglich für ver-

rauschte Bilddaten stellt die Kreuzkorrelation einen besseren Ansatz dar. Im Gegensatz zu dem in Degate implementierten Verfahren detektiert Masalskis die Größe der Durchkontaktierungen. In einem finalen Schritt verwirft er dann falsch-positive Treffer. Diese werden anhand von Desing Rule Checks herausgefiltert. Für diese Regeln, die Parameter der Algorithmen sind, werden die Größe der Durchkontaktierungen und ihr Abstand herangezogen.

In Kapitel 7.3 wurde festgestellt, dass bei der Nutzung der Kreuzkorrelation für die Erkennung von Mustern sich kein Vorteil ergibt, wenn die Eingabedaten mediangefiltert werden. Im Gegenteil – die Erkennungsraten verringern sich sogar. Dieser Zusammenhang wird bei Masalskis nicht untersucht. Im Rahmen dieser Diplomarbeit konnte auch nicht geklärt werden, ob es sich um einen Zufallsbefund handelt, dies abhängig von der Templategröße ist oder sonstige Gründe dafür vorliegen. Ferner hat sich anhand verschiedener Experimente mit Degate herausgestellt, dass die Erkennungsrate von der Schablonengröße abhängt. Masalskis untersucht Durchkontaktierungen in der Größe von 6x6 bis 15x15 Pixeln. Sowohl bei der Erkennung von Durchkontaktierungen als auch Standardzellen mit Degate haben große Muster weniger falsch-positive Treffer bewirkt. Bei der Standardzellensuche ist dies kein ernsthaftes Problem, da die Anzahl der „Messwerte“ vergleichsweise groß ist. Bei kleinerer Anzahl von zu vergleichenden Pixeln vergrößert sich das Konfidenzintervall. Für kleine Suchmuster, etwa bei der Suche von Duchkontaktierungen, ist dies kritisch.

8.6 Evaluation der Erkennung von Durchkontaktierungen

Die Detektion von Durchkontaktierungen mittels der normalisierten Kreuzkorrelation wurde evaluiert. Dazu wurde das in Abbildung 8.12 dargestellte Bild verwendet. Dieses Bildmaterial ist vergleichsweise repräsentativ: Es enthält einige selbst für Menschen schwer erkennbare Duchkontaktierungen. Ferner kreuzen sich Leiterbahnen benachbarter Schichten so, dass diese Kreuzungen fast wie punktförmige Objekte aussehen und von einem Algorithmus fälschlich als Durchkontaktierungen gewertet werden können. Das Aussehen der Durchkontaktierungen ist nicht besonders homogen. An anderer Stelle auf dem Chip sind Speicherbänke vorhanden. Dort sehen die Durchkontaktierungen fast identisch aus, so dass der Algorithmus annähernd alle korrekt erkennt.

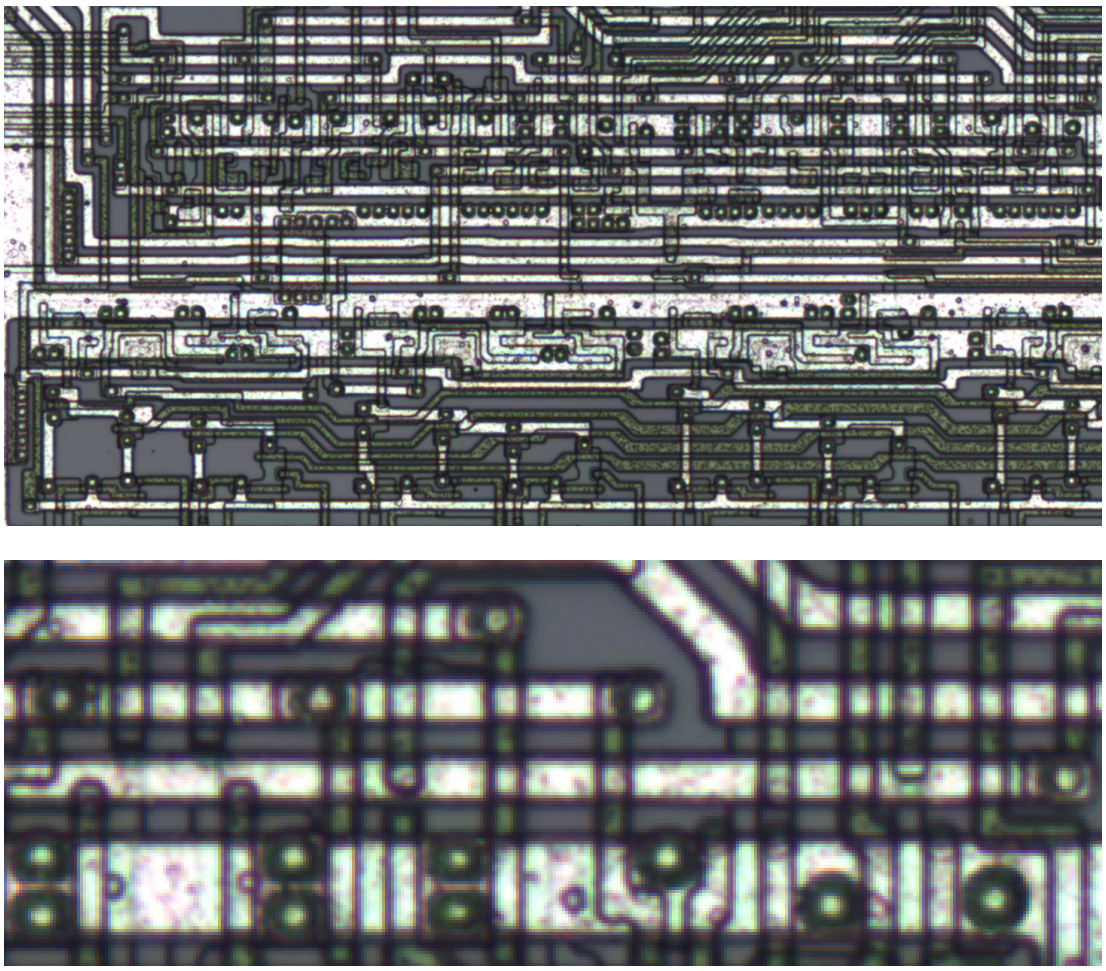

Abbildung 8.12: Testmaterial für die Evaluation der Durchkontaktierungserkennung: Das obere Bild enthält 223 Durchkontaktierungen. Das untere Bild ist eine Vergrößerung der oberen rechten Ecke. Es zeigt 14 Durchkontaktierungen. (Chiptyp: Motorola 68000, Quelle: visual6502.org)

Die Größe des Suchmusters beträgt 16x16 Pixel. Es wurde ein Referenzdatensatz mit 223 Durchkontaktierungen erstellt. Für den Vergleich wurde initial eine „geeignete" Durchkontaktierung gewählt. Für ausgewählte Korrelationsschwellwerte t_{det} suchte Degate in maximal fünf Suchrunden weitere Durchkontaktierungen. Wenn Fehlerkennung auftreten, verringert sich mit jeder weiteren Iteration die Güte des Suchmusters, da für jeden Suchlauf alle bereits gefundenen Durchkontaktierungen zu einem Suchmuster vereint werden. Nach jedem Suchlauf wurde das Degate-Projekt gespeichert und dessen Logikmodell mit dem Referenzprojekt verglichen. Der Vergleich erfolgte wie bei der Evaluation der Standardzellensuche mit Hilfe eines Perl-Skriptes. Dieses ist ebenfalls im git-Repository zu finden.

t_{det}	Treffer	korrekte Treffer	korrekte Treffer [%]	Fehler	Fehler [%]
0.95	124	124	56	0	0.00
0.90	183	183	82	0	0.00
0.875	226	223	100	3	1.33
0.85	234	223	100	11	4.70

Tabelle 8.1: Erkennungsraten und Fehlerkennungsraten für die Detektion von Durchkontaktierungen mit einer Schablonengröße von 16x16 Pixeln.

In Tabelle 8.1 sind die Ergebnisse in Abhängigkeit vom Schwellwert dargestellt. Dieser Schwellwert ist der kleinste Korrelationskoeffizient, der für eine Erkennung akzeptiert wird. Die Spalte „Treffer“ gibt die Anzahl aller vom Algorithmus gefundenen Durchkontaktierungen an. Der Wert setzt sich aus Fehlerkennungen (Spalte „Fehler“) und korrekten Erkennungen (Spalte „korrekte Treffer“) zusammen.

t_{det}	Treffer	korrekte Treffer	korrekte Treffer [%]	Fehler	Fehler [%]
0.95	191	162	73	29	15.18
0.90	495	211	95	248	54.03

Tabelle 8.2: Erkennungsraten und Fehlerkennungsraten für die Detektion von Durchkontaktierungen mit einer Schablonengröße von 10x10 Pixeln.

Die Tabelle 8.2 zeigt die Ergebnisse des Experiments für eine Größe des Suchmusters von 10x10 Pixeln. Dies ist die kleinste Schablonengröße, die für das Testbild möglich ist. Der Duchmesser der hellen Punkte in den Durchkontaktierungen beträgt etwa acht Pixel und es ist notwendig, dass das Muster auch Merkmale der Umgebung enthält. Aus den Werten ergibt sich, dass zwar mehr Durchkontaktierungen gefunden werden, andererseits viele Treffer falsch sind. So sind für den Schwellwert $t_{det} = 0.90$ mehr als die Hälfte aller Treffer keine Durchkontaktierungen.

Zusammenfassend lässt sich feststellen, dass die normalisierte Kreuzkorrelation für die Erkennung von Durchkontaktierungen geeignet ist, sofern hinreichend große Muster bei der Suche Verwendung finden. Daher sollte bei der Benutzung von Degate in den Projekteinstellungen ein möglichst großer Durchmesser für Durchkontaktierungen eingetragen werden, denn die Größe des Suchmusters hängt von den bereits auf der Chipebene platzierten Durchkontaktierungen ab.

Kapitel 9

Code-Generierung, Netzlistenexport und Remodularisierung

Ziel von Degate ist es nicht nur, statische Netzlisten zurückzugewinnen, sondern auch Verhaltensbeschreibungen von Chips bzw. Teilen von Chips zu erzeugen. Diese Verhaltensbeschreibungen können zur Simulation verwendet werden. Damit wäre es möglich, in einer Simulation dynamische Eigenschaften eines Chips zu untersuchen. Darüber hinaus ist die Synthese von Schaltkreisen denkbar, so dass funktional äquivalente Schaltungen auf einem FPGA instanziiert werden können. Die mittels Degate rekonstruierte Schaltung ist zunächst nur funktional äquivalent. Spezifische physikalische Parameter werden nicht erhoben, zumal sie durch optische Inspektion i. d. R. nicht festgestellt werden können – abgesehen von Leiterbahndurchmessern und -längen.

Degate unterstützt für die Beschreibung von Hardware Verilog und VHDL, wobei die Unterstützung von Verilog fortgeschrittener ist. Ursprünglich wurden Sprachen wie VHDL und Verilog zur Beschreibung und Simulation von Hardware entwickelt. Beide Sprachen stammen aus den 1980er Jahren. Während Verilog als Entwicklung in der Privatwirtschaft startete, wurde VHDL vom U.S. Department of Defense initiiert, um Anwendungsspezifische Integrierte Schaltungen (ASICs) beschreiben zu können. Beide Sprachen sind vom Institute of Electrical and Electronics Engineers (IEEE) in mehreren Sprachversionen standardisiert. Mittlerweile dienen diese Hardwarebeschreibungssprachen zum Synthetisieren von Schaltkreisen. Wenngleich es andere Beschreibungssprachen gibt, sind VHDL und Verilog am weitesten verbreitet.

9.1 Verhaltens- und Strukturbeschreibungen

Ein Modul in VDHL bzw. Verilog ist eine Beschreibung eines Bausteins in einer Designhierarchie.[1] Ein Modul kann eine Verhaltens- oder Strukturbeschreibung darstellen. Die Strukturbeschreibung definiert, aus welchen Untermodulen ein Modul besteht und wie diese miteinander verschaltet sind. Die Verhaltensbeschreibung gibt an, wie ein Modul implementiert ist. Ein Design-Paradigma bei der Entwicklung von Schaltkreisen mittels Hardwarebeschreibungssprachen ist, dass Verhaltens- und Strukturbeschreibungen nicht innerhalb von Modulen gemischt werden. Vielmehr soll in einem Modul entweder dessen Verhalten beschrieben werden oder dessen Struktur. Die Vermischung von Struktur- und Verhaltenbeschreibungen ist lediglich bei Testprogrammen üblich. Dabei handelt es sich um Programmcode, ebenfalls in Verilog bzw. VHDL, der ein zu testendes Modul instanziiert, dessen Moduleingänge mit Werten belegt und dessen Ausgabewerte prüft.

Ein Schaltungsentwurf entspricht einer baumartigen Designhierarchie von strukturellen und funktionalen Modulen. Auf den Blattknoten sind i. d. R. die Verhaltensbeschreibungen angesiedelt. Die zur Wurzel gerichteten Schichten bilden Strukturbeschreibungen. Dies sei an einem einfachen Verilog-Beispiel eines Flipflops und der Zusammenfassung mehrerer Flipflops zu einem Schieberegister verdeutlicht. Der Programmquelltext 9.1 beschreibt zunächst das Verhalten eines Flipflops mit asynchronem Reset:

Listing 9.1: Verhaltensbeschreibung eines Flipflops

```
1 module flipflop(input d, input clk, input rst, output reg q);

3         always @(posedge clk or posedge rst)
          if(rst)
5                 q <= 0;
          else
7                 q <= d;

9 endmodule
```

Der Verilog-Code 9.2 baut auf der Verhaltensbeschreibung des Flipflops auf und verbindet drei Flipflops zu einem Schieberegister. Die Eingangssignale des Schieberegistermoduls für den Takt und Reset werden mit den entsprechenden Ports der Flipflops verbunden, so dass diese Signale gleichzeitig anliegen. Die Verilogbeschrei-

[1] Der hier genutzte Begriff „Modul“ stammt aus Verilog und entspricht dort dem Schlüsselwort „module“. In VHDL entspricht dies einer „entity“.

bung verwendet zwei zusätzliche Leiterbahnen, um den Datenpfad zu konstruieren. Abbildung 9.1 zeigt die Verschaltung der Flipflops im Schieberegistermodul auf der Register-Transfer-Ebene.

Listing 9.2: Beispiel einer Strukturbeschreibung: Schieberegister bestehend aus drei Flipflops

```
1 module shiftreg(input d, input clk, input rst, output q);

3         wire w1, w2;

5         flipflop ff0(.d(d), .clk(clk), .rst(rst), .q(w1));
          flipflop ff1(.d(w1), .clk(clk), .rst(rst), .q(w2));
7         flipflop ff2(.d(w2), .clk(clk), .rst(rst), .q(q));

9 endmodule
```

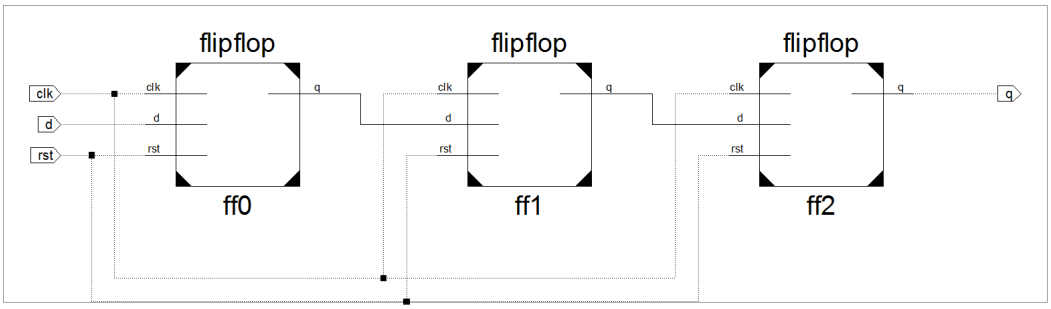

Abbildung 9.1: Schaltplan des Schieberegisters

Diese Aufteilung in Verhaltens- und Strukturbeschreibung existiert ebenfalls in Degate. Dabei müssen die Verhaltensbeschreibungen manuell eingetragen werden. Die Struktur ergibt sich implizit aus dem Logik-Modell und kann als Netzliste exportiert werden.

9.2 Code-Gerüste für Verhaltensbeschreibungen

Da die Standardzellenfestlegungen in Degate vorliegen, können diese Informationen benutzt werden, um Code-Gerüste für die Verhaltensbeschreibung zu erzeugen. Dazu wird im entsprechenden Dialog in Degate der Eintrag „Logic class“ festgelegt (vgl. Abschnitt 3.6). Ist diese Klasse beispielsweise ein Inverter, kann Degate anhand des Namens des Ein- bzw. Ausgangs passenden Code für einen Inverter erzeugen. In

einfachen Fällen ist dieser direkt kompilierbar. Soweit Code-Gerüste für die Verhaltensbeschreibung von Standardzellen existieren, benutzen diese ausschließlich synthetisierbare Sprachkonstrukte.

Einfache Gatter, wie z. B. ein Inverter oder elementare Logikgatter, haben oft nur einen Ausgang und wenige Eingänge. Unter den Eingängen eines Gatters, z. B. bei einem AND-Gatter, liegen keine ausgezeichneten Steuersignale vor, d. h. kein Eingang, der für die Schaltfunktion eine spezielle Bedeutung hat. Dies wäre bei einem Tristate-Inverter der Fall. Dieser verfügt über einen zusätzlichen Eingang, mittels dessen die Invertierung aktiviert bzw. deaktiviert wird. Bei Deaktivierung ist der Ausgang des Tristate-Inverters in einem hochohmigen Zustand, der weder durch die Potentiale „low“ noch „high“ repräsentiert wird, sondern einen dritten Zustand darstellt (daher „tri-state“).

Bedeutung des Ports	**Portnamen**
clock	clk, clock
reset	rst, reset, !rst, !reset, /rst, /reset
enable	en, enable, !en, !enable, /en, /enable
select	sel, select, !sel, !select, /sel, /select
q	q
!q	!q, /q
d	d

Tabelle 9.1: Spezielle Ports, die von Degate anhand des Portnamens erkannt werden.

Degate erkennt gängige Portnamen. Diese sind in Tabelle 9.1 aufgeführt. Wenn diese Portnamen verwendet werden, kann Degate passendere Code-Gerüste generieren. Diese Code-Gerüste sind letztendlich nur Gerüste. Es ist nicht Ziel, Code für alle Standardzellen vollständig automatisiert zu erzeugen. Allgemein ist dies auch nicht möglich. Das verfolgte Ziel der Code-Generierung ist, für gängige Standardzellen soweit Code zu erzeugen, so dass dieser innerhalb kurzer Zeit angepasst werden kann. Der wesentliche Teil einer VHDL- bzw. Verilog-Beschreibung übersteigt für gängige Standardzellen selten fünf Zeilen Code.

Degate generiert für die in der folgenden Tabelle genannten Standardzellentypen Code-Gerüste. Der Grad der Vollständigkeit ist mittels Symbolen angegeben. Das Pluszeichen bedeutet, dass kein vollständiger Code für den Standardzellentypen generiert wird. Aus der letzten Spalte ergeben sich die Portnamen, die Degate als spezielle Funktionsports erkennt.

Standardzelle	Verilog	VHDL	Funktions-Ports
Inverter	✓	✓	
Tristate-Inverter (lo-active)	✓		enable
Tristate-Inverter (hi-active)	✓		enable
nand	✓	✓	
nor	✓	✓	
and	✓	✓	
or	✓	✓	
xor	✓	✓	
xnor	✓	✓	
Buffer	✓	✓	
Tristate-Buffer (lo-active)	✓		enable
Tristate-Buffer (hi-active)	✓		enable
Latch (sync-enable)	✓		
Latch (async-enable)			
Flipflop	✓		d, q, clock (, !q)
Flipflop-sync-rst	✓	+	d, q, clock, reset (, !q)
Flipflop-async-rst	✓	+	d, q, clock, reset (, !q)
Kombinatorische Logik (allg.)	+		
AO	+		
AOI	+		
OA	+		
OAI	+		
Halbaddierer			
Mux	+		
Demux	+		

Tabelle 9.2: Unterstützung für die Code-Gerüst-Generierung in Verilog und VHDL.

Code-Gerüste für Standardzellenbeschreibungen werden durch C++-Code erzeugt. Dies ist hinsichtlich der Erweiterbarkeit unflexibel. Vorstellbar wäre, für die Generierung ein Template-System zu verwenden. Falls sich dies als nicht mächtig genug erweisen würde, wäre die Anbindung von Skriptsprachen denkbar, z. B. mittels des Frameworks Boost.Python.

9.3 Transistorbasierte Beschreibung

Verilog unterstützt Verhaltensbeschreibungen auf der Basis von Transistoren [Lee03, S. 27ff]. Dies ist die einfachste Ebene, auf der ein „Gatterverhalten“ in Verilog beschrie-

ben werden kann. Genau genommen ist dies dann keine Verhaltensbeschreibung, sondern eine strukturelle Beschreibung. Das folgende Code-Beispiel illustriert diese Beschreibungsform anhand eines Inverters:

Listing 9.3: Strukturelle Beschreibung eines Inverters auf der Ebene von Transistoren.

```
module transistor_level_inverter(y, x);

   input x;
   output y;

   supply1 vdd; // predefined high potential
   supply0 gnd; // predefined potential for ground

   pmos(y, vdd, x); // (drain, source, gate)
   nmos(y, gnd, x); // (drain, source, gate)

   /* Instead of instantiating a pmos and a nmos type
      transistor a cmos module can be used.

      cmos(y, x, gnd, vdd);   */

endmodule
```

Der Verilog-Standard definiert verschiedene Transistortypen. Die im Beispiel verwendeten MOS-Transistoren leiten das Signal von der Schnittstelle Source zu Drain. Sie sind nicht bidirektional. Dies führt bei fehlerhafter Verwendung zu hochohmigen Signalen. Für bidirektionale Signale sieht der Verilog-Standard die Typen „tranif0“ und „tranif1“ vor.

Der Vorteil dieser Beschreibungsform für eine Standardzelle besteht darin, dass sie die tatsächliche Verschaltung der Transistoren wiedergibt. Somit wäre bei der Transformation eines Standardzellenbildes in eine „Verhaltensbeschreibung“ keine Interpretation der Netzliste innerhalb einer Standardzelle notwendig. Dadurch reduziert sich für komplexere Standardzellen die Wahrscheinlichkeit fehlerhafter Verhaltensbeschreibungen. Eine derartige Beschreibung auf der Basis von Transistoren könnte auch intermediäres Format des in Abschnitt 3.6 vorgeschlagenen Werkzeugs zur Rekonstruktion von Standardzellen sein.

9.4 Erzeugen von Testprogrammen

Zu jeder VHDL- bzw. Verilog-Verhaltensbeschreibung können Testprogramme in Degate hinterlegt werden. Code-Gerüste für diese Testprogramme können ebenfalls

erzeugt werden, wobei der Teil des Testcodes, der die Ausgabewerte prüft, manuell erstellt werden muss.

Verilog-basierter Testcode kann aus der GUI heraus kompiliert und ausgeführt werden. Dazu muss der Icarus Verilog Compiler[2] installiert sein. In den generierten Testcode werden Verilog-Anweisungen geschrieben, die das Exportieren von Signalformen im VCD-Format bewirken. Degate startet das Open-Source-Werkzeug GTKWave[3], um die aufgezeichneten Signalformen darzustellen.

9.5 Strukturbeschreibungen

Auf der Grundlage von Verhaltensbeschreibungen der Standardzellen bauen strukturelle Beschreibungen auf. Diese geben an, wie elementare Module miteinander verschaltet sind. Beim vorwärtsgerichteten Entwurf erstreckt sich diese Strukturbeschreibung über mehrere Designebenen, da komplexe Systeme nie als monolithische Funktionsblöcke entwickelt werden. Beim Reverse-Engineering fehlt diese Hierarchieinformation. Die Menge aller Standardzelleninstanzen bildet deshalb zunächst ein gruppierendes Hauptmodul. Degate kann die vom Hauptmodul gebildete Netzliste exportieren.

Das Hauptmodul benötigt Schnittstellen, über die Signale in das Modul hinein- und aus dem Modul herausgelangen können. Für das Hauptmodul müssen diese Schnittstellen in Degate explizit festgelegt werden. Dafür werden per Konvention EMarker eingesetzt. EMarker sind in Degate elektrisch verbindbare Objekte, die ausschließlich der Markierung dienen und die kein reales Objekt im Chip repräsentieren. Diese EMarker können z. B. auf einer Leiterbahn platziert werden, die aus dem Schaltkreis heraus in andere Bereiche des Chips führt. Beim Exportieren des Hauptmodules prüft Degate, welche Schnittstellen von Standardzellen des Hauptmoduls mit EMarkern verbunden sind. Die entsprechenden EMarker geben Aufschluss, welche Schnittstellen für das Hauptmodul vorgesehen sind. Der Name der EMarker wird für die Modulschnittstelle verwendet. Die Beschreibung des EMarkers muss per Konvention auf „module-port“ gesetzt werden.

[2] `http://iverilog.icarus.com/`, besucht am 4. Mai 2011
[3] `http://gtkwave.sourceforge.net/`, besucht am 4. Mai 2011

9.6 Remodularisierung

Der Umgang mit Modulen, die viele Elemente enthalten, ist in der Praxis schwierig, insbesondere weil sich komplexe Module nicht sinnvoll visualisieren lassen. Ferner lässt sich monolithischer Code kaum bearbeiten, z. B. um Teile davon auf einen FPGA zu portieren. Daher soll es gezielt möglich sein, ausgewählte Teile eines Schaltkreises in einer Hardwarebeschreibungssprache zu exportieren. Dies erfordert, Teile einer Schaltung zu modularisieren.

Degate unterstützt die Modularisierung von Schaltungsteilen. Dies kann als Remodularisierung bezeichnet werden, denn im vorwärtsgerichteten Entwicklungspfad wurde die Schaltung in modularer Form konstruiert. Beim Reverse-Engineering werden Module zurückgewonnen. Es bleibt lediglich offen, ob diese mit den ursprünglich konstruierten übereinstimmen. Für das Reverse-Engineering ist diese Frage jedoch ohne Bedeutung, denn solange diese neuen Module helfen, die Netzliste zu strukturieren und damit übersichtlicher zu gestalten, erfüllen sie ihren Zweck.

Module in Degate sind ein Container für platzierte Standardzellen und bereits existierende Module. Dabei bildet das Hauptmodul die Wurzel eines Baumes. In diesem Baum sind alle anderen Module eingeordnet. In der GUI wird dieser Modulbaum dargestellt. Der Benutzer kann darin navigieren und Standardzellen einsortieren.

Jedes Modul hat Schnittstellen. Während diese für das Hauptmodul manuell festgelegt werden müssen, werden die Schnittstellen für Untermodule automatisch gewählt. Als Modulschnittstellen werden die Standardzellenports bzw. Untermodulports betrachtet, die mit Leiterbahnen verbunden sind, welche aus dem Modul heraus führen. Als Schnittstellennamen werden im Normallfall die korrespondierenden Portnamen verwendet. Die Schnittstellennamen können einzeln festgelegt werden.

9.7 Visualisieren von Netzlisten

In Anhang A.8 wird der „Connection Inspector“ vorgestellt, der zur Inspektion des Netzmodells dient. Dieser dient hauptsächlich dazu, Schaltpläne zu rekonstruieren. Dabei werden Standardzellen und deren Verschaltungen Stück für Stück auf Papier übertragen. Dies klingt zunächst aufwendiger als es tatsächlich ist. Dennoch bedarf es zukünftig geeigneter Verfahren, um Netzlisten aus Degate heraus direkt zu visualisieren. Eine entsprechende Komponente fehlt derzeit.

Die Netzliste zu einem Degate-Projekte kann als Verilog-Code exportiert werden. Dieser Code lässt sich in gängige Werkzeuge zur Schaltkreisentwicklung importieren, beispielsweise in die Entwicklungsumgebung Webpack ISE von Xilinx oder Quartus II von Altera. Beide ermöglichen die Visualisierung auf RTL-Ebene. Beispielsweise ist Abbildung 9.1 ein Export aus Webpack ISE und die Schaltkreisdarstellungen im Anhang B stammen aus Quartus II. Dabei ergeben sich folgende Nachteile: Damit beide Werkzeuge eine RTL-Verschaltungsdiagramm erstellen können, muss der Code synthetisierbar sein. Dies erfordert, dass eine Verhaltensbeschreibung der Standardzellen vorliegt. Für die Visualisierung der Struktur wäre eine Verhaltensbeschreibung jedoch nicht notwendig. Des Weiteren weisen beide Werkzeuge kaum Einstellungsmöglichkeiten für die grafische Darstellung auf, insbesondere kann auf die Platzierung von Funktionsblöcken kein Einfluss genommen werden. Dass die Schriftgröße nicht anpassbar ist, ist für gedruckte Dokumente ebenfalls problematisch.

Es existieren kommerzielle Werkzeuge für die Visualisierung auf RTL-Ebene, etwa RTLvision PRO[4] der Firma Concept Engineering GmbH. Alternativ lässt sich eine statische Darstellung der Netzliste auch mit Open-Source-Werkzeugen erstellen:

- Xsch ist der „Graphical schematic viewer“ aus dem Alliance-Paket[5]. Die Software kann strukturellen VHDL-Code visualisieren.
- Signs[6] ist eine Erweiterung für die Entwicklungsumgebung Eclipse. Dieses Plugin dient der VHDL-Entwicklung und bietet eine Funktion zur Netzlistenvisualisierung. Seit etwa vier Jahren gibt es keine Aktualisierung des Codes.
- Graphviz[7] ist eine Software zur Visualisierung von Graphen. Die Darstellung von VHDL- bzw. Verilog-Modulen auf der Basis von Graphviz war bereits Gegenstand von Untersuchungen[8]. In der `libdegate` existiert experimenteller Code zum Exportieren von Dot-Files, dem Eingabeformat für Graphviz.

[4] RTLvision PRO, `http://concept.de/rtl_index.html`, besucht am 20. April 2011
[5] Alliance VLSI CAD System, `http://www-asim.lip6.fr/recherche/alliance/`, besucht amt 20. April 2011
[6] Signs – VHDL Hardware Developement, `http://www.iti.uni-stuttgart.de/~bartscgr/signs/wiki/index.php/Main_Page`, besucht am 20. April 2011
[7] Graphviz – Graph Visualization Software, `http://www.graphviz.org/`, besucht am 20. April 2011
[8] VeriViz ist ein „Verilog Visualization Tool“. Die Software wurde von Ravi Saini im Rahmen einer Abschlussarbeit am Indian Institute of Technology Madras entwickelt. Obwohl die Abschlussarbeit und der Sourcecode um 2007 veröffentlicht wurden, sind sie nicht mehr im Internet verfügbar. Die Internet Archive Wayback Machine hat lediglich Fragmente gespeichert.
Das Parsen von VHDL-Projekten und die Darstellung mittels Graphviz war ebenfalls Thema einer Bachelorarbeit von Angelika Kratochwil. Eine Kontaktaufnahme war in beiden Fällen bisher ergebnislos.

Es ist schwierig, große Netzlisten so zu visualisieren, dass ein Überblick möglich ist. Ein Lösung ist die Aufteilung der Netzliste in Module. Bei der Verwendung von Degate sind dabei zwei Anwendungsfälle zu unterscheiden. Zum einen soll es möglich sein, ein Modul beispielsweise zu Dokumentationszwecken darzustellen. I. d. R. wird einem Modul eine handhabbare Menge von platzierten Standardzellen zugeordnet, so dass eine sinnvolle grafische Darstellung möglich ist. Eine rein statische Darstellung erfüllt diesen Zweck. Auf der anderen Seite wäre eine Visualisierung wünschenswert, die bei der Zuordnung von Standardzellen zu einem Modul hilft. Diese interaktive Zuordnung ist wahrscheinlich nur mittels einer eigenen Software möglich. Die Idee besteht dann darin, dass ausgehend von einer ausgewählten Standardzelleninstanz die Netzliste sukzessiv expandiert wird, bis geeignete Modulabgrenzungen erkennbar werden.

9.8 Ideen für die weiterführende Forschung

In der (semi-)automatischen Analyse von Netzlisten steckt ein großes Potential. Das Reverse-Engineering ließe sich teilweise beschleunigen, wenn Module automatisch erkannt würden. Eines dieser Modul könnte z. B. ein Teil eines Verschlüsselungsverfahrens darstellen, das automatisch in einer Netzliste gefunden werden soll. Beispielsweise ist das ein linear rückgekoppeltes Schieberegister, dessen Struktur nicht vollständig bekannt ist. Dieses Suchproblem ist allgemein nicht lösbar. Jedoch sind Verfahren denkbar, die wenigstens potentielle Module erkennen und mittels GUI den Benutzer zur Entscheidung auffordern. Bisher ist keine der folgenden Ideen in Degate implementiert.

Isomorphie von Subgraphen

Bei der Entwicklung von Schaltkreisen werden Module eingesetzt, denn mit Modulen lässt sich eine einzelne Beschreibung für mehrere Modulinstanzen nutzen. Als Schaltung realisierte Module sind bei der Betrachtung als Graph isomorph[9]. Wenn beim Reverse-Engineering eine Modulinstanz bzw. ein erkennbarer Teil davon in einem neuen Modul zusammengefasst wird, dann ließen sich mittels Subgraph-Isomorphie weitere Instanzen finden. Dieses Verfahren wird bereits im kommerziellen Reverse-Engineering-Prozess eingesetzt (vgl. Abschnitt 1.4).

[9] Infolge anschließender Optimierungsschritte kann die Isomorphie möglicherweise aufgehoben werden.

Heuristische Strukturerkennung, Ähnlichkeit von Graphen

Mittels Isomorphie können nur Subgraphen gefunden werden, die als Referenz vorhanden sind. Unbekannte Schaltkreise lassen sich damit nicht finden. Da es verschiedene Möglichkeiten gibt, wie ein Modul als Schaltung realisiert wird, hilft es nicht, eine Menge von Referenzschaltungen in einer Bibliothek zu hinterlegen und diese Referenzschaltungen mittels Subgraph-Isomorphie in der Netzliste zu suchen.

Möglicherweise wäre es zweckmäßig, Schaltungsstrukturen allgemein zu beschreiben und diese Beschreibungen für eine heuristische Suche zu verwenden oder Untergraphen zu finden, die einem Referenzgraphen ähnlich sind.

Kapitel 10

Verteilte Netzlistenrekonstruktion

Eine automatisierte Bildauswertung, mit der sich Netzlisten bei akzeptabler Fehlerrate fast vollständig rekonstruieren lassen, wäre wünschenswert. Leider ist die automatische Netzlistenrekonstruktion immer mit Fehlern verbunden. Im Einzelfall kann die Fehlerrate inakzeptabel hoch sein, so dass nur die manuelle Rekonstruktion von Netzlisten als einzig gangbare Option bleibt. Die manuelle Bearbeitung ist jedoch vergleichsweise zeitintensiv. Andererseits liefert sie jedoch hervorragende Ergebnisse bei einer äußerst geringen Fehlerwahrscheinlichkeit. Die bisherigen Erfahrungen mit Degate und diversen Reverse-Engineering-Projekten in der Vergangenheit belegen dies. Gleichfalls ist diese Beobachtung von dem Team hinter visual6502.org in deren FAQ bestätigt. Eine Möglichkeit, wie die Qualität manueller Netzlistenrekonstruktion bei kürzeren Rekonstruktionszeiten beibehalten werden kann, liegt in der Parallelisierung der Bildauswertung. Dazu benötigt man lediglich mehrere Freiwillige.

In den letzten Jahren ist die Zusammenarbeit vieler Freiwilliger an einem Projekt, dessen Umfang meistens von einer einzelner Person nicht beherrschbar ist, unter dem Namen Crowd-Sourcing bekannt geworden. Diese Art der Ressourcennutzung war bzw. ist bei vielen Projekten erfolgreich, u. a. beim Projekt OpenStreetMap[1], das als Ziel die Schaffung freien Kartenmaterials anstrebt und bei dem zu diesem Zweck auch Satellitenbilder nachgezeichnet werden.

Diese Form der Parallelisierung wäre beim Reverse-Engineering von Chips ebenfalls sinnvoll einsetzbar. Seit einiger Zeit ist eine Komponente für die verteilte Netzlistenrekonstruktion in Degate implementiert. Es gibt jedoch noch keine Erfahrungen

[1] `http://www.openstreetmap.org`, besucht am 18. April 2011

damit, da diese Komponente bisher nicht praktisch eingesetzt wurde. Die Funktionsweise und die Probleme sollen in diesem Kapitel dargestellt werden. Aktuelle Entwicklungen im Projekt visual6502.org sind ebenfalls Gegenstand der Betrachtung.

10.1 Verteilungsaspekte in Degate

Vor der Entwicklung der Verteilungskomponente in Degate waren einige grundsätzliche Fragen zu klären. Wünschenswert wäre es, wenn mit Degate sämtliche Aspekte des Reverse-Engineerings von Chips verteilbar wären. Dies könnte jedoch hohe Konsistenzanforderungen an die Datenmodelle in der Software stellen, bis hin zu der Forderung, dass alle Clients jederzeit die gleiche Sicht auf die Daten haben. Degate soll jedoch grundsätzlich offline nutzbar sein. Zur Wahrung der Datenkonsistenz wäre es dadurch ggf. notwendig, in Konflikt stehende simultane Änderungen an einem Objekt auflösen zu können, was einer Merge-Operation in Versionskontrollsystemen entspricht. Die Hauptschwierigkeit beim Auflösen von Konflikten besteht darin, dass verschiedene Konfliktarten jeweils eigene Anforderungen an die GUI-Komponenten stellen, mit denen Konflikte aufgelöst werden sollen. Während etwa zwei widersprüchliche Objektbenennungen mit einem einfachen textbasierten Dialog gelöst werden können, erfordert die Auflösung von Netzkonflikten eine eigene Form der Visualisierung. Dies wäre eine andere Form der Darstellung als die Auflösung von widersprüchlich platzierten Standardzellen. Um derartige Probleme zu vermeiden, beschränkt sich die Verteilungskomponente in Degate darauf, den langwierigsten Teil im Reverse-Engineering-Prozess zu parallelisieren. Dies ist der Arbeitsschritt, in dem Leiterbahnen und Durchkontaktierungen nachvollzogen werden und auf den bis zu zwei Drittel der gesamten Arbeitszeit entfallen.

Durch diese Beschränkung vereinfacht sich das Problem. Es müssen nunmehr nur Daten von Leiterbahnen und Durchkontaktierungen ausgetauscht werden. In Abschnitt 8.1 wurde beschrieben, dass das manuelle Einzeichnen von Leiterbahnen und Durchkontaktierungen unabhängig von der automatischen Zusammenführung sich berührender elektrisch verbindbarer Objekte ist. D. h. zunächst werden Leiterbahnen und Durchkonktatierungen eingezeichnet und in einem separaten Schritt werden sie verbunden, falls sie sich berühren. Daher müssen Informationen über Netze nicht ausgetauscht werden, denn Objekte werden erst anschließend an zentraler Stelle zu Netzen zusammengeführt.

10.2 Server zum Datenabgleich

Für das kollaborative Arbeiten an einem Projekt gibt es eine zentrale Instanz für den Datenabgleich. Diese Zentrale ist ein Server, der via XML-basierter Prozeduraufrufe (XML-RPC) Kommandos entgegennimmt und Daten zurückliefert. Die Wahl fiel auf XML-RPC, da es für alle gängigen Programmiersprachen Bibliotheken gibt, XML-RPC vergleichsweise einfach ist und dabei hohe Flexibilität bietet.[2]

Der Server bietet zwei aufrufbare Prozeduren: `push` und `pull`. Mittels `push` sendet ein Client ein Datum an den Server. Dieses Datum ist technisch eine Menge von Parameter der Prozedur `push`. Der Server sendet eine Transaktions-ID zurück und speichert das serialisierte Datum in ein Transaktions-Log. Die Transaktions-ID wird im Server mit jedem Aufruf von `push` um eins erhöht.

Ein Degate-Client kann mittels `pull` Transaktionsdaten empfangen. Dazu sendet der Client beim Aufruf der Server-Prozedur einen Parameter mit, ab welcher Transaktions-ID der Client die Änderungen empfangen möchte. Der Server sendet eine Liste von Transaktionsdaten an den Client zurück. Die in den Transaktionen gekapselten Daten sind Kommandos, die Clients sich gegenseitig über den Server senden. Der Server wertet die über ihn verteilten Daten selbst nicht aus, weshalb diese Menge von Parametern serverseitig als einzelnes Datum betrachtbar ist.

Auf der Ebene der Client-zu-Client-Kommunikation werden Befehle ausgetauscht, für die im Folgenden eine (nicht ganz vollständige) Grammatik wiedergegeben ist.

⟨command⟩ → ⟨add-command⟩ | ⟨remove-command⟩
⟨add-command⟩ → ⟨add-wire⟩ | ⟨add-via⟩
⟨remove-command⟩ → **remove** ⟨object-id⟩
⟨add-wire⟩ → **add wire** ⟨layer-id⟩ ⟨from-x⟩ ⟨from-y⟩ ⟨to-x⟩ ⟨to-y⟩ ⟨diameter⟩
⟨add-via⟩ → **add via** ⟨layer-id⟩ ⟨x⟩ ⟨y⟩ ⟨diameter⟩ ⟨direction⟩
⟨direction⟩ → **up | down**

In der libdegate gibt es eine Klasse namens RemoteObject. Alle Objekte von Klassen, die RemoteObject erben, erwerben die Fähigkeit, zu einem Server gesendet zu werden. Aus der Klasse RemoteObject muss dazu lediglich die abstrakte Methode

[2] Auf den Einsatz von SOAP wurde bewusst verzichtet, da es zwischen einzelnen Frameworks Interoperabilitätsprobleme gibt bzw. gab. Eine Architekture wie CORBA wären viel zu schwerfällig für den Zweck gewesen. Als Alternative zu XML-RPC wäre JSON-RPC geeignet.

`push_object_to_server()` implementiert werden. Über die Klasse RemoteObject wird eine weitere Eigenschaft vererbt – die Eigenschaft, eine global eindeutige ID (*remote object ID*, ROID) zu besitzen: Jedes Objekt des Logikmodells in Degate hat eine eigene Objekt-ID. Diese ID wird lokal verwaltet und ist nur in einem Client eindeutig. Die ROID identifiziert ein Logikmodellobjekt eindeutig über mehrere Client-Instanzen. Darüber hinaus dient diese ROID als Kennzeichnung, ob ein Objekt bereits zum Server übertragen wurde. Die vom Server zurückgelieferte Transaktions-ID wird als ROID benutzt. Hat das Objekt keine ROID, wurde es noch nicht übertragen.

10.3 Konfliktbehandlung

Werden Informationen über Leiterbahnen und Durchkontaktierungen zwischen den Clients ausgetauscht, können Konflikte auftreten. Diese Konflikte entstehen z. B., wenn zwei Degate-Benutzer die Vektordaten desselben Leiterbahnsegments zum Server senden und die Vektordaten sich unterscheiden. Beim Abgleich der Daten importieren dann beide Clients das Leiterbahnsegment des jeweils anderen Benutzers. Ein einfacher Test, ob ein Leiterbahnsegment bereits im Logikmodell vorhanden ist, ist nicht möglich.

Überlappende Leiterbahnen oder doppelt platzierte Durchkontaktierungen sind für die Netzlistenrekonstruktion unproblematisch, solange keine Kurzschlüsse mit anderen Leitungen entstehen. In der aktuellen Version von Degate werden daher übereinander platzierte Liniensegmente bzw. Durchkontaktierungen nicht aufgelöst.

10.4 Visual6502.org

Das Projekt visual6502.org hat bisher keine positiven Erfahrungen mit der automatisierten Bildanalyse sammeln können und setzt daher auf manuell rekonstruierte Netzlisten. Um den Prozess zu beschleunigen, soll ebenfalls ein Crowd-Sourcing-Ansatz etabliert werden. In der Umsetzung sind Unterschiede zu Degates Ansatz feststellbar.

Visual6502 entwickelt für das kollaborative Reverse-Engineering eine Webapplikation in Form eines Spiels. Benutzern wird ein Bildausschnitt eines Chips angezeigt, in den er Polygone einzeichnen muss. Die Polygone werden gespeichert. Das Backend der Webapplikation bewertet die Aktivität der Spieler, wobei höchstbewertete Spieler in

einer Highscore-Liste genannt werden. Wie diese Bewertung erfolgen soll, ist derzeit (Februar 2011) noch offen. Eine Idee besteht darin, Bildkacheln mehrfach zeichnen zu lassen und anhand heuristischer Vergleiche zu manuell bewerteten Bildkacheln eine Bewertung abzuleiten. Es besteht zwar die Gefahr, dass die Qualität der Polygonnachzeichnung leidet, wenn mit weniger Aufwand ein höherer Punktestand erzielbar ist. Inwiefern diese Problematik zutrifft, ist dann aber vom Bewertungsschema abhängig, welches bislang nicht existiert.

Der Vorteil gegenüber dem Ansatz in Degate besteht darin, dass die Anwendung webbasiert ist. Dies ermöglicht mehreren Menschen, parallel an einem Reverse-Engineering zu arbeiten, ohne dass ein spezieller Client installiert werden muss. Da für das manuelle Nachzeichnen keine besonderen Techniken erforderlich sind, ist eine Umsetzung der Kernfunktionalität in Javascript einfach. Sollte die Crowd-Sourcing-Komponente in Degate in einem zukünftigen Reverse-Engineering-Vorhaben Verwendung finden, könnte man mit wenig Zeitaufwand einen webbasierten Ansatz nachprogrammieren und den XML-RPC-Server als Backend verwenden.

10.5 Ansatz zur Vergütung von Aufwänden

Degate soll auch in kommerziellen Reverse-Engineering-Projekten Verwendung finden. Für die Verfolgung von Leiterbahnen gibt es daher die Idee, Mitarbeitern Aufwände zu vergüten. Dies könnte z. B. so gestaltet sein, dass ein fixer Betrag entsprechend des individuellen Beitrags zur Leiterbahnverfolgung aufgeteilt und ausgeschüttet wird. Als Maß für den individuellen Beitrag könnte die Menge an bearbeiteten Kacheln oder der Gesamtlänge der Leiterbahnen gewertet werden. Der komplette Prozess von der Verteilung der Bilddaten bis zur Auszahlung von Entgelten ließe sich automatisieren, so dass lediglich eine manuelle Qualitätskontrolle der bearbeiteten Kacheln notwendig wäre.

Kapitel 11

Schlussbetrachtung

11.1 Auswirkungen

Degate ist die erste Software für das Reverse-Engineering von Chips, die vollständig quelloffen ist und nicht lediglich einen einzelnen Aspekt des Reverse-Engineering behandelt (beispielsweise Masken-ROM-Dekodierung). Vielmehr bildet Degate einen großen Teil des Arbeitsprozesses der Funktionsrekonstruktion ab. Zweifellos ist ein Chip-Reverse-Engineering auf der Ebene, wie es in Kapitel 3 beschrieben ist, auch ohne eine spezielle Software möglich, wenngleich der Aufwand um Größenordnungen höher ist. Daher ist das Besondere an Degate, dass die Komplexität eines Chip-Reverse-Engineering unter Einsatz dieser Software beherrschbar wird. Ferner ergibt sich durch Abbildung der Arbeitsschritte in einer Software, dass das Reverse-Engineering ein planbarer gradliniger Prozess wird. Viele Elemente der Software wären von Menschen benutzbar, die nicht im Detail mit dem Reverse-Engineering von Chips vertraut sind. Es genügt, wesentliche Punkte des Arbeitsprozesses nachvollziehen und die Software benutzen zu können. Letzendlich ist es auch ein Ziel dieser Arbeit zu vermitteln, dass für die Schaltkreisrekonstruktion kein elektrotechnisches Spezialwissen notwendig ist.

Da Degate bereits für verschiedene Reverse-Engineering-Projekte eingesetzt und parallel dazu weiterentwickelt wurde, ist die Software über das Stadium einer reinen Machbarkeitsstudie hinaus. Die Ergebnisse diverser Rekonstruktionsprojekte wurden publiziert (z. B. in [Noh+08]), so dass Erkenntnisse über einige proprietäre Verschlüsselungsverfahren und deren Unzulänglichkeiten öffentlich wurden und dies ein

höheres Problembewußtsein bei Nutzern von Sicherheitstechnik bewirkt hat. Die Annahme, dass Chip-Reverse-Engineering ausschließlich spezialisierten Institutionen vorbehalten sei, ist nunmehr nicht mehr statthaft.

Mit der Veröffentlichung der Quelltexte von Degate und der Dokumentation des Arbeitsprozesses ist auch anderen Forschern möglich, einen leichteren Einstieg in das Thema VLSI-Reverse-Engineering zu finden und selbst Sicherheitsverfahren in Chips zu evaluieren. Mittel- und langfristig wird dies zu weiteren Publikationen über sicherheitsrelevante Verfahren führen.

11.2 Offene Forschungsfragen

Neben diesen Auswirkungen gibt es weiterhin Bedarf an einer verbesserten automatisierten Erkennung. Während Instanzen von Standardzellen und (größere) Durchkontaktierungen weitgehend problemlos erkannt werden, ist die automatisierte Erkennung von Leiterbahnen problematisch. Hauptursächlich dafür ist, dass sich im Bildmaterial Leiterbahnen benachbarter Chipebenen kreuzen und häufig die Auflösung zu gering ist. Diese Funktion lässt sich daher noch nicht produktiv einsetzen. Die Möglichkeiten, mittels eigener Skripte Bilddaten auszuwerten und der Crowd-Sourcing-Ansatz für das verteilte manuelle Nachvollziehen, sind Lösungswege, um mit diesem Problem der geringen Erkennungsraten von Leiterbahnen umzugehen. An dieser Stelle ist weitere Arbeit notwendig, die jedoch über das hinausgeht, was im zeitlichen Rahmen einer Diplomarbeit realistisch wäre. Letztendlich kann ein Algorithmus auch nur bedingt „reparieren“, was bei der Wahl des Bildaufnahmeverfahrens berücksichtigt werden müsste.

Degate ist aus einem Freizeitprojekt heraus entstanden und ist bis dato ein Ein-Personen-Projekt. Wenngleich immer wieder Feedback eintrifft, gelegentlich den Autor ein Patch erreicht, um Degate auf nicht direkt unterstützen Plattformen übersetzen zu können, gibt es weltweit nur wenige Nutzer und noch weniger Intensivnutzer. Zugegeben, die Nutzer von Degate sind letztendlich die Tester. Degate ist mittlerweile mit knapp 50.000 Zeilen Code eine komplexe GUI-Anwendung und bietet zahlreiche Wege, wie Funktionen nutzbar sind. Dies führte regelmäßig dazu, dass Nutzer Degate in einer Weise nutzen, die so nicht gedacht war und die jenseits der ursprünglich getesteten Interaktionsform stattfindet. Die Integration von neuen Funktionen in Degate und die Benutzung neuer Funktionen in Reverse-Engineering-Projekten sind

zeitlich nicht direkt aufeinanderfolgend. Dies führte in der Vergangenheit dazu, dass Degate phasenweise eine erhöhte Fehlerdichte hatte, wobei sich Fehler erst bei der intensiven Benutzung bemerkbar machten. I. d. R. ist daher die Stabilität von Degate zum Ende eines Reverse-Engineering-Projektes am höchsten.

In einigen Fällen ist es schwierig, Funktionen so in die GUI zu integrieren, dass die Benutzung intuitiv verständlich ist und sich der Aufwand für die Integration in Grenzen hält. Dafür gibt es zwei Gründe. Zum einen ist gtkmm als Basis für die grafische Benutzerschnittstelle partiell ein Hindernis, weil sich einige Ideen gar nicht oder nur schwer umsetzen lassen. Der zweite Grund liegt in der konzeptuellen Aufbereitung von Funktionen. Im Gegensatz etwa zu Textverarbeitungsprogrammen, die eine umfangreiche GUI-Evolution erfahren haben, sind Programme zum VLSI-Reverse-Engineering eine weitgehend moderne Erscheinung. Da keines dieser Programme (vgl. Abschnitt 1.4) ausprobierbar war, basieren die Benutzungsideen letztendlich auf eigenen Überlegungen. Ob die GUI-Konzepte sinnvoll sind, zeigt sich daher erst bei der Benutzung.

Weiterführende Forschungsfragen ergeben sich hinsichtlich der Härtung von Chips. Diese Arbeit zeigt Methoden für die Schaltkreisrekonstruktion. Offen bleibt die Frage, welche Gegenmaßnahmen die Wirksamkeit der hier beschriebenen Methoden abschwächen oder sogar verhindern bzw. mit welchen Maßnahmen die Kosten für ein Reverse-Engineering erhöht werden. Insbesondere wäre interessant, ob sich die Wirksamkeit von Gegenmaßnahmen in einer Metrik ausdrücken läßt.

11.3 Lizenz und Kommerzialisierung

Häufig gab es die Frage, ob die Wahl des Lizenzmodells GNU Public Licence (GPL) sowohl für die Bibliothek libdegate als auch die GUI geeignet ist und die Überlegung, die Lizenz zu ändern oder auf ein duales Lizenzmodell umzustellen. Die GPL schützt überwiegend das Werk und weniger den Autor. Die GPL verhindert theoretisch, dass in Umlauf gebrachte Änderungen der Software geheim bleiben. Dies ist ein gewünschter Effekt gerade für die Software Degate. Seit Beginn des Projektes war klar, dass es eine derartige Software noch nicht im öffentlichen Raum gibt. Es bestand bzw. besteht die Gefahr, dass bei Wahl einer liberaleren Lizenz Änderungen an der Software vorgenommen werden, die selbst nicht den Weg zurück in die Öffentlichkeit finden. Dies wäre beispielsweise bei der BSD-Lizenz möglich. Das Alleinstellungsmerkmal,

Open-Source zu sein, wäre damit für kommerzielle Zwecke ausnutzbar. Für den Autor ist es entscheidend, dass jede Form von Arbeitszeit, die Dritte in Degate investieren, allen Nutzern gleichermaßen zugute kommt.

Die Wahl des GPL-Lizenzmodells hat sich im Nachhinein als sinnvoll erwiesen. Dem Autor ist eine Stellenanzeige bekannt, in der Degate als Beispiel für eine Chip-Reverse-Engineering-Software genannt wurde, die im Rahmen der Tätigkeit zu programmieren wäre. Darüber hinaus wurde dem Autor angeboten, Degate in einer kommerziellen Version nachzuimplementieren, da Degate in seiner aktuellen Lizenz nicht direkt für die kommerzielle Nutzung geeignet ist. Beide Beispiele zeigen, dass grundsätzlicher Bedarf an Software zum VLSI-Reverse-Engineering von Chips besteht.

Der akkumulierte Arbeitsaufwand für die Entwicklung von Degate beträgt derzeit etwa 12 Monate. Darin ist der Zeitaufwand für die Entwicklung des Prototypen, Recherche, Dokumentation, Testen und Überlegungen zu Verbesserungsmöglichkeiten enthalten. Grundsätzlich hat der Autor nichts gegen eine Nachimplementation einzuwenden. Ohnehin wäre dies legitim und lizenzkompatibel, solange keine Programmteile dafür verwendet werden. Für die Neuentwicklung hin zu einem verkaufbaren Produkt wären vermutlich 18 bis 24 Montate nötig.

Degate bietet in seiner aktuellen Form durchaus Potential für kommerzielle Erweiterungen, die nicht gegen die GPL verstoßen. Zu diesen Erweiterungen zählt die Anbindung von Werkzeugen, die für die Aufnahme von Bilddaten aus Mikroskopen geeignet sind, sowie die automatische Analyse von Bilddaten. Darüber hinaus sind Netzlistenanalysen denkbar, die als Erweiterungen an Degate anbindbar wären.

11.4 Ausblick

Innerhalb der letzten zehn Jahre sind die RAM- und Festplattenkapazitäten handelsüblicher Computer in einem Maße gewachsen, die es erlauben, umfangreiche Bilddaten zu speichern und zu verarbeiten. Finden sich in älteren Aufsätzen über VLSI-CAD-Systeme Passagen über Speicherplatzoptimierungen, tritt dieses Problem im Normalfall zunehmend in den Hintergrund. Die Speicherung eines unkomprimierten Bildes im RGBA-Format in der Größe von 100.000 x 100.000 Pixel benötigt etwa 40 GB Speicher. Die entsprechende Festplattenkapazität dafür kostet derzeit etwa zwei Euro. Zwar übersteigt diese Speichermenge noch die maximale RAM-Kapazität üblicher Computer, dennoch können derartig große Bilder mit geeigneter Software

bearbeitet werden. Insbesondere vor dem Hintergrund, dass die Anschaffungskosten für Bildaufnahmegeräte sinken, verwundert es nicht, dass zunehmend Menschen sich mit dem Thema Reverse-Engineering von Chips auseinandersetzen. Bildaufnahmegeräte nach dem aktuellen Stand der Technik sind zwar immer noch astronomisch teuer, dennoch sind „ältere" Geräte für Analysezwecke ebenfalls geeignet und dabei signifikant günstiger. Dem allgemeinen Trend, dass selbst spezielle High-Tech-Produkte, irgendwann für eine breite Nutzerschaft zur Verfügung stehen, folgen auch Geräte für die VLSI-Inspektion.

Im Laufe der Entwicklung von Degate hat der Autor mehrere Menschen kennengelernt, die sich ebenfalls mit dem Thema Reverse-Engineering von Chips auseinandersetzen. Selbstverständlich wurden Chips schon immer von Konkurrenzunternehmen untersucht und die Pioniertage des VLSI-Reverse-Engineerings lagen zumindest auf der „Hacker-Ebene" in der zweiten Hälfte der 1990er Jahre (vgl. [KK99]). Dennoch sind Rekonstruktion von und Angriffe auf Chips weiterhin ein offenes Forschungsfeld. Trotz der Entwicklungen in den letzten 15 Jahren ist „Hardware-Hacking" auf Chipebene vermutlich erst der Anfang einer langfristigen Entwicklung. Dies wird umso deutlicher, wenn man dazu im gleichen Zeitraum die Fortschritte im Bereich der Software- und Netzwerksicherheit vergleicht. Folglich ist für den Bereich des Chip-Reverse-Engineerings zu erwarten, dass weitere, insbesondere freie Werkzeuge entstehen.

Anhang A

Interaktive Funktionen der Software

Die Interaktivität der Software ist ein wesentliches Merkmal von Degate. Dieses Kapitel soll die GUI-Funktionen der Software Degate beschreiben und illustrieren.

A.1 Zusammenstellen von Ebenen

Das Reverse-Engineering von Chips mittels Degate beginnt mit dem Erstellen einer Projektmappe. Ein Projekt ist ein Container für Bild- und Metadaten, die sich auf einen Chip bzw. einen Teil eines Chips beziehen. Die Mehrschichtigkeit von integrierten Schaltungen wird dadurch abgebildet, dass Projekte aus einem Stapel von Ebenen bestehen. Jede dieser Ebenen kann eigenes Bildmaterial repräsentieren und Elemente des Logik-Modells speichern. Als Logik-Modell wird hier hauptsächlich die Gesamtheit aller Schaltelemente verstanden. Im weiteren Sinne gehören nichtfunktionale Elemente dazu, die lediglich geometrisch verankerte Informationen abbilden (z. B. Annotationen und Unterprojekte) oder die einen gruppierenden Effekt haben (Module).

Der Dialog „Layer Configuration“ dient der Zusammenstellung der Ebenen. Er wird beim Erstellen neuer Projekte angezeigt und kann alternativ über das Menü ► *Layer / Layer configuration* geöffnet werden. Über diesen Dialog kann man das Bildmaterial für einzelne Ebenen auswählen. Ferner wird hier festgelegt, welchen Typs eine Ebene ist. Ebenen können mit einem beschreibenden Kommentar versehen werden. Zudem ist es möglich, Ebenen zu deaktivieren. Dies bedeutet, dass diese Ebenen zwar im Logik-Modell weiterhin existieren, bei der Benutzung der Software aber weitgehend

versteckt sind. Diese Funktion ist nützlich, um alternatives Bildmaterial für bereits importierte Ebenen in der Software verwalten zu können. Mittels Drag-and-Drop lässt sich die Reihenfolge von Ebenen ändern.

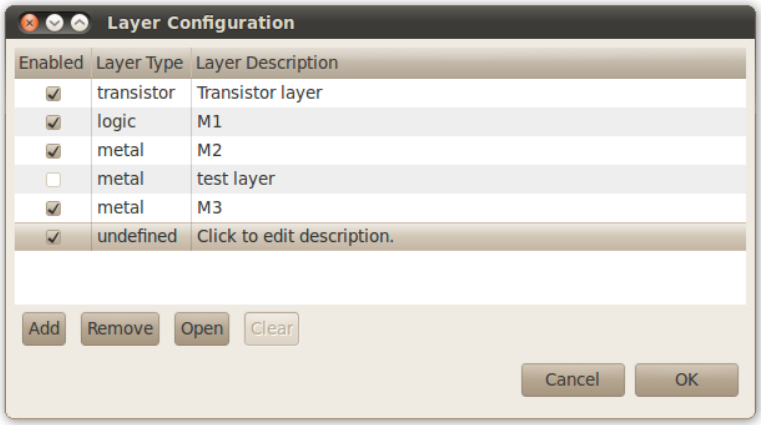

Abbildung A.1: Konfiguration der Ebenen.

Wenn mehrschichtiges Bildmaterial importiert wird, müssen die Ebenen „deckungsgleich“ sein.[1]

Unter dem Menüeintrag ▶ *Project / Project Settings* können allgemeine Informationen über ein Projekt hinterlegt werden (vgl. Abbildung A.2). Ein Projekt sollte einen Namen und eine Beschreibung haben. Im Dialogfenster kann man die Standarddurchmesser für Durchkontaktierungen und Leiterbahnen festlegen. Der ebenfalls in diesem Fenster konfigurierbare Parameter Lambda ist ein Toleranzwert. Er gibt an, wieviele Pixel zwei elektrisch leitende Objekte voneinander entfernt liegen dürfen, damit diese noch als zusammenhängend gelten. Die Server-URI gibt die Adresse eines XML-RPC-Servers an, der beim kollaborativen Arbeiten zum Synchronisieren von Daten verwendet wird. Diese Adresse wird beim Erstellen eines Projektes automatisch erzeugt. Die Angabe „Pixel per μm“ dient der Umrechnung von Längen.

[1] Im Prototyp von Degate bestand die Möglichkeit, jeweils vier Referenzpunkte pro Ebene festzulegen. Zwei der Referenzpunkte dienten dazu, die Übereinstimmung von Punkten mit der darunterliegenden Ebene zu markieren. Die anderen zwei Referenzpunkte markierten Übereinstimmungen mit der darüberliegenden Schicht. Wenn für alle Ebenen diese vier Referenzpunkte festgelegt wurden (für die oberste und unterste Schicht natürlich nur zwei), konnten Skalierungsfaktoren und Translationswerte für alle Ebenen automatisch bestimmt werden. Mittels weiterer Markierungen hätte man Rotationen korrigieren können. Die Funktion zum Ausrichten von Ebenen wurde jedoch nicht benutzt, so dass es bei der Neuentwicklung bisher keinen Bedarf an einer Portierung gab. Höchstwahrscheinlich würden einfache Offsets für eine relative Ausrichtung von Ebenen genügen.

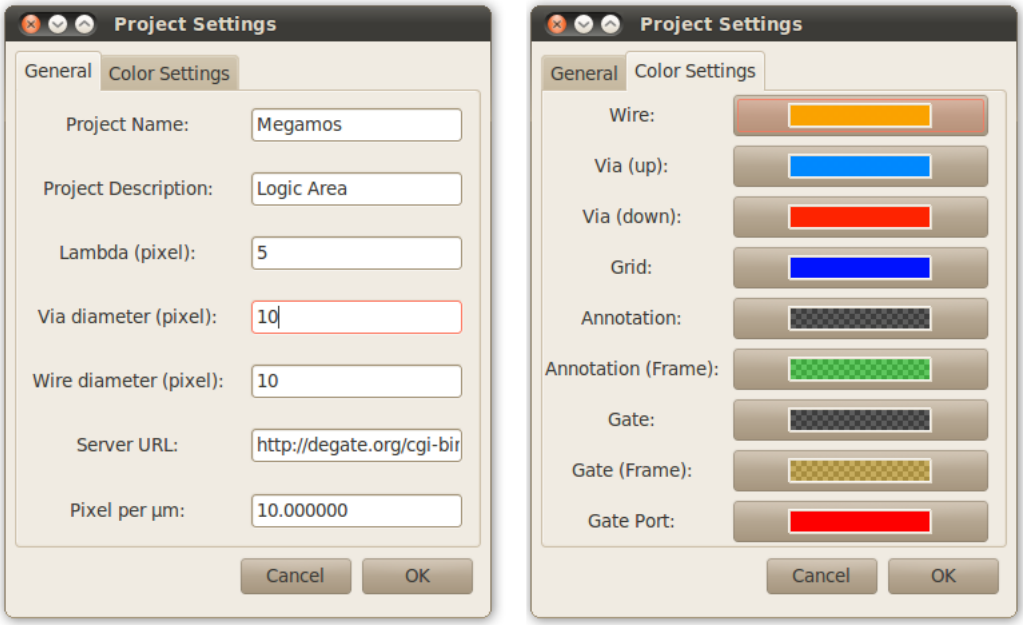

Abbildung A.2: Bearbeiten der Projekteinstellungen

Unter der Registerkarte „Color Settings“ können projektbezogene Standardfarben festgelegt werden, beispielsweise für die Schrift und für Darstellung von Leiterbahnen.

A.2 Hauptfenster, Menü und Navigation

Im Hauptfenster (Abbildung A.3) werden die Bilddaten eines Projektes dargestellt. Es wird stets eine Ebene angezeigt. Über Schaltknöpfe in der Werkzeugleiste und Tastenkürzel kann zur nächsten bzw. vorherigen Ebene gewechselt werden. Inaktive Ebenen werden dabei übersprungen. Ebenen können auch gezielt mittels Tastatur ausgewählt werden. Die Navigation erfolgt wahlweise über die Tastatur, mittels der Maus und mittels GUI-Elementen wie man dies aus Grafikbearbeitungsprogrammen gewohnt ist.

Über das Menü können Aktionen ausgeführt werden. Das Menü ist dabei in folgende Haupteinträge unterteilt:

- Unter dem Eintrag *Project* sind alle allgemeinen projektmappenspezifischen Funktionen zusammengefasst: Das Erstellen von Projekten und Projektarchiven, das Wechseln von und zu Unter-Projekten, Projekteinstellungen und das

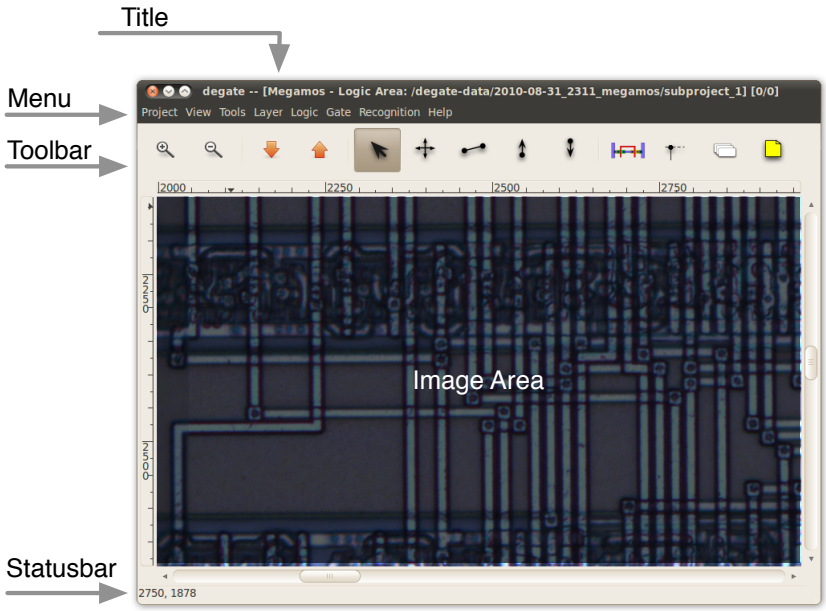

Abbildung A.3: Hauptfenster der Software Degate.

Senden und Empfangen von Änderungsdatensätzen von und zum Kollaborations-Server (vgl. Kapitel 10).

- Der Eintrag *View* fasst Navigationsaktionen zusammen. Darüber hinaus stehen Funktionen zum Einstellen von Hilfslinien und zum Ein- und Ausblenden von GUI-Elementen zur Verfügung.
- Unterhalb von *Tools* kann ein Werkzeug ausgewählt werden. Diese Werkzeuge stehen auch über die Toolbar zur Verfügung. Die Werkzeuge dienen zum Markieren von Bereichen und zur Selektion von Objekten, zum Verschieben des Bildausschnitts, zum Einzeichnen von Leiterbahnen und Durchkontaktierungen.
- Unter dem Menüpunkt *Layer* sind Funktionen zum Im- und Exportieren von Bilddaten, zum Löschen von Bilddaten und zur Konfiguration der Ebenen zusammengefasst.
- Für den Umgang mit dem Logik-Modell können Aktionen unterhalb vom Menüeintrag *Logic* ausgewählt werden, z. B. zum elektrischen Verbinden bzw. Isolieren von Objekten, zum Löschen von Objekten, für den Umgang mit Annotationen und Modulen, zur Introspektion von elektrisch verbundenen Objekten und zum Ausführen von Rule Checks.

- Spezielle Logik-Modell-Funktionen, die überwiegend auf Standardzellen bezogen sind, sind unter dem Menüpunkt *Gate* angeordnet. Dazu gehören u. a. Funktionen zum Erstellen von Standardzellen, zum Platzieren, zum Festlegen der Layout-Orientierung, zum Mischen von Layoutbilddaten, zum Benennen, Finden, Entfernen und Auflisten von Standardzellen sowie zur Festlegung der farblichen Darstellung von Ports.
- Über den Menüeintrag *Recognition* können Funktionen zum automatischen Erkennen von Standardzellenplatzierungen, Leiterbahnen und Durchkontaktierungen gestartet werden.

A.3 Hilfslinienkonfiguration

Das Grid ist ein projektglobales Hilfsraster, und dient hauptsächlich dem Ausrichten von platzierten Standardzellen. Es wird im Hauptfenster der Anwendung dargestellt. Das Raster kann entweder äquidistant oder unregulär sein und aus wahlweise horizontalen und/oder vertikalen Hilfslinien bestehen. Äquidistante Raster werden mittels eines Offset- und Distanzwertes konfiguriert. Dabei ist der Distanzwert eine Gleitkommazahl, um Subpixelgenauigkeit zu ermöglichen. Unregelmäßige Rasterlinien werden mittels der rechten Maustaste im Hauptfenster gesetzt. Aus einem Popup-Menü heraus wählt der Benutzer aus, ob an der angeklickten Stelle eine horizontale oder vertikale Hilfslinie erscheinen soll.

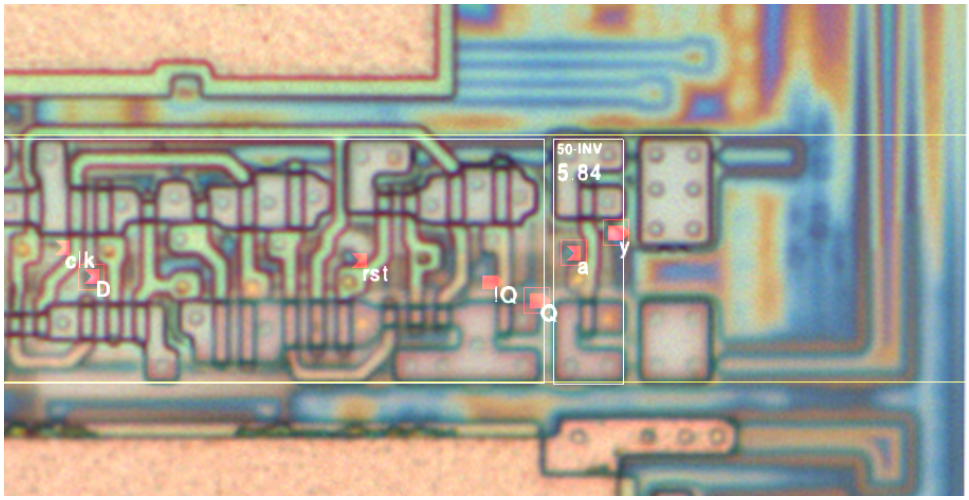

Abbildung A.4: Darstellung von Hilfslinien entlang der Standardzellen.

Das Hilfsraster wird über den Menüeintrag ▶ *View / Grid Configuration* bearbeitet. Dort können ebenfalls Offset-Angaben für unregelmäßige Raster als Zahlenwerte eingestellt werden. Dies ist jedoch weniger praktisch.

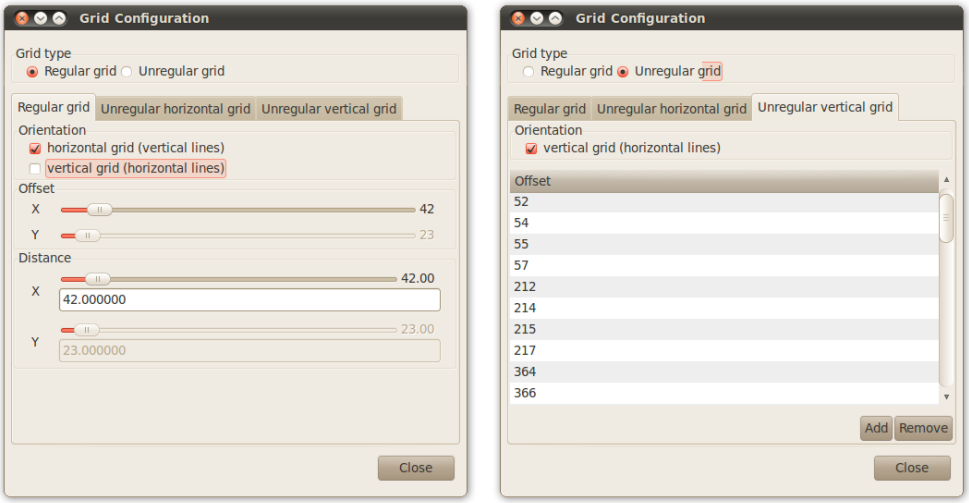

Abbildung A.5: Konfiguration der Hilfslinien.

Standardzellenschablonen sollten an den Hilfslinien ausgerichtet werden, denn dann genügt es, weitere Instanzen entlang der Hilfslinien zu suchen. Damit lässt sich der Suchraum erheblich einschränken. Diese Optimierung wird im Abschnitt 7 behandelt. Ferner kann man mittels des Rasters prüfen, ob das Bildmaterial Verdrehungen oder Wellen aufweisst. Eine Welligkeit kann infolge des Stitchens auftreten, also bei jenem Prozess, bei dem mikroskopierte Bilder zu einem großen Bild zusammengefügt werden. Die Suche nach Standardzelleninstanzen ist bis zu einem gewissen Grad unempfindlich gegenüber Wellen.

A.4 Festlegen und Beschreiben von Standardzellen

In Abschnitt 3.4 wurde beschrieben, wie man Standardzellenlayouts identifizieren kann. Wenn man eine Standardzelle erkannt hat, kann deren Bereich im Hauptfenster mit der Maus markiert werden. Über den Menüeintrag ▶ *Gate / Create Gate by selection* legt man fest, dass der ausgewählte Bereich einer Standardzelle entspricht. Daraufhin öffnet sich ein Dialogfenster. In Abbildung A.6 ist dies beispielhaft für eine Standardzelle vom Typ 3-NAND zu sehen.

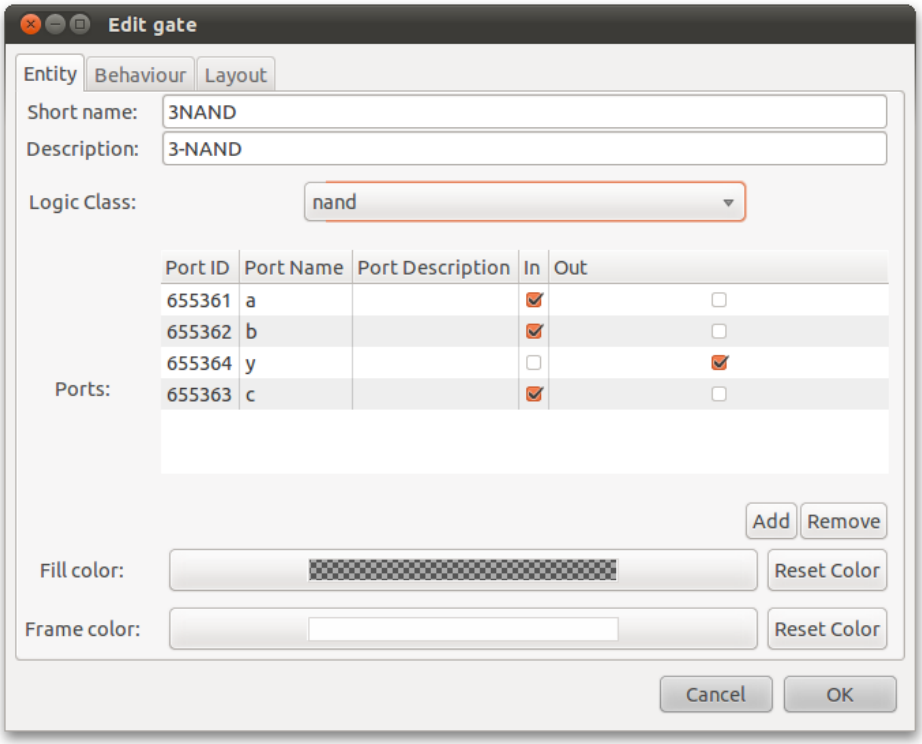

Abbildung A.6: Bearbeiten der Schnittstellen einer Standardzelle.

Der erste Eintrag im Dialogfenster spezifiziert einen Kurznamen. Dieser Name wird im Hauptfenster innerhalb der Umrahmung von Standardzellen dargestellt. Kurznamen sollten nicht mehr als fünf Zeichen umfassen. Zwar wird der Text in der dargestellten Standardzelle skaliert, so dass der Text nicht über deren Umrahmung hinausragt. Dies bedeutet jedoch auch, dass der Text bei schmalen Zellen zum Lesen zu klein werden kann. Für eine detaillierte Erklärung kann das Beschreibungsfeld genutzt werden.

Mit dem Auswahlfeld „Logic Class“ kann der Standardzellentyp spezifiziert werden. Als mögliche Belegungen stehen verschiedene Logikfunktionen zur Verfügung, z. B. (Tristate-)Inverter, (N)AND, (N)OR, X(N)OR. Darüber können Buffer, verschiedene Latch- und Flipflop-Formen, Halb- und Volladdierer, (De-)Multiplexer, Isolationsgatter und mehrstufige kombinatorische Typen (zB AOI) festgelegt werden. Diese Typinformation dient beim Generieren von Codegerüsten dazu, eine geeignete Templatekategorie auszuwählen.

Jede Standardzelle hat ein oder mehrere Ausgänge[2]. Diese Schnittstellen können unter Ports festgelegt werden. Mehrwertige Logiken nach IEEE 1164, einschließlich des Tristate-Zustands, können in Degate abgebildet werden. Dies erfolgt jedoch nicht in der Entitäten-Beschreibung, sondern ist Teil der Verhaltensbeschreibung.

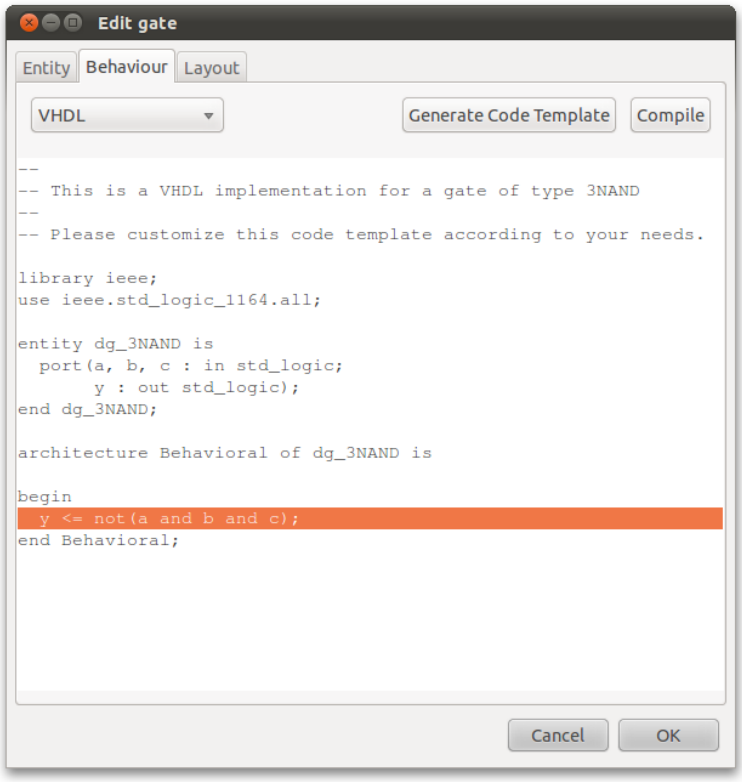

Abbildung A.7: Bearbeiten der Verhaltensbeschreibung einer Standardzelle.

Allen in Degate platzierbaren Objekten, u. a. Standardzellen, können Farbdefinitionen zugewiesen werden. Die Hintergrund- und Rahmenfarbe für alle Standardzelleninstanzen des gleichen Typs, kann man im Dialogfenster auswählen. Dadurch ist es möglich, den Typ anhand von Farbwahl kenntlich zu machen, z. B. die in kryptografischen Algorithmen regelmäßig anzutreffenden Flipflops und XOR-Gatter.

Alle Angaben in diesem Dialogfenster sind zunächst optional. Ebenfalls besteht keine Notwendigkeit, dass Namen eindeutig sind. Es ist dennoch sinnvoll, einen möglichst eindeutigen Kurznamen zu spezifizieren.

[2]Isolationsgatter werden in Degate als Sonderfall einer Standardzelle behandelt. Auf Register-Transferebene, die für die Rekonstruktion von Schaltfunktionen relevant ist, sind die Isolationsgatter bedeutungslos. Daher können sie hier im Wesentlichen ignoriert werden.

Wenn die Funktion der Standardzelle ermittelt ist (vgl. Abschnitt 3.6) kann das Ergebnis unter der Registerkarte „Behaviour“ dokumentiert werden. Dort besteht die Möglichkeit, Verhalten als Freitext oder in Form von Wahrheitswerttabellen zu beschreiben. Darüber hinaus kann das Verhalten mittels Hardwarebeschreibungssprachen hinterlegt werden. Zusätzlich können Testfälle, sogenannte Testbenches, für die Verhaltensbeschreibungen gespeichert werden.

Wenn das Feld „Logic Class“ belegt ist, kann mittels Knopfdruck ein Code-Gerüst für die Verhaltensbeschreibung und die jeweilige Testbench erzeugt werden. Unterstützt werden Verilog und VHDL. Der Code-Generator benutzt die Schnittstellen, die unter der Registerkarte „Entity“ im gleichen Dialog angegeben sind. Spezielle Ports, etwa Reset- und Clock-Ports werden anhand üblicher Namen (z. B. „rst“) erkannt. Mehr dazu in Kapitel 9.

Die Position von Ports innerhalb von Standardzellen wird im Hauptfenster der Anwendung festgelegt. Dazu klickt man mit der rechten Maustaste an die gewünschte Position innerhalb einer Standardzelleninstanz. Man wählt dann den Eintrag „Set port on gate“ in dem sich öffnenden Popup-Menü aus. Daraufhin öffnet sich ein weiteres Dialogfenster. Dieses verzeichnet alle Ports der Standardzelle, in deren Platzierungsbereich geklickt wurde. In dem Dialogfenster wählt man einen Port aus. Die Festlegung gilt dann für alle Instanzen des Standardzellentyps. Definition und Platzierung von Standardzellenports sind separate Schritte. Ports können jederzeit umplatziert werden.

Wenn für einen Bildbereich festgelegt wurde, dass dieser eine Standardzelle repräsentiert, wird nach erfolgreichem Ablauf des Dialogs zur Beschreibung der Entität der gewählte Bildbereich als grafische Repräsentation des Standardzellenlayouts verwendet. Die grafische Repräsentation umfasst dabei mehrere Ebenen des Chips: Die (erste) Transistorebene sowie die Ebenen M1 und M2. Des Weiteren wird in dem markierten Bereich eine erste Standardzelle des Typs platziert. Die grafische Layoutrepräsentation wird in der Standardzellendefinition gespeichert und ist unabhängig davon, ob Standardzellen in einem Projekt platziert sind.

Die Liste der in einem Projekt hinterlegten Standardzellen kann man sich in der „Gate Library“ ansehen (vgl. Abbildung A.8).

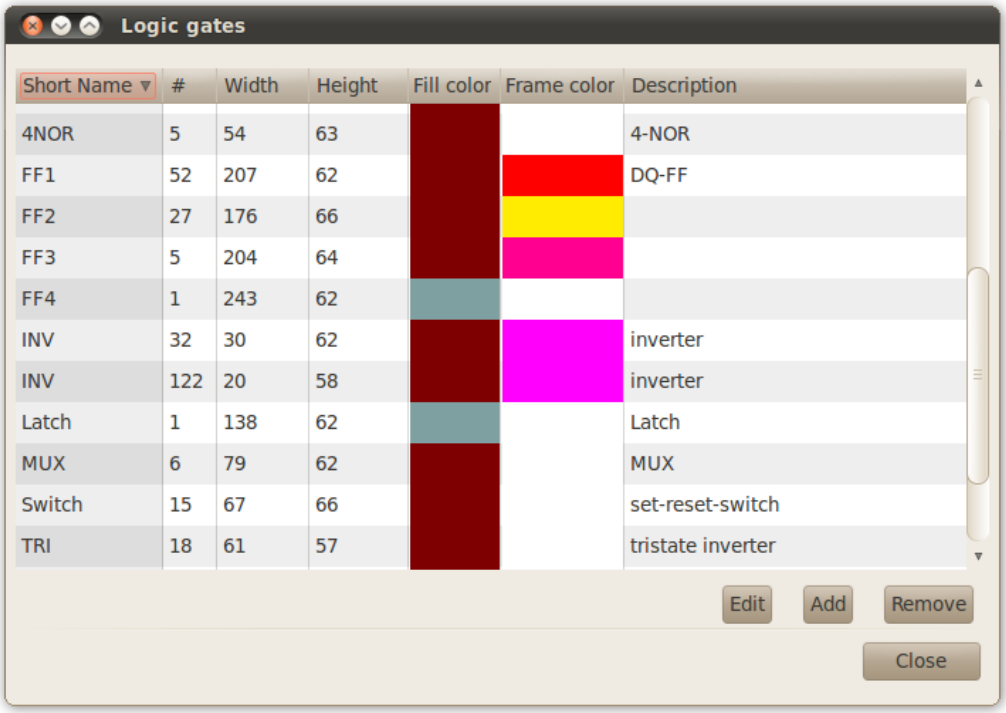

Abbildung A.8: Gatter-Bibliothek: Alle Standardzellenfestlegungen werden in der Gatter-Bibliothek übersichtsartig zusammengefasst.

A.5 Finden von Standardzelleninstanzen

Wenn Standardzellentypen in der Gatterbibliothek mit jeweiligem Bildmaterial als Schablone definiert sind, können Instanzen der Standardzellen gesucht werden. In der grafischen Benutzeroberfläche markiert man mit der Maus einen Ausschnitt, in dem die Suche erfolgen soll. Wird keine Suchbereichseinschränkung festgelegt, erstreckt sich die Suche auf die gesamte Fläche.

Standardzelleninstanzen können auf verschiedenen Ebenen des Chips gesucht werden, wobei die Ebene M1 für die Suche am besten geeignet ist. Die Suche erfolgt dabei stets auf der Schicht, die im Hauptfenster der Anwendung dargestellt ist. Zu einer Chip-Ebene liegt in Degate die Typinformation vor, d. h. ob eine Ebene Transistoren oder Leiterbahnen beherbergt. Dadurch ist es der Software möglich, das passende Bildmaterial als Schablone für die Suche zu verwenden. Standardzellen werden in der grafischen Repräsentation auf allen Ebenen dargestellt. Im Logik-Modell werden sie jedoch auf der ersten Schicht gespeichert, die vom Typ „metal“ ist.

Unter dem Menüpunkt „Recognition“ stehen drei Verfahren zur Verfügung: Das normale „Template Matching“ sucht an beliebigen Stellen im ausgewählten Bildbereich nach Standardzellen. „Template Matching in Rows“ bzw. „Template Matching in Columns“ sind modifizierte Versionen des Suchverfahrens, bei denen die Schablone entlang von Hilfslinien verschoben wird. Dazu müssen die Hilfslinien oberhalb der Zeilen bzw. links von den Spalten definiert sein. Diese Modifikation schränkt nicht nur den Suchraum signifikant ein, sondern reduziert auch erheblich falsch-positive Treffer.

In der Software Degate wird der Tatsache Rechnung getragen, dass Standardzelleninstanzen auf dem Chip häufig als Spiegelung vorkommen. Bei der Suche wird nach allen möglichen Orientierungen gesucht. Die Suche kann aber auch gezielt auf bestimmte Orientierungen eingeschränkt werden.

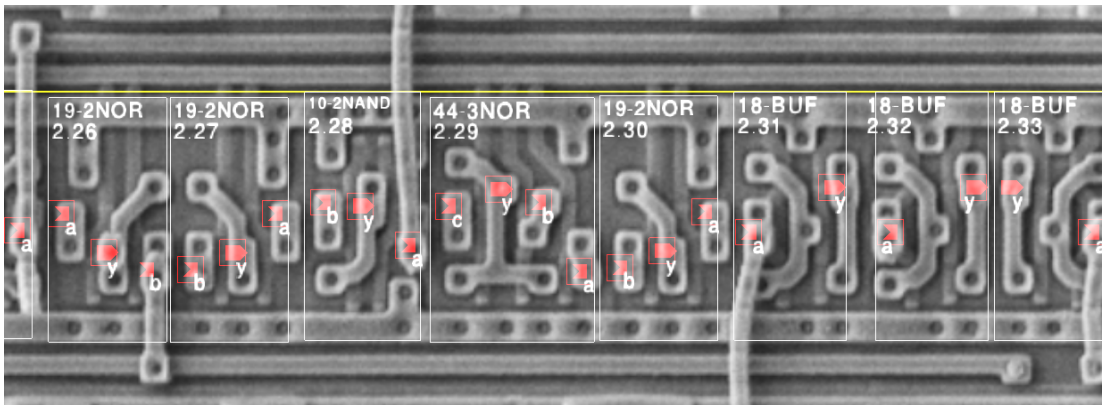

Abbildung A.9: Darstellung von platzierten Standardzellen in Degate.

Wird einer dieser Menüpunkte aufgerufen, öffnet sich zunächst ein Dialogfenster, in dem man einen oder mehrere Standardzellentypen auswählt, die gesucht werden sollen. In einem weiteren Dialogfenster können die Parameter für den Suchvorgang festgelegt werden. Die Funktionsweise des Suchverfahrens wird in Abschnitt 7 im Detail behandelt.

Wenn eine Bildinformation als Schablone für eine Standardzelle festgelegt wurde, ist es zunächst günstig, in einem kleineren Bildausschnitt nach weiteren Instanzen zu suchen. Wenn die Software ein paar Standardzelleninstanzen gefunden hat, können diese markiert werden. Unter dem Menüpunkt ► *Gate / Use images of selected gates* steht eine Funktion zur Verfügung, mit der Bilddaten von mehreren Standardzelleninstanzen zu einem Mischbild zusammengeführt werden, das dann als Muster

für weitere Suchvorgänge verwendet wird. Das zugrundeliegende Problem besteht darin, dass eine einzelne Schablone nur bedingt als Repräsentant geeignet ist. Infolge des Polierens entstehen regelmäßig Störungen auf der Chipoberfläche, z. B. dass Bereiche in unterschiedlichem Maße abgetragen wurden. Für lokale Suchen kann man sich zwar jeweils eine geeignete lokale Schablone wählen. Für eine globale Suche mittels eines Mischbildes als Schablone erhöht sich die Erkennungsrate, und die Fehlerkennungsrate verringert sich.

Die Funktion zum Mischen kann auf eine Menge von Standardzelleninstanzen ausgeführt werden. Die Instanzen dürfen dabei von verschiedenen Typen sein. Die Funktion bezieht unterschiedliche Orientierungen von Instanzen mit ein.

A.6 Electrical & Design Rule Checks

Die Entwicklung integrierter Schaltkreise bewegt sich häufig am Limit des technisch machbaren. Die Integrationsdichte ist für einen Zielprozess in der Halbleitertechnik optimiert: Transistoren, Leiterbahnen und Durchkontaktierungen sind platzsparend dimensioniert und verlegt. Denn um so mehr Chips pro Wafer produziert werden, desto geringer sind die Herstellungskosten pro Chip. Zwar ließen sich mit feineren Strukturgrößen mehr Schaltkreise mit einen Wafer gewinnen, jedoch steigen dann die Produktionskosten für einen Wafer. Die Wahl des Halbleiterprozesses ist daher eine Abwägung zwischen technisch zu erreichenden Parametern (etwa Taktfrequenz des Chips) und dem finanziellen Aufwand.

Um Fehler rechtzeitig zu erkennen, ist es notwendig, die Einhaltung von Regeln zu prüfen. Diese Prüfung ist Teil der Layout-Verifikation und betrifft Layout-Daten und elektrische Parameter. Entsprechend werden diese Prüfungen „Design Rule Checks“ und „Electrical Rule Checks“ genannt. Beispielsweise prüfen „Design Rule Checks“, ob Mindestabstände zwischen Leiterbahnen eingehalten werden. Die dafür relevanten Regeln sind abhängig vom Halbleiterprozess. Electrical Rule Checks stellen sicher, dass elektrische Parameter unverletzt sind. Z. B. kann ein Ausgangsport einer Standardzelle nur begrenzt viele Eingänge anderer Standardzellen treiben. Dieser als Fan-out bezeichnete Parameter ist in den Standardzellenbibliotheken festgelegt.

Beim Reverse-Engineering helfen Rule Checks, Fehler zu entdecken. Die Komplexität der Regeln ist geringer und die meisten Regeln müssen nicht geprüft werden, denn der Chip funktioniert bereits.

Degate unterstützt Rule Checks. Regelwerksverletzungen können über die GUI eingesehen und bearbeitet werden. Electrical Rule Checks erkennen:

- nicht verbundene Eingänge von Standardzellen,
- Verbindungen mehrerer Ausgangs-Ports,
- Eingangs-Ports, die nicht mit wenigstens einem Ausgangsport verbunden sind.

Design Rule Checks erkennen:

- dicht beieinander liegende elektrisch verbindbare Objekte, die nicht in einem Netz miteinander verbunden sind,
- Durchkontaktierungen für die kein Gegenstück auf dem benachbarten Layer existiert.

Design Rules in Degate sind nicht ausschließlich auf Layoutprüfung beschränkt. Vielmehr werden mit ihnen Layout-bezogene Verbindungsprobleme erkannt. Sie sind damit letztendlich eine spezielle Form von Electrical Rules.

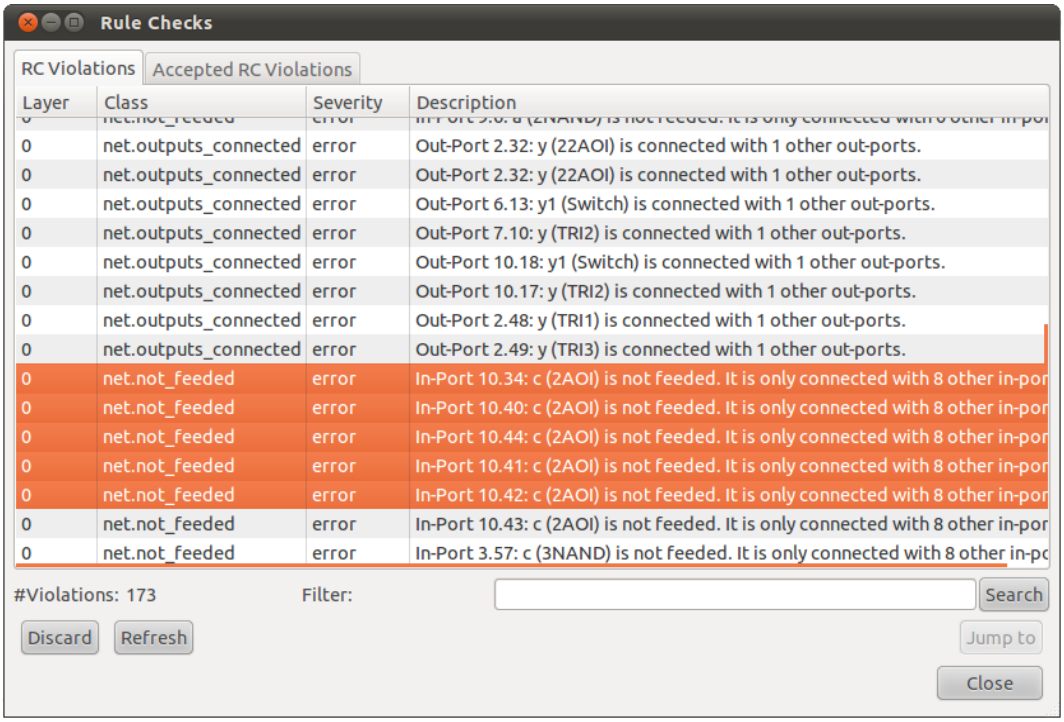

Abbildung A.10: Rule-Check-Dialog

Über den Menüeintrag ▶ *Logic / Rule Checks* wird der RC-Dialog gestartet. Dieser führt die Rule Checks aus und zeigt erkannte und potentielle Probleme in einer

Tabelle an. Wie ernst eine RC-Verletzung ist, gibt die Spalte *Severity* an, anhand der Warnungen und Fehler unterscheidbar sind. Auskunft über die Art des Problem ist der Spalte *Class* zu sehen. *Description* beschreibt den Fehler und welches Logikmodellobjekt betroffen ist. Bei Auswahl einer Tabellenzeile aktiviert sich der Schalter *Jump to*, mittels dem man das betroffene Objekt im Hauptfenster anzeigen kann.

Die Liste der Regelverletzungen lässt sich durchsuchen. Unter der Tabelle im RC-Dialog ist dazu eine Suchmaske platziert. Bei deren Benutzung werden lediglich Zeilen angezeigt, die den Suchausdruck enthalten.

RC-Verletzungen für einzelne Logikmodellobjekte können deaktiviert werden. Während Fehler auf tatsächliche Probleme hinweisen, bedeuten Warnungen, dass Degate das Vorhandensein eines Fehlers nicht entscheiden kann. RC-Fehler sollten daher nie ignoriert werden. Ignorierte RC-Verletzungen sind persistent und werden in einer speraten XML-Datei gespeichert. Über die Registerkarte „Accepted RC-Violations“ kann die Deaktivierung von RC-Problemmeldungen manuell zurückgenommen werden. Degate fragt dann, ob ein erneuter RC-Lauf ausgeführt werden soll. Für behobene Regelverletzungen werden ggf. ignorierte Meldungen automatisch aus der Liste akzeptierter Verletzungen entfernt.

A.7 Gruppieren von Standardzellen

Module sind ein Ordnungsprinzip beim vorwärtsgerichteten Schaltkreisentwurf und dienen der hierarchischen Gliederung und Wiederverwendung von Funktions- bzw. Strukturbeschreibungen (siehe Kapitel 9). Standardzelleninstanzen können in Degate zu Modulen zusammengefasst werden – zum Festhalten von Ergebnissen bei der Analyse von Netzlisten und um Beschreibungen für Module exportieren zu können. Jedes Degate-Projekt besteht aus einem Hauptmodul, in dem zunächst alle Standardzelleninstanzen gruppiert sind. Der Benutzer kann Untermodule hinzufügen und diesen Standardzelleninstanzen zuweisen. Dadurch entsteht eine Baumstruktur von Modulen.

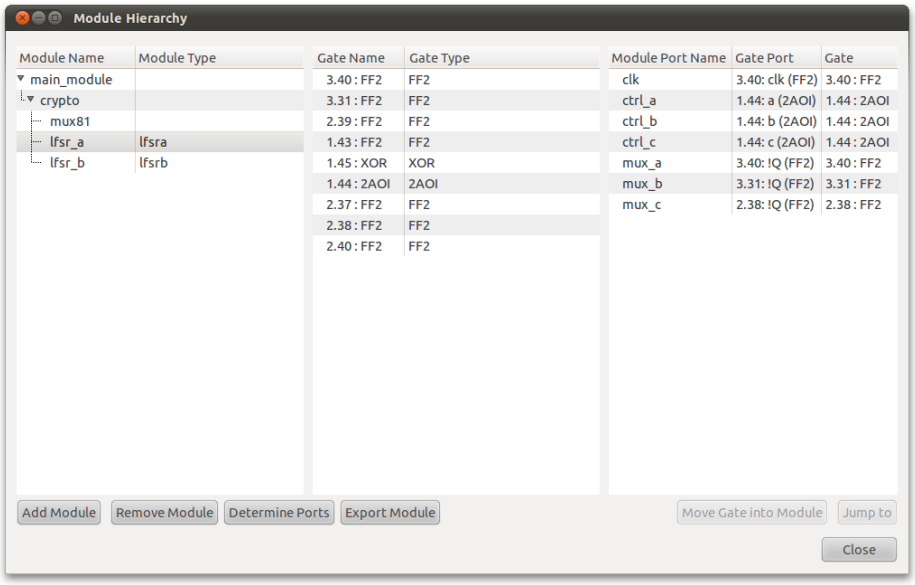

Abbildung A.11: Zusammenfassung von Standardzelleninstanzen zu Modulen.

A.8 Weitere Funktionen

Annotationen sind eine Art Lesezeichen, die auf einem Chip-Layer eingetragen werden können. Sie dienen dem Wiederfinden von Stellen oder Bereichen und deren Beschriftung. Über die Werkzeugleiste bzw. über den Menüeintrag ▶ *Logic / Annotations* ist eine Liste aller Annotationen einsehbar. Diese Liste erlaubt, direkt zur Annotationen zu navigieren.

Bildmaterial wird häufig nur von den Bereichen des Chips aufgenommen, die für eine Analyse relevant sind, beispielsweise wo kryptografische Funktionen vermutet werden oder eine Detailaufnahme von einer Speicherbank, um den Speichertyp identifizieren zu können. Um die Übersicht zu behalten, welchen Bereiche fotografiert wurden, können Unterprojekte benutzt werden. In der Benutzeroberfläche werden diese als Annotationen dargestellt. Durch Doppelklick mit der Maus wechselt man zu einem Unterprojekt. Über den Menüeintrag ▶ *Project / Open parent project* gelangt man zum Elternprojekt zurück. Technisch sind sie eigenständige Degate-Projekte. Projekte stellen einen eigenen Container für Bildmaterial und Objekte dar. Im jeweiligen Elternprojekt wird lediglich die Information gespeichert, wo diese Unterprojekte im Logikmodell verortet sind und wo die Projektdaten im Dateisystem liegen.

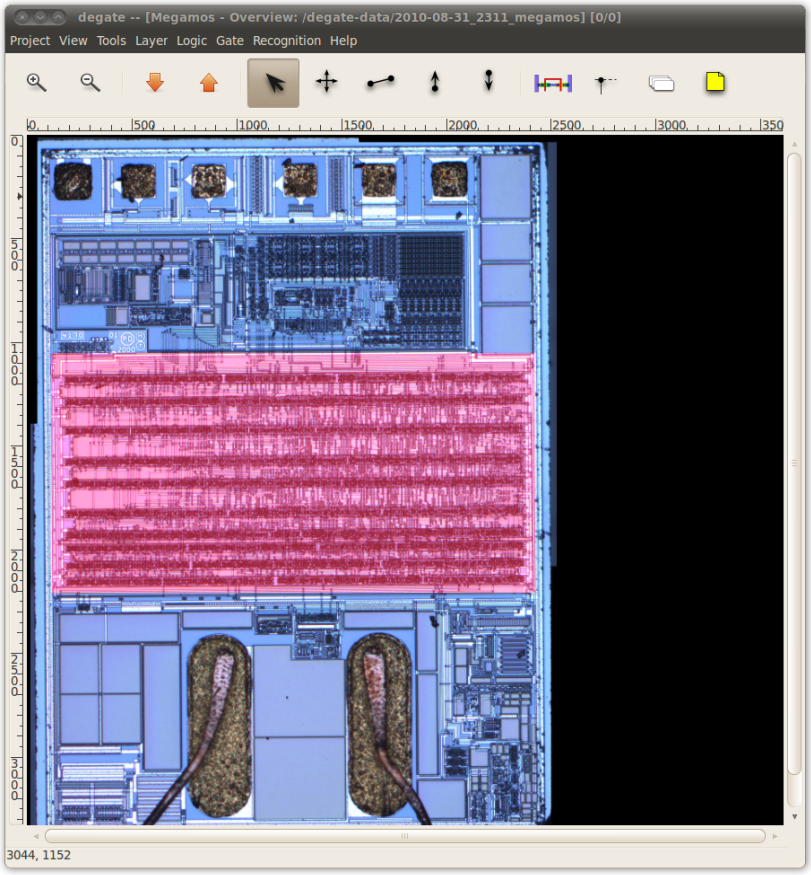

Abbildung A.12: Darstellung von Unterprojekten.

Objekte des Logikmodells, z. B. platzierte Standardzellen, Leiterbahnen und Annotationen, haben Namen. Für platzierte Standardzellen hat es sich als sinnvoll erwiesen, ihnen einen Namen zu geben, der die Position beschreibt. Z. B. bedeutet der Name „5.23“, dass es sich um das 23. Gatter in der fünften Spalte handelt. Dadurch ist es möglich, anhand des Namens räumliche Beziehungen zu anderen Standardzelleninstanzen herzustellen. Da funktional zusammenhängende Standardzellen benachbart platziert werden, gilt heuristisch, dass räumliche Beziehung funktionale Beziehung impliziert. Für die automatische Benennung anhand der Position gibt es unter dem Menüeintrag ▶ *Gate / Generate names along rows* bzw. ▶ *... along columns* entsprechende Funktionen.

Das Dialogfenster „Connection Inspector“ dient dazu, elektrische Verbindungen im Logikmodell nachzuvollziehen. Wird ein Objekt mit der Maus angeklickt, wird es im „Connection Inspector“ in der mittleren Spalte angezeigt. In Abbildung A.13 ist dies

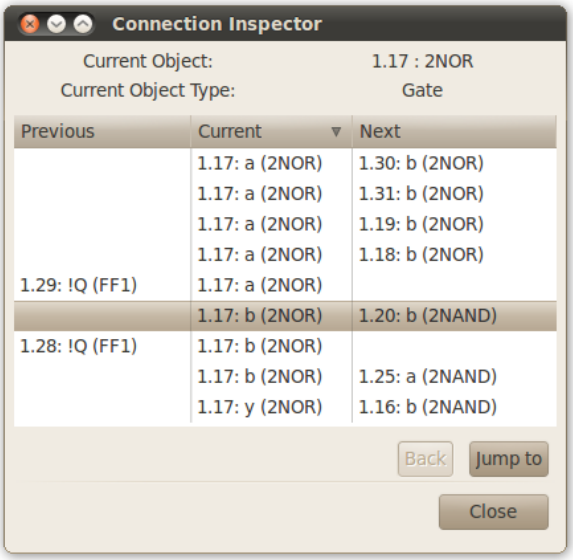

Abbildung A.13: Das Dialogfenster „Connection Inspector“

an einem Beispiel eines NOR-Gatters illustriert. Das NOR-Gatter hat den Namen „1.17“, zwei Eingänge a und b und einen Ausgang y. Diese Ports sind in der mittleren Spalte aufgelistet. In der linken Spalte sind Ausgänge von Standardzellen angegeben, die mit Eingängen des ausgewählten Objektes elektrisch verbunden sind („fan in“). Hier sind dies negierte Ausgänge von zwei verschiedenen Flipflops. Entsprechend sind in der rechten Spalte Eingänge verzeichnet, die mit Ports des ausgewählten Gatters verbunden sind („fan out“). Elektrisch verbindbare Objekte, die keine Ein- oder Ausgänge darstellen, z. B. Leiterbahnen, werden sowohl in der linken als auch rechten Spalte angezeigt.

Das Fenster weist einen Button „Jump to“ auf, um zu elektrisch verbundenen Objekte zu springen. Dabei merkt sich das Fenster, von welchem Element der Sprung erfolgte, so dass man mittels des Buttons „Back“ zurückspringen kann. Der „Connection Inspector“ ist nicht modal, d. h. er kann geöffnet bleiben, während der Benutzer im Hautfenster der Anwendung arbeitet.

Mitunter ist es wünschenswert, Ports von Standardzellen entsprechend ihrer Funktion mit Farben zu kennzeichnen, beispielsweise Reset-Eingänge stets rot und Clock-Schnittstellen grün. Der Dialog „Port Colors“ bietet die Möglichkeit, Farben für Ein- und Ausgänge von Standardzellen projektglobal festzulegen. Die Zuordnung erfolgt dabei anhand von Portnamen.

Anhang B

Anwendungsbeispiel: Legic prime

B.1 Über Legic prime

Legic prime ist ein proprietäres RFID-System der Firma Legic Identsystems AG, welches Anfang der 1990er Jahre entwickelt wurde. Es wird u. a. für Zugangskontroll- und Micropaymentsysteme verwendet. Legic prime verwendet ein proprietäres „Verschlüsselungsverfahren“, um über die Luftschnittstelle übertragene Daten gegen Mitlesen zu „schützen“. Das Protokoll auf Bitübertragungsschicht ist seit dem Versuch bekannt, es als ISO 14443 Annex F zu standardisieren. Protokolle auf höheren Schichten einschließlich der „Verschlüsselung“ waren lange Zeit nicht öffentlich.

Die Aufklärung des „Verschlüsselungssystem“ war der erste Feldversuch, der mit einer früheren Version von Degate durchgeführt wurde. Jan Krißler, Henryk Plötz und Karsten Nohl haben die Sicherheit des RFID-Verfahrens untersucht und ihre Ergebnisse darüber publiziert. [NP07][NP11]

Ein Transponder des Typs Legic prime hat eine geringe Komplexität – nur 604 platzierte Standardzellen sind notwendig, um das proprietäre Protokoll zu implementieren. Ferner ist eine Speicherlogik vorhanden, um Daten persistent abzulegen[1]. Der Chip ist in Abbildung B.1 dargestellt. Der Bereich in der Mitte enthält die untersuchte Logik. Im unteren Teil ist der Speicher zu erkennen sowie die Elektronik für die Speicheransteuerung. Im oberen Bereich sind die beiden Anschlüsse für die externe Spule zu sehen. Diese Spule dient als Antenne.

[1] Die im Folgenden angegebenen statistischen Werte beziehen sich auf die Grundmenge von den 604 Standardzellen.

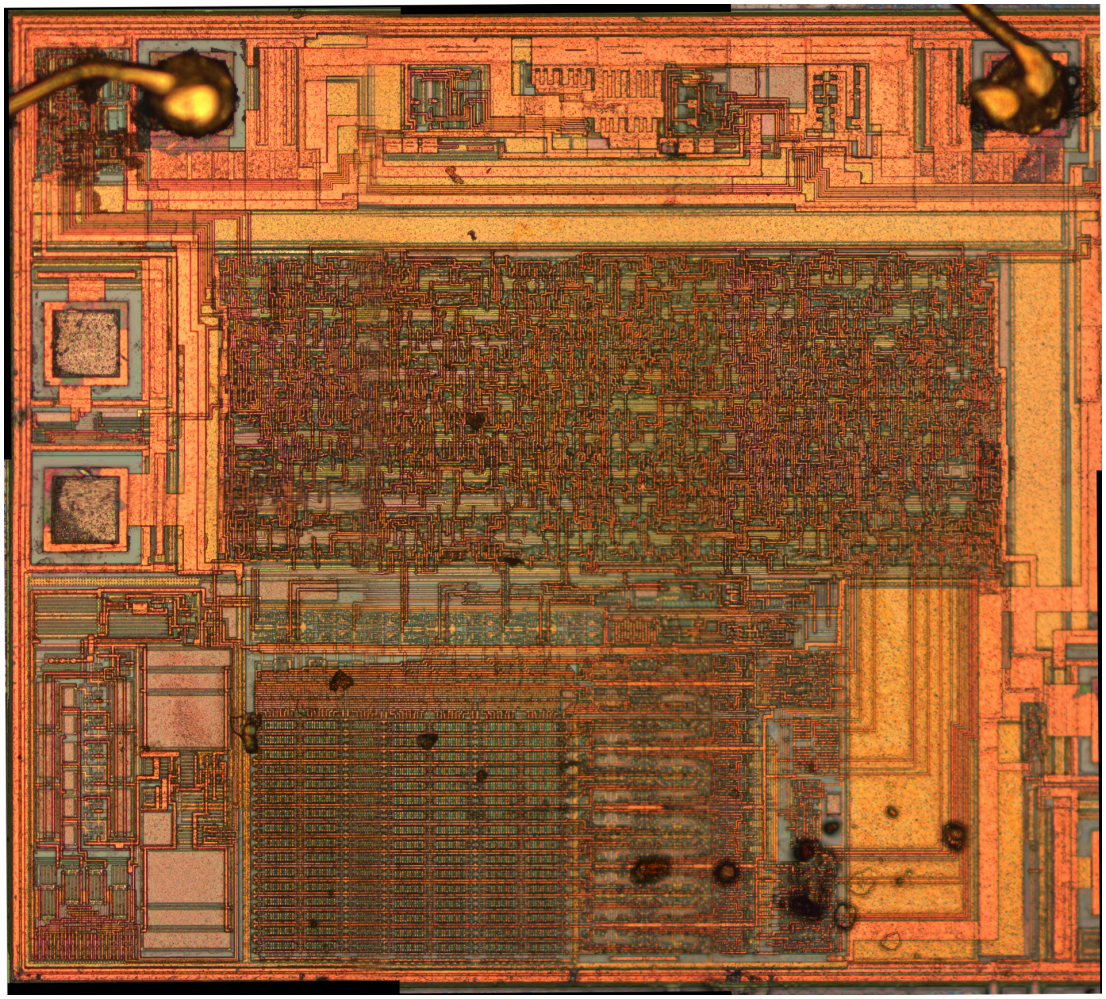

Abbildung B.1: Der Chip in einem Transponder des Typs Legic prime

Wenngleich die Resultate der Analyse bereits veröffentlicht wurden und eine C-Version des „Verschlüsselungsverfahrens“ im öffentlichen Repository des Proxmark-Projektes[2] vorhanden ist, so ist die Implementation auf Hardwareebene bisher nicht beleuchtet worden. Gegenstand dieses Anhangs ist es, dies nachzuholen und zu zeigen, dass ein ein Export von Netzlisteninformation aus Degate heraus möglich ist.

Die im Folgenden angegebenen Verilog-Programmtexte für Standardzellen und Module wurden aus Degate exportiert. Die Verhaltensbeschreibungen der Standardzellen sind manuell anhand des Bildmaterials erstellt worden. Die strukturellen Beschreibungen der Module wurden mittels Degate automatisch generiert. Lediglich die Zuord-

[2]Proxmark.org – A Radio Frequency IDentification Tool, `http://www.proxmark.org/`, besucht am 23. Mai 2011

nung von Standardzellen zu Modulen und die Benennung der Modulschnittstellen erfolgte von Hand.

Der exportierte Verilog-Code wurde in Teilen manuell nachbearbeitet: Dies betrifft die zusätzlich hinzugefügten Modulschnittstellen mit dem Namensprefix `debug_` zur Inspektion der Werte in den Schieberegistern.

B.2 Das Verschlüsselungsverfahren von Legic prime

B.2.1 Überblick

Das „Verschlüsselungsverfahren“ von Legic prime basiert auf einem Stromchiffrierer, der transponder- und leserseitig mit einem sieben Bit kurzen Zufallswert initialisiert wird. Dieser Zufallswert wird beim Aufbau der Kommunikation vom Lesegerät an den Transponder als Klartext gesendet. [NP11] Mit jeder steigenden Taktflanke erzeugt der Stromchiffrierer ein Schlüsselbit, welches mit dem Klartext mittels einer XOR-Operation verknüpft wird, um den zu übertragenden Chiffretext zu erzeugen.

Der Stromchiffrierer besteht aus 52 Standardzellen. Dies entspricht knapp 9 % der Menge aller Standardzellen. Die Standardzellen, welche den Stromchiffrierer bilden wurden in Degate manuell ausgewählt und einem Modul namens „crypto“ zugeordnet. Die Modulgrenzen sind fließend. Dieses rekonstruierte Module „crypto“ erfordert weitere Logik zur Ansteuerung, so dass letztendlich die „Verschlüsselungslogik“ etwa 10 % der platzierten Standardzellen ausmacht.

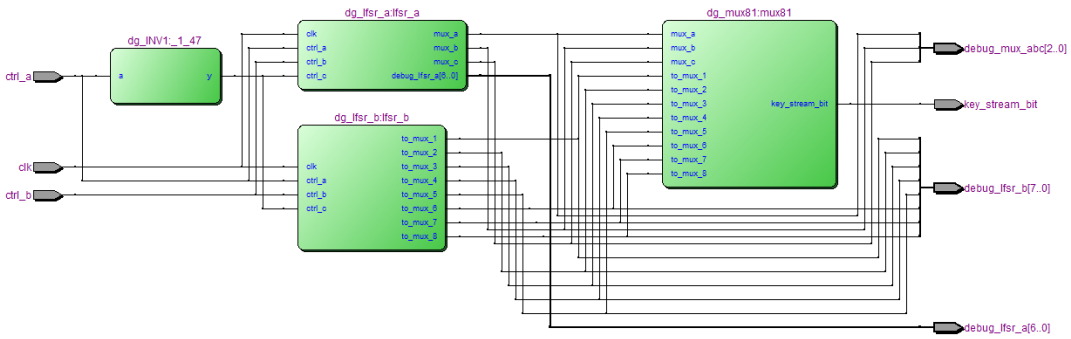

Abbildung B.2: Der Schaltplan zum „Verschlüsselungsverfahren“ von Legic prime im Überblick.

Der Programm-Code in Listing B.1 gibt die Strukturbeschreibung des Hauptmoduls des Schlüsselstromgenerators wieder. Der Generator besteht aus zwei linear rückgekoppelten Schieberegistern und einem Multiplexer. Dabei bilden beide LFSRs Pseudozufallszahlengeneratoren. Einer der beiden LFSRs (`dg_lfsr_b`) erzeugt Pseudozufallsbits und das andere LFSR (`dg_lfsr_a`) erzeugt eine Zufallszahl zwischen null und sieben zur Ansteuerung des Multiplexers, welcher das Schlüsselstrombit aus `dg_lfsr_b` wählt.

Listing B.1: dg_crypto.v

```
/**
 * This is a structural Verilog implementation for a module of type crypto.
 */

module dg_crypto (clk, ctrl_a, ctrl_b,
                  key_stream_bit,
                  debug_lfsr_a,
                  debug_lfsr_b,
                  debug_mux_abc);

  input clk, ctrl_a, ctrl_b;
  output key_stream_bit;

  // output ports for debugging
  output [6:0] debug_lfsr_a;
  output [7:0] debug_lfsr_b;
  output [2:0] debug_mux_abc;

  // net definitions
  wire   to_mux_1, to_mux_2, to_mux_3, to_mux_4,
         to_mux_5, to_mux_6, to_mux_7, to_mux_8;
  wire   mux_a, mux_b, mux_c, ctrl_c;

  // continous assignments for debug lines
  assign debug_mux_abc = {mux_a, mux_b, mux_c};
  assign debug_lfsr_b = {to_mux_1, to_mux_2, to_mux_3, to_mux_4,
                         to_mux_5, to_mux_6, to_mux_7, to_mux_8};

 // sub-modules

  dg_INV1 _1_47 (.a (ctrl_a), .y (ctrl_c) );

  dg_mux81 mux81 (.key_stream_bit (key_stream_bit),
                  .mux_a (mux_a),
                  .mux_b (mux_b),
                  .mux_c (mux_c),
                  .to_mux_1 (to_mux_1),
                  .to_mux_2 (to_mux_2),
                  .to_mux_3 (to_mux_3),
                  .to_mux_4 (to_mux_4),
                  .to_mux_5 (to_mux_5),
```

```
                    .to_mux_6 (to_mux_6),
43                  .to_mux_7 (to_mux_7),
                    .to_mux_8 (to_mux_8) );
45
  dg_lfsr_a lfsr_a (.clk (clk),
47                   .ctrl_a (ctrl_a),
                     .ctrl_b (ctrl_b),
49                   .ctrl_c (ctrl_c),
                     .mux_a (mux_a),
51                   .mux_b (mux_b),
                     .mux_c (mux_c),
53                   .debug_lfsr_a(debug_lfsr_a));

55  dg_lfsr_b lfsr_b (.clk (clk),
                     .ctrl_a (ctrl_a),
57                   .ctrl_b (ctrl_b),
                     .ctrl_c (ctrl_c),
59                   .to_mux_1 (to_mux_1),
                     .to_mux_2 (to_mux_2),
61                   .to_mux_3 (to_mux_3),
                     .to_mux_4 (to_mux_4),
63                   .to_mux_5 (to_mux_5),
                     .to_mux_6 (to_mux_6),
65                   .to_mux_7 (to_mux_7),
                     .to_mux_8 (to_mux_8) );
67
endmodule
```

Das Programm aus Listing B.2 lädt zunächst den Initialisierungsvektor in die Schieberegister und erzeugt dann mit jedem Taktschritt ein Schlüsselstrombit. Die Steuerung dieser beiden Phasen erfolgt über eine kombinatorische Schaltung, die hier als „2AOI“ benannt wurde und eine eigene Standardzelle bildet. Der Name „AOI“ gibt normalerweise an, dass die boolsche Schaltfunktion von innen nach außen gelesen einer AND-OR-Inverter-Struktur entspricht. Der Name ist an der Stelle nicht korrekt, da das Ergebnis auch einen Tristatezustand annehmen kann. Die Funktion des 2AOI wird später beschrieben.

Listing B.2: test_crypto.v

```
1 /**
   * This is Verilog testbench for the Legic's crypto module.
3  *
   * Compile and run: iverilog -DIV=\'hab test_crypto.v && ./a.out
5   */

7 `include "crypto.v"
  `timescale 1ns/100ps
9
  `ifndef IV
11  `define IV 'h55  // default initialisation vector for the obfuscation function
```

```
`endif

module testbench;

   reg [8:0] iv      = ((`IV <<1) | 1);
   reg clk, ctrl_a, ctrl_b;

   wire key_stream_bit;
   wire [6:0] debug_lfsr_a;
   wire [7:0] debug_lfsr_b;
   wire [2:0] debug_mux_abc;

   integer i;

   // create an instance of the device to test
   dg_crypto dut(.clk(clk),
                 .ctrl_a(ctrl_a),
                 .ctrl_b(ctrl_b),
                 .key_stream_bit(key_stream_bit),
                 .debug_lfsr_a(debug_lfsr_a),
                 .debug_lfsr_b(debug_lfsr_b),
                 .debug_mux_abc(debug_mux_abc));

   initial begin

      // initialize clock value
      clk <= 1'b0;


      /**
       * Shift the IV into the LFSRs.
       */

      ctrl_a = 1; // mode for shifting data into the LFSRs

      for(i=0; i < 8; i = i + 1)
        begin

           // The 2-AOI allows only inverted bits to be shifted into the LFSRs.
           ctrl_b = iv[i];

           // Generate a rising clock edge.
           #10 clk = 1'b0; #10 clk = 1'b1; #10 clk = 1'b0;

           // Print state
           $display("[iv-shift] bit: %d a=[%b] b=[%b] count=%1d  a=%h b=%h",
                    key_stream_bit,
                    debug_lfsr_a, debug_lfsr_b,
                    i,
                    debug_lfsr_a, debug_lfsr_b);
        end


      $display("-----");
```

```
67
        /*
69       * Loop and generate key bits.
         */
71
        ctrl_a = 0; // set loop mode
73      ctrl_b = 0;

75      for(i=0; i < 42; i = i + 1)
          begin
77
            // Generate a rising clock edge.
79          #10 clk = 1'b0; #10 clk = 1'b1; #10 clk = 1'b0;

81          // Print state.
            $display({"[round] key bit: %d  lfsr_a=[%b] lfsr_b=[%b] ",
83                    "abc=%1d count=%-3d a=%h b=%h"},
                    key_stream_bit,
85                  debug_lfsr_a, debug_lfsr_b,
                    debug_mux_abc, i+1,
87                  debug_lfsr_a, debug_lfsr_b);
          end
89
    end
91
endmodule // testbench
```

Das Testprogramm erzeugt die nachstehende Ausgabe. In der Phase „iv-shift“ wird der Initialisierungsvektor in die LFSRs geladen. Dies ist hier für den Initialisierungsvektor 0x55 angegeben. In der Phase „round“ erzeugen die Schieberegister mittels Rückkopplung neue Registerzustände. Die Ausgabe zeigt das Schlüsselstrombit, den Inhalt der beiden Schieberegister als Binär- und Hexadezimalzahl, die Eingabe für den Multiplexer „abc“ zur Auswahl des Schlüsselbits sowie die Rundennummer.

Listing B.3: Ausgabe der Testbench

```
  [iv-shift] bit: 1 a=[0xxxxxx] b=[0xxxxxxx] count=0  a=Xx b=Xx
2 [iv-shift] bit: 1 a=[00xxxxx] b=[00xxxxxx] count=1  a=Xx b=Xx
  [iv-shift] bit: 1 a=[100xxxx] b=[100xxxxx] count=2  a=4x b=Xx
4 [iv-shift] bit: 1 a=[0100xxx] b=[0100xxxx] count=3  a=2X b=4x
  [iv-shift] bit: 0 a=[10100xx] b=[10100xxx] count=4  a=5X b=aX
6 [iv-shift] bit: 0 a=[010100x] b=[010100xx] count=5  a=2X b=5X
  [iv-shift] bit: 0 a=[1010100] b=[1010100x] count=6  a=54 b=aX
8 [iv-shift] bit: 0 a=[0101010] b=[01010100] count=7  a=2a b=54
------
10 [round] key bit: 0  lfsr_a=[0010101] lfsr_b=[10101010] abc=5 count=1   a=15 b=aa
   [round] key bit: 0  lfsr_a=[1001010] lfsr_b=[01010101] abc=2 count=2   a=4a b=55
12 [round] key bit: 1  lfsr_a=[1100101] lfsr_b=[00101010] abc=4 count=3   a=65 b=2a
   [round] key bit: 0  lfsr_a=[0110010] lfsr_b=[10010101] abc=1 count=4   a=32 b=95
14 [round] key bit: 0  lfsr_a=[0011001] lfsr_b=[11001010] abc=3 count=5   a=19 b=ca
```

```
[round] key bit: 0  lfsr_a=[1001100] lfsr_b=[01100101] abc=6 count=6   a=4c b=65
[round] key bit: 0  lfsr_a=[1100110] lfsr_b=[00110010] abc=4 count=7   a=66 b=32
[round] key bit: 0  lfsr_a=[1110011] lfsr_b=[00011001] abc=1 count=8   a=73 b=19
[round] key bit: 0  lfsr_a=[0111001] lfsr_b=[00001100] abc=3 count=9   a=39 b=0c
[round] key bit: 0  lfsr_a=[1011100] lfsr_b=[00000110] abc=7 count=10  a=5c b=06
[round] key bit: 1  lfsr_a=[1101110] lfsr_b=[10000011] abc=6 count=11  a=6e b=83
[round] key bit: 0  lfsr_a=[1110111] lfsr_b=[01000001] abc=5 count=12  a=77 b=41
[round] key bit: 0  lfsr_a=[0111011] lfsr_b=[10100000] abc=3 count=13  a=3b b=a0
[round] key bit: 0  lfsr_a=[1011101] lfsr_b=[11010000] abc=7 count=14  a=5d b=d0
[round] key bit: 0  lfsr_a=[0101110] lfsr_b=[11101000] abc=6 count=15  a=2e b=e8
[round] key bit: 1  lfsr_a=[0010111] lfsr_b=[01110100] abc=5 count=16  a=17 b=74
[round] key bit: 1  lfsr_a=[1001011] lfsr_b=[10111010] abc=2 count=17  a=4b b=ba
[round] key bit: 1  lfsr_a=[0100101] lfsr_b=[01011101] abc=4 count=18  a=25 b=5d
[round] key bit: 0  lfsr_a=[1010010] lfsr_b=[10101110] abc=1 count=19  a=52 b=ae
[round] key bit: 0  lfsr_a=[1101001] lfsr_b=[11010111] abc=2 count=20  a=69 b=d7
[round] key bit: 0  lfsr_a=[0110100] lfsr_b=[11101011] abc=5 count=21  a=34 b=eb
[round] key bit: 1  lfsr_a=[0011010] lfsr_b=[11110101] abc=3 count=22  a=1a b=f5
[round] key bit: 1  lfsr_a=[0001101] lfsr_b=[11111010] abc=6 count=23  a=0d b=fa
[round] key bit: 1  lfsr_a=[1000110] lfsr_b=[01111101] abc=4 count=24  a=46 b=7d
[round] key bit: 1  lfsr_a=[1100011] lfsr_b=[10111110] abc=0 count=25  a=63 b=be
[round] key bit: 1  lfsr_a=[0110001] lfsr_b=[11011111] abc=1 count=26  a=31 b=df
[round] key bit: 0  lfsr_a=[1011000] lfsr_b=[01101111] abc=3 count=27  a=58 b=6f
[round] key bit: 1  lfsr_a=[1101100] lfsr_b=[10110111] abc=6 count=28  a=6c b=b7
[round] key bit: 0  lfsr_a=[1110110] lfsr_b=[11011011] abc=5 count=29  a=76 b=db
[round] key bit: 0  lfsr_a=[1111011] lfsr_b=[11101101] abc=3 count=30  a=7b b=ed
[round] key bit: 0  lfsr_a=[0111101] lfsr_b=[01110110] abc=7 count=31  a=3d b=76
[round] key bit: 1  lfsr_a=[1011110] lfsr_b=[10111011] abc=7 count=32  a=5e b=bb
[round] key bit: 0  lfsr_a=[1101111] lfsr_b=[11011101] abc=6 count=33  a=6f b=dd
[round] key bit: 1  lfsr_a=[0110111] lfsr_b=[01101110] abc=5 count=34  a=37 b=6e
[round] key bit: 1  lfsr_a=[1011011] lfsr_b=[00110111] abc=3 count=35  a=5b b=37
[round] key bit: 1  lfsr_a=[0101101] lfsr_b=[00011011] abc=6 count=36  a=2d b=1b
[round] key bit: 1  lfsr_a=[1010110] lfsr_b=[00001101] abc=5 count=37  a=56 b=0d
[round] key bit: 0  lfsr_a=[1101011] lfsr_b=[10000110] abc=2 count=38  a=6b b=86
[round] key bit: 0  lfsr_a=[0110101] lfsr_b=[01000011] abc=5 count=39  a=35 b=43
[round] key bit: 0  lfsr_a=[1011010] lfsr_b=[10100001] abc=3 count=40  a=5a b=a1
[round] key bit: 0  lfsr_a=[1101101] lfsr_b=[01010000] abc=6 count=41  a=6d b=50
[round] key bit: 0  lfsr_a=[0110110] lfsr_b=[00101000] abc=5 count=42  a=36 b=28
```

B.2.2 Linear rückgekoppelte Schieberegister

Linear rückgekoppelte Schieberegister werden häufig für die Erzeugung von Pseudozufallszahlen verwendet. Sie bestehen oft aus verketteten D-Flipflops, bei denen der Ausgang eines Flipflops mit dem Eingang des nächsten Flipflops verbunden ist. Mit dem Takt schieben sich die Registerdaten um eine Stelle durch die Kette. Neue Daten werden dadurch gewonnen, dass Bits an ausgewählten Positionen im Schieberegister i. d. R mittels XOR-Operation verknüpft und das Ergebnis wieder in das Schieberegister eingespeist wird.

Die beiden LFSRs im „Verschlüsselungsverfahren“ von Legic prime entsprechen dem Fibonacci-Typ, bei dem die Einspeisung am Beginn der Schieberegisterkette beginnt und die Weiterleitung von Flipflop zu Flipflop ohne Änderung erfolgt. Im Gegensatz dazu sind bei Galois-LFSRs die XOR-Elemente im Datenpfad der Schieberegister.

a	b	c	d	y	Kommentar
0	0	0	0	1	
0	0	0	1	1	
0	0	1	0	1	benutzbar für Rückkoppelung
0	0	1	1	0	benutzbar für Rückkoppelung
0	1	0	0	1	
0	1	0	1	z	
0	1	1	0	1	
0	1	1	1	0	
1	0	0	0	1	IV-Shift: 1
1	0	0	1	1	IV-Shift: 1
1	0	1	0	z	
1	0	1	1	0	
1	1	0	0	0	IV-Shift: 0
1	1	0	1	0	IV-Shift: 0
1	1	1	0	0	
1	1	1	1	0	

Tabelle B.1: Schaltfunktion der Standardzelle „2AOI“

Die Rückkopplung erfolgt in den Schieberegistern `dg_lfsr_a` und `dg_lfsr_b` nicht direkt, sondern wird über die bereits benannte Standardzelle „2AOI“ realisiert, und zwar getrennt für beide Schieberegister. Dieser Standardzellentyp hat vier Eingänge a, b, c, d, einen Ausgang y und realisiert die Schaltfunktion wie in Tabelle B.1 angegeben. Der Ausgang der jeweiligen 2AOI-Standardzelle dient als Eingang für das jeweilige Schieberegister. Das rückgekoppelte Signal ist auf Eingang d geschaltet. Aufgrund eines Inverters, der das Eingangssignal von a invertiert und auf den Eingang c leitet, stehen lediglich die zwei Steuerleitungen a und b zum Umschalten des Modus bereit: Wenn $a = 1$ gilt, kann man mittels des Eingangs b steuern, welcher Wert in die Schieberegister geladen wird. Mit $b = 0$ wird eine 1 geladen, anderenfalls eine 0. Dies erfolgt unabhängig von der Belegung des Ports d. Um den Rückkopplungsmodus zu aktivieren, muss $a = 0$ und $b = 1$ gesetzt werden. Die Ausgabe entspricht dann $\overline{d}$. Als Verknüpfungsoperation wird hier kein XOR verwendet. Vielmehr kommt ein XNOR zum Einsatz, also eine XOR-Operation mit anschließender Negation, welche

die Negation der 2AOI-Standardzelle ausgeleicht. Es gilt $xnor(\overline{p},\overline{q}) = xnor(p,q)$. Die XNOR-Operation ist unabhängig von der Negation der beiden Operanden. Dies gilt ebenso für die XOR-Operation. Entsprechend sind für das Modul `dg_lfsr_b` Inverter in den Rückkopplungspfad eingearbeitet, da drei Operanden XNOR-verknüpft werden.

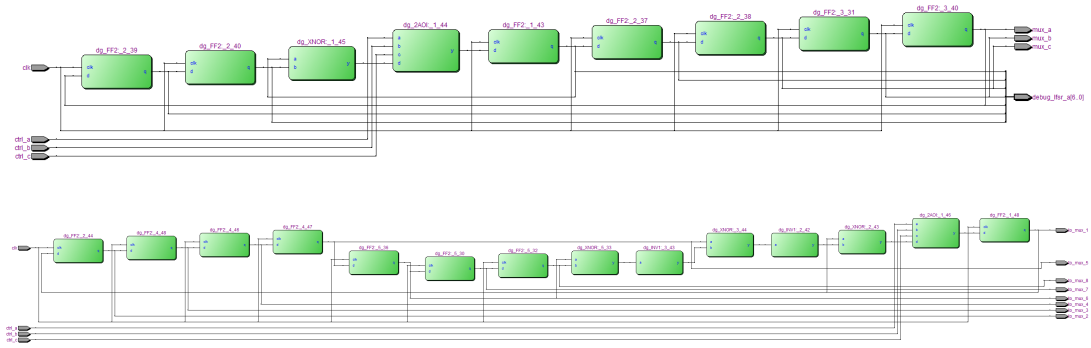

Abbildung B.3: Die beiden Schieberegister: `dg_lfsr_a` (oben) und `dg_lfsr_b` (unten)

Die Programme B.4 und B.5 geben die Strukturbeschreibungen der beiden LFSRs in Verilog an. Die Eingangsleitungen $ctrl\_a$, $ctrl\_b$ und $ctrl\_c$ steuern den 2AOI-Schalter, wobei der Inverter, der das Signal von $ctrl\_a$ invertiert, außerhalb der beiden Module ist. Die Schnittstellen mux_a, mux_b und mux_c sind Ausgänge des Schieberegisters `dg_lfsr_a` und steuern den Multiplexer.

Listing B.4: dg_lfsr_a.v

```
/**
 * This is a structural Verilog implementation for a module of type lfsr_a.
 */


module dg_lfsr_a (clk, ctrl_a, ctrl_b, ctrl_c,
                  mux_a, mux_b, mux_c,
                  debug_lfsr_a);

   input clk, ctrl_a, ctrl_b, ctrl_c;
   output mux_a, mux_b, mux_c;
   output [6:0] debug_lfsr_a;

   // net definitions
   wire         w5, w6, w2, w4, w3, w7;

   assign debug_lfsr_a = {w3, w7, mux_c, mux_b, mux_a, w2, w6};

   // sub-modules
```

```
21    dg_FF2 _1_43 (.clk (clk),
                    .q (w3),
23                  .d (w4) );

25    dg_FF2 _2_37 (.clk (clk),
                    .d (w3),
27                  .q (w7) );

29    dg_FF2 _2_38 (.q (mux_c),
                    .clk (clk),
31                  .d (w7) );

33    dg_FF2 _3_31 (.d (mux_c),
                    .q (mux_b),
35                  .clk (clk) );

37    dg_FF2 _3_40 (.q (mux_a),
                    .d (mux_b),
39                  .clk (clk) );

41    dg_FF2 _2_39 (.d (mux_a),
                    .q (w2),
43                  .clk (clk) );

45    dg_FF2 _2_40 (.d (w2),
                    .clk (clk),
47                  .q (w6) );

49    dg_XNOR _1_45 (.a (w3),
                     .y (w5),
51                   .b (w6) );

53    dg_2AOI _1_44 (.c (ctrl_c),
                     .a (ctrl_a),
55                   .y (w4),
                     .b (ctrl_b),
57                   .d (w5) );

59 endmodule
```

Listing B.5: dg_lfsr_b.v

```
1  /**
    * This is a Verilog implementation for a gate of type lfsr_b.
3   */

5  module dg_lfsr_b (clk, ctrl_a, ctrl_b, ctrl_c,
                     to_mux_1, to_mux_2, to_mux_3, to_mux_4,
7                    to_mux_5, to_mux_6, to_mux_7, to_mux_8);

9     input clk, ctrl_a, ctrl_b, ctrl_c;
      output to_mux_1, to_mux_2, to_mux_3, to_mux_4,
11           to_mux_5, to_mux_6, to_mux_7, to_mux_8;
```

```
13    // net definitions
      wire   w6, w7, w8, w2, w4, w5;
15
      // sub-modules
17
      dg_FF2 _4_48 (.d (to_mux_2),
19                  .q (to_mux_3),
                    .clk (clk) );
21
      dg_FF2 _2_44 (.q (to_mux_2),
23                  .clk (clk),
                    .d (to_mux_1) );
25
      dg_XNOR _3_44 (.y (w2),
27                   .a (to_mux_5),
                     .b (w4) );
29
      dg_INV1 _3_43 (.a (w5), .y (w4) );
31
      dg_2AOI _1_46 (.y (w6),
33                   .b (ctrl_b),
                     .d (w7),
35                   .a (ctrl_a),
                     .c (ctrl_c) );
37
      dg_FF2 _1_48 (.d (w6),
39                  .q (to_mux_1),
                    .clk (clk) );
41
      dg_INV1 _2_42 (.a (w2),
43                   .y (w8) );

45    dg_XNOR _2_43 (.b (w8),
                     .y (w7),
47                   .a (to_mux_1) );

49    dg_FF2 _4_46 (.d (to_mux_3),
                    .q (to_mux_4),
51                  .clk (clk) );

53    dg_FF2 _5_30 (.q (to_mux_7),
                    .d (to_mux_6),
55                  .clk (clk) );

57    dg_FF2 _4_47 (.d (to_mux_4),
                    .q (to_mux_5),
59                  .clk (clk) );

61    dg_FF2 _5_32 (.d (to_mux_7),
                    .q (to_mux_8),
63                  .clk (clk) );

65    dg_XNOR _5_33 (.a (to_mux_6),
                     .y (w5),
```

```
67                      .b (to_mux_8) );

69      dg_FF2 _5_36 (.clk (clk),
                      .d (to_mux_5),
71                    .q (to_mux_6) );

73  endmodule
```

B.2.3 Multiplexer

Der hier beschriebene 8:1-Multiplexer wählt eine Position des Schieberegisters anhand der Steuerleitungen *mux_a*, *mux_b* und *mux_c* aus, um den an der indizierten Stelle gespeicherten Flipflopinhalt als Schlüsselbit zu verwenden. Dazu werden alle Flipflopausgänge des Schieberegisters `dg_lfsr_b` auf Tristateinverter geschaltet. Eine kombinatorische Schaltungen aktiviert entsprechend der drei Steuerleitungen einen der acht Tristateinverter. Die Negation durch die Tristateinverter wird mit einem nachfolgenden Inverter korrigiert.

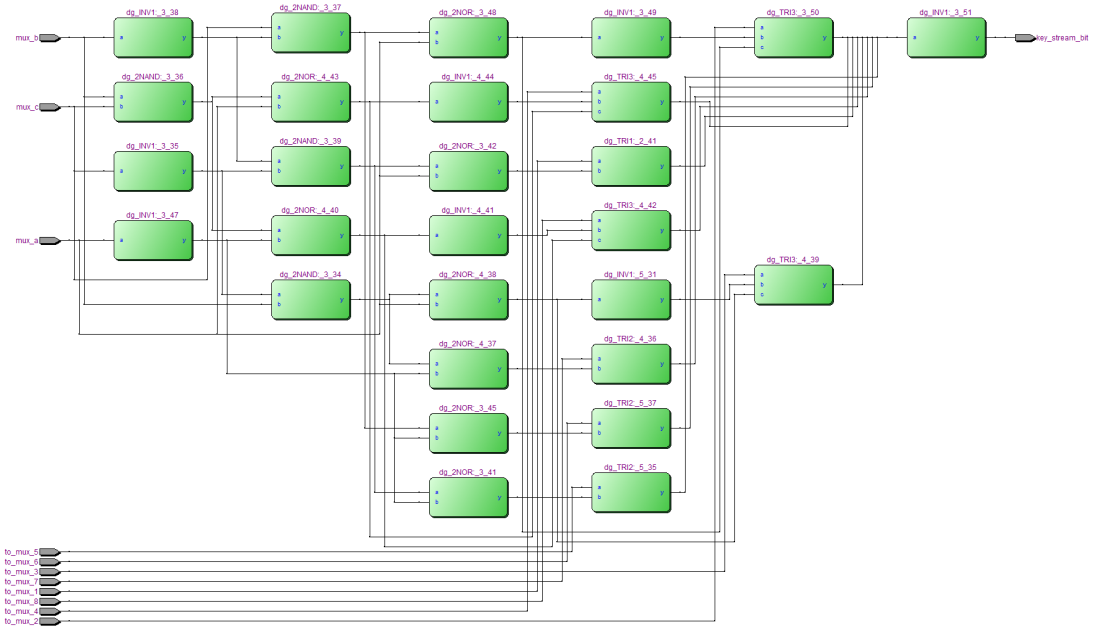

Abbildung B.4: Der 8:1-Multiplexer

Listing B.6: dg_mux81.v

```
1   /**
     * This is a structural Verilog implementation for a module of type mux81.
3    */
```

```
5 module dg_mux81 (mux_a, mux_b, mux_c,
                   to_mux_1, to_mux_2, to_mux_3, to_mux_4,
7                  to_mux_5, to_mux_6, to_mux_7, to_mux_8,
                   key_stream_bit);
9
  input mux_a, mux_b, mux_c,
11       to_mux_1, to_mux_2, to_mux_3, to_mux_4,
         to_mux_5, to_mux_6, to_mux_7, to_mux_8;
13 output key_stream_bit;

15 // net definitions
  wire w5, w4, w13, w9, w7, w8, w15, w6, w10, w16, w12,
17      w19, w14, w3, w1, w2, w0, w18, w11, w17;

19 // sub-modules

21  dg_2NAND _3_39 (.y (w0),
                    .a (w1),
23                  .b (w2) );

25  dg_2NAND _3_36 (.y (w3),
                    .b (mux_c),
27                  .a (mux_b) );

29  dg_2NOR _3_42 (.b (mux_a),
                   .a (w0),
31                 .y (w4) );

33  dg_2NOR _3_41 (.y (w5),
                   .a (w0),
35                 .b (w6) );

37  dg_2NOR _3_48 (.b (mux_a),
                   .y (w7),
39                 .a (w8) );

41  dg_2NOR _3_45 (.b (w6),
                   .y (w9),
43                 .a (w8) );

45  dg_2NAND _3_34 (.y (w10),
                    .b (mux_b),
47                  .a (w2) );

49  dg_2NAND _3_37 (.y (w8),
                    .a (mux_c),
51                  .b (w1) );

53  dg_TRI3 _3_50 (.b (w11),
                   .c (w7),
55                 .a (to_mux_2),
                   .y (w12) );
57
```

```
    dg_INV1 _3_49 (.y (w11), .a (w7) );
59
    dg_INV1 _3_51 (.y (key_stream_bit),
61                  .a (w12) );

63  dg_INV1 _3_47 (.y (w6),
                    .a (mux_a) );
65
    dg_INV1 _3_35 (.y (w2),
67                  .a (mux_c) );

69  dg_INV1 _3_38 (.a (mux_b),
                    .y (w1) );
71
    dg_TRI3 _4_45 (.a (to_mux_4),
73                  .y (w12),
                    .b (w13),
75                  .c (w14) );

77  dg_TRI1 _2_41 (.y (w12),
                    .a (to_mux_1),
79                  .b (w4) );

81  dg_2NOR _4_43 (.y (w14),
                    .b (mux_a),
83                  .a (w3) );

85  dg_2NOR _4_40 (.a (w3),
                    .y (w15),
87                  .b (w6) );

89  dg_2NOR _4_38 (.y (w16),
                    .b (mux_a),
91                  .a (w10) );

93  dg_2NOR _4_37 (.y (w17),
                    .b (w6),
95                  .a (w10) );

97  dg_TRI3 _4_42 (.b (w18),
                    .a (to_mux_8),
99                  .c (w15),
                    .y (w12) );
101
    dg_TRI3 _4_39 (.y (w12),
103                 .c (w16),
                    .a (to_mux_3),
105                 .b (w19) );

107 dg_INV1 _4_41 (.y (w18), .a (w15) );

109 dg_INV1 _4_44 (.a (w14), .y (w13) );

111 dg_TRI2 _4_36 (.a (to_mux_7),
```

```
                    .b (w17),
113                 .y (w12) );

115    dg_INV1 _5_31 (.a (w16),
                    .y (w19) );
117
       dg_TRI2 _5_37 (.a (to_mux_6),
119                 .b (w9),
                    .y (w12) );
121
       dg_TRI2 _5_35 (.b (w5),
123                 .a (to_mux_5),
                    .y (w12) );
125


127
endmodule
```

Das folgende Testprogramm prüft für alle Belegungen der Steuerleitungen, welcher Tristateinverter aktiviert wird.

Listing B.7: test_mux81.v

```
`include "crypto.v" // helper include file, that includes all files
2 `timescale 1ns/100ps

4 module testbench;
     reg [2:0] sel;
6    reg [7:0] in;
     wire y;
8    integer select, i;

10   dg_mux81 dut( .mux_a(sel[2]),
                   .mux_b(sel[1]),
12                 .mux_c(sel[0]),
                   .to_mux_1(in[7]),
14                 .to_mux_2(in[6]),
                   .to_mux_3(in[5]),
16                 .to_mux_4(in[4]),
                   .to_mux_5(in[3]),
18                 .to_mux_6(in[2]),
                   .to_mux_7(in[1]),
20                 .to_mux_8(in[0]),
                   .key_stream_bit(y) );
22
     initial begin
24      for(select = 0; select <= 7; select = select + 1)
          begin
26           sel = select;
             $display("sel: %b", sel);
28
             for(i = 0; i <= 7; i = i + 1)
30             begin
```

```
                in = (1 << i);
32              #10;
                if(y == 1)
34                begin
                    $display("\t\tfor mux input %b input line %1d outputs keybit: %d",
36                          in, 7-i, y);
                  end
38            end
        end
40  end

42 endmodule // testbench
```

B.3 Standardzellenbibliothek

B.3.1 Überblick

Dieser Abschnitt listet alle Standardzellen auf, die beim Reverse-Engineering eines RFID-Transponders vom Typ Legic prime gefunden wurden. Für ausgewählte Standardzellentypen werden nähere Informationen angegeben. Dies betrifft alle Standardzellentypen, die in den zuvor beschriebenen Modulen Verwendung finden.

Standardzellenname	Instanzen	Standardzellenbeschreibung	Abschnitt
FF1	52	DQ-FF	
FF2	27		B.3.2
FF3	5		
Switch	15	set-reset-switch	
2AOI	47		B.3.3
MUX	6	MUX	
XNOR	11	2-XNOR	B.3.4
TRI1	8	tristate inverter	B.3.7
4NOR	5	4-NOR	
3NAND	27	3-NAND	
2NAND	101	2-NAND	B.3.5
22AOI	9	22-AOI	
2NOR	89	2-NOR	B.3.6
3NOR	8	3-NOR	
2OAI	4	2-OAI	
222AOI	1	222-AOI	
TRI3	7	tristate inverter / mux	B.3.9
TRI2	18	tristate inverter	B.3.8
INV1	122	inverter	B.3.10
INV2	32	inverter	
4NAND	8	4-NAND	
FF4	1		
Latch	1	Latch	

B.3.2 Standardzelle: FF2

Layout

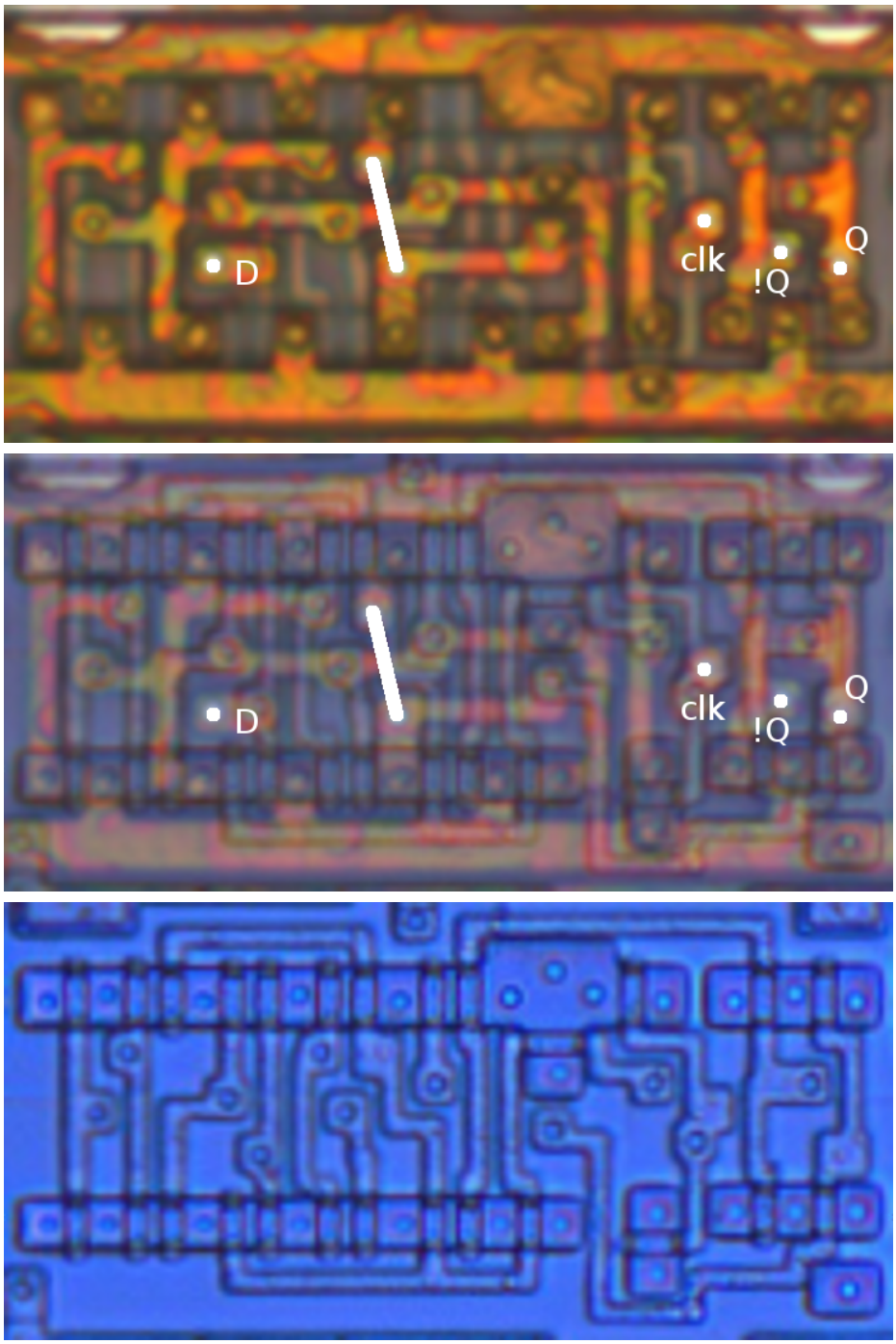

Abbildung B.5: Layout eines DQ-Flipflops

Schnittstellen

Schnittstelle	Richtung	Beschreibung
D	in	D
clk	in	clk
Q	out	Q
!Q	out	!Q

Funktion

Listing B.8: dg_ff2.v

```
/**
 * This is a Verilog implementation for a gate of type FF2.
 */

module dg_FF2 (clk, d, notq, q);

  // input ports
  input clk;
  input d;

  // output ports
  output notq;
  output q;

  reg q;
  reg notq;

  always @(posedge clk)
    begin
      q <= d;
      notq <= ~q;
    end

endmodule
```

B.3.3 Standardzelle: 2AOI

Layout

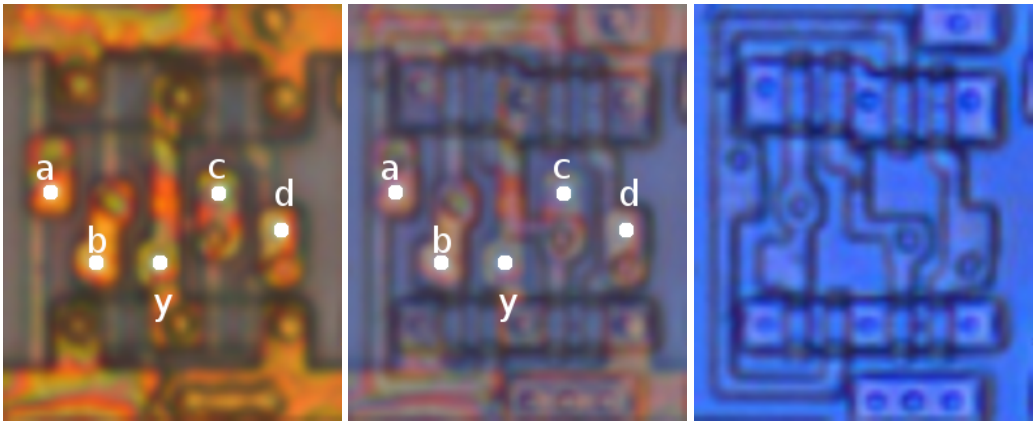

Abbildung B.6: Layout der Schaltung

Schnittstellen

Schnittstelle	Richtung	Beschreibung
a	in	a
c	in	c
b	in	b
d	in	d
y	out	y

Funktion

Listing B.9: dg_2aoi.v

```
/**
 * This is a Verilog implementation for a gate of type 2AOI.
 */
module dg_2AOI (a, b, c, d, y);
  input a, b, c, d;
  output y;

`ifdef IMPL_WITH_TRANS
   wire   w1, w2, w3, w4;
   supply1 vdd; // predefined high potential
   supply0 gnd; // predefined potential for ground

   pmos p1(w1, vdd, b);   // (drain, source, gate)
   pmos p2(y, w1, c);
   pmos p3(y, w2, a);
   pmos p4(w2, vdd, d);
   nmos n1(w3, gnd, b);   // (drain, source, gate)
   nmos n2(y, w3, a);
   nmos n3(y, w4, c);
   nmos n4(w4, gnd, d);
`else // !`ifdef IMPL_WITH_TRANS
   reg     y;
   always @*
     begin
        casex({a, b, c, d})
          4'b0000 : y = 1'b1;
          4'b0001 : y = 1'b1;
          4'b000x : y = 1'b1;
          4'b0010 : y = 1'b1;
          4'b0011 : y = 1'b0;
          4'b0100 : y = 1'b1;
          4'b0101 : y = 1'bz;
          4'b0110 : y = 1'b1;
          4'b0111 : y = 1'b0;
          4'b1000 : y = 1'b1;
          4'b1001 : y = 1'b1;
          4'b100x : y = 1'b1;
          4'b1010 : y = 1'bz;
          4'b1011 : y = 1'b0;
          4'b1100 : y = 1'b0;
          4'b110x : y = 1'b0;
          4'b1110 : y = 1'b0;
          4'b1111 : y = 1'b0;
          4'b111x : y = 1'b0;
          default: y = 1'b0;
        endcase
     end
`endif // !`ifdef IMPL_WITH_TRANS
endmodule
```

B.3.4 Standardzelle: XNOR

Layout

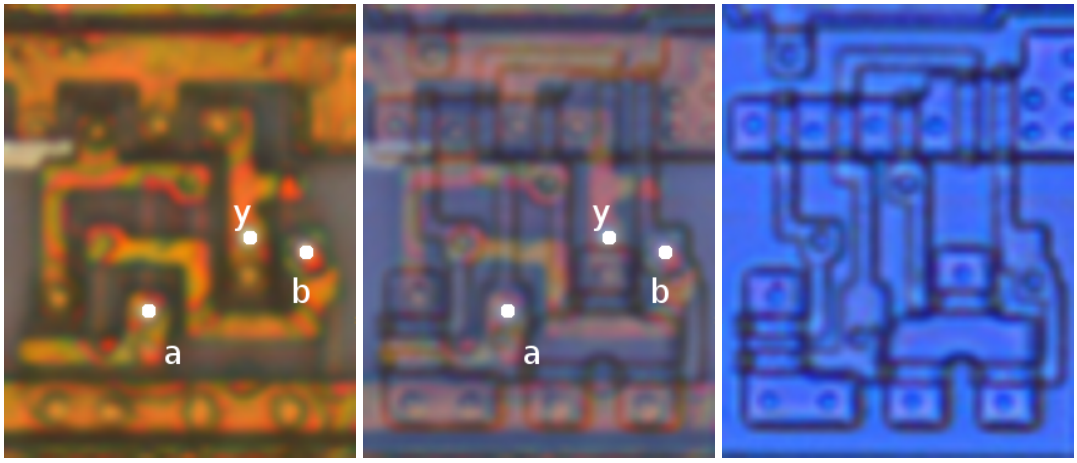

Abbildung B.7: Layout eines XNOR-Standardzelle

Schnittstellen

Schnittstelle	Richtung	Beschreibung
b	in	b
y	out	y
a	in	a

Funktion

Listing B.10: dg_xnor.v

```
/**
 * This is a Verilog implementation for a gate of type XNOR.
 */

module dg_XNOR (a, b, y);
    input a, b;
    output y;
    assign y = ~(a ^ b);
endmodule
```

B.3.5 Standardzelle: 2NAND

Layout

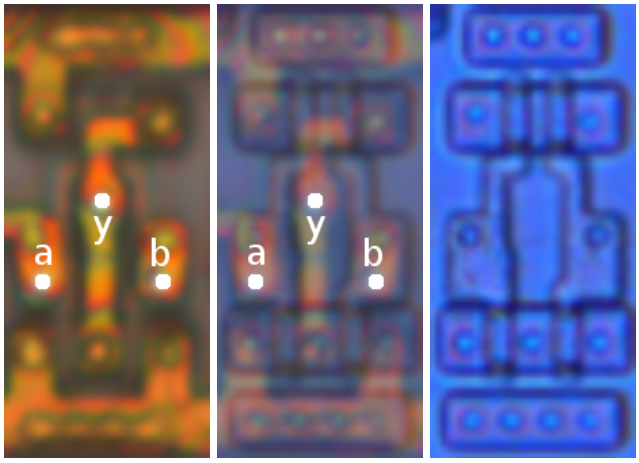

Abbildung B.8: Layout eines NANDs

Schnittstellen

Schnittstelle	Richtung	Beschreibung
b	in	b
y	out	y
a	in	a

Funktion

Listing B.11: dg_2nand.v

```
/**
 * This is a Verilog implementation for a gate of type 2NAND.
 */

module dg_2NAND (a, b, y);
    input a, b;
    output y;
    assign y = ~(a & b);
endmodule
```

B.3.6 Standardzelle: 2NOR

Layout

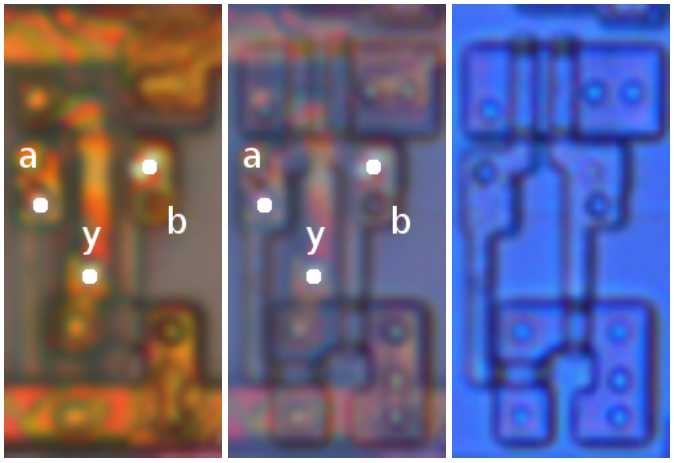

Abbildung B.9: Layout eines NOR-Gatters

Schnittstellen

Schnittstelle	Richtung	Beschreibung
a	in	a
b	in	b
y	out	y

Funktion

Listing B.12: dg_2nor.v

```
1 /**
   * This is a Verilog implementation for a gate of type 2NOR.
3  */

5 module dg_2NOR (a, b, y);
    input a, b;
7   output y;
    assign y = ~(a | b);
9 endmodule
```

B.3.7 Standardzelle: TRI1

Layout

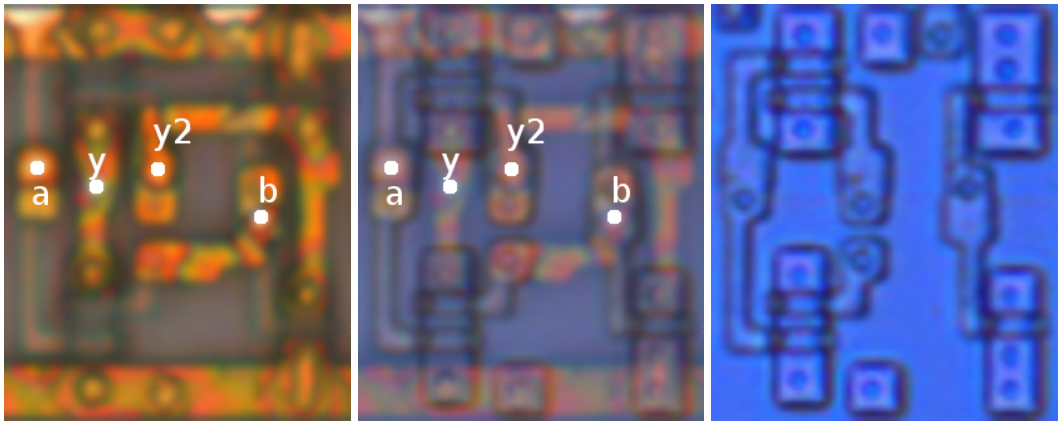

Abbildung B.10: Layout eines Tristate-Inverters

Schnittstellen

Schnittstelle	Richtung	Beschreibung
a	in	a
b	in	b
y	out	y
y2	out	y2

Funktion

Listing B.13: dg_tri1.v

```
/**
 * This is a Verilog implementation for a gate of type TRI1.
 */

module dg_TRI1 (
  a, b,
  y, y2
);

  // input ports
  input a;
  input b;

  // output ports
  output y;
  output y2;

  reg y;

  always @*
    begin
      y = 1'b0; // default
      case({a, b})
        2'b00 : y = 1'bz;
        2'b01 : y = ~a;
        2'b10 : y = 1'bz;
        2'b11 : y = ~a;
      endcase
    end


  assign y2 = !b;

endmodule
```

B.3.8 Standardzelle: TRI2

Layout

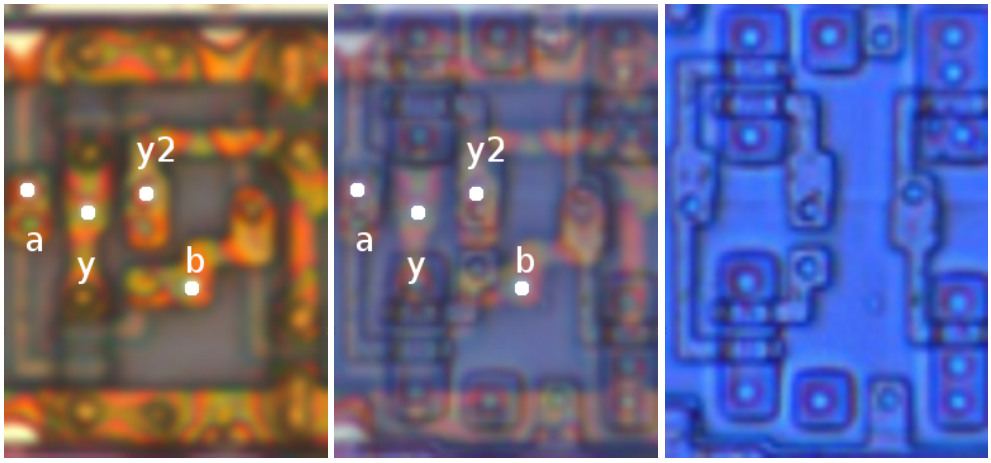

Abbildung B.11: Layout eines Tristate-Inverters

Schnittstellen

Schnittstelle	Richtung	Beschreibung
a	in	a
b	in	b
y	out	y
y2	out	y2

Funktion

Listing B.14: dg_tri2.v

```
/**
 * This is a Verilog implementation for a gate of type TRI2.
 */


module dg_TRI2 (a, b, y, y2);

  // input ports
  input a;
  input b;

  // output ports
  output y;
  output y2;

  reg y;

  always @*
    begin
      y = 1'bz; // default
      case({a, b})
        2'b00 : y = 1'bz;
        2'b01 : y = ~a;
        2'b10 : y = 1'bz;
        2'b11 : y = ~a;
      endcase
    end


  assign y2 = ~b;


endmodule
```

B.3.9 Standardzelle: TRI3

Layout

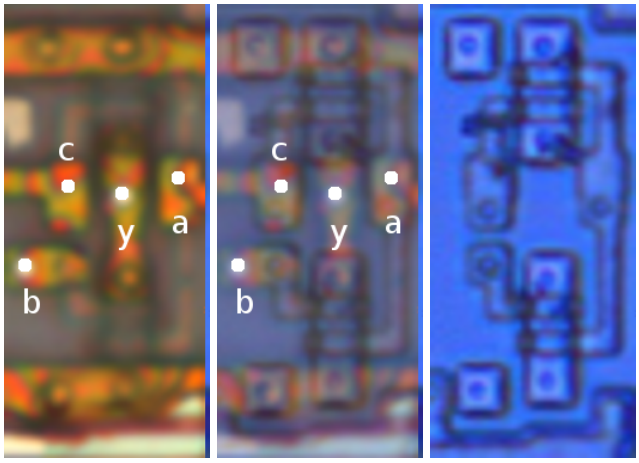

Abbildung B.12: Layout eines Tristate-Inverters

Schnittstellen

Schnittstelle	Richtung	Beschreibung
c	in	c
y	out	y
b	in	b
a	in	a

Funktion

Listing B.15: dg_tri3.v

```
/**
 * This is a Verilog implementation for a gate of type TRI3.
 */

module dg_TRI3 (a, b, c, y);

  // input ports
  input a;
  input b;
  input c;

  // output ports
  output y;

  reg y;

  always @*
    begin
      y = 1'bz; // default
      case({a, b, c})
        3'b000 : y = 1'b1;
        3'b001 : y = ~a; //
        3'b010 : y = 1'bz;
        3'b011 : y = 1'bz;
        3'b100 : y = 1'bz;
        3'b101 : y = ~a; // 0
        3'b110 : y = 1'bz;
        3'b111 : y = 1'b0;
      endcase
    end

endmodule
```

B.3.10 Standardzelle: INV1

Layout

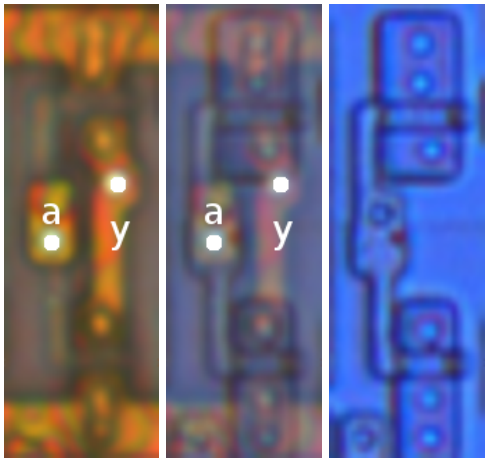

Abbildung B.13: Layout eines Inverters

Schnittstellen

Schnittstelle	Richtung	Beschreibung
y	out	y
a	in	a

Funktion

Listing B.16: dg_inv1.v

```
/**
 * This is a Verilog implementation for a gate of type INV1.
 */

module dg_INV1 (a, y);
  input a;
  output y;
  assign y = ~a;
endmodule
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[Ale01] Andrei Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Professional, 2001. ISBN: 0201704315.

[Bal04] John Baliga. »Chips go vertical«. In: *IEEE Spectrum* 41.3 (2004), S. 43–47. ISSN: 0018-9235. DOI: 10.1109/MSPEC.2004.1270547.

[Bec98] Friedrich Beck. *Präparationstechniken für die Fehleranalyse an integrierten Halbleiterschaltungen*. VCH Verlagsgesellschaft mbH, 1998. ISBN: 3-527-26879-9.

[BH01] K. Briechle und Uwe D. Hanebeck. *Template Matching Using Fast Normalized Cross Correlation*. 2001. DOI: 10.1117/12.421129. URL: http://isas.uka.de/downloads/briechle_spie2001.pdf (besucht am 23. 02. 2011).

[Bly+93] Simon Blythe u. a. »Layout reconstruction of complex silicon chips«. In: *IEEE Journal of Solid-State Circuits* 28 (2 1993), S. 138 –145. ISSN: 0018-9200. DOI: 10.1109/4.192045. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=192045&isnumber=4948 (besucht am 09. 04. 2011).

[Bou+02] N. G. Bourbakis u. a. »A knowledge-based expert system for automatic visual VLSI reverse-engineering: VLSI layout version«. In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*. 2002, S. 428–436. DOI: 10.1109/TSMCA.2002.805765. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=1046073&isnumber=22416 (besucht am 23. 05. 2010).

[BR91] Nikolaos G. Bourbakis und C.V. Ramamoorthy. »Specifications for the development of an expert tool for the automatic optical understanding of

electronic circuits: VLSI Reverse Engineering«. In: *VLSI Test Symposium, 1991. 'Chip-to-System Test Concerns for the 90's'*. 1991, S. 98 –103. DOI: 10.1109/VTEST.1991.208140. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=208140 (besucht am 23. 05. 2010).

[Erh08] Angelika Erhardt. *Einführung in die Digitale Bildverarbeitung. Grundlagen, Systeme und Anwendungen*. Teubner B.G. GmbH, 2008. ISBN: 9783519004783.

[Fin02] Klaus Finkenzeller. *RFID-Handbuch. Grundlagen und praktische Anwendungen induktiver Funkanlagen, Transponder und kontaktloser Chipkarten*. München: Carl Hanser Verlag, 2002. ISBN: 3-446-22071-2.

[KK99] Oliver Kömmerling und Markus G. Kuhn. »Design principles for tamper-resistant smartcard processors«. In: *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*. Chicago, Illinois: USENIX Association, 1999. URL: http://portal.acm.org/citation.cfm?id=1267115.1267117 (besucht am 02. 03. 2011).

[KNP08] Jan Krißler, Karsten Nohl und Henryk Plötz. »Chiptease«. In: *c't* (8 2008), S. 80–85. ISSN: 0724-8679.

[Kum00] Jean Kumagai. »Chip Detectives«. In: *IEEE Spectrum* 37 (11 2000), S. 43–49. ISSN: 0018-9235. DOI: 10.1109/6.880953. URL: http://ece.iisc.ernet.in/~navakant/vlsi/2008/chip_detective.pdf (besucht am 02. 03. 2011).

[LA97] Dimitri Lagunovsky und Sergey Ablameyko. »Fast line and rectangle detection by clustering and grouping«. In: *Computer Analysis of Images and Patterns* (1997), S. 503–510. DOI: 10.1007/3-540-63460-6_156. URL: http://www.springerlink.com/content/k03727633k053u68/ (besucht am 02. 03. 2011).

[LA99] Dimitri Lagunovsky und Sergey Ablameyko. »Straight-line-based primitive extraction in grey-scale object recognition«. In: *Pattern Recognition Letters* 20.10 (1999), S. 1005–1014. DOI: 10.1016/S0167-8655(99)00067-7. URL: http://linkinghub.elsevier.com/retrieve/pii/S0167865599000677 (besucht am 02. 03. 2011).

[LAK96] Dimitri Lagunovsky, Sergey Ablameyko und M. Kutas. »Extraction of topological features of integrated circuit from grey-scale image«. In: *IAPR Workshop on Machine Vision Applications* (1996), S. 1–4. URL: http://www.mva-org.jp/Proceedings/CommemorativeDVD/1996/papers/1996271.pdf (besucht am 02. 03. 2011).

[LAK98] Dimitri Lagunovsky, Sergey Ablameyko und M. Kutas. »Recognition of Integrated Circuit Images in Reverse Engineering«. In: *International Conference on Pattern Recognition* 2 (1998), S. 1640. ISSN: 1051-4651. DOI: 10.1109/ICPR.1998.712032. URL: http://www.computer.org/portal/web/csdl/doi/10.1109/ICPR.1998.712032 (besucht am 02. 03. 2011).

[Lee03] Weng Fook Lee. *Verilog Coding for Logic Synthesis*. John Wiley & Sons, Inc., 2003. ISBN: 978-0-471-42976-0.

[Mal87] W. Maly. *Atlas of Ic Technologies. An Introduction to Vlsi Processes*. Benjamin-Cummings Pub Co, 1987. ISBN: 0805368507.

[Mas10] Giedrius Masalskis. »Development and Analysis of Integrated Circuit Topology Element Recognition System. Summary of Doctoral Dissertation«. 2010. URL: http://vddb.laba.lt/fedora/get/LT-eLABa-0001:E.02~2011~D_20110125_093925-55797/DS.005.1.01.ETD (besucht am 02. 04. 2011).

[MN08] G. Masalskis und R. Navickas. »Reverse Engineering of CMOS Integrated Circuits«. In: *Electronics and Electrical Engineering* (Okt. 2008), S. 25–28. ISSN: 1392-1215. URL: http://www.ee.ktu.lt/journal/2008/8/05_ISSN_1392-1215_Reverse%20Engineering%20of%20CMOS%20Integrated%20Circuits.pdf (besucht am 02. 03. 2011).

[MN10] G. Masalskis und R. Navickas. »Time-Efficient Adaptive Segmentation Algorithm for IC Layers Images«. In: *Electronics and Electrical Engineering* 106 (10 2010), S. 133–138. ISSN: 1392-1215. URL: http://www.ktu.lt/lt/mokslas/zurnalai/elektros_z/z106/30__ISSN_1392-1215_Time-Efficient%20Adaptive%20Segmentation%20Algorithm%20forIC%20Layers%20Images.pdf (besucht am 02. 03. 2011).

[MOP07] Stefan Mangard, Elisabeth Oswald und Thomas Popp. *Power Analysis Attacks. Revealing the Secrets of Smart Cards*. Berlin: Springer, 2007. ISBN: 9780387308579.

[Noh+08] Karsten Nohl u. a. »Reverse-Engineering a Cryptogrphic RFID Tag«. In: *Proceedings of the 17th conference on Security symposium*. San Jose, CA: USENIX Association, 2008, S. 185–193. URL: http://portal.acm.org/citation.cfm?id=1496711.1496724 (besucht am 23. 02. 2011).

[NP07] Karsten Nohl und Henryk Plötz. *26th Chaos Communication Congress: Legic Prime: Obscurity in Depth*. Dez. 2007. URL: http://events.ccc.de/congress/2009/Fahrplan/events/3709.en.html (besucht am 21. 11. 2010).

[NP11] Karsten Nohl und Henryk Plötz. *Peeling Away Layers of an RFID Security System. Financial Cryptography and Data Security '11, February 28 - March 4, 2011, Saint Lucia*. März 2011. URL: http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2011-03/SAR-PR-2011-03_.pdf (besucht am 02. 02. 2011).

[RE08] Wolfgan Rankl und Wolfgang Effing. *Handbuch der Chipkarten. Aufbau - Funktionsweise - Einsatz vin Smart Cards*. 5., überarbeitete und erweiterte Auflage. München: Carl Hanser Verlag, 2008. ISBN: 978-3446404021.

[Sai05] Fumihiko Saitoh. »Image template matching based on edge-spin correlation«. In: *Electrical Engineering in Japan* 153 (2 2005). ISSN: 1520-6416. DOI: 10.1002/eej.20184. URL: http://dx.doi.org/10.1002/eej.20184 (besucht am 03. 09. 2010).

[Sch10] Martin Schobert. *Reverse-Engineering von Logik-Gattern in Integrierten Schaltkreisen*. März 2010. URL: http://www.degate.org/documentation/reverse_engineering_logic.pdf (besucht am 23. 05. 2010).

[Sko05] Sergei P. Skorobogatov. *Semi-invasive attacks. A new approach to hardware security analysis*. Techn. Ber. UCAM-CL-TR-630. University of Cambridge, Computer Laboratory, 2005. URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf (besucht am 03. 03. 2010).

[SV02] Raouf Kh. Sadykhov und Maksim E. Vatkin. »An Application of "Neocognitron" Neural Network for Integral Chip Images Processing«. In: *International Scientific Journal of Computing* 1 (2 2002). URL: http://neuroface.narod.ru/files/Neo_2_4_uk.pdf (besucht am 11.05.2011).

[Tan99] Tanner Consulting & Engineering Services. *MTSMS035DL Digital Low Power Standard Cell Library For TSMC 0.35µ Sub-micron Process. Revision A*. 1999.

[Tec08] TechInsights. *TechInsights Announces and Strengthens Professional Services Team with Acquisition of Sanguine Microelectronics*. Dez. 2008. URL: http://www.semiconductor.com/resources/press_releases/TechPR_Sanguine_FINAL.pdf (besucht am 23.05.2010).

[TJ09] Randy Torrance und Dick James. »The State-of-the-Art in IC Reverse Engineering«. In: *Lecture Notes in Computer Science: Cryptographic Hardware and Embedded Systems - CHES 2009*. Bd. 5747/2009. 2009, S. 363–381. DOI: 10.1007/978-3-642-04138-9_26. URL: http://www.springerlink.com/content/w8568j6533634w74/ (besucht am 23.05.2010).

[Tor09] Randy Torrance. *The state-of-the-art in Semiconductor Reverse Engineering at Chipworks*. Workshop on Cryptographic Hardware and Embedded Systems 2009 (CHES 2009). Sep. 2009. URL: http://www.iacr.org/workshops/ches/ches2009/presentations/12_Invited_Talk_III/CHES2009_torrance.pdf (besucht am 23.05.2010).

[WH05] Neil H. E. Weste und David Harris. *CMOS VLSI Design. A Circuits and Systems Perspective / International Edition*. 3. Aufl. Pearson Education, Inc., 2005. ISBN: 0-321-26977-2.

[Yao+05] H.P. Yao u. a. »Circuitry analyses by using high quality image acquisition and multi-layer image merge technique«. In: *Physical and Failure Analysis of Integrated Circuits, 2005. IPFA 2005. Proceedings of the 12th International Symposium*. 2005, S. 294 –297. ISBN: 0-7803-9301-5. DOI: 10.1109/IPFA.2005.1469182. URL: http://www.ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1469182 (besucht am 23.05.2010).

Listings