Design Entropy

A Measure for Complexity

Vom Promotionsausschuss der
Technischen Universität Hamburg-Harburg
zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation

von
Dipl.-Phys. oec., LL.B. Benjamin Menhorn

aus
Bietigheim-Bissingen

2017

Gutachter:	Prof. Dr. Heiko Falk
	Prof. Dr.-Ing. Wolfgang Krautschneider
Tag der mündlichen Prüfung:	11. Mai 2017

Kurzzusammenfassung

Die vorliegende Dissertation stellt eine neue Methode zur Berechnung der Komplexität und der Entwurfsgröße von digitalen Hardwareprojekten und integrierten Schaltungen vor. Sie basiert auf dem Ansatz, dass ein System aus Komponenten besteht, welche miteinander Nachrichten austauschen. Abgeleitet von der Informationsentropie wird das Maß der Entwurfsentropie definiert und angewendet.

Abstract

This PhD-Thesis presents a new method for calculating complexity and design size of digital hardware projects and integrated circuits. It is based on the assumption that systems consist of components interchanging messages with each other. Derived from information entropy, the measure of Design Entropy is defined.

Inhaltsverzeichnis

1 Einleitung

1941, vor 75 Jahren, hatte Konrad Zuse in Berlin den Z3 fertiggestellt, den ersten funktionsfähigen und turingvollständigen Digitalrechner der Welt [190, S. 53] [250, S. 55]. Er bestand aus 2.000 elektromagnetischen Relais [250, S. 55]. Als erster programmierbarer und rein elektronischer Universalrechner gilt der 1946 von John Eckert und John Mauchly entwickelte Electronic Numerical Integrator and Computer (ENIAC), in welchem etwa 18.000 Vakuumröhren eingesetzt wurden [64, S. 97]. Bereits 1925 beschrieb und patentierte Julius Lilienfeld einen Aufbau mit den Eigenschaften einer Elektronenröhre in einem Festkörper [127] [128]. Der erste Halbleiter-Feldeffekttransistor wurde 1934 von Oskar Heil konstruiert und patentiert [76].

Mit der Weiterentwicklung des Transistors wurden die ersten Computer auf Transistorbasis vorgestellt. 1953 wurde der Manchester University Transistor Computer präsentiert [124, S. 37], 1954 der Transistorized Airborne Digital Computer (TRADIC) [105, S. 34], 1955 der Harwell Transistor Electronic Digital Automatic Computer (Harwell CADET) [44, S. 104] und 1956 der Transistorized Experimental Computer Zero (TX-0) [135, S. 1]. Mit der Verwendung von Transistoren statt Röhren und der zunehmenden Entwicklung in Richtung integrierter Schaltungen begann sich die digitale Hardware durchzusetzen.

Auch heute, über 60 Jahre später, bilden Transistoren den Kern digitaler Hardware. Ständige Entwicklungen und Miniaturisierungen erlauben es, auf einem Halbleiterchip mehrere Milliarden Transistoren unterzubringen [158, S. 50]. Dies stellt eine hohe Herausforderung an die Entwicklung digitaler Hardware dar. Zu deren Bewältigung werden immer wieder neue Methoden und Werkzeuge vorgestellt, welche die Komplexität bei der Entwicklung senken sollen. Dabei versuchen die Entwicklerinnen, Entwickler, Hersteller und Herstellerinnen, die Projekte möglichst effizient und effektiv zu gestalten.

Allgemein ist es für das Projektmanagement und insbesondere für das Projektmanagement im Bereich der Informatik wichtig, über eine adäquate Bestimmung der Quantität (Umfang) und der Komplexität eines Entwurfs zu verfügen. In den Anfängen der digitalen Schaltungen war es noch ausreichend, Transistoren zu zählen oder das Gewicht zu bestimmen. Mit dem steigenden Umfang und der steigenden Komplexität heutiger Entwicklungen sind diese Methoden jedoch unzureichend und ungenau.

Aus diesen Gründen wird innerhalb der vorliegenden Dissertation eine neue Methode zur Berechnung der Entwurfsgröße und Komplexität von digitaler Hardware vorgestellt. Diese beruht auf abstrakten Zuständen und geht auf die Theorie des Informationsaustauschs der Nachrichtentechnik zurück. Der hier vorgestellten Methode liegt das Verständnis zugrunde, dass Komponenten eines Systems miteinander Nachrichten austauschen. Um diesen Nachrichtenaustausch zu ermöglichen, müssen die Komponenten mithilfe von Signalen miteinander kommunizieren können, insbesondere über elektrische Leitungen. Somit ist es beim Entwurf digitaler Systeme erforderlich, neben den Komponenten selbst, auch die Kommunikation zwischen diesen zu entwerfen. Hiernach können die Entwurfsgröße und die Komplexität über die realisierten Verbindungen und den Nachrichtenaustausch zwischen den Komponenten berechnet werden. Dies stellt das Maß der Entwurfsentropie dar, welches in der vorliegenden Arbeit vorgestellt wird.

Hierzu ist die Dissertation wie folgt aufgebaut: Im nachfolgenden Kapitel 2 wird die Arbeit anhand der Bedeutung der Komplexitäts- und Quantitätsbestimmung für den Entwurfsprozess und das Projektmanagement motiviert. Darauf basierend werden die Ziele der Arbeit festgelegt. Kapitel 3 stellt die in Zusammenhang stehenden Arbeiten vor und bildet gleichzeitig den Stand der Forschung ab. Kern der Arbeit bildet Kapitel 4, in welchem das Modell der Entwurfsentropie beschrieben wird, die Formeln zur deren Berechnung hergeleitet werden und die Anwendung auf Hardwareentwürfe gezeigt wird. Zudem wird in diesem Kapitel das im Rahmen dieser Arbeit entwickelte Analysewerkzeug für Hardwareentwürfe vorgestellt sowie als Exkurse die Möglichkeit der Zustandsreduktion beschrieben und die grundsätzliche Anwendbarkeit auf Software gezeigt. In Kapitel 5 wird die Entwurfsentropie anschaulich für kompakte Hardwareentwürfe berechnet. Kapitel 6 wendet die Entwurfsentropie im Rahmen mehrerer Fallstudien an. Die Arbeit endet in Kapitel 7 mit der Zusammenfassung und dem Ausblick.

2 Motivation und Zielsetzung

Regelmäßig werden in der Literatur Aussagen über die steigende Transistoranzahl und die steigende Komplexität im Bereich der digitalen Hardware beziehungsweise der integrierten Schaltungen (Integrated Circuit – IC) getroffen. Dabei haben die Entwurfsgröße und die Komplexität eines Projekts insbesondere auf den Entwurfsprozess und das Projektmanagement entscheidende Auswirkungen. Das Bedürfnis einer adäquaten Bestimmung der beiden Größen bildet die Motivation der vorliegenden Arbeit, welche im Folgenden dargestellt wird. Hierauf aufbauend werden am Ende des Kapitels die Ziele der Arbeit festgelegt.

2.1 Komplexität

Innerhalb der Literatur besteht insoweit Einigkeit, als dass die Komplexität digitaler Hardware beziehungsweise integrierter Schaltungen immer weiter ansteigt [20, S. 357] [23, S. 87] [24, S. 1881] [33, S. 11] [34, S. 9] [58, S. xi, 1] [63, S. 413] [75, S. 1] [109, S. 172] [110, S. 24 f.] [117, S. 5] [122, S. 115] [153, S. 63] [186, S. 1] [211, S. 1] [237, S. 15]. Als Maß hierfür wird vielfach die steigende Anzahl an Transistoren oder Gattern pro Chip zugrunde gelegt und damit das Mooresche Gesetz, das in Abbildung 2.1 dargestellt ist, als Maß für die Steigerung der Komplexität herangezogen [23, S. 87] [24, S. 1881] [63, S. 413] [75, S. 1] [110, S. 24] [153, S. 63] [158, S. 52] [211, S. 1] [227, S. 36] [228, S. 8] [229, S. 8]. Für die Grafik wurde die Anzahl an Transistoren von Intel-Prozessoren mit den Jahreszahlen ihrer Einführung als Datenpunkte aufgenommen, um so das Mooresche Gesetz als Regressionsgerade zu erhalten.

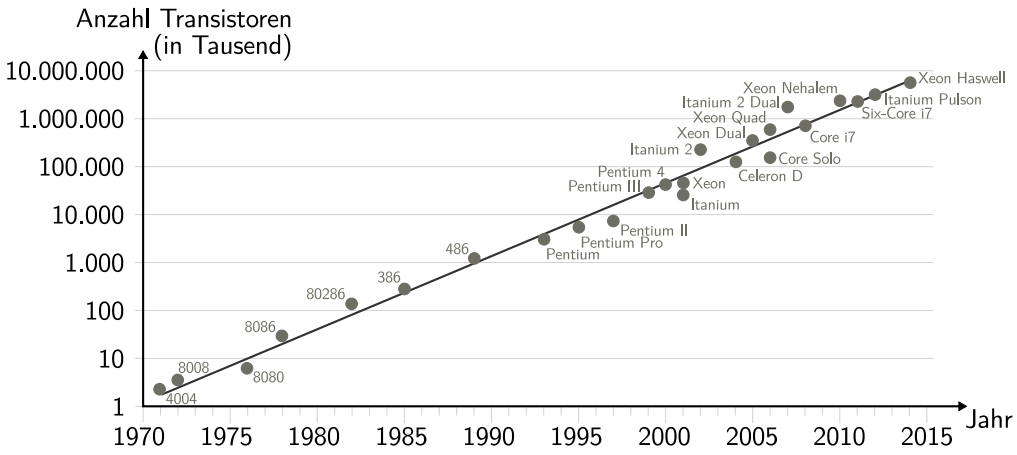


Abbildung 2.1: Mooresches Gesetz (Daten aus [8], [47], [97], [188] und [204])

Ursprünglich hatte Gordon Moore eine jährliche Verdopplung der Transistoranzahl vorhergesagt [154, S. 115], welche er 1975 in einer Rede auf zwei Jahre korrigierte [155, S. 13]. Abhängig von der Quelle wird heutzutage eine Verdopplung der Anzahl an Transistoren alle 18 bis 24 Monate beobachtet und vorhergesagt [96, S. 1] [110, S. 23] [116]. Obwohl es sich beim Mooreschen Gesetz um kein Naturgesetz handelt, zeigen die Daten und die Anerkennung innerhalb der Literatur, dass mit dieser „Gesetzmäßigkeit“ die Entwicklung der Anzahl an Transistoren/Gattern eines Halbleiters zutreffend abgebildet und vorhergesagt werden kann. Gleichzeitig stellt das Mooresche Gesetz eine selbsterfüllende Prophezeiung dar, da versucht wird, die Entwicklung an dieser „Gesetzmäßigkeit“ auszurichten [117, S. 5] [226].

Dabei stellt sich die Frage, ob die Komplexität einer Schaltung allein durch die Anzahl an Transistoren oder Gattern bestimmt wird. Bei einer genaueren Betrachtung zeigt das Mooresche Gesetz grundsätzlich nur, dass es mit der Zeit möglich ist, immer mehr Transistoren/Gatter auf der gleichen Fläche unterzubringen. Somit steigt grundsätzlich der Projektumfang, gemessen an der Anzahl an Transistoren/ Gattern. Jedoch folgt daraus nicht gleichzeitig, dass die Entwurfskomplexität steigt, wobei eine gewisse Korrelation sicher nicht von der Hand zu weisen ist. Dies führt zur Kernfragestellung der vorliegenden Arbeit: Wie kann die Komplexität digitaler Hardwareprojekte unter gleichzeitiger Berücksichtigung der Entwurfsgröße gemessen werden? Diese zwei Parameter (Entwurfsgröße und Komplexität) spielen sowohl für den Entwurfsprozess als auch für das Projektmanagement eine entscheidende Rolle, wie im Folgenden dargestellt wird.

2.2 Entwurfsprozess

Der Entwurfsprozess ist die Abfolge von Schritten während des Entwurfs integrierter/digitaler Schaltungen, ausgehend von der Spezifikation bis zur Fertigungsfreigabe [20, S. 357] [56, S. 11]. Dieser Prozess folgt grundsätzlich einer strukturierten Vorgehensweise [198, S. 33], wobei er vom jeweiligen zu entwickelnden Produkt und Unternehmen abhängig ist [56, S. 11]. Regelmäßig kommen rechnerunterstützte Konstruktionswerkzeuge (Computer-Aided Design – CAD) [56, S. 15] und Entwurfsautomatisierungswerkzeuge (Electronic Design Automation – EDA) [110, S. 28] zum Einsatz, welche den Entwurf auf einer höheren Abstraktionsebene zulassen und Schritte des Entwurfs automatisieren [58, S. 2] [117, S. 11]. Zusätzlich bieten viele Entwurfswerkzeuge und Datenbanken wiederverwendbare Schaltungen an, welche in neue Schaltungen integriert werden können [56, S. 12 f.] [109, S. 172 f.] [218, S. 141 f.]. Dies ermöglicht den Entwurf auf einem sehr hohen Abstraktionsniveau, insbesondere auf der Systemebene [117, S. 11].

Grundsätzlich beginnt jeder Entwurf mit einer (Produkt-)Spezifikation, welche die Funktionalität, Schnittstellen und weitere Details festlegt [56, S. 11] [117, S. 11] [132, S. 11] [198, S. 31]. Der eigentliche Schaltungsentwurf kann dabei in drei Domänen repräsentiert werden: als Verhaltensbeschreibung, als Strukturbeschreibung oder als geometrische Beschreibung [58, S. 3]. Diese drei Domänen werden anschaulich als Achsen im Y-Diagramm (Gajski-Diagramm) in Abbildung 2.2 dargestellt und im Folgenden kurz beschrieben.

1. Verhaltensbeschreibung:
   Die Verhaltensbeschreibung oder auch funktionale Beschreibung gibt das Verhalten eines Entwurfs an [198, S. 31]. Ähnlich einer Blackboxsicht, wird es als Funktion der Eingaben und gegebenenfalls des Zeitverhaltens beschrieben [56, S. 2]. Dadurch wird ausschließlich das Verhalten in Abhängigkeit der Eingaben beschrieben, nicht aber die jeweilige Implementierung und/ oder Umsetzung mithilfe einzelner Bausteine [56, S. 2].

2. Strukturbeschreibung:
   Im Gegensatz zur Verhaltensbeschreibung definiert die Strukturbeschreibung, wie ein Entwurf mit den jeweiligen Komponenten aufgebaut ist und wie diese miteinander verbunden sind [20, S. 358] [56, S. 2]. Dadurch wird die jeweilige Implementierung spezifiziert, ohne dabei die Funktionalität zu bestimmen [56, S. 2] [198, S. 34].

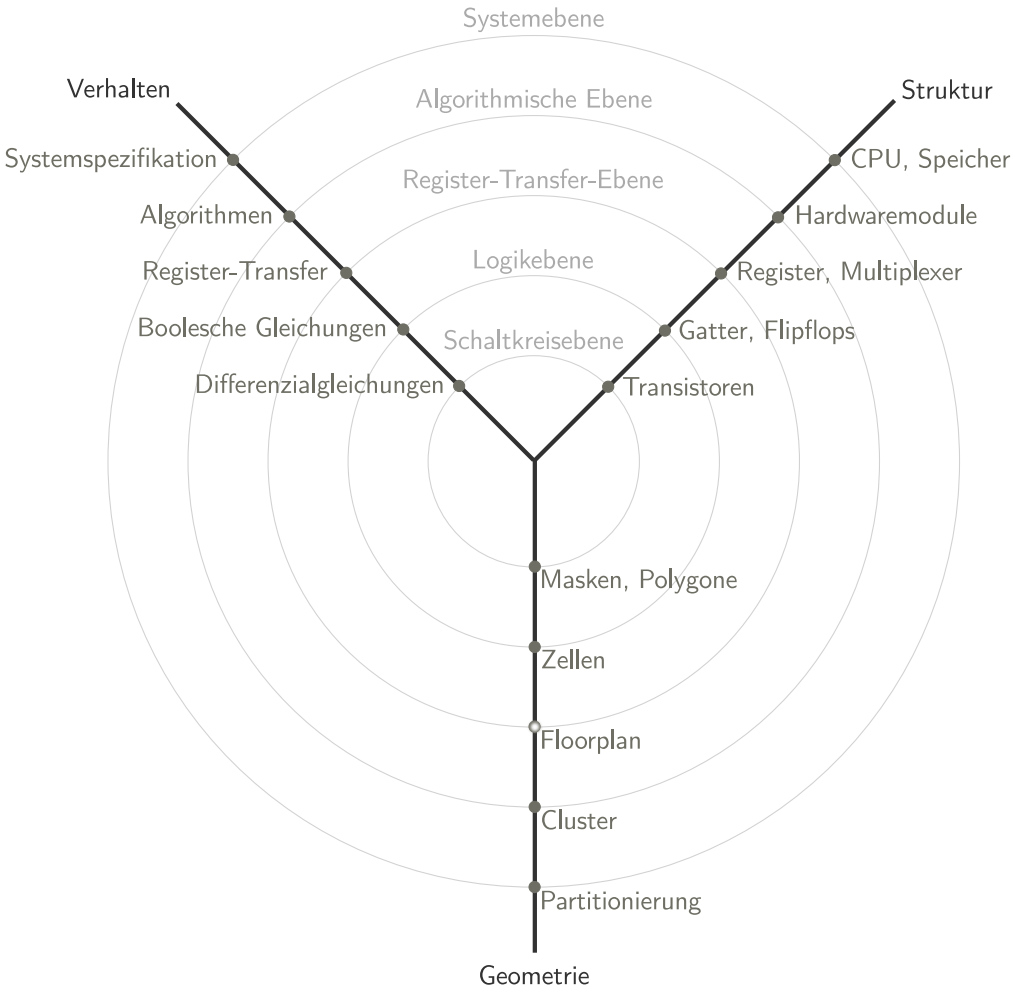


Abbildung 2.2: Y-Diagramm (Gajski-Diagramm) (nach [57, S. 13] [58, S. 4])

3. Geometrische Beschreibung:
   Die geometrische oder auch physikalische Beschreibung zeigt den (physikalischen) Aufbau eines Entwurfs mit Abmessungen, Lagen und tatsächlichen Verbindungen der Komponenten [56, S. 2]. Regelmäßig zeigt die Geometrie einen Entwurf, wie er schlussendlich gefertigt wird [56, S. 2 f.].

Entlang der Achsen des Y-Diagramms aus Abbildung 2.2 werden die einzelnen Abstraktionsebenen dargestellt, wobei der Grad der Abstraktion nach außen zunimmt [20, S. 359] [58, S. 3]. Die Kreise verdeutlichen, dass eine Abstraktion in allen drei Domänen erfolgen kann. Dabei verwendet jede Domäne unterschiedliche Objekte auf den einzelnen Abstraktionsebenen. Eine Auswahl der Objekte, welche die jeweilige Abstraktionsebene in der jeweiligen Domäne kennzeichnen, ist an den Schnittpunkten der Achsen mit den Kreisen gegeben. Somit definieren die beim

Entwurf verwendeten Objekte die einzelnen Abstraktionsebenen [56, S. 9]. Abbildung 2.3 illustriert die einzelnen Entwurfsebenen mit den für die jeweilige Ebene typischen Objekten. Wie beim Y-Diagramm nimmt der Grad der Abstraktion von der Systemebene hin zur Schaltkreisebene ab. Die einzelnen Ebenen werden im Folgenden kurz beschrieben.

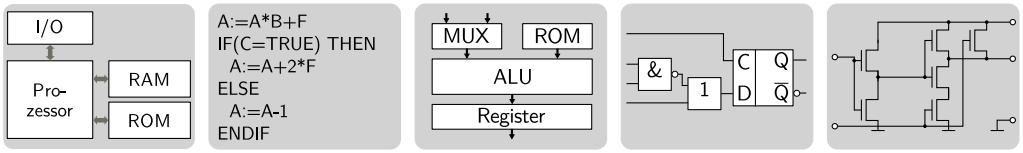


Abbildung 2.3: Entwurfsebenen [nach 20, S. 359]

1. Systemebene:
   Die Systemebene legt die grundlegenden Eigenschaften elektronischer Systeme fest. Innerhalb der Verhaltensbeschreibung werden Blockschaltbilder verwendet, wobei auf Signale und Zeitverhalten abstrahiert wird [58, S. 5]. Innerhalb der Strukturbeschreibung werden Komponenten wie Prozessoren oder Speicher verwendet [58, S. 5].

2. Algorithmische Ebene:
   Auf der algorithmischen Ebene werden grundlegende Entscheidungen über die Architektur eines Entwurfs festgelegt [198, S. 36]. Basiselemente der Strukturbeschreibung sind Hardwaremodule und bei der Verhaltensbeschreibung Algorithmen, wie addieren, subtrahieren oder Schleifen [198, S. 36].

3. Register-Transfer-Ebene:
   Auf der Register-Transfer-Ebene wird das Verhalten durch kommunizierende Register und Logikeinheiten beschrieben [58, S. 5]. Ein wesentlicher Bestandteil dieser Ebene sind Taktzyklen, um die einzelnen Operationen zu planen [132, S. 88] [198, S. 36].

4. Logikebene:
   Innerhalb der Logikebene wird das Verhalten durch Boolesche Gleichungen und Sequenzen beschrieben [58, S. 5]. Bei der Strukturbeschreibung kommen dagegen Gatter und Flipflops zum Einsatz [59, S. 4].

5. Schaltkreisebene:
   Als niedrigste Abstraktionsebene wird auf der Schaltkreisebene das Verhalten durch Transferfunktionen beschrieben, insbesondere durch Differenzialgleichungen [58, S. 5]. Die Strukturbeschreibung verwendet auf dieser Ebene Transistoren und deren Verbindungen [58, S. 5].

Während des Entwurfsprozesses können mehrere Abstraktionsebenen und Domänen zur Umsetzung verwendet werden [56, S. 11]. Ziel ist dabei, von der Spezifikation zu einem niedrigeren Abstraktionslevel innerhalb des Y-Diagramms zu kommen, sodass durch eine geometrische Beschreibung der Entwurf umgesetzt werden kann [198, S. 35]. Die häufigste im Entwurfsprozess verwendete Methodik ist die Verhaltensbeschreibung und eine Synthese der Strukturbeschreibung mithilfe von CAD-Werkzeugen [56, S. 11]. Für die Synthese werden Objekte aus einem niedrigeren Abstraktionslevel verwendet [56, S. 13]. Zur Beschreibung eines Entwurfs werden als CAD- beziehungsweise EDA-Werkzeuge im Bereich der digitalen Hardware insbesondere Hardwarebeschreibungssprachen verwendet, beispielsweise VHDL oder Verilog [20, S. 357] [56, S. 15 f.] [198, S. 35]. Zudem kommen auf höheren Abstraktionsebenen zum Beispiel SystemC sowie Modellierungen in Matlab/Simulink in Betracht [22, S. 15] [117, S. 12 f.] [122, S. 243] [132, S. 87] [186, S. 347 ff.].

Zusammenfassend kann der Entwurf sowohl in unterschiedlichen Domänen, auf verschiedenen Abstraktionsebenen als auch mit unterschiedlichen Entwurfswerkzeugen erfolgen. Hierdurch ist vielfach eine Vergleichbarkeit des Entwurfsaufwands nicht direkt möglich. Gemeinsam ist aber allen Entwurfsmethoden, dass das zu entwerfende System aus Objekten zusammengesetzt wird. Die zur Verfügung stehenden Objekte unterscheiden sich in den jeweiligen Domänen und auf den jeweiligen Ebenen. Sie werden aber jeweils so miteinander verbunden, dass sie miteinander kommunizieren können. Die vorliegende Arbeit baut auf diesem Entwurfsprinzip auf, sodass Objekte (Komponenten) miteinander Nachrichten austauschen. Durch die abstrakte Definition von Komponenten und deren Kommunikation kann die Methode der Entwurfsentropie auf verschiedenen Ebenen und in verschiedenen Domänen angewendet werden.

2.3 Bedeutung für die Wirtschaft

Der Entwurfsprozess bestimmt nicht nur den Erfolg eines Projekts im Allgemeinen, sondern auch den wirtschaftlichen Erfolg im Besonderen. Dabei ist die Halbleiterindustrie zu einem der wichtigsten Wirtschaftszweige herangewachsen, mit einem weltweiten Marktvolumen von 336 Milliarden US-Dollar im Jahr 2014 [200, S. 10] [230]. Im Durchschnitt wuchs der Markt zwischen 1994 und 2014 jährlich um 11,5 % [200, S. 2], wie Abbildung 2.4 zeigt. Trotz mehrerer kurzer Rückgänge, wie beispielsweise nach dem Jahr 2000, befindet sich der Markt weiter im Wachstum, sodass zu erwarten ist, dass auch in Zukunft die Frage nach dem wirtschaftlichen Erfolg von Halbleiterprojekten eine zentrale Stellung beim Entwurf einnehmen wird.

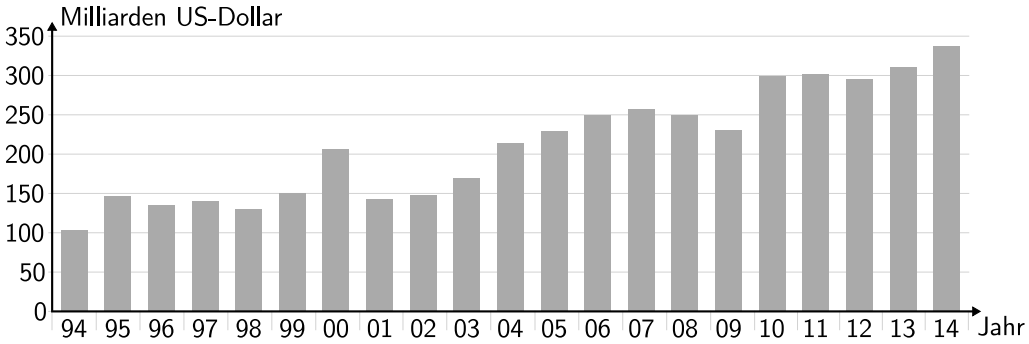


Abbildung 2.4: Weltweiter Halbleitermarkt [nach 200, S. 2]

Dabei ist die Halbleiterindustrie kein geschlossener Markt, sondern gilt auch für andere Industriezweige als einer der wichtigsten Wirtschaftstreiber. Beispielsweise hatte die Kommunikationsindustrie 2014 einen Anteil von 34,4 % am weltweiten Halbleitermarkt, die Konsumgüterindustrie von 12,9 % und die Automobilindustrie von 10,4 % [200, S. 10]. Somit stellt die Entwicklung von Halbleitern und digitaler Hardware eine wesentliche Komponente für den Erfolg von Projekten anderer Industriezweige dar.

Dabei gelten Projekte als Wertschöpfungsmotor und ermöglichen, dem zunehmenden Termin- und Kostendruck adäquat zu begegnen [50, S. 2]. Allerdings konnten mehrere Erhebungen feststellen, dass viele Halbleiterprojekte ihren Projektplan nicht einhalten. Eine Studie zeigt, dass nur 45 % der neu eingeführten Produkte ihren ursprünglichen Zeitplan einhalten, es lediglich 59 % der Entwürfe in die Produktion schaffen und weniger als 60 % der Projekte die geplanten Produktionskosten einhalten [168, S. 5]. Zudem brauchen mehr als 60 % der Chipentwürfe eine

grundlegende Überarbeitung, über 40 % der Projekte überschreiten den geplanten Finanzrahmen und 83 % der bei der Validierung festgestellten Fehler sind auf den Entwurf zurückzuführen [168, S. 5]. Noch gravierender fiel die Erhebung des Unternehmens Numetrics von 2009 aus [206, S. 7], welches ein Entwurfswerkzeug zur Planung von Hardwareprojekten anbot. Untersucht wurden die Abweichungen in Wochen der tatsächlichen Fertigstellung von IC-Projekten zur ursprünglichen Planung, wozu die Kundinnen und Kunden des Unternehmens befragt wurden [206, S. 7]. Die Ergebnisse sind in Abbildung 2.5 dargestellt und zeigen, dass etwa 60 % der IC-Projekte eine Verzögerung von mehr als einem Quartal haben und 16 % sogar eine Verzögerung von mehr als einem Jahr [206, S. 7].

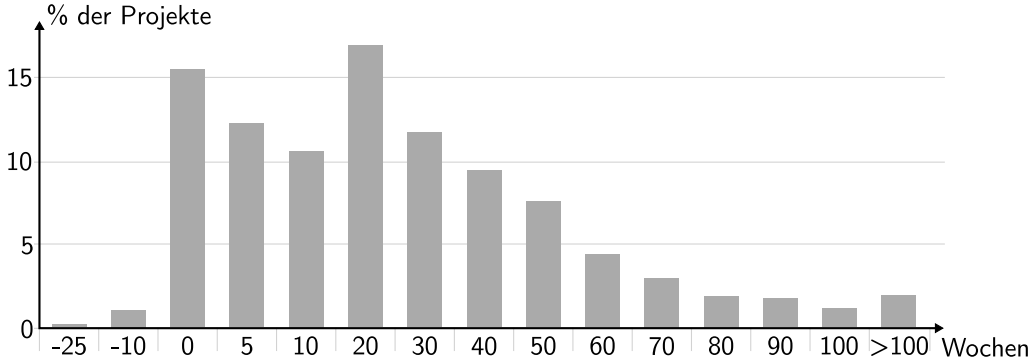


Abbildung 2.5: Abweichungen vom Zeitplan in Wochen [nach 206, S. 7]

In diesem Zusammenhang steht auch die sogenannte Entwurfslücke (Design (Productivity) Gap) aus der folgenden Abbildung 2.6. Zum einen wird in der Grafik das Mooresche Gesetz aufgetragen, welches die Entwicklung der Transistoranzahl pro Chip über die Zeit abbildet und das technologisch Machbare beziehungsweise die Entwurfskomplexität repräsentiert. Zum anderen wird die Produktivität ebenfalls über die Zeit aufgetragen. Sie wird hierzu als Arbeitsleistung eines Entwicklers oder einer Entwicklerin pro Monat aufgefasst, die als Anzahl umgesetzter Transistoren pro Monat angegeben wird. Der Entwurf kann dabei nicht nur auf der Transistorebene erfolgen, sondern auch auf höheren Abstraktionsebenen, sodass in diesem Fall die Transistoranzahl der Umsetzung betrachtet wird, im Sinne einer Äquivalenz des Entwurfs in Transistoren. Um die Entwicklung darzustellen, wird für beide Datenreihen die gleiche logarithmische Skalierung gewählt, wobei sich die beiden Geraden im ersten Jahr (1981) schneiden. Die gewonnene Grafik zeigt, dass die Entwurfskomplexität mit 58 % pro Jahr wächst, während die Entwurfsproduktivität mit nur 21 % pro Jahr ansteigt [227, S. 36]. Somit kommt es zu einer Lücke zwischen dem, was technologisch machbar ist und dem, was beim Entwurf geleistet werden kann.

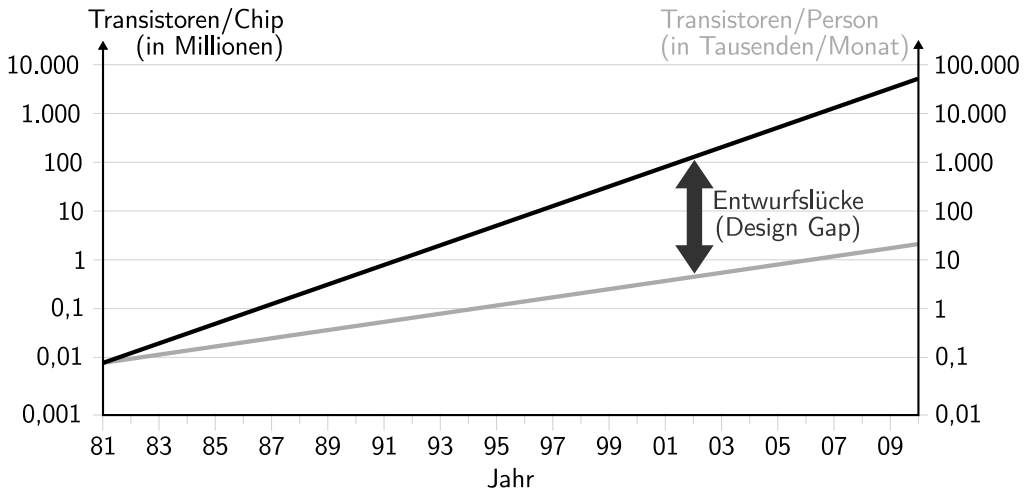


Abbildung 2.6: Entwurfslücke (nach [227, S. 36] [228, S. 8] [229, S. 8])

Dabei stellen die geforderte Leistung, die beanspruchten Einsatzmittel und die benötigte Zeit die Grundparameter eines Entwicklungsvorhabens dar [34, S. 23]. Ziel des Projektmanagements ist es, diese Parameter in ein optimales Verhältnis zueinander zu bringen [34, S. 23]. Dieser Zusammenhang wird durch das sogenannte „magische Projektmanagementdreieck“ in Abbildung 2.7 dargestellt [34, S. 23] [62, S. 110 f.]. Leistung, Einsatzmittel und Zeit stehen dabei in einer gegenseitigen Wechselwirkung [34, S. 23]. Je nach Zielrichtung ist das optimale Verhältnis der Grundparameter unterschiedlich [34, S. 23 f.]. Gemeinsam ist aber allen Zielrichtungen, dass das Projektmanagement Daten über das Projekt benötigt, um eine Optimierung vornehmen zu können [209, S. 267, 269]. Hierzu gehören insbesondere Informationen über den Umfang (Quantität) und die Komplexität eines Produkts [77, S. 202].

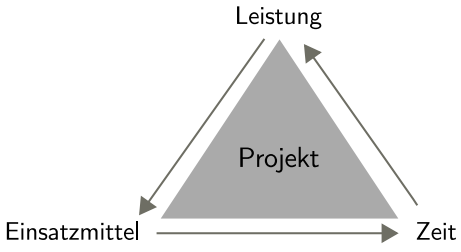


Abbildung 2.7: Magisches Projektmanagementdreieck [nach 34, S. 23]

Das Projektmanagement ist ein Problemlösungsprozess, der die Aufgaben hat, diesen Prozess zu planen, die Einhaltung der Pläne zu überwachen und bei Abweichungen steuernd einzugreifen [108, S. 6]. Der Projektverlauf kann in vier

grundlegende Phasen gegliedert werden [nach 34, S. 12], welche in Abbildung 2.8 dargestellt sind: Projektdefinition, Projektplanung, Projektkontrolle und Projektabschluss. Während des Verlaufs hat das Projektmanagement dabei verschiedene Funktionen [34, S. 12], wobei die Aufwandsschätzung während der einzelnen Phasen eine besondere Rolle spielt [77, S. 206 f.], wie im Folgenden beschrieben wird.

Abbildung 2.8: Phasen des Projektverlaufs [nach 34, S. 12]

1. Projektdefinition:
   Zur Projektdefinition gehören die Projektgründung, die Definition der Projektziele sowie die Organisation des Projekts und der Prozesse [34, S. 13]. Die Projektdefinition ist die Grundlage einer zuverlässigen Aufwandsschätzung [62, S. 105], welche auch dazu dient, die veranschlagten Kosten und Ressourcen zu überprüfen [34, S. 35].

2. Projektplanung:
   Kern dieser Phase ist die Strukturplanung, welche die Aufwandsschätzung, die Arbeitsplanung und die Kostenplanung umfasst [34, S. 14]. Die Aufwandsschätzung stellt die Basis für die Arbeitsplanung dar [34, S. 14]. Hierzu wird das Projekt in der Regel in Teilschritte zerlegt [62, S. 55] [77, S. 203]. Zudem werden innerhalb dieser Phase verschiedene Alternativen zur Umsetzung des Projekts evaluiert. Eine adäquate Abschätzung des Aufwands der Alternativen bietet eine zuverlässige Entscheidungsmöglichkeit. Kommt es dabei zu einer nicht angemessenen Aufwandsschätzung, werden regelmäßig die geplanten Projektkosten überschritten und/oder das Projekt erst nach dem beabsichtigten Fertigstellungstermin beendet [62, S. 105 ff.].

3. Projektkontrolle:
   Während der Projektumsetzung ist für das Projektmanagement vor allem die Projektkontrolle vorrangig [77, S. 205]. Hierzu gehören die Terminkontrolle, die Projektfortschrittskontrolle, die Qualitätskontrolle, die Projektdokumentation, die Aufwands- und die Kostenkontrolle [34, S. 15 f.]. Während der Projektdurchführung ist ein Ist-Soll-Vergleich wesentlicher Bestandteil des Projektmanagements, um Abweichungen vom Projektplan frühzeitig erkennen zu können [34, S. 10] [77, S. 205]. Dabei ist es erforderlich, dass die

angewendete Methode zur Bestimmung des Aufwands diesen zuverlässig abbildet, sodass ein Ist-Soll-Vergleich aussagekräftige Informationen liefert [34, S. 197 ff.].

4. Projektabschluss:
   Der Projektabschluss umfasst die Produktabnahme, die Projektabschlussanalyse, die Erfahrungssicherung und die Projektauflösung [34, S. 16]. Insbesondere bei der Projektabschlussanalyse sowie der Erfahrungssicherung dient der tatsächliche Aufwand eines Projekts dazu, als Kennzahl für weitere Projekte zur Verfügung zu stehen [34, S. 303 ff.] [107, S. 224]. Hierfür ist entscheidend, dass die verwendeten Kennzahlen auch den tatsächlichen Aufwand abbilden, sodass dieser als Grundlage weiterer Schätzungen dienen kann [34, S. 305 ff.]. Regelmäßig findet in dieser Phase auch ein Vergleich verschiedener Projekte hinsichtlich ihres Aufwands, ihrer Dauer und ihrer Kosten statt, welcher erfordert, dass die hierfür gewählten Aufwandskennzahlen den tatsächlich erbrachten Aufwand widerspiegeln.

Die Betrachtung der einzelnen Projektphasen zeigt, dass die Bestimmung des Projektaufwands während des gesamten Projektverlaufs eine wichtige Rolle spielt. Um den Aufwand zu bestimmen, können verschiedene Aufwandsschätzmethoden eingesetzt werden, welche im folgenden Kapitel vorgestellt werden. Basis dieser Methoden bilden unterschiedliche Kennzahlen, mit welchen insbesondere der Personalaufwand als wesentlicher Kostentreiber abgeschätzt wird. Neben den projektspezifischen Faktoren wie der Erfahrung oder Teamzusammensetzung hängt der Aufwand insbesondere vom zu entwickelnden Produkt ab. Anhand von charakteristischen Merkmalen und Daten des Produkts werden die Quantität und Komplexität bestimmt. An dieser Stelle setzt die in der vorliegenden Arbeit vorgestellte Entwurfsentropie (Design Entropy) an, welche insbesondere bei Nachrichten verarbeitenden Systemen eingesetzt werden kann, was der folgende Abschnitt zu den Zielen beschreibt.

2.4 Ziele der Arbeit

Das grundlegende Ziel der vorliegenden Arbeit ist die Bereitstellung einer neuen Kennzahl für digitale Hardwareprojekte und digitale Schaltungen, welche im Rahmen der Aufwandsbestimmung eingesetzt werden kann. Aufbauend auf den vorherigen Abschnitten werden zusammenfassend folgende Ziele der Arbeit festgelegt, welche gleichzeitig die Anforderungen an die Metrik und die Kennzahl darstellen:

- Entwurfsbezogenheit:
  Die Entwurfsentropie soll sowohl die Quantität (Entwurfsgröße) als auch die (Schaltungs-)Komplexität berücksichtigen.

- Unabhängigkeit:
  Die Metrik soll unabhängig von (äußeren) Einflussfaktoren sein und die im Entwurf/Produkt enthaltene (objektive) Komplexität widerspiegeln.

- Theoriegeleitetheit:
  Die Metrik soll aus einer theoretischen Grundlage bestehen, welche wiederum auf einer anerkannten Methodik basiert, sodass sich die Entwurfsentropie algebraisch berechnet.

- Nachvollziehbarkeit:
  Die mithilfe der Metrik berechnete Entwurfsentropie soll so nachvollziehbar sein, dass Entwicklerinnen und Entwickler die Ergebnisse einordnen können.

- Hierarchie:
  Es soll möglich sein, einzelne Komponenten eines Systems zu übergeordneten Komponenten zusammenzufassen (Bottom-up-Methode). Ebenfalls soll die Partitionierung eines Entwurfs in Teilentwürfe möglich sein (Top-down-Methode).

- Wiederverwendbarkeit:
  Die Metrik soll die Wiederverwendbarkeit von Komponenten sowie den Rückgriff auf Bibliotheken bei der Berechnung der Komplexität/Quantität berücksichtigen.

- Direkte Berechenbarkeit:
  Die Entwurfsentropie soll direkt anhand eines Hardwareentwurfs berechnet werden können, ohne den Rückgriff auf, beziehungsweise die Kalibrierung durch, umfangreiche empirische Daten.

- Automatisierung:
  Die Metrik soll so umgesetzt werden, dass es möglich ist, die Kennzahl der Entwurfsentropie für einen in einer Hardwarebeschreibungssprache vorliegenden Entwurf computergestützt/automatisiert zu berechnen.

- Universalität:
  Die Metrik soll so definiert werden, dass sie in unterschiedlichen Domänen und auf verschiedenen Abstraktionsebenen angewendet werden kann.

- Abstraktheit:
  Eine abstrakte Formulierung der Metrik soll die Anwendung auf andere Nachrichten verarbeitende Systeme ermöglichen, insbesondere auf Softwareprojekte und Eingebettete Systeme.

- Erweiterbarkeit:
  Durch eine abstrakte Definition der Metrik soll es zudem möglich sein, diese um weitere Randbedingungen zu erweitern.

Im Kern soll somit die Metrik Hardwareentwürfe hinsichtlich ihrer Quantität (Entwurfsgröße) und (Schaltungs-)Komplexität analysieren. Die Hardwareentwürfe bilden dabei die Grundlage der Berechnung der Entwurfsentropie, wie Abbildung 2.9 illustriert. Sie werden hierzu als Aufbau einzelner Komponenten betrachtet, sodass diese Komponenten die Messobjekte darstellen. Die Messvariablen sind die Anzahlen möglicher Zustände der Komponentenschnittstellen, mit welchen sich die Entwurfsentropie berechnen lässt.

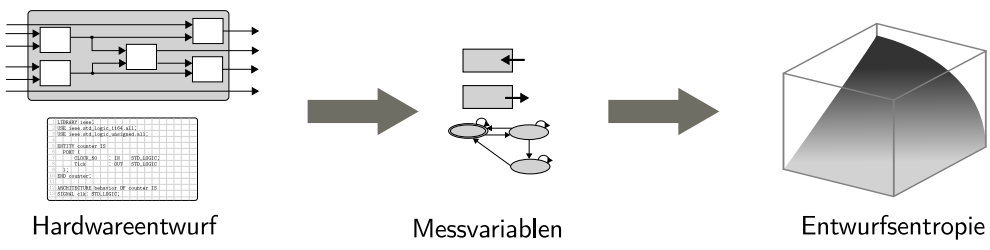


Abbildung 2.9: Ansatz der Entwurfsentropie

Durch die Anzahl an Komponenten eines Systems wird grundsätzlich die Entwurfsgröße berücksichtigt. Dabei fallen Komponenten unterschiedlich aus, sodass nicht die Anzahl an Komponenten direkt berücksichtigt wird, sondern die Schnittstellen der Komponenten mit ihren möglichen Zuständen. Das heißt, jede Schnittstelle hat die Möglichkeit, eine bestimme Anzahl an Nachrichten zu senden oder zu empfangen. Durch die Betrachtung der Anzahl an Schnittstellen der Komponenten wird die Entwurfsgröße und durch die Betrachtung der möglichen Nachrichten die Komplexität berücksichtigt, sodass die berechnete Entwurfsentropie eine gemeinsame Kennzahl für beide Dimensionen bildet.

Hintergrund der Entwicklung der Metrik ist das Verständnis, dass beim Entwurf digitaler Schaltungen im Kern die Kommunikation der Komponenten untereinander entworfen wird. Das heißt, die Komponenten werden so miteinander verbunden/ verdrahtet, dass das Gesamtsystem die vorgesehenen Aufgaben erfüllen kann. Dies bildet die in einem Entwurf liegende Komplexität ab, ohne dabei durch Randbedingungen beeinflusst zu werden, wie etwa der Teamzusammensetzung oder dem subjektiven Komplexitätseindruck. Durch den abstrakten Ansatz kann die Metrik auf Entwürfe in unterschiedlichen Domänen und auf verschiedenen Ebenen angewendet werden.

Insgesamt stellt das Maß der Entwurfsentropie eine Alternative zu heutigen Kennzahlen dar, welche speziell im Bereich der digitalen Schaltungen und allgemein im Bereich der Nachrichten verarbeitenden Systeme verwendet werden. Eine Übersicht über die mit der vorliegenden Arbeit in Zusammenhang stehenden Arbeiten, insbesondere im Bereich der Aufwandsbestimmung sowie über den aktuellen Stand der Forschung, gibt das folgende Kapitel.

3 In Zusammenhang stehende Arbeiten

Das vorherige Kapitel hat gezeigt, dass die Kenntnisse von Umfang und Komplexität eines Entwurfs von zentraler Bedeutung für das Projektmanagement und für Entscheidungen innerhalb des Entwurfsprozesses sind. Diese beiden Faktoren bilden die Kerngrößen der Aufwandsbestimmung, wobei auch weitere Kriterien einfließen. Daher wird zunächst ein Überblick über die Aufwandsbestimmung im Allgemeinen und die verwendeten Einflussfaktoren im Speziellen gegeben. Hierauf aufbauend werden die unterschiedlichen Klassen von Aufwandsbestimmungsmethoden vorgestellt. Der darauf folgende Abschnitt stellt die Methoden im Einzelnen vor und gibt dabei auch den Stand der Forschung wieder. Hieraus wird das Fazit der Literaturrecherche gezogen. Den Abschluss dieses Kapitels bildet eine kurze Einführung in die Informationstheorie, welche die theoretische Basis der vorliegenden Arbeit bildet.

3.1 Übersicht über die Aufwandsbestimmung

Grundsätzlich hat die Aufwandsbestimmung das Ziel, den Personalaufwand als wesentlichen Kostentreiber bei der Entwicklung zu bestimmen, welcher in Personenstunden, -tagen oder -monaten angegeben wird [161, S. 4]. Die Kosten ergeben sich aus der Multiplikation des Personalaufwands mit einem Verrechnungssatz [161, S. 4]. Zudem dient die Aufwandsbestimmung zur Planung der Dauer eines Projekts (Projektlaufzeit) [209, S. 269 f.] und ist eine Funktion des Aufwands (Dauer $= f(\text{Aufwand})$), welche insbesondere von der Teamgröße und der Verfügbarkeit von Projektmitarbeitenden abhängt [65, S. 119].

Für ihre Berechnungen stellen die jeweiligen Methoden einen funktionalen Zusammenhang zwischen bestimmten Datenquellen und dem zu bestimmenden Aufwand

beziehungsweise den Kosten her [34, S. 98]. Voraussetzung ist, dass die Daten messbar oder bestimmbar sind und mit dem Aufwand des Projekts korrelieren [34, S. 98], wobei die Quantifizierung der Einflussfaktoren die zentrale Herausforderung darstellt [161, S. 7]. Die Einflussfaktoren sind unabhängige Variablen, deren Veränderung zu einer Änderung einer oder mehrerer abhängiger Variablen führt [77, S. 432]. Beispielsweise führt die Änderung des Projektumfangs in Anzahl Quellcodezeilen (Lines of Code – LoC) zu einer Änderung des Projektaufwands in Personenmonaten (PM) [77, S. 434]. Grundsätzlich basieren die Aufwandsbestimmungsmethoden einerseits auf den Einflussfaktoren und andererseits auf empirischen Daten, wie Abbildung 3.1 zeigt. Grundlage der empirischen Daten sind Informationen über abgeschlossene Projekte [77, S. 433 f.]. Die Einflussfaktoren lassen sich in zwei Gruppen einteilen: produktbezogen und projektbezogen [77, S. 433 f.]. Die produktbezogenen Einflussfaktoren basieren auf charakteristischen Merkmalen und Daten des Produkts, während die projektbezogenen Einflussfaktoren auf bestimmten Merkmalen und Daten des Projekts basieren. Abhängig von der jeweils eingesetzten Methode unterscheiden sich die benötigten Daten für die Bestimmung des Aufwands.

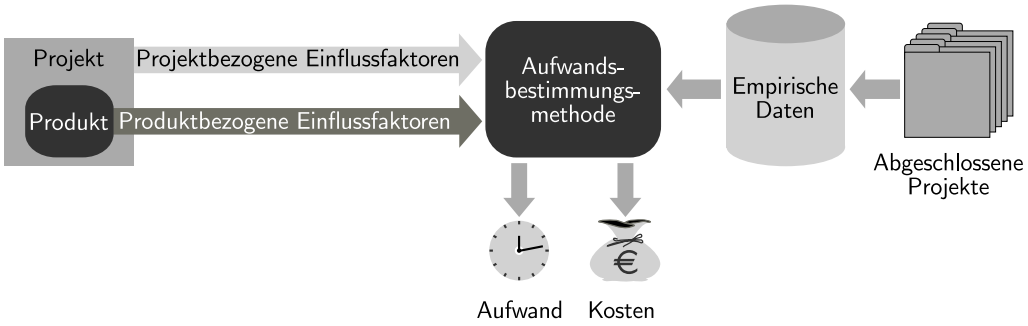


Abbildung 3.1: Einflussfaktoren und empirische Daten zur Aufwandsbestimmung

Die produktbezogenen Einflussfaktoren umfassen die Quantität (Produktgröße), die Komplexität und die Qualität [77, S. 434] [161, S. 7] [209, S. 277]. Dabei geht die Qualität nicht eigenständig in die Schätzung ein [77, S. 434], da kaum praktikable Lösungen zur Quantifizierung bestehen [161, S. 11]. Regelmäßig fließt die Qualität indirekt über die Quantität beziehungsweise über die Komplexität ein [161, S. 13]. Die zweite Gruppe bilden die projektbezogenen Einflussfaktoren, welche die Projektdauer, die Personalqualität und die Entwicklungsumgebung umfassen, wobei die beiden Letztgenannten die Produktivität bestimmen [77, S. 434].

Neben den Daten aus dem aktuellen Projekt fließen auch Daten von abgeschlossenen Projekten ein. Je nach Methode dienen diese empirischen Daten dazu, eine Kalibrierung der Aufwandsbestimmung vorzunehmen und/oder bestimmte Aspekte des aktuellen Projekts mit diesen Daten zu vergleichen, sodass eine Aussage über das aktuelle Projekt anhand der abgeschlossenen Projekte möglich ist. Dadurch, dass die Entwurfsentropie ein Maß für die Entwurfsgröße und die Komplexität darstellt, ist diese als produktbezogener Einflussfaktor einzuordnen, welcher keine empirischen Daten benötigt. Somit liegt der Schwerpunkt der folgenden Darstellung auf den produktbezogenen Einflussfaktoren. Da die innere Funktionsweise der Methoden auf einige grundlegende Klassen zurückgeführt werden kann, gibt der folgende Unterabschnitt zunächst eine Übersicht über diese.

3.1.1 Einteilung in Klassen

Die Methoden zur Aufwandsbestimmung versuchen mit unterschiedlichen Ansätzen den Projektaufwand beziehungsweise die Kosten abzuschätzen. Dabei lassen sie sich hinsichtlich ihrer inneren Funktionsweise in drei Hauptklassen einteilen [angelehnt an 34, S. 98]: Vergleichsmethoden, Kennzahlenmethoden und algorithmische Methoden. Abbildung 3.2 gibt einen Überblick der Einteilung und über die Methodenklassen, welche im Folgenden einzeln vorgestellt werden.

Abbildung 3.2: Klassen der Aufwandsbestimmungsmethoden

Zusätzlich existieren noch Vorgehensweisen, welche keine Methoden im eigentlichen Sinne sind, sondern das Vorgehen zur Aufwandsbestimmung beschreiben [ähnlich 33, S. 182, 232]. Die Verfahren hierbei schließen von einem bereits geschätzten Teil auf einen anderen Teil des Projekts. Eine Zwischenstellung zwischen den Methoden und den Vorgehensweisen nehmen die Experten- und Expertinnenbefragungen ein. Auf der einen Seite stellen sie Vorgehensweisen dar, um bestimmte Einflussfaktoren für andere Methoden zu bestimmen. Auf der anderen Seite können die Befragungen auch unabhängig eingesetzt werden, um den Aufwand eines Projekts direkt

abzuschätzen, sodass sie in gewissem Maße auch als Methoden fungieren. Die in Abbildung 3.2 dargestellten Klassen und Vorgehensweisen werden im Folgenden mit ihren charakteristischen Eigenschaften und Funktionsweisen beschrieben.

Vergleichsmethoden

Grundlage aller Vergleichsmethoden sind Daten über bereits abgeschlossene Projekte [34, S. 103]. Anhand von aussagekräftigen Vergleichskriterien wird versucht, den Aufwand eines Projekts anhand der Erfahrungsdaten aus abgeschlossenen Projekten zu bestimmen [34, S. 103]. Somit werden die Einflussfaktoren nicht direkt für die Schätzung herangezogen, sondern anhand dieser wird versucht, ein vergleichbares Projekt zu finden [77, S. 435]. Die Vergleichsmethoden können weiter in Analogiemethoden und Relationsmethoden eingeteilt werden [34, S. 98].

Bei den Analogiemethoden wird die Ähnlichkeit der Projekte für den Vergleich herangezogen [34, S. 103] [77, S. 436]. Dabei wird versucht aus den scheinbar unzusammenhängenden, abgeschlossenen Projekten dasjenige Projekt auszuwählen, welches dem zu bewertenden Projekt am ähnlichsten ist [34, S. 103]. Ausschlaggebend für das Auffinden der Analogie ist die Erfahrung der schätzenden Person [161, S. 22]. Aus der gefundenen Analogie wird dann versucht, Rückschlüsse auf den Aufwand, die Kosten und die Dauer zu ziehen [34, S. 103 f.]. Dies erfordert, dass die Vergleichskriterien sowohl für die abgeschlossenen Projekte als auch für das zu bewertende Projekt bestimmt sind [77, S. 435].

Die Relationsmethoden verwenden grundsätzlich den gleichen Ansatz wie die Analogiemethoden [34, S. 106] [77, S. 435] [161, S. 22]. Es wird aber nicht mehr dem oder der Schätzenden allein überlassen, das ähnlichste Projekt zu finden [161, S. 22], sondern die Auswahl basiert auf einem Formalismus, welcher anhand von Indikatoren das ähnlichste Projekt auswählt [34, S. 106] [77, S. 435]. Dies erfordert jedoch, dass den in der Erfahrungsdatenbank vorhandenen Projekten umfangreiche Indikatoren zugeordnet sind [34, S. 106]. Mustererkennungsroutinen ermöglichen dann den Vergleich mit dem zu bewertenden Projekt [161, S. 22]. Die Einflussfaktoren des zu bewertenden Projekts können auch mit mehreren abgeschlossenen Projekten verglichen werden, sodass sich der geschätzte Aufwand als (gewichteter) Mittelwert ergibt [77, S. 435].

Kennzahlenmethoden

Wie die Vergleichsmethoden setzen auch die Kennzahlenmethoden einen umfangreichen Bestand an Daten abgeschlossener Projekte voraus [34, S. 106]. Aus diesen Daten werden eine oder mehrere aussagekräftige Kennzahlen abgeleitet, welche den Aufwand mit einem produktbezogenen Einflussfaktor verbinden [33, S. 178]. Zu den Kennzahlenmethoden gehören als Unterklassen die Multiplikatormethoden und die Produktivitätsmethoden [34, S. 98].

Bei den Multiplikatormethoden ergibt sich der Aufwand aus der Multiplikation einer Kennzahl (beispielsweise 30 $\frac{\text{PM}}{\text{LoC}}$) mit einem produktbezogenen Einflussfaktor (beispielsweise LoC) des zu bewertenden Projekts, sodass ein einfacher linearer Zusammenhang besteht [34, S. 107] [77, S. 436]. Ein anderes Vorgehen der Multiplikatormethoden ist die Zerlegung des Systems in mehrere Teile, wobei diesen Teilen ein (einheitlicher) Aufwand zugeordnet wird, sodass sich der Gesamtaufwand als Produkt der Anzahl an Teilsystemen und dem Aufwand pro Teilsystem ergibt [161, S. 22].

Die Produktivitätsmethoden stellen ebenfalls einen linearen Zusammenhang her, wobei die Produktivität, das heißt, die Ergebniseinheiten je Kosten/Aufwand (beispielsweise 30 $\frac{\text{LoC}}{\text{PM}}$) als Kennzahl verwendet wird [34, S. 108]. Auch hier besteht die Möglichkeit, mehrere Kennzahlen zu verwenden, sodass sich der Aufwand $A = \frac{M}{P} \cdot \prod E_i$ aus der Division der Ergebnismenge M durch die Produktivität P multipliziert mit den Einflussfaktoren E_i ergibt [34, S. 108].

Algorithmische Methoden

Die algorithmischen Methoden bestimmen den Aufwand anhand einer oder mehrerer Formeln [34, S. 98]. Die Formeln sowie deren Faktoren werden empirisch und/oder mathematisch bestimmt [34, S. 98]. Somit erfordern auch die algorithmischen Methoden eine umfangreiche Datenbasis bereits abgeschlossener Projekte [203, S. 5]. Mithilfe einer Korrelations-/Regressionsanalyse der Einflussparameter (projektbezogene Einflussfaktoren) kann ein Zusammenhang zwischen der Ergebnisgröße (produktbezogene Einflussfaktoren) und dem Aufwand (meist in Personenmonaten) hergestellt werden [34, S. 99] [77, S. 435]. Dieser Zusammenhang ist in Abbildung 3.3 dargestellt. Anhand von empirisch bestimmten Einflussparametern wird ein Zusammenhang zwischen einer Ergebnisgröße und dem zu erwartenden

Aufwand hergestellt. Mithilfe der erwarteten Ergebnisgröße eines zu planenden Projekts kann der erwartete Aufwand bestimmt werden. Im Gegensatz zu den Kennzahlenmethoden besteht somit nicht nur ein einfacher linearer Zusammenhang [34, S. 107]. Zu den algorithmischen Methoden gehören die parametrischen Methoden und die Faktorenmethoden (auch Gewichtungsmethoden genannt) [34, S. 98].

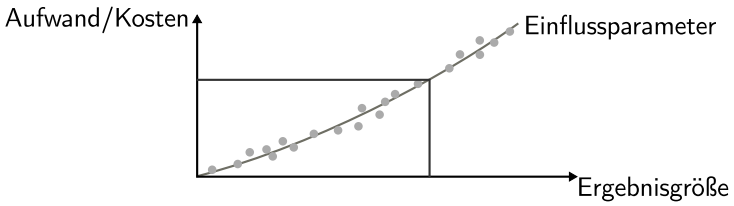


Abbildung 3.3: Prinzip der algorithmischen Methoden [nach 34, S. 99]

Bei den parametrischen Methoden ist der Personalaufwand $A = f(M, E_i)$ eine Funktion der Menge einer Ergebnisgröße M und der Einflussparameter E_i [34, S. 99]. Die Funktion ist jeweils spezifisch für die einzelnen Methoden, wobei sich grundsätzlich der Zusammenhang zwischen Aufwand und Ergebnisgröße über die Einflussparameter durch die Analyse einer großen Anzahl bereits abgeschlossener Projekte ergibt [34, S. 99].

Die Faktoren- beziehungsweise Gewichtungsmethoden bestimmen den Zusammenhang zwischen den Einflussgrößen und dem Aufwand durch bestimmte Faktoren und deren Gewichtung [34, S. 101]. Das heißt, es werden einzelne Faktoren bestimmt, welche den Aufwand bestimmen (beispielsweise der Komplexitätsgrad) und diese für das zu bewertende Projekt entsprechend gewichtet (beispielsweise einfach, mittel, schwer) [34, S. 101 f.]. Somit ist es erforderlich, dass zunächst ein für die Aufwandsschätzung relevantes System von Faktoren erstellt wird [161, S. 23]. Der jeweilige Einfluss der gewichteten Faktoren auf den Aufwand wird durch die Analyse abgeschlossener Projekte bestimmt [161, S. 23].

Im Einzelfall kann eine Abgrenzung zwischen den einzelnen Klassen schwierig sein, da regelmäßig mehrere Ansätze eingesetzt werden. Insbesondere ist eine eindeutige Zuordnung zu den algorithmischen Methoden oder den Kennzahlenmethoden nicht immer möglich, da eine hohe Verdichtung der Erfahrungswerte nur eine oder wenige Kennzahlen entstehen lässt, sodass der formelmäßige Zusammenhang in den Hintergrund rückt [ähnlich 34, S. 102].

Expertinnen- und Expertenbefragung

Auch wenn die Experten- und Expertinnenbefragung keine analytische Methode im eigentlichen Sinne darstellt, ist es wohl die am meisten verwendete Methode zur Aufwandsbestimmung [34, S. 123]. Teilweise wird sie in Fachbüchern sogar als einzige Methode vorgestellt [7, S. 63 ff.] [62, S. 107 ff.] [130, S. 194 f.]. Die Befragung setzt das Vorhandensein eines ausreichenden Erfahrungsschatzes bei den befragten Personen voraus [34, S. 124] [107, S. 139]. Die Befragungen können in vier grundlegende Arten eingeteilt werden: die Einzelbefragung, die Mehrfachbefragung, die Delphimethode und die Schätzklausur (Gruppenschätzung) [34, S. 124]. Sie unterscheiden sich in der Anzahl an befragten Personen sowie in der Durchführung des Befragungsverfahrens. Die Arten werden in den Unterabschnitten 3.2.1 bis 3.2.4 einzeln beschrieben.

Die Expertinnen- und Expertenbefragung kann nicht nur dazu eingesetzt werden, den Aufwand abzuschätzen, sondern auch, um einzelne Einflussfaktoren für andere Aufwandsbestimmungsmethoden zu schätzen. Daher stellen die Befragungen ebenfalls Vorgehensweisen dar.

Verfahren

Die Verfahren zur Aufwandsbestimmung nehmen im Kern keine eigene Schätzung vor, sondern schließen anhand der (vorhandenen) Schätzung eines Teils auf den Aufwand für die anderen Teile des Projekts [so auch 33, S. 182, 232]. Somit setzen alle Verfahren voraus, dass der Aufwand für einen oder mehrere Teile bereits bestimmt ist, entweder durch eine bereits vorgenommene Schätzung oder den Abschluss [77, S. 436] [161, S. 23]. Zu den Verfahren gehören die Prozentsatzmethode, die Bottom-up-Methode und die Top-down-Methode.

Bei der Prozentsatzmethode wird der Aufwand für eine zu schätzende Phase A_j anhand des Prozentsatzes g der bereits bestimmten Phase A_i mithilfe eines einfachen Zusammenhangs $A_j = g \cdot A_i$ übertragen [34, S. 110, 119 f.]. Abbildung 3.4 zeigt beispielhaft die Anwendung. Sind der Aufwand für die erste Phase (Definition) und die Prozentsätze der Phasen bekannt, so kann der voraussichtliche Aufwand auf die folgenden Phasen übertragen werden. Dies setzt voraus, dass empirisch abgeleitete Prozentsatzreihen für die Entwicklungsprojekte vorhanden sind [34, S. 110] [77, S. 436]. Somit wird nur die Information über die prozentuale Verteilung

des Projektaufwands für die Schätzung weiterer Projekte benötigt [77, S. 436] [161, S. 23]. Jedoch kann dieses Vorgehen nur dann gewählt werden, wenn zumindest für einen Teil der Aufwand bereits bekannt ist.

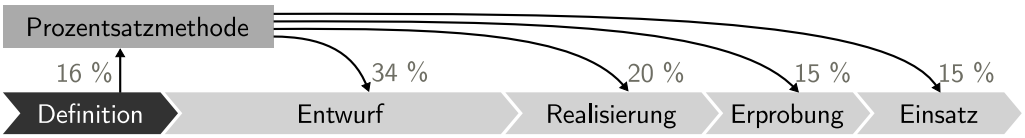


Abbildung 3.4: Beispielhafte Anwendung der Prozentsatzmethode mit bekanntem Aufwand der Definitionsphase [nach 34, S. 109]

Bei der Bottom-up-Methode wird das Projekt in kleinere Teile zerlegt. Um den Gesamtaufwand des Projekts zu bestimmen, bestehen grundsätzlich zwei Vorgehensweisen: Bei der ersten Möglichkeit werden die Teile so weit zerlegt, dass eine einfache Schätzung der Teile möglich ist und sich der Gesamtaufwand durch eine Berechnungsmethode ergibt, zum Beispiel durch Aufsummieren. Die zweite Möglichkeit besteht darin, kleine, repräsentative Ausschnitte eines Projekts zu finden und aus diesen den Gesamtaufwand zu extrapolieren [34, S. 111] [77, S. 436].

Umgekehrt verhält es sich bei der Top-down-Methode, welche versucht, aus der Gesamtschätzung auf den Aufwand der einzelnen Projektteile zu schließen, insbesondere im Verlauf des Projekts [34, S. 111]. Dabei wird die Schätzung immer detaillierter und genauer [34, S. 111].

3.1.2 Anwendbarkeit auf digitale Hardware

Neben der Unterscheidung der Methoden hinsichtlich ihrer inneren Funktionsweise können diese auch dahin gehend eingeteilt werden, ob sie speziell für Hardwareprojekte entwickelt wurden. Für einen umfassenden Überblick werden im Folgenden auch Methoden vorgestellt, welche beim Entwurf von Hardwareprojekten angewendet werden können, ohne dass diese speziell hierfür entwickelt wurden. Denn die den Methoden zugrunde liegenden Ansätze sind regelmäßig auch auf Hardwareprojekte anwendbar. Teilweise sind dabei Anpassungen und/oder Erweiterungen erforderlich. Aufgrund der Vielzahl derartiger Methoden in Literatur und Wirtschaft beschränkt sich die folgende Darstellung einerseits auf Methoden, welche in einem engen Zusammenhang mit der vorliegenden Arbeit stehen und andererseits auf Methoden, welche die Vielfalt der unterschiedlichen Ansätze zeigen.

Tabelle 3.1 gibt einen Überblick über die im Folgenden dargestellten Methoden. Dabei sind diese alphabetisch sortiert in drei Kategorien eingeteilt. Die erste Kategorie bilden diejenigen Methoden, welche explizit für Hardwareprojekte entwickelt wurden. Methoden der zweiten Kategorie wurden zwar nicht speziell für die Aufwandsberechnung von Hardwareprojekten entwickelt, können aber direkt hierauf angewendet werden. Hierzu zählen insbesondere die Methoden der Expertinnen- und Expertenbefragung. In die dritte Kategorie sind diejenigen Methoden eingeordnet, welche für die Aufwandsbestimmung von Softwareprojekten entwickelt wurden und durch Anpassungen und/oder Erweiterungen auch indirekt auf Hardwareprojekte angewendet werden können.

Metriken explizit für Hardwareprojekte	Direkte Anwendbarkeit auf Hardwareprojekte	Indirekte Anwendbarkeit auf Hardwareprojekte
ACM	CAD-Datensammlung	Aron
COSYSMO	Delphimethode	COCOMO 81
Hardwarekostenmodell	Einzelschätzung	COCOMO II
Metrics	Mehrfachbefragung	Funktionspunktanalyse
Numetrics	Modeling Metric Tool	Halstead-Metrik
PRICE H	n-dim System	Handbuchverfahren
Produktiv+	Schätzklausur	Mark II
Progress	McCabe-Metrik	Objektpunkte
SEER-H		SLIM
SOG-Modell		Walston-Felix
Synopsys		Wolverton
μComplexity		Softwareentropie

Tabelle 3.1: Anwendbarkeit der Methoden [angelehnt an 33, S. 182]

3.2 Methoden zur Aufwandsbestimmung

Im Folgenden werden die Methoden zur Aufwandsbestimmung jeweils einzeln vorgestellt. Da die Entwurfsentropie als produktbezogener Einflussfaktor einzuordnen

ist, liegt der Fokus auf den verwendeten Einflussfaktoren. Zudem wird die grundsätzliche Funktionsweise der Methoden beschrieben, um eventuell vorhandene Schwächen bei der Abschätzung aufzeigen zu können.

Einige der neueren Methoden bauen auf den älteren auf, sodass die folgenden Unterabschnitte vorrangig chronologisch sortiert sind, wobei thematisch zusammenhängende Methoden gemeinsam oder aufeinander folgend dargestellt werden. Zunächst werden die Experten- und Expertinnenbefragungen vorgestellt, da viele der Einflussfaktoren anderer Methoden dadurch bestimmt werden (müssen). Hierbei erfolgt die Qualifizierung und teilweise die Quantifizierung durch die Einschätzung von Expertinnen und Experten. Die in den vier folgenden Unterabschnitten vorgestellten Formen der Experten- und Expertinnenbefragung sind in Abbildung 3.5 illustriert. Die Punkte symbolisieren die involvierten Personen und die Verbindungen die Kommunikation zwischen ihnen.

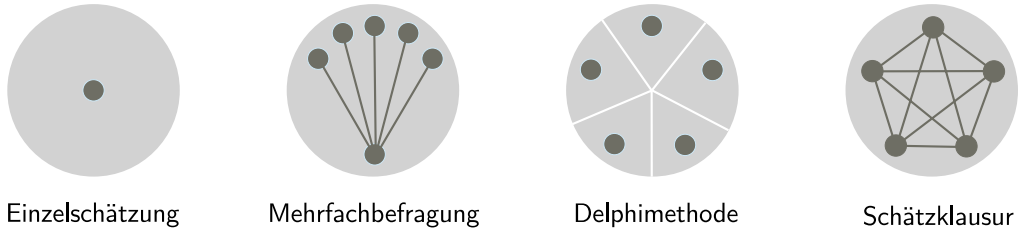


Abbildung 3.5: Formen der Experten- und Expertinnenbefragung [nach 34, S. 124]

3.2.1 Einzelschätzung

Bei der Einzelschätzung legt eine Person den Aufwand für eine bestimmte Menge oder Ausprägung fest [34, S. 124]. Wenn nur eine Person den Aufwand festlegt, kommt es zu einer Einseitigkeit der Bewertung, sodass auch die Kontrolle der Richtigkeit fehlt [34, S. 124]. Durch die Subjektivität der Schätzungen sind diese nur schwer mit anderen zu vergleichen und es fehlt der Erfahrungsaustausch zwischen den Personen [62, S. 108].

3.2.2 Mehrfachbefragung

Bei der Mehrfachbefragung werden mehrere Personen aus unterschiedlichen organisatorischen Einheiten unabhängig voneinander befragt [34, S. 125]. Die endgültige

Schätzung wird regelmäßig als Durchschnittswert festgelegt: beispielsweise als arithmetisches Mittel, als Mittelwert zwischen dem Minimal- und Maximalwert oder als arithmetisches Mittel ohne Extremwert [34, S. 125].

3.2.3 Delphimethode

Wie bei der Mehrfachbefragung werden auch bei der Delphimethode mehrere Personen befragt [31, S. 3] [46, S. 458 ff.]. Bei der Standard-Delphimethode werden die Expertinnen und Experten zunächst einzeln befragt [46, S. 458 f.]. Von der koordinierenden Person werden die Ergebnisse dann zusammengefasst und den befragten Personen wieder vorgelegt [46, S. 459]. Dieser Prozess wiederholt sich so lange, bis sich die Schätzungen hinreichend angenähert haben [193, S. 4 f.].

Bei der Breitband-Delphimethode werden die Schätzungen weiterhin unabhängig voneinander vorgenommen, jedoch finden zwischen den Iterationen gemeinsame Sitzungen statt, bei welchen die einzelnen Schätzungen diskutiert werden [25, S. 335] [215, S. 39 ff.]. Dieser Prozess wird ebenfalls beendet, wenn sich die Schätzungen weit genug angenähert haben [25, S. 335].

Daneben existieren weitere Abwandlungen und Erweiterungen der Delphimethode. Beispielsweise werden bei der Cooke-Methode [43, S. 12 ff.] die Meinungen der Expertinnen und Experten auf Basis ihres Wissens und ihrer Fähigkeiten, relevante Unsicherheiten zu beurteilen, gewichtet [12, S. 294]. Beim Policy-Delphi wird statt der Findung einer gemeinsamen Schätzung darauf abgezielt, den Prozess zu strukturieren und die unterschiedlichen Sichtweisen des gewünschten Projektverlaufs zu diskutieren [231, S. 159 ff.]. Das Disaggregative-Policy-Delphi verwendet eine Clusteranalyse, um verschiedene Szenarien aus der letzten Iteration des Delphiprozesses abzuleiten [224, S. 92 ff.]. Das Argument-Delphi fokussiert dagegen den Diskussionsprozess selbst und klassifiziert die von den Teilnehmenden vorgebrachten Argumente [120, S. 177 ff.].

3.2.4 Schätzklausur

Auch bei der Schätzklausur werden mehrere Personen befragt. Jedoch werden diese nicht mehr anonym und unabhängig voneinander befragt, sondern in einem gemeinsamen Meeting [34, S. 126 f.]. Das Projekt wird dabei in einem Bottom-

up-Vorgehen in Komplexe gegliedert, welche dann geordnet nach Größe und Komplexität in eine Matrix eingetragen werden [86, S. 222]. Dabei wird einer der Komplexe als Referenzkomplex mit großer Genauigkeit geschätzt [34, S. 127]. Anhand dieses Referenzkomplexes wird auf den Aufwand der anderen Komplexe geschlossen [86, S. 222]. Exemplarisch ist die Berechnung des Gesamtaufwands mithilfe der Referenzmatrix in Tabelle 3.2 dargestellt.

	Größe			
Komplexität	klein	mittel	groß	Summe in PT
gering	4 8 PT	3 12 PT	2 15 PT	32+36+30=98
mittel	1 10 PT	2 15 PT	2 20 PT	10+30+40=80
hoch	0 15 PT	2 20 PT	1 25 PT	0+40+25=65
			Gesamtaufwand	243 PT

Anzahl
Aufwand

Referenzkomplex
PT = Personentage

Tabelle 3.2: Referenzmatrix der Schätzklausur [nach 34, S. 127]

Im obigen Beispiel wurde das Projekt zunächst in 17 Komplexe unterteilt, welche anhand ihrer Komplexität und Größe bewertet wurden. Ein großer Komplex mit mittlerer Komplexität wurde als Referenzkomplex ausgewählt. Für diesen wurde geschätzt, dass die Umsetzung 20 Personentage beansprucht. Ausgehend von dieser Schätzung wurde dann der Aufwand für die anderen Komplexe in der Matrix abgeleitet. Anhand der Anzahl an Komplexen einer Größe und Komplexität und dem erwarteten Aufwand in Personentagen kann der Gesamtaufwand für das Projekt berechnet werden.

3.2.5 Handbuchverfahren

Eine der ältesten Methoden speziell zur Aufwandsschätzung stammt von IBM und wurde im Jahr 1968 veröffentlicht [90]. Dabei wird die Programmierzeit in Tagen anhand von fünf Gewichtungsfaktoren ($A \ldots E$) abgeschätzt [49, S. 148]. Aus einer Matrix an Einflussfaktoren ergibt sich der jeweilige Gewichtungsfaktor [49, S. 149 ff.]. Die Programmierzeit T in Personentagen berechnet sich mit der folgenden Formel (3.1) [49, S. 148, 154].

$$T = (A + B) \cdot (C + D) \cdot (1 + E) \tag{3.1}$$

mit A = Programmieraufwand für die Ein- und Ausgabe
B = Programmieraufwand für den Verarbeitungsteil
C = Geforderte Problemkenntnisse der/des Programmierenden
D = Erforderliche Programmiererfahrung
E = Störungen der Programmierung

Durch den Einsatz der Faktoren gehört das IBM-Handbuchverfahren zur Klasse der Gewichtungs- beziehungsweise Faktorenmethoden, sodass die Methode auch als IBM-Faktorenmethode bezeichnet wird [34, S. 102] [161, S. 24]. Grundsätzlich eignen sich alle Ansätze dieser Klasse auch für Hardwareprojekte [34, S. 102]. Dabei ist es erforderlich, spezifische Gewichtungsfaktoren für Hardware zu finden.

3.2.6 Aron

Zur Klasse der Produktivitätsmethoden gehört die von Joel Aron im Jahr 1970 vorgestellte und nach ihm benannte Aron-Methode [34, S. 109]. Sie basiert auf der Produktivität der Programmiererinnen und Programmierer, welche anhand der Anzahl ausgelieferter Instruktionen pro Zeiteinheit angegeben wird [11, S. 72]. Für die Abschätzung des Aufwands werden die Anzahl auszuliefernder Instruktionen, der Schwierigkeitsgrad (einfach, mittel, schwer) und die Projektdauer (bis 12 Monate, $12 - 24$ Monate, mehr als 24 Monate) abgeschätzt [11, S. 74 f.]. Größere Projekte können dabei in Teile zerlegt werden [11, S. 70]. Anhand einer Tabelle mit Produktivitätskennzahlen für die neun möglichen Kombinationen aus Schwierigkeitsgrad und Dauer berechnet sich der voraussichtliche Aufwand in Personenmonaten durch eine einfache Division (auszuliefernde Instruktionen/Instruktionen pro Personenmonat) [11, S. 75 f.]. Werden Instruktionen durch Gatter, Chipfläche, Transistoren oder Logikeinheiten ersetzt und eine hardwarespezifische Produktivitätstabelle aufgestellt, dann kann diese Metrik auch auf Hardwareentwicklungen angewendet werden. Jedoch merkte bereits Joel Aron an, dass die Methode nur eine begründete Vermutung darstellt und keinen Bezug zur Entwurfskomplexität herstellt [11, S. 72]. Somit ist nicht zu erwarten, dass eine hierauf basierende Hardwaremetrik bessere Ergebnisse als die Sotwaremetrik liefert.

3.2.7 Wolverton

Ray Wolverton stellte im Jahr 1974 eine Metrik vor, welche versucht, die Produktivität von Entwicklern und Entwicklerinnen anhand der Anzahl an Instruktionen zu messen [246, S. 624 f.], was heute etwa Programmzeilen entsprechen würde [161, S. 76]. Die Methode gehört zur Klasse der Multiplikatormethoden [34, S. 107 f.].

Die Produktivität wird als Instruktionen pro Personenmonat angegeben, wobei sechs Schwierigkeitslevel betrachtet werden, dabei drei Schwierigkeitsgrade (einfach, mittel, schwer) für zwei Arten von Projekten (neu, alt) [246, S. 625]. Zusätzlich werden die Projekte noch in sechs Kategorien eingeteilt: Steuerprogramme, Ein-/Ausgabeprogramme, Datenaufbereitungsprogramme, Rechenprogramme, Datenverwaltungsprogramme und zeitkritische Programme [246, S. 628]. Für alle Kategorien und Schwierigkeitslevel sind typische Produktivitäten angegeben, welche sich aus empirisch ermittelten Daten ergeben [246, S. 630].

Zur Berechnung des Aufwands wird ein Projekt in seine Teile gemäß der Kategorieneinteilung zerlegt, die Instruktionsanzahl abgeschätzt und die typische Produktivität durch diese dividiert [246, S. 629 f.]. Die Summe dieser Einzelbewertungen ergibt den Projektaufwand in Personenmonaten [246, S. 626]. Dieses Vorgehen kann grundsätzlich auch bei Hardwareprojekten eingesetzt werden. Die Schwierigkeitslevel können übernommen werden, jedoch müssen die Kategorien angepasst werden. Aus dieser Einteilung könnte auch, bei entsprechenden empirischen Daten, eine mittlere Produktivität für alle Kategorien und Schwierigkeitslevel angegeben werden. Im Kern besteht somit die Frage, durch welche Metrik die Instruktionsanzahl ersetzt werden soll oder ob diese durch die Verwendung von Hardwarebeschreibungssprachen mit der Anzahl an Quellcodezeilen weiterverwendet werden kann.

3.2.8 McCabe-Metrik

Die als McCabe-Metrik oder auch als zyklomatische Komplexität bekannte Berechnung der Komplexität von Softwaremodulen wurde 1976 von Thomas McCabe vorgestellt [134]. Als Grundlage für die Metrik dient die Anzahl an linear unabhängigen Pfaden durch ein Modul auf Basis seines Kontrollflussgraphen [134, S. 308]. Der Kontrollflussgraph gibt alle möglichen Pfade an, welche bei der Abarbeitung eines Programms durchlaufen werden können. Dabei werden die Basisblöcke als

Knoten dargestellt [52, S. 108]. Basisblöcke sind diejenigen Programmteile, welche zusammenhängend durchlaufen werden, das heißt, die insbesondere keine Verzweigungen oder Sprünge enthalten [52, S. 108]. Die Basisblöcke werden über Kanten miteinander verbunden, welche insbesondere Verzweigungen und Sprünge darstellen [52, S. 108]. Zudem enthält jeder Kontrollflussgraph einen Eingangsknoten und einen Ausgangsknoten [52, S. 108]. Aus der Anzahl an Kanten e, der Anzahl an Knoten n und der Anzahl an zusammenhängenden Komponenten p des Kontrollflussgraphen G ergibt sich die zyklomatische Komplexität $V(G) = e - n + p$ eines Moduls [134, S. 308 f.]. Die Anzahl zusammenhängender Komponenten ist diejenige Anzahl an Knoten, welche paarweise durch eine Kantenfolge des Graphen verbunden sind.

Somit hängt die zyklomatische Komplexität nur von der Anzahl an Verzweigungen eines Programms ab. Sonstige Anweisungen haben keinen Einfluss. Folglich liefert die Metrik eine Kennzahl für die Anzahl erforderlicher Testfälle, sodass alle Pfade des Graphen durchlaufen werden. Daher ist diese Metrik als Komplexitätsmaß nicht geeignet. Eine Forschungsarbeit wendet die zyklomatische Komplexität auf Verhaltensbeschreibungen in einer Hardwarebeschreibungssprache an [133, S. 484]. Auch hier musste festgestellt werden, dass die Metrik als Maß für die Wartbarkeit und Testbarkeit betrachtet werden muss [133, S. 484].

3.2.9 Walston-Felix

1977 stellten Claude Walston und Charles Felix eine Produktivitätsmethode zur Aufwandsschätzung vor [238]. Dabei wurden die unterschiedlichen Einflussfaktoren auf die Produktivität durch spezielle Projektbedingungen und Produktanforderungen berücksichtigt [34, S. 109]. Hierzu wurde eine Regressionsanalyse für 29 Einflussgrößen anhand von Daten abgeschlossener Projekte vorgenommen [34, S. 109] [161, S. 18 f., 132 f.]. Für die jeweiligen Einflussgrößen ergibt sich jeweils eine Produktivität, welche in einer Produktivitätstabelle zusammengestellt ist und anhand welcher ein Produktivitätsindex bestimmt ist, woraus sich die anzunehmende Produktivität ableitet [34, S. 109]. Eine Division der angenommenen Anzahl an Instruktionen durch die angenommene Produktivität ergibt den Personalaufwand [34, S. 109] [161, S. 19 f.]. Dabei wird jedoch nicht angegeben, wie die Anzahl an Quellcodezeilen abgeschätzt wird und die Komplexität fließt nur über die „Kundeninterface Komplexität“ sowie indirekt über weitere Einflussgrößen ein. Grundsätzlich kann diese Methode auf jegliche Art von Projekten angewendet werden, wenn auch mit anderen Einflussfaktoren.

3.2.10 Halstead-Metrik

Ein analytisches Verfahren aus dem Jahr 1977 stammt von Marice Halstead [67]. Dabei wurde der Ansatz gewählt, dass Programme aus Operanden (beispielsweise Variablen und Konstanten) und Operatoren (beispielsweise Vergleiche und Schlüsselwörter) aufgebaut sind [67, S. 6]. Die Komplexität D eines Programms ergibt sich nach Formel (3.2) [67, S. 27 f.]. Dabei ist η_1 die Anzahl unterschiedlicher Operatoren, η_2 die Anzahl unterschiedlicher Operanden und N_2 die Anzahl aller verwendeten Operanden [67, S. 6]. Der Aufwand E berechnet sich dann mit $E = D \cdot N \cdot \log_2(\eta_1 + \eta_2)$, wobei N die Anzahl aller Operanden und Operatoren ist [67, S. 47].

$$D = \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2} \tag{3.2}$$

3.2.11 SLIM

Das Softwarelebenszyklusmodell (Software-Lifecycle-Management – SLIM) ist eine Makroschätzmethode von Lawrence Putnam aus dem Jahr 1978, welcher der Lebenszyklus eines Produkts für die Schätzung zugrunde liegt [180, S. 345]. Sie gehört zur Klasse der parametrischen Methoden [34, S. 100]. Die SLIM-Methode verfolgt den Ansatz, dass während der Entwicklungsphasen (beispielsweise Definition, Entwurf, Realisierung, Erprobung und Einsatz) ein unterschiedlich hoher Personalbedarf besteht, welcher einer empirischen Verteilungskurve folgt [180, S. 348 f.]. Diese Verteilung wird durch die Norden-Rayleigh-Kurve beschrieben [160, S. 157 f.]. Abbildung 3.6 zeigt diese Kurve, einmal als Prozentsatz des Gesamtaufwands und einmal als kumulierter Aufwand in Prozent. Exemplarisch sind in der Abbildung auch die einzelnen Entwicklungsphasen dargestellt.

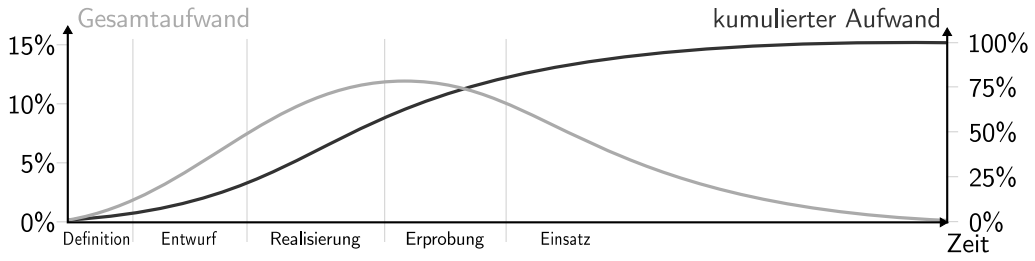


Abbildung 3.6: Norden/Rayleigh-Kurve [nach 160, S. 158] mit Entwicklungsphasen [aus 34, S. 87 f.]

Wird innerhalb der einzelnen Phasen die Fläche unter der Prozentsatzkurve des Gesamtaufwands betrachtet, dann ergibt sich die Anzahl an ausgelieferten Quellcodezeilen S aus dem Integral über die durchschnittliche Produktivität $\overline{P}$ und dem Arbeitseinsatz $\dot{y}_t$ zur Zeit t [180, S. 353 ff.] nach Gleichung (3.3). Anhand der Verteilungskurve lässt sich der Arbeitseinsatz $\dot{y}_t$ zur Zeit t angeben, sodass sich die Anzahl an Quellcodezeilen S in Gleichung (3.3) als Produkt einer Technologiekonstanten C, die aus früheren Projekten kalibriert werden kann, dem Gesamtaufwand K und der Entwicklungszeit T ergibt [180, S. 360] [181, S. 829]. Hieraus ergibt sich der Entwicklungsaufwand E in Personenmonaten mit $E = 0{,}4 \cdot K$ gemäß Gleichung (3.4) [180, S. 355].

$$S = \int_0^\infty \overline{P} \cdot \dot{y}_t \, dt = C \cdot K^{\frac{1}{3}} \cdot T^{\frac{4}{3}} \tag{3.3}$$

$$E = 0{,}4 \cdot S^3 \cdot C^{\frac{1}{3}} \cdot T^{\frac{1}{4}} \tag{3.4}$$

Mittlerweile wurde eine ganze Gruppe an SLIM-Methoden entworfen [152, S. 3 f.], welche vom Unternehmen Quantitative Software Management vertrieben werden [182]. Dabei wurde seit 1978 eine Datenbank mit Projektdaten erstellt, welche mehr als 10.000 Datensätze enthält, wobei die überwiegende Mehrheit von Softwareprojekten stammt, jedoch auch Daten von Hardwareprojekten vorhanden sind [10, S. 9 f.]. Statt Quellcodezeilen werden heute auch Funktionspunkte, welche im folgenden Unterabschnitt vorgestellt werden, als Kennzahl eingesetzt [18, S. 62 ff.]. Die Methodik basiert weiterhin auf der Rayleigh-Verteilung und der von Lawrence Putnam entwickelten Methode, wobei weitere Einflussfaktoren und umfangreichere Daten hinzugekommen sind [9, S. 35 ff.].

3.2.12 Funktionspunktanalyse

1979 veröffentlichte Allan Albrecht die für das Unternehmen IBM entwickelte Methode der Funktionspunktanalyse (Function-Point-Analysis – FPA) [3, S. 83]. Durch die Systematisierung auf funktionaler Ebene zählt die Funktionspunktanalyse zur Klasse der Analogiemethoden [34, S. 104]. Grundlage der Methode sind Funktionspunkte, welche die Anzahl an Eingaben, Ausgaben, Abfragen und Stammdaten einer Anwendung aus der Sicht eines Nutzers oder einer Nutzerin ausdrücken [3, S. 85]. Zunächst wurde separat die Anzahl der vier Funktionstypen nach der Funktionalität einer Anwendung bestimmt [3, S. 85]. Später wurden

die Stammdaten als interne Datenbestände gezählt und die Schnittstellen der Anwendung separat gezählt, sodass fünf Funktionstypen existieren [4, S. 641]. Der Ablauf der Funktionspunktanalyse mit den von Allan Albrecht vorgeschlagenen Gewichtungen der Komplexität [4, S. 647] ist in Abbildung 3.7 dargestellt und wird im Folgenden beschrieben.

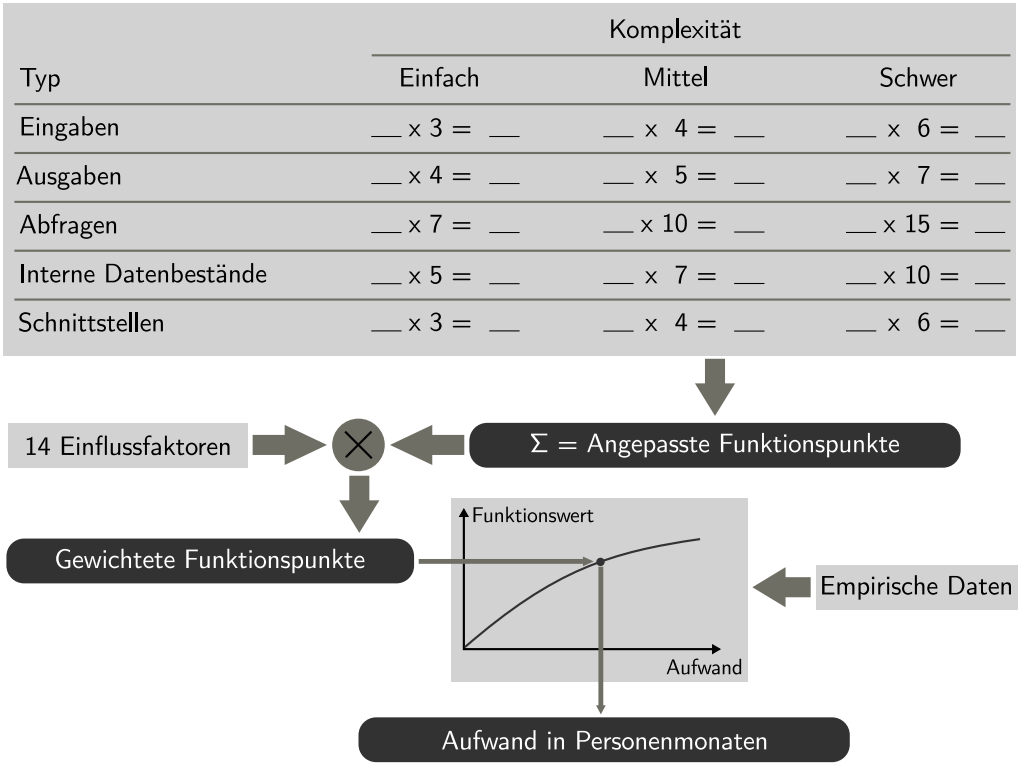


Abbildung 3.7: Ablauf der Funktionspunktanalyse (angelehnt an [1, S. 172 ff.] [4, S. 647] [161, S. 90])

Zu Beginn werden die einzelnen Funktionspunkte (der jeweiligen Typen) nach ihrer Komplexität gewichtet (einfach, mittel, schwer) [4, S. 646 ff.]. Die Anzahl Funktionspunkte in den einzelnen Einteilungen wird jeweils mit einem Faktor multipliziert, welcher entweder aus der Literatur oder aus vorherigen Projekten abgeleitet werden kann. Die Summe dieser (multiplizierten) Funktionspunkte bildet die Anzahl angepasster Funktionspunkte, welche mit einem Gewichtungsfaktor für die Komplexität der Datenverarbeitung multipliziert wird, welcher sich aus vierzehn weiteren Einflussfaktoren bestimmt [4, S. 647]. Hieraus ergeben sich die gewichteten Funktionspunkte [4, S. 647]. Mit einer Funktionswertkurve, die sich aus einer Funktionspunktanalyse von vorherigen/abgeschlossenen Projekten ergibt, können die gewichteten Funktionspunkte in den Aufwand in Personenmonaten umgerechnet werden [34, S. 104].

Das Verfahren der Funktionspunktanalyse wurde durch die International Function Point Users Group weiter verfeinert [100] und als Norm ISO/IEC 20926 standardisiert [103]. Für die Anwendung der Funktionspunktanalyse auf digitale Hardwareprojekte wird vorgeschlagen, als Funktionstypen Eingaben, Ausgaben, Abhängigkeiten, Neuerungen, Algorithmen und Unterkomponenten zu verwenden [101, S. 8].

Die zweite Generation der Funktionspunkte wurde 1998 vom internationalen COSMIC Konsortium (Common Software Measurement International Consortium – COSMIC) entwickelt [41, S. 13]. Die COSMIC-FPA basiert auf funktionalen Anforderungen und ist auch auf die Entwicklung von Geschäfts-, Echtzeit- und Infrastruktursoftware anwendbar [41, S. 14]. Als Funktionstypen gehen die Anforderungsdefinitionsartefakte, die Artefakte aus der Zerlegung der funktionalen Anforderungen sowie die Datenanalyse- und Modellierungsartefakte ein [40, S. 4]. Die COSMIC-FPA ist ebenfalls als Norm ISO/IEC 19761 standardisiert [104].

3.2.13 Mark II

Basierend auf der Funktionspunktanalyse entwickelte Charles Symons die Mark II Funktionspunktanalyse [221], welche als Norm ISO/IEC 20968 standardisiert ist [102]. Sie setzt auf logischen Transaktionen und auf Einheiten, durch welche die Anforderungen an die Software und die funktionelle Spezifikation typischerweise ausgedrückt werden, auf [232, S. 3]. Eine logische Transaktion ist dabei die niedrigste Stufe eines selbstständigen Prozesses und besteht aus drei Komponenten: die Eingabe über die Anwendungsgrenze, die Verarbeitung unter Einbeziehung gespeicherter Daten innerhalb der Grenze und die Ausgaben zurück über die Grenze [232, S. 8].

Dieser Ansatz könnte auch auf digitale Hardwareprojekte angewendet werden. Jedoch erfordern alle Arten der Funktionspunktanalyse einen umfangreichen Datenbestand abgeschlossener Projekte, welcher für die Kalibrierung der Gewichtungsfaktoren, für die Komplexität der Funktionspunkte und die Gewichtung der Komplexität durch die Datenverarbeitung erforderlich ist.

3.2.14 Objektpunkte

Die Objektpunktmethode nach Rajiv Banker [14] aus dem Jahr 1992 verfolgt einen ähnlichen Ansatz wie die Funktionspunktmethode. Sie geht davon aus, dass der Aufwand durch diejenigen Teile einer Anwendung wiedergegeben werden kann, welche einen hohen Entwicklungsaufwand benötigen [167, S. 31]. Somit gehört die Objektpunktmethode auch zur Klasse der Analogiemethoden [34, S. 105].

Bei der Objektpunktmethode stellen die Objekte die Bausteine dar, die für die gewünschte Funktionalität der Anwendung verwendet werden [14, S. 136]. Durch diese Platzhalter kann die Objektpunktmethode bereits frühzeitig in der Systementwicklung eingesetzt werden [89, S. 55]. Als Objekte werden Regelsätze, Bildschirmmasken, generierte Berichte und 3GL-Module gezählt [14, S. 136]. Dabei können Regelsätze wiederum andere Regelsätze, Bildschirmmasken, generierte Berichte oder 3GL-Module enthalten, sodass hieraus eine Hierarchie aufgebaut werden kann [15, S. 172]. 3GL-Module sind diejenigen Module, welche in einer Sprache der dritten Generation (third-generation language – 3GL) geschrieben sind, das heißt, wiederverwendbare Software [167, S. 32]. Wie bei der Funktionspunktanalyse werden die einzelnen Objekte nach ihrer Komplexität (einfach, mittel, schwer) gewichtet [167, S. 32]. Die Summe aus der Anzahl der gewichteten Objekte ergibt die Objektpunkte [89, S. 55]. Im Gegensatz zur Funktionspunktanalyse werden jedoch keine weiteren Einflussfaktoren berücksichtigt. Stattdessen wird der Grad der Wiederverwendung r berücksichtigt, sodass sich der Aufwand als angepasste Objektpunkte ergibt, welcher $(1-r)$ mal der Summe der Objektpunkte ist [167, S. 32]. Eine Weiterentwicklung sind die erweiterten Objektpunkte (Enhanced Object Points), welche eine Sammlung von mehrdimensionalen Kennzahlen sind, statt nur eine bestimmte Kennzahl zu verwenden wie bei den normalen Objektpunkten [216, S. 2 ff.]. Durch die unterschiedlichen Kennzahlen kann die Größe von verschiedenen Arten von Projekten bestimmt werden [216, S. 2]. Somit wäre es möglich, innerhalb dieses Ansatzes eine Sammlung von Kennzahlen für Hardwareentwürfe zu finden.

Eine andere, unabhängig voneinander entwickelte Methode der Objektpunkte wurde von Harry Sneed 1996 vorgestellt [208]. Sie basiert auf der Arbeit von Luiz Laranjeira aus dem Jahr 1990, welche die Objekthierarchie als Basis der Abschätzung der Größe eines Projekts verwendet [123, S. 515]. Dabei wird ein Objekt durch seinen Zustand, seine Methoden und seine Beziehungen zu anderen Objekten charakterisiert [123, S. 515]. Durch das Zählen der neuen Objekteigen-

schaften innerhalb der Hierarchie sowie der nicht funktionalen Anforderungen sollte es ermöglicht werden, die Anzahl an Codezeilen zu schätzen [123, S. 519]. Diesen Ansatz greift Harry Sneed auf, jedoch entwickelt er aus den Objektpunkten ein eigenes Maß für den Aufwand [208, S. 136]. Dabei erfasst er in drei Modellen die drei Dimensionen von Software: Objektmodell, Kommunikationsmodell und Prozessmodell [208, S. 136], wie Abbildung 3.8 zeigt.

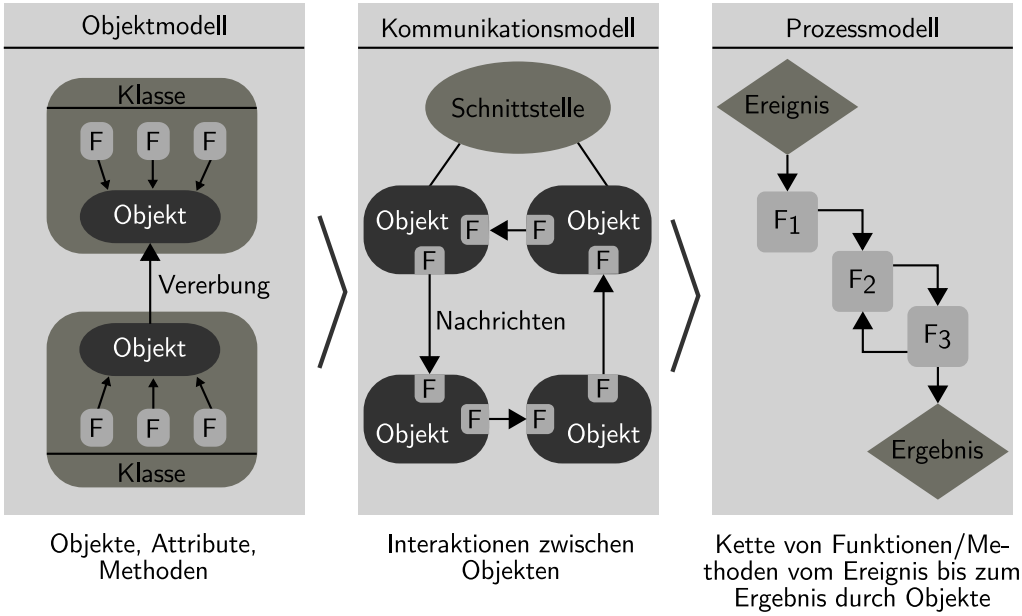


Abbildung 3.8: Quellen der Objektpunkte [nach 210, S. 294]

Aus jedem der Modelle kann jeweils eine Beziehungstabelle aufgestellt werden: eine Klassentabelle, eine Nachrichtentabelle und eine Prozesstabelle [208, S. 136 f.]. Dabei werden Klassenpunkte (Class-Points) berechnet, welche den Klassenentwicklungsaufwand wiedergeben und sich aus der Anzahl an Funktionen, Attributen, Relationen und der Wiederverwendungsrate ergeben; Nachrichtenpunkte (Message-Points), die den Integrationsaufwand wiedergeben und sich aus der Anzahl der Parameter, Quellen, Ziele, Komplexität und Wiederverwendungsrate ergeben; sowie Prozesspunkte (Process-Points) die den Systemtestaufwand wiedergeben und sich aus dem Prozesstyp, der Anzahl der Prozessvarianten und der Prozesskomplexität ergeben [208, S. 138]. Somit liegt ein ähnliches Vorgehen wie bei der Funktionspunktanalyse mit Produktivitätstabellen, Komplexitätsbewertungen (einfach, mittel, schwer) und Einflussfaktoren zugrunde [210, S. 293]. Das Aufwandsmaß der Objektpunkte ergibt sich als Summe der drei Einzelbewertungen (Klassen-, Nachrichten- und Prozesspunkte) [208, S. 138].

3.2.15 COCOMO 81

Eine der bekanntesten Methoden zur Bestimmung des Aufwands ist das „Constructive Cost Model“ (COCOMO, auch COCOMO 81) von Barry Boehm aus dem Jahr 1981 [25, S. 58 ff.]. Es gehört zur Klasse der parametrischen Methoden [34, S. 99].

Im einfachen Modell ergibt sich der Aufwand E in Personenmonaten aus der Anzahl ausgelieferter Programmbefehle L in Tausenden (kilo Delivered Source Instructions – kDSI) gemäß einer der Gleichungen aus Formel (3.5) in Abhängigkeit der Art (Komplexität) des Softwareprojekts [25, S. 57, 75]. Die Komplexitätseinstufung nach einfach (organic), mittel (semidetached) und schwer (embedded) richtet sich vorrangig nach der Projektgröße, der Teamzusammensetzung, dem Grad der Innovation sowie der Entwicklungsumgebung [25, S. 78 ff.].

$$\begin{aligned} \text{Einfach:} \quad & E = 2{,}4 \cdot L^{1{,}05} \\ \text{Mittel:} \quad & E = 3{,}0 \cdot L^{1{,}12} \\ \text{Schwer:} \quad & E = 3{,}6 \cdot L^{1{,}20} \end{aligned} \tag{3.5}$$

Die Multiplikatoren und Potenzen der vorherigen Formeln wurden durch eine Regressionsanalyse unter Berücksichtigung der Einflussparameter aus 63 Softwareprojektdaten gewonnen [25, S. 83 ff.]. Für die Anzahl ausgelieferter Programmbefehle werden grundsätzlich diejenigen Zeilen des Quellcodes gezählt, welche mit dem endgültigen Produkt ausgeliefert werden [25, S. 58 f.]. Sie enthalten jeglichen Code, der von den Projektmitarbeitenden geschaffen wurde und in irgendeiner Form weiterverarbeitet wurde [25, S. 59]. Nicht dazu zählen Kommentare und Code, welcher unverändert aus Bibliotheken übernommen wird [25, S. 59]. Eine Instruktion gilt als eine Zeile Code, auch wenn diese mehrere Befehle enthält [25, S. 59].

Beim erweiterten COCOMO-Verfahren (Intermediate COCOMO) werden fünfzehn weitere Kostentreiber berücksichtigt, welche die Produkteigenschaften, die Rechnereigenschaften, die Personaleigenschaften und die Projekteigenschaften abbilden [25, S. 115 f.]. Hieraus wird ein Faktor errechnet, welcher mit dem nach Formel (3.5) berechneten Aufwand in Personenmonaten multipliziert wird [25, S. 117]. Neben der Berechnung des Aufwands stellen alle COCOMO-Methoden auch Formeln zur Berechnung der (geplanten) Projektdauer bereit [25, S. 57, 124].

3.2.16 COCOMO II

Eine Erweiterung des ursprünglichen COCOMO 81 Modells ist das COCOMO II Modell aus dem Jahr 2000, welches den Neuerungen der Softwareentwicklung Rechnung trägt, wie beispielsweise der objektorientierten Programmierung oder der Planung anhand von Spiralmodellen [26, S. XXIX ff.]. Als Schätzgröße wird die Anzahlen an Quellcodezeilen (Source Lines of Codes – SLoC), an Objektpunkten (Object Points – OP) oder an nicht angepassten Funktionspunkten (Unadjusted Function Points – UFP) herangezogen [26, S. 12 ff, S. 192 ff.].

Der Modellansatz des Nach-Architekturentwurfs (Post-Architecture Model) setzt auf der Anzahl an Quellcodezeilen auf [26, S. 13 f.]. Als Quellcodezeilen werden diejenigen Codezeilen mitgezählt, welche neu geschaffen wurden sowie kopierter, modifizierter Code [26, S. 14]. Zur Abschätzung der Anzahl dienen Daten früherer Projekte und eine Checkliste zur Einstufung des Projekts [26, S. 14 ff.]. Ähnlich dem COCOMO 81 Verfahren werden zusätzlich Kostentreiber berücksichtigt, wobei das COCOMO II Verfahren um vier weitere Kostentreiber erweitert wurde [26, S. 41 ff.]. Die Berechnung des Aufwands P in Personenmonaten ergibt sich nach der Grundformel in Gleichung (3.6) aus den Konstanten A und B, welche durch eine Kalibrierung bestimmt werden müssen, der Größe des Projekts S in Tausenden Quellcodezeilen (kSLoC), n Aufwandsmultiplikatoren M_i und fünf Skalierungsfaktoren F [26, S. 13].

$$\begin{aligned} P &= A \cdot S^E \cdot \prod_{i=1}^{n} M_i \\ \text{mit} \qquad E &= B + 0.01 \cdot \sum_{j=1}^{5} F_j \end{aligned} \tag{3.6}$$

Das frühe Entwurfsmodell (Early Design Model) setzt auf unangepassten Funktionspunkten auf, welche bereits frühzeitig im Produktlebenszyklus verfügbar sind [26, S. 15]. Dabei werden von COCOMO II die Funktionspunkte aus der Funktionalität der Software und den individuellen Projektfaktoren nach den Kriterien von Charles Behrens [19], Edward Kunkler [119] und nach dem Handbuch der International Function Point Users Group [99] berechnet, welche auf der in Unterabschnitt 3.2.12 dargestellten Funktionspunktanalyse von Allan Albrecht beruhen [26, S. 15]. Anhand einer Tabelle können die unangepassten Funktionspunkte in Abhängigkeit von der verwendeten Programmiersprache in die Anzahl

Quellcodezeilen umgerechnet werden [26, S. 19 f.]. Zusammen mit den projektspezifischen Aufwandsmultiplikator- und Skalierungsfaktoren können diese in die Grundformel (3.6) eingesetzt werden [26, S. 29].

Das Modell der frühen Prototypenstufe (Application Composition Model) setzt auf den Objektpunkten von Rajiv Banker [14] auf [26, S. 193]. Diese eignen sich während früher Projektphasen besser als Funktionspunkte zur Aufwandsschätzung [26, S. 193]. Dabei wurden zwei Anpassungen vorgenommen: Zum einen wurden die Objektpunkte Anwendungspunkte (Application Points – AP) genannt, um Verwechslungen mit anderen Metriken zu vermeiden, die ebenfalls den Begriff Objektpunkte verwenden (siehe Unterabschnitt 3.2.14 zu den Objektpunkten) und zum anderen wurde eine Bewertungsskala für die Produktivität mit $\frac{\mathrm{AP}}{\mathrm{PM}}$ zur Umrechnung in den Aufwand in Personenmonaten eingeführt [26, S. 194].

Durch den Rückgriff der COCOMO-Modelle auf die ausgelieferten Programmbefehle beziehungsweise auf die Quellcodezeilen können diese auch grundsätzlich auf Hardwareentwürfe angewendet werden, welche durch eine Hardwarebeschreibungssprache spezifiziert wurden. Dabei ist es erforderlich, die jeweiligen Parameter und Konstanten durch die Analyse abgeschlossener Projekte zu bestimmen [so auch 34, S. 112]. Grundsätzlich ist dies auch für das frühe Entwurfsmodell mit dem Rückgriff auf Funktionspunkte und das frühe Prototypenstufenmodell mit den Anwendungspunkten möglich.

3.2.17 COSYSMO

Eine Variante des COCOMO II ist das Constructive Systems Engineering Cost Model (COSYSMO), welches für den Bereich der Systementwicklung (System Engineering) konzipiert wurde und sich auch für digitale Hardwareprojekte eignet [234, S. 5 f.]. Die beiden wichtigsten Unterschiede sind das Arbeiten auf der Systemebene (statt auf Basis von Quellcodezeilen) und die Verwendung spezieller Kostentreiber (Einflussgrößen) der Systementwicklung [234, S. 5 f.].

Für die parametrische Berechnung des Aufwands in Personenmonaten dient neben den Kalibrierungsdaten und Aufwandsmultiplikatoren (Einflussgrößen) die Projektgröße [235, S. 5]. Dabei wird die Projektgröße aus der gewichteten Summe der Anzahl der Systemanforderungen, der Hauptschnittstellen, der kritischen Algorithmen und der Einsatzszenarien gewonnen [234, S. 34 ff.], welche dann als Größe in Gleichung (3.6) des COCOMO II einfließt [234, S. 33].

3.2.18 Softwareentropie

Den Informationsgehalt von Software als Maß für die Komplexität zu verwenden geht auf die grundlegende Arbeit „An Entropy-Based Measure of Software Complexity“ [68] von Warren Harrison aus dem Jahr 1992 zurück. Ziel seiner Arbeit war die Definition einer Komplexitätsmetrik, welche die Fehleranzahl eines Programms vorhersagen kann [68, S. 1025].

In vorhergegangenen Arbeiten konnte bereits festgestellt werden, dass die „Schwierigkeit“ eines Programms [21, S. 779] [42, S. 1] und die Fehleranzahl [67, S. 84 ff.] mit dem Informationsgehalt eines Programms korrelieren [68, S. 1025]. Ursprünglich wurden die Häufigkeiten f_i aller vorkommenden Tokens (Operanden und Operatoren) $i = 1, \ldots, \eta$ und die Wahrscheinlichkeit ihres Auftretens p_i als Variablen für das Maß der Komplexität $M = -\sum_{i=1}^{\eta} f_i \log p_i$ verwendet [21, S. 778]. Dabei zeigte sich, dass die Entropie für große Programme mit vielen gleichen Tokens ähnlich ausfällt wie für kleine Programme mit vielen verschiedenen Tokens [42, S. 13]. In einer weiteren Vorarbeit wurde daher die Komplexität M durch die Instruktionsanzahl I dividiert [42, S. 13].

Aufbauend auf diesen Ansätzen nutzte Warren Harrison eine empirische Verteilung der Operatoren als Grundlage [68, S. 1025], da diese eine bestimmte Wahrscheinlichkeitsverteilung in Software aufweisen [67, S. 22 ff., 73 ff.]. Sein Ansatz beruht darauf, dass die Softwarekomplexität invers proportional zum durchschnittlichen Informationsgehalt der Operatoren ist [68, S. 1025]. Um den Zusammenhang zwischen Codegröße und Informationsgehalt aufzulösen, wird die Wahrscheinlichkeit p_i für das Auftreten des i-ten Operators als Prozentsatz der Anzahl seiner Verwendung f_i und der Gesamtanzahl der Operatoren N mit $p_i = \frac{f_i}{N}$ angegeben [68, S. 1025]. Hieraus ergibt sich die durchschnittliche Informationseinstufung I nach Formel (3.7) [68, S. 1026].

$$I = -\sum_{i=1}^{\eta} \frac{f_i}{N} \log_2 \frac{f_i}{N} \tag{3.7}$$

Der Fehlerbereich S (Error Span), das heißt, die Anzahl Quellcodezeilen, welche durchschnittlich zwischen dem Auftreten zweier Fehler liegt, berechnet sich nach Formel (3.8) mit den aus einer Regressionsanalyse zu bestimmenden Koeffizienten A bis F, der Anzahl einzelner Operatoren η, der Anzahl aller Operanden M, der

Anzahl aller Operatoren N und der Anzahl an Quellcodezeilen Q [68, S. 1026 f.]. Die Informationseinstufung I stellt nach Warren Harrison dabei lediglich eine Ordnung der Softwarekomplexität dar, sodass aus der Metrik kein Rückschluss auf das Verhältnis der Komplexität zwischen zwei Programmen gezogen werden kann [68, S. 1026].

$$\ln(S) = A + B \cdot I + C \cdot \eta_1 + D \cdot M + E \cdot N - F \cdot Q \tag{3.8}$$

Ausgehend von der Idee, die Informationsentropie für die Aufwandsschätzung in Softwareprojekten einzusetzen, wurde eine Methode entwickelt, welche die Softwarekomplexität auf Basis der Informationstheorie berechnet [111, S. 2094]. Die Softwarekomplexität beruht dabei nicht auf dem Ansatz von Warren Harrison, sondern wird durch den Wuli-Shili-Renli-Ansatz einbezogen [111, S. 2099]. Die drei Begriffe und die dahinter stehende Theorie bilden drei Komplexitätsdimensionen ab [111, S. 2101]. Der Wuli-Shili-Renli-Ansatz ist grundsätzlich ein philosophischer Ansatz aus Asien [66, S. 13] [249, S. 22]. Wuli bezeichnet die Gesetzmäßigkeiten der realen Umwelt, Shili die Sichtwiese sowie das Tun und Renli die Grundsätze der zugrunde liegenden menschlichen Interaktionen [66, S. 13].

Beim Entwurf von Software entsteht nach dem Verständnis des Autors die Komplexität aus der Kombination der drei Dimensionen, das heißt, die Entwicklerin oder der Entwickler muss sich der Gesetzmäßigkeiten bewusst sein, die entsprechende Sichtweise haben und entsprechende Fähigkeiten bei der menschlichen Interaktion aufweisen [111, S. 2094]. Um diese Dimensionen der Komplexität quantitativ ausdrücken zu können, wird die Hierarchie der Organisation (Projektgruppe) zugrunde gelegt und betrachtet, welche Personen über welche anderen Personen eine Kontrolle ausüben können sowie welche Informationen den einzelnen Teammitgliedern von anderen Teammitgliedern zur Verfügung gestellt werden [111, S. 2104 f.]. Für beide Faktoren ergibt sich jeweils ein Tupel, beispielsweise gibt $p(M_i, M_j)$ an, welche gewichtete Kontrolle das Mitglied M_i über das Gruppenmitglied M_j hat [111, S. 2104]. p wird dabei als Wahrscheinlichkeit aufgefasst und hieraus die Komplexität als Entropie durch die Formel $H = -p(M_i, M_j) \cdot \ln\, p(M_i, M_j)$ berechnet, wobei über alle Kombinationen von M_i und M_j summiert wird [111, S. 2104].

Richtigerweise haben die Teamzusammensetzung, die Hierarchien und die Kommunikation der Projektgruppe einen Einfluss auf das Projekt. So wird die (subjektive) Komplexität auch dadurch beeinflusst, dass beispielsweise bestimmte Informationen

fehlen. Jedoch kann die Komplexität der Software nicht allein durch diese Faktoren ausgedrückt werden. Auch ist es bei diesem Ansatz nicht nachvollziehbar, warum das Verhältnis der Gruppenmitglieder zueinander und deren Kommunikation als Wahrscheinlichkeiten in eine Entropieformel eingehen.

3.2.19 Hardwarekostenmodell

Auf Grundlage der (monetären) Kosten von Basiskomponenten wird ein Hardwarekostenmodell vorgestellt [156]. Als Basiskomponenten werden Gatter, Flipflops, Multiplexer, Tristatetreiber und Speicherzellen verwendet [157, S. 8] [156, S. 9]. Jeder dieser Komponenten werden dabei Kosten zugeordnet [157, S. 8] [156, S. 9]. Als Basis hierzu dient das Nicht-Gatter, dessen Kosten auf Eins normiert werden [157, S. 7]. Die Kosten der übrigen Basiskomponenten werden anhand der Bauteilekosten zweier Hersteller (Motorola und Venus) in Relation zur Normierung angegeben [156, S. 9]. Für einige Komponenten sind die Kosten des Modells in Tabelle 3.3 wiedergegeben. Die Kosten als Aufwand für den Entwurf eines kombinatorischen Schaltkreises werden nun als Summe der Kosten für die Basiskomponenten angegeben [156, S. 11].

Komponente	Kosten	Komponente	Kosten
Nicht-Gatter	1	Flipflop	8
(Nicht-)Und-Gatter/(Nicht-)Oder-Gatter	2	RAM-Zelle	2
Antivalenz-Gatter/Äquivalenz-Gatter	4	ROM-Zelle	0,25

Tabelle 3.3: Basiskomponentenkosten des Hardwarekostenmodell [aus 157, S. 8]

Somit bezieht sich der berechnete Aufwand allein auf die (finanziellen) Kosten für die Bauteile. Er bezieht sich nicht auf den Aufwand oder die Komplexität für den Entwurf einer Schaltung. Dies wird auch bei den Anwendungen deutlich, denn auch für wiederverwendete Schaltungen werden immer sämtliche Kosten für die Basiskomponenten mitberechnet. Somit gibt die Metrik lediglich die monetären Kosten für eine Schaltung an.

3.2.20 n-dim System

Die n-dim Gruppe [159] stellte einen Ansatz zur Unterstützung des Entwurfsprozesses vor [219, S. 1]. Dabei wird davon ausgegangen, dass jedes Mitglied eines Entwicklungsteams in einem sogenannten Informationsraum arbeitet, welcher durch Erfahrungswerte sowie die verfügbaren Informationen gekennzeichnet ist [48, S. 5, 8 ff.] [126, S. 4 ff.]. Anhand von Studien zur Arbeit von Entwicklerinnen und Entwicklern während des Entwurfsprozesses wurden Charakteristika eines Projekts abgeleitet, welche die Komplexität maßgeblich beeinflussen [184, S. 70 ff.]. Anhand dieser Charakteristika werden die Informationen als graphenbasierte, kanonische Darstellung innerhalb des n-dim Systems abgebildet und hieraus die Komplexität des Projekts abgeleitet [219, S. 5 ff.]. Insgesamt rückt der Ansatz die ausführenden Personen, die Entwickler und Entwicklerinnen, in den Vordergrund. Jedoch wird die Komplexität nur durch die Informationen bestimmt, welche diesen Personen zur Verfügung stehen. Dabei wird der eigentliche Entwurfsprozess darauf reduziert, dass, wenn alle Informationen bereits zur Verfügung stehen würden, das Projekt keine Komplexität hätte. Somit findet nur eine Verlagerung der Problematik der Bestimmung der Kennzahlen statt, denn nun müssen geeignete Kennzahlen zur Quantifizierung der „Informationskomplexität“ gefunden werden. Zudem wird die Frage, wie komplex ein Projekt ist, auf die Frage verlagert, welche Informationen für den (erfolgreichen) Abschluss erforderlich sind.

3.2.21 Metrics

Das Projekt Metrics befasst sich mit den Grundfragen des Verstehens, der Diagnose, der Optimierung und der Vorhersage von Systementwurfsprozessen [54, S. 705]. Hierzu sammelt die webbasierte Anwendung die Eigenschaften von Designartefakten, Entwurfsprozessen und die Kommunikation der eingesetzten Werkzeuge untereinander während der Systementwicklung, um diese Daten analysieren und mit vorhandenen Daten vergleichen zu können [54, S. 706 ff.]. Insgesamt werden etwa 60 verschiedene Daten gesammelt [54, S. 708]. Eine Erweiterung des Modells erlaubt die Sammlung von Daten über den Zeitverlauf hinweg und integriert Standardextraktionstechniken und Beurteilungen [115, S. 82 f.]. Dabei ist das grundsätzliche Ziel von Metrics, eine standardisierte Datenerfassungsschnittstelle für verschiedene Entwurfswerkzeuge zu bieten, um so den Entwurfsprozess zu optimieren [54, S. 710]. Jedoch lassen sich aus den Daten und der Anwendung zu jedem Entwicklungszeitpunkt auch Vorhersagen hinsichtlich der Ressourcen

(Schätzungen für Arbeitskraft, Zeit, Technologie, EDA-Lizenzen, die Wiederverwendung vorgefertigter Funktionsblöcke und so weiter) sowie konkrete Entscheidungen ableiten, ob ein Projekt weitergeführt werden soll oder nicht [54, S. 706]. Der Ansatz der verwendeten Metrik sowie der Prozess der Entscheidungsfindung wird in den Veröffentlichungen nicht vorgestellt.

3.2.22 CAD-Datensammlung

Es wurden mehrere Ansätze veröffentlicht, die auf Basis von CAD-Umgebungen (CAD-Frameworks) Daten aus dem Entwurfsprozess sammeln [30] [39] [106] [113] [220]. Eric Johnson hatte dabei die Möglichkeit untersucht, diese Datensammlung zur Verbesserung des Entwurfsprozesses zu verwenden [113, S. 4 f.]. Innerhalb seiner Arbeit wurde ein Prozessmodell zur Beschreibung des Konstruktionsprozesses entwickelt und hierauf basierend Techniken zur Sammlung von Prozessdaten vorgestellt [113, S. 41 ff.]. Die Prozessdaten werden in den logischen Entwurfsablauf integriert, wobei zur Analyse und Vorhersage der Projektabläufe Markovmodelle eingesetzt werden [113, S. 42 ff.]. Die Kalibrierung des Prozessmodells erfolgt über die gesammelten Prozessdaten [113, S. 64 ff.]. Anhand einer Reihe von entwickelten Techniken zur Analyse des Entwurfsprozesses wurden dann Experimente durchgeführt [113, S. 71 ff.]. Dabei zeigte sich, dass die Aussagen stark vom zu analysierenden Entwurf abhängen und die Methodik daher hauptsächlich geeignet ist, den Entwurfsprozess zu begleiten und Optimierungen während des Prozesses zu ermöglichen [113, S. 81 ff.]. Zudem wurde festgestellt, dass es notwendig ist, die Komplexität der Designartefakte zu berücksichtigen, was das bisherige Modell nicht leistet [113, S. 121]. Innerhalb des Ausblicks der Arbeit wird vorgeschlagen, die Komplexität der Designartefakte über die physikalischen Charakteristika zu berücksichtigen, insbesondere durch die Anzahl an Gattern oder die Anzahl an Eingängen [113, S. 121].

3.2.23 SOG-Modell

Das SOG-Modell soll die finale Anzahl Codezeilen eines VHDL-Projekts auf Basis einer Highlevelbeschreibung vorhersagen [55, S. 207]. Hierzu wird angenommen, dass ein Projekt, welches sich in einer frühen Phase der Entwicklung befindet, nur ein unvollständiges Projekt ist, das sich nach dem Top-down-Ansatz in einem Prozess der „Entwicklung durch Verfeinerung“ befindet [55, S. 208].

Für das Modell werden Syntaxobjekte definiert, welche grundsätzlich einer Komponente (Entity) einer VHDL-Implementierung entsprechen [55, S. 208]. Diese Syntaxobjekte werden in einem Syntaxobjektgraphen (Syntax Object Graph – SOG) hierarchisch geordnet und zusätzlich wird als Information die Vollständigkeit des jeweiligen Syntaxobjekts hinzugefügt [55, S. 208]. Der zweite Baustein des Modells sind die Bunde (Bunches), welche ein Syntaxobjekt bilden, das heißt, den Inhalt einer Entität [55, S. 208]. Die Bunde eines Syntaxobjekts werden nun nach (hierarchischen) Leveln geordnet [55, S. 209]. Anhand bereits bekannter Implementierungen und der Tiefe eines Bundes oder Syntaxobjekts wird die Anzahl finaler Quellcodezeilen hochgerechnet [55, S. 210]. Sind alle Syntaxobjekte und Bunde unbekannt, findet die Hochrechnung auf Basis externer Daten statt [55, S. 209]. Somit handelt es sich im Ergebnis um einen klassischen Bottom-up-Ansatz, welcher im Kern die Projektgröße nicht selbst bestimmt, sondern die Gesamtgröße über die einzelnen Objekte errechnet (siehe hierzu Unterabschnitt 3.1.1 über die Bottom-up-Methode als Vorgehensmodell). In einer ausführlichen Studie wurde gezeigt, dass der Ansatz eine zuverlässige Schätzung liefert, wenn ausreichend Daten von bereits abgeschlossenen Teilen vorliegen [195, S. 457 ff.]. Dieser Ansatz ist für die Entwurfsentropie interessant, da hierdurch eine strukturierte Anwendung in einer frühen Entwurfsphase ermöglicht werden kann.

3.2.24 Progress

Im Forschungsprojekt Progress wurde ein Modell zur Prognose von Entwicklungszeiten aus betriebswirtschaftlicher Sicht entwickelt [242, S. 3]. Die bessere Planbarkeit von Entwicklungsprojekten sollte durch eine genauere Prognose der Projektdauer ermöglicht werden und dadurch die Zeit bis zur Markteinführung halbiert werden [242, S. 23]. Im Vordergrund des Projekts standen die Beschleunigung der Entwicklung durch die Optimierung der Prozessgestaltung und die Entwicklung von Methoden hierfür [242, S. 24].

Innerhalb des Projekts wurde zunächst eine Erfolgsmessung anhand der betriebswirtschaftlichen Kriterien Zeit, Qualität, Kosten und Wirtschaftlichkeit vorgenommen [242, S. 24, 166 ff.]. Für das Vorhersagemodell wurde ein empirischer Ansatz gewählt, welcher die Einflussgrößen mit der Entwicklungszeit in Zusammenhang bringt [242, S. 277 f.]. Dabei beschränkt sich das Projekt auf die acht Haupteinflussgrößen, die sogenannten Zeittreiber: Planung, Organisation, Faktor Mensch, Technologie, Kunde/Kundin/Markt, Prozessgestaltung, Lieferantinnen/Lieferanten und Methodeneinsatz [242, S. 284 f.]. Diesen Zeittreibern liegen wiederum

50 Einflussgrößen zugrunde [242, S. 284 f.]. Für den Zeittreiber Technologie war das entscheidende Maß die Komplexität, die sich aus der Anzahl Variablen, der Vernetztheit, der Unklarheit und der Eigendynamik zusammensetzt [242, S. 289].

Für die minimale Entwicklungszeit wurde angenommen, dass die Einflussgrößen eine optimale Ausprägung haben müssen [242, S. 295]. Durch die Kalibrierung des Modells mit Referenzprojekten berechnet sich die Entwicklungszeit zukünftiger Projekte durch einen prozentualen Aufschlag anhand der einzelnen Einflussgrößen gegenüber den definierten Referenzprojekten, welche dadurch als Zeittreiber wirken [242, S. 300]. Die Einflussgrößen werden dabei durch ein Team von Experten und Expertinnen geschätzt [242, S. 303], somit auch die Komplexität.

Insgesamt ist das Projekt Progress stark durch eine betriebswirtschaftliche Sicht geprägt und stellt im Kern die Einflussgrößen bereit, welche durch Befragungen geschätzt werden müssen und in das Vorhersagemodell einfließen. Für die Wissenschaft sind daher besonders die identifizierten Zeittreiber von Interesse.

3.2.25 Modeling Metric Tool

Von MathWorks stammt das Modeling Metric Tool [88, S. 1], welches die Produktivität beim modellbasierten Entwurf bestimmt [207, S. 1 ff.]. Hierzu werden die momentanen Benutzerinnen- und Benutzereingaben (Maus- und Tastatureingaben) anhand von bereits aufgezeichneten Daten ausgewertet [87, S. 16 ff.]. Ein anderer Ansatz zur Bestimmung der Produktivität innerhalb von MATLAB/Simulink unter Berücksichtigung der Modularität des Entwurfs zeigt, dass die Anzahl Fehler mit der Komplexität eines Entwurfs korreliert [45, S. 101 ff.]. Genauere Daten und Informationen über die Verfahren sind nicht veröffentlicht.

3.2.26 μComplexity

μComplexity ist eine Methode, um den Entwurfsaufwand zu messen und eine Aufwandsschätzung in Personenmonaten vorzunehmen [16, S. 10]. Für die Entwicklung der Methode wurden zunächst die Entwicklerinnen und Entwickler von vier Prozessoren nach dem jeweiligen Entwurfsaufwand in Personenmonaten befragt [16, S. 14]. Im nächsten Schritt werden die fertigen Entwürfe der Prozessoren in einzelne Teile (zum Beispiel Speicher, Pipeline, Cache) aufgeteilt und mit verschiedenen

Metriken untersucht: Anzahl an Logikkegeln (logic cone – Logikgruppe, welche durch Register, primäre Ein-/Ausgänge oder Blackboxen begrenzt ist), Anzahl Quellcodezeilen, Anzahl HDL-Anweisungen, Anzahl Netze, Anzahl Standardzellen, Flächenverbrauch der Logikeinheiten, Flächenverbrauch der Speicherzellen, dynamische Leistungsaufnahme, statische Leistungsaufnahme sowie Anzahl Flipflops [16, S. 15]. Die so gewonnenen Daten wurden daraufhin mit dem tatsächlichen Aufwand verglichen, um so aus der Kombination mehrerer Metriken einen Entwurfsaufwandschätzer 1 (Design Effort Estimator 1 – DEE 1) abzuleiten [16, S. 16]. Der geschätzte Aufwand für die Kombination zweier Metriken e_{ij} ergab sich aus den Ergebnissen der einzelnen Metriken m_i und m_j mit deren jeweiligen Koeffizienten (Gewichtung) w_i und w_j sowie einem Produktivitätsfaktor p gemäß Formel (3.9) [16, S. 12].

$$e_{ij} = \frac{1}{p} \cdot ((w_i \cdot m_i) + (w_j \cdot m_j)) \tag{3.9}$$

Dabei zeigte sich, dass die Verwendung der Anzahl an Logikkegeln und der Anzahl an HDL-Anweisungen die geringste Standardabweichung für den Vergleich zwischen dem mit DEE 1 geschätzten und dem tatsächlichen Aufwand ergab [16, S. 16]. In welchem Verhältnis die Linearkombination der beiden Methoden in den DEE 1 eingeflossen ist, wird nicht angegeben. Auch der Produktivitätsfaktor p wird nicht angegeben. Es wird nur beschrieben, dass der Produktivitätsfaktor durch eine Regressionsanalyse, bei welcher die Standardabweichung minimiert wurde, berechnet wurde [16, S. 17]. Mit anderen Metriken wurde der Ansatz auch auf die Aufwandsschätzung bei der Leiterplattenentwicklung (printed circuit board design – PCB design) angewendet, wobei hier eine Kombination aus der Anzahl an Komponenten, Komponentendichte und Pindichte als optimal angesehen wurde [17, S. 264 f.]. Insgesamt stellt somit μComplexity keine eigenständige Metrik dar, sondern nur eine Verfahrensweise, wie andere Metriken gemeinsam einsetzt werden können. Dass für Schätzungen grundsätzlich mehrere Metriken und Verfahren genutzt werden sollten, um die Ergebnisse abzusichern, gilt mangels einer wirklich überzeugenden Metrik als Basis für jede Aufwandsschätzung.

3.2.27 Produktiv+

Das Forschungsprojekt Produktiv+ hatte die Modellierung, quantitative Messung und vorausschauende Abschätzung der Entwurfsproduktivität in der Halbleiterindustrie zum Gegenstand [29, S. 2]. Hierzu wurde der Ansatz verfolgt, dass eine (bessere) Planbarkeit von IC-Entwicklungen nur dann gegeben ist, wenn die Produktivität bekannt ist [13, S. 934]. Somit ist der Kern des Projekts die Entwicklung eines Produktivitätsmodells [29, S. 5]. Die Produktivität ist dabei der Quotient aus Ergebnis (Output) und Faktoreinsatz (Input), welcher üblicherweise nicht linear ist [29, S. 6]. Wie in Abbildung 3.9 beispielhaft gezeigt, führt die Erhöhung des Einsatzes nicht zu einer proportionalen Erhöhung des Ergebnisses (Output).

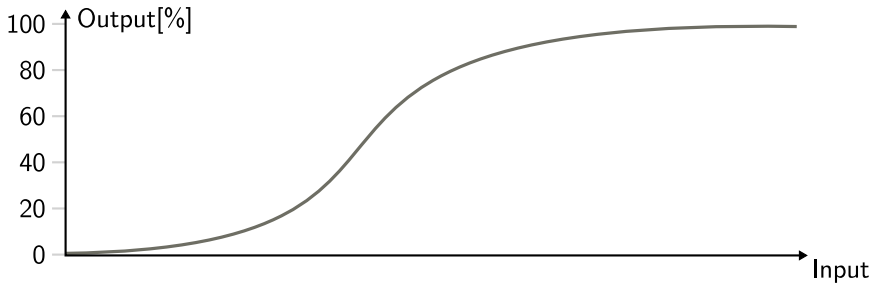


Abbildung 3.9: Beispiel einer Produktivitätskennlinie [nach 29, S. 7]

Die Bestimmung des Inputs findet grundsätzlich in Anlehnung an vorhandene Konzepte der Fertigungstechnik statt und betrachtet Personal, Zeit und Sachmittel [29, S. 7]. Für die Erfassung des Outputs sowie der Einflussfaktoren, welche die Produktivitätskennlinie eines Designsystems prägen, wurde ein Datenmodell und eine Systemarchitektur zur automatisierten Erfassung und Analyse der Daten entworfen [248, S. 3]. Die automatisierte Datenerfassung umfasst dabei sechs Charakteristika eines Entwurfsprojekts: monetäre Rahmenbedingungen, Spezifikation der Entwurfsartefakte, Entwurfsfluss, Ressourcen, Personal und Zuverlässigkeit [82, S. 328].

Innerhalb des Produktivitätsmodells wurde ein allgemeingültiges Modell des Designsystems entwickelt und die Produktivität als Sammlung von Maßzahlen aufgefasst [29, S. 19]. Die Eigenschaften der Modellelemente, mit welchen die Produktivität beschrieben wird, umfasst folgende Bereiche, welche durch Ontologien beschrieben werden [29, S. 22]:

- Performance und Produktivität
- Komplexität
- Qualifikation und Erfahrung
- Qualität
- Messung und Messbarkeit
- Designobjekt
- Designwert

Das Designsystem selbst besteht aus mehreren Komponenten, welche bei der Entwicklung als Designartefakte in einem Designprozess zusammenwirken [212, S. 448]. Der Wert der Ausbringungsmenge (Output) bestimmt sich durch die Einflüsse des Designsystems und wird als Designwert (Design Value) angegeben [71, S. 670]. Er bestimmt sich durch die Komplexität, Qualität und den Abstraktionsgrad [29, S. 29]. Die Komplexität hängt dabei nur von objektiven, designspezifischen Parametern ab und ist für ein Design konstant [74, S. 60]. Sie ist unabhängig von dem oder der Betrachtenden, dem Prozess, der Software und den Ressourcen [74, S. 60]. Die Komplexität ist somit projekt-, firmen- und zeitunabhängig [125, S. 2068] und hängt nur von der Fläche, der Funktionalität, dem Zeitverhalten, der Leistung, der Energie, der Entwurfstechnologie und den Interaktionen ab [74, S. 61].

Für das Simulationsverfahren zur quantitativen Produktivitätsprognose wurden zunächst Data-Mining-Verfahren eingesetzt um die umfangreichen, aufgezeichneten Daten hinsichtlich der Korrelation mit betriebswirtschaftlichen Kennzahlen zu untersuchen [29, S. 35 f.]. Da eine Projektvorhersage mit algorithmischen Methoden unvorhergesehene Ereignisse und die Erfahrung der Mitarbeitenden, welche zu unterschiedlichen Bearbeitungszeiten führen, nicht berücksichtigen kann, wurde ein Simulationsansatz gewählt [211, S. 4]. Dieser basiert auf einem Request-Service-Modell [73, S. 194]. Innerhalb des Request-Service-Modells bieten verschiedene Ressourcen bestimmte Dienstleistungen (Services) an, wie die Arbeitsleistung eines Mitarbeitenden oder die Rechenzeit eines Computers [72, S. 2]. Diese Dienstleistungen werden von den einzelnen Aufgaben beziehungsweise Aktivitäten gegen ein Entgelt angefordert (Request), um ihre Inputfaktoren (Komplexität, Qualität und Abstraktionsgrad) wertschöpfend zu transformieren [70, S. 35]. Innerhalb der Simulation werden die Dauer der Aktivitäten und die Qualität des Outputs simuliert [73, S. 194].

Für die Simulation müssen der komplette Verlauf und alle betrachteten Eigenschaften des Designprozesses ermittelt werden und als Funktionen in den Modellen dargestellt werden [29, S. 36] [211, S. 4 f.]. Hieraus ergibt sich auch die Möglich-

keit, Maßnahmen zur Outputsteigerung zu entwickeln und betriebswirtschaftlich relevante Kennzahlen für Projektentscheidungen zu gewinnen [80, S. 47 ff.] [81, S. 279 ff.] [83, S. 1062 ff.] [84, S. 117 ff.] sowie die Untersuchung der Auswirkungen, wenn sich der Entwurfsablauf ändert [118, S. 70 ff.].

Nach Angaben des Projekts sollten die Ergebnisse in einen ProjectNavigator der Firma Cadence Design Systems [36] überführt werden [29, S. 52]. Bei den aktuell angebotenen Werkzeugen der Firma ist jedoch kein solches Werkzeug (Tool) aufgelistet [35]. Eine Nachfrage bei Cadence ergab, dass die Entwicklung des ProjectNavigator eingestellt wurde und auch keine weiteren Pläne in die Richtung existieren.

3.2.28 ACM

Das sehr einfache Maß Apparatus Complexity Measure (ACM) zum Messen der Hardwarekomplexität wurde 2014 veröffentlicht [51, S. 2]. Dabei bestimmt sich der Aufwand A aus dem Betrag der Anzahl Elemente E auf einer Entwurfsebene: $A = |E|$ [51, S. 2]. Es wird postuliert, dass sich durch die steigende Elementanzahl auf den niedrigeren Entwurfsebenen der Entwurfsaufwand erhöht [51, S. 2]. Der Ansatz wird anhand der Anzahl an Transistoren in Prozessoren im Zeitraum von 1971 bis 2012 evaluiert [51, S. 3 f.]. Dabei konnte festgestellt werden, dass die Apparatus Complexity mit der Komplexität des Mooreschen Gesetzes übereinstimmt [51, S. 3] und basierend auf diesem Ergebnis wurde eine Prognose für die Zukunft vorgenommen [51, S. 5]. Es ist wenig überraschend, dass eine Metrik, welche die Anzahl Transistoren zählt, mit dem Mooreschen Gesetz übereinstimmt, welches ebenfalls die Transistoranzahl zugrunde legt (siehe hierzu Abschnitt 2.1).

3.2.29 PRICE H

Das Unternehmen PRICE System (Parametric Review of Information for Costing and Evaluation – PRICE) stellt eine ganze Familie von Anwendungen im Bereich der Aufwandsschätzung bereit, wie in Abbildung 3.10 dargestellt [174, S. 2]. Diese Anwendungen bilden den Rahmen für unterschiedliche Aufwandsschätzmodelle in verschiedenen Bereichen, wie beispielsweise PRICE S für Software oder PRICE H für Hardware [172, S. 3]. Die Methoden gehören zur Klasse der parametrischen Methoden [34, S. 100]. Die PRICE Anwendungen umfassen folgende Teile:

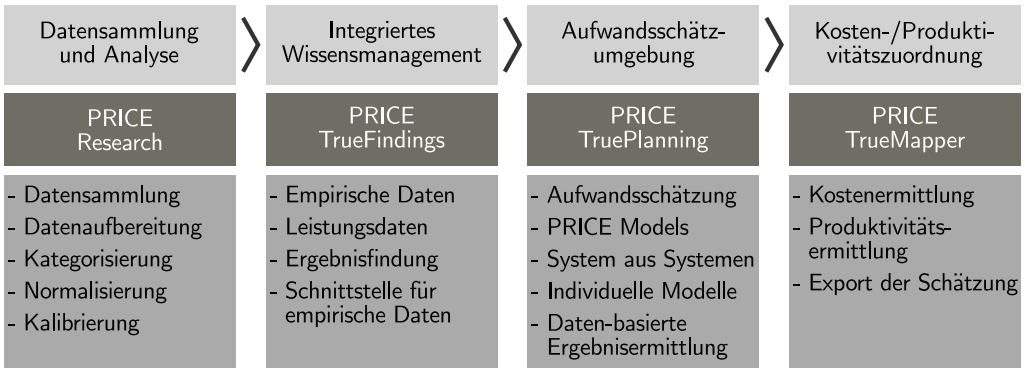


Abbildung 3.10: PRICE Anwendungen zur Aufwandsschätzung [nach 174, S. 1]

- PRICE Research: Die von PRICE gesammelten Daten über abgeschlossene Projekte werden von unternehmenseigenen Analystinnen und Analysten analysiert, Abweichungen gefiltert sowie Muster und Ausnahmen identifiziert [174, S. 1]. Die Daten werden normalisiert und kategorisiert, sodass eine einheitliche Datenbasis entsteht [174, S. 1]. Diese Daten bilden die Grundlage für die PRICE Kostenmodelle sowie für individuelle Kostenmodelle beziehungsweise Aufwandsmodelle von Kundinnen und Kunden [175, S. 2].

- PRICE TrueFindings: Dieses Modul unterstützt die Anwenderinnen und Anwender darin, die Daten abgeschlossener Projekte auszuwerten und aufzubereiten, um hiermit zukünftige Projekte zu schätzen [171, S. 1]. Elementarer Bestandteil ist eine Datenbank mit empirischen Daten, einschließlich der Kosten, Kostentreiber und Leistungsmerkmale [171, S. 1]. Hierzu werden die vorhandenen Daten aufbereitet, kategorisiert, normalisiert, kalibriert und nach neuen Kostentreibern analysiert [171, S. 1]. Somit bildet dieses Modul die Schnittstelle zwischen den empirischen Daten und der Schätzung neuer Projekte innerhalb der Aufwandsschätzumgebung [171, S. 1].

- PRICE TruePlanning: Dieses Modul stellt die Umgebung zur Aufwandsschätzung dar und enthält verschiedene Kostenmodelle [173, S. 1]. Dabei werden auch Kernkostentreiber verwendet, wie Projektgröße und Komplexität sowie Daten aus vorherigen Projekten, welche durch die oben genannten Teile gesammelt und aufbereitet wurden [173, S. 1, 3].

- PRICE TrueMapper: Die im Vorherigen gewonnenen Aufwandsschätzungen können innerhalb dieses Moduls organisiert, präsentiert und verglichen werden [176, S. 1]. Hierbei besteht die Möglichkeit, den Aufwand in Kos-

ten oder als Produktivität auszudrücken, sodass mehrere Schätzergebnisse zusammengefügt werden können oder eine Schätzung über andere Kosten charakterisiert werden kann [176, S. 1].

Kern der Aufwandsschätzung von PRICE sind die mehr als 40 Aufwandsmodelle (PRICE Cost Models), welche auf Daten von mehr als 21.000 Projekten beruhen [172, S. 3]. Für den Bereich der Hardware stehen die Modelle PRICE H für Hardware und PRICE HL für den Hardwarelebenszyklus [177] sowie Modelle für spezifische Hardwareprojekte, wie den militärischen Bereich [178] oder die Raumfahrt [179], zur Verfügung.

PRICE H setzt auf der parametrischen Methode True H auf, welche eine Methodik bereitstellt, die Erfahrungen aus scheinbar unzusammenhängenden Projekten für die Kostenschätzung korrelieren [170, S. 11, 13]. Basis hierzu sind mehrere hundert Gleichungen, welche die Eingabeparameter mit den Kosten in Verbindung setzen [170, S. 14]. Dabei wird zwischen Entwicklungs- und Produktionskostentreibern unterschieden [170, S. 25], wobei im Folgenden nur die Kostentreibergruppen der Entwicklung vorgestellt werden [170, S. 26 ff.]:

- Anzahl höherer Hierarchie-/Partitionierungsstufen und zusätzlicher Prototypen
- Betriebsspezifikation
- Gewicht der Struktur und Gewicht der Elektronik
- Fertigungsaufwand für den Aufbau
- Prozent der neuen Struktur (der Hardwarekomponente)
- Prozent der veränderten Strukturbestandteile (der eingekauften Standardkomponenten)
- Prozent der Mehrfachverwendung an Strukturbestandteilen
- Prozent neuer Elektronik (der Hardwarekomponente)
- Prozent der veränderten Elektronikbestandteile (der eingekauften Standardkomponenten)
- Prozent der Mehrfachverwendung an Elektronikbestandteilen
- Entwicklungskomplexität
- Einsatz neuer Technologie und Jahr der Technik

Aufgrund dieser quantitativen und qualitativen Daten kann auf Basis des algorithmischen Zusammenhangs der Aufwand eines Projekts abgeschätzt werden [34, S. 100]. Dabei bilden das Gewicht und der Technologieindex den Kern für die Hardwareabschätzung [203, S. 6].

3.2.30 Numetrics

Ähnlich wie PRICE Systems bot auch das Unternehmen Numetrics Systems Management eine Planungssoftware für den Hardwareentwurf an. Mittlerweile wurde die Firma von McKinsey Solution, einem Teil von McKinsey & Company, aufgekauft [138] [139]. Diese bietet einen IC-Projektplaner an, dessen Kern ein empirisches Modell ist, um die Komplexität, das heißt, die Schwierigkeit beim Entwurf eines Chips, zu berechnen [137]. Das Komplexitätsmodell wird mithilfe einer Datenbank aus Benchmarkprojekten kalibriert [140]. Die Datenbank enthält mehr als 2.000 Benchmarks von mehr als 75 Halbleiterunternehmen und 34 Anwendungsgebieten [136].

Kern des ursprünglichen Komplexitätsmodells von Numetrics waren verschiedene Normalisierungsalgorithmen, welche die Erfahrungen aus scheinbar unzusammenhängenden Projekten nutzten, um den Aufwand eines Projekts mit der Anzahl Transistoren zu korrelieren [162, S. 4]. Durch die Betrachtung von Einflussfaktoren, welche das jeweilige Projekt charakterisieren, wie Schaltungstyp, zeitliche Randbedingungen, Höhe der Wiederverwendung, Architektur oder Firmware, konnten normalisierte Transistoren (Normalized Transistors) als Komplexitätseinheit (Complexity Unit) berechnet werden [162, S. 5 f.]. Hieraus ließ sich der Aufwand beziehungsweise die Produktivität (normalisierte Transistoren pro Personenwoche) ableiten [162, S. 6]. Nicht bekannt ist, ob und inwieweit dieses Modell in den IC-Projektplaner des Unternehmens McKinsey Solution aufgenommen wurde. McKinsey Solution selbst gibt keinerlei Informationen über die tatsächliche Funktionsweise und die Berechnungen des IC-Projektplaners an. Mehrere Nachfragen, ob und inwieweit das ursprüngliche Modell von Numetrics übernommen wurde, blieben unbeantwortet.

3.2.31 Synopsys

Auch das Unternehmen Synopsys stellt einen Ansatz zur Messung der Entwurfsproduktivität vor [213, S. 2]. Dabei werden Daten aus den fünf Hauptschritten des Chipentwurfs (Synthese, Design for Testability, Floorplanning, Place and Route und Fertigungsfreigabe) in Zusammenhang mit der jeweiligen Projektphase in einer Datenbank gespeichert [213, S. 5, 7]. Hierzu wurde ein RTL-to-GDSII Tool entwickelt, welches die Daten aus der Register-Transfer-Ebene (Register-Transfer-Level – RTL) unter Berücksichtigung von vorgefertigten Funktionsblöcken in das

grafische Datensystem II Format (Graphic Data System II – GDSII) umwandelt [169, S. 4 ff.] [189, S. 2 ff.]. Aus diesen Daten wird ein Komplexitätsfaktor berechnet, welcher sich aus den Daten von fünf Hauptkomponenten zusammensetzt: Größe des Designs (in Instanzen), Verfahrenstechnik, Zieltaktfrequenz, Quellen der vorgefertigten Funktionsblöcke und Anwendungseigenschaften [213, S. 8].

Zusammen mit den Daten aus abgeschlossenen Projekten kann eine normalisierte Komplexität für den Entwurf angegeben werden [213, S. 8 f.], ähnlich wie bei den normalisierten Transistoren von Numetrics. Auch bei dieser kommerziellen Anwendung sind keine Informationen verfügbar, wie sich aus den Hauptkomponenten der Komplexitätsfaktor im Detail berechnet.

3.2.32 SEER-H

Das SEER-H Modell (System Evaluation and Estimation of Resources Hardware Model) des Unternehmens Galorath wird verwendet, um Hardware, Elektronik und Systementwicklungen zu planen [60]. Die Kostenschätzfunktionen basieren auf einer umfangreichen Datenbank aus abgeschlossenen Projekten, Verhaltensmodellen und anderen internen Metriken [61, S. 1]. Für das parametrische Schätzverfahren werden primär das Gewicht, das Volumen und die Zuordnung der Elemente zur SEER-Datenbank verwendet [114, S. 4] [236, S. 5]. Die genaue Funktionsweise ist dabei nicht bekannt.

3.3 Fazit der Literaturrecherche

Insgesamt zeigt die Darstellung der einzelnen Methoden die zu Beginn dieses Kapitels genannten Gemeinsamkeiten auf: Sie benötigen für die Berechnung oder zumindest die Kalibrierung ihrer Metriken (umfangreiche) empirische Daten bereits abgeschlossener Projekte. Zudem ist die Berücksichtigung von Einflussfaktoren zentraler Bestandteil aller Methoden und insbesondere die Komplexität spielt regelmäßig eine wichtige Rolle. Jedoch beschränken sich die meisten Metriken bei der Messung der produktspezifischen Einflussgrößen auf die Quantität, das heißt, auf die Größe eines Entwurfs (Anzahl an Quellcodezeilen, Funktions- oder Objektpunkte). Die Komplexität fließt entweder als projektspezifische Entwicklungsgröße, als qualitative Wertigkeit oder über empirische Daten ein.

Einzig das Projekt Produktiv+ ging der Frage nach, ob und inwieweit es erforderlich ist, andere oder neue produktspezifische Einflussfaktoren hinsichtlich der Komplexität zu berücksichtigen. Hierzu wurde der Ansatz einer Produktivitätsmethode mit einer Simulation verbunden. Im Endeffekt musste jedoch die Messung der Komplexität durch ein aufwendiges Datenerhebungsverfahren und deren Verarbeitung kalibriert werden, sodass wiederum eine Metrik entstand, welche stark von empirischen Faktoren bestimmt ist. Diese gehen zwar über eine Simulation ein, bilden aber weiterhin den Kern der Metrik und somit der Komplexitätsbestimmung.

Der in der vorliegenden Arbeit entwickelte Ansatz folgt zunächst dem Ansatzpunkt des Produktiv+ Projekts: Die Komplexität für einen Entwurf ist konstant und von subjektiven Komplexitätsfaktoren (wie der Erfahrung oder der Verfügbarkeit von Entwurfswerkzeugen) zu trennen [74, S. 60] [80, S. 11] [125, S. 2068]. Dieser Ansatz wird in der vorliegenden Arbeit dahin gehend weiterverfolgt, dass die Komplexität in einem Entwurf selbst zu finden sein muss.

Vorteil des vorliegenden Ansatzes ist die Trennung der produktspezifischen Einflussgrößen (Quantität und Komplexität) von projektspezifischen Einflussgrößen. Somit kann die Metrik der Entwurfsentropie als Einflussgröße für etablierte Metriken verwendet werden. Denn auch mit der Bestimmung der Quantität und Komplexität eines Entwurfs werden für eine Aufwandsbestimmung weiterhin projektspezifische Einflussgrößen eine zentrale Bedeutung spielen. Durch die Entwurfsentropie wird jedoch nicht nur der Projektumfang, sondern auch die Komplexität eines Entwurfs als produktspezifische Einflussfaktoren direkt berücksichtigt.

Dabei benötigt die Entwurfsentropie keine umfangreichen empirischen Daten. Somit ist es möglich, die Entwurfsentropie als Maß auch dann einzusetzen, wenn keine Daten abgeschlossener Projekte zur Verfügung stehen. Dies spielt insbesondere bei kleineren Unternehmungen, beim Einsatz neuer Technologien, bei Vorhaben, welche einem starken Technologiewandel unterliegen oder bei einem geringen Budget eine entscheidende Rolle. Denn in diesen Fällen ist es vielfach nicht möglich, einen so umfangreichen Datenbestand anzulegen, dass dieser als Referenz für den aktuell zu bewertenden Entwurf dienen kann.

Durch das Erfordernis des umfangreichen Datenbestands der meisten Metriken sind kommerzielle Schätzwerkzeuge sehr verbreitet. Diese stellen die erforderlichen Daten den Anwenderinnen und Anwendern bereit. In der Regel erfolgt die Datensammlung hierfür dadurch, dass diese nach Abschluss ihrer Projekte ihre Daten den kommerziellen Anbietern und Anbieterinnen zur Verfügung stellen. Jedoch sind weder die Metriken noch die verwendeten Methoden bekannt, mit welchen der Aufwand abgeschätzt wird. Doch auch ein Großteil der nicht-kommerziellen Metriken benötigt umfangreiche empirische Daten.

Das Erfordernis des umfangreichen Datenbestands für die Berechnung und/oder die Kalibrierung der jeweiligen Metrik erlaubt es nur schwer, einen Vergleich mit diesen Metriken zu ziehen. Zudem ist eine direkte Anwendung auf einzelne Entwürfe nicht möglich. Hierfür wäre eine Datenbasis von vielen abgeschlossenen Projekten erforderlich, welche im Rahmen eines einzelnen universitären Forschungsprojekts nicht erhoben werden können. In diesem Zusammenhang bietet die Entwurfsentropie eine Lösung an, welche keine Daten von vorherigen Projekten benötigt und sofort auf einen Entwurf angewendet werden kann.

Um eine frühzeitige Abschätzung der Entwurfsentropie vornehmen zu können, bietet das SOG-Modell einen interessanten Ansatz [55, S. 207 ff.]. Hierauf basierend könnte, bei Kenntnis der Entwurfsentropie einiger Teilkomponenten, die Entwurfsentropie des gesamten Designs mit einer Art der Prozentsatzmethode abgeschätzt werden. Da zunächst die Grundlagen der Entwurfsentropie in den Forschungen zu der vorliegenden Arbeit erarbeitet wurden, kann aufbauend auf diesen in weiterführende Arbeiten die frühzeitige Abschätzung untersucht werden.

Um die Entwurfsentropie zu berechnen, wird die Kommunikation zwischen den Komponenten betrachtet. Dieser Ansatz basiert auf der Informationstheorie, welche sich mit dem Austausch von Informationen beschäftigt und deren wichtigste Arbeiten und Grundlagen im folgenden Abschnitt vorgestellt werden.

3.4 Informationstheorie

Die Informationstheorie bietet eine exakte mathematische Beschreibung von Informationen, deren Codierung, Übertragung und Speicherung [27, S. 2] [197, S. 9]. Informationen werden hierdurch eine messbare Größe [27, S. 14] und können als beseitigte Unbestimmtheit aufgefasst werden [197, S. 12]. Basis der Informationstheorie ist die Wahrscheinlichkeitsrechnung [27, S. 2] [197, S. 9, 13]. Die Elemente der Kommunikation und Speicherung von Informationen werden allgemein durch das abstrakte mathematische Modell der Informationstheorie beschrieben [27, S. 23 f.], welches in Abbildung 3.11 dargestellt ist. Dabei wird von der physikalischen Realisierung abstrahiert und Gegenstand wird die Übertragung von Informationen [197, S. 10].

Abbildung 3.11: Modell der Informationstheorie (nach [27, S. 23] und [197, S. 10])

Die (diskrete) Quelle ist der Ausgangspunkt von digitalisierten Symbolen [27, S. 23]. Wobei auch analoge Ausgangssignale eine Quelle sein können, welche dann zunächst digitalisiert werden müssen [112, S. 12]. Die Symbole der Quelle werden im Codierer verarbeitet, das heißt, es findet vor der Übertragung eine Signalverarbeitung statt, wie beispielsweise Kompression, Erzeugung von Prüfwerten oder Modulation [112, S. 12]. Der Kanal ist eine abstrakte Beschreibung der Übertragung von Symbolen durch bedingte Wahrscheinlichkeiten [27, S. 23]. Der Decodierer hat die Aufgabe, Fehler zu erkennen und diese, wenn möglich, zu korrigieren sowie die Daten wieder in Symbole für die Senke umzuwandeln [27, S. 24].

Die ersten Studien der Informationstheorie gehen auf die Arbeiten „Certain Factors Affecting Telegraph Speed“ von Harry Nyquist aus dem Jahr 1924 [163] und „Transmission of Information“ von Ralph Hartley aus dem Jahr 1928 [69] zurück [32, S. 26] [79, S. 1] [131]. Harry Nyquist erkannte, dass Kommunikationskanäle ein maximales Datenübertragungsverhältnis W haben, welches von der Anzahl an Übertragungssymbolen m abhängt und gab diese gemäß Gleichung (3.10) für einen störungsfreien Kanal an [163, S. 333].

$$W = \log m \tag{3.10}$$

Ralph Hartley stellte wenige Jahre später die ersten mathematischen Grundlagen der Informationstheorie auf [131] und gab die Informationsmenge einer Nachricht H in Abhängigkeit von der Anzahl an Zeichen der Nachricht n und der Anzahl verfügbarer Zeichen s gemäß Gleichung (3.11) an [69, S. 540]. Dabei ging er davon aus, dass die verfügbaren Zeichen mit der gleichen Wahrscheinlichkeit auftreten [69, S. 541].

$$H = n \cdot \log s = \log s^n \tag{3.11}$$

Als Geburtsstunde der modernen Informationstheorie gilt die Arbeit „A Mathematical Theory of Communication“ von Claude Shannon aus dem Jahr 1948 [202], welcher realisierte, dass die Kommunikationssignale getrennt von der zu übermittelnden Nachricht behandelt werden müssen [131]. Hierdurch änderte sich die Sicht auf die Nachrichtentechnik von einer physikalischen hin zu einer mathematischen Sichtweise [27, S. 13]. Im Gegensatz zur Arbeit von Ralph Hartley nahm Claude Shannon nicht mehr eine Gleichverteilung an, sondern erkannte, dass die Symbole bestimmte Auftrittswahrscheinlichkeiten haben [202, S. 392 ff.]. Zudem wurden die Auswirkungen von Störungen auf den Kanal miteinbezogen [202, S. 406 ff.].

Als Maß für die Unsicherheit gab Claude Shannon Gleichung (3.12) an und nannte das Maß Entropie, in Anlehnung an das Boltzmannsche H-Theorem [202, S. 393 f.]. Dabei ist die Entropie einer gedächtnislosen Quelle $H(X)$ mit den möglichen Symbolen $X = \{x_1, \ldots, x_N\}$ und deren Auftrittswahrscheinlichkeiten $p(x_i)$ für $i = 1, \ldots, N$ mit $\sum_{i=1}^{N} p(x_i) = 1$ durch Gleichung (3.12) gegeben [197, S. 15] [202, S. 393] [240, S. 3 ff.]. $I = \log \frac{1}{p(x)}$ wird dabei als Selbstinformation I eines Zeichens x bezeichnet [28, S. 23] [85, S. 9 f.].

$$H(X) = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) = \sum_{i=1}^{N} p(x_i) \cdot \log \frac{1}{p(x_i)} \tag{3.12}$$

Eine gedächtnislose Quelle, welche aus dem Alphabet $X = \{1, 0\}$ mit den Wahrscheinlichkeiten $p(0) = p$ und $p(1) = 1 - p$ besteht, wird als Binärquelle bezeichnet, für welche die Entropie $H(X) = -p \cdot \log(p) - (1 - p) \cdot \log(1 - p)$ ist [27, S. 41]. Die Verwendung der logarithmischen Funktion in den Gleichungen stellt die natürlichste Wahl („most natural choice“ [202, S. 379]) dar, wie es Claude Shannon in Bezug auf die Arbeit von Ralph Hartley anführte [202, S. 379], der durch die Wahl der Logarithmusbasis eine Konstante weglassen konnte [69, S. 540]. Als

Beispiel, dass der Logarithmus die natürlichste Wahl ist, führt er die Codierung einer Lochkarte an [202, S. 380]. Hat diese N Möglichkeiten für (k)ein Loch, so können 2^N verschiedene Nachrichten auf einer Karte „gespeichert“ werden [240, S. 1]. Bei zwei Lochkarten ist zu erwarten, dass diese die doppelte Kapazität einer Karte haben [202, S. 380]. Daher „drängt“ [240, S. 1] sich die Verwendung des Logarithmus auf: Für eine Lochkarte gilt $\log\left(2^N\right) = N \cdot \log(2)$ und für zwei Lochkarten gilt $\log\left(2^{2 \cdot N}\right) = 2 \cdot N \cdot \log(2)$ [240, S. 1].

Innerhalb der Rechnertechnik und Datenverarbeitung wird grundsätzlich die Einheit der Informationsmenge als Bit bezeichnet [197, S. 18]. Dabei ist der Entscheidungsgehalt zweier unabhängiger, gleichwahrscheinlicher Ereignisse einer Quelle als $H_0 = 1 \frac{\text{bit}}{\text{Ereignis}}$ festgelegt [197, S. 18]. Innerhalb der vorliegenden Arbeit wird das Maß der Entwurfsentropie als dimensionslos angegeben. Das heißt, die Werte werden ohne Einheiten angegeben, da das Maß zur Messung der Komplexität und Quantität eines Systems dient und nicht für informationstechnische Berechnungen, wie der Kanalkapazität oder der Quellcodierung, verwendet wird. Das auf der Informationstheorie basierende Modell der Entwurfsentropie sowie die Formeln zu deren Berechnung werden im folgenden Kapitel vorgestellt.

4 Die Entwurfsentropie

Die Entwurfsentropie basiert auf dem Modell, dass eine digitale Schaltung ein System aus Komponenten ist, die miteinander Informationen beziehungsweise Nachrichten austauschen. Daher werden im folgenden Abschnitt zunächst das Modell der Entwurfsentropie und der Zusammenhang mit dem Nachrichtenaustausch der Informationstheorie vorgestellt. Anschließend werden die verwendeten Begriffe definiert und die Formeln zur Berechnung der Entwurfsentropie hergeleitet. Der Bezug zur digitalen Hardware wird in den darauf folgenden Abschnitten hergestellt und das für die automatisierte Analyse entwickelte Werkzeug vorgestellt. Die beiden letzten Abschnitte umfassen Exkurse über die Zustandsreduktion und die Analyse von Softwareprogrammen.

Im Zusammenhang mit der vorliegenden Dissertation sind acht Publikationen entstanden, welche innerhalb dieses und der folgenden Kapitel referenziert sind. Dabei weisen die eckigen Doppelklammern [[X]] darauf hin, dass es sich bei den referenzierten Arbeiten um Veröffentlichungen des Autors der vorliegenden Arbeit handelt.

4.1 Modell der Entwurfsentropie

Die Entwurfsentropie verfolgt den Ansatz, dass beim Entwurf digitaler Schaltungen die Kommunikation zwischen den Komponenten die Komplexität eines Systems bestimmt. Durch die Anzahl an Kommunikationsverbindungen, welche zwischen den Komponenten entworfen werden, fließt neben der Komplexität auch die Entwurfsgröße in das Maß ein. In Abhängigkeit von den übertragenen Nachrichten fallen die Kommunikationsverbindungen unterschiedlich aus, sodass statt der Verbindungen die übertragenen Nachrichten betrachtet werden. Abbildung 4.1 illustriert den Nachrichtenaustausch zwischen zwei Komponenten als Ansatz der Entwurfsentropie [[148]]. Dabei stellt die linke Komponente die Nachrichtenquelle

dar, welche die Verbindung mit der rechten Komponente nutzt, um Nachrichten zu übertragen. Das heißt, die Verbindung ist der Kanal der Nachrichtenübertragung. Die rechte Komponente ist die Nachrichtensenke, welche die übertragenen Nachrichten empfängt.

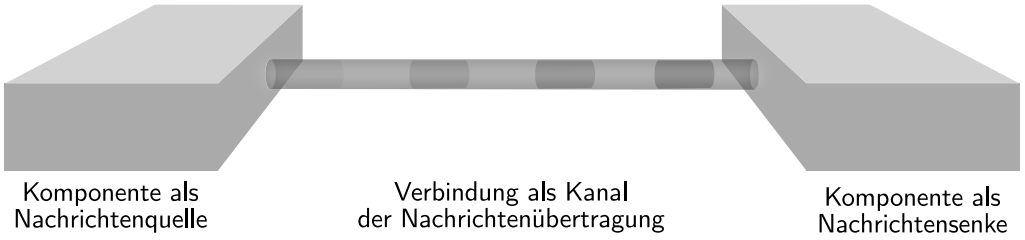


Abbildung 4.1: Nachrichtenübertragung zwischen zwei Komponenten [[145]]

Der Informationsgehalt einer Nachricht ist durch die Entropie gegeben [183, S. 320] und Informationen sind durch die Informationstheorie eine messbare Größe [27, S. 14]. Die Nachricht selbst kann auf unterschiedliche Arten übertragen werden. In der digitalen Hardware findet die Übertragung insbesondere über elektrische Signale zwischen den Komponenten statt. Das bedeutet, dass über eine elektrische Leitung die Nachricht mithilfe eines elektrischen Signals von der Quelle zur Senke gesendet wird. Diese Übertragung wird beispielsweise durch die Zuweisung eines Signals b zu einem anderen Signal a realisiert (a <= b) [[144]]. Beim Programmieren von Software werden dagegen nicht elektrische Signale entworfen, welche beispielsweise in einem Prozessor verarbeitet werden, sondern es werden Nachrichten auf Basis von Variablen, Funktionen und so weiter miteinander in Bezug gesetzt. Somit erfolgt die Nachrichtenübertragung beispielsweise durch die Zuweisung des Werts einer Variablen b zu einer anderen Variablen a (a = b) [[144]].

Die Komponenten eines Systems verarbeiten die empfangenen Nachrichten und können selbst Nachrichten an andere Komponenten senden. Zum Beispiel stellt im Hardwarebereich eine Und-Verknüpfung (x <= a AND b) eine Komponente dar. Die Und-Verknüpfung hat als Eingaben die Zustände der Signale a und b. Die Informationen werden von der Komponente verarbeitet und die verarbeitete Information als Signal am Ausgang x ausgegeben. Im Softwarebereich stellt beispielsweise eine Bedingung (a == b) eine Komponente dar, welche die Informationen der beiden Eingangsvariablen a und b verarbeitet und als Ausgabe die Information liefert, ob die Bedingung wahr oder falsch ist.

Zunächst wird nur eine einzelne Komponente betrachtet. Komponenten haben die Eigenschaft, Informationen in irgendeiner Form zu verarbeiten. So stellt die eben

genannte Und-Verknüpfung in Form eines Und-Gatters eine solche Komponente dar. Sie verarbeitet die Informationen der Eingänge und gibt das Ergebnis als logische Eins oder Null aus. Neben der Betrachtung als Einzelkomponenten werden diese vorerst auch als abgeschlossene Einheiten betrachtet. Das heißt, es ist zunächst unerheblich, wie eine Komponente intern aufgebaut ist. Für das Und-Gatter ist es somit nicht entscheidend, ob es intern beispielsweise aus Nicht-Und-Gattern aufgebaut ist, als CMOS-Schaltung realisiert wurde oder mithilfe von Relais auf einer Platine zusammengesetzt ist. Vorerst ist allein das (logische) Verhalten einer Komponente entscheidend.

Wird aus dem Und-Gatter eine einfache Schaltung aufgebaut, ist es erforderlich, das Verhalten des Gatters zu kennen. Das Und-Gatter gibt genau dann eine logische Eins aus, wenn an beiden Eingängen eine logische Eins anliegt. In allen anderen Fällen wird eine logische Null ausgegeben. Für den Entwurf dieser Schaltung ist es nur erforderlich das Verhalten zu kennen, das heißt, welche Nachrichten eine Komponente als Eingaben benötigt und basierend auf deren Verarbeitung, welche Nachrichten (Ereignisse) eine Komponente als Ausgaben liefert [[148]]. Dabei stellt der Ausgang einer Komponente selbst eine Informationsquelle dar.

Um den mittleren Informationsgehalt einer Nachrichtenquelle zu berechnen wird auf die Informationstheorie zurückgegriffen. Die Selbstinformation $I = -\log(p(x))$ eines Ereignisses x ist durch seine Auftrittswahrscheinlichkeit $p(x)$ gegeben [28, S. 23] [85, S. 9 f.]. Als Maß für den mittleren Informationsgehalt dient die Informationsentropie, welche auch als Maß für die beseitigte Unbestimmtheit bezeichnet wird [197, S. 12]. Das heißt, wenn bekannt ist, dass bestimmte Ereignisse wahrscheinlicher als andere Ereignisse eintreten, dann ist die Entropie geringer, als wenn keine Kenntnisse vorliegen. Als Maß für die Unsicherheit gab Claude Shannon die Gleichung (4.1) der folgenden Definition 1 an, welche er in Anlehnung an das Boltzmannsche H-Theorem Entropie nannte [202, S. 393 f.].

Definition 1 (Informationsentropie)

Die Informationsentropie $H(Q)$ einer diskreten Quelle Q mit den Ereignissen $X = \{x_1, \ldots, x_N\}$ und den zugehörigen Auftrittswahrscheinlichkeiten $p(x_i)$ mit $i = 1, \ldots, N$ und $\sum_{i=1}^{N} p(x_i) = 1$ ist gegeben durch [197, S. 15] [202, S. 393] [240, S. 3 ff.]:

$$H(Q) = \sum_{i=1}^{N} p(x_i) \cdot \log \frac{1}{p(x_i)} = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) \tag{4.1}$$

Innerhalb der Informationstheorie wird der Logarithmus zur Basis zwei (Logarithmus Dualis) verwendet und die Einheit für die Informationsmenge als Bit bezeichnet [28, S. 23] [197, S. 18]. Die Entwurfsentropie ist dagegen dimensionslos und verwendet den Logarithmus (log) zur Basis zehn (Dekadischer Logarithmus). Hierdurch wird verdeutlicht, dass die Entwurfsentropie ein Maß für die Komplexität und Quantität ist und nicht als nachrichtentechnisches Maß mit dem Logarithmus Dualis oder als naturwissenschaftliches Maß mit dem Logarithmus Naturalis verwendet wird. Für eine einheitliche Darstellung wird daher bereits im Folgenden der Logarithmus (log) als Zehnerlogarithmus verwendet und die Entropie dimensionslos angegeben.

Beim vorherigen Anwendungsfall des Und-Gatters kann der Ausgang zwei mögliche Zustände annehmen: Es kann eine logische Eins oder eine logische Null ausgegeben werden. Das heißt, die Ereignismenge $X = \{0, 1\}$ besteht aus zwei (binären) Elementen. Die Wahrscheinlichkeiten für das Auftreten der beiden komplementären Ereignisse können mit $p(0) = p$ und $p(1) = 1 - p$ angegeben werden [27, S. 41]. Werden die Wahrscheinlichkeiten in Formel (4.1) eingesetzt, ergibt sich die binäre Informationsentropie der folgenden Definition 2 [27, S. 41] [197, S. 18].

Definition 2 (Binäre Informationsentropie)

Die binäre Informationsentropie $H(Q)$ einer diskreten Quelle Q mit zwei Ereignissen $X = \{x_1, x_2\}$ und den zugehörigen Auftrittswahrscheinlichkeiten $p(x_1) = p$ und $p(x_2) = (1 - p)$ mit $0 \leq p \leq 1$ ist gegeben durch:

$$H(Q) = -p \cdot \log p - (1 - p) \cdot \log (1 - p) \tag{4.2}$$

Die binäre Informationsentropie $H(Q)$ einer diskreten Quelle Q hängt nur von der Wahrscheinlichkeit p ab und ist als Funktion $H(Q)$ für $0 \leq p \leq 1$ in Abbildung 4.2 gezeigt. Da die Wahrscheinlichkeit, dass das Ereignis x_1 eintritt, p ist, folgt, dass die Eintrittswahrscheinlichkeit des Ereignisses x_2 gleich $(1 - p)$ ist. Je geringer die Wahrscheinlichkeit des Eintritts eines bestimmten Ereignisses ist, desto niedriger ist die Informationsentropie. Oder anders formuliert, es besteht eine maximale Unsicherheit, welches der beiden Ereignisse eintritt, genau dann, wenn die Wahrscheinlichkeiten für den Eintritt der Ereignisse gleich sind. Das heißt, die Informationsentropie ist dann maximal, wenn die Wahrscheinlichkeiten, dass eines der beiden möglichen Ereignisse eintritt, gleichverteilt sind. Für eine Binärquelle mit $N = 2$ möglichen Ereignissen zeigt Abbildung 4.2, dass die Informationsentropie maximal ist, wenn $p = p(x_1) = p(x_2) = 0{,}5$ gilt.

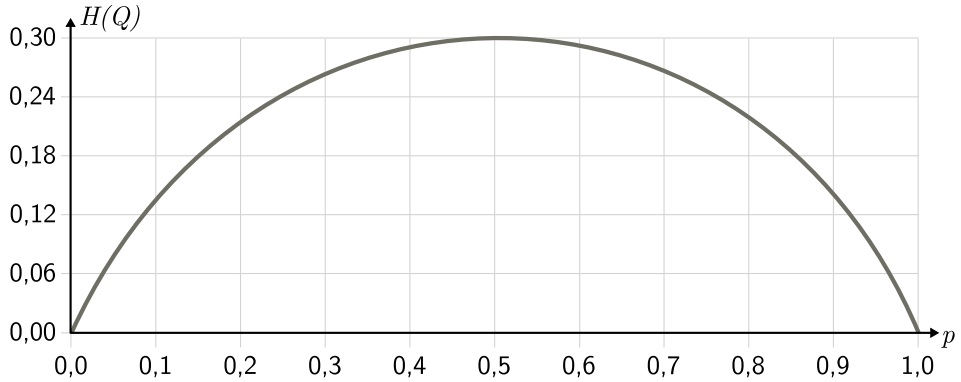


Abbildung 4.2: Binäre Informationsentropie (Zehnerlogarithmus) [[143]]

Um für eine beliebige Anzahl $N \geq 1$ an Ereignissen x_i mit $i = 1, \ldots, N$ die maximale Informationsentropie angeben zu können, wird Formel (4.1) partiell nach den Wahrscheinlichkeiten abgeleitet, um den Extremwert zu finden. Hierfür wird der Lagrangesche Multiplikator λ verwendet [angelehnt an 197, S. 16], sodass sich aus Gleichung (4.1) die folgende Hilfsfunktion F mit der Nebenbedingung $\sum_{i=1}^{N} p_i = 1$ ergibt:

$$\begin{aligned} F &= -\sum_{i=1}^{N} p_i \cdot \log p_i - \lambda \left(\sum_{i=1}^{N} p_i - 1 \right) \\ &= -\sum_{i=1}^{N} \left(p_i \cdot \log p_i + \lambda \cdot p_i \right) + \lambda \end{aligned} \tag{4.3}$$

Zur Bestimmung des Extremwerts wird die partielle Ableitung von Gleichung (4.3) nach p_i mit $i = 1, \ldots, N$ gebildet, welche am Extremwert null sein muss:

$$\frac{\partial F}{\partial p_i} = -\log p_i - p_i \frac{1}{p_i \cdot \log_e 10} - \lambda \stackrel{!}{=} 0 \tag{4.4}$$

Dabei ist e die Eulersche Zahl. Mit der Basisumrechnung des Logarithmus $\log_b x = \frac{\log_a x}{log_a b}$ kann Gleichung (4.4) nach nach $\log p_i$ aufgelöst werden:

$$\log p_i \stackrel{!}{=} \log e - \lambda \tag{4.5}$$

Da e und λ nicht von i abhängen und somit konstant sein müssen, kann die Bedingung in Gleichung (4.5) nur dann erfüllt werden, wenn p_i für alle $i = 1, \ldots, N$

ebenfalls konstant ist. Aus der Nebenbedingung $\sum_{i=1}^{N} p_i = 1$ folgt, dass der Extremwert bei $p_i = \frac{1}{N}$ existiert. Da die zweite partielle Ableitung negativ ist, liegt ein Maximum vor. Wird $p_i = \frac{1}{N}$ in Gleichung (4.1) für die Informationsentropie eingesetzt, ergibt sich die maximale Informationsentropie:

$$\begin{aligned} H_{\max}(Q) &= -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) = -\sum_{i=1}^{N} \frac{1}{N} \cdot \log \frac{1}{N} \\ &= -N \cdot \frac{1}{N} \cdot \log N^{-1} = \log N \end{aligned} \tag{4.6}$$

Somit ist für eine beliebige Anzahl $N \geq 1$ an Ereignissen x_i mit $i = 1, \ldots, N$ die Informationsentropie maximal, wenn die Eintrittswahrscheinlichkeiten $p(x_i) = \frac{1}{N}$ gleichverteilt sind. Aus Gleichung (4.6) ergibt sich die folgende Definition 3 für die maximale Informationsentropie. Diese entspricht der Hartley-Entropie aus Gleichung (3.11), welche in Abschnitt 3.4 vorgestellt wurde.

Definition 3 (Maximale Informationsentropie)

Die Informationsentropie $H(Q)$ aus Definition 1 ist maximal, wenn die Ereignisse $X = \{x_1, \ldots, x_N\}$ die gleichen Auftrittswahrscheinlichkeiten $p(x_i) = \frac{1}{N}$ mit $i = 1, \ldots, N$ und $\sum_{i=1}^{N} p(x_i) = 1$ haben. Der Maximalwert $H_{\max}(Q)$ der Informationsentropie einer Quelle Q ist dann gegeben als:

$$H_{\max}(Q) = \log N \tag{4.7}$$

Wird die Entropie als beseitigte Unbestimmtheit aufgefasst [197, S. 12], dann ist mit der Information, dass bestimmte Ereignisse wahrscheinlicher sind als andere Ereignisse, die Entropie geringer. Mit dieser Information können Kanäle oder Quellcodierungen entworfen werden, welche optimal sind. Liegen dagegen keine Informationen über die Wahrscheinlichkeiten vor, besteht eine maximale Unsicherheit mithin also eine maximale Entropie. Aus entwurfstechnischer Sicht ist es jedoch unerheblich, mit welcher Wahrscheinlichkeit bestimmte Ereignisse eintreten. In diesem Zusammenhang wird keine optimale Quellcodierung oder eine optimale Kanalkapazität gesucht. Das Maß der Entropie wird zur Beschreibung der Komplexität und der Entwurfsgröße verwendet. Unabhängig davon, wie unwahrscheinlich ein bestimmter Zustand ist, trägt dieser zum Entwurfsaufwand bei, wenn er beim Entwurf berücksichtigt wurde [[141]]. Nur Zustände, welche nicht erreicht werden können oder Schnittstellen, welche nur einen bestimmen Zustand haben, tragen nicht zur Komplexität einer Schaltung bei [[141]].

Somit ist keines der möglichen Ereignisse zu bevorzugen, da alle Ereignisse gleichermaßen zur Komplexität beitragen. Als Kriterium dient dabei das Indifferenzprinzip, welches auch als das Prinzip vom mangelnden zureichenden Grunde bezeichnet wird [38, S. 3]: „Wenn keine Gründe dafür bekannt sind, um eines von verschiedenen möglichen Ereignissen zu begünstigen, dann sind die Ereignisse als gleichwahrscheinlich anzusehen“ [38, S. 3]. Dass die Gleichverteilung für die Berechnung der Entwurfsentropie naheliegend ist, wird anhand der folgenden Beispiele verdeutlicht.

Gegeben ist eine Komponente, welche das binäre Ausgabealphabet $X = \{0, 1\}$ mit den Wahrscheinlichkeiten $p = p(0)$ und $p(1) = (1 - p)$ hat. Ist $p = 0{,}5$ dann ist gemäß Gleichung (4.2) die Entropie maximal:

$$\begin{aligned} H &= -p \cdot \log(p) - (1-p) \cdot \log(1-p) \\ &= -0{,}5 \cdot \log(0{,}5) - 0{,}5 \cdot \log(0{,}5) = -\log(0{,}5) \approx 0{,}30 \end{aligned}$$

Liegt dagegen für den Fall $p = 0{,}25$ und somit $(1-p) = 0{,}75$ keine Gleichverteilung vor, dann ist die Informationsentropie geringer:

$$\begin{aligned} H &= -p \cdot \log(p) - (1-p) \cdot \log(1-p) \\ &= -0{,}25 \cdot \log(0{,}25) - 0{,}75 \cdot \log(0{,}75) \approx 0{,}24 \end{aligned}$$

Wird die Informationsentropie mit den tatsächlichen Wahrscheinlichkeiten für das Auftreten eines Zustands als Maß für die Komplexität zugrunde gelegt, dann sind Komponentenausgänge mit nicht gleichverteilten Wahrscheinlichkeiten weniger komplex als Ausgänge mit gleichen Wahrscheinlichkeiten für die Ereignisse. Bei digitalen Schaltungen kann dies im Folgenden anhand einer Und-Verknüpfung und einer Antivalenz-Verknüpfung verdeutlicht werden.

Für die Schaltfunktion $y = a \cdot b$ einer Und-Verknüpfung beziehungsweise für ein Und-Gatter in Hardware gilt $y = 1$ für $a = b = 1$ und $y = 0$ für andere Kombinationen von a und b. Bei einer Gleichverteilung der Eingangszustände $p_{a/b}(0) = p_{a/b}(1) = 0{,}5$ für die statistisch unabhängigen Eingänge a und b ist die Wahrscheinlichkeit, dass eine Null am Ausgang y anliegt, $p_y(0) = \frac{3}{4}$ und, dass eine Eins anliegt, $p_y(1) = (1 - p_y(0)) = \frac{1}{4}$. Somit ist nach der vorherigen Rechnung mit $p = 0{,}25$ die Informationsentropie $H(\text{Und}) \approx 0{,}24$. Dagegen ist bei einer Antivalenz-Verknüpfung $y = a \oplus b$ beziehungsweise bei einem Antivalenz-

Gatter (XOR) der Ausgang $y = 0$, wenn $a = b$ gilt und $y = 1$ bei $a \neq b$. Wenn die Eingangszustände gleich wahrscheinlich sind, dann sind auch die Ausgangszustände gleich wahrscheinlich, das heißt, $p_y(0) = p_y(1) = 0{,}5$. Somit ist nach der vorherigen Rechnung mit $p = 0{,}5$ die Informationsentropie $H(\text{XOR}) \approx 0{,}30$.

Da die Entropie als Maß für die Komplexität eines Systems verwendet wird, würde die Berechnung mit den tatsächlichen Wahrscheinlichkeiten dazu führen, dass eine Antivalenz-Verknüpfung komplexer ist als eine Und-Verknüpfung. Dabei ist es jedoch für den Entwurf und die darin enthaltene Komplexität unerheblich, um welche der beiden Verknüpfungen beziehungsweise Gatter es sich handelt. Beide haben objektiv eine gleich komplexe Wahrheitstabelle beziehungsweise gleich komplexe Verbindungen. Wird eine Gleichverteilung der Ausgangswahrscheinlichkeiten angenommen, dann sind beide Verknüpfungen beziehungsweise Gatter gleich komplex. Dabei wird die Verknüpfung beziehungsweise das Gatter nur von außen als abgeschlossene Einheit betrachtet und nicht der konkrete Aufbau. Das heißt, es wird die Komplexität der Verdrahtung einer Komponente angegeben und nicht, wie der innere Aufbau der Komponente ist. Für die Berechnung des inneren Aufbaus einer Komponente wird die Strukturentropie verwendet, welche im folgenden Abschnitt vorgestellt wird.

Insbesondere bei der Verbindung mehrerer Bauteile beziehungsweise bei der Verbindung mehrerer Schaltfunktionen wird deutlich, dass selbst die gleiche Komponente eine unterschiedliche Komplexität hätte, wenn keine Gleichverteilung angenommen wird. Dienen die Ausgänge zweier Und-Verknüpfungen als Eingänge einer weiteren Und-Verknüpfung, dann ist die Wahrscheinlichkeit, dass an der weiteren Und-Verknüpfung eine logische Eins am Ausgang anliegt, $\frac{1}{4} \cdot \frac{1}{4} = \frac{1}{16}$, wenn die Eingangszustände der beiden ersten Und-Verknüpfungen gleichwahrscheinlich sind. Der Aufbau mit drei Und-Gattern ist in Abbildung 4.3 zur Verdeutlichung dargestellt.

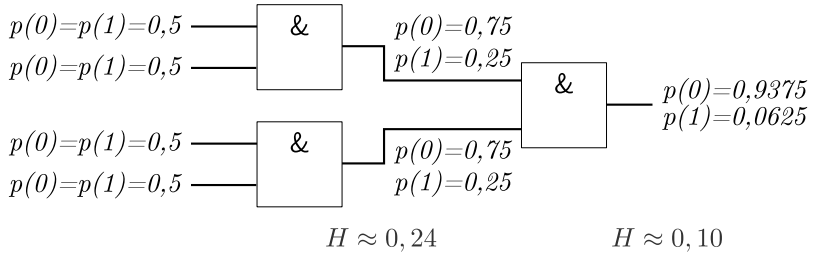


Abbildung 4.3: Entropien dreier Und-Gatter

Wird die Wahrscheinlichkeit von $p = \frac{1}{16}$ in Formel (4.2) der binären Informationsentropie eingesetzt, dann beträgt diese $H \approx 0{,}10$. Somit haben die Ausgänge der ersten beiden Und-Gatter eine Informationsentropie von jeweils $H \approx 0{,}24$ und die des weiteren Und-Gatters, dessen Eingänge mit den Ausgängen der ersten beiden Gatter verbunden sind, eine Informationsentropie von $H \approx 0{,}10$. Aus informationstheoretischer Sicht ist dieses Ergebnis zutreffend, jedoch führt es zu einem unbefriedigenden Ergebnis für die Berechnung der Entwurfskomplexität. Werden die Wahrscheinlichkeiten als gleichverteilt und somit als maximal angenommen, dann ist die Komplexität aller drei Und-Verknüpfungen gleich, während gleichzeitig die informationstheoretische Berechnung mit der maximalen Informationsentropie zutreffend bleibt.

Dass die Betrachtung der tatsächlichen Wahrscheinlichkeiten zu einem unbefriedigenden Ergebnis führt, wird auch anhand einer Bedingung, wie diese beispielsweise in Software oder bei Verhaltensbeschreibungen in Hardware verwendet wird, deutlich. Die Bedingung kann entweder „wahr“ oder „falsch“ sein. Ist die Wahrscheinlichkeit für den Eintritt eines der beiden Ereignisse sehr gering, ergibt sich eine sehr geringe Informationsentropie. Für eine Wahrscheinlichkeit von $p(0) = 10^{-3}$ beispielsweise beträgt die Informationsentropie $H \approx 0{,}00043$ nach Gleichung (4.2). Jedoch ist es für den Entwurf unerheblich, dass die Eintrittswahrscheinlichkeit des einen Ereignisses sehr gering ist. Denn die Komplexität und der in der Schaltung beziehungsweise dem Programm enthaltene Aufwand zum Entwurf der Bedingung ist der gleiche, unabhängig davon mit welcher Wahrscheinlichkeit der Falsch- oder der Wahr-Fall eintritt.

Die vorherige Darstellung zeigt, dass das Modell der Entwurfsentropie auf Basis der Informationstheorie nur dann eine Aussage über die Komplexität einer Komponente treffen kann, wenn die Auftrittswahrscheinlichkeiten der möglichen Zustände eines Ausgangs als gleich wahrscheinlich angenommen werden [[143]]. Dadurch wird die maximale Entropie eines Ausgangs berücksichtigt, die dann vorliegt, wenn für N mögliche Zustände $p(x_i) = \frac{1}{N}$ mit $i = 1, \ldots, N$ gilt [[143]].

Bis jetzt wurde für eine Komponente nur ein Ausgang betrachtet. Jedoch können Komponenten auch mehrere Ausgänge besitzen. Die Informationsentropien von mehreren statistisch unabhängigen Ausgängen sind additiv, wie die folgende Definition 4 besagt [27, S. 44]. Dies ist gleichbedeutend mit der Entropie mehrerer unabhängiger Quellen [27, S. 44].

Definition 4 (Informationsentropie mehrerer Ausgänge)
Die Informationsentropie $H(Q)$ einer Quelle Q mit M unabhängigen Ausgängen m_k mit $k = 1, \ldots, M$ ist die Summe der Einzelentropien $H(Q_k)$ der Ausgänge.

$$H(Q) = \sum_{k=1}^{M} H(Q_k) \tag{4.8}$$

Mithilfe der vorherigen Definition kann die Entropie einer Quelle mit mehreren Ausgängen berechnet werden. Dabei besitzt eine Komponente, wie beispielsweise das Und-Gatter, regelmäßig auch Eingänge. Für die Entwurfsentropie der Eingänge wird zunächst der Kanal, das heißt, die Verbindung zwischen zwei Komponenten, betrachtet. Informationstheoretisch ist der Kanal die Verbindung zwischen Sender und Empfänger der Nachrichten [197, S. 75]. Als Modell liegt das Bergersche Kanalmodell zugrunde [197, S. 77 f.], welches in Abbildung 4.4 dargestellt ist. In diesem Modell wird ein diskreter Kanal betrachtet, das heißt, es wird auf dem Kanal nur eine endliche Anzahl diskreter Signalzustände unterschieden [197, S. 77].

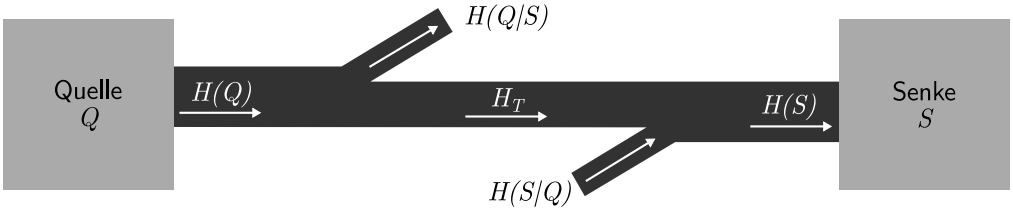


Abbildung 4.4: Bergersches Kanalmodell [nach 197, S. 78]

In der Abbildung hat die Quelle Q eine Quellenentropie von $H(Q)$ [197, S. 78 f.]. Bei der Übertragung können Informationen verloren gehen [197, S. 79]. Dieser Anteil wird durch die bedingte Entropie $H(Q|S)$ beschrieben [197, S. 79]. Die Differenz ist die übertragene Nutzinformation, die als Transinformation H_T bezeichnet wird [197, S. 79]. Neben dem Verlust von Informationen können auch Störungen bei der Übertragung auftreten, welche durch die Störentropie $H(S|Q)$ beschrieben werden [197, S. 79]. Die Summe aus Transinformation und Störentropie bildet die Senkenentropie $H(S)$. Für die Senke (Empfänger) S stellt der Kanal eine Nachrichtenquelle dar [197, S. 78]. Dabei entspricht bei einem verlustfreien ungestörten Kanal die Quellenentropie der Senkenentropie, das heißt, dass die bedingten Entropien $H(Q|S)$ und $H(S|Q)$ null sind, was durch die folgende Definition 5 für den idealen Kanal ausgedrückt wird [197, S. 79 f.].

Definition 5 (Idealer Kanal)

Bei einem idealen Kanal, das heißt, bei einem verlustfreien ungestörten Kanal entspricht die Entropie der Quelle $H(Q)$ sowohl der Transinformation H_T als auch der Entropie am Kanalausgang $H(S)$, welches die Eingangsentropie der Senke S ist.

$$H(Q) = H(S) = H_T \tag{4.9}$$

Für den idealen Kanal kann somit die Entropie am Ausgang einer Quelle als Eingangsentropie der Senke verwendet werden. In diesem Fall gilt nach dem Bergerschen Kanalmodell, dass die Entropie am Eingang des Kanals gleich der Entropie am Ausgang des Kanals ist, wie Definition 5 besagt. Somit muss nicht die Entropie des Kanals beziehungsweise der Verbindung direkt betrachtet werden, sondern es können die Eingangszustände und somit die Eingangsentropien der Komponente betrachtet werden. Da die Zustände eines Eingangs als gleichwahrscheinlich angenommen werden und die Eingänge statistisch unabhängig sind, können die Informationsentropien mehrerer Eingänge analog Definition 4 aufsummiert werden.

Der Austausch von Informationen beziehungsweise Nachrichten bildet die Grundlage und zugleich das Modell der Entwurfsentropie. Beim Entwurf eines digitalen Systems werden dessen Komponenten derart miteinander verbunden, dass die auszuführenden Aufgaben erfüllt werden. Durch die Messung des mittleren Informationsgehalts der Nachrichten kann die Komplexität eines Systems bestimmt werden. Die Berücksichtigung der einzelnen Schnittstellen der Komponenten lässt zudem die Entwurfsgröße in die Maßzahl mit einfließen. Dabei wird die in einem System liegende Komplexität und Entwurfsgröße gemessen und nicht die subjektiv empfundene Komplexität. Das heißt, die Entwurfsentropie trifft eine Aussage über das jeweils vorliegende System und nicht darüber, wie die minimale, optimale oder effizienteste Umsetzung wäre.

Hierauf aufbauend werden im folgenden Abschnitt die für die Berechnung der Entwurfsentropie erforderlichen Begrifflichkeiten definiert. Dabei wird sowohl auf das System, dessen Komponenten und die Verbindungen eingegangen als auch das Verfahren vorgestellt, wie der Aufbau aus einzelnen Komponenten analysiert wird.

4.2 Definitionen

Die Entwurfsentropie ist zunächst ein abstrakter Ansatz, welcher in verschiedenen Bereichen Anwendung finden kann [[143]]. Daher sind die folgenden Definitionen so formuliert, dass die universelle Anwendung der Metrik möglich ist. Eine Konkretisierung der Definitionen erfolgt zum einen durch die anschließende Herleitung der Formeln und zum anderen durch den Bezug zur digitalen Hardware in der zweiten Hälfte dieses Kapitels. Ziel der Definitionen ist primär, die jeweiligen Begrifflichkeiten einzuführen. Wie im vorherigen Abschnitt dargestellt, liegt der Berechnung der Entwurfsentropie immer das tatsächliche System zugrunde, das heißt, ein Entwurf, eine Umsetzung, ein Programm oder eine Implementierung. Hieraus ergeben sich die ersten grundlegenden Definitionen für die Entwurfsentropie, die Komplexität und das System.

Definition 6 (Entwurfsentropie)

Die Entwurfsentropie ist ein Maß für die Quantität (Größe) und die Komplexität eines Systems.

Definition 7 (Komplexität)

Die Komplexität umfasst die „Gesamtheit aller voneinander abhängigen Merkmale und Elemente, die in einem vielfältigen aber ganzheitlichen Beziehungsgefüge (System) stehen. Unter Komplexität wird die Vielfalt der Verhaltensmöglichkeiten der Elemente und die Veränderlichkeit der Wirkungsverläufe verstanden. Die Komplexität ist durch Anzahl und Art der Elemente und deren Beziehungen untereinander bestimmbar.“ [53]

Definition 8 (System)

Ein System bezeichnet die strukturierte systematische Gesamtheit von miteinander in Bezug stehenden Komponenten und Verbindungen, welche als aufgaben-, sinn- oder zweckgebundene Einheit angesehen werden, ohne dabei Teil einer übergeordneten Komponente zu sein.

Ein System gemäß Definition 8 stellt beispielsweise der jeweilige Hardwareentwurf oder integrierte Schaltkreis dar, für welchen die Entwurfsentropie aus Definition 6 berechnet werden soll. In der Hardwarebeschreibungssprache VHDL entspricht das System der Top-Level-Entität, bei Verilog dem Top-Level-Modul und bei SystemC der Top-Level Funktion (`main`) [[143]]. Zur besseren Übersicht ist ein System mit seinen Teilen in der folgenden Abbildung 4.5 dargestellt.

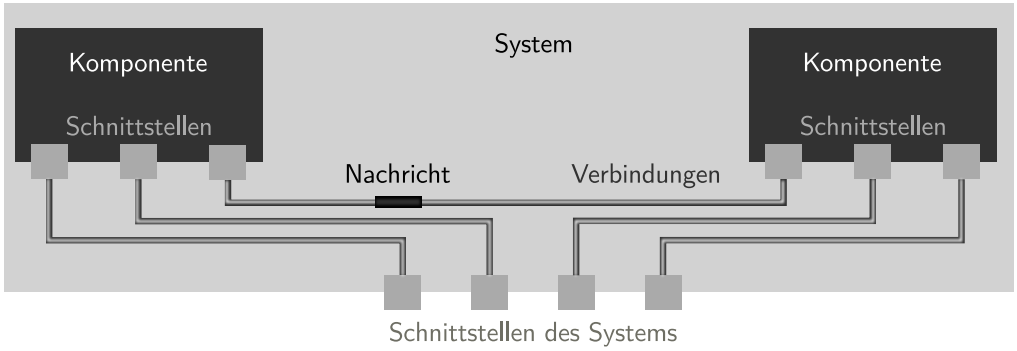


Abbildung 4.5: System mit seinen Teilen

Das System besteht aus Komponenten, welche selbst wiederum Komponenten enthalten können, sowie aus Verbindungen zwischen den Schnittstellen der Komponenten, welche dem Nachrichtenaustausch dienen [[143]]. Dabei stellt das System selbst eine Komponente dar, welche aber nicht Teil einer anderen Komponente ist, das heißt, es gibt keine „Oberkomponente“ zu einem System. Komponenten kommunizieren mit ihrer Umwelt über definierte Schnittstellen. Hieraus ergeben sich die nächsten Definitionen für die Komponenten, deren Umwelt und die Elementarkomponenten.

Definition 9 (Komponente)

Eine Komponente ist entweder das System selbst oder ein zusammenhängender Teil des Systems, welche Nachrichten in irgendeiner Form verarbeitet [[143]]. Sie hat die Möglichkeit mit ihrer Umwelt über definierte Schnittstellen zu kommunizieren. Eine Komponente kann wiederum selbst aus Komponenten bestehen und/oder kann, mit Ausnahme des Systems als Komponente, Teil einer anderen Komponente sein.

Definition 10 (Umwelt)

Die Umwelt ist alles außerhalb der jeweils betrachteten Komponente beziehungsweise des jeweils betrachteten Systems.

Definition 11 (Elementarkomponente)

Eine Elementarkomponente ist eine Komponente, welche keine weiteren (Unter-)Komponenten und Verbindungen enthält, die beim Entwurf des vorliegenden Systems entworfen oder verdrahtet wurden [[142]].

Anhand von Definition 9 für die Komponente zeigt sich der „rekursive“ Ansatz der Entwurfsentropie. Komponenten können selbst Komponenten enthalten. Erst die Elementarkomponenten enthalten keine weiteren Komponenten mehr [[142]]. Das heißt, die Elementarkomponenten sind diejenigen Komponenten, welche nicht mehr selbst entworfen wurden, sondern unverändert verwendet werden. Daher muss deren Struktur/Aufbau nicht mehr berücksichtigt werden. Elementarkomponenten können beispielsweise Gatter sein, die verwendet werden, oder auch ganze Objekte/ Bausteine aus einer Bibliothek, wie zum Beispiel Speicher.

Gemäß Definition 11 ist es dabei unerheblich, ob eine verwendete Elementarkomponente tatsächlich nicht mehr in weitere Komponenten zerlegt werden kann. Ausschlaggebendes Merkmal ist, dass eine Elementarkomponente nicht innerhalb des Entwurfs des aktuell betrachteten Systems entwickelt wurde. Somit umfasst der Begriff der Elementarkomponente auch wiederverwendete Komponenten anderer Systeme und solche aus Bibliotheken.

Damit eine Komponente mit ihrer Umwelt kommunizieren kann, benötigt diese Schnittstellen. Über diese können Nachrichten mit anderen Schnittstellen und der Umwelt ausgetauscht werden. Hierfür sind Verbindungen zwischen den Schnittstellen erforderlich. Die Nachrichten werden als mögliche Zustände aufgefasst.

Definition 12 (Schnittstelle)

Eine Schnittstelle ermöglicht einer Komponente den Austausch von Nachrichten mit ihrer Umwelt und den in ihr enthaltenen Unterkomponenten. Sie ist für die Umwelt und die Komponente selbst sichtbar und bildet entweder eine Nachrichtensenke, eine Nachrichtenquelle oder eine Kombination der beiden.

Definition 13 (Verbindung)

Eine Verbindung erlaubt den Austausch von Nachrichten zwischen mindestens zwei Schnittstellen und verbindet diese miteinander oder zwischen mindestens einer Schnittstelle und der Umwelt [[143]].

Definition 14 (Nachrichten)

Nachrichten sind jegliche für eine Komponente verwertbare Daten, welche einen definierten oder definierbaren Zustand haben.

Definition 15 (Zustände)

> Zustände sind die abzählbare Gesamtheit der Nachrichten, welche eine Schnittstelle aufnehmen oder abgeben kann und welche zur vollständigen Beschreibung der momentanen Eigenschaften einer Schnittstelle erforderlich sind, sofern diese nicht eine unveränderliche Eigenschaft der Schnittstelle darstellen [[143]] [[144]].

Eine Komponente hat Eingänge, Ausgänge und/oder bidirektionale Schnittstellen. Dabei dienen die Schnittstellen für die Umwelt dazu, mit der Komponente zu kommunizieren und für die Komponente selbst, um Nachrichten über Verbindungen mit den in ihr enthaltenen Komponenten auszutauschen. Die Abgrenzung zwischen einer Komponente und einer Verbindung bezieht sich auf die Nachrichten: Werden ausschließlich Nachrichten ausgetauscht, dann handelt es sich um eine Verbindung; werden Nachrichten verarbeitet, handelt es sich um eine Komponente.

Wie im Vorherigen beschrieben, besteht ein Unterschied zwischen dem Verwenden und dem Entwurf einer Komponente. Wird eine Komponente nur verwendet, ist das Verhalten der Komponente ausschlaggebend [[143]]. Der Entwickler oder die Entwicklerin benötigt die Kenntnis, welche Nachrichten von der Komponente wie verarbeitet werden [[143]]. Dabei ist es für die Verwendung nicht erforderlich, den inneren Aufbau einer Komponente zu kennen, das heißt, welche innere Struktur die Komponente hat. In Anlehnung an die Domänen des Entwurfsprozesses, welche in Abschnitt 2.2 vorgestellt wurden, wird diese Art der Komplexität Verhaltensentropie genannt und ist in Definition 16 definiert. Durch die Verhaltensentropie wird somit nicht der innere Aufbau beziehungsweise die innere Struktur einer Komponente ausgedrückt, sondern nur die Komplexität ihres Verhaltens gegenüber ihrer Umwelt [[143]] [[146]].

Dabei enthält ein Entwurf nicht nur Komponenten, welche in Form von Elementarkomponenten oder wiederverwendeten Komponenten miteinander verbunden werden, sondern es werden auch eigene Komponenten aufgebaut. Denn bereits das System selbst stellt eine Komponente im Sinne der Entwurfsentropie dar, die entworfen wird. Beispielsweise kann aus einem Antivalenz- und einem Und-Gatter ein Halbaddierer als neue Komponente aufgebaut werden, die an mehreren Stellen des Entwurfs verwendet wird. Dies hat für die Entwurfsentropie zwei Folgen: Zum einen reduziert sich damit regelmäßig der Entwurfsaufwand, da die neue Komponente an mehreren Stellen verwendet werden kann. Dies wird durch die Entwurfsentropie dadurch berücksichtigt, dass für jede Verwendung einer Komponente nur deren Verhaltensentropie und nicht die Strukturentropie berücksichtigt

wird. Zum anderen ist zunächst Aufwand erforderlich, um die neue Komponente aufzubauen. Dies wird durch die Strukturentropie berücksichtigt. Dabei ist zu beachten, dass der Aufwand für den Entwurf der neuen Komponente nur ein Mal angefallen ist, sodass die Strukturentropie einer Komponente nur einmalig berücksichtigt wird. Somit stehen für die Berechnung der Entwurfsentropie eines Systems zwei verschiedene Entropien zur Verfügung: Die Verhaltensentropie, welche mit einer Blackboxsicht vergleichbar ist, und die Strukturentropie, welche mit einer Whiteboxsicht vergleichbar ist [[143]]. Der Unterschied zwischen der Struktur und dem Verhalten ist in Abbildung 4.6 illustriert.

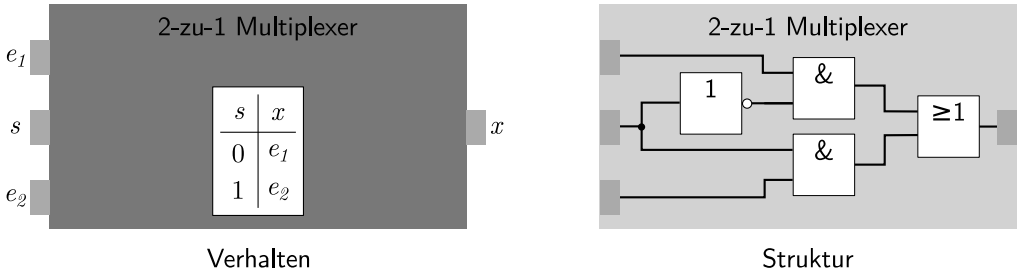


Abbildung 4.6: Verhalten und Struktur im Sinne der Entwurfsentropie

Für den dargestellten Multiplexer wird bei der Verhaltensentropie nur das Verhalten des Bausteins gegenüber seiner Umwelt berücksichtigt. Das heißt, dass der Ausgang x dem Eingang e_1 entspricht, wenn s null ist und e_2, wenn s eins ist. Dabei ist es nicht erforderlich, den Aufbau der Komponente zu kennen, zum Beispiel ob diese mit Gattern oder mithilfe eines Relais aufgebaut ist.

Beim Aufbau der Komponente aus Gattern wird dagegen von der Strukturentropie der jeweilige Aufbau berücksichtigt. Der Multiplexer besteht aus einem Nicht-Gatter, zwei Und-Gattern und einem Oder-Gatter. Diese Bauteile bilden mit der entsprechenden Verdrahtung den Multiplexer. Die Bauteile haben selbst eine Verhaltensentropie, aus der sich die Strukturentropie der Komponente berechnet. Der entworfene Multiplexer als Komponente kann an mehreren Stellen des Systems eingesetzt werden, ohne dass dieser jeweils neu aufgebaut werden muss. Daher wird die Strukturentropie einer aufgebauten Komponente nur ein Mal innerhalb eines Systems berücksichtigt, während der Aufwand für die Verdrahtung der aufgebauten Komponente mit anderen Komponenten bei jeder Verwendung anfällt, sodass die Verhaltensentropie bei jeder Verwendung einer Komponente berücksichtigt wird. Die Strukturentropie ist damit vergleichbar, dass von einer Komponente mehrere Instanzen gebildet werden oder, bei der objektorientierten Programmierung, von

einer Klasse mehrere Objekte instanziiert werden. Für alle Instanzen wurde nur eine Entität entworfen beziehungsweise eine Klasse implementiert. Die Aussagen der Struktur- und der Verhaltensentropie werden in den beiden folgenden Definitionen beschreiben.

Definition 16 (Verhaltensentropie)

Die Verhaltensentropie gibt die Komplexität an, welche bei der Verwendung einer Komponente, das heißt, der Verbindung der Schnittstellen dieser Komponente mit anderen Schnittstellen oder der Umwelt, entsteht [[143]].

Definition 17 (Strukturentropie)

Die Strukturentropie gibt die Quantität und Komplexität einer Komponente an, welche durch den Aufbau der Komponente aus anderen Komponenten mit deren Schnittellen und Verbindungen entsteht [[143]].

Dadurch, dass die Strukturentropie nur ein Mal berücksichtigt wird, muss bei der Berechnung der Entwurfsentropie eines Systems betrachtet werden, von welchen Komponenten die Strukturentropie bereits berücksichtigt wurde und für welche diese noch berechnet werden muss. Entscheidend dabei ist, dass die Strukturentropie einer Komponente, welche verwendet wird und innerhalb des Systementwurfs entstanden ist, nur ein Mal in die Entwurfsentropie einfließt, wobei die Verhaltensentropie für jede Verwendung einer Komponente einfließt. Die nachfolgende Abbildung 4.7 zeigt ein abstraktes System mit Komponenten, wobei mehrfach verwendete Komponenten in der gleichen Farbe dargestellt sind. Der linke Teil der Abbildung zeigt den Aufbau des Systems, der rechte Teil diejenigen Komponenten, welche in die Berechnung der Entwurfsentropie eingehen.

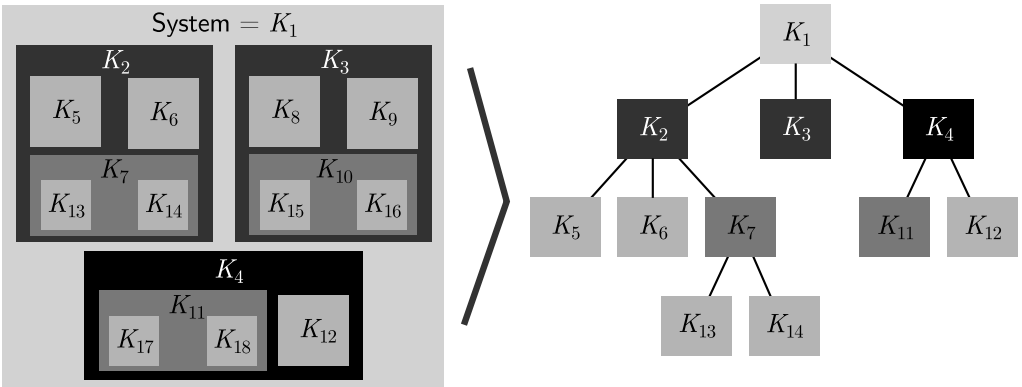


Abbildung 4.7: Verhaltens- und Strukturentropie bei mehreren Komponenten

Das System selbst stellt eine Komponente dar, sodass die Entwurfsentropie des Systems der Strukturentropie von K_1 entspricht. Es besteht aus den Unterkomponenten K_2, K_3 und K_4, wobei K_2 und K_3 identisch sind. Somit wird die Strukturentropie nur für K_2 (oder nur für K_3) berechnet. K_2 wiederum besteht aus den beiden Elementarkomponenten K_5 und K_6 sowie der Komponente K_7. Daher gehen in die Berechnung der Strukturentropie von K_2 die Verhaltensentropie der drei Komponenten sowie die Strukturentropie der Komponente K_7 ein, welche aus den Verhaltensentropien der Elementarkomponenten K_{13} und K_{14} besteht.

Für die Strukturentropie von K_1 wird noch die Strukturentropie von K_4 benötigt. Dabei enthält K_4 die Komponente K_{11}, welche mit K_7 (beziehungsweise K_{10}) identisch ist. Daher werden für die Berechnung der Strukturentropie von K_4 nur die Verhaltensentropien von K_{11} und K_{12} benötigt, da die Strukturentropie von K_{11} bereits innerhalb von K_2 berücksichtigt wurde. Somit ergibt sich für die Berechnung der Entwurfsentropie von K_1 der dargestellte Baum. Für alle Knoten mit Ausnahme von K_1 muss die Verhaltensentropie berücksichtigt werden. Zusätzlich muss für alle Knoten, welche Kinder haben, die Strukturentropie berücksichtigt werden. Wie die Verhaltens- und die Strukturentropie im Detail berechnet werden, wird in den beiden folgenden Abschnitten gezeigt.

4.3 Verhaltensentropie

Die Verhaltensentropie gibt die Komplexität der Verwendung einer Komponente an, welche dadurch entsteht, dass die Schnittstellen der Komponente mit anderen Schnittstellen verbunden werden. Hierzu werden die möglichen Zustände an den Schnittstellen als gleichwahrscheinlich angenommen, um unabhängig von den jeweiligen Zustandswahrscheinlichkeiten die Komplexität einer Komponente beschreiben zu können, wie beim Modell der Entwurfsentropie im ersten Abschnitt 4.1 herausgearbeitet wurde. Durch die statistische Unabhängigkeit kann die Entropie einer Komponente gemäß Definition 4 als Summe der Einzelentropien berechnet werden. Abbildung 4.8 zeigt eine Komponente mit getrennten Ein- und Ausgängen sowie eine Komponente mit einer (bidirektionalen) Schnittstelle. Die linke Komponente hat die Eingänge n_1 und n_2 sowie die Ausgänge m_1 und m_2. Die mittlere Komponente besitzt die beiden bidirektionalen Schnittstellen s_1 und s_2. Es ist auch möglich, dass Komponenten sowohl (bidirektionale) Schnittstellen als auch dedizierte Ein- und Ausgänge haben, wie die rechte Komponente in Abbildung 4.8 mit einer Schnittstelle, einem Ein- und einem Ausgang zeigt.

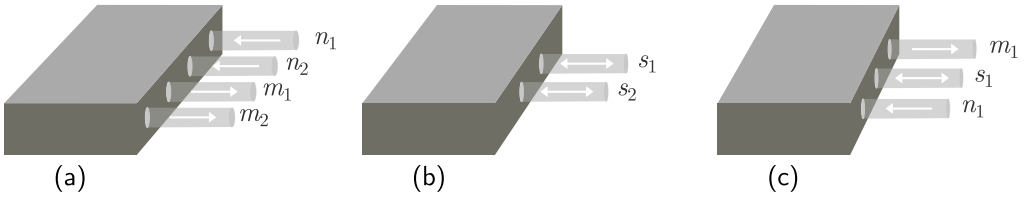


Abbildung 4.8: Komponente mit (a) dedizierten Ein- und Ausgängen; (b) Schnittstellen; (c) Schnittstelle, Ein- und Ausgang [[143]]

Eine Komponente ist nach Definition 9 eine abstrakte Beschreibung des Systems oder eines Teils davon, die Nachrichten verarbeitet. Elementarkomponenten sind nach Definition 11 diejenigen Komponenten, welche keine weiteren Unterkomponenten enthalten, die beim Entwurf des aktuell betrachteten Systems entworfen wurden. In der digitalen Hardware stellen in der Strukturdomäne die jeweils verwendeten Bausteine die Elementarkomponenten dar, welche miteinander verbunden werden. Dies können beispielsweise Gatter sein oder auch ganze Speicher. Abbildung 4.9 illustriert auf der linken Seite einen solchen strukturellen Aufbau. In der Verhaltensdomäne stellen die einzelnen Operationen und Anweisungen Komponenten dar. Wird beispielsweise ein Algorithmus als Graph dargestellt, so sind die einzelnen Knoten die verwendeten Komponenten, wie Rechenoperationen oder Verzweigungen. In der rechten Hälfte von Abbildung 4.9 ist ein Graph mit den Komponenten gezeigt.

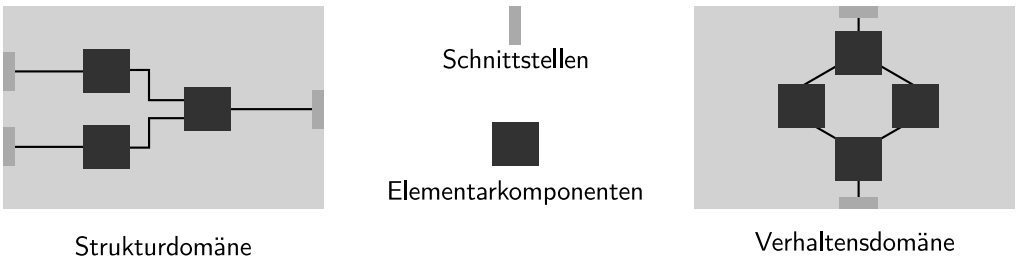


Abbildung 4.9: Komponenten in den Domänen

Damit Schnittstellen Nachrichten austauschen können, müssen diese miteinander verbunden werden. Gemäß Definition 13 erlaubt eine Verbindung „den Austausch von Nachrichten“. Für einen verlustfreien ungestörten Kanal (idealer Kanal) entspricht die Quellenentropie der Senkenentropie [197, S. 78], wie Definition 5 besagt. Gemäß Formel (4.9) entspricht die Transinformation H_T der Verbindung der Eingangsentropie der Senke S, wie die folgende Abbildung 4.10 verdeutlicht.

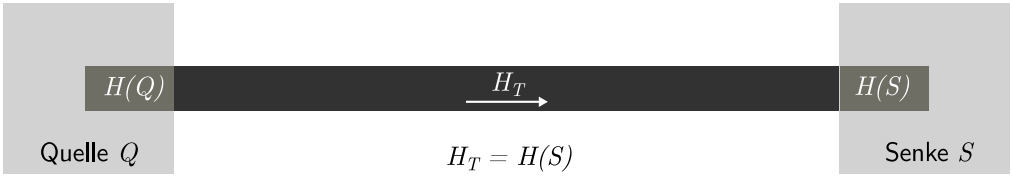


Abbildung 4.10: Transinformation entspricht der Senkenentropie

Somit bildet die Summe der Einzelentropien der Ein- und Ausgänge die Verhaltensentropie einer Komponente k, wie die folgende Formel verdeutlicht. Dabei sind die Eingänge einer Komponente k als $n_i(k)$ mit $i = 1, \ldots, n(k)$ und Ausgänge als $m_j(k)$ mit $j = 1, \ldots, m(k)$ bezeichnet.

$$H(k) = \sum_{i=1}^{n(k)} H(n_i(k)) + \sum_{j=1}^{m(k)} H(m_j(k)) = \sum_{i=1}^{n(k)} \log z(n_i(k)) + \sum_{j=1}^{m(k)} \log z(m_j(k))$$

Für das Modell ist es dabei unwesentlich, ob es sich wirklich um getrennte Ein- und Ausgänge handelt oder um Schnittstellen $s_l(k)$ mit $l = 1, \ldots, s(k)$ und den verschiedenen Zuständen $z(s_l(k))$ für die Ein- und Ausgaben, da durch die Anzahl der Zustände diese entweder innerhalb der einzelnen Ein- und Ausgänge oder innerhalb einer Schnittstelle auftreten können:

$$H(k) = \log \left(\prod_{i=1}^{n(k)} z(n_i(k)) \cdot \prod_{j=1}^{m(k)} z(m_j(k)) \right) = \log \left(\prod_{l=1}^{s(k)} z(s_l(k)) \right)$$

Die Verbindungen können auf drei grundlegende Varianten zurückgeführt werden, welche in Abbildung 4.11 illustriert sind: die *1*-zu-*1* Verbindung, die *1*-zu-*n* Verbindung und die *m*-zu-*1* Verbindung.

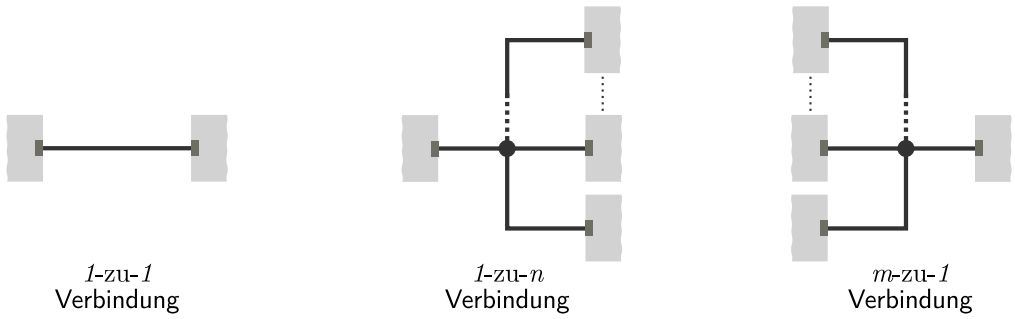


Abbildung 4.11: Verbindungsarten

Den Grundfall bildet die Verbindung eines Ausgangs mit einem anderen Eingang (*1*-zu-*1* Verbindung). Dabei werden beim Entwurf diejenigen Nachrichten betrachtet, welche zum einen ausgegeben werden und zum anderen diejenigen Nachrichten, welche als (mögliche) Eingaben dienen. Die Schnittstellen sind in der Form miteinander verbunden, dass der Nachrichtenfluss und die Nachrichtenverarbeitung durch die Komponenten die Aufgaben des Systems erfüllen.

Im zweiten Fall wird ein Ausgang mit zwei oder mehr Eingängen verbunden (*1*-zu-n Verbindung). Der hierdurch steigende Verdrahtungsaufwand wird nicht durch die Verbindung selbst berücksichtigt, sondern dadurch, dass die möglichen Zustände an den n Eingängen anliegen können. Denn für den idealen Kanal entspricht die Transinformation der Senkenentropie, das heißt, der Entropie an den Eingängen der Komponenten. Dies folgt dem zugrunde liegenden Ansatz der Entwurfsentropie, dass beim Entwurf die Kenntnis über die möglichen Eingangszustände erforderlich ist, auch wenn eine gemeinsame Quelle für die Nachrichten besteht. Die Nachrichten der einen Quelle dienen den n Eingängen als Eingaben, wobei für jeden dieser Eingänge die möglichen Eingangszustände bekannt sein müssen.

Dabei können in der Praxis nicht unendlich viele Bauelemente direkt miteinander verbunden werden. Der Lastfaktor am Ausgang (Fan-Out) gibt an, wie viele Eingänge anderer Logikelemente der gleichen Art angesteuert werden können [214, S. 78]. Die Höchstgrenzen für die Lastfaktoren sind für die jeweilige Logikfamilie oder die einzelnen Bausteine in den Datenblättern hinterlegt und auch den Entwurfswerkzeugen bekannt. Erfolgt die Beschreibung der Hardware mithilfe eines Entwurfswerkzeugs, werden gegebenenfalls erforderliche Treiberbausteine vom Werkzeug selbstständig platziert, wenn die Anzahl anzusteuernder Eingänge den maximal zulässigen Lastfaktor eines Bauelements überschreitet. Somit entsteht beim Entwurf kein zusätzlicher Aufwand, das heißt, keine Komplexität für die Berücksichtigung der Anzahl Eingänge. Wird dagegen beim Entwurf berücksichtigt, dass die Ansteuerungskapazität überschritten wird, dann werden zusätzliche Bausteine benötigt, welche als Komponenten bei der Berechnung der Entwurfsentropie berücksichtigt werden. Somit wird der in einer Schaltung enthaltene zusätzliche Aufwand durch diese Komponenten mit berücksichtigt.

Der dritte Fall bildet das Zusammenführen mehrerer Ausgänge auf einen Eingang ab (m-zu-*1* Verbindung). Dabei ist zunächst zu berücksichtigen, dass die Ausgänge nicht direkt miteinander verbunden werden können [214, S. 79]. Für die Zusammenführung mehrerer Signale existieren zwei Vorgehensweisen: Zum einen kann der Ausgang einen weiteren hochohmigen Zustand (Tri-State/Open

Collector/Open Drain) haben [214, S. 76 f.]. Dieser muss durch eine zusätzliche Schaltung beziehungsweise eine zusätzliche Komponente innerhalb oder außerhalb der zu verbindenden Komponenten realisiert werden, gegebenenfalls mit zusätzlichen Steuersignalen. Zum anderen können mehrere Ausgänge über ein zusätzliches Gatter (insbesondere ein Oder-Gatter) verbunden werden. In beiden Fällen ist somit mindestens eine weitere Komponente erforderlich, welche die Zusammenführung von Ausgangssignalen ermöglicht. Diese Komponenten gehen in die Berechnung der Entwurfsentropie ein. Werden diese Komponenten dagegen durch das Entwurfswerkzeug eingefügt, werden sie nicht von der Entwurfsentropie berücksichtigt, da sie, wie im Fall des Fan-Out, nicht in der zugrunde liegenden Schaltung enthalten sind.

Eine allgemeine m-zu-n Verbindung mit n Eingängen und m Ausgängen ergibt sich als Kombination der m-zu-*1* und der *1*-zu-n Verbindung. Dadurch, dass eine weitere Komponente erforderlich ist, um mehrere Ausgänge zusammenzuführen, bildet im allgemeinen Fall einer m-zu-n Verbindung diese zusätzliche Komponente den Kern. Der Ausgang der Komponente kann dann, wie im Fall der *1*-zu-n Verbindung, wiederum mit mehreren Eingängen anderer Komponenten verbunden werden.

Basierend auf den vorherigen Darstellungen berechnet sich die Verhaltensentropie anhand der jeweils möglichen Zustände an den Schnittstellen einer Komponente. Durch diesen Ansatz werden alle Verbindungsarten und der dadurch unterschiedlich ausfallende Verdrahtungsaufwand berücksichtigt. Die Verhaltensentropie beschreibt das Verhalten einer Komponente unabhängig von der Entwurfsdomäne. Es wird dabei nicht der Aufbau der Komponente betrachtet, sondern nur das Verhalten ihrer Schnittstellen gegenüber der Umwelt. Aufbauend auf Definition 16, welche die Aussage der Verhaltensentropie festlegt, lässt sich diese mithilfe von Formel (4.10) der folgenden Definition 18 berechnen.

Definition 18 (Berechnung der Verhaltensentropie)

Sei k eine Komponente mit Schnittstellen $s_l(k)$ mit $l = 1, \ldots, s(k)$, wobei $s(k)$ die Anzahl an Schnittstellen der Komponente k ist und sei $z(s_l(k))$ die Anzahl möglicher Zustände der Schnittstelle $s_l(k)$.

Beziehungsweise seien $n_i(k)$ für $i = 1, \ldots, n(k)$ die Eingänge mit den möglichen Zuständen $z(n_i(k))$ und $m_j(k)$ für $j = 1, \ldots, m(k)$ die Ausgänge mit den möglichen Zuständen $z(m_j(k))$ der Komponente k [[143]].

Dann ist die Verhaltensentropie $H_B(k) \in \mathbb{R}_0^+$ einer Komponente k gegeben durch [[143]]:

$$\begin{aligned} H_B(k) &= \log \left(\prod_{l=1}^{s(k)} z(s_l(k)) \right) \\ &= \log \left(\prod_{i=1}^{n(k)} z(n_i(k)) \cdot \prod_{j=1}^{m(k)} z(m_j(k)) \right) \end{aligned} \tag{4.10}$$

Wie bereits beschrieben, wird der Logarithmus (log) als Zehnerlogarithmus (Dekadischer Logarithmus) verwendet, um zu verdeutlichen, dass die Entwurfsentropie ein Maß für die Komplexität und Quantität darstellt und nicht als nachrichtentechnisches Maß mit dem Logarithmus Dualis oder als naturwissenschaftliches Maß mit dem Logarithmus Naturalis verwendet wird. Eine andere Basis wirkt sich rein als „Skalierungsfaktor“ aus, das heißt, als Multiplikation/Division mit einer Konstanten: $\log_b(x) = \frac{\log_{10}(x)}{\log_{10}(b)}$. Zwar würde sich bei einer binären Logik grundsätzlich der Zweierlogarithmus anbieten, jedoch ist die Entwurfsentropie so formuliert, dass diese auf unterschiedliche Systeme angewendet werden kann. Somit wird nicht der Logarithmus Dualis verwendet, um nicht den Eindruck zu erzeugen, dass die Entwurfsentropie auf binäre Systeme beschränkt ist. Zudem tritt auch beim Entwurf von Hardwaresystemen regelmäßig der Fall ein, dass mehr als zwei Zustände verwendet werden, wie beispielsweise bei einem Zustandsautomaten oder bei Signalen mit zusätzlichen unbestimmten oder hochohmigen Zuständen [[143]]. Bei der Analyse von Softwareprojekten tritt dagegen häufiger der Fall ein, dass die Anzahl an Zuständen keine Zweierpotenz ist [[146]]. Sowohl die Entwurfsentropie insgesamt als auch die Verhaltens- und Strukturentropie sind dimensionslos, wie in Abschnitt 3.4 beschrieben. Zudem wird der Logarithmus ohne Angabe einer Basis verwendet, um auszudrücken, dass die Basis grundsätzlich keine Rolle spielt, sofern diese einheitlich gewählt wird. Die folgenden Berechnungen verwenden entsprechend den Logarithmus zur Basis zehn.

Anhand der Verhaltensentropie kann das Verhalten einer Komponente berechnet werden. Um die Entwurfsentropie eines gesamten Systems zu berechnen, welches sich aus mehreren Komponenten zusammensetzt, wird zusätzlich die Strukturentropie benötigt, deren Berechnung im folgenden Abschnitt vorgestellt wird.

4.4 Strukturentropie

Die Strukturentropie trägt den Tatsachen Rechnung, dass ein System aus mehreren Komponenten bestehen kann und Komponenten wiederum aus Unterkomponenten aufgebaut sein können [[143]]. Dieser hierarchische Aufbau, der zu einer rekursiven Analyse des Systems führt, wurde am Ende von Abschnitt 4.2 bei den Definitionen vorgestellt. Die Berechnung der Strukturentropie baut auf diesem rekursiven/ hierarchischen Ansatz auf und ist als Formel (4.11) der folgenden Definition 19 gegeben. Neben der Definition werden im Verlauf dieses Abschnitts zusätzlich eine mathematische Beschreibung sowie eine algorithmische Beschreibung der Berechnung gegeben, welche alle äquivalent sind.

Definition 19 (Berechnung der Strukturentropie)

Sei k eine Komponente mit direkten Unterkomponenten k_u. Seien $k_n \subseteq k_u$ diejenigen Unterkomponenten von k, welche beim Entwurf des Systems entworfen wurden (das heißt, keine Elementarkomponenten) und deren Strukturentropie bei der Berechnung der Entwurfsentropie des Systems noch nicht berücksichtigt wurde.

Dann ist die Strukturentropie $H_S(k) \in \mathbb{R}_0^+$ für die Komponente k durch die Summe der Verhaltensentropien aller direkten Unterkomponenten k_u und der Strukturentropien aller direkten Unterkomponenten k_n gegeben durch [[143]]:

$$H_S(k) = \sum_{i \in k_u} H_B(i) + \sum_{j \in k_n} H_S(j) \tag{4.11}$$

Anhand der zweiten Summe in Formel (4.11) ist der „rekursive“ Aufbau der Entwurfsentropie zu erkennen [[143]]. Bei der Analyse eines Systems wird die Strukturentropie von Komponenten, welche mehrfach verwendet werden, nur ein Mal berücksichtigt, da diese nur einmalig entworfen wurden. Zudem ist die Strukturentropie von Elementarkomponenten null, da diese nicht im Rahmen des zu analysierenden Systems entworfen wurden. Daher enthält k_n nur diejenigen Unterkomponenten der Komponente k, welche keine Elementarkomponenten sind und deren Strukturentropien bei der Berechnung der Entwurfsentropie des Systems noch nicht berücksichtigt wurden. Hieraus ergibt sich das Ende der Rekursion immer dann, wenn eine Komponente keine weiteren Unterkomponenten enthält und somit nur eine Verhaltensentropie besitzt aber keine Strukturentropie.

Die erste Summe der vorherigen Formel (4.11) summiert über die Verhaltensentropie aller direkten Unterkomponenten und die zweite Summe über die Strukturentropie derjenigen Komponenten, welche keine Elementarkomponenten oder bereits berücksichtigte Komponenten sind. Enthält die Komponente dabei keine weiteren Unterkomponenten mit einer Strukturentropie, dann ist $k_n = \emptyset$, sodass die Rekursion beendet ist, da die zweite Summe keine Elemente enthält, für die eine Strukturentropie berechnet werden muss.

Mathematisch kann die Berechnung der Strukturentropie mithilfe eines Baumes beschrieben werden: Eine Komponente θ ist ein gewurzelter Baum $\theta = (\Theta,E)$. Θ ist die abzählbare Multimenge aller Komponenten $\theta_i \in \Theta$ mit $i = 1, \ldots, |\Theta|$ des Baums. $w \in \Theta$ ist die Wurzel des Baums. Die gerichtete Kante $e \in E$ ist von der Form $e = (u,v)$ mit gewissen $E \subseteq \{(u,v)|u,v \in \Theta, u \neq v\}$, wobei u als Anfangsknoten und v als Endknoten von e bezeichnet wird. $\Theta' \subseteq \Theta$ ist die abzählbare Multimenge aller direkten Unterkomponenten θ' von θ mit $\Theta' = \{v : (w,v) \in E\}$. Φ ist die abzählbare Menge der Schnittstellen einer Komponente θ und $\sigma(\varphi)$ die Anzahl an Zuständen der Schnittstelle $\varphi \in \Phi$. Das System ist eine Komponente $\theta_T = (\tilde{\Theta},E)$, wobei $\tilde{\Theta}$ die abzählbare Multimenge aller Komponenten $\tilde{\theta}_j \in \tilde{\Theta}$ mit $j = 1, \ldots, |\tilde{\Theta}|$ des Systems θ_T ist. Die Äquivalenzrelation $\tilde{\theta}_j \equiv \theta'_i$ drückt aus, dass die Komponente $\tilde{\theta}_j \in \tilde{\Theta}$ der Komponente $\theta'_i \in \Theta'$ entspricht.

Hieraus ergibt sich die Verhaltensentropie $\Omega_B(\theta)$ einer Komponente θ mit:

$$\Omega_B(\theta) = \log\left(\prod_{\varphi\in\Phi} \sigma(\varphi)\right) \tag{4.12}$$

Die Strukturentropie $\Omega_S(\theta)$ einer Komponente θ ist dann gegeben als:

$$\Omega_S(\theta) = \sum_{\theta'\in\Theta'} \Omega_B(\theta') + \sum_{\substack{\theta'_i\in\Theta' \\ i=1,\ldots,|\Theta'| \\ \theta'_i\notin\{\theta'_{i+1},\ldots,\theta'_{|\Theta'|}\} \\ \tilde{\theta}_j\in\tilde{\Theta} \\ \tilde{\theta}_j\equiv\theta'_i \\ \theta'_i\notin\{\tilde{\theta}_{j+1},\ldots,\tilde{\theta}_{|\tilde{\Theta}|}\}}} \Omega_S(\theta'_i) \tag{4.13}$$

Die Entwurfsentropie Ω des Systems θ_T ist somit:

$$\Omega = \Omega_S(\theta_T) \tag{4.14}$$

Die Berechnung der Entwurfsentropie beginnt bei der mathematischen Beschreibung mit Formel (4.14). Dabei entspricht die Entwurfsentropie des Systems Ω der Strukturentropie des Systems $\Omega_S(\theta_T)$, welche sich nach Formel (4.13) berechnet. Die erste Summe dieser Formel ist die Verhaltensentropie Ω_B aller direkten Unterkomponenten. Direkte Unterkomponenten θ' einer Komponente θ sind dadurch gekennzeichnet, dass eine gerichtete Kante von der Wurzel des Baum $w \in \Theta$ zum Knoten existiert ($\Theta' = \{v : (w,v) \in E\}$). Die Verhaltensentropie berechnet sich nach Formel (4.12) aus dem Logarithmus des Produkts der Anzahl an Zuständen der Schnittstellen $\sigma(\varphi)$ der betrachteten Unterkomponente.

Die zweite Summe der Strukturentropieberechnung in Formel (4.13) summiert über die Strukturentropien aller direkten Unterkomponenten $\theta' \in \Theta'$ der aktuellen Komponente θ. Die Unterkomponenten werden dabei mit dem Index i durchnummeriert. Jedoch wird die Strukturentropie nur dann berechnet, wenn die Unterkomponente nicht ein weiteres Mal vorkommt. Das heißt, ausgehend von der aktuellen Unterkomponente θ'_i, darf diese nicht in der Menge der noch zu berechnenden Unterkomponenten $\{\theta'_{i+1}, \ldots, \theta'_{|\Theta'|}\}$ enthalten sein. Da jede Komponente einen Baum bildet, sind zwei Komponenten identisch, wenn sie den gleichen Baum haben. Da eine Unterkomponente auch außerhalb der aktuell zu analysierenden Komponente verwendet werden kann, darf die aktuelle Unterkomponente θ'_i zudem nicht in der Menge der noch zu analysierenden Komponenten $\{\tilde{\theta}_{j+1}, \ldots, \tilde{\theta}_{|\tilde{\Theta}|}\}$ enthalten sein. Dabei wird die Äquivalenzrelation verwendet, um den Index der aktuell zu analysierenden Komponente in der Menge aller Komponenten des Systems zu finden ($\tilde{\theta}_j \equiv \theta'_i$). Die Rekursion ist dadurch beendet, dass eine Komponente keine weiteren Unterkomponenten mehr enthält, da in diesem Fall die Menge der Unterkomponenten der leeren Menge entspricht: $\Theta' = \emptyset$.

Für die mathematische Beschreibung wird angenommen, dass der Baum mit allen Komponenten zu Beginn der Analyse bekannt ist. Dadurch kann geprüft werden, ob die Strukturentropie einer Komponente noch „später“ berechnet wird und somit aktuell nicht berechnet werden muss. Für die Umsetzung als Algorithmus und zur Darstellung der Berechnung als Pseudocode 4.1 bietet sich eine andere Vorgehensweise an, wobei der Anfang der mathematischen Beschreibung entspricht: Die Berechnung der Entwurfsentropie eines Systems beginnt mit der Analyse des Systems, welches selbst eine Komponente darstellt. Somit wird die Entwurfsentropie eines Systems dadurch berechnet, dass die Strukturentropie des Systems als Komponente berechnet wird. Der folgende Code 4.1 zeigt den Ablauf der Berechnung der Strukturentropie als (Pseudo-)Algorithmus.

```
1  function Strukturentropie(Komponente k; Liste L)
2    H_S = 0                          Für eine Elementarkomponente ist die Strukturentropie 0
3    for all k_u ∈ k                  Für alle Unterkomponenten der aktuellen Komponente k
4      H_S += Verhaltensentropie(k_u) Berücksichtigung der Verhaltensentropie
5      if k_u ∉ L                     Wenn die Strukturentropie noch nicht berücksichtigt wurde
6        H_S += Strukturentropie(k_u, L) Berechnung der Strukturentropie (rekursiver Aufruf)
7        L = L ∪ {k_u}                Komponente zur Liste der analysierten Komponenten hinzufügen
8      end if
9    end for
10   return H_S                       Das Ergebnis ist die Strukturentropie
11 end function
12
13 function Verhaltensentropie(Komponente k)
14     H_B = 1
15     for all s ∈ k                  Für alle Schnittstellen der Komponente
16         H_B *= (z_in(s) + z_out(s)) Anzahl möglicher Ein- und Ausgangszustände
17     end for
18     return log H_B
19 end function
```

Code 4.1: Pseudocode zur Berechnung der Strukturentropie

Die Strukturentropie wird durch den Aufruf der Strukturentropiefunktion in Zeile 1 des Pseudocodes berechnet, welcher zum einen die zu analysierende Komponente k und eine Liste L mit bereits analysierten Komponenten übergeben wird. In Zeile 2 wird zunächst die Strukturentropie der Komponente auf null gesetzt. Denn Elementkomponenten oder wiederverwendete Komponenten haben eine Strukturentropie von null [[143]]. Die Entwicklung dieser Komponenten fand außerhalb der Entwicklung des zu analysierenden Systems statt, sodass die Komplexität (und Quantität) der Struktur nicht zur Komplexitätssteigerung (und Quantitätssteigerung) des Systems beiträgt. Diese Komponenten werden nur innerhalb des Entwurfs verwendet, sodass hier nur die Verhaltensentropie zu einer Steigerung der Entwurfsentropie beiträgt. Da diese Komponenten somit keine zu analysierenden Unterkomponenten enthalten, ist die Menge k_u in Zeile 3 leer, sodass die Schleife nicht betreten wird und dadurch auch kein rekursiver Aufruf in Zeile 6 erfolgt.

Besitzt die Komponente zu analysierende Unterkomponenten, wird für jede dieser Komponenten in Zeile 4 die Verhaltensentropie der Komponente mithilfe der Verhaltensentropiefunktion in den Zeilen 13 – 19 berechnet. In dieser Funktion wird zunächst die Variable H_B auf eins gesetzt. Da $\log(1)$ null ist, hat dies keine Auswirkungen auf die Verhaltensentropie, erleichtert jedoch die Berechnung. Dadurch kann für jede Schnittstelle der Komponente (Zeile 15) die Anzahl an

Zuständen, das heißt, die Summe der Ein- und Ausgangszustände, mit der Variablen H_B in Zeile 16 multipliziert werden. Die Rückgabe der Funktion ist der Logarithmus der Variablen in Zeile 18. Die berechnete Verhaltensentropie wird dann in Zeile 4 zur Strukturentropie der zu analysierenden Komponente hinzu addiert.

Um zu prüfen, ob von den Unterkomponenten der aktuellen Komponente die Strukturentropie berechnet werden muss, ermittelt Zeile 5 des Pseudocodes, ob die Unterkomponente in der Liste der bereits analysierten Komponenten enthalten ist. Ist dies nicht der Fall, wird die Strukturentropie der Unterkomponente durch den rekursiven Aufruf der Strukturentropiefunktion in Zeile 6 berechnet und die Komponente anschließend in Zeile 7 zur Liste L der bereits analysierten Komponenten hinzugefügt. Enthält die zu analysierende Komponente keine Unterkomponenten, die noch analysiert werden müssen, wird die Funktion in Zeile 6 nicht mehr rekursiv für die Unterkomponenten aufgerufen, sodass es, wie bei den Elementarkomponenten, zu einem Ende der Rekursion kommt. Die Rückgabe der Strukturentropiefunktion in Zeile 10 ist die berechnete Strukturentropie.

Der Unterschied zwischen der algorithmischen Beschreibung und der mathematischen Beschreibung ist demnach, dass der Algorithmus als Rekursion umgesetzt ist. Die mathematische Beschreibung dagegen baut darauf auf, dass der vollständige Baum bereits vor der Analyse bekannt ist. Dies würde als Algorithmus ein zweimaliges Durchlaufen erfordern, was durch die Rekursion nicht erforderlich ist. Dabei lassen alle drei Beschreibungsformen die Berechnung der Entwurfsentropie zu. Im Folgenden wird für die Berechnung der Strukturentropie Formel (4.11) aus Definition 17 verwendet. Die Formel enthält bereits die Rekursion, welche der Pseudoalgorithmus nutzt. Die Verhaltensentropie wird mithilfe von Formel (4.10) aus Definition 18 des vorherigen Abschnitts berechnet.

Wie beschrieben, wurden die Terme „Verhaltensentropie“ und „Strukturentropie“ den Domänen des Entwurfsprozesses (Verhaltensbeschreibung und Strukturbeschreibung) entliehen. Sie stehen aber nicht mit diesen in einem direktem Zusammenhang. Das heißt, die Verhaltensentropie berechnet nicht ausschließlich die Komplexität in der Verhaltensdomäne und die Strukturentropie nicht ausschließlich die Komplexität in der Strukturdomäne. Die Berechnung der Entwurfsentropie ist unabhängig von den Domänen. Die Verhaltensentropie beschreibt die Komplexität der Verwendung einer Komponente (vergleichbar einer Blackboxsicht) in allen Domänen. Die Strukturentropie beschreibt die Komplexität und Entwurfsgröße des Aufbaus einer Komponente (vergleichbar einer Whiteboxsicht) in allen Domänen. Somit kann auch eine Verhaltensbeschreibung eine Strukturentropie haben, wenn

diese wiederum aus Komponenten aufgebaut ist. Die drei folgenden Abschnitte gehen detailliert darauf ein, wie die Entwurfsentropie in den jeweiligen Domänen und auf den unterschiedlichen Abstraktionsebenen berechnet wird.

4.5 Strukturdomäne

Je nachdem auf welcher Abstraktionsebene ein Entwurf umgesetzt wird, werden unterschiedliche Komponenten verwendet, wie in Abschnitt 2.2 im Zusammenhang mit dem Entwurfsprozess und dem Y-Diagramm beschrieben wurde. Diese Komponenten bilden die Basis eines Entwurfs. Sie werden als „Grundbausteine“ verwendet, das heißt, ihr innerer Aufbau ist nicht Teil des Entwurfs, sodass diese Komponenten Elementarkomponenten im Sinne der Entwurfsentropie darstellen. Nach Definition 11 sind Elementarkomponenten diejenigen Komponenten eines Systems, „welche keine weiteren (Unter-)Komponenten und Verbindungen [enthalten], die beim Entwurf des vorliegenden Systems entworfen oder verdrahtet wurden“. Beispielsweise stellen auf der RTL-Ebene Register und Rechenwerke die Elementarkomponenten dar. Auf der Systemebene sind die Elementarkomponenten beispielsweise Prozessoren und Speicher. Auf der Logikebene kommen Gatter und Flipflops zum Einsatz.

Typisch für die Strukturbeschreibung sind kombinatorische Schaltungen, die auf Logikgattern basieren und eine eindeutige Verknüpfung zwischen den Eingangs- und Ausgangssignalen herstellen [121, S. 121]. Das heißt, dass keine Rückkopplung innerhalb einer Schaltung besteht [191, S. 215] und die Ausgaben nur von den Eingangssignalen abhängen [205, S. 139]. Kombinatorische Schaltungen können durch Boolesche Funktionen dargestellt werden, welche auch deren technische Umsetzung mit Gattern bilden [121, S. 121].

Beim Aufbau einer kombinatorischen Schaltung mit Gattern oder anderen Elementen stellen diese die (Elementar-)Komponenten dar. Mehrere Gatter können zu einer neuen Komponente zusammengefasst werden. Beispielsweise kann aus einem Und-Gatter und einem Antivalenz-Gatter ein Halbaddierer aufgebaut werden, wie Abbildung 4.12 zeigt. Der Halbaddierer stellt eine Komponente im Sinne der Entwurfsentropie dar, jedoch keine Elementarkomponente mehr, da er innerhalb des Entwurfs mithilfe von zwei Gattern aufgebaut wurde. Somit besitzt der Halbaddierer im Gegensatz zu den Gattern eine Strukturentropie.

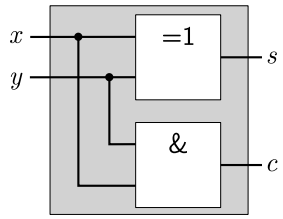


Abbildung 4.12: Halbaddierer aus einem Antivalenz- und einem Und-Gatter [[143]]

Aus den zwei Halbaddierern und einem zusätzlichen Oder-Gatter kann ein Volladdierer aufgebaut werden. Der Aufbau ist in Abbildung 4.13 dargestellt. Der linke Halbaddierer liefert als Summe den Eingangsübertrag für den rechten Halbaddierer. Die beiden Ausgangsüberträge der Halbaddierer werden mithilfe eines Oder-Gatters zum Ausgangsübertrag des Volladdierers verbunden.

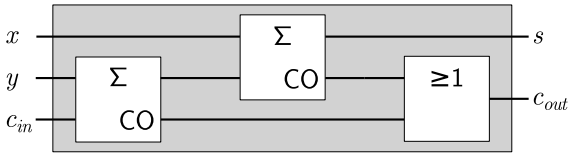


Abbildung 4.13: Volladdierer aus Halbaddierern [[143]]

Durch den modularen Aufbau eines Systems aus Unterkomponenten kann ein komplexeres System in Teile zerlegt werden [233, S. 107]. In der Hardwarebeschreibungssprache VHDL beschreibt eine Komponente (Entity) eine Entwurfseinheit (design unit), welche in anderen VHDL-Komponenten verwendet werden kann [233, S. 107]. Mithilfe von drei Zwischensignalen (`c1`, `c2` und `s1`) kann der Volladdierer als VHDL-Strukturbeschreibung in Code 4.2 umgesetzt werden. Dabei werden durch die `PORT MAP` zwei „Instanzen“ der Halbaddiererkomponente gebildet und die entsprechenden Signale zugewiesen. Zusätzlich ist eine Instanz der Oder-Komponente erforderlich, welche die Ausgangsüberträge der beiden Halbaddierer zum Ausgangsübertrag des Volladdierers verbindet. Statt der Verwendung einer Instanz eines Oder-Gatters kann auch alternativ die übliche Schreibweise der Verhaltensbeschreibung mit `c_out <= c1 OR c2` gewählt werden.

```
1   HA1: half_adder PORT MAP(y,c_in,s1,c1);
2   HA2: half_adder PORT MAP(x,s1,s,c2);
3   OR1: or_gate PORT(c1,c2,c_out);
```

Code 4.2: VHDL-Code des Volladdierers aus Abbildung 4.13

Beim Entwurf können auch Elementarkomponenten verschiedener Abstraktionsebenen verwendet werden. Beispielsweise können für ein Eingebettetes System ein Softcoreprozessor (Systemebene), Register (Register-Transfer-Ebene) und Logikgatter (Logikebene) verwendet werden. Da die Elementarkomponenten (Prozessor, Register, Gatter) nicht innerhalb des Entwurfs umgesetzt wurden, haben diese nur eine Verhaltensentropie und keine Strukturentropie. Der gesamte Entwurf hat dagegen eine Strukturentropie, da dieser die Elementarkomponenten verwendet und diese miteinander verbindet. Hierdurch wird bei der Umsetzung die Kommunikation zwischen den (Elementar-)Komponenten entworfen. Dabei finden sich Komponenten nicht nur in der Strukturdomäne, sondern auch in der Verhaltensdomäne wieder, sodass der Komponentenansatz auch auf Verhaltensbeschreibungen angewendet werden kann, wie der folgende Abschnitt zeigt.

4.6 Verhaltensdomäne

Auch in der Verhaltensdomäne bilden die Elementarkomponenten die Basis eines Entwurfs. Dabei werden die für diese Domäne typischen Elemente eingesetzt. Bei Booleschen Operationen bilden die Operatoren die Elementarkomponenten. Beispielsweise für die Schaltfunktion $y = a \cdot b$ einer Und-Verknüpfung stellt der Und-Operator $(\cdot)$ die Elementarkomponente dar, die verwendet wird. Als Eingänge der Komponente dienen die Signale/Variablen a und b und als Ausgang das Signal/ die Variable y. Wie in der Strukturdomäne können mehrere Schaltfunktionen zu einer neuen Komponente zusammengefügt werden:

$$x = a \cdot b$$
$$y = c \cdot d$$
$$z = x \cdot y$$

Aus den drei Und-Verknüpfungen kann eine neue Komponente gebildet werden, welche die Eingänge a bis d und den Ausgang z hat. Grafisch können die Und-Verknüpfungen als Blöcke mit Ein- und Ausgängen dargestellt werden. So zeigt Abbildung 4.14 in der linken Hälfte die Darstellung der einzelnen Gleichungen und in der rechten Hälfte die neue Komponente. Diese verwendet die Und-Verknüpfungen und besitzt damit eine Strukturentropie, das heißt, die neue Komponente besteht aus den Booleschen Und-Operatoren. Selbstverständlich kann auch direkt eine Elementarkomponente verwendet werden, welche vier Eingangssignale hat

und diese durch ein „Und“ verknüpft. In diesem Fall würde die in Abbildung 4.14 rechts dargestellte Komponente eine Elementarkomponente darstellen, da diese nicht, wie im vorherigen Beispiel, aus einzelnen Verknüpfungen aufgebaut wurde.

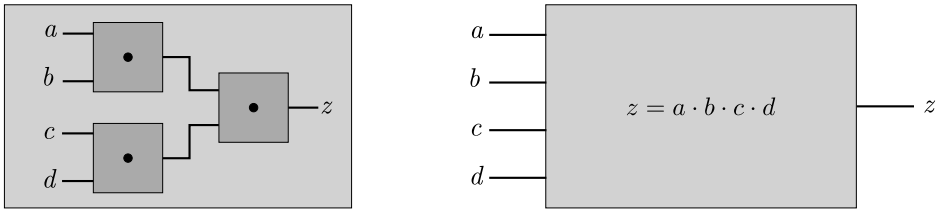


Abbildung 4.14: Blockdarstellung der Und-Verknüpfungen

Im Zusammenhang mit der Booleschen Algebra besteht bei manchen Umsetzungen die Möglichkeit, eine Logikminimierung vorzunehmen, beispielsweise mithilfe eines KV-Diagramms. Dabei ist die Logikminimierung nicht Gegenstand der Entwurfsentropie. Das heißt, es ist unbeachtlich, ob eine Gleichung oder Schaltung noch minimiert werden kann. Die Entwurfsentropie ist nach Definition 6 „ein Maß für die Quantität (Größe) und die Komplexität eines Systems.“ Wurde ein System aufwendiger umgesetzt, als die minimale Schaltung/Gleichung dies erfordern würde, enthält diese Umsetzung die höhere Komplexität und Entwurfsgröße. Denn der Aufwand für die Umsetzung des Systems mit mehr als den erforderlichen Gattern/ Gleichungen ist beim Entwurf angefallen. Und es ist das Ziel der Entwurfsentropie, die im System enthaltene Größe und Komplexität zu messen. Daher wird die zu analysierende Schaltung betrachtet und nicht das minimale oder optimale System.

Gleichungen als Grundlage einer Verhaltensbeschreibung werden in der Hardwarebeschreibungssprache VHDL als Signalzuweisungen umgesetzt, was anhand des folgenden Beispiels verdeutlicht wird [entnommen aus 233, S. 101 f.]:

$$y1 = x1 + \overline{x2} \cdot x3 + \overline{x1} \cdot x2$$
$$y2 = x3 + \overline{x1} \cdot x3 + \overline{x1} \cdot \overline{x2}$$
$$y3 = x2 + \overline{x2} \cdot x3$$

Die logischen Gleichungen verwenden die Boolesche Algebra, um Relationen auszudrücken. Das Plus (+) stellt eine Oder-Verknüpfung dar, das Mal (·) eine Und-Verknüpfung und das Makron ($^-$) eine Negation. Der folgende Code 4.3 zeigt die Umsetzung in VHDL als Signalzuweisungen:

```
1   y1 <= x1 OR (NOT x2 AND x3) OR (NOT x1 AND x2);
2   y2 <= x3 OR (NOT x1 AND x3) OR (NOT x2 AND NOT x1);
3   y3 <= x2 OR (NOT x2 AND x3);
```

Code 4.3: VHDL-Code der logischen Gleichungen [nach 233, S. 102]

Die Anweisungen für die logischen Gleichungen sind nebenläufig, das heißt, sie werden gleichzeitig ausgeführt, sodass die Reihenfolge unbeachtlich ist [233, S. 102]. Um ein sequenzielles Verhalten zu beschreiben, werden in VHDL Prozesse eingesetzt [233, S. 103]. Wichtige Teile von Prozessen sind Bedingungen, das heißt, if-then-else-Anweisungen und case-Anweisungen. Kern dieser Anweisungen bilden Vergleiche mit einer oder mehreren Bedingungen, die entweder wahr oder falsch sein können.

Die Vergleiche werden bei der Berechnung der Entwurfsentropie als Komponenten betrachtet. Jeder Vergleich kann zwei mögliche Ausgangszustände haben: Entweder ist er richtig oder falsch. Wie bei anderen Komponenten auch, richten sich die Eingangszustände nach den jeweiligen Signalen. Die Vergleiche als Kern der If-Bedingung und der Case-Anweisung lassen sich grafisch als die in den Abbildungen 4.15 und 4.16 gezeigten Komponenten darstellen.

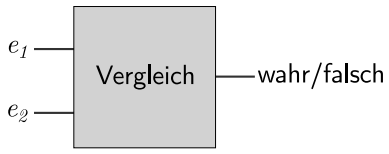


Abbildung 4.15: Vergleichskomponente der If-Bedingung

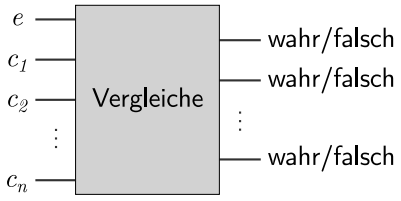


Abbildung 4.16: Vergleichskomponente der Case-Anweisung

Die Berechnung der Verhaltensentropie der beiden Komponenten erfolgt anhand der jeweiligen Ein- und Ausgangszustände der Komponenten. Die Vergleichskomponente der in Abbildung 4.15 gezeigten If-Bedingung hat zwei Eingänge, e_1 und e_2 sowie einen Ausgang, welcher die Zustände „wahr“ oder „falsch“ annehmen kann. Der eigentliche Vergleich der beiden Eingänge ist eine Elementarkomponente, da dieser nicht im vorliegenden Entwurf umgesetzt wurde, sondern nur verwendet wird. Somit hat dieser nur eine Verhaltensentropie, welche sich gemäß Formel (4.10) der Verhaltensentropie folgendermaßen berechnet, wobei $z(e_1)$ und $z(e_2)$ die Anzahl möglicher Zustände der beiden Eingänge sind:

$$
\begin{aligned}
H_B(\text{Vergleich}) &= \log\left(\prod_{i=1}^{n(k)} z(n_i(k)) \cdot \prod_{j=1}^{m(k)} z(m_j(k))\right) \\
&= \log\left(\prod_{i=1}^{2} z(e_i) \cdot \prod_{j=1}^{1} \underbrace{z(m_j)}_{=2}\right) \\
&= \log\left(z(e_1) \cdot z(e_2) \cdot 2\right)
\end{aligned}
$$

Nach dem gleichen Ansatz kann die Verhaltensentropie von mehreren Vergleichen berechnet werden, wie es für eine Case-Anweisung erforderlich ist. Die in Abbildung 4.16 visualisierte Komponente vergleicht das Eingangsignal e mit den Konstanten c_1 bis c_n. Das Eingangssignal e hat $z(e)$ mögliche Zustände. Jeder der Vergleiche kann dabei „wahr“ oder „falsch“ sein. Somit hat die Vergleichskomponente n Ausgänge mit jeweils zwei möglichen Zuständen. Eine Konstante kann dabei nur einen möglichen Zustand annehmen, sodass $z(c_i) = 1$ für $i = 1, \ldots, n$ ist. Damit lässt sich die Verhaltensentropie wie folgt berechnen:

$$
\begin{aligned}
H_B(\text{Vergleiche}) &= \log\left(\prod_{i=1}^{n(k)} z(n_i(k)) \cdot \prod_{j=1}^{m(k)} z(m_j(k))\right) \\
&= \log\left(\prod_{i=0}^{n} z(n_i(k)) \cdot \prod_{j=1}^{n} z(m_j(k))\right) \\
&= \log\left(z(e) \cdot \prod_{i=1}^{n} \underbrace{z(c(k))}_{=1} \cdot \prod_{j=1}^{n} \underbrace{z(m_j(k))}_{=2}\right) \\
&= \log\left(z(e) \cdot 1^n \cdot 2^n\right) = \log\left(z(e) \cdot 2^n\right)
\end{aligned}
$$

Die vorherigen Beispiele zeigen, dass die Entwurfsentropie mit dem Komponentenansatz sowohl in der Verhaltens- als auch in der Strukturdomäne angewendet werden kann. Die Bedingungen und Vergleiche bilden dabei die Grundlage, um sequenzielle Schaltungen zu entwerfen. Bevor auf einzelne sequenzielle Elemente, wie Flipflops, eingegangen wird, zeigt der folgende Abschnitt, wie die Analyse von Komponenten erfolgt, die als Verhaltensbeschreibung gegeben sind.

4.7 Komponenten

Typischerweise wird eine sequenzielle Verarbeitung mithilfe von Clock-Signalen umgesetzt. In VHDL werden hierzu getaktete Prozesse verwendet, sodass bei einer Flanke bestimmte Aktionen ausführt werden, wie beispielsweise das Inkrementieren eines Zählers oder das Speichern eines Zustands in einem Flipflop. Für die Flankenerkennung kann in VHDL der Makro rising_edge(clock) eingesetzt werden, welcher eine steigende Flanke eines Taktsignals erkennt. In Abbildung 4.17 ist der Makro grafisch als Komponente mit dem Eingang clock dargestellt. Wird eine steigende Flanke des Taktsignals erkannt, dann ist der Ausgang „wahr“ (1) andernfalls „falsch“ (0).

Der Makro stellt eine Elementarkomponente dar, sodass dieser nur eine Verhaltensentropie hat. Das Clock-Signal und die Ausgabe haben jeweils zwei mögliche Zustände. Somit ergibt sich für den Makro eine Verhaltensentropie von $H_B(\text{Makro}) = 2 \cdot \log(2) \approx 0{,}60$.

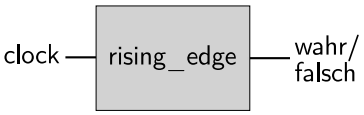


Abbildung 4.17: Rising-Edge-Makro

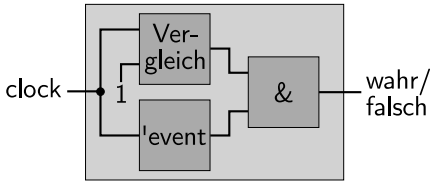


Abbildung 4.18: Flankenerkennung

Die Flankenerkennung kann auch mithilfe eines Vergleichs, einer Eventerkennung und eines Und-Gatters entworfen werden, wie Abbildung 4.18 zeigt. Die Eventerkennung ('event) detektiert eine Zustandsänderung des Eingangssignals. Wenn das Clock-Signal von 0 auf 1 wechselt, sind beide Ausgänge (von 'event und clock = '1') wahr, das heißt, beide Ausgänge sind eins. Durch die Und-Verknüpfung der beiden Ausgänge ist dann der Ausgang der gesamten Komponente eins. Der zugehörige VHDL-Code lautet: IF(clock = '1' AND clock'event) THEN ... END IF.

Die aus den drei Bausteinen aufgebaute Komponente der Flankenerkennung ist keine Elementarkomponente, da diese weitere Komponenten enthält und innerhalb des Entwurfs aufgebaut wurde. Somit hat die Flankenerkennungskomponente sowohl eine Verhaltens- als auch eine Strukturentropie. Die Verhaltensentropie der Flankenerkennung ist $H_B(\text{Flankenerkennung}) = 2 \cdot \log(2) \approx 0{,}60$. Sie entspricht

dabei der des Rising-Edge-Makros, da die Komponente von außen betrachtet ebenfalls einen Clock-Eingang mit zwei möglichen Zuständen und einen Ausgang mit zwei möglichen Zuständen hat.

Die Strukturentropie berechnet sich mithilfe der Verhaltensentropie derjenigen Komponenten, welche für den Aufbau verwendet wurden. Der Vergleich (clock = 1) hat zwei Eingänge, wobei einer der Eingänge mit einer Konstanten (1) beschaltet ist. Der Ausgang kann wiederum zwei Zustände annehmen (0 und 1). Die Komponente, welche eine Flanke erkennt ('event), hat einen Eingang mit zwei möglichen Clock-Zuständen und einen Ausgang mit zwei möglichen Zuständen. Das Und-Gatter hat zwei Eingänge mit je zwei möglichen Zuständen und einen Ausgang mit den möglichen Zuständen „wahr“ (1) oder „falsch“ (0). Somit ergibt sich für die Strukturentropie der Flankenerkennung:

$$\begin{aligned} H_S(\text{Flankenerkennung}) &= H_B(\text{Vergleich}) + H_B(\text{'event}) + H_B(\text{AND}) \\ &= 2 \cdot \log(2) + 2 \cdot \log(2) + 3 \cdot \log(2) \\ &= 7 \cdot \log(2) \approx 2{,}11 \end{aligned}$$

Der Aufbau der Flankenerkennung mit den drei Bausteinen zeigt, dass auch bei der Verhaltensbeschreibung eine Struktur mit Komponenten im Sinne der Entwurfsentropie möglich ist. Die von außen betrachtete Komponente hat eine Verhaltensentropie, wobei der Aufbau der Komponente eine Strukturentropie hat. Die Bausteine (Vergleich, 'event, AND) sind Elementarkomponenten im Sinne der Entwurfsentropie. Diese werden miteinander in Bezug gesetzt, sodass eine Kommunikation zwischen diesen ermöglicht wird. Dadurch wird die neue Komponente „Flankenerkennung“ aufgebaut.

Zusammenfassend findet die Verhaltensentropie immer dann Anwendung, wenn eine Komponente verwendet wird, unabhängig in welcher Domäne und auf welcher Entwurfsebene. Die Strukturentropie findet immer dann Anwendung, wenn der Aufbau einer Komponente analysiert werden soll, das heißt, wenn die Komponente weitere Unterkomponenten enthält, welche innerhalb des zu analysierenden Systems zum Aufbau der Komponente miteinander verbunden wurden. Das Zusammenfügen von Komponenten zu einer neuen, übergeordneten Komponente kann ebenfalls in allen Domänen und auf unterschiedlichen Abstraktionsebenen erfolgen, sodass immer die Strukturentropie Anwendung findet.

4.8 Sequenzielle Elemente

Bei der Berechnung der Anzahl möglicher Zustände besitzen sequenzielle Elemente und Schaltungen die Besonderheit, dass die Ausgangssignale auch von den inneren Zuständen einer Schaltung abhängen [191, S. 222]. Dabei besteht eine sequenzielle Schaltung sowohl aus logischen Schaltnetzen als auch aus speichernden Elementen [205, S. 139], wie Abbildung 4.19 zeigt. Zur Speicherung von digitalen Informationen werden insbesondere Latches, Flipflops (bistabile Kippstufen) und Register eingesetzt [205, S. 139].

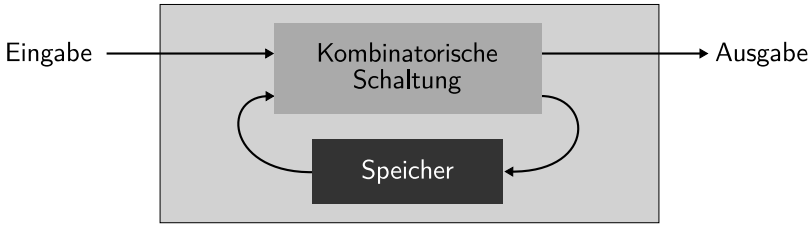


Abbildung 4.19: Aufbau einer sequenziellen Schaltung [nach 191, S. 222]

Die Entwurfsentropie berücksichtigt innere Zustände am Ausgang eines Bauelements, wenn diese für die Umwelt sichtbar sind. Die inneren Zustände lassen sich anschaulich anhand einer Wahrheitstabelle darstellen. Als Beispiel dient ein D-Latch, dessen Symbol in Abbildung 4.20 gezeigt ist. Am Ausgang Q liegt der gespeicherte Wert Q_0 an, wenn das Clock-Signal C null ist. Ist das Signal C dagegen eins, dann liegt am Ausgang Q der Wert des Eingangs D an. Dieses Verhalten ist als Wahrheitstabelle in Tabelle 4.1 gegeben.

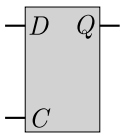


Abbildung 4.20: Symbol eines D-Latches [nach 56, S. 218]

C	0	0	1	1
D	0	1	0	1
Q	Q_0	Q_0	0	1

Tabelle 4.1: Wahrheitstabelle eines D-Latches [nach 56, S. 218]

Die Wahrheitstabelle zeigt, dass am Ausgang Q der zusätzliche (innere) Zustand Q_0 anliegen kann. Somit werden drei mögliche Zustände für den Ausgang Q berücksichtigt. Dabei können auch weitere Zustände am Ausgang anliegen. Beispielsweise kann bei einem RS-Latch ein undefinierter Zustand durch eine potenzielle Racecondition auftreten [56, S. 214 f.]. Dadurch hat das in Abbildung 4.21 gezeigte RS-Latch in der zugehörigen Wahrheitstabelle (Tabelle 4.2) als weiteren Zustand den undefinierten Zustand X [56, S. 214 f.]. Liegt an den Eingängen R und S jeweils eine Eins an, dann ist der Ausgang Q undefiniert (X). Somit kann der Ausgang Q des RS-Latches insgesamt vier mögliche Zustände annehmen: 0, 1, Q_0 und X.

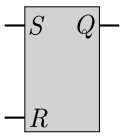


Abbildung 4.21: Symbol eines RS-Latches [nach 56, S. 218]

S	0	0	1	1
R	0	1	0	1
Q	Q_0	0	1	X

Tabelle 4.2: Wahrheitstabelle eines RS-Latches [nach 56, S. 213]

Durch die Berücksichtigung der inneren Zustände am Ausgang kann das Verhalten unterschiedlicher sequenzieller Elemente berücksichtigt werden. Dies wird auch durch Definition 15 für die Zustände ausgedrückt, da die Anzahl möglicher Zustände auch diejenigen Zustände umfasst, „welche zur vollständigen Beschreibung der momentanen Eigenschaften einer Schnittstelle erforderlich sind“. Die Beschreibung der momentanen Eigenschaft einer Schnittstelle umfasst folglich auch die inneren Zustände, sofern diese für die Umwelt sichtbar sind, das heißt, dass diese an dem Ausgang anliegen können. Somit berechnet sich die Entwurfsentropie auch für sequenzielle Komponenten anhand der Zustände der Schnittstellen.

Die vorherigen Darstellungen haben gezeigt, dass die Berechnung der Entwurfsentropie immer nach den gleichen Regeln erfolgt, unabhängig von der Entwurfsebene und Domäne. Um den Bedürfnissen einer schnellen Analyse gerecht werden, wurde ein Analysewerkzeug entwickelt, welches im folgenden Abschnitt vorgestellt wird.

4.9 Analysewerkzeug

Das Analysewerkzeug ermöglicht die automatisierte Berechnung der Entwurfsentropie. Als Eingabe dienen Beschreibungen von Hardwareentwürfen, da der Fokus der vorliegenden Arbeit auf der Analyse von Hardwareprojekten liegt. Als Grundlage stehen verschiedene Hardwarebeschreibungssprachen zur Verfügung. Zu den bekanntesten und am weitest verbreiteten Sprachen gehören Verilog und VHDL [117, S. 24]. Diese Computersprachen wurden entwickelt, um die Struktur, Konstruktion und den Betrieb elektrischer Schaltungen und insbesondere digitaler Logikschaltungen zu beschreiben [122, S. 244]. Gegenüber Softwaresprachen unterscheiden sich Hardwarebeschreibungssprachen grundsätzlich darin, dass Nebenläufigkeiten von Hardwarekomponenten beschrieben werden können und das Zeitverhalten modelliert werden kann [132, S. 65]. Dies erlaubt eine Synthese der Realisierungen in der geplanten Zieltechnologie [122, S. 244].

Neben den Hardwarebeschreibungssprachen werden auch Systembeschreibungssprachen eingesetzt, in welchen ein Entwurf auf einer höheren Ebene spezifiziert werden kann [122, S. 243]. Zu den Bekanntesten zählen SystemC und Matlab/Simulink. SystemC erlaubt die Beschreibung eines Entwurfs auf der Systemebene [132, S. 87], wobei zunächst keine Trennung zwischen dem Hardware- und Softwareteil erfolgt [122, S. 277]. Matlab/Simulink setzt auf einer sehr hohen Abstraktionsebene an, um die Systemanforderungen und die Signalverarbeitungsalgorithmen zu beschreiben [22, S. 15]. Beim Entwurf auf einer hohen Abstraktionsebene sind zusätzlich Hochsprachensynthesewerkzeuge (High-Level Synthesis – HLS) erforderlich, um eine RTL-Beschreibung des Hardwareentwurfs zu erzeugen [117, S. 13].

Abbildung 4.22 ordnet die hier angesprochenen Sprachen im Bezug auf deren Einsatzfeld ein. Sie zeigt, dass SystemC und Matlab/Simulink eine sehr abstrakte Beschreibung ermöglichen. Die Entwurfsentropie soll im Rahmen der vorliegenden Arbeit für Hardwareentwürfe berechnet werden, welche als Verhaltens- und Strukturbeschreibungen auf der Gatter- und Registertransferebene gegeben sind. Wie die Grafik zeigt, eignet sich insbesondere VDHL als Beschreibungssprache, da diese alle verwendeten Ebenen und Domänen abdeckt. Folglich wurde das Analysewerkzeug so entwickelt, dass VHDL-Hardwareentwürfe in der Verhaltens- und Strukturdomäne analysiert werden können [[147]].

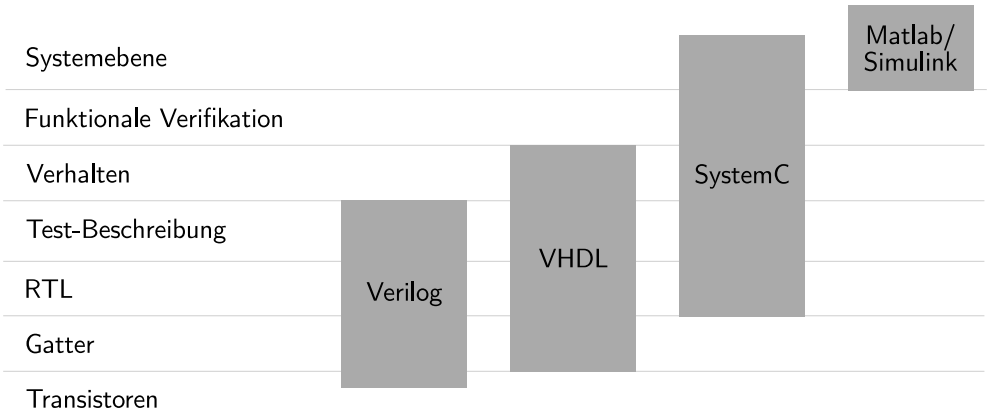


Abbildung 4.22: Vergleich verschiedener Entwicklungssprachen (nach [22, S. 15] und [122, S. 279])

VHDL (Very High Speed Integrated Circuit Hardware Description Language – VHDL) ist seit 1987 im IEEE-Standard 1076 [92] definiert. Die aktuelle Überarbeitung stammt aus dem Jahr 2008 [95]. Komponenten werden als Entitäten (entity oder design entity) modelliert, welche die Entitätsdeklaration sowie eine oder mehrere Architekturen enthalten [132, S. 66]. Die Anschlüsse (Ports) ermöglichen es, eine Komponente innerhalb anderer Komponenten zu verschalten [117, S. 25]. Die Anschlüsse sind entweder Eingänge, Ausgänge oder bidirektionale Verbindungen [117, S. 25]. Die Architektur beschreibt die innere Funktion (Verhaltensbeschreibung) oder die Struktur (Strukturbeschreibung) einer Komponente [117, S. 27].

Die oberste Einheit, welche die Verbindung zur Außenwelt darstellt und alle weiteren Entitäten enthält, wird Top-Level-Entität genannt. Sie definiert die Ein- und Ausgänge des Systems zur Außenwelt, wie Schnittstellen zu Schaltern oder Anzeigeelementen. Jede Entität kann wieder andere Entitäten enthalten, sodass sich hieraus eine hierarchische Struktur ergibt. Dabei können Entitäten auch wiederverwendet werden, das heißt, dass eine Entität mehrfach verwendet wird. Auch können fremde oder geschlossene Entitäten eingebunden werden.

Die Umsetzung des Analysewerkzeugs basiert auf den Grammatikregeln von VHDL, welche im IEEE-Standard 1076 [95] als EBNF (Erweiterte Backus-Naur-Form) beschrieben sind. Das Analysewerkzeug verarbeitet VHDL-Beschreibungen auf Basis eines Compiler-Frontends (ausführlich zum Compiler zum Beispiel [2, S. 7 ff.] und [244, S. 3 ff.]). Es beginnt mit der Top-Level-Entität und nimmt den Namen der VHDL-Beschreibung als Eingabe [[147]]. Die Arbeitsweise des Analysewerkzeugs ist in Abbildung 4.23 gezeigt und wird im Folgenden beschrieben.

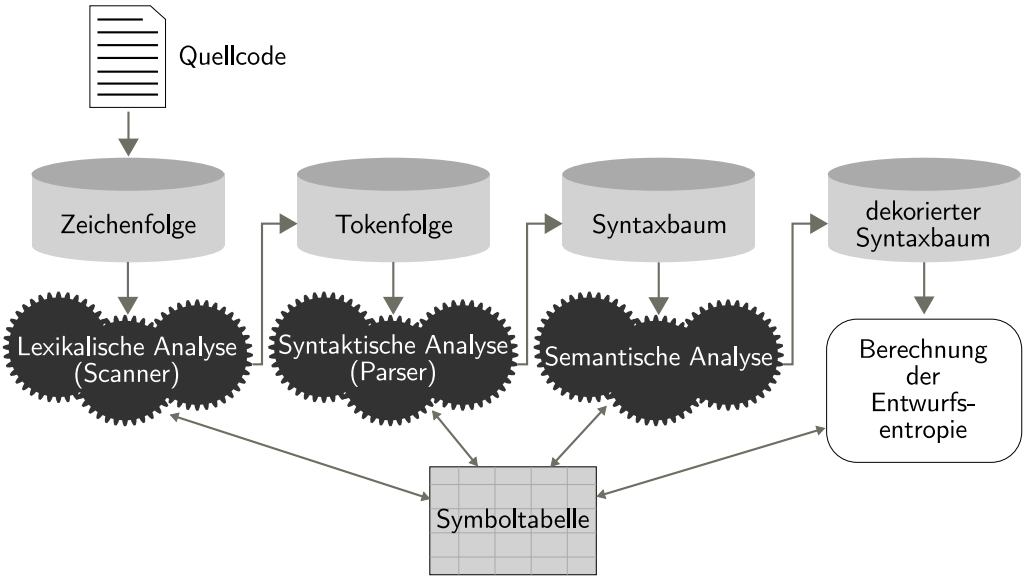


Abbildung 4.23: Arbeitsweise des Analysewerkzeugs [[147]]

Der VHDL-Quelltext wird als Zeichenfolge eingelesen und von der lexikalischen Analyse (Scanner) in lexikalische Einheiten (Tokens) gruppiert [[147]]. Dabei werden die Symbole und weitere Informationen in einer Symboltabelle gespeichert [[147]]. Anhand der Tokenfolge kann die syntaktische Analyse den (hierarchischen) Syntaxbaum erzeugen, wobei die Bedeutung einiger Tokens nur aus dem Kontext erkennbar ist, sodass ein (semantisches und syntaktisches) Vorausschauen (lookahead) erforderlich ist [[147]]. Diese Zuordnung der Tokens übernimmt die semantische Analyse, welche als Ergebnis einen dekorierten Syntaxbaum liefert [[147]].

Um die einzelnen Teile des Compiler-Frontends zu erstellen, wird ein Compiler-Compiler (Parsergenerator) eingesetzt, der den Aufbau erleichtert [2, S. 343]. Ein Parsergenerator erzeugt nicht nur den Parser selbst, sondern meist auch den Scanner und die semantische Analyse [2, S. 15, 1199]. Als Quellen dienen die formalen Beschreibungen der Syntax und der Semantik einer Sprache [194, S. 11], im vorliegenden Fall von VHDL.

Als Parsergenerator wird ANTLR (ANother Tool for Language Recognition – ANTLR) in der Version 4.5.1 [165] eingesetzt. Die Basis der Grammatik bildet der VHDL-Standard 1076 von 2008 [95], welcher um weitere Regeln für die lexikalische Analyse erweitert wurde [[147]]. In die Grammatikdatei können Prozesse (actions) eingebracht werden, welche beim Übersetzerbau normalerweise dazu verwendet werden, den Quelltext in die Zielsprache zu übersetzen [164, S. 4]. Anstatt den Quelltext zu übersetzen, verwendet das Analysewerkzeug die Prozesse für die

Berechnung der Entwurfsentropie [[147]]. Mithilfe von geschweiften Klammern können die Prozesse direkt in die Grammatikdatei in der Zielsprache, im vorliegenden Fall Java, aufgenommen werden [164, S. 78]. Dabei kann auf Tokens (Attribute) zugegriffen werden [164, S. 81]. Die grundlegende Funktionsweise, wie Prozesse in die Grammatikdatei eingefügt werden, wird anhand der folgenden Signaldefinition gezeigt [[147]]:

```
1        SIGNAL a,b : BIT_VECTOR(1 downto 0);
```

Code 4.4: VHDL-Signaldefinition

Für die Berechnung der Entwurfsentropie muss zunächst gespeichert werden, welche möglichen Zustände ein Signal hat. Hierzu wird die Signaldefinition analysiert. Code 4.4 legt fest, dass die Signale a und b Bit-Vektoren mit jeweils zwei Bit sind. Somit kann jedes Signal vier mögliche Zustände annehmen (00, 01, 10, 11). Das Analysewerkzeug nutzt die EBNF-Beschreibung der VHDL-Grammatik, um den Quelltext (VHDL-Code) zu parsen. Auf Basis der Grammatik ergibt sich für die Signaldefinition der in Abbildung 4.24 dargestellte Syntaxbaum (parse tree), wobei nur die verwendeten Pfade dargestellt sind.

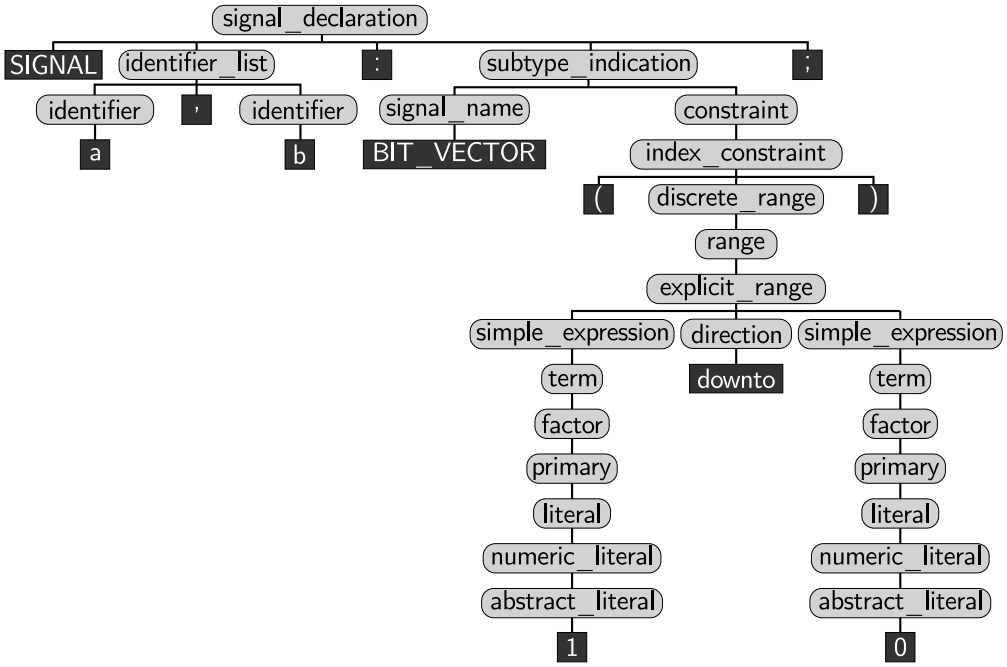


Abbildung 4.24: Syntaxbaum einer VHDL-Signaldefinition [[147]]

Die Wurzel der Signaldefinition ist die Grammatikregel signal_declaration. Diese besteht aus dem Token SIGNAL und der identifier_list, auf welche ein Doppelpunkt folgt. Die subtype_indication definiert im vorliegenden Fall die Art des Signals. Beendet wird die Anweisung mit einem Semikolon. Die einzelnen Grammatikregeln enthalten wiederum weitere Unterregeln. Um zu demonstrieren, wie Prozesse in Form von Java-Anweisungen in die Grammatikregeln eingefügt werden können, sind einige Regeln der Grammatikdatei des Analysewerkzeugs in verkürzter und vereinfachter Form im folgenden Code 4.5 gezeigt.

```
1  signal_declaration
2    : SIGNAL identifier_list COLON
3    subtype_indication ( signal_kind )? ( VARASGN expression )? SEMI
4    {                                                          Parsen der Signalnamen
5      String input = $identifier_list.text;
6      String[] parts = input.split(Pattern.quote(","));
7      for(int i = 0; i< parts.length; i++ )
8      { a.add(parts[i],0,$subtype_indication.states,$subtype_indication.width); }
9    }
10   ;
11
12 subtype_indication returns [int states, int width]
13   : signal_name { $states=$signal_name.states; }
14     ( signal_name { $states+=$signal_name.states; } )?
15     ( constraint { $width += $constraint.width; })?
16     ( tolerance_aspect { error("tolerance_aspect not implemented yet); } )?   Fehlermeldung
17   ;
18
19 signal_name returns [int states]                                  Vereinfacht
20   : BIT { $states = 2; }
21   | BITVECTOR { $states = 2; }
22   ...                                                           Gekürzt
23    ;
24
25 explicit_range returns [int width]
26   : value1 = simple_expression direction value2 = simple_expression
27   {$width = Math.abs($value1.value − $value2.value) + 1;}
28   ;
```

Code 4.5: Grammatikregeln mit Prozess-Anweisungen [[147]]

Basis der Signaldefinition ist die Grammatikregel signal_declaration in den Zeilen 1 – 10 von Code 4.5. Die Regel enthält im Attribut identifier_list die Namen der Signale. Dieses Attribut wird in den Zeilen 4 – 8 mithilfe von Java-Anweisungen analysiert (geparst). Da eine Signaldefinition mehrere, durch Kommas getrennte Signale enthalten kann, wird die Zeichenfolge an den Kommas aufgetrennt (Zeile 6) und für jedes Signal die Anzahl möglicher Zustände gespeichert (Zeile 8). Das Speichern erfolgt mithilfe einer „globalen“ Tabelle (Hashmap), welche zum einen die einzelnen Signale mit den möglichen Zuständen enthält und zum anderen die berechneten Werte aller Komponenten eines Systems [[147]].

Um die Anzahl möglicher Zustände abspeichern zu können, hat die Regel subtype_indication in den Zeilen 12 – 17 als Rückgabewerte die Anzahl an Zuständen (states) und die Breite des Bitvektors (width) [[147]]. Die Grammatikregel enthält zudem in Zeile 16 eine Anweisung, welche für eine TOLERANCE-Anweisung einen Fehler ausgibt, da hinter dieser Grammatikregel noch keine Berechnungen hinterlegt sind. Somit ist es bei der Analyse direkt erkennbar, ob in einem VHDL-Quellcode Anweisungen verwendet werden, die vom Analysewerkzeug bis jetzt noch nicht analysiert werden können. Zwar ist bereits die vollständige VHDL-Grammatik hinterlegt, jedoch bricht die Analyse ab, wenn hinter bestimmten Grammatikregeln noch keine Berechnungen hinterlegt sind, da in diesem Fall keine zuverlässige Aussage über die Entwurfsentropie möglich ist [[147]]. Dadurch, dass die vollständige Grammatik bereits umgesetzt ist, kann der Code effizient um weitere Berechnungen erweitert werden [[147]]. Bei der Anwendung des Analysewerkzeugs kommt es in der Regel zu keinem Abbruch, da nur bei sehr selten verwendeten Anweisungen (wie beispielsweise der Tolerance-Anweisung) keine Berechnungen hinterlegt sind [[147]].

Die subtype_indication-Regel greift bei der Bestimmung der Anzahl an Zuständen auf die Rückgaben der Regeln signal_name in den Zeilen 19 – 23 und explicit_range in den Zeilen 25 – 27 zurück [[147]]. Hierzu haben die Regeln constraint, index_constraint, discrete_range und range ebenfalls den Wert für die Breite als Rückgaben [[147]]. Mithilfe der Informationen der Blätter, dass der Bit-Vektor von 1 bis 0 geht, kann innerhalb der Regel explicit_range die Breite berechnet werden. Zeile 8 der Regel signal_declaration liegen nun alle Informationen vor, um die Signale mit der möglichen Anzahl an Zuständen in die Tabelle (Hashmap) einzufügen [[147]]. Die add-Funktion wird dabei von der Tabellenklasse bereitgestellt [[147]].

Der Parsergenerator übersetzt den in der Grammatik enthaltenen Java-Code mit und erzeugt zwei Hauptklassen: den Parser und den Scanner (Lexer) [[147]]. Beide Klassen werden vom Java-Hauptprogramm eingebunden, das zur Analyse aufgerufen wird [[147]]. Das Hauptprogramm Run enthält die Methode calculate(String name), welcher der Name der VHDL-Datei übergeben wird [[147]]. Anhand des Namens wird zunächst überprüft, dass die Strukturentropie nicht bereits berechnet wurde [[147]]. Dann wird geprüft, ob die VHDL-Datei vorhanden ist [[147]]. Ist keine zugehörige VHDL-Datei vorhanden, geht die Analyse davon aus, dass es sich um eine wiederverwendete Komponente oder eine Elementarkomponente handelt, für welche die Strukturentropie null ist [[147]]. Zudem können Dateien mit dem Kommentar „-- Reused“ als wiederverwendet gekennzeichnet werden, sodass auch

in diesem Fall die Strukturentropie null ist [[147]]. Abbildung 4.25 zeigt die Ausgabe des Analysewerkzeugs für einen Volladdierer mit einem wiederverwendeten Halbaddierer.

```
$ java Run volladdierer.vhd
--> Starting analysis with file: volladdierer.vhd
--> Starting analysis with file: halbaddierer.vhd

Finished analysis: Structure entropy of all files:
================================================
halbaddierer.vhd                          (Reused)
volladdierer.vhd                             3.31
================================================
--> Design Entropy for input file volladdierer.vhd: 3.31
```

Abbildung 4.25: Ausgabe des Analysewerkzeugs bei wiederverwendeten Komponenten [[147]]

Wird auf Basis des dekorierten Syntaxbaums eine Entität gefunden, welche noch nicht analysiert wurde und die nicht wiederverwendet wurde, wird eine neue Instanz der Entwurfsentropieberechnung (Run.calculate($id.text+".vhd")) erzeugt, welche den Namen der gefundenen Entität als Eingabe hat [[147]]. Dieser „rekursive“ Aufruf der Berechnung ist in Code 4.6 gezeigt. Die Berechnung der Entwurfsentropie dieser Komponente wird dann auf Basis des jeweiligen Syntaxbaums durchgeführt [[147]].

```
1  component_declaration
2    : COMPONENT id = identifier ( IS )?
3      {
4          Run.calculate($id.text+".vhd");        Rekursiver Aufruf mit der Unterkomponente
5      }
6      ( generic_clause
7      ( component_port_clause )?
8      END COMPONENT ( identifier )? SEMI
9    ;
```

Code 4.6: Rekursiver Aufruf der Berechnung [[147]]

Dem kompilierten Hauptprogramm Run wird entweder die zu analysierende Datei als Argument übergeben oder der Name der VHDL-Datei nach Aufforderung eingegeben [[147]]. Besteht das VHDL-Projekt aus mehreren Dateien, dann ist der Name des Systems (top-level entity) zu übergeben [[147]]. Enthaltene Unterkomponenten werden durch den oben beschriebenen Aufruf der Berechnungsmethode vom Programm selbst analysiert, bis das Ergebnis für das komplette System vorliegt [[147]].

Innerhalb der vorliegenden Arbeit wird das Analysewerkzeug dazu eingesetzt, die Entwürfe der Fallstudien in Kapitel 6 zu analysieren. Um die Berechnungen der Entwurfsentropie für digitale Hardwareentwürfe anschaulich darzustellen, wird im folgenden Kapitel zunächst auf den Einsatz der automatischen Analyse verzichtet. Dabei kann anhand kompakter Beispiele die Funktionsweise der Entwurfsentropieberechnung gezeigt werden. Für umfangreichere Entwürfe finden dabei automatisiert die gleichen Berechnungen statt. Jedoch sind durch den Entwurfsumfang und die Kombination der im folgenden Kapitel einzeln gezeigten Aspekte umfangreichere Entwürfe nicht mehr anschaulich darstellbar, sodass die Darstellung anhand kompakter Beispiele gewählt wurde. Zunächst werden in den folgenden beiden Abschnitten noch zwei Exkurse vorgestellt: die Zustandsreduktion und die Anwendung auf Software.

4.10 Exkurs: Zustandsreduktion

Im Vorherigen wurde für die Anzahl an Zuständen die Anzahl maximal möglicher Zustände betrachtet. Möglich sind nach Definition 15 diejenigen Zustände, „welche eine Schnittstelle aufnehmen oder abgeben kann und welche zur vollständigen Beschreibung der momentanen Eigenschaften einer Schnittstelle erforderlich sind“. Im vorherigen Abschnitt wurde bei der Beschreibung des Analysewerkzeugs gezeigt, dass sich bei der Hardwareanalyse die Anzahl möglicher Zustände insbesondere aus der Signaldeklaration ergibt. Für ein Signal mit n Bit sind es 2^n mögliche Zustände, wenn ein Bit die Zustände 0 und 1 annehmen kann.

Für die Bestimmung der Anzahl möglicher Zustände ist es dabei erforderlich, dass die Zustände auch tatsächlich erreicht werden können. Denn ein nicht erreichbarer Zustand ist kein möglicher Zustand einer Schnittstelle. Einen Unterfall stellen Schnittstellen mit nur einem möglichen Zustand dar. Wird ein Eingang einer Komponente mit einer „Konstanten“ beschaltet, das heißt, ein Signal dauerhaft auf „0“ oder „1“ gelegt oder statt einer Variablen eine Konstante verwendet, dann ergibt sich nur ein möglicher Zustand für diesen Eingang. Für einen möglichen Zustand ist die Verhaltensentropie dieses Eingangs null ($\log(1) = 0$), wie sich aus Gleichung (4.10) für die Verhaltensentropie ergibt. Dies wird auch anhand von Definition 15 für die Zustände deutlich, welche besagt, dass diese keine „unveränderliche Eigenschaft der Schnittstelle darstellen“ dürfen. Somit trägt eine dauerhafte und konstante Beschaltung nicht zur Entwurfsentropie bei.

Konstante Signale werden bei Hardwarebeschreibungen regelmäßig für Vergleiche oder die Beschaltung einzelner Eingänge verwendet. Dies führt grundsätzlich aber nicht dazu, dass auch gleichzeitig die Verhaltensentropie einer Komponente null wird, sondern nur, dass die Verhaltensentropie geringer wird. Hieraus ergibt sich, dass es bei einem Entwurf aufwendiger ist, die Verdrahtung von Signalen vorzunehmen, welche verschiedene Zustände annehmen können als die Verschaltung von Signalen, welche nur einen möglichen Zustand haben. Es folgt somit dem Grundansatz der Entwurfsentropie, dass die Komplexität eines Entwurfs durch die Realisierung des Nachrichtenaustauschs zwischen den Komponenten entsteht.

Beim Entwurf werden auch Signale oder Variablen deklariert, die mehr Zustände als die für die Beschreibung notwendige Anzahl an Zuständen annehmen können. Insbesondere beim Hardwareentwurf werden regelmäßig Signale als sogenannte mehrwertige Logik deklariert. Diese mehrwertige Logik erlaubt es, beispielsweise die Stärke eines Signals zu berücksichtigen [132, S. 68]. Auch können unbestimmte oder hochohmige Zustände mit diesen Signalen modelliert werden [185, S. 73 ff.]. In VHDL wird die mehrwertige Logik durch den IEEE-Standard 1164 [94] ausgedrückt, welcher mittlerweile in die aktuelle Überarbeitung des VHDL-IEEE-Standards 1076-2008 eingeflossen ist [95, S. 1]. Die Standardlogik (std_(u)logic) umfasst dabei die neun in Tabelle 4.3 dargestellten Zustände [95, S. 514].

X	Unbekannt	L	Schwache 0
0	Logische 0	H	Schwache 1
1	Logische 1	U	Nicht initialisiert
Z	Hochohmig	-	Beliebige Eingabe
W	Schwaches Signal		

Tabelle 4.3: Standardlogik [95, S. 514]

Beim Entwurf eines Systems macht es für die Komplexität einen Unterschied, ob eine binäre Logik mit zwei Zuständen verwendet wird oder ob zusätzliche Zustände berücksichtigt werden [[147]]. Dabei kommt es auch vor, dass eine Person innerhalb eines Entwurfs durchgängig eine Logik mit neun Zuständen für mehrere Treiber (std_logic) verwendet, obwohl binäre Zustände ausreichend wären [[147]]. Dies ist für die Entwurfsentropie dahingehend unproblematisch, da sich durch die logarithmische Funktion die (konstant) höhere Zustandsanzahl nur als „Skalierungsfaktor“

auswirkt [[147]]. Somit muss beim Vergleich verschiedener Implementierungen gegebenenfalls nur ein Skalierungsfaktor berücksichtigt werden [[147]]. Dieser ergibt sich aus der Basisumrechnung des Logarithmus: $\log_b(x) = \frac{\log_{10}(x)}{\log_{10}(b)}$.

Anders gestalten sich die Fälle, wenn unterschiedliche Signaldeklarationen verwendet werden. Wird beispielsweise eine zweiwertige Logik für die einen Signale verwendet und für andere Signale eine mehrwertige Logik, dann wirkt sich dies nicht mehr als lineare Skalierung auf den Entwurf aus, welcher mithilfe eines einheitlichen Skalierungsfaktors umgerechnet werden kann. Durch die Beachtung der möglichen Zustände hat dies einen entscheidenden Einfluss auf die Entwurfsentropie.

Als Lösung hierfür kann statt der Anzahl aller möglicher Zustände nur die Anzahl derjenigen Zustände zugrunde gelegt werden, welche für die Umsetzung des Entwurfs erforderlich waren. Diese Zustände werden zur Abgrenzung im Folgenden als Entwurfszustände bezeichnet. Der Unterschied zwischen den zwei Anzahlen wird anhand der beiden folgenden Beispielcodes verdeutlicht. Beide Beispiele zeigen einen minimalistischen Zähler (ohne Taktung oder Ähnliches). Der Wert wird jeweils um eins erhöht. Das heißt, ist der Wert 0 beziehungsweise "00", dann wird er zu 1 beziehungsweise zu "01" und so weiter. Ist der Wert 3 beziehungsweise "11" erreicht, wird der Zähler auf 0 beziehungsweise auf "00" zurückgesetzt. Im linken Code 4.7 wird für den Zähler ein Bit-Vektor mit der Breite zwei verwendet. Dadurch hat das Signal count 2^2 mögliche Zustände. Im rechten Code 4.8 wird stattdessen eine Integervariable als Zähler verwendet, sodass das count-Signal 2^{32} mögliche Zustände hat. Die Case-Anweisungen sind gleichwertig bis auf den Unterschied, dass ein Mal die Werte als Vektoren angegeben sind ("00" ... "11") und das andere Mal als Integerwerte (0 ... 3).

```
1  ARCHITECTURE behavior OF automat IS
2  SIGNAL count: BIT_VECTOR(0 TO 1);
3
4  BEGIN
5   PROCESS
6   BEGIN
7    CASE count IS
8     WHEN "00" =>        count <= "01";
9     WHEN "01" =>        count <= "10";
10    WHEN "10" =>        count <= "11";
11    WHEN OTHERS =>      count <= "00";
12   END CASE;
13  END PROCESS;
14 END behavior;
```

Code 4.7: Bit-Vektor-Zähler

```
1  ARCHITECTURE behavior OF automat IS
2  SIGNAL count : INTEGER;
3
4  BEGIN
5   PROCESS
6   BEGIN
7    CASE count IS
8     WHEN 0 =>           count <= 1;
9     WHEN 1 =>           count <= 2;
10    WHEN 2 =>           count <= 3;
11    WHEN OTHERS =>      count <= 0;
12   END CASE;
13  END PROCESS;
14 END behavior;
```

Code 4.8: Integer-Zähler

Die Verhaltensentropie der beiden Case-Anweisungen errechnet sich über die Vergleiche, wie in Abschnitt 4.6 über die Verhaltensdomäne gezeigt. Für den linken Code mit dem Bit-Vektor ist die Anzahl an Zuständen des Eingangssignals $z(e) = 2^2 = 4$. Hieraus ergibt sich nach folgender Rechnung eine Verhaltensentropie von $H_B(\text{Bit-Vektor}) \approx 1{,}81$. Bei beiden Codes ist die Anzahl an Ausgängen $n = 4$.

$$H_B(\text{Bit-Vektor}) = \log\left(z(e) \cdot 2^n\right) = \log\left(4 \cdot 2^4\right) \approx 1{,}81$$

Für den rechten Code, der statt des Bitvektors eine Integervariable verwendet, ist die Anzahl an Eingangszuständen $z(e) = 2^{32}$, da der Integerwert 32 Bit breit ist [95, S. 39]. Nach der folgenden Berechnung ergibt sich eine Verhaltensentropie von $H_B(\text{Integer}) \approx 10{,}84$.

$$H_B(\text{Integer}) = \log\left(z(e) \cdot 2^n\right) = \log\left(2^{32} \cdot 2^4\right) \approx 10{,}84$$

Wird statt der Anzahl aller möglicher Zustände ($= 2^{32}$) nur die Anzahl derjenigen Zustände zugrunde gelegt, welche für den Entwurf erforderlich waren, das heißt, die Entwurfszustände, kommt es zu einem Gleichlauf der Berechnungen. Denn für den rechten Code mit der Integervariablen sind von den 2^{32} möglichen Zuständen nur $2^2 = 4$ Zustände tatsächlich verwendet worden. Somit hat die Variable des rechten Codes nur 2^2 Entwurfszustände. Anders formuliert geht es bei den Entwurfszuständen darum, welche der (theoretisch) möglichen Zustände beim Entwurf tatsächlich betrachtet wurden. Denn nur für diese Zustände ist beim Entwurf Aufwand angefallen, sodass nur Arbeit für die Umsetzung dieser Zustände aufgewendet wurde. Für die nicht betrachteten/erforderlichen Zustände wurde keine Arbeit für die Umsetzung aufgewendet. Der Entwurf enthält somit nur diejenige Komplexität, welche durch die Entwurfszustände in den Entwurf eingebracht wurde. Dies kann auch auf den nachrichtentechnischen Ansatz zurückgeführt werden, da Zustände, welche nicht erforderlich sind beziehungsweise nicht umgesetzt wurden, mit einer Wahrscheinlichkeit von null auftreten. Sie tragen somit nicht zur Entropie einer Komponente bei. Anhand der vorherigen Darstellungen kann die folgende Definition 20 für die Entwurfszustände abgeleitet werden.

Definition 20 (Entwurfszustände)

Die Anzahl an Entwurfszuständen ist diejenige Anzahl möglicher Zustände nach Definition 15, welche tatsächlich beim Entwurf betrachtet/umgesetzt wurden, das heißt, welche zur Umsetzung des Entwurfs erforderlich waren.

Durch den Bezug der Definition für die Entwurfszustände zur maximalen Anzahl an möglichen Zuständen ist die Anzahl an Entwurfszuständen immer kleiner oder gleich der Anzahl maximal möglicher Zustände. Handelt es sich um kein konstantes Signal oder eine Konstante, so ist die Mindestanzahl an Zuständen zwei. Denn es besteht die Möglichkeit, das Signal oder die Variable zu verbinden oder diese nicht zu verbinden. Diese zwei Möglichkeiten werden bei einem Entwurf mindestens betrachtet. Erfolgt dagegen keine Verbindung einer Schnittstelle mit einer anderen Schnittstelle, dann ist dies gleich zu setzen mit einem konstanten Signal beziehungsweise mit einer Konstanten, sodass nur ein Zustand betrachtet wird. Das Kriterium der Notwendigkeit in Definition 20 („welche zur Umsetzung des Entwurfs erforderlich waren“) richtet sich dabei nach der tatsächlichen Umsetzung. Das heißt, es ist unbeachtlich, ob und inwieweit die umgesetzten Zustände minimal sind oder ob diese tatsächlich erforderlich waren. Analog zur vorherigen Darstellung, dass es unbeachtlich ist, ob eine Schaltung minimiert werden kann, berechnet die Entwurfsentropie diejenige Komplexität, welche tatsächlich umgesetzt wurde und nicht diejenige Komplexität, mit welcher die Aufgaben des Systems hätten minimal erreicht werden können. Somit kommt es zu einem Gleichlauf der beiden Ansätze hinsichtlich der Frage der Minimierung: Eine mögliche Minimierung ist unbeachtlich, wenn die zusätzliche Komplexität und Quantität in der Beschreibung/ Schaltung enthalten sind.

Für die Bestimmung der Anzahl an Entwurfszuständen kann der Ansatz des Analysewerkzeugs verwendet werden: Wie im vorherigen Abschnitt beschrieben, wird bisher die maximale Zustandsanzahl aus den Signal- und Schnittstellendefinitionen gewonnen. Für jedes Signal wird diese Information in einer Tabelle (Hashmap) gespeichert. Auf Basis des dekorierten Syntaxbaums ist es aber auch möglich, die Anzahl der im Entwurf umgesetzten Zustände (Entwurfszustände) für jedes Signal zu berechnen. Bei jeder Verwendung eines Signals wird analysiert, welche Zustände tatsächlich umgesetzt wurden. Diese zusätzlichen Informationen können in einer erweiterten Tabelle gespeichert und am Ende der Analyse der jeweiligen Komponente ausgewertet werden. Für das Codebeispiel 4.8 mit der Integervariablen als Zähler würde für Zeile 2 zunächst gespeichert werden, dass das Signal `count` deklariert wurde. Im Gegensatz zur Analyse mit der maximalen Anzahl an Zuständen, welche an dieser Stelle abspeichert, dass das Signal 2^{32} mögliche Zustände besitzt, würden zunächst keine Entwurfszustände abgespeichert werden. Aus der Analyse der `case`-Anweisung in den Zeilen 7 – 12 wird die Information gewonnen, dass für das Signal `count` die Zustände „1“, „2“, „3“ und „andere“ betrachtet werden. Diese Information wird in der erweiterten Tabelle gespeichert. Die Auswertung der Anzahl an Entwurfszuständen erfolgt erst am Ende der Komponentenanalyse,

da weitere Anweisungen das count-Signal verwenden können und dadurch weitere Entwurfszustände hinzukommen können. Da der Beispielcode 4.8 keine weiteren Anweisungen enthält, welche das count-Signal verwenden, ergibt die Auswertung, dass dieses Signal vier Entwurfszustände hat. Die Anzahl an Entwurfszuständen fließt dann, statt der möglichen Anzahl an Zuständen, in Formel (4.10) für die Verhaltensentropie ein.

Die Lösung über die Entwurfszustände hat zwei entscheidende Vorteile: Zum einen werden der tatsächliche Aufwand für die Umsetzung sowie die in einer Schaltung enthaltene Komplexität praxisbezogener ausgedrückt, insbesondere in den Fällen, wenn unterschiedliche Signaldeklarationen zu einem nicht linearen Skalierungsfaktor führen. Weiterhin werden einfache Verdrahtungen nicht nach der unterschiedlichen Zustandsanzahl gewichtet: Nach dem Grundansatz ist die Verhaltensentropie der Signalzuweisung a <= b von der Anzahl maximal möglicher Zustände abhängig. Sind die beiden Signale a und b 16-Bit-Vektoren, dann hat jedes Signal 2^{16} mögliche Zustände und die Verhaltensentropie beträgt $H_B = 2 \cdot \log(2^{16})$. Sind die beiden Signale ein Bit breit, dann beträgt die Verhaltensentropie $H_B = 2 \cdot \log(2)$. Dabei ist es weder für die Komplexität noch für den Aufwand der Zuweisung (a <= b) entscheidend, wie viele Zustände die beiden Signale haben. Werden die Entwurfszustände zugrunde gelegt, dann ist die Zuweisung unabhängig davon, wie viele Zustände die Signale maximal haben, da bei der einfachen Zuweisung lediglich die Zustände „verbunden“ und „nicht verbunden“ betrachtet wurden.

Der zweite Vorteil der Entwurfszustände ist die Möglichkeit der Anwendung der Entwurfsentropie auf unterschiedliche Entwicklungsstufen eines Projekts. In einem frühen Entwicklungsstadium kann die Entwurfsentropie mithilfe der Port- und Signaldefinitionen (Schnittstellendefinition) abgeschätzt werden, ohne dass bereits die tatsächliche Umsetzung der Komponenten bekannt ist. Das heißt, es wird zunächst davon ausgegangen, dass alle möglichen Zustände beim Entwurf umgesetzt werden. Während der Projektumsetzung erfolgt nach dem Top-down-Ansatz ein Prozess der Entwicklung durch Verfeinerung, sodass durch die Umsetzung der Komponenten der Entwurf immer detaillierter wird. Dabei kommt es regelmäßig zu einer Zustandsreduktion, da nicht alle möglichen Zustände umgesetzt werden. Das heißt, in einem frühen Projektstadium wird angenommen, dass noch nicht bekannt ist, welche Zustände genau für die Umsetzung erforderlich sind, sodass zunächst von allen möglichen Zuständen ausgegangen wird. Während der detaillierteren Planung und insbesondere während der Umsetzung des Projekts reduziert sich die Anzahl möglicher Zustände auf die Anzahl an Entwurfszuständen.

Insgesamt wird durch die Eingrenzung des Zustandsbegriffs auf die Entwurfszustände regelmäßig eine Zustandsreduktion erreicht, außer in den Fällen, wenn alle möglichen Zustände auch umgesetzt werden. Um die Auswirkungen auf die Berechnung der Entwurfsentropie zu analysieren, wurde das Analysewerkzeug für einen kleinen Teil der Grammatik um die Berechnung der Entwurfszustände erweitert. Damit konnten Schaltungen mit Logikgattern und einfache Zähler untersucht werden. Zur Evaluation wurden von sechs Versuchspersonen jeweils neun Aufgaben in VHDL implementiert. Bei der Analyse zeigten sich außer einer Skalierung der Ergebnisse jedoch kaum Auswirkungen zwischen der Berechnung der Entwurfsentropie mit den Entwurfszuständen und der Berechnung mit der Anzahl maximal möglicher Zustände. Naheliegend ist, dass sich die natürliche Wahl der Signale/Variablen bereits aus der Aufgabenstellung ergab und/oder die Personen einheitlich beispielsweise Integervariablen statt Bit-Signalen verwendeten. Für eine wissenschaftlich und experimentell fundierte Aussage müsste die Analyse von umfangreicheren Entwürfen erfolgen. Da dies im Rahmen der vorliegenden Dissertation noch nicht im wissenschaftlich erforderlichen Umfang erfolgt ist, wurde die Zustandsreduktion als Exkurs aufgenommen. In zukünftigen Forschungen soll dieser Ansatz weiterverfolgt werden.

Die vorherigen Ausführungen zeigen zugleich die Grenzen der Entwurfsentropie und der Aufwandsbestimmung im Allgemeinen. Ist das (Mess-)Verfahren hinreichend bekannt, besteht die Möglichkeit, dass die Entwicklung auf dieses Maß „optimiert“ wird, das heißt, dass die Metrik nicht mehr den tatsächlichen Aufwand berechnet, sondern denjenigen Aufwand, welcher von einer Person in eine bestimmte Richtung hin optimiert wurde. Für die Entwurfsentropie ist es möglich, durch die Wahl geeigneter Signale und Variablen die Zustandsanzahl erheblich zu erhöhen, sodass die Entwurfsentropie wesentlich höher ausfällt. Bei den Fallstudien in Kapitel 6 handelt es sich vorwiegend um Projekte, welche unabhängig von der Metrik umgesetzt wurden. Somit sind die Ergebnisse nicht durch den Einfluss der Messmethode verfälscht worden.

Bei Softwareprojekten werden regelmäßig Variablen eingesetzt, welche wesentlich mehr mögliche Zustände als Entwurfszustände haben. Daher eignet sich der Ansatz der Entwurfszustände insbesondere für die Analyse von Softwareprogrammen. Der folgende Abschnitt zeigt die Anwendung der Entwurfsentropie auf Software. Da der Fokus der vorliegenden Dissertation auf Hardware liegt, stellt der folgende Abschnitt ebenfalls einen Exkurs dar.

4.11 Exkurs: Software

Der Ansatz, nicht mehr die Anzahl aller möglichen Zustände, sondern die Entwurfszustände zu verwenden, erlaubt die Anwendung der Entwurfsentropie auf Software. Auch dabei bilden Komponenten sowie der Informationsaustausch zwischen diesen die Basis der Berechnung [[146]]. Als Beispiel wird zunächst die in Code 4.9 gezeigte Addition in C betrachtet. In Zeile 1 des Codes werden die Variablen a, b und result als Integerwerte deklariert. In der zweiten Zeile des Codes werden die Werte der Variablen a und b addiert und das Ergebnis als result gespeichert.

```
1  int a, b, result;
2  result = a + b;
```

Code 4.9: Addition in C

Wie bei der Analyse von Hardwarebeschreibungen, besitzt die reine Deklaration in Zeile 1 keine Entwurfsentropie [[146]]. Es wird nur festgelegt, dass die Variablen a, b, und result vom Typ Integer sind. Abhängig vom System müssen Integerwerte mindestens 2 Byte umfassen, wobei auf heutigen Desktopsystemen eine Integervariable in der Regel 4 Byte groß ist [245, S. 69 f.], sodass $2^{(4 \cdot 8)} = 2^{32} = 4.294.967.296$ mögliche Werte gespeichert werden können, wenn ein Byte aus acht Bit besteht. Die Addition in Zeile 2 wird als Komponente im Sinne der Entwurfsentropie aufgefasst, welche in Abbildung 4.26 grafisch dargestellt ist. Wird von der maximalen Anzahl möglicher Zustände ausgegangen, dann hat diese Komponente eine Verhaltensentropie von $H_B(\text{Add}_{\text{int}}) = 3 \cdot \log(2^{32}) = 96 \cdot \log(2) \approx 28{,}90$.

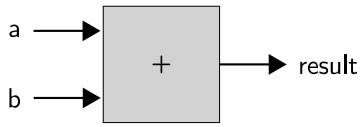


Abbildung 4.26: Addition in C als Komponente

Bei der Verwendung von long-Variablen mit 64-Bit statt int-Variablen würde die Addition mit $H_B(\text{Add}_{\text{long}}) = 3 \cdot 64 \cdot \log(2) \approx 57{,}80$ eine doppelt so hohe Verhaltensentropie aufweisen. Dabei wurde bei der Programmierung der Additionszeile nicht beachtet, welche konkreten Werte die Variablen annehmen können, sondern nur, dass eine Addition der Werte erfolgt. Beispielsweise wird nicht berücksichtigt, ob die Summe der Variablen a und b größer als der maximale Wertebereich der Variablen result ist.

Wird die Anzahl an Entwurfszuständen für die Berechnung der Verhaltensentropie zugrunde gelegt, wurden für jede Verbindung nur die zwei möglichen Mindestzustände betrachtet: ob diese verbunden sind oder nicht. Die Verhaltensentropie mit den Entwurfszuständen beträgt somit $H_B(\mathrm{Add}_{\mathrm{EZ}}) = 3 \cdot \log(2) = 0{,}90$. Für die Verwendung der Entwurfszustände ist es dabei unbeachtlich, welche möglichen Zustände die Variablen annehmen können. Es wird ausschließlich berücksichtigt, welche der möglichen Zustände tatsächlich betrachtet wurden. Wie im vorherigen Abschnitt beschrieben, ist die Mindestanzahl an Entwurfszuständen für eine Verbindung einer Nichtkonstanten zwei.

Um die Berechnung der Entwurfsentropie von Softwareprogrammen zu verallgemeinern, eignen sich als Basis (Kontroll-)Flussdiagramme. Ein Kontrollflussgraph ist ein gerichteter Graph, dessen Knoten Berechnungen darstellen [225, S. 68]. Eine Berechnung ist eine Abfolge von Berechnungsschritten, das heißt, eine Abfolge einzelner Anweisungen [199, S. 8 f.]. Eine Berechnung wird als sogenannter Basisblock verstanden [52, S. 108] (auch Grundblöcke genannt [2, S. 642]). Ein Basisblock ist eine möglichst lange Sequenz von Anweisungen, wobei der Block nur durch den ersten Befehl betreten und durch den letzten Befehl verlassen werden kann [2, S. 642]. Dadurch enthält ein Basisblock keine Sprünge oder Verzweigungen mit Ausnahme des letzten Befehls, sodass die Befehle eines Blocks immer zusammenhängend und nacheinander durchlaufen werden [2, S. 642]. Eine alternative Darstellungsform von Kontrollflussgraphen fasst die Knoten als einzelne Programmschritte auf, die während der Berechnung durchlaufen werden, sodass die Kanten den einzelnen Berechnungsschritten entsprechen [199, S. 8]. Auf die Möglichkeit der Unterbrechung, zum Beispiel durch Interrupts, soll an dieser Stelle nicht näher eingegangen werden, sodass zugrunde gelegt wird, dass der Code nach einer Unterbrechung an der entsprechenden Stelle wieder fortgeführt wird und die Unterbrechung somit keinen Sprung oder eine Verzweigung des eigentlichen Codes darstellt [ähnlich 2, S. 642].

Knoten, die mehrere Nachfolger haben, werden Verzweigungsknoten genannt und beschreiben einen alternativen Kontrollfluss, welcher von einem Booleschen Ausdruck abhängig ist [225, S. 68]. Die Kanten des Graphen drücken als Nachfolgerelationen den (sequenziellen) Kontrollfluss aus [225, S. 68]. Zudem enthält jeder Kontrollflussgraph einen Eingangsknoten und einen oder mehrere Ausgangsknoten [52, S. 108] [199, S. 9]. Normalerweise werden (Kontroll-)Flussgraphen für Korrektheitsbeweise von Programmanalysen [199, S. 8], zur Codeerzeugung und Optimierung bei Compilern [2, S. 642 ff.] oder für Testfallgenerierungen für Softwareprogramme [196, S. 146] eingesetzt.

Die Darstellung eines Softwareprogramms als (Kontroll-)Flussdiagramm, welches um weitere Informationen erweitert wird, ist die Basis der Entwurfsentropieberechnung. Zur Abgrenzung wird dieser Graph als Komponentendiagramm bezeichnet. Die Knoten des Graphen stellen Komponenten im Sinne der Entwurfsentropie dar. Abbildung 4.27 zeigt allgemein einen Knoten des Komponentendiagramms. Grundsätzlich enthält eine Komponente jeweils eine Anweisung des Softwareprogramms, sodass die Knoten Elementarkomponenten darstellen. Jedoch können auch mehrere Knoten zu einer neuen Komponente zusammengefasst werden, insbesondere in Form von Funktionen, Prozeduren oder Klassen.

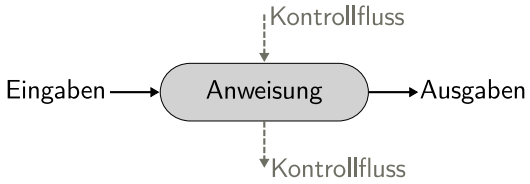


Abbildung 4.27: Knoten/Komponente des Komponentendiagramms

Wie bei den Flussdiagrammen werden die Komponenten mithilfe von Kanten verbunden, welche die Nachfolgerelationen des (sequenziellen) Kontrollflusses darstellen (gestrichelte Linien). Ein Knoten kann bei Verzweigungen und/oder Schleifen auch mehrere Vorgänger beziehungsweise Nachfolger haben. Daneben haben die Knoten/Komponenten noch Ein- und Ausgaben, welche den Ein- und Ausgabedaten der Anweisungen entsprechen. Sie bilden dabei den Datenfluss der Komponente ab, sodass in gewisser Weise die Knoten auch die datenverarbeitenden und -speichernden Knoten eines Datenflussgraphen darstellen. Da die Knoten des Komponentendiagramms sowohl einzelne Anweisungen, Anweisungen, welche aus mehreren Operatoren bestehen als auch Komponenten mit mehreren Unterkomponenten darstellen können, kann einer dieser Knoten eine Vielzahl an Knoten des Datenflussgraphen enthalten. Die Ein- und Ausgangsdaten werden dabei als horizontale Linien im Komponentendiagramm dargestellt. Die Verknüpfung des (Kontroll-)Flussdiagramms, des Datenflusses und des Komponentenansatzes der Entwurfsentropie erzeugt eine universelle Darstellungsmöglichkeit von Softwareprogrammen und erlaubt die Berechnung der Entwurfsentropie.

Die Darstellung eines Softwareprogramms als Komponentendiagramm wird anhand des folgenden Beispiels anschaulich gezeigt [Beispiel angelehnt an 196, S. 147]. Code 4.10 enthält eine C-Funktion, welcher der strukturierte Datentyp `person` übergeben wird. Ist die Person ein Studierender, wird abhängig vom Alter ein

Rabatt gewährt, anderenfalls wird kein Rabatt gewährt. Ist der Studierende unter 26 Jahren, dann beträgt der Rabatt 50 %, anderenfalls 33 %. Der Rückgabewert der Funktion ist der Rabatt als double.

```
1  double rabatt(struct person p) {
2      double rabatt = 0.0;
3      if (p.studi) {
4          if (p.alter < 26) {
5              rabatt = 0.5;
6          } else {
7              rabatt = 0.33;
8          }
9      }
10     return rabatt;
11 }
```

Code 4.10: Rabattberechnung [angelehnt an 196, S. 147]

Wie bei der Analyse von Hardwareentwürfen stehen zwei verschiedene Entropien zur Verfügung: Die Verhaltensentropie, welche die Komponente als abgeschlossene Einheit betrachtet und die Strukturentropie, welche den Aufbau der Komponente betrachtet, insbesondere deren Anweisungen [[146]]. Für das Beispiel der Rabattberechnung ist die abgeschlossene Komponente die Funktion, welche als Eingabe die Variable p und als Ausgabe den Rabatt hat. Grundsätzlich würde die Eingabe eine (fast) unendliche Kombination an möglichen Zuständen besitzen, insbesondere, wenn der Struct person noch weitere Daten, wie Name, Adresse und so weiter enthält. Jedoch werden von der rabatt-Funktion nur drei Zustände berücksichtigt: Ist die Person ein Studierender unter 26 Jahren, ist die Person ein Studierender über 25 Jahren oder ist die Person kein Studierender. Analog hätte der Rückgabewert grundsätzlich 2^{64} mögliche Zustände, von denen in der Funktion nur drei Entwurfszustände betrachtet werden: 0.0, 0.33 und 0.5. Werden die tatsächlich betrachteten Zustände, das heißt, die Entwurfszustände, zugrunde gelegt, dann haben sowohl die Eingangsvariable als auch die Ausgangsvariable jeweils drei Entwurfszustände. Hieraus ergibt sich eine Verhaltensentropie der Rabatt-Funktion von $H_B(rabatt) = \log(2^3 \cdot 2^3) = 6 \cdot \log(2) \approx 1{,}81$.

Die Berechnung der Strukturentropie setzt auf dem Komponentendiagramm auf. Für den Beispielcode 4.10 der Rabatt-Funktion ist dieses in Abbildung 4.28 gezeigt. Wie zuvor beschrieben, werden die einzelnen Anweisungen als Knoten dargestellt. Die gestrichelten Linien stellen den Kontrollfluss des Programms dar [angelehnt an 196, S. 147]. Zusätzlich enthält jeder Knoten noch die Ein- und Ausgaben der Anweisungen.

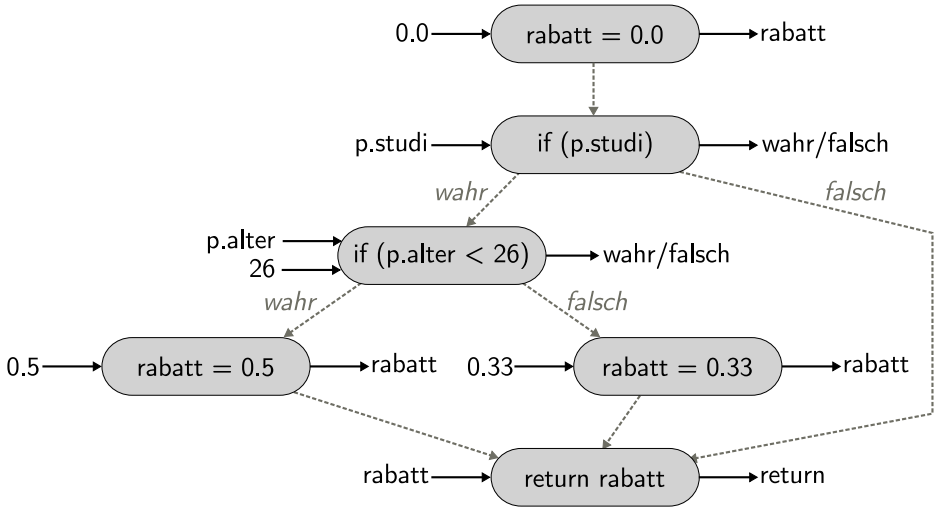


Abbildung 4.28: Komponentendiagramm der Rabattberechnung

Die Zuweisung rabatt = 0.0 hat als Eingabe die Konstante 0.0, welche als rabatt gespeichert wird. Die nächste Anweisung des Codes ist die Bedingung, ob die Person ein Studierender ist (if (p.studi)). Als Eingabevariable dient p.studi, welche entweder null (entspricht falsch) oder sonstige Integerwerte (entspricht wahr) annehmen kann. Die Ausgabe ist das Ergebnis des Vergleichs, welches entweder wahr (Person ist ein Studierender) oder falsch (Person ist kein Studierender) sein kann. Analog verhält es sich mit der zweiten Bedingung (if (p.alter < 26)), welche prüft, ob die Person unter 26 Jahren alt ist. Neben der Variablen p.alter hat diese zusätzlich den Vergleichswert (26) als Eingang, welcher eine Konstante ist. Die Zuweisungen der beiden Zweige haben jeweils eine Konstante (0.33 beziehungsweise 0.5) als Eingang und liefern als Ausgang den Rabatt. Die letzte Anweisung des Codes hat die Variable rabatt als Eingang und liefert als Ausgabe return.

Zusammengefasst werden bei der Rabatt-Funktion für die Variablen rabatt und return jeweils drei Entwurfszustände betrachtet: 0.0, 0.33 und 0.5. Für die Variablen p.studi und p.alter werden jeweils zwei Entwurfszustände betrachtet. Die Strukturentropie der Rabatt-Funktion berechnet sich gemäß Formel (4.11) aus der Summe der Verhaltensentropien der Komponenten. Diese sind in Tabelle 4.4 für jede Anweisung berechnet. Analog zur Berechnung bei Hardwareimplementierungen haben Konstanten nur einen möglichen Zustand, sodass gilt: $\log(1) = 0$. Als Summe ergibt sich eine Strukturentropie für die Rabatt-Funktion von $H_S = 4 \cdot \log(2) + 5 \cdot \log(3) \approx 3{,}59$.

Zeile	Anweisung	Eingabe	Ausgabe	H_B
2	rabatt = 0.0	0.0	rabatt	$\log(1) + \log(3)$
3	if (p.studi)	p.studi	wahr/falsch	$\log(2) + \log(2)$
4	if (p.alter < 26)	p.alter, 26	wahr/falsch	$\log(2) + \log(1) + \log(2)$
5	rabatt = 0.5	0.5	rabatt	$\log(1) + \log(3)$
7	rabatt = 0.33	0.33	rabatt	$\log(1) + \log(3)$
10	return rabatt	rabatt	return	$\log(3) + \log(3)$
				$H_S = 4 \cdot \log(2) + 5 \cdot \log(3)$

Tabelle 4.4: Strukturentropie von Code 4.10

Die Zeilen 4 – 8 des Rabattcodes 4.10 können auch mithilfe eines ternären Operators umgesetzt werden, wie Code 4.11 zeigt (angelehnt an [[146]]). Abhängig vom Alter der Person wird ein Rabatt in Höhe von 50 % oder 33 % gewährt. Neben der Variablen p.alter, welche zwei Entwurfszustände hat, sind die weiteren Eingabedaten der Anweisung die Konstanten 26, 0.33 und 0.5. Die Ausgabe der Anweisung ist der Rabatt.

```
1  rabatt = ( p.alter < 26) ? 0.5 : 0.33;
```

Code 4.11: Ternärer Operator

Wie beim vorherigen Code kann auch der ternäre Operator als Knoten des Komponentendiagramms dargestellt werden, welcher in Abbildung 4.29 dargestellt ist. Der ternäre Operator ist als Komponente im Sinne der Entwurfsentropie anzusehen, da dieser nur verwendet wird und keine Strukturentropie besitzt. Das heißt, er kann nicht in weitere Teile zerlegt werden, sondern bildet eine Elementarkomponente. Im Gegensatz zum Kontrollflussdiagramm erzeugt der ternäre Operator beim Komponentendiagramm keine Verzweigung, da es sich um eine Elementarkomponente handelt.

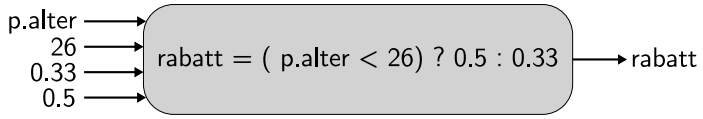


Abbildung 4.29: Komponentendiagramm des ternären Operators

Wird zugrunde gelegt, dass Code 4.11 mit dem ternären Operator anstatt der Zeilen 4 – 8 mit der if-else-Bedingung im ursprünglichen Code 4.10 der Rabattberechnung eingesetzt wird, dann werden für die Variable rabatt auch in diesem Fall drei Zustände als Entwurfszustände betrachtet: 0.0, 0.33 und 0.5. Die Konstanten an den Eingängen haben eine Entropie von null ($\log(1) = 0$). Somit hat die Codezeile des ternären Operators eine Verhaltensentropie von $H_B = \log(2 \cdot 3) \approx 0{,}78$.

Bei einer Fallunterscheidung (Switch-Case-Anweisung) wird deutlich, dass die Entwurfszustände die Anzahl an Zuständen widerspiegeln, welche für eine Variable bei der Programmierung betrachtet werden [[146]]. Im folgenden Codebeispiel 4.12 werden für die Variablen j drei Zustände betrachtet: 1, 2 und andere Werte.

```
1  switch(j) {  case 1:  return  10;
2               case 2:  return 100;
3               default: return   0;}
```

Code 4.12: Switch-Case-Anweisung (angelehnt an [[146]])

Das Komponentendiagramm der Switch-Case-Anweisung ist in Abbildung 4.30 dargestellt. Die konstanten Eingabedaten der Knoten sind nicht eingezeichnet, da diese keinen Einfluss auf die Entwurfsentropie haben, wie bereits gezeigt wurde. Die Variable j hat drei Entwurfszustände. Der Switch-Case-Block hat somit drei Ausgänge, welche jeweils „wahr“ oder „falsch“ sein können. Dabei bedingen sich im vorliegenden Fall die Ausgänge gegenseitig.

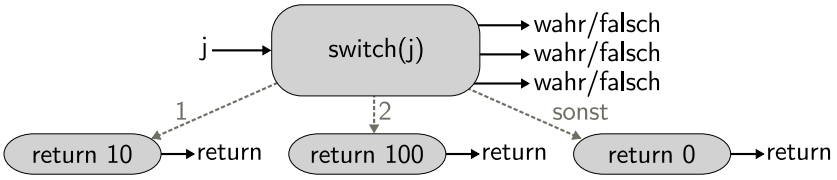


Abbildung 4.30: Komponentendiagramm der Switch-Case-Anweisung

Nach dem Modell der Entwurfsentropie aus Abschnitt 4.1 werden die Ausgänge als linear unabhängig angesehen. Das heißt, es wird nicht betrachtet, dass diese sich gegenseitig bedingen können. Dabei müssen die beiden anderen Ausgänge des vorherigen Beispiels „falsch“ sein sobald einer der Ausgänge „wahr“ ist. Daher muss in weiteren Forschungen zur Entwurfsentropie untersucht werden, ob eine abweichende Beurteilung für sich gegenseitig bedingende Ausgänge (und gegebenenfalls Eingänge) erforderlich ist. Da es sich bei der Anwendung auf Software um einen

Exkurs handelt und der Fokus auf der Analyse von Hardwarebeschreibungen liegt, ist dieser Bereich noch nicht abschließend untersucht worden. Die Entwicklung einer vollständigen Softwaremetrik würde den Rahmen der vorliegenden Dissertation übersteigen. Jedoch zeigen die Darstellungen in diesem Abschnitt, dass die Entwurfsentropie durchaus vielversprechende Ergebnisse bei der Komplexitäts- und Quantitätsanalyse von Softwareprogrammen liefert.

Der Ansatz, Entwurfszustände auf Softwareprogramme anzuwenden, weist gewisse Ähnlichkeiten mit der zyklomatischen Komplexität aus Unterabschnitt 3.2.8 auf. Die Schwäche der zyklomatischen Komplexität liegt darin, dass die Maßzahl allein auf den Verzweigungen eines Programms beruht. Sie ist daher als Softwaremetrik kaum geeignet, sondern liefert die Anzahl erforderlicher Testfälle, um alle Pfade des Graphen zu durchlaufen. Der in diesem Abschnitt vorgestellte Entwurfsentropieansatz, Softwareprogramme auf Basis eines Kontrollflussgraphen zu analysieren, folgt zunächst dem Ansatz der zyklomatischen Komplexität. Die Schwäche der zyklomatischen Komplexität wird dadurch überwunden, dass die Knoten nicht mehr als Basisblöcke betrachtet werden, sondern die einzelnen Anweisungen als Elementarkomponenten. Damit hängt die Metrik nicht mehr ausschließlich von den Verzweigungen ab. Zudem wird die Darstellung des Kontrollflusses um die Ein- und Ausgabedaten in Anlehnung an den Datenfluss erweitert, sodass auch die Daten und nicht nur der Kontrollfluss betrachtet werden. Das entstandene Komponentendiagramm kann sehr flexibel eingesetzt werden, um die unterschiedlichsten Anweisungen, Befehle und Programme zu analysieren.

Dass die Komponentendiagramme auf den Kontroll- und Datenflussgraphen basieren, bietet weiterhin den Vorteil, dass für den Aufbau solcher Graphen bereits umfangreiche Ansätze existieren. Insbesondere kann für die Berechnung der Entwurfsentropie von Softwareprogrammen ein ähnlicher Ansatz wie bei der Entwicklung des Analysewerkzeugs für Hardwareentwürfe gewählt werden: Mithilfe eines Compilers kann der Softwarequellcode als Flussgraph dargestellt werden. Ein Parsergenerator kann die Entwicklung eines Analysewerkzeugs unterstützen. Für die Berechnung der Anzahl an Entwurfszuständen kann auf die Set-Based-Analysis (oder auch Constraint-Based-Analysis) aufgebaut werden (insbesondere auf [78]). Bei der Set-Based-Analysis werden Programme durch eine Abstraktion beschrieben: Programmvariablen werden als Sätze von Werten betrachtet. Im Wesentlichen entspricht dies einem Ignorieren der Abhängigkeiten zwischen Variablen, ohne dabei eine Approximation der dahinterstehenden Werte vorzunehmen. Die Set-Based-Analysis bietet eine intuitive, abstrakte und nicht-algorithmische Sicht dessen, was auch die Datenflussanalyse erreicht.

4.12 Zwischenfazit

Die Entwurfsentropie basiert auf dem Modell des Nachrichtenaustauschs zwischen den Komponenten eines Systems. Beim Entwurf werden die Komponenten derart miteinander verbunden, dass das System die beabsichtigten Aufgaben erfüllen kann. Die Entwurfsentropie misst den mittleren Informationsgehalt der Nachrichten, wodurch die Komplexität eines Systems wiedergegeben wird. Die Informationen beziehungsweise Nachrichten werden als Zustände beschrieben. Die Berechnung und die Ableitung der Formeln beruht auf der Informationstheorie und deren Entropieberechnung. Durch die Berücksichtigung der einzelnen Schnittstellen der Komponenten fließt zudem die Entwurfsgröße in die Maßzahl ein. Dabei wird die in einem System liegende Komplexität und Entwurfsgröße gemessen und nicht die subjektiv empfundene Komplexität. Das heißt, die Entwurfsentropie trifft eine Aussage über das jeweilige System und nicht darüber, wie die minimale, optimale oder effizienteste Umsetzung wäre.

Kern der Berechnung der Entwurfsentropie ist zum einen die Verhaltensentropie und zum anderen die Strukturentropie. Die Verhaltensentropie drückt die Komplexität der Verwendung einer Komponente aus, das heißt, die Komponente wird als abgeschlossene Einheit betrachtet. Dabei ist es für die Verwendung nicht erforderlich, den inneren Aufbau einer Komponente zu kennen, das heißt, welche innere Struktur die Komponente hat. Für den Entwurf ist es nur erforderlich das Verhalten zu kennen, das heißt, welche Nachrichten eine Komponente als Eingaben benötigt und basierend auf deren Verarbeitung, welche Nachrichten (Ereignisse) die Komponente als Ausgaben liefert.

Neben der reinen Verwendung von Komponenten werden auch Komponenten während eines Entwurfs entwickelt. Dies erlaubt zum einen die Partitionierung des Systems und zum anderen die Wiederverwendung von Komponenten. Dabei drückt die Strukturentropie die Komplexität und Entwurfsgröße einer Komponente aus, die innerhalb des Systementwurfs aufgebaut wurde. Da der Aufwand für den Entwurf einer solchen Komponente nur einmalig angefallen ist, wird die Strukturentropie dieser neuen Komponente nur ein Mal berücksichtigt, unabhängig davon, wie oft die Komponente eingesetzt wird. Dagegen wird die Verhaltensentropie der Komponente bei jeder Verwendung berücksichtigt.

Nach dem Komponentenansatz ist das System, für das die Entwurfsentropie berechnet werden soll, selbst eine Komponente. Für diese wird die Strukturentropie dadurch berechnet, dass die enthaltenen Komponenten analysiert werden. Handelt es sich um keine Elementarkomponenten, also Komponenten, die nicht im Entwurf entwickelt wurden, dann besteht eine Komponente wiederum aus Unterkomponenten. Diese „rekursive“ Analyse wird so lange fortgeführt, bis nur noch Elementarkomponenten oder wiederverwendete Komponenten zum Aufbau der zu analysierenden Komponente verwendet wurden. Der Ansatz der Rekursion wird der Tatsache gerecht, dass Entwürfe regelmäßig aus einer Vielzahl verschachtelter Komponenten bestehen. Weiterhin können Teile wiederverwendet werden, was von der Strukturentropie berücksichtigt wird, da diese nur einmalig in die Berechnung eingehen.

Unabhängig von der Entwurfsdomäne und der Abstraktionsebene findet die Verhaltensentropie immer bei der Verwendung einer Komponente Anwendung und die Strukturentropie immer bei der Analyse des Aufbaus einer Komponente. Durch diesen abstrakten Ansatz ist es möglich auf unterschiedlichsten Abstraktionsebenen und in verschiedenen Entwurfsdomänen Komponenten zu definieren und basierend auf diesen die Entwurfsentropie zu berechnen. Die Abstraktheit lässt auch die Anwendung auf Software zu, wie im vorherigen Abschnitt beschrieben. Im folgenden Kapitel wird die Entwurfsentropie auf kompakte Hardwareentwürfe angewendet, um die Berechnungen nachvollziehbar zu machen. Für umfangreichere Entwürfe wurde ein Analysewerkzeug entwickelt, welches Hardwarebeschreibungen in VHDL analysiert. Dieses wird im übernächsten Kapitel verwendet, um innerhalb mehrerer Fallstudien die Entwurfsentropie zu berechnen.

5 Anwendung

Ziel dieses Kapitels ist, die Berechnung der Entwurfsentropie anschaulich darzustellen. Hierzu werden im Verlauf mehrere kompakte Hardwareentwürfe vorgestellt, anhand welcher unterschiedliche Aspekte der Entwurfsentropieberechnung gezeigt werden. Beginnend bei Logikgattern, über Halbaddierer und Volladdierer, wird in der ersten Hälfte des Kapitels ein Ripple-Carry-Addierer aufgebaut. Die zweite Hälfte beschäftigt sich mit sequenziellen Schaltungen, insbesondere mit D-Flipflops und Zustandsautomaten.

Für die bessere Darstellbarkeit der folgenden Schaltungen werden Blockdiagramme eingesetzt. Die Entwurfsentropie wird mithilfe von Formel (4.10) für die Verhaltensentropie und Formel (4.11) für die Strukturentropie berechnet, welche im vorherigen Kapitel eingeführt wurden und an den entsprechenden Stellen wiederholt werden. Zudem wird zunächst eine zweiwertige (binäre) Logik mit „Wahr“/ „Falsch“ beziehungsweise mit (logischen) Einsen und Nullen verwendet.

5.1 Logikgatter

Grundlage der folgenden Schaltungen bilden Verknüpfungsglieder, welche auch als (Logik-)Gatter bezeichnet werden [214, S. 75]. Sie setzen Boolesche Operationen durch die Verknüpfung einer oder mehrerer Schaltvariablen (Eingangsvariablen) zu einer Ausgangsvariablen um [214, S. 75]. In Abhängigkeit der Eingangssignale ist das Ausgangssignal deterministisch bestimmt. Grenzfälle sowie Laufzeiten werden zunächst nicht berücksichtigt, sodass der reine Verschaltungsaufwand betrachtet wird. Eine Übersicht der verwendeten Logikgatter mit ihren Symbolen, Booleschen Funktionen und Wahrheitstabellen ist in Tabelle 5.1 gegeben. Dabei werden die Symbole mithilfe der rechteckigen Form dargestellt, wie in den ANSI/ IEEE Standards 91-1984 [91] und 91a-1991 [93] festgelegt.

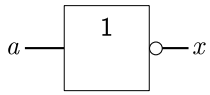
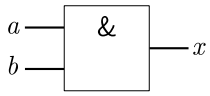
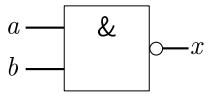
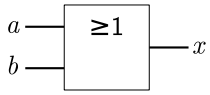
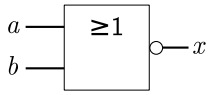
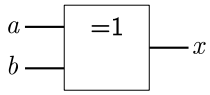
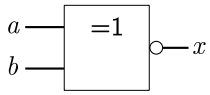
Name	Symbol [nach 93, S. 60 ff.]	Funktion	Wahrheitstabelle
Nicht (NOT)		$x = \overline{a}$	a 0 1 x 1 0
Und (AND)		$x = a \cdot b$	a 0 0 1 1 b 0 1 0 1 x 0 0 0 1
Nicht-Und (NAND)		$x = \overline{a \cdot b}$	a 0 0 1 1 b 0 1 0 1 x 1 1 1 0
Oder (OR)		$x = a + b$	a 0 0 1 1 b 0 1 0 1 x 0 1 1 1
Nicht-Oder (NOR)		$x = \overline{a + b}$	a 0 0 1 1 b 0 1 0 1 x 1 0 0 0
Antivalenz (XOR)		$x = a \oplus b$	a 0 0 1 1 b 0 1 0 1 x 0 1 1 0
Äquivalenz (XNOR)		$x = \overline{a \oplus b}$	a 0 0 1 1 b 0 1 0 1 x 1 0 0 1

Tabelle 5.1: Logikgatter

Die Gatter werden in den folgenden Schaltungen als Elementarkomponenten im Sinne der Entwurfsentropie verwendet, sodass diese nur eine Verhaltensentropie haben. Die Verhaltensentropie einer Komponente berechnet sich nach Formel (4.10) aus Definition 18 auf Basis der Ein- und Ausgangszustände:

Die Eingänge einer Komponente k werden mit $n_i(k)$ bezeichnet, wobei $i = 1, \ldots, n(k)$ gilt, sodass $n(k)$ die Anzahl Eingänge ist. Die möglichen Zustän-

de der Eingänge werden mit $z(n_i(k))$ bezeichnet. Die Ausgänge werden mit $m_j(k)$ bezeichnet. Analog ist $m(k)$ die Anzahl Ausgänge, sodass $j = 1, \ldots, m(k)$ gilt und $z(m_j(k))$ die Anzahl möglicher Zustände der Ausgänge ist. Hiermit berechnet sich die Verhaltensentropie $H_B(k)$ der Komponente k mit:

$$H_B(k) = \log \left(\prod_{i=1}^{n(k)} z(n_i(k)) \cdot \prod_{j=1}^{m(k)} z(m_j(k)) \right)$$

Zunächst wird die Verhaltensentropie des Nicht-Gatters als Komponente berechnet, sodass $k = \text{NOT}$ ist. Dieses besitzt einen Eingang und einen Ausgang, somit sind $n(\text{NOT}) = m(\text{NOT}) = 1$. Beide Schnittstellen können jeweils zwei binäre Zustände annehmen: $z(n_1(\text{NOT})) = z(m_1(\text{NOT})) = 2$. Eingesetzt in die vorherige Formel ergibt sich als Verhaltensentropie des Nicht-Gatters $H_B(\text{NOT}) \approx 0{,}60$, welche sich wie folgt berechnet [[141]]:

$$\begin{aligned} H_B(\text{NOT}) &= \log \left(\prod_{i=1}^{1} \underbrace{z(n_i(k))}_{=2} \cdot \prod_{j=1}^{1} \underbrace{z(m_j(k))}_{=2} \right) \\ &= \log\,(2 \cdot 2) = \log\,(4) \approx 0{,}60 \end{aligned}$$

Die übrigen Gatter in Tabelle 5.1 besitzen zwei Eingänge und einen Ausgang, sodass $n(k) = 2$ und $m(k) = 1$ für $k = \{\text{AND}, \text{NAND}, \text{OR}, \text{NOR}, \text{XOR}, \text{XNOR}\}$ gilt. Jeder dieser Ein- und Ausgänge kann zwei Zustände annehmen, das heißt, $z(n_1(k)) = z(n_2(k)) = z(m_1(k)) = 2$. Hieraus ergibt sich eine Verhaltensentropie von $H_B(k) \approx 0{,}90$ der zweiwertigen Gatter aus [[143]]:

$$\begin{aligned} H_B(k) &= \log \left(\prod_{i=1}^{2} z(n_i(k)) \cdot \prod_{j=1}^{1} z(m_j(k)) \right) \\ &= \log\,(2 \cdot 2 \cdot 2) = \log\,(8) \approx 0{,}90 \end{aligned}$$

Wie eingangs beschrieben werden die Gatter dazu verwendet, schrittweise einen Ripple-Carry-Addierer aufzubauen. Als Basis sind im folgenden Abschnitt mehrere Möglichkeiten dargestellt, einen Halbaddierer aufzubauen. Anhand dieser Realisierungen wird gezeigt, wie die Entwurfsentropie mit der Anzahl an Bauelementen zusammenhängt und wie die berechneten Entropiewerte miteinander verglichen werden können.

5.2 Halbaddierer

Ein Halbaddierer summiert zwei binär codierte Eingänge nach deren dualen Wertigkeit und gibt die Summe sowie den Ausgangsübertrag (carry) aus [214, S. 82]. Er besitzt hierfür zwei Eingänge und zwei Ausgänge, wie Abbildung 5.1 mit dem Symbol eines Halbaddierers nach dem ANSI/IEEE Standard 91a-1991 [93, S. 85] zeigt.

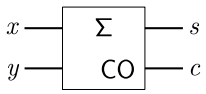


Abbildung 5.1: Symbol eines Halbaddierers [nach 93, S. 85]

x	0	0	1	1
y	0	1	0	1
s	0	1	1	0
c	0	0	0	1

Tabelle 5.2: Wahrheitstabelle eines Halbaddierers [nach 223, S. 174]

Das Verhalten des Halbaddierers wird durch die Wahrheitstabelle abgebildet (Tabelle 5.2). Sind beide Eingänge x und y null, sind sowohl die Summe s als auch der Ausgangsübertrag c null. Ist einer der beiden Eingänge eins, dann ist die Summe eins und der Ausgangsübertrag null. Sind beide Eingänge eins, dann ist die Summe null und der Ausgangsübertrag eins.

Im Folgenden werden drei Realisierungen A, B und C für einen Halbaddierer betrachtet. Dabei stellt der Halbaddierer das System dar. Nach Definition 9 ist das System selbst eine Komponente. Wird eine neue Komponente aus Elementarkomponenten aufgebaut, dann berechnet sich die Entwurfsentropie dieser neuen Komponente mithilfe der Strukturentropie. Die Strukturentropie einer Komponente k ergibt sich gemäß Formel (4.11) von Definition 19 aus der Summe der Verhaltensentropien $H_B(i)$ aller direkten Unterkomponenten $i \in k_u$:

$$H_S(k) = \sum_{i \in k_u} H_B(i)$$

Da die folgenden Halbaddierer alle aus Gattern aufgebaut sind, muss für die Logikgatter folglich keine zusätzliche Strukturentropie berücksichtigt werden. Denn die Gatter stellen Elementarkomponenten im Sinne der Entwurfsentropie dar.

Realisierung A besteht aus zwei Nicht-Gattern, drei Und-Gattern und einem Oder-Gatter [[143]]. Der Aufbau ist in Abbildung 5.2 dargestellt. Der Ausgangsübertrag c wird direkt durch ein Und-Gatter umgesetzt. Die Summe ergibt sich aus der Oder-Verknüpfung der Ausgänge der weiteren beiden Und-Gatter, welche jeweils einen Eingang und den invertierten anderen Eingang besitzen.

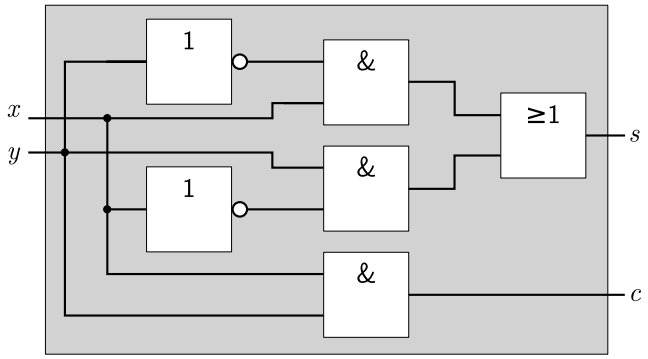


Abbildung 5.2: Halbaddierer – Realisierung A [[143]]

Wie zu Beginn dieses Abschnitts beschrieben, berechnet sich die Strukturentropie des Halbaddierers aus der Summe der Verhaltensentropien seiner Komponenten. Gemäß der folgenden Berechnung ergibt sich eine Strukturentropie des Halbaddierers aus Abbildung 5.2 von $H_S(\mathrm{HA_A}) \approx 4{,}82$ [[143]]:

$$\begin{aligned} H_S(\mathrm{HA_A}) &= \sum_{i \in k} H_B(i) \\ &= 2 \cdot H_B(\mathrm{NOT}) + 3 \cdot H_B(\mathrm{AND}) + H_B(\mathrm{OR}) \\ &= 2 \cdot \log\,(4) + 3 \cdot \log\,(8) + \log\,(8) \\ &= (2 \cdot 2 + 3 \cdot 3 + 3) \cdot \log\,(2) \approx 4{,}82 \end{aligned}$$

Wird der Halbaddierer als abgeschlossene Komponente betrachtet, dann ist nur das Verhalten der Komponente gegenüber der Umwelt ausschlaggebend. Hierfür wird die Verhaltensentropie verwendet. Für den Halbaddierer aus Abbildung 5.2 ergibt sich dabei eine Verhaltensentropie von $H_B(\mathrm{HA_A}) \approx 1{,}20$, wie die folgende Rechnung zeigt [[143]]:

$$H_B(\mathrm{HA_A}) = \log \left(\prod_{i=1}^{2} z(n_i(k)) \cdot \prod_{j=1}^{2} z(m_j(k)) \right)$$

$$= \log\,(z(x) \cdot z(y) \cdot z(s) \cdot z(c))$$
$$= \log\,(2 \cdot 2 \cdot 2 \cdot 2) = \log\,(16) \approx 1{,}20$$

Realisierung B beschreibt den Fall, dass beim Entwurf nur eine bestimmte Art von Gattern zur Verfügung steht. Für diese Realisierung wurden Nicht-Und-Gatter gewählt. Fünf dieser Logikgatter erlauben den Aufbau des Halbaddierers, welcher in Abbildung 5.3 dargestellt ist.

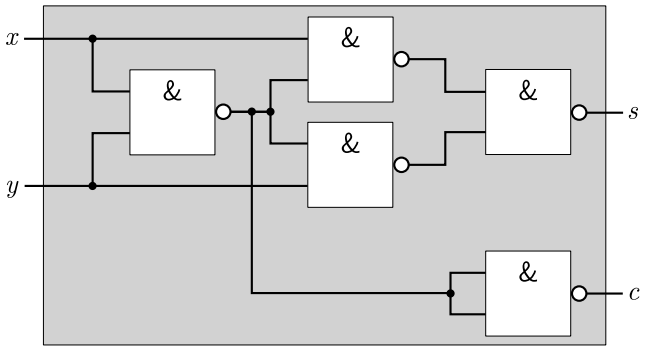


Abbildung 5.3: Halbaddierer – Realisierung B

Die Strukturentropie des Halbaddierers aus Abbildung 5.3 berechnet sich anhand der Verhaltensentropie der fünf Nicht-Und-Gatter. Sie beträgt $H_S(\mathrm{HA_B}) \approx 4{,}52$ nach der folgenden Rechnung:

$$H_S(\mathrm{HA_B}) = 5 \cdot H_B(\mathrm{NAND}) = 5 \cdot \log\,(8) = (5 \cdot 3) \cdot \log\,(2) \approx 4{,}52$$

Da die Verhaltensentropie Komponenten als abgeschlossen ansieht, wird der tatsächliche Aufbau bei der Berechnung nicht betrachtet. Daher entspricht die Verhaltensentropie von Realisierung B derjenigen von Realisierung A: $H_B(\mathrm{HA_B}) = H_B(\mathrm{HA_A}) \approx 1{,}20$ [[143]]. Diese entspricht auch der Verhaltensentropie der folgenden Realisierung C: $H_B(\mathrm{HA_C}) \approx 1{,}20$ [[143]].

Realisierung C bildet den einfachsten Fall ab, einen Halbaddierer aus zwei Gattern aufzubauen. Nach der Wahrheitstabelle des Halbaddierers (Tabelle 5.2) entspricht die Summe einer Antivalenz-Verknüpfung und der Ausgangsübertrag einer Und-Verknüpfung. Somit besteht Realisierung C aus einem Antivalenz-Gatter und einem Und-Gatter [[143]]. Der Aufbau ist in Abbildung 5.4 dargestellt.

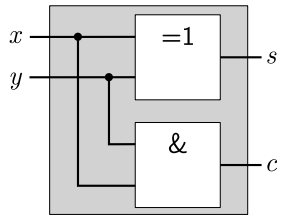


Abbildung 5.4: Halbaddierer – Realisierung C [[143]]

Wie bei den vorherigen Realisierungen setzt sich die Strukturentropie aus der Verhaltensentropie der verwendeten Gatter zusammen. Für Realisierung C ergibt sich somit eine Strukturentropie von $H_S(\mathrm{HA_C}) \approx 1{,}81$ [[143]]:

$$H_S(\mathrm{HA_C}) = H_B(\mathrm{XOR}) + H_B(\mathrm{AND}) = \log\,(8) + \log\,(8) = 6 \cdot \log\,(2) \approx 1{,}81$$

Anhand der im Vorherigen berechneten Werte werden die drei Realisierungen miteinander verglichen. Hierzu sind die berechneten Werte in Tabelle 5.3 eingetragen und grafisch in Abbildung 5.5 dargestellt. Wie bereits angesprochen, ist die Verhaltensentropie für alle drei Halbaddierer gleich, da der Aufbau des Halbaddierers bei der Verhaltensentropie nicht betrachtet wird und die Anzahl Schnittstellen sowie die Anzahl möglicher Zustände immer gleich sind [[143]].

Realisierung	H_B	H_S
A	1,20	4,82
B	1,20	4,52
C	1,20	1,81

Tabelle 5.3: Vergleich der Realisierungen

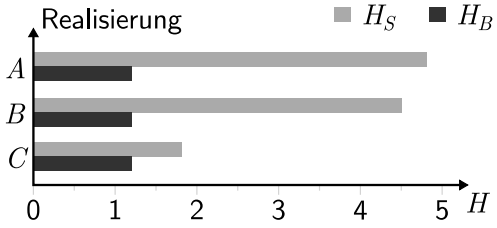


Abbildung 5.5: Vergleich der Realisierungen

Im Gegensatz zur Verhaltensentropie weisen die Strukturentropien der Realisierungen unterschiedliche Werte auf [[143]]. So ist Realisierung C mit nur zwei Gattern die Einfachste, was auch durch den geringsten Wert für die Strukturentropie ausgedrückt wird [[143]]. Aufwendiger sind die Realisierungen A und B, welche höhere Strukturentropiewerte haben [[143]]. Durch die Verwendung von mehr Gattern ist dieses Ergebnis plausibel [[143]]. Dennoch fallen die Werte für

die Realisierungen A und B unterschiedlich aus, da Realisierung A zwei Nicht-Gatter verwendet, welche zusammen eine höhere Verhaltensentropie haben als ein zweiwertiges Gatter [[143]].

Die direkte Berechnung der Verhaltens- und Strukturentropie erlaubt den unmittelbaren Vergleich der Realisierungen. Im folgenden Abschnitt wird gezeigt, dass das Zerlegen eines Systems in Teilsysteme hilft, die Komplexität zu reduzieren. Hierzu wird ein Volladdierer zunächst mit zwei Halbaddierern und anschließend direkt mit Gattern aufgebaut. Dabei wird im Folgenden immer davon ausgegangen, dass alle verwendeten Komponenten, mit Ausnahme der Logikgatter, innerhalb des Entwurfs des jeweiligen Systems aufgebaut wurden, sodass die Berechnung der Strukturentropie über mehrere Hierarchiestufen gezeigt werden kann.

5.3 Volladdierer

Die im vorherigen Abschnitt aufgebauten Halbaddierer erlauben die Addition zweier ein Bit breiter Binärzahlen. Um die Summe mehrstelliger Binärzahlen zu bilden, wird ein Volladdierer benötigt, der zusätzlich noch den Ausgangsübertrag des niederwertigeren Bits addieren kann [223, S. 174]. Wie Abbildung 5.6 mit dem Symbol des Volladdierers zeigt, besitzt dieser hierfür einen dritten Eingang. Liegt an den Eingängen (x, y, c_{in}) eine ungerade Anzahl logischer Einsen an, dann ist die Summe s eins [223, S. 174]. Bei einer geraden Anzahl sowie wenn an allen Eingängen eine logische Eins anliegt, ist der Ausgangsübertrag c_{out} eins [223, S. 174]. Dieses Verhalten wird durch die Wahrheitstabelle (Tabelle 5.4) beschrieben.

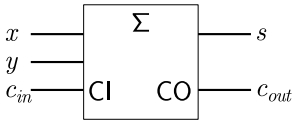


Abbildung 5.6: Symbol eines Volladdierers [nach 93, S. 85]

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
c_{in}	0	1	0	1	0	1	0	1
s	0	1	1	0	1	0	0	1
c_{out}	0	0	0	1	0	1	1	1

Tabelle 5.4: Wahrheitstabelle eines Volladdierers [nach 223, S. 175]

Praktischerweise können für den Aufbau eines Volladdierers zwei Halbaddierer und ein zusätzliches Oder-Gatter verwendet werden [223, S. 174]. Diese erste Variante des Aufbaus ist in Abbildung 5.7 gezeigt. Der linke Halbaddierer liefert als Summe den Eingangsübertrag für den rechten Halbaddierer. Die beiden Ausgangsüberträge der Halbaddierer werden mithilfe eines Oder-Gatters zum Ausgangsübertrag des Volladdierers verbunden.

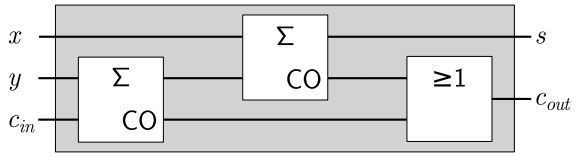


Abbildung 5.7: Volladdierer aus Halbaddierern [[143]]

Um die hierarchische Struktur der Entwurfsentropie zu zeigen wird angenommen, dass der Aufbau des Halbaddierers innerhalb des Volladdiererentwurfs vorgenommen wurde. Somit wird nicht nur die Verhaltensentropie des Halbaddierers, sondern auch dessen Strukturentropie berücksichtigt. Nach Definition 19 berechnet sich die Strukturentropie einer Komponente k aus der Summe der Verhaltensentropien der direkten Unterkomponenten k_u und der Summe der Strukturentropien derjenigen Unterkomponenten $k_n \subseteq k_u$, welche beim Entwurf des Systems entworfen wurden (das heißt, keine Elementarkomponenten) und deren Strukturentropien bei der Berechnung der Entwurfsentropie des Systems noch nicht berücksichtigt wurden:

$$H_S(k) = \sum_{i \in k_u} H_B(i) + \sum_{j \in k_n} H_S(j)$$

Von den drei vorgestellten Möglichkeiten eines Halbaddierers wird Realisierung A des vorherigen Abschnitts gewählt, welche aus zwei Nicht-Gattern, drei Und-Gattern und einem Oder-Gatter besteht. Dieser Halbaddierer besitzt eine Verhaltensentropie von $H_B(\mathrm{HA}) \approx 1{,}20$ und eine Strukturentropie von $H_S(\mathrm{HA}) \approx 4{,}82$. Für den Volladdierer werden zwei Halbaddierer benötigt, sodass die Verhaltensentropie zwei Mal in die Berechnung eingeht. Zudem wird ein Oder-Gatter verwendet, dessen Verhaltensentropie ebenfalls in die Berechnung eingeht:

$$\begin{aligned} H_S(\mathrm{VA}_1) &= \sum_{i \in k_u} H_B(i) + \sum_{j \in k_n} H_S(j) \\ &= 2 \cdot H_B(\mathrm{HA}) + H_B(\mathrm{OR}) + H_S(\mathrm{HA}) \end{aligned}$$

$$= 2 \cdot \log\,(16) + \log\,(8) + 16 \cdot \log\,(2)$$
$$= (2 \cdot 4 + 3 + 16) \cdot \log(2) \approx 8{,}13$$

Die Strukturentropie des Halbaddierers geht nur ein Mal in die vorherige Berechnung ein, obwohl er zwei Mal verwendet wird. Wie im vorherigen Kapitel 4 beschrieben, wird die Verhaltensentropie bei jeder Verwendung einer Komponente berücksichtigt, die Strukturentropie aber nur einmalig, da der Aufwand für den Aufbau der Komponente nur ein Mal angefallen ist. Somit ergibt sich eine Strukturentropie von $H_S(\mathrm{VA}_1) \approx 8{,}13$ für die erste Variante des Volladdierers.

Als zweite Variante wird der Volladdierer direkt mit den Gattern der Halbaddiererrealisierung A aufgebaut, das heißt, mit insgesamt drei Oder-Gattern, vier Nicht-Gattern und sechs Und-Gattern. Dabei ergibt sich der in Abbildung 5.8 gezeigte Aufbau.

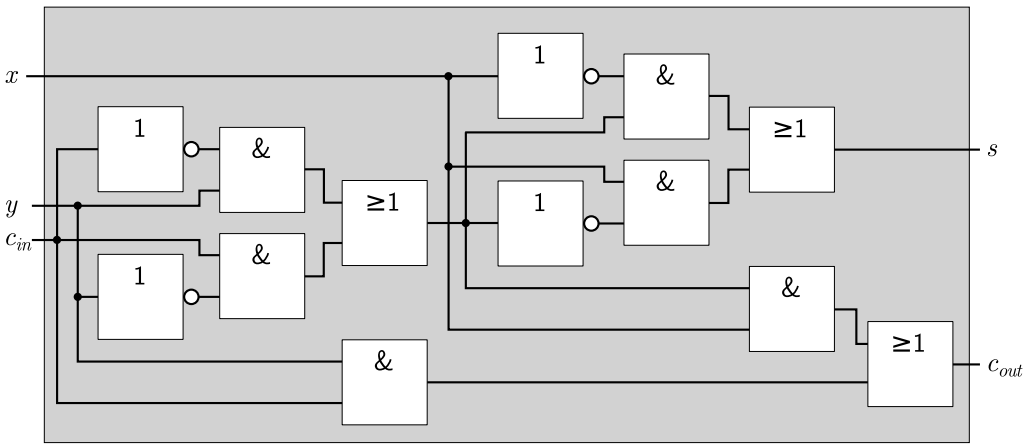


Abbildung 5.8: Volladdierer aus Gattern

Da die zweite Variante des Volladdierers keine Unterkomponenten verwendet, berechnet sich die Strukturentropie aus der Summe der Verhaltensentropien der Logikgatter. Wie bei den Halbaddierern bilden diese die Elementarkomponenten der Schaltung:

$$H_S(\mathrm{VA}_2) = 3 \cdot H_B(\mathrm{OR}) + 4 \cdot H_B(\mathrm{NOT}) + 6 \cdot H_B(\mathrm{AND})$$
$$= (3 \cdot 3 + 4 \cdot 2 + 6 \cdot 3) \cdot \log(2) \approx 10{,}54$$

Mit $H_S(\mathrm{VA}_2) \approx 10{,}54$ ist die Entwurfsentropie der zweiten Variante größer als die der ersten Variante mit $H_S(\mathrm{VA}_1) \approx 8{,}13$. Dies zeigt, dass der weitverbreitete Ansatz, Projekte in kleinere Teilprojekte aufzuteilen, grundsätzlich zu einer Reduktion der Komplexität führt. Dabei ist es das Ziel, die Teilprojekte derart zu gestalten, dass die Komplexität einfach beherrschbar ist [62, S. 18] [117, S. 11] [243, S. 1]. Die Komplexitätsreduktion zeigt sich als Ergebnis der berechneten Strukturentropien von Variante eins und zwei.

Jedoch kann die Aufteilung eines Systems in „unvorteilhafte“ Teile auch dazu führen, dass die Entwurfsentropie steigt, insbesondere dann, wenn der zusätzliche Aufwand für das Aufteilen des Systems nicht mehr durch die Wiederverwendungseinsparung kompensiert wird. Dies kann anhand der zweiten Variante des Volladdierers verdeutlicht werden, wenn dieser in zwei verschiedene Unterkomponenten (X und Y) aufgeteilt wird, wie Abbildung 5.9 als dritte Variante zeigt. Die gleichartigen Unterkomponenten sind jeweils farblich hervorgehoben.

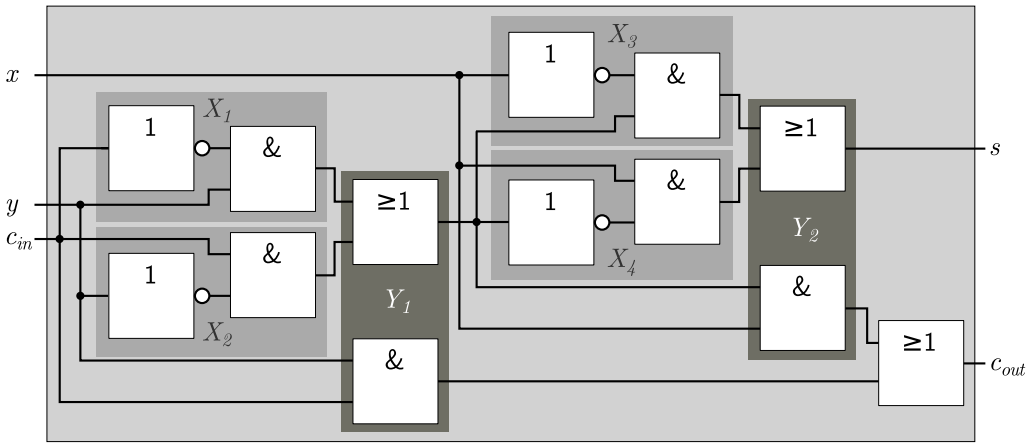


Abbildung 5.9: Volladdierer mit „unvorteilhaften“ Unterkomponenten

Wie bei der ersten Variante berechnet sich die Strukturentropie der dritten Variante über die Verhaltens- und Strukturentropien der Komponenten. Komponente X besitzt zwei Eingänge sowie zwei Ausgänge und enthält ein Nicht-Gatter sowie ein Und-Gatter. Komponente Y hat vier Eingänge und ebenfalls zwei Ausgänge. Sie besteht aus einem Oder- und einem Und-Gatter. Komponente X wird vier Mal verwendet und Komponente Y zwei Mal. Somit ergibt sich folgende Strukturentropie von Variante drei:

$$H_S(\mathrm{VA}_3) = 4 \cdot H_B(\mathrm{X}) + 2 \cdot H_B(\mathrm{Y}) + H_B(\mathrm{OR}) + H_S(\mathrm{X}) + H_S(\mathrm{Y})$$

$$= 4 \cdot \log(8) + 2 \cdot \log(64) + \log(8) + \log(32) + \log(64)$$
$$= (4 \cdot 3 + 2 \cdot 6 + 3 + 5 + 6) \cdot \log(2) \approx 11{,}44$$

Die Strukturentropie der dritten Variante fällt mit $H_S(\mathrm{VA}_3) \approx 11{,}44$ höher aus als die Strukturentropien der ersten beiden Varianten. Dies zeigt, dass nicht jede Partitionierung eines Systems und der Aufbau von Unterkomponenten zu einer Reduktion der Entwurfsentropie, mithin zu einer Aufwandsreduktion führen. Denn für den Aufbau der Unterkomponenten sowie für deren Instanziierung fällt ebenfalls Aufwand an, der durch die Strukturentropie berücksichtigt wird.

Als vierte und letzte Variante wird der Volladdierer ausschließlich aus Nicht-Und-Gattern aufgebaut. Dabei zeigt sich, dass der direkte Aufbau auch dazu führen kann, dass weniger Gatter benötigt werden. Basis des Volladdierers in Abbildung 5.10 bilden zwei Halbaddierer der Realisierung B aus Abschnitt 5.2, welche aus Nicht-Und-Gattern aufgebaut wurden.

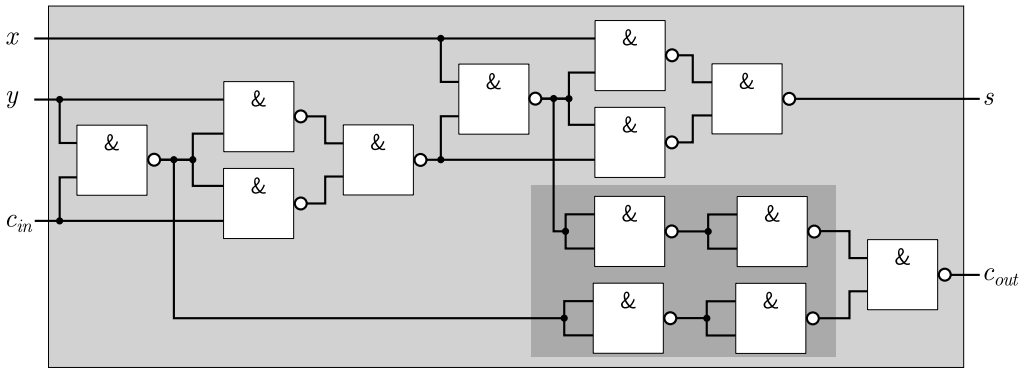


Abbildung 5.10: Volladdierer aus dreizehn Nicht-Und-Gattern

Das für den Volladdierer erforderliche Oder-Gatter beziehungsweise die Oder-Verknüpfung ist ebenfalls aus Nicht-Und-Gattern aufgebaut. Dabei gilt: $a + b = \overline{\left(\overline{(a \cdot a)} \cdot \overline{(b \cdot b)}\right)}$. Da der Übertragsausgang c des Halbaddierers bereits eine Nicht-Und-Verknüpfung besitzt, können vier Logikgatter eingespart werden, da $\overline{\overline{a}} = a$ beziehungsweise $\overline{\left(\overline{(a \cdot a)} \cdot \overline{(a \cdot a)}\right)} = a$ gilt. Die Einsparung ist in der vorherigen Abbildung 5.10 farblich hervorgehoben. Somit werden für den Volladdierer statt dreizehn nur neun Nicht-Und-Gatter benötigt, da bei der Umsetzung wohl kaum eine doppelte Negation verwendet wird. Die folgende Abbildung 5.11 zeigt den so aufgebauten Volladdierer.

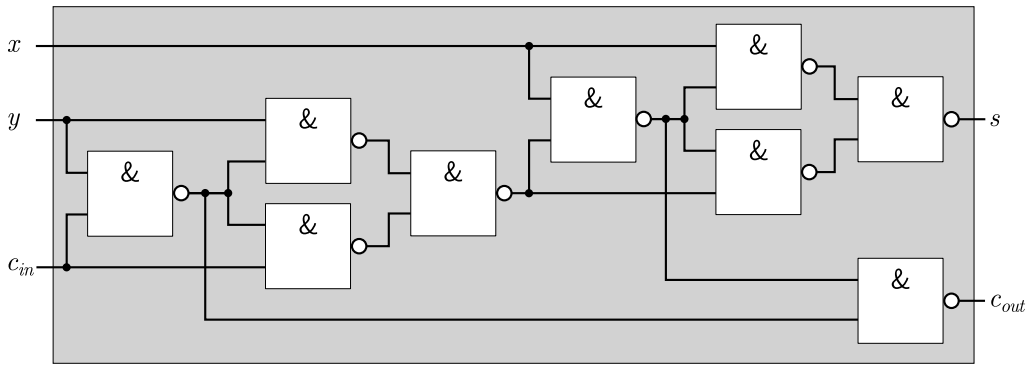


Abbildung 5.11: Volladdierer aus neun Nicht-Und-Gattern

Analog zur Berechnung der Strukturentropie des Halbaddierers aus Nicht-Und-Gattern, ergibt sich für die vierte Variante des Volladdierers aus neun Nicht-Und-Gattern eine Strukturentropie von $H_S(\mathrm{VA}_4) \approx 8{,}13$:

$$H_S(\mathrm{VA}_4) = 9 \cdot H_B(\mathrm{NAND}) = 9 \cdot 3 \cdot \log(2) \approx 8{,}13$$

Eine Übersicht der vier berechneten Werte ist in Tabelle 5.5 gegeben. Sie zeigt, dass eine sinnvolle Partitionierung eines Entwurfs zu einer Reduktion der Entwurfsentropie, mithin zur Reduktion der Komplexität führt. Jedoch resultiert nicht jede Partitionierung in einer Komplexitätsreduktion, wie die dritte Variante mit den „unvorteilhaften“ Unterkomponenten zeigt.

Variante	Aufbau	Strukturentropie
1	Zwei Halbaddierer und ein Oder-Gatter	8,13
2	Aufbau aus Gattern	10,54
3	„Unvorteilhafte“ Unterkomponenten	11,44
4	NAND-Gatter	8,13

Tabelle 5.5: Vergleich der Volladdierervarianten

Ein ähnliches Ergebnis liefert auch die Verwendung von Realisierung C des Halbaddierers aus Abschnitt 5.2, welche aus einem Antivalenz- und einem Und-Gatter besteht. Hierbei ist ebenfalls der direkte Aufbau des Volladdierers mit einem Oder-Gatter, zwei Antivalenz-Gattern und zwei Und-Gattern weniger aufwendig als der

Aufbau mit Komponenten, da die Aufwandseinsparung durch die Wiederverwendung der Halbaddiererkomponente geringer ist als die direkte Verdrahtung der fünf Logikgatter. Zwar sinkt grundsätzlich die Komplexität mit der Partitionierung in (Teil-)Komponenten, jedoch entsteht wiederum Aufwand durch die zusätzliche Arbeit der Partitionierung sowie den Entwurf und die Verwendung der Komponenten. Das heißt, der Entwurf einer Komponente erfordert ebenfalls Aufwand, wodurch die Entwurfsgröße steigt. Hieran zeigt sich, dass sowohl die Komplexität als auch die Entwurfsgröße in das Maß der Entwurfsentropie einfließen. Somit drückt die Entwurfsentropie beide Faktoren als produktspezifische Einflussfaktoren aus.

Da aus dem Volladdierer im folgenden Abschnitt ein Ripple-Carry-Addierer aufgebaut wird, wird zum Abschluss dieses Abschnitts noch dessen Verhaltensentropie berechnet. Diese bestimmt sich nach dessen Ein- und Ausgangszuständen. Der Volladdierer hat drei Eingänge und zwei Ausgänge mit jeweils zwei möglichen Zuständen [[143]]. Somit beträgt die Verhaltensentropie für den Volladdierer $H_B(\mathrm{VA}) \approx 1{,}51$, wie die folgende Rechnung zeigt [[143]]:

$$H_B(\mathrm{VA}) = \log\left(2^3 \cdot 2^2\right) = \log\left(32\right) \approx 1{,}51$$

Im folgenden Abschnitt wird aus mehreren Volladdierern ein Vier-Bit-Ripple-Carry-Addierer aufgebaut. Anhand dessen wird die Strukturentropie mit mehreren Komponenten, welche wiederum Unterkomponenten enthalten, dargestellt sowie, dass statt einzelner Ein- und Ausgänge auch Schnittstellen und Signale mit mehreren Zuständen verwendet werden können.

5.4 Ripple-Carry-Addierer

Der Ripple-Carry-Addierer erlaubt die Addition zweier mehrstelliger Binärzahlen. Hierzu werden mehrere Volladdierer derart miteinander verbunden, dass der Ausgangsübertrag des niederwertigeren Addierers als Eingangsübertrag des nächsten Addierers dient [223, S. 175]. Als Anwendungsfall liegt diesem Abschnitt ein aus vier Volladdierern aufgebauter Ripple-Carry-Addierer zugrunde. Somit können zwei vier Bit Zahlen und ein zusätzlicher Eingangsübertrag addiert werden. Die Ausgabe besteht aus der binären Summe und dem Ausgangsübertrag. Das Blockschaltbild mit den vier Volladdierern ist in der folgenden Abbildung 5.12 gezeigt.

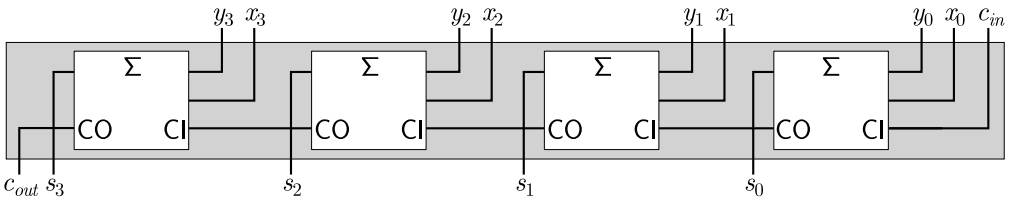


Abbildung 5.12: Ripple-Carry-Addierer [[143]]

Der Aufbau mit Komponenten wird dadurch verdeutlicht, dass die Volladdierer innerhalb des Systementwurfs umgesetzt werden. Damit die folgende Darstellung übersichtlich bleibt, wird für den Aufbau des Volladdierers die Realisierung C des Halbaddierers aus Abschnitt 5.2 mit einem Antivalenz- und einem Und-Gatter verwendet. Der vollständige Baum mit allen Komponenten und Gattern ist in Abbildung 5.13 dargestellt. Dabei stellen die meisten Komponenten Kopien („Instanzen“) dar. Da die Strukturentropie von aufgebauten Komponenten nur einmalig in die Entwurfsentropie eingeht, werden nur diejenigen Komponenten bei der Berechnung der Entwurfsentropie berücksichtigt, welche in Abbildung 5.13 hellgrau hinterlegt sind.

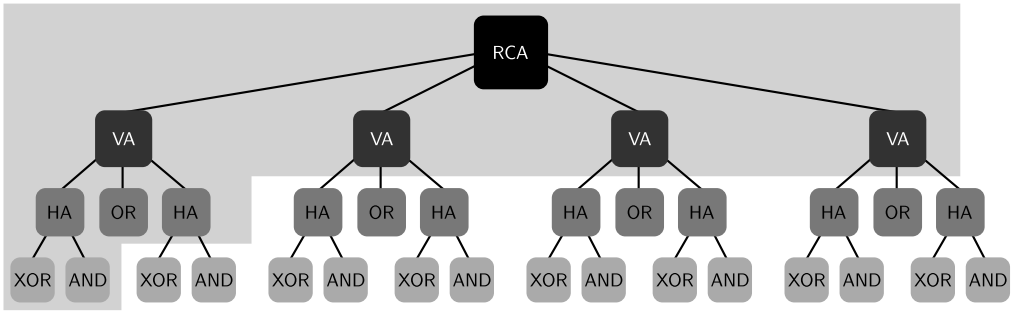


Abbildung 5.13: Komponenten des Ripple-Carry-Addierers

Die Strukturentropie des Ripple-Carry-Addierers setzt sich aus der Summe der Verhaltensentropien der Volladdiererkomponenten und der Strukturentropie des Volladdierers zusammen. Die Strukturentropie des Volladdierers setzt sich wiederum aus der Summe der Verhaltensentropien der beiden Halbaddierer und des Oder-Gatters sowie der Strukturentropie des Halbaddierers zusammen. Die Strukturentropie des Halbaddierers ist die Summe der Verhaltensentropien des Antivalenz- und des Und-Gatters. Somit ergibt sich als Strukturentropie des Ripple-Carry-Addierers:

$$
\begin{aligned}
H_S(\mathrm{RCA}) &= 4 \cdot H_B(\mathrm{VA}) + H_S(\mathrm{VA}) \\
&= 4 \cdot \underbrace{\log(32)}_{=H_B(\mathrm{VA})} + \underbrace{2 \cdot H_B(\mathrm{HA}) + H_B(\mathrm{OR}) + H_S(\mathrm{HA})}_{=H_S(\mathrm{VA})} \\
&= 4 \cdot \log(32) + 2 \cdot \underbrace{\log(16)}_{=H_B(\mathrm{HA})} + \underbrace{\log(8)}_{=H_B(\mathrm{OR})} + \underbrace{H_B(\mathrm{XOR}) + H_B(\mathrm{AND})}_{=H_S(\mathrm{HA})} \\
&= 4 \cdot \log(32) + 2 \cdot \log(16) + \log(8) + \underbrace{\log(8)}_{=H_B(\mathrm{XOR})} + \underbrace{\log(8)}_{=H_B(\mathrm{AND})} \\
&\approx 11{,}14
\end{aligned}
$$

Anhand der einzelnen Zeilen der vorherigen Berechnung zeigt sich der rekursive/ hierarchische Aufbau der Entwurfsentropie: Die Strukturentropie der direkten Unterkomponenten einer Komponente setzt sich wiederum aus den Verhaltens- und Strukturentropien der direkten „Unter-Unterkomponenten“ der Unterkomponente zusammen. Für die praktische Anwendung sowie für die automatisierte Analyse bietet es sich an, für jede aufgebaute Komponente deren Struktur- und Verhaltensentropie getrennt/vorab zu berechnen, sodass bei jeder Verwendung direkt auf das Ergebnis zurückgegriffen werden kann.

Der Ripple-Carry-Addierer eignet sich auch dazu, die Entwurfsentropieberechnung von Komponenten mit mehr als zwei Zuständen an den Schnittstellen zu zeigen [[143]]. Hierzu wird der Ripple-Carry-Addierer als abgeschlossene Komponente betrachtet, um dessen Verhaltensentropie zu berechnen. Der zu Beginn dieses Abschnitts in Abbildung 5.12 gezeigte Aufbau besitzt neun ein Bit breite Eingänge und fünf ein Bit breite Ausgänge [[143]]. Hieraus ergibt sich eine Verhaltensentropie von $H_B(\mathrm{RCA}) = (9+5) \cdot \log(2) = 14 \cdot \log(2) \approx 4{,}21$ [[143]]. Der so als abgeschlossene Komponente betrachtete Ripple-Carry-Addierer ist in Abbildung 5.14 dargestellt.

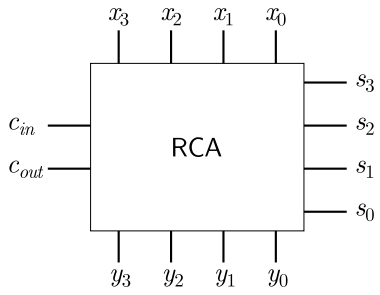


Abbildung 5.14: 1-Bit Signale [[143]]

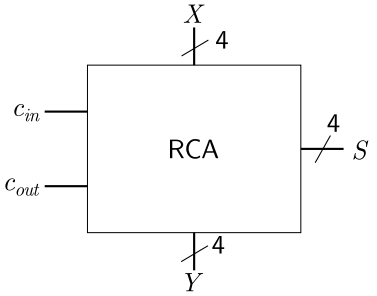


Abbildung 5.15: 4-Bit Vektoren [[143]]

Alternativ können für die beiden Eingangsbinärzahlen X und Y sowie die Binärsumme S auch vier Bit breite Vektoren verwendet werden, wie Abbildung 5.15 zeigt. Die Vektoren haben jeweils $2^4 = 16$ mögliche Zustände. Der Ein- und Ausgangsübertrag sind weiterhin jeweils ein Bit breite Signale. Eingesetzt in Formel (4.10) für die Verhaltensentropie ergibt sich nach der folgenden Rechnung eine Verhaltensentropie von $H_B(\mathrm{RCA}) \approx 4{,}21$, welche der Verhaltensentropie der Komponente mit den ein Bit breiten Verbindungen entspricht [[143]].

$$H_B(\mathrm{RCA}) = \log\left(\underbrace{2^4}_{\hat{=}X} \cdot \underbrace{2^4}_{\hat{=}Y} \cdot \underbrace{2^4}_{\hat{=}S} \cdot \underbrace{2}_{\hat{=}c_{in}} \cdot \underbrace{2}_{\hat{=}c_{out}}\right) = 14 \cdot \log(2) \approx 4{,}21$$

Hierbei zeigt sich der Vorteil der logarithmischen Funktion: Die Potenzen werden zu Multiplikatoren. Insbesondere im Bereich der digitalen Hardware können Zustände regelmäßig als Zweierpotenzen ausgedrückt werden [[141]]. Zu deren Beschreibung werden Signale eingesetzt, welche die Kommunikation modellieren und physikalischen Verbindungen entsprechen, das heißt, Drähten oder Leitungen [132, S. 65]. Diese zeichnen sich grundsätzlich durch die logischen Zustände null und eins aus. Jedoch ist es beim Entwurf teilweise auch erforderlich, Signale mit mehr als zwei Zuständen modellieren zu können, um beispielsweise die Stärke eines Signals [132, S. 68] oder unbestimmte und hochohmige Zustände zu modellieren [186, S. 75 ff.]. Da das Konzept der Entwurfsentropie ein abstrakter Ansatz ist, können auch unterschiedliche Zustandsanzahlen berücksichtigt werden [[146]].

In Abschnitt 4.10 des vorherigen Kapitels wurde beschrieben, dass für den Vergleich verschiedener Realisierungen teilweise ein Skalierungsfaktor erforderlich ist. Beispielsweise wenn bei einem Entwurf durchgängig zweiwertige Signale verwendet werden und bei einem anderen Entwurf eine neunwertige Logik. Handelt es sich ansonsten um die gleichen Entwürfe, fällt die Entwurfsentropie dennoch unterschiedlich aus, da diese die Zustandsanzahl berücksichtigt. Mit der Basisumrechnung des Logarithmus kann ein Skalierungsfaktor berechnet werden, um die Entwürfe vergleichbar zu machen. Wird beispielsweise für den Ripple-Carry-Addierer eine neunwertige Logik verwendet, ergibt sich eine Verhaltensentropie von $H_B(\mathrm{RCA}_9) = \log(9^9 \cdot 9^5) \approx 13{,}36$. Die Multiplikation des Ergebnisses mit dem Skalierungsfaktor $\left(\frac{\log_{10}(2)}{\log_{10}(9)} \approx 0{,}31546\right)$ ergibt die Verhaltensentropie für eine zweiwertige Logik: $13{,}36 \cdot 0{,}31546 \approx 4{,}21$. Diese entspricht der im Vorherigen berechneten Verhaltensentropie des Ripple-Carry-Addierers mit den zwei Bit breiten Ein- und Ausgängen. Analog kann die Skalierung auch auf die Strukturentropie angewendet werden.

Die zweite Hälfte dieses Kapitels behandelt sequenzielle Schaltungen. Dabei wird zunächst ein D-Flipflop aufgebaut, an welchem die Berechnung der Entwurfsentropie in der Verhaltens- und der Strukturdomäne gezeigt wird. Der darauf folgende Abschnitt stellt die Berechnung von verschiedenen (Zustands-)Automaten vor.

5.5 D-Flipflop

In den vorherigen Abschnitten wurden für die anschauliche Darstellung der Schaltungen Blockdiagramme basierend auf Logikbausteinen und daraus aufgebauten Komponenten verwendet. Dabei ist nicht zwischen einer Verhaltens- und einer Strukturbeschreibung unterschieden worden. Um zu zeigen, dass die Entwurfsentropie unabhängig von der jeweiligen Entwurfsdomäne angewendet werden kann, stellt dieser Abschnitt die Umsetzung eines D-Flipflops in den beiden Domänen vor. Zusätzlich werden die VHDL-Beschreibungen der Schaltungen angegeben, um die Analyse auf Basis der Umsetzungen zu zeigen.

Das Verhalten eines D-Flipflops wird durch dessen Wahrheitstabelle beschrieben, welche in Tabelle 5.6 angegeben ist. Der Ausgang Q ist von den beiden Eingängen C und D sowie dem inneren Zustand Q_0 abhängig. Liegt am Clock-Eingang C eine steigende Flanke ($\uparrow$) an, wird der Wert des Dateneingangs D gespeichert und am Ausgang Q ausgegeben. Bei allen anderen Zuständen des Clock-Eingangs (0, 1 und fallende Flanke $\downarrow$) liegt am Ausgang Q der gespeicherte Wert Q_0 an.

D	0	1	0	1
C	$\uparrow$	$\uparrow$	$0\|1\|\downarrow$	$0\|1\|\downarrow$
Q	0	1	Q_0	Q_0

Tabelle 5.6: Wahrheitstabelle eines D-Flipflops

Auf Basis der Wahrheitstabelle ist das D-Flipflop als VHDL-Verhaltensbeschreibung in Code 5.1 umgesetzt [angelehnt an 192, S. 108 f.]. Die Entitätsdeklaration in den Zeilen 1 – 6 enthält die beiden Eingangssignale C und D sowie das Ausgangssignal Q. Innerhalb der Architektur wird in den Zeilen 11 – 18 ein Prozess verwendet, um das Verhalten zu beschreiben. Liegt eine steigende Flanke an, wird das Eingangssignal D als innerer Zustand Q_0 übernommen (Zeile 13 und 14), andernfalls wird der innere Zustand beibehalten (Zeile 16). Auf die Fragen nach der

Initialisierung der Signale sowie der Sensitivität des Prozesses (sensitivity list) wird an dieser Stelle nicht eingegangen, da der Gegenstand der Analyse die tatsächliche Umsetzung ist und nicht die Frage, welcher Code in welcher Form synthetisiert werden kann und wie die Simulation abläuft. Festzuhalten ist, dass der folgende Code auch ohne Initialisierung synthetisiert und in Hardware umgesetzt werden kann.

```
1  ENTITY dff_v IS
2    PORT
3    (C : IN BIT;
4     D : IN BIT;
5     Q : OUT BIT);
6  END dff_v;
7
8  ARCHITECTURE behavior OF dff_v IS
9  SIGNAL q0 : BIT;
10 BEGIN
11   PROCESS
12   BEGIN
13     IF RISING_EDGE(C) THEN
14       q0 <= D;
15     ELSE
16       q0 <= q0;
17     END IF;
18   END PROCESS;
19   Q <= q0;
20 END behavior;
```

Code 5.1: D-Flipflop als Verhaltensbeschreibung

Die Bedingung, ob eine steigende Flanke anliegt, kann entweder wahr oder falsch sein. Hieraus ergibt sich die in Abschnitt 4.7 berechnete Verhaltensentropie des Rising_Edge-Makros von $H_B = 2 \cdot \log(2) \approx 0{,}60$, wobei vorliegend ein zusätzlicher Ausgang für den Sonst-Fall (else) berücksichtigt werden muss. Die Signalzuweisungen haben jeweils einen Eingang und einen Ausgang. Durch die Signaldefinition als Bit können diese jeweils zwei Zustände annehmen. Für die Strukturentropie ist es dabei unbeachtlich, dass es sich bei der Schaltung um ein Flipflop handelt, welches ein Bit speichern kann. Die Speichereigenschaft ergibt sich erst beim Verhalten des Flipflops als Komponente, das heißt, wenn die Schnittstellen (C, D und Q) von außen betrachtet werden. Insgesamt ergibt sich somit eine Strukturentropie $H_S(\mathrm{DFF}_v) \approx 2{,}71$ der Verhaltensbeschreibung aus der folgenden Berechnung:

$$\begin{aligned} H_S(\mathrm{DFF}_v) &= H_B(\mathrm{Rising\_Edge}) + 3 \cdot H_B(\mathrm{Signalzuweisung}) \\ &= 3 \cdot \log(2) + 3 \cdot 2 \cdot \log(2) = 9 \cdot \log(2) \approx 2{,}71 \end{aligned}$$

Als Strukturbeschreibung kann das taktflankengesteuerte D-Flipflop mithilfe von Logikbausteinen aufgebaut werden, wie Abbildung 5.16 zeigt. Vorliegend werden acht Nicht-Und-Gatter sowie ein Nicht-Gatter verwendet. Das D-Flipflop beruht auf dem Master-Slave-Prinzip, wobei die jeweils linken beziehungsweise rechten vier Nicht-Und-Gatter in Abbildung 5.16 ein D-Latch bilden. Das Clock-Signal des Master-Latches ist durch ein Nicht-Gatter invertiert.

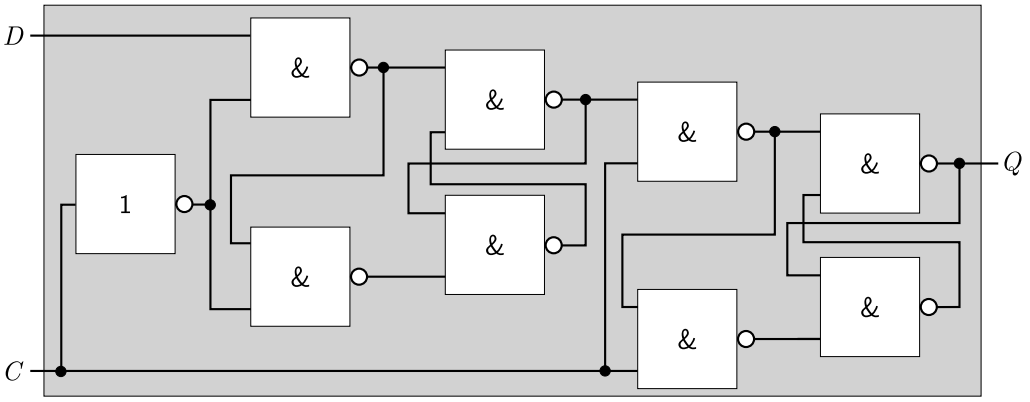


Abbildung 5.16: Logikschaltung eines D-Flipflops

Wie das Blockdiagramm besteht auch der folgende VHDL-Code 5.2 des D-Flipflops aus den Nicht-Und-Gattern und dem Nicht-Gatter. Diese dienen als Komponenten, welche in den Zeilen 9 – 16 eingebunden werden. Da sie als Elementarkomponenten verwendet werden, besitzen sie keinen eigenen Code und somit auch keine Strukturentropie. Die „Instanzen“ der Komponenten werden entsprechend dem Aufbau in den Zeilen 20 – 28 miteinander über (Zwischen-)Signale verbunden.

```
1  ENTITY dff_s IS
2    PORT
3    (C : IN    BIT;
4     D : IN    BIT;
5     Q : OUT BIT);
6  END dff_s;
7
8  ARCHITECTURE structure OF dff_s IS
9  COMPONENT nand_gate
10   PORT(a,b: IN    BIT;
11            c:   OUT BIT);
12 END COMPONENT;
13 COMPONENT inverter
14   PORT(a: IN    BIT;
15            c: OUT BIT);
16 END COMPONENT;
17 SIGNAL nc, z1, z2, z3, z4, z5, z6, z7, z8 :BIT;
18
19 BEGIN
20   INV1: inverter PORT MAP (C, nc);
21   NAND1: nand_gate PORT MAP (D,nc,z1);
22   NAND2: nand_gate PORT MAP (z1,nc,z2);
23   NAND3: nand_gate PORT MAP (z1,z4,z3);
24   NAND4: nand_gate PORT MAP (z2,z3,z4);
25   NAND5: nand_gate PORT MAP (z3,C,z5);
26   NAND6: nand_gate PORT MAP (z5,C,z6);
27   NAND7: nand_gate PORT MAP (z5,z8,z7);
28   NAND8: nand_gate PORT MAP (z6,z7,z8);
29   Q <= z7;
30 END structure;
```

Code 5.2: D-Flipflop als Strukturbeschreibung

Gemäß Abschnitt 5.1 hat ein Nicht-Und-Gatter eine Verhaltensentropie von $H_B(\text{NAND}) = 3 \cdot \log(2) \approx 0{,}90$ und ein Nicht-Gatter von $H_B(\text{NOT}) = 2 \cdot \log(2) \approx 0{,}60$. Hieraus ergibt sich die Strukturentropie der Strukturbeschreibung des D-Flipflops von $H_S(\text{DFF}_s) \approx 8{,}43$ mit $H_S(\text{DFF}_s) = H_B(\text{NOT}) + 8 \cdot H_B(\text{NAND}) + H_B(\text{Signalzuweisung})$.

In der Praxis wird regelmäßig nicht streng zwischen der Struktur- und Verhaltensbeschreibung unterschieden, sodass ein Entwurf sowohl Teile enthalten kann,

welche auf einer Strukturbeschreibung basieren, als auch Teile, welche auf einer Verhaltensbeschreibung basieren [129, S. 175]. Daher ist es beim vorherigen Aufbau naheliegend, statt der Einbindung von Gatter-Komponenten (`NAND1: nand_gate PORT MAP (D,nc,z1)`) eine direkte Verschaltung auf Basis der Verhaltensbeschreibung zu wählen (`z1 <= D NAND nc`). Wie im vorherigen Kapitel in den Abschnitten 4.5 – 4.7 dargestellt, ist die Berechnung der Entwurfsentropie unabhängig von der verwendeten Domäne. Daher ist eine Trennung der Domänen für die Berechnung nicht erforderlich und es können auch „gemischte“ Entwürfe analysiert werden. Der folgende Abschnitt, welcher sich mit den Zustandsautomaten beschäftigt, nimmt daher keine Trennung der Entwurfsdomänen bei den VHDL-Beschreibungen vor.

5.6 Zustandsautomaten

Zustandsautomaten (Finite State Machines – FSM) sind eine wesentliche Grundlage beim Entwurf und der Beschreibung digitaler Systeme [186, S. 119]. Sie werden eingesetzt, um zyklische Funktionsabläufe zu realisieren, andere Logikschaltungen zu steuern oder dienen zur Synchronisation mehrerer Komponenten und zählen zu den sequenziellen Schaltungen, da eine Abfolge von Zuständen durchlaufen wird, welche regelmäßig durch ein periodisches Taktsignal gesteuert wird [186, S. 119]. Zustandsautomaten können als Moore- oder Mealy-Automaten dargestellt werden [186, S. 119], auf welche sich die folgenden Beispiele konzentrieren.

Beide Automaten (Moore und Mealy) lassen sich in drei Funktionsblöcke einteilen: das Übergangsschaltnetz, den Zustandsspeicher und das Ausgangsschaltnetz, wobei das Erst- und Letztgenannte rein kombinatorische Schaltungen sind [186, S. 120]. Das Übergangsschaltnetz legt den Folgezustand fest, während das Ausgangsschaltnetz die Ausgaben bestimmt [186, S. 120]. Der Zustandsspeicher hat die Aufgaben, den aktuellen Zustand zu speichern und in Abhängigkeit des Taktsignals den Automaten in den nächsten Zustand zu bringen [186, S. 120]. Regelmäßig hat der Zustandsspeicher als weitere Eingangssignale ein Reset- und ein Aktivsignal (Enable) [186, S. 120 f.], auf welche im Folgenden zum Zweck der Übersichtlichkeit verzichtet wird. Für einen Moore-Automaten sind die drei Funktionsblöcke und ihre Beziehungen in Abbildung 5.17 dargestellt. Die Signale a, e, z und z^+ sind dabei als Vektoren angegeben, das heißt, als logische Gruppen zusammengefasster Mehrfachsignalleitungen [186, S. 120].

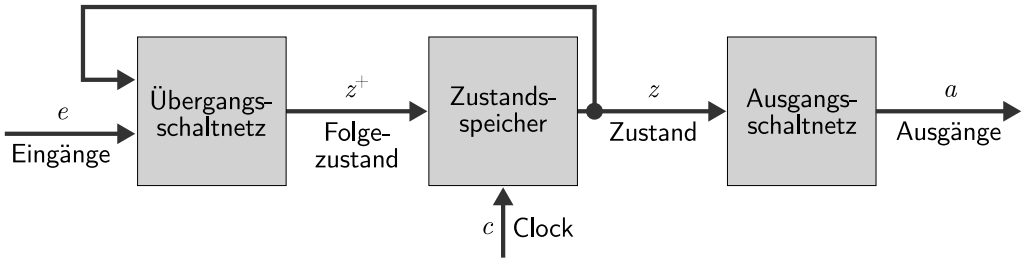


Abbildung 5.17: Moore-Automat [angelehnt an 186, S. 120]

Charakteristisch für den Moore-Automaten ist, dass dessen Ausgaben ausschließlich von seinem inneren Zustand abhängen [186, S. 119]. Das heißt, die Ausgangssignale a sind eine Funktion des aktuellen Zustands z: $a = f(z)$ [186, S. 120]. Ein Zustandsübergang in den Folgezustand z^+ erfolgt bei einer Taktflanke und ist eine Funktion der Eingangssignale e und des aktuellen Zustands z: $z^+ = f(e, z)$ [186, S. 120]. Hieraus ergeben sich folgende Beziehungen der einzelnen Funktionsblöcke:

- Übergangsschaltnetz:
  Der Folgezustand z^+ wird mithilfe einer kombinatorischen Logik auf Basis des aktuellen Zustands z und der Eingangssignale e vorausberechnet [186, S. 120].

- Zustandsspeicher:
  Mit jeder (positiven und/oder negativen) Flanke des Taktsignals c wird der Automat vom Zustand z in den Zustand z^+ überführt und dieser Zustand im Zustandsspeicher gespeichert [186, S. 120].

- Ausgangsschaltnetz:
  Anhand des aktuellen Zustands z werden mithilfe einer kombinatorischen Logik die Ausgangssignale a berechnet [186, S. 120].

Beim Moore-Automaten bestimmen somit die Eingangssignale e die Ausgangssignale a nur mittelbar über den Zustand z [186, S. 121]. Bei einem Mealy-Automaten sind die Eingangssignale e dagegen direkt mit dem Ausgangsschaltnetz verbunden, sodass die Ausgabe a eine Funktion des aktuellen Zustands z und der Eingabesignale $a = f(e, z)$ ist [186, S. 121], wie in Abbildung 5.18 zeigt. Der Folgezustand z^+ ist auch beim Mealy-Automaten eine Funktion der Eingangssignale e und des aktuellen Zustands z: $z^+ = f(e, z)$ [186, S. 121].

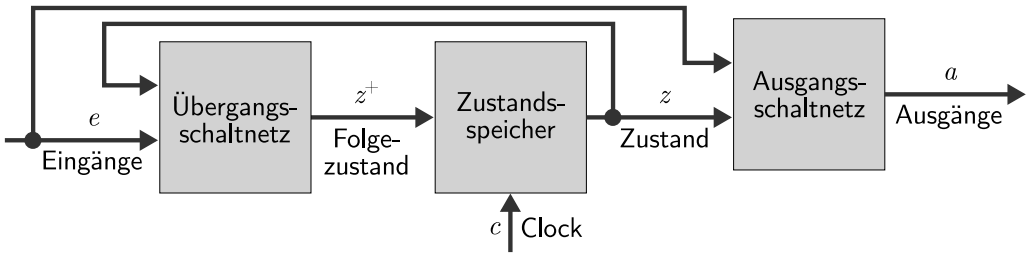


Abbildung 5.18: Mealy-Automat [angelehnt an 186, S. 121]

Zur Verdeutlichung des Unterschieds sind die Grundstrukturen beider Automaten in den Codebeispielen 5.3 und 5.4 gegeben. Die drei Blöcke der schematischen Darstellungen aus den Abbildungen 5.17 und 5.18 wurden jeweils als eigene Prozesse umgesetzt.

```
1  ENTITY moore IS
2   PORT( c : IN  BIT;
3         e : IN  BIT_VECTOR(0 TO X);
4         a : OUT BIT_VECTOR(0 TO Y) );
5  END moore;
6
7  ARCHITECTURE behavior OF moore IS
8  TYPE zustaende IS (z1,z2,...,zn);
9  SIGNAL zustand, folgezustand: zustaende;
10
11 BEGIN
12  zustandsspeicher: PROCESS(c)
13  BEGIN
14   IF (c='1' AND c'EVENT) THEN
15    zustand <= folgezustand;
16   END IF;
17  END PROCESS;
18
19  Uebergangsschaltnetz: PROCESS(e, zustand)
20  BEGIN
21   CASE zustand IS
22    WHEN z1 => IF e=... THEN
23     folgezustand <= ...; END IF;
24    ...
25    WHEN zn => IF e=... THEN
26     folgezustand <= ...; END IF;
27   END CASE;
28  END PROCESS;
29
30  Ausgangsschaltnetz: PROCESS(zustand)
31  BEGIN
32   CASE zustand IS
33    WHEN z1 =>
34     a <= ...;
35    ...
36    WHEN zn =>
37     a <= ...;
38   END CASE;
39  END PROCESS;
40 END behavior;
```

Code 5.3: Moore-Automat in VHDL

```
1  ENTITY mealy IS
2   PORT( c : IN  BIT;
3         e : IN  BIT_VECTOR(0 TO X);
4         a : OUT BIT_VECTOR(0 TO Y) );
5  END mealy;
6
7  ARCHITECTURE behavior OF mealy IS
8  TYPE zustaende IS (z1,z2,...,zn);
9  SIGNAL zustand, folgezustand: zustaende;
10
11 BEGIN
12  zustandsspeicher: PROCESS(c)
13  BEGIN
14   IF (c='1' AND c'EVENT) THEN
15    zustand <= folgezustand;
16   END IF;
17  END PROCESS;
18
19  Uebergangsschaltnetz: PROCESS(e, zustand)
20  BEGIN
21   CASE zustand IS
22    WHEN z1 => IF e=... THEN
23     folgezustand <= ...; END IF;
24    ...
25    WHEN zn => IF e=... THEN
26     folgezustand <= ...; END IF;
27   END CASE;
28  END PROCESS;
29
30  Ausgangsschaltnetz: PROCESS(e, zustand)
31  BEGIN
32   CASE zustand IS
33    WHEN z1 =>
34     IF e=... THEN a <= ...; END IF;
35    ...
36    WHEN zn =>
37     IF e=... THEN a <= ...; END IF;
38   END CASE;
39  END PROCESS;
40 END behavior;
```

Code 5.4: Mealy-Automat in VHDL

Wie angesprochen, besteht der Unterschied zwischen den Automaten darin, dass beim Mealy-Automaten die Ausgangssignale auch von den Eingangssignalen abhängen. Dies zeigt sich in den beiden Codebeispielen (Codes 5.3 und 5.4) darin, dass die Ausgangsschaltnetze unterschiedlich umgesetzt sind. Daher wird im Folgenden untersucht, ob sich dieser Unterschied auch auf die Entwurfsentropie der Automaten auswirkt. Hierzu wird als Beispiel eine Impulsfolgenerkennung betrachtet [Beispiel angelehnt an 186, S. 122 ff.]. Wenn die Folge „0,1,1“ nacheinander an einem ein Bit breiten Eingang anliegt, dann soll der Ausgang einen Taktzyklus eins sein. Dabei wird das Eingangssignal ein Mal pro Taktzyklus „gelesen“.

Für die Umsetzung als Moore-Automat werden vier Zustände benötigt, sodass sich der in Abbildung 5.19 gezeigte Automat ergibt. Wie im Vorherigen wurde für die Übersichtlichkeit auf ein Reset- und ein Aktivsignal verzichtet, sodass der Automat keinen Startknoten beziehungsweise eine Startkante besitzt. Daher wird angenommen, dass sich der Automat zu Beginn im Zustand z_1 befindet.

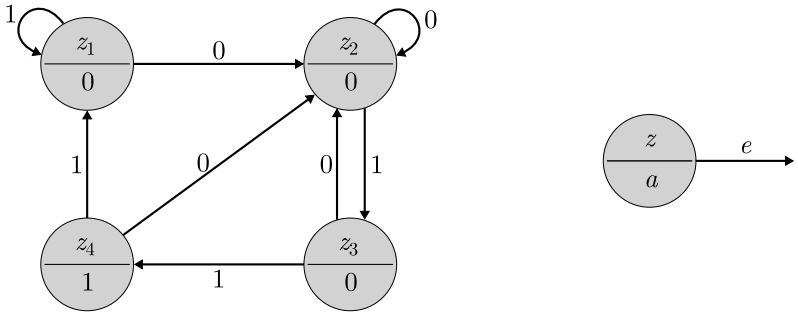


Abbildung 5.19: Moore-Automat Impulsfolgenerkennung [angelehnt an 186, S. 122]

Wenn sich der Automat im Zustand z_1 befindet, dann bleibt er in diesem Zustand, wenn am Eingang e eine Eins anliegt. Bei einer Null geht er vom Zustand z_1 in z_2 über, in welchem er so lange verbleibt, wie eine Null am Eingang e anliegt. Bei einer Eins am Eingang e geht er von Zustand z_2 in z_3 über, von welchem er bei einer weiteren Eins in den Zustand z_4 übergeht. Liegt im Zustand z_3 oder z_4 eine Null an, wechselt der Automat in den Zustand z_2. Im Zustand z_4 ist die Eingabefolge vollständig und es wird am Ausgang a für einen Taktzyklus eine Eins ausgegeben. In allen anderen Zuständen ist die Ausgabe a null. Liegt im Zustand z_4 eine Eins am Eingang e an, geht er in den Zustand z_1 über. Dieses Verhalten wird durch den folgenden VHDL-Code 5.5 umgesetzt, welcher, wie die Grundstruktur des Moore-Automaten in Code 5.3, aus drei Prozessen besteht.

```
1  ENTITY ifeMoore IS
2   PORT( clk : IN  BIT;
3         e   : IN  BIT;
4         a   : OUT BIT
5        );
6  END ifeMoore;
7
8  ARCHITECTURE behavior OF ifeMoore IS
9   TYPE zustaende IS (z1,z2,z3,z4);
10  SIGNAL zustand, folgezustand: zustaende;
11
12 BEGIN
13  zustandsspeicher: PROCESS
14  BEGIN
15   IF RISING_EDGE(clk) THEN
16    zustand <= folgezustand;
17  END IF;
18  END PROCESS;
19
20  Uebergangsschaltnetz: PROCESS
21  BEGIN
22   CASE zustand IS
23    WHEN z1 =>
24     IF e='1'
25      THEN folgezustand <= z1;
26      ELSE folgezustand  <= z2;
27     END IF;
28
29    WHEN z2 =>
30     IF e='1'
31      THEN folgezustand <= z3;
32      ELSE folgezustand  <= z2;
33     END IF;
34
35    WHEN z3 =>
36     IF e='1'
37      THEN folgezustand <= z4;
38      ELSE folgezustand  <= z2;
39     END IF;
40
41    WHEN z4 =>
42     IF e='1'
43      THEN folgezustand <= z1;
44      ELSE folgezustand  <= z2;
45     END IF;
46   END CASE;
47  END PROCESS;
48
49  Ausgangsschaltnetz: PROCESS
50  BEGIN
51   CASE zustand IS
52    WHEN z4 =>           a <= '1';
53    WHEN OTHERS =>  a <= '0';
54   END CASE;
55  END PROCESS;
56 END behavior;
```

Code 5.5: VHDL-Code der Impulsfolgenerkennung als Moore-Automat

Die Berechnung der Entwurfsentropie erfolgt mithilfe der Komponenten des Automaten, welche in Abbildung 5.20 gezeigt sind. Die Eingänge mit konstanten Signalen sind für eine bessere Übersichtlichkeit nicht dargestellt. Für jede Komponente ist in der Abbildung die Verhaltensentropie multipliziert mit der Anzahl an Verwendungen der Komponente angegeben.

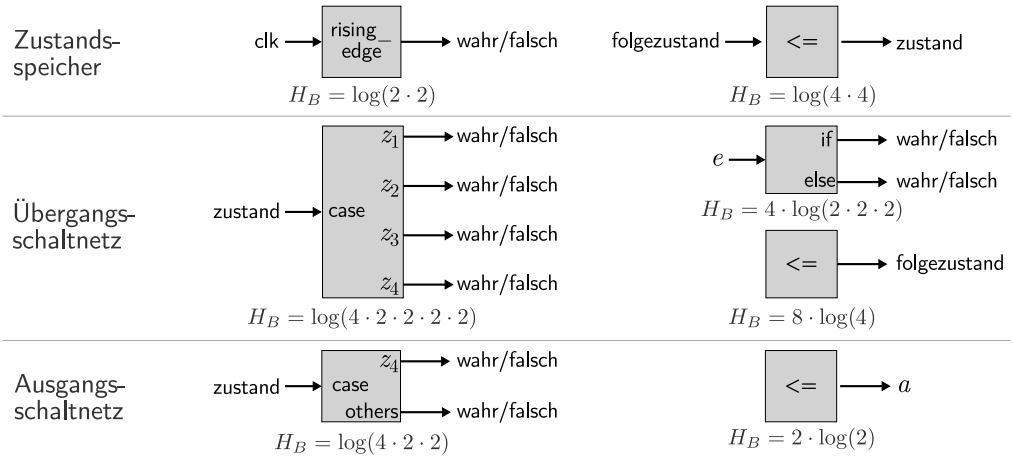


Abbildung 5.20: Komponenten des Moore-Automaten der Impulsfolgenerkennung

Anhand der Komponenten der vorherigen Abbildung 5.20 berechnet sich die Strukturentropie als Summe der Verhaltensentropien. Da es sich bei allen Komponenten um Elementarkomponenten handelt, muss für keine dieser Komponenten eine Strukturentropie berücksichtigt werden. Somit ergibt sich eine Entwurfsentropie der Impulsfolgenerkennung des Moore-Automaten von $H_S(\text{IFE}_{\text{Moore}}) \approx 13{,}85$.

Zum Vergleich ist die Impulsfolgenerkennung als Mealy-Automat in der folgenden Abbildung 5.21 gezeigt. Dadurch, dass die Kanten nicht nur die Eingangssignale, sondern auch die Ausgangssignale abbilden, werden nur drei Zustände benötigt. Der vierte Zustand des Moore-Automaten entspricht der Kante vom dritten zum ersten Zustand beim Mealy-Automaten.

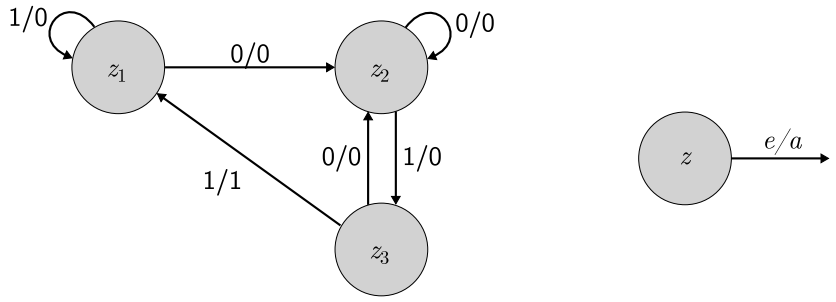


Abbildung 5.21: Mealy-Automat Impulsfolgenerkennung [angelehnt an 186, S. 127]

Für die Umsetzung des Mealy-Automaten werden somit nur drei Zustände (z_1, z_2 und z_3) benötigt, wie der folgende Code 5.6 zeigt. Er beruht auf der Grundstruktur des Mealy-Automaten in Code 5.4, welcher aus drei Prozessen besteht. Da der vierte Zustand des Moore-Automaten beim Mealy-Automaten innerhalb des Ausgangsschaltnetzes umgesetzt wird, reduziert sich nicht nur die Anzahl an Automatenzuständen, sondern es werden auch weniger Bedingungen/Vergleiche als beim Moore-Automaten benötigt.

Die Berechnung der Entwurfsentropie der Mealy-Umsetzung erfolgt analog zum Moore-Automaten. Das heißt, es werden die Komponenten der Umsetzung betrachtet, welche ähnlich wie die Komponenten aus Abbildung 5.20 ausfallen, mit den Hauptunterschieden, dass sich die Anzahl an Schnittstellen sowie die Zustandsanzahl bei einigen Komponenten reduziert. Aus der Summe der Verhaltensentropien der Komponenten ergibt sich eine Strukturentropie des Mealy-Automaten von $H_S(\text{IFE}_{\text{Mealy}}) \approx 10{,}79$. Die Entwurfsentropie fällt somit geringer aus als beim Moore-Automaten mit $H_S(\text{IFE}_{\text{Moore}}) \approx 13{,}85$. Da der Mealy-Automat nur drei statt vier Zustände umsetzt, ist dieses Ergebnis plausibel.

```
1  ENTITY ifeMealy IS
2   PORT( clk : IN  BIT;
3         e   : IN  BIT;
4         a   : OUT BIT
5        );
6  END ifeMealy;
7
8  ARCHITECTURE behavior OF ifeMealy IS
9  TYPE zustaende IS (z1,z2,z3);
10 SIGNAL zustand, folgezustand: zustaende;
11
12 BEGIN
13  zustandsspeicher: PROCESS
14  BEGIN
15   IF RISING_EDGE(clk) THEN
16    zustand <= folgezustand;
17   END IF;
18  END PROCESS;
19
20  Uebergangsschaltnetz: PROCESS
21  BEGIN
22   CASE zustand IS
23    WHEN z1 =>
24     IF e='1'
25      THEN folgezustand <= z1;
26      ELSE folgezustand  <= z2;
27     END IF;
28
29    WHEN z2 =>
30     IF e='1'
31      THEN folgezustand <= z3;
32      ELSE folgezustand  <= z2;
33     END IF;
34
35    WHEN z3 =>
36     IF e='1'
37      THEN folgezustand <= z1;
38      ELSE folgezustand  <= z2;
39     END IF;
40   END CASE;
41  END PROCESS;
42
43  Ausgangsschaltnetz: PROCESS
44  BEGIN
45   IF (zustand=z3)
46    IF(e='1')
47     THEN a <= '1';
48     ELSE a <= '0';
49    END IF;
50   END IF;
51  END PROCESS;
52 END behavior;
```

Code 5.6: VHDL-Code der Impulsfolgenerkennung als Mealy-Automat

Wenn jedoch die Ausgaben der beiden Umsetzungen betrachtet werden, weisen diese ein unterschiedliches Verhalten auf [angelehnt an 186, S. 126 ff.]. Hierzu ist der Signalverlauf in Abbildung 5.22 gezeigt. Eingangs wurde festgelegt, dass das Ausgangssignal einen Takt eine Eins sein soll. Dadurch, dass beim Mealy-Automaten das Ausgangssignal vom Eingang abhängt, folgt dieses im Zustand z_3 dem Eingang, sodass einerseits das Ausgangssignal bereits eins ist, obwohl die Eingangssequenz noch nicht zwei Takte lang eins war. Andererseits ist nicht garantiert, dass das Ausgangssignal für genau einen Takt lang eins ist.

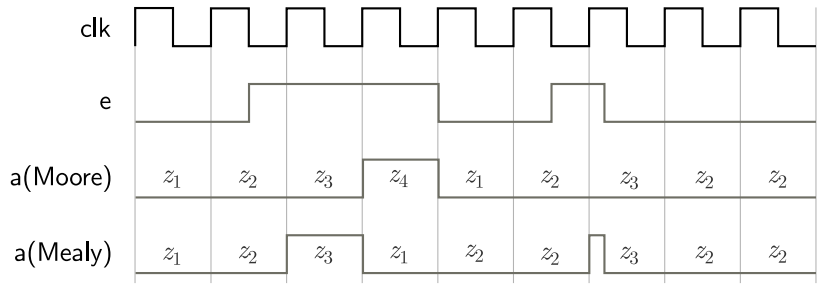


Abbildung 5.22: Signalverlauf der Impulsfolgenerkennung

Um die Taktsynchronizität des Mealy-Automaten zu erreichen, ist ein weiterer Zustand erforderlich, wie Abbildung 5.23 zeigt. Dadurch kann erreicht werden, dass

der Übergang vom Zustand z_4 zum Zustand z_1 unabhängig vom Eingangssignal ist. Hieran zeigt sich, dass es nicht das Ziel der Entwurfsentropie ist, sicherzustellen, dass auch die geforderte Funktionalität umgesetzt wird. Grundlage der Analyse ist allein die tatsächliche Umsetzung.

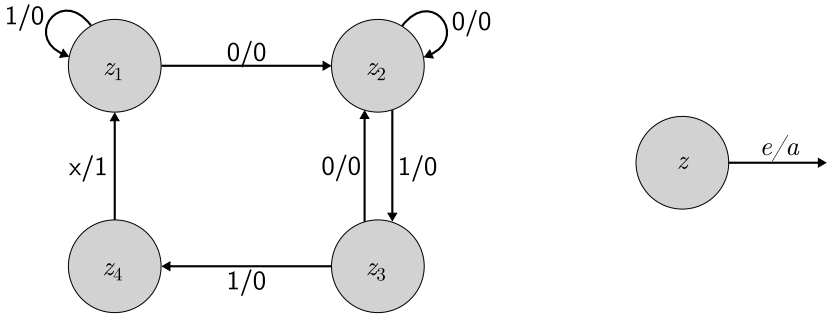


Abbildung 5.23: Mealy-Automat mit vier Zuständen

Da in der digitalen Hardware Zustandsautomaten regelmäßig mithilfe von zwei Prozessen als sogenannte Huffman-Normalform beschrieben werden [186, S. 133], kommt es in der Regel zu einem Gleichlauf zwischen den Moore- und Mealy-Automaten. Bei der Huffman-Normalform sind das Übergangs- und das Ausgangsschaltnetz als Schaltnetz zusammengefasst [186, S. 133]. Der Vorteil ist, dass die Fallunterscheidung (case ... when) für den aktuellen Zustand z nur ein Mal implementiert werden muss [186, S. 134]. Die in Abbildung 5.24 dargestellte Huffman-Normalform eines Zustandsautomaten zeigt die Wirkungspfeile im Schaltnetzblock für beide Automaten, wobei die gestrichelte Linie nur beim Mealy-Automaten existiert [186, S. 133].

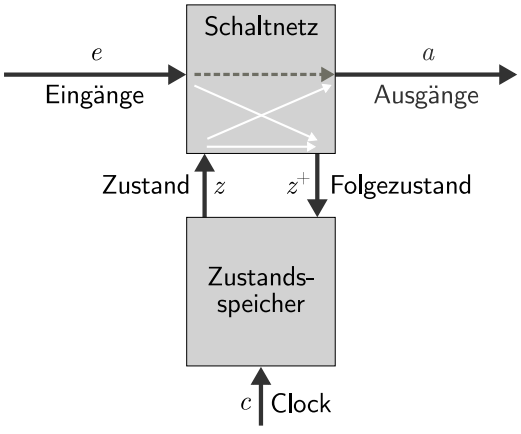


Abbildung 5.24: Huffman-Normalform [angelehnt an 186, S. 133]

Für die Impulsfolgenerkennung ist die VHDL-Beschreibung als Huffman-Modell in Code 5.7 gegeben. Dabei ist das Ausgangssignal im Zustand z_4 unabhängig vom Eingangssignal, sodass gewährleistet ist, dass das Ausgangssignal genau einen Takt eins ist. Basierend auf den Komponenten aus Abbildung 5.20 ergibt sich für die Umsetzung mit zwei Prozessen eine Strukturentropie von $H_S(\text{Huffman}) \approx 13{,}25$. Diese ist damit geringer als die Strukturentropie der Moore-Umsetzung, welche ebenfalls vier Zustände verwendet.

```
1  ENTITY ifeHuffman IS
2   PORT( clk : IN  BIT;
3         e   : IN  BIT;
4         a   : OUT BIT
5       );
6  END ifeHuffman;
7
8  ARCHITECTURE behavior OF ifeHuffman IS
9  TYPE zustaende IS (z1,z2,z3,z4);
10 SIGNAL zustand, folgezustand: zustaende;
11
12 BEGIN
13  zustandsspeicher: PROCESS
14  BEGIN
15   IF RISING_EDGE(clk) THEN
16    zustand <= folgezustand;
17   END IF;
18  END PROCESS;
19
20  Uebergangsschaltnetz: PROCESS
21  BEGIN
22   CASE zustand IS
23    WHEN z1 =>
24     a <= '0';
25     IF e='1'
26      THEN folgezustand <= z1;
27      ELSE folgezustand  <= z2;
28     END IF;
29
30    WHEN z2 =>
31     a <= '0';
32     IF e='1'
33      THEN folgezustand <= z3;
34      ELSE folgezustand  <= z2;
35     END IF;
36
37    WHEN z3 =>
38     a <= '0';
39     IF e='1'
40      THEN folgezustand <= z1;
41      ELSE folgezustand  <= z2;
42     END IF;
43
44    WHEN z4 =>
45     a <= '1';
46     IF e='1'
47      THEN folgezustand <= z1;
48      ELSE folgezustand  <= z2;
49     END IF;
50   END CASE;
51  END PROCESS;
52 END behavior;
```

Code 5.7: VHDL-Code der Impulsfolgenerkennung mit 2 Prozessen

Wie beschrieben, liegt der Analyse die jeweilige Umsetzung zugrunde, unabhängig davon, ob diese die effizienteste Umsetzung darstellt. Für die Impulsfolgenerkennung wäre die naheliegende Umsetzung die Verwendung eines Schieberegisters, wie Abbildung 5.25 zeigt.

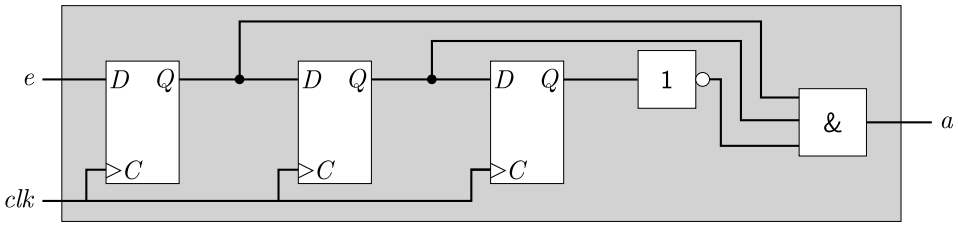


Abbildung 5.25: Blockdiagramm der Impulsfolgenerkennung

Die D-Flipflops sind so miteinander verschaltet, dass der Ausgang des vorherigen Flipflops das Eingangssignal des nächsten Flipflops bildet. Mit jeder Taktflanke werden die Speicherzustände weitergereicht. Das erste Flipflop hat als Eingabe das Eingangssignal. Wenn die gesuchte Sequenz gespeichert ist, wird durch das Und-Gatter eine Eins ausgegeben. Die Taktung gewährleistet dabei, dass das Ausgangssignal genau einen Takt lang eins ist. Der folgende Code 5.8 zeigt die Umsetzung der Impulsfolgenerkennung als VHDL-Beschreibung.

```
1  ENTITY ifeRegister IS
2    PORT( clk : IN    BIT;
3          e   : IN    BIT;
4          a   : OUT BIT );
5  END ifeRegister;
6
7  ARCHITECTURE structure OF ifeRegister IS
8  COMPONENT dff
9  PORT (c : IN BIT;
10         d : IN BIT;
11         q : OUT BIT);
12 END COMPONENT;
13 SIGNAL z1, z2, z3: BIT;
14
15 BEGIN
16   ff1: dff PORT MAP (clk, e,z1);
17   ff2: dff PORT MAP (clk,z1,z2);
18   ff3: dff PORT MAP (clk,z2,z3);
19   a <= z1 AND z2 AND NOT z3;
20 END structure;
```

Code 5.8: Impulsfolgenerkennung als Schieberegister

Für das D-Flipflop der Schieberegisterschaltung kann die Verhaltensbeschreibung des D-Flipflops aus dem vorherigen Abschnitt 5.5 verwendet werden. Für dieses wurde eine Strukturentropie von $H_S(\text{DFF}) = 2{,}71$ berechnet. Wie in Abschnitt 4.8 über sequenzielle Elemente beschrieben, werden die inneren Zustände eines Speicherbausteins als Ausgangszustände betrachtet. Somit weißt jedes D-Flipflop eine Verhaltensentropie von $H_B(\text{DFF}) = \log(2) \cdot \log(2) \cdot \log(3) \approx 1{,}08$ auf. Zusammen mit der Und-Verknüpfung für den Ausgang a ergibt sich somit eine Entwurfsentropie für die Schieberegisterschaltung von $H_S(\text{Schieberegister}) \approx 7{,}75$. Dabei zeigt sich, dass die naheliegendste Umsetzung mit den Schieberegistern zugleich diejenige Schaltung ist, welche die geringste Entwurfsentropie aufweist. Das heißt, die Entwurfsentropie kann die Einschätzung der Schaltungen direkt in Zahlenwerten ausdrücken.

Wie bei den vorherigen Schaltungen ist es somit nicht das Ziel der Entwurfsentropie, die effizienteste Umsetzung, die minimale Automatenzustandsanzahl oder alle möglichen Automatenzustände anzugeben, sondern die tatsächliche Umsetzung zu bewerten. Folglich ist die Grundlage immer die tatsächliche Umsetzung, sodass die Analyse von Zustandsautomaten dem Grundprinzip der Entwurfsentropie folgt, dass die jeweilige Implementierung analysiert wird. Es ist nicht erforderlich, dass alle möglichen Zustände eines Systems, wie beispielsweise eines Prozessors, bekannt sein müssen. Die einzelnen Komponenten, wie Pipelines, Speicher etc. können zunächst getrennt analysiert werden. Dies bildet den Entwurf ab, bei

welchem die Teile einzeln entworfen werden. Eine übergeordnete Logik fasst dann diese Komponenten zu einer Einheit zusammen. Diese kann wiederum mithilfe der Entwurfsentropie analysiert werden, sodass die einzelnen Teile die Komponenten darstellen.

Die Analyse der Zustandsautomaten erfolgt anhand der verwendeten Komponenten. Wie bei den Schaltungen der vorherigen Abschnitte ist es auch möglich, Unterkomponenten zu verwenden, die innerhalb des Entwurfs aufgebaut werden, sodass zusätzlich deren Strukturentropie berücksichtigt werden muss. Zudem zeigte sich bei der Analyse der Zustandsautomaten, dass auch Zustandsanzahlen auftreten, welche nicht als Zweierpotenzen ausgedrückt werden können. Durch die abstrakte Formulierung der Entwurfsentropie sind keine Anpassungen für die unterschiedlichen Zustandsanzahlen oder für die Analyse der Zustandsautomaten im Allgemeinen erforderlich.

5.7 Zwischenfazit

Ziel dieses Kapitels war es, die Berechnung der Entwurfsentropie anschaulich darzustellen. Hierzu wurden im Verlauf mehrere kompakte Hardwareentwürfe vorgestellt, anhand welcher unterschiedliche Aspekte der Entwurfsentropieberechnung gezeigt wurden. Auch bei umfangreicheren Entwürfen kann die Berechnung auf diese grundlegenden Beispiele zurückgeführt werden. Dabei führt deren Größe jedoch dazu, dass die Darstellung der Berechnungen wesentlich umfangreicher ausfällt und durch die Vermischung der hier einzeln gezeigten Arbeitsweisen diese nur aufwendig dargestellt werden können. Aus diesen Gründen wurden innerhalb dieses Kapitels die Beispielentwürfe so gewählt, dass die wesentlichen Aspekte der Berechnung gezeigt werden konnten, ohne dabei die Anschaulichkeit zu vernachlässigen.

Zusammenfassend konnte innerhalb des vorliegenden Kapitels Folgendes veranschaulicht werden:

- Der Analyse liegt immer die tatsächliche Umsetzung zugrunde. Es ist nicht das Ziel der Entwurfsentropie, die minimale Schaltung, die effizienteste Umsetzung und/oder alle möglichen Zustände zu finden.

- Die Formeln zur Berechnung der Entwurfsentropie können auf unterschiedliche Beschreibungen von Entwürfen angewendet werden.

- Die Entwurfsentropie drückt die Komplexität und Größe eines Entwurfs als quantitative Größe aus.

- Verschiedene Umsetzungen können miteinander verglichen werden.

- Die Berechnung der Entwurfsentropie basiert auf dem Komponentenansatz, welcher in verschiedenen Domänen und auf unterschiedliche Ebenen angewendet werden kann.

- Das Maß berücksichtigt die Wiederverwendung von Komponenten und die Partitionierung von Entwürfen.

- Die Aufteilung eines Entwurfs wird in dem Maße berücksichtigt, wie diese sich auf die Struktur oder die Partitionierung auswirkt.

- Innere Zustände werden durch die Anzahl möglicher Zustände an den Ausgängen berücksichtigt.

- Es können sowohl Entwürfe analysiert werden, die als Strukturbeschreibung gegeben sind, als auch Entwürfe, die als Verhaltensbeschreibung gegeben sind. Zudem ist eine Trennung zwischen den Domänen nicht erforderlich, da die Entwurfsentropie auch Entwürfe analysieren kann, welche Teile aus beiden Ebenen verwenden sowie Entwürfe, die auf unterschiedlichen Abstraktionsebenen erfolgen.

- Die Entwurfsentropie findet gleichermaßen Anwendung auf kombinatorische wie auf sequenzielle Schaltungen.

Um zu zeigen, dass die Entwurfsentropie auch umfangreichere Entwürfe analysieren kann, werden im folgenden Kapitel mehrere Fallstudien vorgestellt. Dabei werden wiederum verschiedene Aspekte des digitalen Hardwareentwurfs aufgegriffen und anhand der Fallstudien aufgezeigt. Während innerhalb des vorliegenden Kapitels die Berechnung und die Funktionsweise der Entwurfsentropie im Vordergrund standen, stehen im folgenden Kapitel die Ergebnisse der Berechnungen im Vordergrund. Das heißt, es wird gezeigt, wie anhand der (quantitativen) Ergebnisse die Entwürfe miteinander verglichen werden können und welche Aussagen die Werte im Bezug auf einen Entwurf zulassen.

6 Fallstudien

Die Entwurfsentropie wird in diesem Kapitel auf verschiedene Hardwarebeschreibungen innerhalb mehrerer Fallstudien angewendet, um der Frage nachzugehen, welche Aussagen die gewonnenen Entropiewerte zulassen. Der erste Abschnitt beschäftigt sich mit der Regelmäßigkeit von Strukturen im Zusammenhang mit der Entwurfslücke. Der darauf folgende Abschnitt berechnet die Entwurfsentropie für die Verbindung mehrerer Hardwarekomponenten, um festzustellen, wann ein Tristatebus effizienter ist als ein Multiplexverfahren. Die folgenden beiden Abschnitte untersuchen Projekte von Studierendengruppen und vergleichen deren Entwurfsentropien. Für die Analysen innerhalb dieses Kapitels wird das in Abschnitt 4.9 vorgestellte Analysewerkzeug eingesetzt, welches die Entwurfsentropie von VHDL-Umsetzungen berechnet.

6.1 Regelmäßigkeit von Strukturen

Zu Beginn der Arbeit wurde im Zusammenhang mit der Bedeutung der Komplexitätsbestimmung für die Wirtschaft die Entwurfslücke vorgestellt (Abschnitt 2.3). Das Diagramm ist als Abbildung 6.1 an dieser Stelle noch einmal wiedergegeben. Es zeigt zum einen das Mooresche Gesetz, welches die Entwicklung der Transistoranzahl pro Chip über die Zeit abbildet. Zum anderen ist die Anzahl an umgesetzten Transistoren pro Person und Monat ebenfalls über die Zeit aufgetragen. Um die unterschiedliche Entwicklung darzustellen, ist die gleiche logarithmische Skalierung gewählt, wobei sich die beiden Kurven im Jahr 1981 schneiden. Der Abstand zwischen den Kurven wird Entwurfslücke genannt [227, S. 36] [228, S. 8] [229, S. 8]. Die Grafik zeigt, dass die Entwurfskomplexität mit 58 % pro Jahr wächst, während die Entwurfsproduktivität mit nur 21 % pro Jahr ansteigt [227, S. 36]. Daraus wird gefolgert, dass die Produktivität nicht mit der Komplexitätssteigerung Schritt halten kann [227, S. 36] [228, S. 8] [229, S. 8].

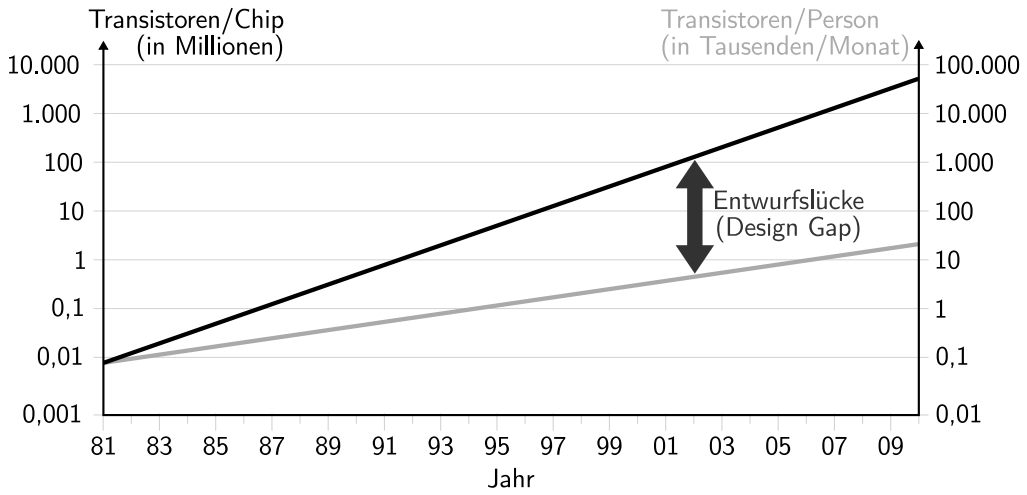


Abbildung 6.1: Entwurfslücke (nach [227, S. 36] [228, S. 8] [229, S. 8])

Basierend auf dem Ansatz des Diagramms, die Entwicklungen der Entwurfskomplexität und der Entwurfsproduktivität darzustellen, wird im Folgenden untersucht, wie sich die Komplexitäten regelmäßig und unregelmäßig aufgebauter Strukturen unterscheiden. Der Strukturunterschied lässt sich anhand der beiden folgenden Abbildungen 6.2 und 6.3 verdeutlichen. Das linke Bild zeigt die Mikrostruktur eines Flashspeichers der Unternehmen Intel und Micron in NAND-Technologie und das rechte Bild die Mikrostruktur des Intel Pentium 4 Prozessors.

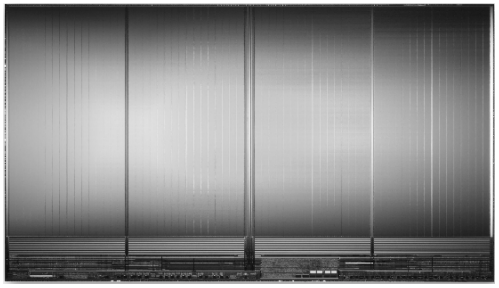


Abbildung 6.2: NAND-Speicher [98]

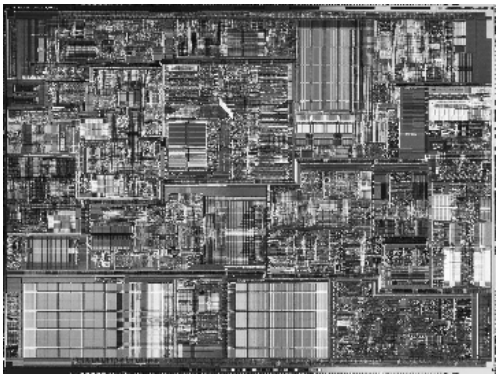


Abbildung 6.3: Intel Pentium 4 [149]

Beim 8 GB NAND-Speicher füllen die Speicherzellen fast die komplette Chipfläche aus und ergeben eine relativ regelmäßige Struktur. In der Abbildung sind die vier Speicherbänke mit jeweils 2 GB zu erkennen [151, S. 18]. Diese bestehen aus je 2048 Blöcken, welche wiederum aus 64 Seiten aufgebaut sind [151, S. 18]. Nur die Steuerungs-/Kontrolleinheit am unteren Rand des Chips unterbricht diese Regelmäßigkeit. Gemessen an der Fläche hat diese nur einen geringen Anteil. Wobei auch

hier, bei den Adressdecodierern (lange Zeilen unterhalb der Speicherbänke) und den Registern (Block rechts der Mitte) gewisse Regelmäßigkeiten zu finden sind. Durch diesen Aufbau stellt der Speicher eine sehr regelmäßige Struktur dar [[141]] [[143]] [[145]]. Der hierarchische Aufbau des Speichers mit Bänken, Blöcken und Seiten sowie weiteren kleineren Einheiten bis hin zu Datenwörtern, welche auf der Abbildung nicht mehr zu erkennen sind, erlaubt mit der Entwicklung einer Speicherzelle einen Speicher in (theoretisch fast) beliebiger Größe zu entwerfen [[141]] [[143]] [[145]]. Gemessen an der Anzahl umgesetzter Speicherzellen muss nur ein kleiner Teil der Bauelemente explizit entwickelt werden, da der überwiegende Anteil „Kopien“ darstellt [[141]] [[143]] [[145]].

Im Gegensatz hierzu kann bei einem Prozessor nicht nur eine elementare „Prozessorzelle“ entwickelt werden, die beliebig oft kopiert werden kann [[143]] [[145]]. Dies drückt sich in einer unregelmäßigen Struktur aus, die anhand des Intel Pentium 4 Prozessors in Abbildung 6.3 sichtbar ist. Die einzigen (größeren/sichtbaren) Strukturen, welche gewisse Regelmäßigkeiten aufweisen, sind die beiden Level-2-Datencaches mit deren gemeinsamen Puffer im unteren Teil der Abbildung, der Level-1-Datencache im Bereich der Mitte und der Tracecache im rechten, oberen Teil des Chips [223, S. 56], welche alle wiederum Varianten von Speichern darstellen. Der Tracecache ist eine Erweiterung des Level-1-Instruktionscaches, der statt undecodierter Maschinenbefehle die übersetzten Mikrooperationen in der zeitlichen Reihenfolge speichert, wie diese im Programm auftreten [217, S. 17]. Daneben existieren noch kleinere, für sich genommen regelmäßige Strukturen, wie beispielsweise Teile der arithmetisch-logischen Einheit (Arithmetic Logic Unit – ALU) oberhalb der Mitte, Teile der Sprungvorhersage oder Teile des Ablaufplaners. Doch auch diese regelmäßigen Strukturen besitzen keine gemeinsame „Elementarzelle“, welche kopiert werden kann. Erst recht besitzt der gesamte Prozessor keine gemeinsame „Elementarzelle“. Vielmehr besteht der Entwurf eines Prozessors darin, viele heterogene Elemente zu entwickeln und diese zu einer Einheit zusammenzufügen [[145]].

Aufbauend auf diesem Unterschied wird im Folgenden untersucht, wie sich die Regelmäßigkeit von Strukturen auf den Entwurf, mithin auf die Entwurfsentropie auswirkt. Der Ansatz ist, dass das Mooresche Gesetz das technologisch Machbare ausdrückt, was durch einen Speicher mit einer regelmäßigen Struktur abgebildet werden kann [[141]] [[143]] [[145]]. Im Gegensatz hierzu bestimmt sich die Produktivität danach, dass unregelmäßige Strukturen entworfen werden, was durch den Entwurf eines Prozessors abgebildet werden kann, welcher großteils aus individuellen Komponenten besteht [[143]] [[145]].

6.1.1 Prozessor

Für die Berechnung der Komplexität eines Prozessors und dessen Komponenten werden drei Mikroprozessorsysteme (Micro Processor Unit – MPU) untersucht, welche auf einer RISC-Architektur (Reduced Instruction Set Computer – RISC) beruhen: ein 12-Bit-Mikroprozessor, ein 16-Bit-Mikroprozessor und ein 16-Bit-Mikroprozessor mit Pipelining (Prozessorentwürfe basierend auf [239, S. 107 ff., 203 ff., 293 ff.]). Beim 12-Bit-Prozessor (MPU_{12}) ist der Operationscode fünf Bit breit, wobei von den 32 möglichen Befehlen vier Operationscodes nicht verwendet werden, sodass der Befehlssatz 28 Befehle umfasst [239, S. 55]. Der 12-Bit-Akkumulator beherrscht neben dem Löschen, Laden und Speichern des Akkumulatorwerts, die folgenden fünf Operationen: Addition, Subtraktion, Nicht-Und, links und rechts Schieben [239, S.74].

Der 16-Bit-Prozessor (MPU_{16}) ist ähnlich wie der 12-Bit-Prozessor strukturiert, sodass dieser eine Erweiterung des 12-Bit-Systems darstellt [239, S. 169]. Der Operationscode ist statt fünf sieben Bit breit und besteht aus vier Befehlsgruppen mit jeweils 32 möglichen Befehlen, wobei eine Gruppe nicht verwendet wird [239, S. 170 ff.]. Die drei verwendeten Befehlsgruppen umfassen die Registerbefehle, die Lade-/Speicherbefehle und die Adressbefehle [239, S. 170 ff.]. Dabei werden von den 128 möglichen Befehlen nur 31 verwendet [239, S. 174]. Neben den Funktionen links und rechts Schieben des Akkumulators beherrscht die ALU neun weitere Operationen: Addition, Subtraktion, Und, Oder, Antivalenz, Inkrementieren, Dekrementieren, Invertieren und Vergleichen [239, S. 186].

Der 16-Bit-Prozessor mit Pipelining (MPU_P) baut auf dem 16-Bit-System auf, welches um ein Befehlspipelining (Phasenpipelining) erweitert wurde [239, S. 241, 257]. In Abhängigkeit der jeweiligen Befehle können dadurch die vier Befehlsverarbeitungsphasen (Befehl hohlen, Befehl decodieren, Befehl ausführen und Zurückschreiben) für bis zu vier Befehlsverarbeitungen um jeweils eine Phase versetzt parallel ausgeführt werden [239, S 257 f.]. Für das Pipelining sind mehrere Änderungen am Operations- und Steuerwerk erforderlich [239, S. 257]. Das Steuerwerk ist so angepasst worden, dass das Pipelining unterstützt wird und in Abhängigkeit der Anzahl Taktzyklen eines Befehls der folgende Befehl gegebenenfalls erst dann geladen/ausgeführt wird, wenn dies dem Steuerwerk mithilfe der entsprechenden Signale mitgeteilt wurde (insbesondere, dass eine Ausgabe abgeschlossen ist, oder dass ein Mehrzyklenbefehl ausgeführt wurde) [239, S. 266 ff., 324]. Der Befehlssatz des Pipeliningprozessors entspricht den 31 Befehlen

des 16-Bit-Prozessors [239, S. 264 f.]. Innerhalb des Operationswerks werden für das Pipelining zusätzliche Multiplexer, Demultiplexer und Register benötigt, um die Teilergebnisse zwischen den einzelnen Befehlsphasen zwischenzuspeichern [239, S. 278].

Da die drei Mikroprozessorsysteme vergleichbar strukturiert sind, wird exemplarisch der Aufbau des 12-Bit-Prozessors vorgestellt (ausführlich zum Aufbau [239]). Dessen Blockdiagramm ist in Abbildung 6.4 gezeigt. Die Eingangssignale der Komponenten sind jeweils links und die Ausgangssignale jeweils rechts eingezeichnet. Die Bezeichnung einer Komponente ist jeweils innerhalb der Komponente in Fettschrift gegeben. Der Prozessor besteht aus zwei Hauptteilen: dem Steuerwerk und dem Operationswerk [239, S. 53]. Das Steuerwerk ist ein getakteter Automat, der in Abhängigkeit der Eingangssignale (insbesondere in Abhängigkeit des Operationscodes) den Ansteuerungsvektor für das Operationswerk bereitstellt [239, S. 94 ff.]. Das Operationswerk besteht neben mehreren Registern, Multiplexern und einem Kellerspeicher aus dem Programmzähler, dem Tristatepuffer für das Schreiben auf den Systembus sowie dem Akkumulator, welcher die ALU enthält [239, S. 70 ff.].

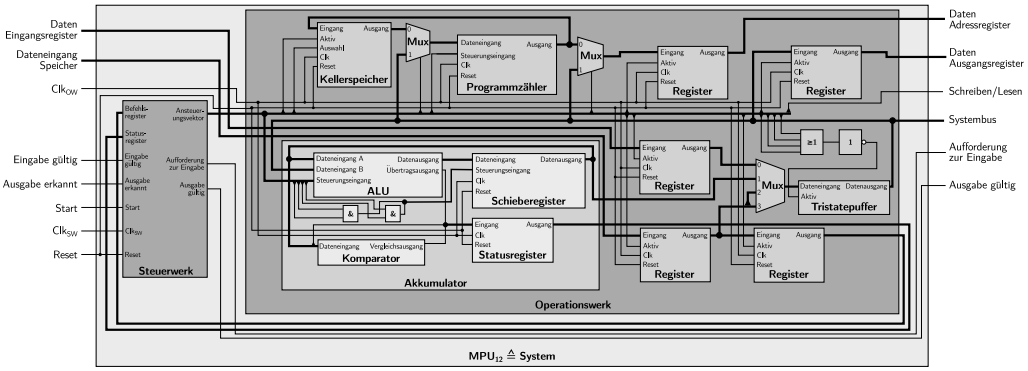


Abbildung 6.4: Blockdiagramm der MPU_{12} [basierend auf 239, S. 53 ff.]

Die folgende Abbildung 6.5 verdeutlicht die Hierarchie der einzelnen Komponenten. Das Steuerwerk enthält dabei keine weiteren aufgebauten Komponenten, sodass nur Elementarkomponenten verwendet werden, wie Bedingungen und Zuweisungen. Das Operationswerk besteht aus einer Vielzahl an aufgebauten Komponenten. Dabei enthalten der Programmzähler, die beiden Multiplexer, der Kellerspeicher, der Tristatepuffer und die beiden Gatter keine weiteren aufgebauten Unterkomponenten, sondern nur Elementarkomponenten. Die beiden Register sind wiederum aus weiteren entworfenen Unterkomponenten aufgebaut. Ebenfalls ist der Akkumulator aus weiteren aufgebauten Unterkomponenten umgesetzt.

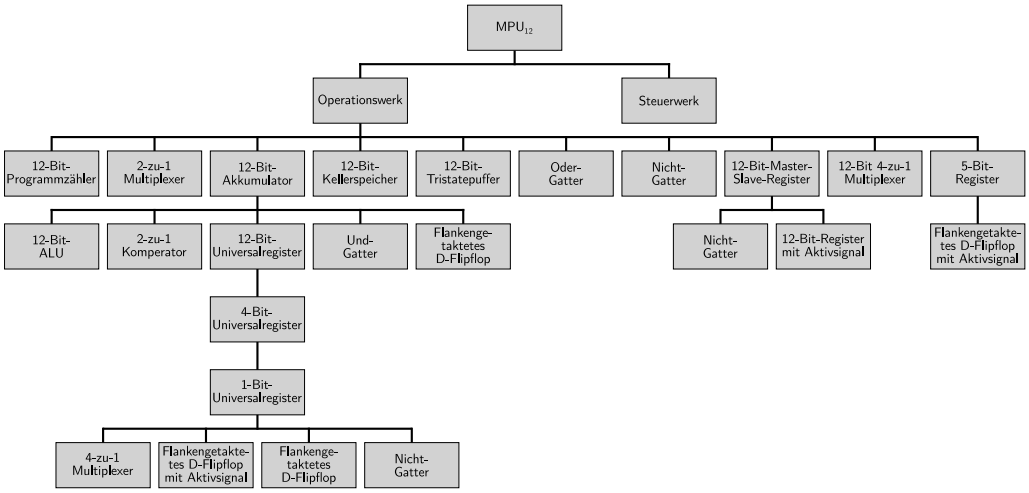


Abbildung 6.5: Komponentenhierarchie der MPU_{12}

Der Mikroprozessor (MPU_{12}) ist die Top-Level-Entität und stellt somit das System im Sinne der Entwurfsentropie dar. Bei der Berechnung der Entwurfsentropie werden ausgehend vom System die enthaltenen Komponenten analysiert. Besteht eine Komponente wiederum aus Unterkomponenten, welche keine Elementarkomponenten sind, wird die Analyse dieser Unterkomponenten durchgeführt. Wie in den vorherigen Kapiteln gezeigt, besitzen Elementarkomponenten nur eine Verhaltensentropie aber keine Strukturentropie. Dadurch, dass Komponenten mehrfach verwendet werden können, wie beispielsweise das flankengetriggerte D-Flipflop bei der MPU_{12}, muss vom Analysewerkzeug berücksichtigt werden, welche Komponenten bereits analysiert wurden, das heißt, von welchen Komponenten die Strukturentropie bereits berücksichtigt wurde. Durch die rekursive Analyse der Komponenten entsteht ein Analysebaum, wie in Abschnitt 4.9 über das Analysewerkzeug beschrieben. Für den 12-Bit-Prozessor entspricht der Analysebaum der Komponentenhierarchie aus der vorherigen Abbildung 6.5, wobei die Elementarkomponenten der Übersichtlichkeit halber nicht aufgelistet sind. Für die mehrfach verwendeten Komponenten, wie das flankengetriggerte D-Flipflop, wird die Strukturentropie nur ein Mal berücksichtigt. Dies zeigt sich bei der Ausgabe des Analysewerkzeugs, welches jeweils die Strukturentropie von den aufgebauten Komponenten ausgibt, wie die folgende Abbildung 6.6 zeigt. Dabei geben die einzelnen Werte jeweils die Strukturentropie einer Komponente ohne die Berücksichtigung der Strukturentropie der Unterkomponenten an, sodass nur die Verhaltensentropie der Unterkomponenten einer Komponente ausgegeben wird. Dadurch ist gewährleistet, dass die Strukturentropie von mehrfach verwendeten Komponenten nur ein Mal in die Berechnung der Entwurfsentropie des Systems eingeht.

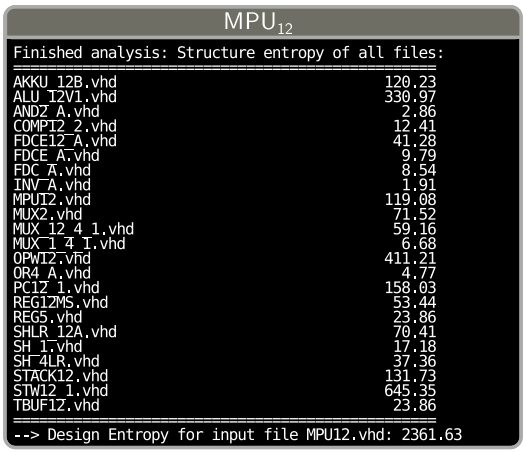


Abbildung 6.6: Ausgabe des Analysewerkzeugs für die MPU_{12}

Mit Ausnahme des Steuerwerks und der Systemkomponente werden alle anderen Komponenten innerhalb des Operationswerks verwendet, sodass unter Berücksichtigung dieser Unterkomponenten das Operationswerk eine Entwurfsentropie von $2204{,}94 - 119{,}08 - 645{,}35 = 1.440{,}51$ aufweist. Als einzelne Komponente ist dagegen das Steuerwerk am aufwendigsten, wie das folgende Kreisdiagramm (Abbildung 6.7) anschaulich zeigt. Dabei sind die zugehörigen Zahlenwerte in der nebenstehenden Tabelle 6.1 angegeben. Alle Bausteine mit einer geringeren Entwurfsentropie als die des Systems wurden zur Gruppe der sonstigen Komponenten zusammengefasst, um die Darstellung übersichtlich zu halten. Für die aufgelisteten Unterkomponenten wurde jeweils deren Entwurfsentropie von der ihnen übergeordneten Komponente abgezogen. Somit stellen die Werte, wie bei der Ausgabe des Analysewerkzeugs, jeweils die Summe der Verhaltensentropien der direkten Unterkomponenten dar, mithin also den Verdrahtungsaufwand.

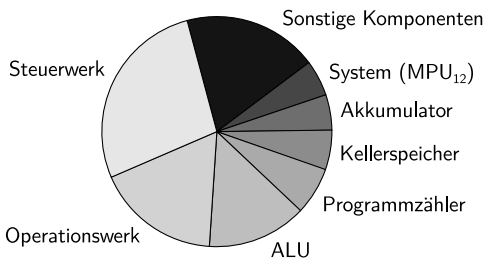


Abbildung 6.7: Entwurfsentropieverteilung der MPU_{12}

Steuerwerk	645,35
Operationswerk	411,21
ALU	330,97
Programmzähler	158,03
Kellerspeicher	131,73
Akkumulator	120,23
System (MPU12)	119,08
Sonstige Komponenten	445,03

Tabelle 6.1: Entwurfsentropien der MPU_{12}-Komponenten

Wird die Entwurfsentropie für alle drei Prozessoren mithilfe des Analysewerkzeugs berechnet, ergeben sich die in der folgenden Tabelle 6.2 eingetragen Entwurfsentropiewerte. Der 12-Bit-Prozessor weist mit einem Wert von $H_S(\text{MPU}_{12}) = 2361{,}63$ die geringste Entwurfsentropie auf. Höher fallen die Werte für den 16-Bit-Prozessor mit $H_S(\text{MPU}_{16}) = 5062{,}39$ und $H_S(\text{MPU}_\text{P}) = 5688{,}44$ für den 16-Bit-Prozessor mit Pipelining aus.

	MPU_{12}	MPU_{16}	MPU_P
System	119,08	192,80	281,50
Steuerwerk	645,35	1093,17	1021,13
Operationswerk	1597,20	3776,42	4385,81
Summe	2361,63	5062,39	5688,44

Tabelle 6.2: Ergebnisse der MPU-Auswertung

Das System ist die Top-Level-Entität, welches die Instanzen der beiden Hauptkomponenten (Steuerwerk und Operationswerk) enthält und diese miteinander sowie mit der Umwelt verbindet. Beim Vergleich des 12-Bit-Prozessors mit dem 16-Bit-Prozessor haben alle Komponenten eine höhere Entwurfsentropie. Das System der MPU_{16} hat eine etwa 1,6-fache Entwurfsentropie im Vergleich zu der des Systems der MPU_{12}. Zum einen ist dies auf die höhere Datenbreite zurückzuführen und zum anderen darauf, dass das Operationswerk ein zusätzliches Ein- und Ausgangsregister enthält, dessen Verbindungen nach außen geleitet werden. Die Entwurfsentropie des Operationswerks ist etwa um den Faktor 2,4 höher als beim 12-Bit-Prozessor. Dies ergibt sich aus der Kombination der breiteren Datenleitungen in Verbindung mit dem zusätzlichen Register und den Steuerleitungen. Zudem ist die ALU des 12-Bit-Prozessors als Verhaltensbeschreibung ohne Unterkomponenten entworfen worden und beim 16-Bit-Prozessor eine Strukturbeschreibung, welche 15 Unterkomponenten aufweist. Dabei sind auch die elementaren Operationen (Und, Oder und so weiter) sowie die Addierer (Voll- und Halbaddierer) als Strukturbeschreibung gegeben. Das Steuerwerk des 16-Bit-Prozessors hat etwa eine 1,7-fache Entwurfsentropie des 12-Bit-Prozessors. Bei der MPU_{16} umfasst die Automatentabelle 60 Zustände und es wurden 31 Befehle umgesetzt. Dagegen hat die Automatentabelle der MPU_{12} nur 40 Zustände und es werden 28 Befehle unterstützt. Hieraus ergibt sich ein Verhältnis von $\frac{60}{40} \cdot \frac{31}{28} \approx 1{,}7$. Dieser zusätzliche Aufwand wird direkt durch die Entwurfsentropie ausgedrückt. Dies zeigt auch das folgende Diagramm (Abbildung 6.8), welches die Entwurfsentropien der Hauptkomponenten als gestapelte Balken zeigt.

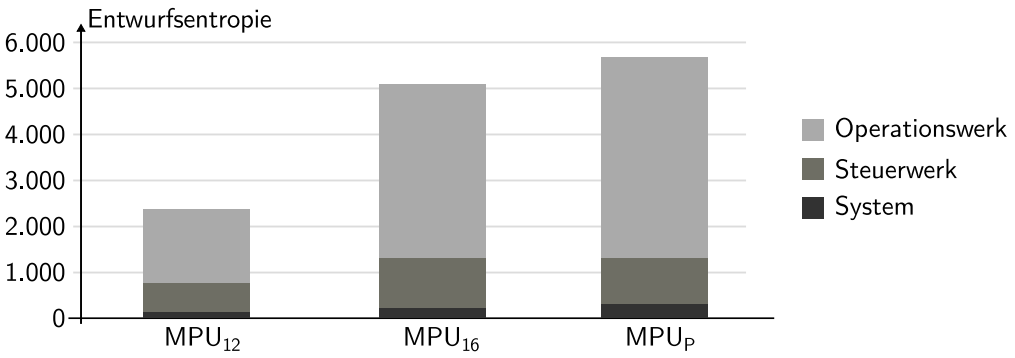


Abbildung 6.8: Entwurfsentropien der Hauptkomponenten

Beim 16-Bit-Pipeliningsystem ist die Entwurfsentropie des Systems (Top-Level-Entität) höher als die des 16-Bit-Systems ohne Pipelining. Durch das Pipelining hat das Operationswerk mehr Ein- und Ausgänge, die verdrahtet werden müssen, da statt eines Systembusses wie bei den ersten beiden Prozessoren ein (externer) Arbeitsspeicher mit getrennten Befehls- und Datenbereichen verwendet wird, welche jeweils eine separate Adressierung benötigen [239, S. 278]. Gemessen am Gesamtsystem ist jedoch die Entwurfsentropiesteigerung der Systemkomponente vernachlässigbar, wie das vorherige Balkendiagramm zeigt.

Das Steuerwerk der $\mathrm{MPU_P}$ hat eine geringfügig niedrigere Entwurfsentropie als das Steuerwerk der $\mathrm{MPU_{16}}$, obwohl es zusätzlich die Steuerung des Pipelinings übernimmt. Der geringere Wert ist auf die Vereinfachung des Zustandsautomaten zurückzuführen: Statt sieben Zuständen wie bei der $\mathrm{MPU_{16}}$ kommt der Zustandsautomat der $\mathrm{MPU_P}$ durch das Pipelining mit vier Zuständen aus [239, S. 291]: Der erste Zustand ist der Startzustand; im zweiten Zustand laufen alle Akkumulatorbefehle ab; die restlichen Befehle, welche drei Taktzyklen benötigen, laufen im dritten Zustand ab; und der vierte Zustand umfasst die Ein- und Ausgabebefehle, durch welche sich die Ausführung verzögern kann, bis eine Bestätigung vorliegt [239, S. 291].

Die umfassendsten (strukturellen) Änderungen bei der $\mathrm{MPU_P}$ im Vergleich zur $\mathrm{MPU_{16}}$ waren am Operationswerk erforderlich. Für das Pipelining sind zusätzliche Multiplexer, Demultiplexer und Register erforderlich. Da die meisten dieser Komponenten bereits aufgebaute Basis- und Speicherkomponenten sind, müssen diese nur verdrahtet werden. Somit fällt bei der Verwendung nur deren Verhaltensentropie an, da ihre Strukturentropie bereits innerhalb des Systems berücksichtigt wurde. Aus diesem Grund steigt die Entwurfsentropie für das Operationswerk nur auf etwa das 1,2-Fache des ursprünglichen Werts der $\mathrm{MPU_{16}}$ an.

Bis jetzt wurden die Prozessoren mit allen (aufgebauten) Unterkomponenten analysiert. Jedoch können einige aufgebaute Komponenten bei mehreren Prozessoren verwendet werden. Um den Einfluss der Wiederverwendung zu untersuchen, sind diejenigen Komponenten, welche unverändert vom 12-Bit-Prozessor bei den beiden 16-Bit-Prozessoren verwendet wurden, im Implementierungscode als wiederverwendet markiert. Somit geht deren Strukturentropie nicht in die Berechnung der Entwurfsentropie bei den 16-Bit-Prozessoren ein. Zusätzlich sind beim Pipeliningprozessor alle vom 16-Bit-Prozessor übernommenen Komponenten durch einen Kommentar im Implementierungscode als wiederverwendet markiert. Die folgende Tabelle 6.3 führt die berechneten Werte und deren Summen auf.

	MPU_{12}	MPU_{16}	MPU_P
System	119,08	192,80	281,50
Steuerwerk	645,35	1093,17	1021,13
Operationswerk	1597,20	3702,69	2134,77
Summe mit Wiederverwendung	2361,63	4988,66	3437,40
Einsparung	0,00	73,73	2251,04
Summe ohne Wiederverwendung	2361,63	5062,39	5688,44

Tabelle 6.3: Entwurfsentropie der Prozessoren bei Wiederverwendung

Da das System (Top-Level-Entität) jeweils spezifisch für den Prozessor ist, kommt es zu keiner Reduktion durch die Wiederverwendung. Ebenfalls stimmen die Werte für das Steuerwerk mit den Werten aus Tabelle 6.2 überein, bei welchen die Wiederverwendung nicht berücksichtigt ist. Da das Steuerwerk keine aufgebauten Unterkomponenten enthält, können auch keine Komponenten wiederverwendet werden. Anders verhält es sich beim Operationswerk. Für die MPU_{16} führt die Berücksichtigung der Wiederverwendung zu einer geringfügigen Entwurfsentropiereduktion, da die Basisbausteine, wie das Nicht-Gatter, der zwei Bit breite Multiplexer oder das D-Flipflop wiederverwendet werden können. Durch die Änderung der Daten- und Instruktionsbreite müssen jedoch aus den Basiskomponenten wieder neue Komponenten, wie ein 16-Bit-Register statt eines 12-Bit-Registers, aufgebaut werden. Somit ist die Einsparung durch die Wiederverwendung relativ gering, wie auch die folgende Abbildung 6.9 verdeutlicht. Beim Balkendiagramm ist die Einsparung durch die Wiederverwendung als schraffierte Fläche verdeutlicht.

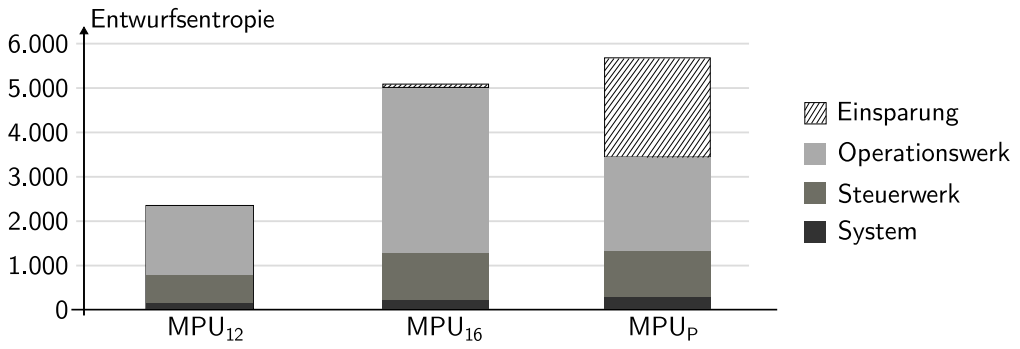


Abbildung 6.9: Entwurfsentropieeinsparung durch Wiederverwendung

Beim 16-Bit-Pipeliningsystem können wesentlich mehr Komponenten vom 16-Bit-System wiederverwendet werden. Somit ist die Einsparung durch die Wiederverwendung wesentlich größer als beim 16-Bit-System. Für den Aufbau des Pipeliningoperationswerks sind insbesondere zusätzliche Register und Multiplexer erforderlich, die bereits bei der MPU_{16} aufgebaut wurden, sodass das Pipeliningsystem nur weitere Instanzen dieser Komponenten verwendet. Hierdurch muss nur die zusätzliche Verhaltensentropie berücksichtigt werden. Somit umfasst der Aufwand für den Entwurf des Operationswerks nur den Verdrahtungsaufwand der bereits aufgebauten Komponenten. Somit zeigt die Analyse der Wiederverwendung, dass die Änderung der Datenbreite dazu führt, dass viele Komponenten neu aufgebaut werden müssen, während durch den Aufbau einer Pipeliningarchitektur viele Komponenten wiederverwendet werden können. Mithilfe der Entwurfsentropie können die Auswirkungen auf den Entwurf direkt dargestellt werden. In diesem Zusammenhang untersucht der folgende Unterabschnitt anhand von Speichern, wie sich die Wiederverwendung auf die Entwurfsentropie von hierarchischen und gleichzeitig relativ regelmäßigen Strukturen auswirkt.

6.1.2 Speicher

Als Grundlage für den Aufbau eines Speichers dient das Datenblatt des NAND-Speichers der Firma Micron [150]. Jedoch wurden nicht alle Funktionalitäten umgesetzt, sodass der Speicher ausschließlich die beiden Grundfunktionen Lesen und Schreiben beherrscht. Zudem hat ein Speicherblock nur 512 Bit, da das im folgenden Unterabschnitt verwendete Synthesewerkzeug die Analyse von umfangreicheren, vollständig aus Gattern aufgebauten Speichern abbrach. Kern des Speichers bilden Speicherzellen aus einem getakteten D-Latch mit zusätzlichem Tristatetreiber, deren Aufbau die folgende Abbildung 6.10 zeigt.

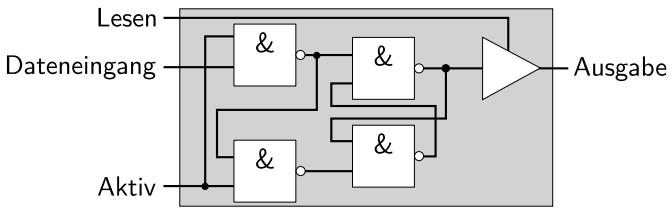


Abbildung 6.10: Speicherzelle aus Nicht-Und-Gattern und Tristatetreiber

Das getaktete D-Latch ist aus vier Nicht-Und-Gattern aufgebaut. Um die Ausgänge der einzelnen Speicherzellen direkt miteinander verbinden zu können, wird ein Tristatetreiber eingesetzt, welcher in Abhängigkeit des Lesesignals entweder den gespeicherten Zustand an den Ausgang weiterleitet oder den Ausgang auf hochohmig (Z) schaltet. Somit sind die Ausgangssignale der Speicherzellen als Standardlogik (std_logic) umgesetzt. Weiterhin besitzt die Speicherzelle als Eingang ein Aktivsignal. Ist es eins, dann wird der Wert am Dateneingang im D-Latch gespeichert, anderenfalls wird der innere Wert beibehalten.

Analog zum Speicher von Micron [150, S. 12] ist aus den einzelnen Speicherzellen die Speicherbank hierarchisch aufgebaut. Die nachfolgende Abbildung zeigt den verwendeten Aufbau als sogenannte Arrayorganisation [angelehnt an 150, S. 12 ff.]. Ein Datenwort (word) besteht aus acht Speicherzellen (bit). Vier Wörter bilden eine Seite (page). Aus vier Seiten ist ein Block (block) aufgebaut und vier Blöcke bilden eine Speicherbank (plane).

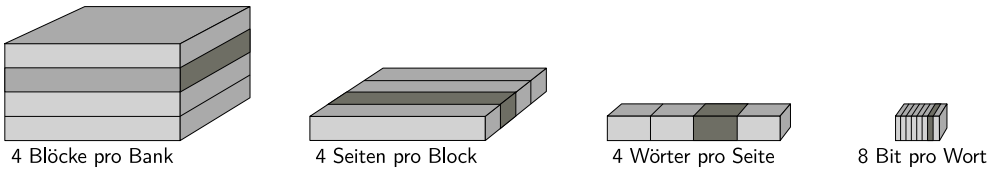


Abbildung 6.11: Arrayorganisation des Speichers [angelehnt an 150, S. 12 ff.]

Die Speicherbänke bilden den Kern des Speicherchips, wie das folgende Blockdiagramm (Abbildung 6.12) für zwei Bänke zeigt. Für die Ansteuerung der Bänke sind eine Steuerungskomponente sowie weitere Komponenten erforderlich. Wie bei der Vorlage verfügt der Speicherchip über eine bidirektionale Datenschnittstelle (IO-Daten), welche im vorliegenden Fall acht Bit breit ist. Sie dient neben dem Lesen und Schreiben der Daten aus und in die Speicherbänke dazu, die Adressen zu übertragen.

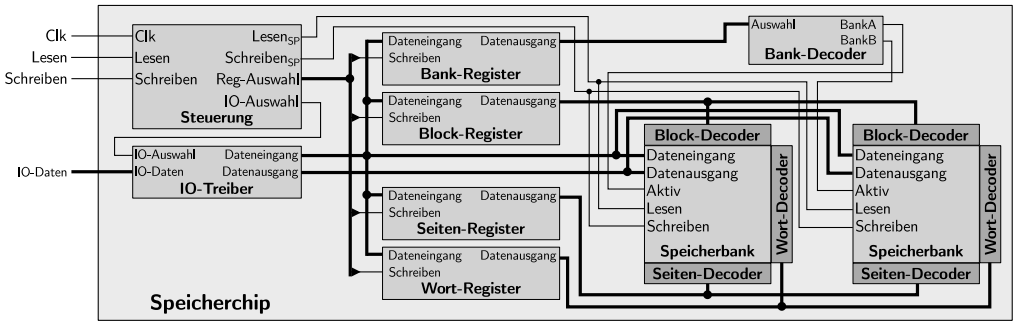


Abbildung 6.12: Blockdiagramm des Speichers [angelehnt an 150, S. 6]

Für die Übertragung der Adressen verwendet Micron mehrere (Takt-)Zyklen, sodass im ersten und zweiten Takt die Spaltenadresse und im dritten bis fünften Takt die Reihenadresse übertragen wird [150, S. 12]. Dieser Aufbau der Adressübertragung ist grundsätzlich übernommen, nur dass im ersten Takt die Bankadresse, im zweiten Takt die Blockadresse, im dritten Takt die Seitenadresse und im vierten Takt die Wortadresse übertragen werden. Somit ist die primäre Aufgabe der Steuerung, die Adressdaten zunächst in den entsprechenden Registern zu speichern, nachdem ein Lese- oder Schreibsignal erkannt wurde. Hierzu ist die Steuerung als getakteter Automat umgesetzt. Mit jeder Taktflanke werden die Bank-, Block-, Seiten- und Wortadressen in dieser Reihenfolge übertragen. Durch das Registerauswahlsignal (Reg-Auswahl) werden die Eingangsdaten im jeweiligen Register gespeichert. Sollen Daten gelesen werden, legt die Steuerung im folgenden Takt das Lesesignal für die Speicherbänke (Lesen_{SP}) auf eins und schaltet den IO-Treiber mithilfe des Auswahlsignals (IO-Auswahl) so, dass die Daten an der bidirektionalen Schnittstelle (IO-Daten) ausgegeben werden. Die Auswahl des jeweiligen Datenworts erfolgt mithilfe des jeweiligen Decoders für die vier Adressen. Sollen Daten in den Speicher geschrieben werden, leitet der IO-Treiber die Daten an die Dateneingänge der Speicherbänke weiter. Mithilfe des Schreibsignals ($\text{Schreiben}_{\text{SP}}$), welches mit den Aktivsignalen der jeweils ausgewählten Speicherzellen (Aktiv) verbunden ist, werden die Daten in die Speicherzellen übernommen.

Wird der Steuerung noch das Dateneingangssignal übergeben, können auch die Steuerungsbefehle des Micron-Chips umgesetzt werden (zum Beispiel seitenweises Lesen, internes Verschieben, Löschen und so weiter) [150, S. 22]. Dabei weist die Steuerung Ähnlichkeiten zu einem Steuerwerk auf, wie es in einem Prozessor zu finden ist. Da in diesem Abschnitt der Unterschied zwischen regelmäßigen und unregelmäßigen Strukturen im Vordergrund steht, wurde die Steuerung und damit der Speicherchip nur mit den Grundfunktionalitäten ausgestattet, da das

Steuerwerk eine unregelmäßige Struktur darstellt, wie im vorherigen Unterabschnitt gezeigt wurde. Somit erfolgt die externe Ansteuerung des Chips nur mithilfe des Lese- und Schreibsignals zusammen mit dem Taktsignal.

Durch die hierarchische Organisation des Speichers kann dieser sehr einfach in unterschiedlichen Größen erstellt werden. Auch bei den Speicherchips von Micron erfolgt die Änderung der Datenbreite dadurch, dass die Datenwörter vergrößert werden [150, S. 12 ff.]. Die Änderung der Speichergröße erfolgt durch das Zusammenfügen mehrerer Speicherblöcke [151, S. 16 ff.]. Wird dieser Ansatz auf den vorliegenden Entwurf angewendet, dann ergeben sich die in der folgenden Tabelle 6.4 eingetragenen Entropiewerte beim Aufbau mit einer bis sechzehn Bänken.

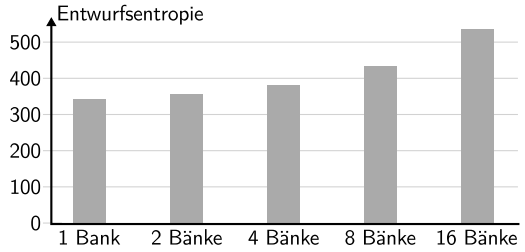


Abbildung 6.13: Speichervergleich

# Bänke	Entwurfsentropie
1	341,73
2	354,49
4	379,99
8	430,99
16	533,01

Tabelle 6.4: Speichervergleich

Obwohl sich die Speichergröße jeweils verdoppelt, ist der Entwurfsentropiezuwachs nur minimal, wie das Diagramm anschaulich zeigt (Abbildung 6.13). Werden mehr Speicherbänke hinzugefügt, fällt lediglich der Aufwand für deren Verdrahtung sowie gegebenenfalls die Anpassung der Steuerung und des Bankregisters an. So würde beim Grundspeicher mit einer Bank kein Taktzyklus für die Bankauswahl benötigt und somit kein Register zum Speichern der Bankadresse. Bei zwei Speicherblöcken könnte die Registeradresse in einem einzelnen D-Latch gespeichert werden. Erst bei vier Speicherblöcken werden ein zusätzliches Register und ein Bankdecoder benötigt. Da das Register und der Decoder von den anderen drei Adressen wiederverwendet werden können, fällt hierbei nur der Aufwand für die Verdrahtung an.

Wird der Aspekt der Wiederverwendung in die Auswertung einbezogen, dann fällt der Hauptaufwand für die Entwicklung der Basiskomponenten an. Da auch für mehr als einen Block keine neuen Komponenten entwickelt werden müssen, sondern nur die vorhandenen Komponenten verdrahtet werden, wird allein die Verhaltensentropie dieser zusätzlichen Komponenten berücksichtigt. Hierzu sind alle Komponenten, welche bereits beim Grundspeicher entwickelt wurden und

unverändert in den größeren Speichern eingesetzt werden, als „wiederverwendet“ im Quellcode markiert, sodass deren Strukturentropie nicht in die Berechnung eingeht. Zur Übersichtlichkeit wird angenommen, dass die Steuerung beim Grundspeicher bereits über den Zustand zum Einlesen der Bankadresse verfügt, obwohl diese nicht benötigt wird. Dies entspricht auch dem Entwurf von Micron, bei welchem das Einlesen der Adressdaten für alle Varianten eines Chips gleich bleibt [150, S. 12 ff.] [151, S. 16 ff.]. Um die Werte vergleichen zu können, sind auch die Komponenten bei der Grundvariante als wiederverwendet markiert. Somit berechnet die Entwurfsentropie jeweils den Aufwand für die Verdrahtung der bereits entwickelten Komponenten. Das Ergebnis der Berechnung ist in der folgenden Tabelle 6.5 dargestellt.

# Bänke	1	2	4	8	16
Entwurfsentropie	53,00	65,76	91,26	142,26	244,28
Differenz	-	12,76	25,50	51,00	102,02
Pro zusätzlicher Bank	-	12,76	12,75	12,75	12,75

Tabelle 6.5: Entwurfsentropie der Verdrahtung der Speicherkomponenten

Wird die Differenz betrachtet und durch die Anzahl zusätzlicher Bänke dividiert, zeigt sich, dass der Entwurfsentropiezuwachs pro Bank konstant ist. Die gleichen Differenzen zwischen den Werten ergeben sich auch, wenn die Differenzen von den Entwurfsentropien aus der vorletzten Tabelle 6.4 gebildet werden. Dies zeigt, dass die Speicher sich ausschließlich in der Verdrahtung geändert haben und alle Komponenten für jede Speichergröße gleich bleiben.

Durch die Regelmäßigkeit des Speichers und der hohen Wiederverwendungsmöglichkeit der Komponenten ist dieser gut skalierbar. Dies zeigt sich auch in vielen Hardwarebeschreibungen, bei welchen die Speichergrößen durch eine generische Beschreibung einfach veränderbar sind. Im Vergleich mit dem Prozessor ist der Anteil an Komponenten, welche für die jeweilige Speichergröße geändert werden müssen, relativ gering. Selbst bei der Erweiterung des 16-Bit-Prozessors um ein Pipelining waren sowohl umfangreiche Änderungen am Operationswerk als auch am Steuerwerk erforderlich. Diese Informationen werden im folgenden Unterabschnitt dafür verwendet, die Entwurfslücke aus einem anderen Blickwinkel zu betrachten.

6.1.3 Entwurfslücke

Die Motivation dieser Fallstudie ergab sich aus der Entwurfslücke, welche einerseits die Entwicklung der Transistoranzahl von Chips und andererseits die umgesetzten Transistoren pro Entwicklerin und Entwickler als Geraden zeigt. Der Abstand zwischen den Geraden wird als Entwurfslücke bezeichnet, da das technologisch Machbare schneller ansteigt als die Produktivität. Dabei wurde die Frage gestellt, ob sich die Entwurfslücke tatsächlich aus der zu geringen Produktivität ergibt oder ob vielmehr die Regelmäßigkeit der Strukturen diese bereits bedingt. Aus diesem Grund wurden Prozessoren mit relativ unregelmäßigen Strukturen und Speicher mit relativ regelmäßigen Strukturen untersucht.

Um die Entwurfsentropie der Speicher und Prozessoren miteinander vergleichen zu können, wurde für die Entwürfe die Anzahl an umgesetzten Logikelementen bestimmt. Somit kann der Entwurfsaufwand in das Verhältnis zur Logikelementanzahl gebracht werden. Die Anzahl an Logikelementen wurde hierfür mithilfe des Synthesewerkzeugs Quartus Prime von Altera (Version 16.0.0) für den FPGA Cyclone IV E bestimmt [6]. Der Speicher musste dabei von ursprünglich 8 GB sukzessive verkleinert werden, bis dieser synthetisierbar war.

Das folgende Diagramm (Abbildung 6.14) stellt das Ergebnis der Auswertung dar. Die x-Achse zeigt die Anzahl an Logikelementen und die y-Achse die Entwurfsentropie. Für die Darstellbarkeit sind für den Speicher nur die ersten beiden Datenpunkte mit einem beziehungsweise zwei Blöcken eingetragen, da der Speicher mit vier Blöcken bereits 4.258 Logikelemente verwendet und der Speicher mit 16 Blöcken sogar 15.850 Logikelemente benötigt. Für den Prozessor sind die Werte für alle drei Entwürfe im Diagramm eingetragen. Zusätzlich ist für beide Datenreihen eine Regressionsgerade eingezeichnet.

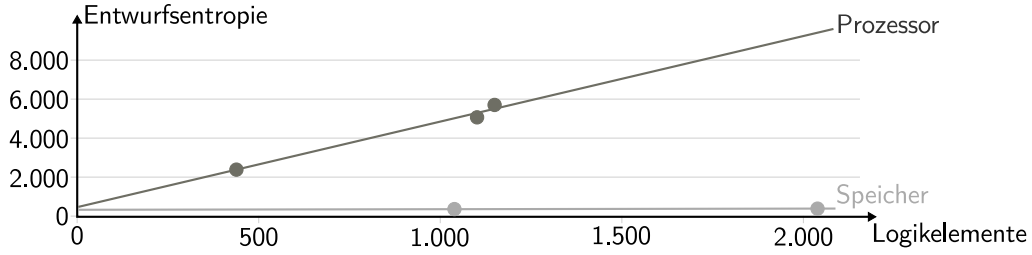


Abbildung 6.14: Entwurfsentropie im Verhältnis zur Logikelementanzahl

Wie das Diagramm anschaulich zeigt, steigt die Logikelementanzahl beim Speicher schnell an, obwohl der zusätzliche Aufwand für die Entwicklung eines größeren Speichers nur minimal ist. Anders verhält es sich beim Prozessor: Die Logikelementanzahl wächst im Vergleich zum Speicher wesentlich langsamer. Für einen umfangreicheren Prozessor mit mehr Logikelementen ist somit ein wesentlich höherer Aufwand erforderlich. Bei den beiden Datenpunkten der 16-Bit-Prozessoren zeigt sich auch, dass durch das Pipelining nur unwesentlich mehr Logikelemente benötigt werden, da der Hauptunterschied in den zusätzlichen Registern und Multiplexern liegt. Jedoch ist der höhere Aufwand durch nur 48 zusätzliche Logikelemente nicht zufriedenstellend abgebildet. Somit zeigten die Ergebnisse dieses Abschnitts im Hinblick auf die Entwurfslücke, dass die Transistoranzahl beziehungsweise Logikgatteranzahl als Maßzahlen relativ ungeeignet sind, da bei regelmäßigen Strukturen eine hohe Transistoranzahl mit verhältnismäßig geringem Aufwand erzeugt werden kann. Im Gegensatz dazu ist bei unregelmäßigen Strukturen vielfach ein hoher Aufwand für den Entwurf erforderlich, welcher sich nicht zwangsläufig auch in einer hohen Anzahl an Logikgattern niederschlägt.

An dieser Stelle soll und muss darauf hingewiesen werden, dass die zugrunde gelegte Analogie zwischen der Regelmäßigkeit und der Entwurfslücke nur sehr eingeschränkt eine Aussage über die tatsächliche Entwicklung trifft. Vielmehr steht ein kritischer Blick auf die Entwurfslücke im Vordergrund. So zeigt diese Fallstudie, dass es sich bereits aus der Natur der Sache ergibt, dass der Entwicklungsaufwand von unregelmäßigen Strukturen höher ausfallen muss als bei der Entwicklung von regelmäßigen Strukturen. Das heißt, die Entwurfslücke zeigt eigentlich nicht, dass Entwurfswerkzeuge, Abstraktionsebenen und neue Ansätze fehlen. Diese Faktoren führen nur dazu, den Abstand zwischen den Geraden zu verringern. Die Lücke ergibt sich bereits dadurch, dass immer mehr Transistoren auf einem Chip untergebracht werden können (Mooresches Gesetz). Hierfür sind natürlicherweise bereits mehr Personen erforderlich, um diese Transistoren auch zu realisieren. Das heißt, ein anderer Blickwinkel wäre, die Entwurfslücke als Motivation zu verwenden, dass Chipherstellerinnen oder Chiphersteller immer mehr Mitarbeiterinnen und Mitarbeiter benötigen, um das technologisch Machbare umzusetzen. Aus dem Blickwinkel des Hardwareentwurfs bedeutet dies, dass mehr Transistoren auch dadurch umgesetzt werden können, dass unregelmäßige Strukturen zugunsten von regelmäßigen Strukturen ersetzt werden beziehungsweise das Verhältnis zugunsten der regelmäßigen Strukturen verbessert wird. Dies zeigt auch die tatsächliche Entwicklung: Neben der höheren Taktung wurden zunächst immer mehr Speicherzellen (Register) in den Prozessoren eingesetzt. Zudem weist der RISC-Befehlssatz eine wesentlich höhere Regelmäßigkeit auf, sodass CISC-Befehle (Complex Instruction

Set Computer – CISC) zunehmend prozessorintern in RISC-Befehle übersetzt wurden (CISC-Emulationsschicht). Der nächste Schritt war die Entwicklung von Mehrkernprozessoren, bei welchen wiederum die Wiederverwendung eine hohe Rolle spielt. Ähnlich dem Speicher war es hierdurch möglich, mit der Entwicklung einer Prozessorzelle (Kern) den Prozessor zu skalieren, sodass der weitere Aufwand bei der Verbindung der Zellen lag (und teilweise auf die Software übertragen wurde, da diese nun die Aufgabe übernehmen musste, die Kerne auch zu nutzen). Hinzu kamen weitere Konzepte auf Befehlsebene, wie das Pipelining oder die superskalaren Architekturen. Dabei konnte im Vorherigen gezeigt werden, dass das Pipelining eine wesentlich höhere Wiederverwendung zulässt, als wenn die Datenbreite erhöht wird.

6.2 Bussysteme

Beim Entwurf von Digitalschaltungen müssen regelmäßig mehrere Hardwarekomponenten miteinander verdrahtet werden, sodass diese miteinander Daten austauschen können. Dazu kann jede Hardwarekomponente mit allen anderen Komponenten verbunden werden. Mit der Anzahl an Komponenten steigt dabei zum einen der Verdrahtungsaufwand und zum anderen der Platzbedarf für die Leitungen. Daher werden Bussysteme als universelle Verbindungssysteme eingesetzt, welche über Sammelleitungen verfügen, sodass diese von mehreren Systemkomponenten verwendet werden können [214, S. 102]. Die beiden grundlegenden Topologien hierzu sind einerseits ein Multiplexbus (Logikbus) und andererseits ein Tristatebus [117, S. 326 ff.]. Der Logikbus basiert auf dem Prinzip eines Multiplexers, sodass in Abhängigkeit eines Auswahlsignals eine der Systemkomponenten auf den Bus geschaltet wird. Beim Tristatebus sind alle Komponenten mit den gemeinsamen Busleitungen verbunden, sodass sich grundsätzlich der Hardwareaufwand verringert, jedoch zunächst ein Tristatetreiber aufgebaut werden muss [117, S. 332].

In diesem Zusammenhang wird mithilfe der Entwurfsentropie untersucht, ab welcher Systemkomponentenanzahl der Aufbau eines Bussystems mit Tristatetreibern effizienter ist als ein Multiplexverfahren. Als Fallstudie wird ein unidirektionaler Bus analysiert, welcher über einen Busmaster und mehrere Systemkomponenten verfügt. Er kann beispielsweise eingesetzt werden, um Daten von externen Baugruppen zu lesen oder innerhalb eines Prozessors, um Daten aus den Registern zu lesen. Abbildung 6.15 zeigt die Schnittstellen des Busmasters und der Systemkomponenten.

Abbildung 6.15: Schnittstellen des Busmasters und der Systemkomponente

Der Busmaster (zum Beispiel ein Prozessor) hat als Ausgänge neben einem Taktsignal (Clk), Resetsignal (Reset) und Lesesignal (Lesen) auch die Chipauswahl (Chip-Sel), um die jeweilige Systemkomponente auszuwählen. Diese kann alternativ auch als Adressbus umgesetzt werden, um beispielsweise einen oder mehrere Speicher zu adressieren. Der Dateneingang (Daten) ist acht Bit breit und unidirektional, sodass nur Daten gelesen werden können. Die Systemkomponenten haben als Eingänge jeweils das Taktsignal (Clk), das Resetsignal (Reset) und das Lesesignal (Lesen), welches gleichzeitig als Aktivierungssignal dient. Ist das Lesesignal eins, werden am Datenausgang (Daten) die Daten für den Busmaster angelegt.

Die erste Möglichkeit der Umsetzung basiert auf einem Multiplexverfahren, sodass der Busmaster mit dem Multiplexer verbunden ist. Von diesem aus gehen die einzelnen Leitungen zu den jeweiligen Systemkomponenten, wie Abbildung 6.16 für drei Systemkomponenten zeigt. Die als Multiplexer bezeichnete Komponente besteht im Kern aus zwei Teilen: Zum einen beinhaltet sie mehrere eins-zu-zwei Demultiplexer für das Lesesignal, um dieses anhand des decodierten Chipauswahlsignals an den entsprechenden Ausgang zu leiten. Zum anderen sind mehrere acht Bit breite zwei-zu-eins Multiplexer enthalten, um in Abhängigkeit des Chipauswahlsignals das jeweilige Datensignal von den Systemkomponenten zum Busmaster zu leiten. Abhängig von der Systemkomponentenanzahl ist das Chipauswahlsignal unterschiedlich breit, sodass jeweils eine unterschiedliche Anzahl an (De-)Multiplexern benötigt wird.

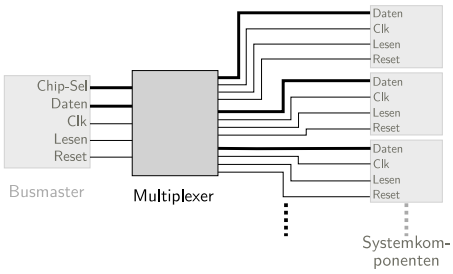


Abbildung 6.16: Topologie des Multiplexverfahrens

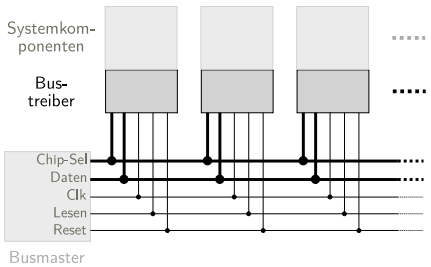


Abbildung 6.17: Topologie des Tristatebusses

Abbildung 6.17 zeigt die zweite Möglichkeit der Verbindung, welche auf gemeinsamen Leitungen für die Signale basiert. Jede Systemkomponente ist mit dem Chipauswahl-, Daten-, Takt-, Lese- und Resetsignal verbunden. Für die Verbindung der Datenausgänge ist jeweils ein Tristatetreiber erforderlich. Ist der jeweilige Baustein nicht ausgewählt, dann wird der Datenausgang auf einen hochohmigen Zustand (Z) gesetzt. Die Auswahl erfolgt mithilfe des Chipauswahlsignals, für dessen Decodierung eine Logik erforderlich ist. Durch die Verbindung der Systemkomponenten reduziert sich die Anzahl benötigter Leitungen. Jedoch entsteht zusätzlicher Aufwand für den Entwurf des Tristatetreibers. Zudem muss dieser für jede Systemkomponente entweder intern oder extern instanziiert werden.

Aus dem Aufbau der beiden Topologien wird die Entwurfsentropie für die benötigten Komponenten (Bustreiber beziehungsweise Multiplexer) für zwei bis acht Systemkomponentenanzahlen berechnet. Die Ergebnisse im folgenden Diagramm (Abbildung 6.18) zeigen, dass die Entwurfsentropie für vier Systemkomponenten (fast) gleich ist: 60,51 für das Multiplexverfahren und 59,84 für den Tristatebus.

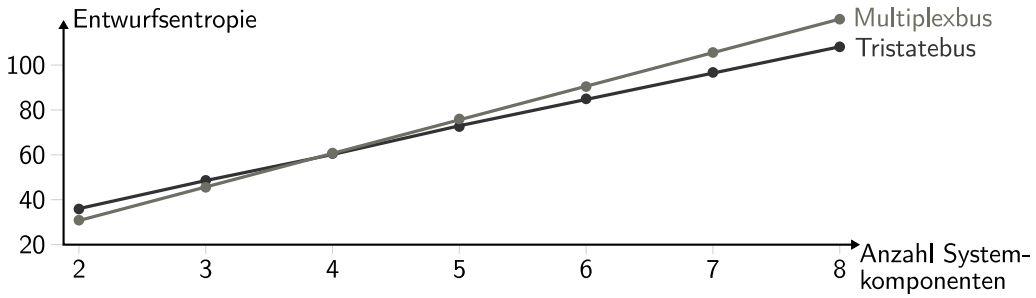


Abbildung 6.18: Vergleich eines Multiplex- und Tristatebusses

Da der Tristatetreiber aufwendiger zu entwerfen ist als der Multiplexer, ist die Entwurfsentropie für das Multiplexersystem bei einer geringeren Anzahl niedriger als für das Tristatesystem. Bei mehr Systemkomponenten wird dieser höhere Aufwand durch den geringeren Verdrahtungsaufwand beim Tristatebus kompensiert. Somit ist es effizienter, ein Multiplexingverfahren zu verwenden, wenn weniger als vier Komponenten angebunden werden und effizienter ein Bussystem zu entwerfen, wenn mehr als vier Komponenten angebunden werden. Dabei ist zu beachten, dass das Ergebnis vom aufgebauten Bus, der Art des Busses sowie der Leitungsanzahl abhängt. Bei aufwendigeren und insbesondere breiteren Bussen wird regelmäßig der Aufbau eines Bussystems bereits bei weniger als vier Systemkomponenten effizienter sein, da die Leitungsanzahl des Multiplexverfahrens schneller ansteigt.

6.3 MIPS-Prozessor

Im Rahmen der Lehrveranstaltung „Rechnerarchitektur“ an der Universität Ulm wurde von den Teilnehmerinnen und Teilnehmern eine CPU auf Basis der MIPS-Architektur (Microprocessor without Interlocked Pipeline Stages – MIPS) entworfen, welche grundsätzlich einer RISC-Architektur entspricht [223, S. 77]. Die MIPS basiert auf der Beschreibung von David Patterson und John Hennessy [166], welche in der folgenden Abbildung 6.19 gezeigt ist. Die Studierenden, welche in Gruppen von zwei bis drei Personen zusammenarbeiteten, entwarfen eine Einzyklenumsetzung ohne Pipelining. Zudem wurden nicht der vollständige Befehlssatz und alle möglichen ALU-Operationen umgesetzt, da dies den Rahmen der Veranstaltung überstiegen hätte. Die tatsächliche Umsetzung wurde den Gruppen selbst überlassen, sodass sich die Prozessoren hinsichtlich des Funktionsumfangs und der Strukturtiefe unterscheiden. Jedoch waren die Studierenden angehalten, die wesentlichen Komponenten aus Abbildung 6.19 umzusetzen. Das bedeutet, dass die Prozessoren zumindest über Steuerwerk, ALU, Register und getrennte Speicher verfügen sollten.

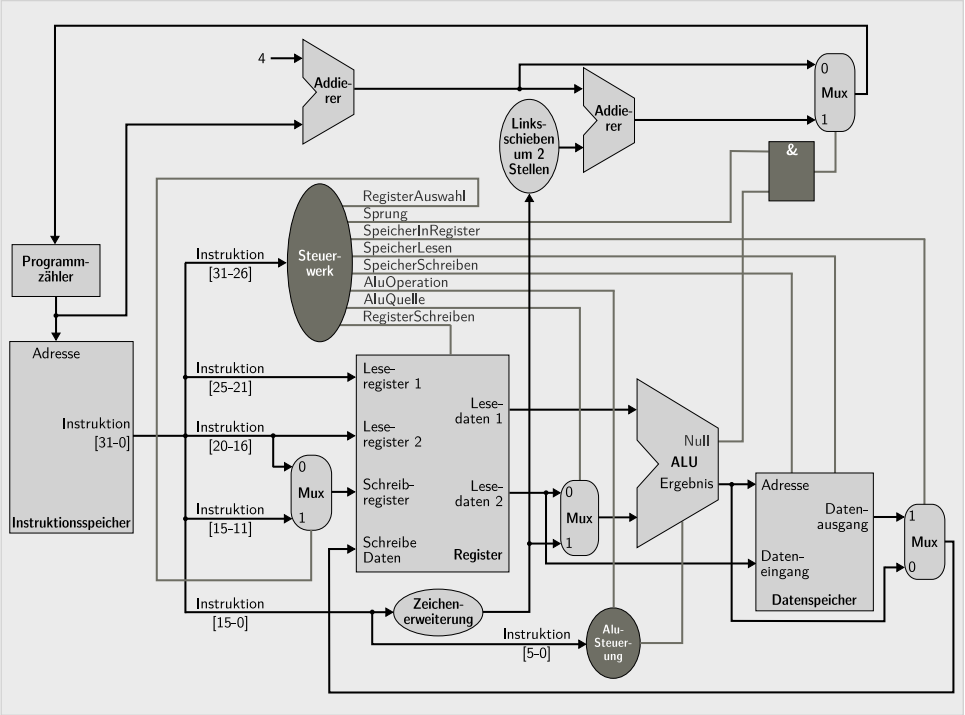


Abbildung 6.19: MIPS-Architektur [nach 166, S. 252]

Sechs Gruppen haben nach Abschluss der Veranstaltung ihre Implementierungen für diese Fallstudie bereitgestellt. Neben der Entwurfsentropie ist zum Vergleich die Anzahl an Quellcodezeilen angegeben. Hierzu wurde das Programm SLOCCount (Version 2.26) von David Wheeler [241] verwendet, welches die Anzahl an physikalischen Quellcodezeilen (Source Line of Code – SLoC) exklusive Kommentaren und Leerzeilen für alle verwendeten VHDL-Dateien ausgibt. Die Ergebnisse der Analyse für die Implementierungen der Gruppen G_1 bis G_6 sind in der folgenden Tabelle 6.6 eingetragen, wobei die Indizes für die Gruppen nach der Entwurfsentropie aufsteigend gewählt sind. Zusätzlich ist die Anzahl unterstützter Befehle für die folgende Diskussion der Ergebnisse mit aufgenommen worden.

	G_1	G_2	G_3	G_4	G_5	G_6
Entwurfsentropie	2.055	2.223	2.647	2.834	3.704	10.715
SLoC	324	470	312	432	817	1.148
Befehlsanzahl	3	9	7	6	10	58

Tabelle 6.6: Ergebnisse der MIPS-Auswertung

Wie der erste Abschnitt gezeigt hat, wirkt sich der Speicher stark auf die Ergebnisse aus. Daher wurden für die Vergleichbarkeit der Umsetzungen die Entitäten/ Codezeilen für die Daten- und Instruktionsspeicher entfernt, da teilweise ein externer Speicher angebunden wurde, ein interner Speicher realisiert wurde und teilweise kein Speicher, sondern nur eine direkte Ein- und Ausgabe entworfen wurde. Durch das Entfernen dieser Teile ist der Prozessor nicht mehr synthetisierbar. Zudem wurden die Entwürfe für unterschiedliche Zielarchitekturen umgesetzt, sodass keine Gatteranzahl zum Vergleich gewonnen werden konnte.

Bei den Werten in Tabelle 6.6 hebt sich die Implementierung von Gruppe G_6 ab, welche mit 58 Befehlen und Pipeliningunterstützung sehr umfangreich ausfällt. Dies drückt sich auch durch eine hohe Entwurfsentropie aus. Bei der Durchsicht des Quellcodes konnte festgestellt werden, dass die Umsetzung zu großen Teilen mit dem MIPS-Prozessor des Plasma-Projekts [187] von OpenCores übereinstimmt. Die Plasma-CPU ist ein synthetisierbarer 32-Bit-RISC-Mikroprozessor auf Basis der MIPS-I, der mit Ausnahme der patentieren Befehle alle Befehle des MIPS-I Befehlssatzes ausführen kann und wahlweise eine Zwei- oder Dreistufenpipeline besitzt [187]. Praktischerweise kann durch die Übernahme des Codes vom Plasma-Projekt die Implementierung von Gruppe G_6 als Referenz für eine vollständige Umsetzung der MIPS mit Pipelining herangezogen werden.

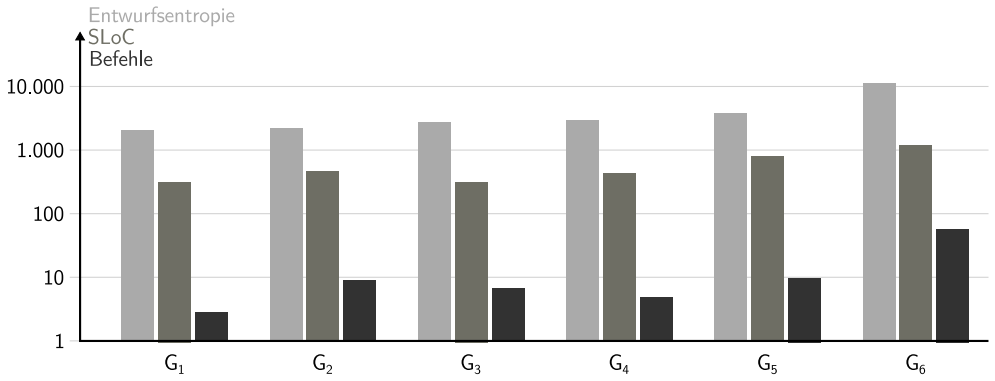


Abbildung 6.20: Grafische Darstellung der MIPS-Auswertung

Das vorherige Balkendiagramm (Abbildung 6.20) mit den Ergebnissen aus Tabelle 6.6 zeigt, dass die erste Gruppe eine sehr minimalistische Umsetzung gewählt hat, welche nur drei Befehle (addi, sub, beq) unterstützt. Beim Entwurf der ALU wurde auf das Arithmetikpaket zurückgegriffen, sodass die Operationen als Verhaltensbeschreibung mit den Operatoren Plus, Minus, Oder und Nicht-Oder umgesetzt sind. Zudem wurde der Programmcode fest im Quellcode hinterlegt und die Ausgabe erfolgt ausschließlich über die Siebensegmentanzeigen und Leuchtdioden des Entwicklungsboards. Gruppe zwei hat einen ähnlichen Ansatz gewählt, wobei neun Befehle unterstützt werden. Zusätzlich wurde für die ALU-Operationen Addieren und Subtrahieren ein Volladdierer aus Halbaddierern aufgebaut. Aufwendiger fallen die Umsetzungen der Gruppen drei und vier aus, welche hinsichtlich der Struktur und Funktionalität vergleichbar sind. Beide Gruppen haben sich eng an den Aufbau der MIPS aus Abbildung 6.19 gehalten und auch die einzelnen Addierer und Multiplexer umgesetzt. Gruppe drei hat zusätzlich eine Befehl-Holen-Entität entwickelt, welche die Datenpfade vom und zum Instruktionsspeicher aus Abbildung 6.19 umsetzt. Gruppe vier hat diese Funktionalität direkt in die Hauptentität eingebaut.

Hierarchisch sehr tief gehend fällt die Umsetzung von Gruppe fünf aus, welche die ALU-Operationen mithilfe einer Strukturbeschreibung bis hin zu den Bit-Verknüpfungen (Und, Oder, Nicht-Oder) implementiert hat und für die arithmetischen Operationen einen Aufbau aus Halb- und Volladdierern gewählt hat. Auch die Speicherzellen für die Register und Zwischenspeicher wurden mithilfe von D-Flipflops selbst umgesetzt. Abgesehen von Gruppe sechs weist Gruppe fünf sowohl die höchste Entwurfsentropie, die höchste Anzahl an Quellcodezeilen als auch den umfangreichsten Befehlssatz auf.

Die vorherige Darstellung zeigt, dass die Entwurfsentropie den Umfang und die Strukturtiefe der Entwürfe gut abbilden kann. Die Unterschiede in den Implementierungen spiegeln sich in den unterschiedlichen Werten für die Entwurfsentropie wider. Zusätzlich zeigt die Anwendung, dass auch umfangreichere Entwürfe analysiert werden können, welche auf unterschiedliche Arten implementiert sind. Für die Analyse benötigt das Analysewerkzeug nur wenige Sekunden. Um einen Vergleich der Entwurfsentropie mit der Anzahl an Logikelementen vornehmen zu können, wird im folgenden Abschnitt die Auswertung einer Lüfterregelung vorgestellt. Die vier Umsetzungen wurden jeweils für die gleiche Zielarchitektur in Form eines Entwicklungsboards entwickelt und das gleiche Synthesewerkzeug eingesetzt.

6.4 Lüfterregelung

Im Rahmen eines Studierendenprojekts an der Universität Ulm wurde eine Lüfterregelung entworfen. Diese bestand zum einen aus der Ansteuerung eines Lüfters mithilfe einer Pulsweitenmodulation (Pulse-Width Modulation – PWM) und zum anderen in der Messung der Drehzahl des Lüfters. Das Tachosignal (die Drehzahl) wird anhand der Taktzyklen zwischen zwei Flanken bestimmt, die vom Lüfter ausgegeben werden. Beide Komponenten wurden als Hardwarekomponenten umgesetzt, welche mithilfe eines Multiplexbusses an den Softcoreprozessor angebunden sind. Die eigentliche Regelung findet in Software auf dem Prozessor statt. Für die nachfolgende Analyse werden ausschließlich die beiden Hardwarekomponenten betrachtet, ohne die Anbindung an den Multiplexbus (Altera-Avalon-Interface), da die Anbindung durch das Synthesewerkzeug erfolgt.

Wie bei den MIPS-Implementierungen arbeiteten die Studierenden auch bei diesem Projekt in Gruppen mit maximal drei Personen zusammen und wurden nach Abschluss gefragt, ob sie ihre Implementierungen für eine Fallstudie zur Verfügung stellen. Daraufhin haben vier Gruppen ihre Umsetzungen für die Analyse bereitgestellt. Die Entwurfsentropiewerte für die Implementierungen sind in der folgenden Tabelle 6.7 eingetragen. Zum Vergleich mit der Anzahl physikalischer Quellcodezeilen wurde wie zuvor das Programm SLOCCount (Version 2.26) von David Wheeler [241] eingesetzt. Zusätzlich wurde die Anzahl an Logikelementen ausgewertet, welche mithilfe der Entwurfssoftware Quartus II (Version 12.1 SP 1) von Altera [5] für das eingesetzte Entwicklungsboard DE-2 ermittelt wurde.

	G_1	G_2	G_3	G_4
Entwurfsentropie	603	862	947	1.005
Logikelemente	176	180	164	237
SLoC	60	73	163	231

Tabelle 6.7: Ergebnisse der Auswertung der PWM-Komponenten

Die Gruppen eins und zwei haben die PWM-Komponente durch nur eine Entität umgesetzt, während die Gruppen drei und vier diese mit weiteren aufgebauten Unterkomponenten umgesetzt haben. Die Implementierung der ersten Gruppe hat im Vergleich zu den anderen drei Gruppen die geringste Entwurfsentropie und Quellcodezeilenanzahl. Dies ist darauf zurückzuführen, dass die Implementierung sehr minimalistisch ausgefallen ist und alles, was nur möglich war in den Treiber und somit in die Software ausgelagert wurde. Hieran zeigt sich auch die Gruppenzusammensetzung. Während die Gruppen zwei bis vier Studierende der Informationstechnik und/oder der Elektrotechnik waren, war Gruppe eins eine reine Informatikergruppe, welche mit Software vertrauter war als mit Hardware und somit die Umsetzung in Software bevorzugte. Für die visuelle Darstellung sind die Ergebnisse aus Tabelle 6.7 im folgenden Diagramm (Abbildung 6.21) gezeigt.

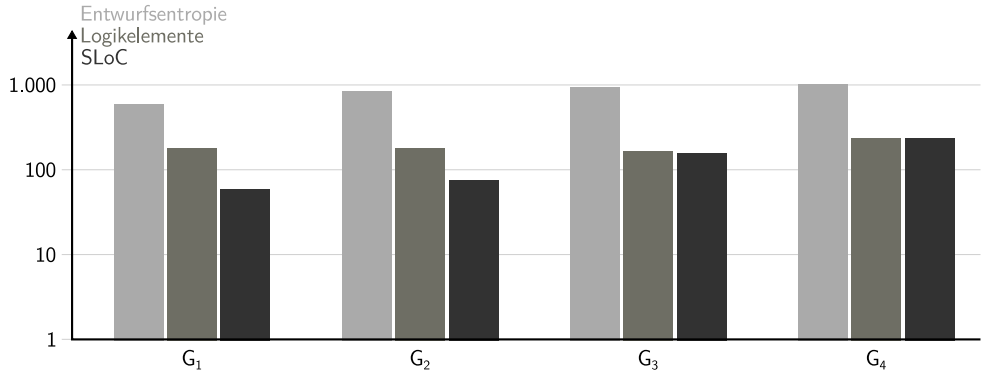


Abbildung 6.21: Grafische Darstellung der Auswertung der PWM-Komponente

Die höhere Entwurfsentropie und Logikelementanzahl von Gruppe vier ist darauf zurückzuführen, dass der 32-Bit-Zähler aus einzelnen D-Flipflops aufgebaut ist, welche ebenfalls selbst aufgebaut wurden. Der Unterschied zwischen den Implementierungen von Gruppe zwei und drei besteht darin, dass Gruppe zwei zur Speicherung der Periode und der Pulsweite ein Signal verwendete und Gruppe drei

hierfür eine eigene Register-Entität aufbaut hat, welche zur Speicherung vier Mal instanziiert wurde. Wird die Anzahl Logikgatter für die Gruppen G_1, G_2 und G_3 betrachtet, zeigen sich kaum Unterschiede trotz der im Vorherigen beschriebenen Unterschiede der Implementierungen. Auch die Anzahl an Quellcodezeilen ist sehr stark von der Anzahl an aufgebauten Entitäten abhängig und spiegelt kaum den tatsächlichen Aufwand und die Unterschiede wider. Im Gegensatz dazu werden die Unterschiede in den Implementierungen durch die unterschiedlichen Werte für die Entwurfsentropie gut abgebildet. Dieses Resultat zeigt sich auch bei der Auswertung der zweiten Hardwarekomponente. Die Ergebnisse der Analyse der Tacho-Komponente sind in der folgenden Tabelle 6.8 eingetragen.

	G_1	G_2	G_3	G_4
Entwurfsentropie	1.301	764	1.474	1.401
Logikelemente	220	135	302	238
SLoC	161	88	313	375

Tabelle 6.8: Ergebnisse der Auswertung der Tacho-Komponenten

Bei der Tacho-Komponente hebt sich die Implementierung von Gruppe zwei ab. Das Studium der Quelldateien zeigt, dass diese Gruppe die Ausgabe der Registerinhalte als eine kompakte Strukturbeschreibung mit einer einzigen with ... select-Anweisung umgesetzt hat und mit zwei Prozessen zur Verhaltensbeschreibung auskommt. Die anderen Gruppen haben sich bei der Implementierung der Tachokomponente stark an die PWM-Komponente angelehnt, was die Implementierungen aufwendiger macht. Der Unterschied der Entwurfsentropiewerte ist an der grafischen Darstellung in der folgenden Abbildung 6.22 gut erkennbar.

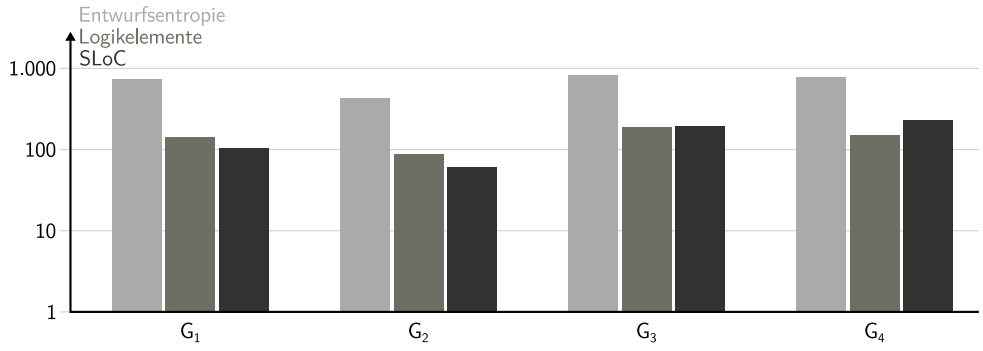


Abbildung 6.22: Grafische Darstellung der Auswertung der Tacho-Komponente

Im Gegensatz zur PWM-Komponente fällt die Umsetzung der Tachokomponente von Gruppe eins umfangreicher aus und zeigt viele Charakteristika einer sequenziellen Verarbeitung, wie diese in Software üblich ist. Dadurch fallen sowohl die Entwurfsentropie als auch die Anzahl an Quellcodezeilen höher aus als bei der ersten Gruppe. Gruppe vier hat wiederum den Ansatz gewählt, sowohl den 16-Bit- als auch 32-Bit-Counter aus einzelnen D-Flipflops aufzubauen. Die dritte Gruppe hat die Tacho-Komponente als Zustandsmaschine umgesetzt. Durch die unterschiedlichen Ansätze der Gruppen eins, drei und vier fallen die Entwurfsentropiewerte jeweils höher aus als der Wert von Gruppe zwei. Jedoch liegen diese drei Werte alle in einem vergleichbaren Bereich, obwohl unterschiedliche Ansätze verwendet wurden. Diese Unterschiede sind weder durch die Anzahl an Logikelementen noch durch die Anzahl an Quellcodezeilen zu erkennen.

6.5 Zwischenfazit

Anhand der vorherigen Ergebnisse zeigt sich, dass die Entwurfsentropie sehr gut geeignet ist, die Komplexität und Quantität eines Entwurfs zu bewerten. Zudem konnte durch die Analyse fremder Implementierungen gezeigt werden, dass das Analysewerkzeug verschiedenste Arten von VHDL-Anweisungen bei der Berechnung der Entwurfsentropie berücksichtigen kann. Dabei sind weder Anpassungen des Analysetools selbst noch der Quelldateien erforderlich.

Für zukünftige Studien sind insbesondere Projekte aus der Industrie von Interesse, bei welchen zum einen andere Metriken angewendet wurden und zum anderen auch der tatsächliche Aufwand bekannt ist. Somit könnte die Entwurfsentropie direkt mit dem Aufwand verglichen werden und auch als produktspezifischer Einflussfaktor für andere Aufwandsmetriken dienen. Bis jetzt war es jedoch nicht möglich, einen Zugang zu tatsächlichen Projektdaten mit den zugehörigen Aufwandswerten zu erlangen. Somit liegen den beiden letzten Fallstudien dieser Arbeit die Projekte von Studierenden zugrunde. Bereits an diesen Entwürfen lässt sich das Potenzial der Entwurfsentropie gut erkennen.

Den Studierenden wurden dabei die gleichen Aufgaben gestellt und es stand durch den Umfang der Lehrveranstaltung jeweils ein ähnlicher Zeitrahmen zur Verfügung. Bei der Auswertung des Prozessors (MIPS) zeigte sich, dass die Unterschiede in den Implementierungen durch die Entwurfsentropie abgebildet werden können. Die Aufgabenstellung der Lüfterregelung war wesentlich enger gefasst, sodass die

meisten Umsetzungen zu einer ähnlichen Entwurfsentropie führten. Für beide Hardwarekomponenten war jeweils eine Implementierung vorhanden, welche eine abweichende Entwurfsentropie besaß. Diese Abweichungen konnten direkt mit den Unterschieden bei den Implementierungen in Verbindung gebracht werden. Diese Resultate waren weder mit der Anzahl an Quellcodezeilen noch mit der Logikelementanzahl sichtbar.

Auch in der Literatur wurde das Maß der Entwurfsentropie als Komplexitätsmaß eingesetzt und es konnte festgestellt werden, dass dieses wesentlich genauer ist als lexikalische Metriken [201, S. 609].

7 Zusammenfassung und Ausblick

Die vorliegende Dissertation hat das Konzept der Entwurfsentropie vorgestellt, welches ein neues Maß beim Entwurf digitaler Schaltungen ist. Die Motivation zur Entwicklung dieses Maßes beruht darauf, dass insbesondere in den Bereichen der digitalen Hardware und integrierten Schaltungen keine zuverlässigen Methoden zur Komplexitätsbestimmung existieren. Dies kann unter anderem darauf zurückgeführt werden, dass es nur wenige Aufwandsbestimmungsmethoden gibt, welche speziell die digitale Hardware zum Gegenstand haben. Diejenigen Methoden, welche für Hardwareprojekte entwickelt wurden, benötigen einen umfangreichen Datenbestand aus bereits abgeschlossenen Projekten. Mithilfe dieses Datenbestands wird versucht, Rückschlüsse auf zukünftige Entwicklungen zu ziehen. Gerade im Bereich der Computerwissenschaften und speziell in den Bereichen der digitalen Hardware, integrierten Schaltungen und Eingebetteten Systeme unterliegt die Technologie einem sehr schnellen Wandel und einer enormen Weiterentwicklung. Daher ist es vielfach nicht möglich, genügend Daten bereits abgeschlossener Projekte zu sammeln, um hieraus zuverlässige Schlüsse für zukünftige Entwicklungen ziehen zu können. Zudem sind die in der Praxis eingesetzten Metriken durchgängig von Unternehmen entwickelt worden, sodass die tatsächliche Funktionsweise der Methoden unbekannt ist, vor allem hinsichtlich der genauen Einflussfaktoren. An dieser Stelle bietet die Entwurfsentropie eine Lösung an, da sie keine empirischen Daten für die Berechnung benötigt sowie offen und einheitlich definiert ist.

Allgemein bilden Einflussfaktoren die Basis jeder Aufwandsbestimmung, welche in zwei Gruppen eingeteilt werden können: produktbezogene Einflussfaktoren und projektbezogene Einflussfaktoren. Die Erstgenannten umfassen die Quantität (Entwurfsgröße) und die Komplexität eines Entwurfs. Andere Metriken beschränken sich bei den produktbezogenen Einflussfaktoren fast durchgängig auf die Betrachtung der Entwurfsgröße. Die Komplexität wird meist nur über eine qualitative Wertung innerhalb der projektbezogenen Einflussfaktoren berücksichtigt. Dagegen liegt der Entwurfsentropie der Ansatz zugrunde, dass die Komplexität für einen Entwurf konstant ist und von subjektiven Komplexitätsfaktoren (wie der Erfahrung

oder der Verfügbarkeit von Entwurfswerkzeugen) zu trennen ist. Somit muss diese objektive Komplexität in einem Entwurf beziehungsweise in einem Produkt selbst zu finden sein.

Die Komplexität eines Entwurfs kann jedoch nicht isoliert betrachtet werden, sondern nur zusammen mit der Entwurfsgröße. So stellt auch der Umfang eines Entwurfs eine produktspezifische Einflussgröße dar. Daher ist die Entwurfsentropie eine gemeinsame Maßzahl für die Quantität und die Komplexität. Sie kann somit auch als produktbezogener Einflussfaktor für andere Metriken verwendet werden. Für die Berechnung der Entwurfsentropie wird der Ansatz verfolgt, dass die Komplexität und Entwurfsgröße in einem Entwurf selbst zu finden sind und nicht über qualitative, subjektive Zuweisungen bestimmt werden müssen. Somit sind für die Berechnung der Entwurfsentropie keine umfangreichen Datenbestände aus abgeschlossenen Projekten erforderlich, welche zunächst ausgewertet werden müssen.

Als abstraktem Modell liegt der Entwurfsentropie das Verständnis zugrunde, dass ein System aus Komponenten besteht, welche Nachrichten austauschen. Dieses Verständnis erlaubt es, den vorgestellten Ansatz grundsätzlich auf alle Nachrichten verarbeitenden Systeme anzuwenden. Ein System ist dabei als strukturierte, systematische Gesamtheit von miteinander in Bezug stehenden Komponenten und Verbindungen zu verstehen, welche als aufgaben-, sinn- oder zweckgebundene Einheit angesehen werden kann. Für die digitale Hardware ist dies der zugrunde liegende Hardwareentwurf beziehungsweise der integrierte Schaltkreis. Diese Gesamtheit enthält Komponenten, welche über Verbindungen Nachrichten austauschen können, das heißt, insbesondere über elektrische Leitungen, welche durch Signale Nachrichten in Form von Nullen und Einsen übertragen.

Wird dieser Ansatz auf den Entwurfsprozess angewendet, dann ist der Kern einer Entwicklung, die Verbindungen zwischen den Komponenten herzustellen und dadurch den Nachrichtenaustausch in der Form zu ermöglichen, dass das System die vorgesehenen Aufgaben erfüllen kann. Neben der Verbindung von Komponenten werden innerhalb eines Entwurfs auch Komponenten aufgebaut. Dabei können diese Komponenten weitere (Unter-)Komponenten enthalten. Jedoch wird nicht jede Komponente innerhalb des Entwurfsprozesses aufgebaut. So können beispielsweise Gatter, Transistoren oder auch ganze Bausteine, wie Speicher, als sogenannte Elementarkomponenten verwendet werden. Daher besteht ein System zum einen aus Komponenten, welche durch Unterkomponenten entworfen werden und zum anderen aus Elementarkomponenten, welche innerhalb des Entwurfsprozesses nur

verwendet werden. Dabei kann auch eine entworfene Komponente an einer anderen Stelle des Systems wiederverwendet werden. Jedoch fällt der Aufwand für den Entwurf einer solchen Komponente nur ein Mal an. Dagegen fällt der Aufwand für die Verbindung der Schnittstellen dieser Komponente mit jeder Verwendung an. Aufgrund dieses Unterschieds berechnet sich die Entwurfsentropie zum einen aus der zuvor beschriebenen Strukturentropie, welche die Größe und Komplexität des Entwurfs einer Komponente ausdrückt und der Verhaltensentropie, welche die Komplexität der Verwendung einer Komponente ausdrückt.

Für die Verhaltensentropie werden die Schnittstellen einer Komponente betrachtet. Die Verbindung der Schnittstellen ermöglicht den Nachrichtenaustausch zwischen den Komponenten. Dabei ist es für den Entwurf erforderlich, das Verhalten einer Komponente zu kennen, das heißt, welche Nachrichten eine Komponente als Eingaben benötigt und basierend auf deren Verarbeitung, welche Nachrichten die Komponente als Ausgaben liefert. Mithilfe dieser Informationen können die Komponenten so verbunden werden, dass das Gesamtsystem die vorgesehenen Aufgaben erfüllen kann. Durch die Anzahl an Kommunikationsverbindungen, welche zwischen den Komponenten entworfen werden, fließt neben der Komplexität zusätzlich die Entwurfsgröße in das Maß ein. In Abhängigkeit von den übertragenen Nachrichten fallen die Kommunikationsverbindungen unterschiedlich aus, sodass statt der Verbindungen die übertragenen Nachrichten betrachtet werden, welche als mögliche Zustände der Schnittstellen aufgefasst werden.

Die Formeln zur Berechnung der Entwurfsentropie leiten sich aus der Informationstheorie ab. Innerhalb der Nachrichtentechnik beschreibt die Informationstheorie den Austausch von Informationen beziehungsweise Nachrichten über einen Kanal. Diesem Ansatz folgt die Entwurfsentropie, welche den Nachrichtenaustausch zwischen den Komponenten als Basis für die Berechnung nimmt. Somit basiert die Entwurfsentropie auf der Informationsentropie, von welcher sie auch ihren Namen bezieht.

Zusammenfassend zeichnet sich das in der vorliegenden Arbeit vorgestellte Konzept der Entwurfsentropie durch folgende Merkmale aus:

- Entwurfsbezogenheit:
  Durch die Verhaltens- und Strukturentropie wird sowohl die Quantität (Entwurfsgröße) als auch die (Schaltungs-)Komplexität berücksichtigt.

- Unabhängigkeit:
  Die Entwurfsentropie berechnet die in einem Entwurf/Produkt enthaltene (objektive) Komplexität und Quantität. Diese sind unabhängig von projektbezogenen Einflussfaktoren.

- Universalität:
  Die abstrakten Definitionen, der Komponentenansatz und das Verständnis des Nachrichtenaustauschs lassen die Anwendung der Entwurfsentropie auf unterschiedliche Bereiche der Nachrichten verarbeitenden Systeme zu, insbesondere auf Hardwareentwürfe und Softwareprogramme.

- Theoriegeleitetheit:
  Die Metrik leitet sich direkt von der Informationsentropie der Nachrichtentechnik ab. Die Berechnung der Entwurfsentropie erfolgt anhand von algebraischen Formeln.

- Nachvollziehbarkeit:
  Die berechnete Entwurfsentropie ist allein auf den Entwurf zurückzuführen, sodass die Berechnung nachvollziehbar bleibt, da keine Kalibrierung oder Ähnliches erforderlich ist.

- Hierarchie:
  Das System selbst besteht aus Komponenten, welche wiederum (Unter-) Komponenten enthalten können. Die Metrik berücksichtigt sowohl das Zusammenfassen als auch die Partitionierung von (Teil-)Systemen.

- Wiederverwendbarkeit:
  Die Unterscheidung zwischen der Verhaltens- und der Strukturentropie berücksichtigt sowohl die Wiederverwendbarkeit von Komponenten als auch den Rückgriff auf Bibliotheken.

- Direkte Berechenbarkeit:
  Die Metrik benötigt für die Berechnung der Entwurfsentropie keine empirischen Daten von abgeschlossenen Projekten, sodass sie direkt auf einen Entwurf angewendet werden kann.

- Automatisierung:
  Für die Analyse digitaler Hardwareprojekte wurde ein Analysewerkzeug entwickelt. Es erlaubt die automatisierte Berechnung der Entwurfsentropie für Hardwarebeschreibungen in VHDL.

- Abstraktheit:
  Indem Zustände als Basis für die Berechnung der Entwurfsentropie dienen, können weitere Randbedingungen über die Einschränkung des Zustandsraums berücksichtigt werden.

- Erweiterbarkeit:
  Die abstrakten Definitionen erlauben grundsätzlich den Einsatz der Metrik bei allen Nachrichten verarbeitenden Systemen, insbesondere in den Bereichen des Softwareengineerings und der Eingebetteten Systeme.

Im Rahmen von zwei Exkursen wurde zum einen die Möglichkeit der Zustandsreduktion vorgestellt. Dabei wird die Anzahl möglicher Zustände durch die Anzahl an Entwurfszuständen ersetzt, welche diejenigen Zustände umfasst, die innerhalb des Entwurfs tatsächlich umgesetzt wurden. Zum anderen konnte die grundsätzliche Anwendbarkeit auf Softwareprogramme gezeigt werden. Für diesen Zweck wurde ein sogenanntes Komponentendiagramm entwickelt. Dieses verbindet das (Kontroll-)Flussdiagramm mit dem Datenfluss und dem Komponentenansatz der Entwurfsentropie, sodass eine universelle Darstellungsmöglichkeit von Softwareprogrammen bereitsteht, welche die direkte Berechnung der Entwurfsentropie erlaubt.

Innerhalb der Fallstudien wurde gezeigt, dass die Entwurfsentropie dafür verwendet werden kann, den unterschiedlichen Entwurfsaufwand bei regelmäßigen und unregelmäßigen Strukturen abzubilden. Dies führte zu einem kritischen Blick auf die Entwurfslücke, da die Ergebnisse darauf schließen lassen, dass die Transistorbeziehungsweise Gatteranzahl als Maßzahlen relativ ungeeignet sind. Bei der Frage, ob ein Tristatebus oder ein Multiplexverfahren effizienter ist, um mehrere Systemkomponenten miteinander zu verbinden, konnte dargestellt werden, dass auf Basis der Entwurfsentropie auch Entwurfsentscheidungen getroffen werden können. Im Rahmen der Analyse von Studierendenprojekten zeigte sich unter anderem, dass das Analysewerkzeug in der Lage ist, fremde und auch umfangreichere Entwürfe zu analysieren, sowie dass Unterschiede in den Umsetzungen durch das Maß abgebildet werden können.

Insgesamt stellt die Entwurfsentropie einen vollständig neuen Ansatz dar, die Komplexität und die Entwurfsgröße von Nachrichten verarbeitenden Systemen zu berechnen und die Ergebnisse für den Entwurfsprozess nutzbar zu machen. Die Berechnung der Entwurfsentropie für reale Hardwareimplementierungen zeigt vielversprechende Ergebnisse, sodass die Entwurfsentropie viele der Schwächen heutiger Metriken überwinden kann. Zudem bietet sie eine Lösung an, um ohne umfangreiche empirische Daten den Aufwand von Projekten zu berechnen und Entwürfe direkt miteinander zu vergleichen.

Mit diesen Grundlagen erlaubt die vorliegende Dissertation, Forschungen und weitere Untersuchungen in vielfältigen Bereichen weiter zu verfolgen. Daher werden im Folgenden einige direkte Anknüpfungsmöglichkeiten an die vorliegende Arbeit vorgestellt.

Zunächst kann untersucht werden, wie sich der Einsatz von Verilog statt VHDL auf die Entwurfsentropie auswirkt. Grundsätzlich erlauben beide Sprachen eine vergleichbare Beschreibung von Systemen. Durch die unterschiedlichen Grammatiken fallen Verilog-Beschreibungen gemessen in der Anzahl an Codezeilen regelmäßig kürzer aus als VHDL-Beschreibungen. Die Hypothese ist, dass eine Vergleichbarkeit anhand der Quellcodezeilenanzahl nur sehr eingeschränkt möglich ist, während die Entwurfsentropie für beide Sprachen einen Vergleich zulässt, da jeweils vergleichbare Anzahlen an Zuständen und Verbindungen umgesetzt werden.

Neben verschiedenen Hardwarebeschreibungssprachen existieren auch unterschiedliche Synthesewerkzeuge für digitale Schaltungen. Bekannte Programme sind die Vivado Design Suite des Unternehmens Xilinx [247], Quartus Prime des Herstellers Altera [6], Synplify der Firma Synopsys [222] und die System Development Suite von Cadence [37]. Jede Umgebung stellt verschiedene Abstraktionsstufen, Werkzeuge und Bibliotheken mit vorgefertigten Komponenten zur Verfügung, welche die Umsetzung effizienter gestalten sollen. Aus wissenschaftlicher Sicht ist neben dem Vergleich der Werkzeuge hinsichtlich der Reduktion des Entwurfsaufwands insbesondere von Interesse, inwieweit Abstraktionsstufen, Werkzeuge und vorgefertigte Komponenten helfen, die Entwurfslücke zu schließen.

Im Zusammenhang mit den Entwurfswerkzeugen steht die Frage, wie sich die Komplexität und Entwurfsgröße einer Beschreibung vor und nach der Synthese verändert. Interdisziplinär kann diese Fragestellung nicht nur mithilfe der Entwurfsentropie erforscht werden, sondern es kann auch der nachrichtentechnische Informationsgehalt einer Hardwarebeschreibung betrachtet werden. Dabei wird

betrachtet, wie sich zum einen die Entwurfsentropie und zum anderen der Informationsgehalt einer Schaltung vor und nach der Synthese verändern. Mithilfe dieser Untersuchung lassen sich Aussagen über Gemeinsamkeiten und Unterschiede zwischen dem Informationsgehalt und der Entwurfsentropie gewinnen. Dies ist insbesondere vor dem Hintergrund interessant, als dass sich die Entwurfsentropie von der Informationstheorie ableitet.

Die Untersuchung des Informationsgehalts ist direkt mit der Fragestellung verbunden, welche Informationen für einen Entwurf tatsächlich erforderlich sind. Als Maß für die Entwurfsgröße und -komplexität analysiert die Entwurfsentropie die tatsächliche Umsetzung, unabhängig davon, ob eine Schaltung minimiert, optimiert oder präziser umgesetzt werden kann. Jedoch könnte sich aus dem Zusammenspiel mit der Nachrichtentechnik die Möglichkeit ergeben, dass eine minimal mögliche Schaltung angegeben werden kann. Hierzu kann auf der einen Seite der Informationsgehalt analysiert werden, das heißt, ob Informationen in einer Schaltung redundant vorhanden sind. Auf der anderen Seite kann auf Basis von Optimierungs-, Kompressions- und Codierungsalgorithmen der Nachrichtentechnik untersucht werden, wann eine Schaltung optimal ist, das heißt, ob eine Schaltung existiert, bei der keine Informationen mehr entfernt werden können.

Ein weiterer Forschungsbereich steht im Zusammenhang mit der Regelmäßigkeit von Strukturen. Wie in Abschnitt 6.1 anhand von Speichern und Prozessoren gezeigt wurde, hängt die Produktivität und damit die Entwurfslücke mit der Regelmäßigkeit der zu entwerfenden Strukturen zusammen. Für den Speicher und den Pipeliningprozessor konnte bereits gezeigt werden, dass die Wiederverwendung von Komponenten zu weitreichenden Einsparungen beim Entwurf führt. Heutzutage bestehen fast alle Prozessoren aus mehreren Kernen, wobei die einzelnen Kerne aus relativ unregelmäßigen Strukturen bestehen. Durch das Zusammenfügen mehrerer Kerne besteht der komplette Prozessor jedoch wieder aus regelmäßigen Kernen. Der Aufbau von Multicoreprozessoren erlaubt es somit, der Entwurfslücke zu begegnen. Jedoch entsteht wiederum Aufwand für zusätzliche Komponenten, wie beispielsweise von den Kernen gemeinsam genutzte Busse, Speicher oder Zwischenspeicher. Dieses Vorgehen ähnelt dem Aufbau eines Speichers, bei welchem durch das Zusammenfügen einzelner Speicherblöcke die Speichergröße skaliert werden kann und die Anpassung vorrangig innerhalb der Steuerung erfolgt. Von Interesse ist daher, wie Multicorearchitekturen den Entwurf beeinflussen, das heißt, wie die Auswirkungen auf die Produktivität sind. Für die Analyse eignet sich die Entwurfsentropie, da diese die Wiederverwendung von Komponenten berücksichtigt, wobei gleichzeitig der zusätzliche Verschaltungsaufwand beachtet wird.

Wie bei den Fallstudien im vorherigen Kapitel 6 angesprochen, sind für zukünftige Anwendungsstudien insbesondere auch Projekte aus der Industrie von Interesse, bei welchen zum einen andere Metriken angewendet wurden und zum anderen auch der tatsächliche Aufwand bekannt ist. Hierdurch kann sowohl festgestellt werden, in welchem Grad das Maß der Entwurfsentropie den Aufwand von industriellen Projekten erfassen kann als auch, ob das Maß als Einflussfaktor für andere Aufwandsbestimmungsverfahren geeignet ist und dadurch zu einer Verbesserung der Aufwandsschätzung beitragen kann.

In Abschnitt 4.10 wurde im Rahmen der Zustandsreduktion das Konzept der Entwurfszustände vorgestellt. Bei den Beispielen zeigte sich der entscheidende Vorteil der Entwurfszustände: Unabhängig von der Signal-/Variablendeklaration werden nur die tatsächlich umgesetzten Zustände betrachtet, sodass eine Verfälschung der Ergebnisse durch Signale/Variablen mit einer hohen Anzahl möglicher Zustände reduziert wird. Dabei muss weiter untersucht werden, wie sich Entwurfszustände auf die Berechnung der Entwurfsentropie auswirken. Das heißt, ob die Entwurfszustände tatsächlich helfen, Entwürfe mit unterschiedlichen Variablen-/ Signaldeklarationen vergleichbar zu machen.

Ebenfalls wurde im Zusammenhang mit den Entwurfszuständen auf die frühzeitige Abschätzung des Aufwands eingegangen. Wie alle Aufwandsschätzmetriken steht auch die Entwurfsentropie vor der Frage, wie die Einflussfaktoren frühzeitig abgeschätzt werden können. Der vorgestellte Ansatz basiert darauf, dass ein Projekt, welches sich in einer frühen Phase der Entwicklung befindet, nur ein unvollständiges Projekt ist, das sich nach dem Top-down-Ansatz in einem Prozess der Entwicklung durch Verfeinerung befindet. Dabei wird am Anfang eines Projekts die Entwurfsentropie mithilfe der Schnittstellendefinitionen abgeschätzt. Das heißt, zu Beginn wird angenommen, dass noch nicht bekannt ist, welche Zustände für die Umsetzung erforderlich sind, sodass zunächst von allen möglichen Zuständen ausgegangen wird. Während der Projektumsetzung erfolgt nach dem Top-down-Prinzip ein Prozess der Entwicklung durch Verfeinerung, sodass durch die Umsetzung der Komponenten der Entwurf immer detaillierter wird und es regelmäßig zu einer Zustandsreduktion kommt, da nicht alle möglichen Zustände umgesetzt werden. Dabei reduziert sich die Anzahl möglicher Zustände auf die Anzahl an Entwurfszuständen.

Ein anderer Ansatz für eine frühzeitige Abschätzung der Entwurfsentropie setzt auf der Kombination des in Unterabschnitt 3.2.23 vorgestellten SOG-Modells mit der Entwurfsentropie auf. Beim SOG-Modell wird die finale Anzahl Codezeilen

eines VHDL-Projekts auf Basis einer Highlevelbeschreibung vorhergesagt [55, S. 207]. Im Kern handelt es sich um einen klassischen Bottom-up-Ansatz, welcher den produktspezifischen Einflussfaktor (Quellcodezeilenanzahl) über die einzelnen Objekte (Teilkomponenten) errechnet. Statt der Vorhersage von Quellcodezeilen ist das Ziel der Kombination, die Entwurfsentropie beziehungsweise die Anzahl an Verbindungen und Zuständen vorherzusagen. Hierauf basierend kann, bei Kenntnis der Entwurfsentropie einiger Teilkomponenten, die Entwurfsentropie des gesamten Systems mit einer Art der Bottom-up-Methode abgeschätzt werden.

Neben der frühzeitigen Abschätzung ist für das Projektmanagement auch von Interesse, wie die Entwurfsentropiewerte in ökonomische Schlüsselvariablen überführt werden können. Hierzu muss zunächst ein quantitativer Zusammenhang zwischen einer „Entwurfsentropieeinheit“ und der Projektdauer, den Projektressourcen und dem Projektfortschritt gefunden werden. Bezüglich des Fortschritts eines Projekts wird erwartet, dass durch eine Berechnung der Entwurfsentropie über die Zustände ein gleichmäßiger Projektfortschritt beobachtet werden kann. Mit fortschreitendem Projekt steigt die Anzahl umgesetzter Zustände und Verbindungen an. Durch die logarithmische Funktion der Entwurfsentropieberechnung wird dies berücksichtigt. Somit wird die Metrik der Tatsache gerecht, dass das späte Hinzufügen oder Verändern von Komponenten einen oft weitreichenden Einfluss auf den bereits realisierten Teil hat.

Nach den bisherigen Studien eignet sich die Entwurfsentropie auch für die Anwendung auf Softwareprojekte, wie in Abschnitt 4.11 gezeigt wurde. Für die Berechnung der Entwurfsentropie und als universelle Darstellungsmöglichkeit hat sich das entwickelte Komponentendiagramm als geeignet erwiesen. Es verknüpft das (Kontroll-)Flussdiagramm, den Datenfluss und den Komponentenansatz der Entwurfsentropie miteinander. Mit dieser Grundlage kann ein Analysewerkzeug für Softwareprogramme entwickelt werden. Dieses erlaubt auf der einen Seite, die Qualität der entwickelten Entwurfsentropiesoftwaremetrik mit anderen Softwaremetriken zu vergleichen. Auf der anderen Seite kann durch die Identifikation bestehender Schwächen der jeweiligen Metriken untersucht werden, ob und inwieweit die in der vorliegenden Arbeit entwickelte Metrik bessere Ergebnisse liefert.

Um die Entwurfsentropie auf Eingebettete Systeme anwenden zu können, muss zunächst der Frage nachgegangen werden, wie „kontinuierliche Zustände“ in das Modell der Entwurfsentropie einbezogen werden können. Denn nicht nur die Komplexität und die Entwurfsgröße spielen bei digitalen Hardwareprojekten eine Rolle. Vielfach werden an Entwicklungen extra-funktionale Anforderungen gestellt. Bei-

spielsweise muss das Layout, das Zeitverhalten, die Wärmeentwicklung oder der Energieverbrauch berücksichtigt werden. Die Betrachtung dieser extra-funktionalen Anforderungen während eines Entwurfs erhöht die Komplexität. Da das Modell der Entwurfsentropie auf diskreten Zuständen aufbaut, stellt sich die Frage, wie physikalische Randbedingungen mit kontinuierlichen Werten berücksichtigt werden können. Physikalische Größen, wie Ort, Zeit, Wärme und Strom sind grundsätzlich kontinuierlich. Jedoch erfolgt beim Entwurf vielfach eine (indirekte) Diskretisierung: Transistoren werden an einem Raster ausgerichtet, die Analyse der Zeit erfolgt in Zeiteinheiten, die Wärme wird in ganzzahligen Grad Celsius gemessen und die Stromaufnahme bestimmt sich nach den einzelnen Bausteinen. Somit können Randbedingungen ebenfalls als mögliche Zustände betrachtet werden. Dieser Ansatz erlaubt es, extra-funktionale Anforderungen in die Entwurfsentropie einzubeziehen. Hiermit ist es auch möglich, die Effizienz von heutigen und zukünftigen Entwurfswerkzeugen zu untersuchen und ermöglicht es Informatikern, Informatikerinnen, Ingenieurinnen und Ingenieuren bei zukünftigen Projekten besser abzuschätzen, ob bestimmte Entwurfswerkzeuge und Technologien eingesetzt werden sollen.

Neben der Anwendung der Entwurfsentropie in der Geometriedomäne ist auch die Anwendung auf höheren Abstraktionsebenen interessant, insbesondere auf der Systemebene. Die Möglichkeit, Systembeschreibungssprachen wie SystemC zu analysieren, erlaubt es, die Komplexität von Eingebetteten Systemen sowie Hardware-Software-Systemen zu untersuchen. SystemC bietet sich in diesem Zusammenhang an, da hier sowohl die Software- als auch die Hardwarekomponenten in einer Umgebung beschrieben werden können. Zudem können aus der gleichen Beschreibung beide Arten von Komponenten synthetisiert werden. Aufbauend auf den vorherigen Punkten und den Ergebnissen dieser Arbeit können Regeln und Definitionen für die Anwendung der Entwurfsentropie auf die Grammatik und Struktur von SystemC gefunden werden, sodass der in der vorliegenden Arbeit gewählte Ansatz zur Umsetzung eines Analysewerkzeugs auch für die Analyse von SystemC-Beschreibungen dienen kann. Die Entwicklung einer Metrik für Hardware-Software-Systeme eröffnet dabei weitere weitreichende Anwendungs- und Forschungsgebiete.

Anhang

Abkürzungsverzeichnis

3GL	third-generation language
ACM	Apparatus Complexity Measure
ALU	Arithmetic Logic Unit (Arithmetisch-logische Einheit)
ANSI	American National Standards Institute
AP	Application Points (Anwendungspunkte)
CAD	Computer-Aided Design (Rechnerunterstützte Konstruktion)
COCOMO	Constructive Cost Model
COSMIC	Common Software Measurement International Consortium
COSYSMO	Constructive Systems Engineering Cost Model
CPU	Central Processing Unit (Hauptprozessor)
DEE	Design Effort Estimator (Entwurfsaufwandschätzer)
DSI	Delivered Source Instructions (Ausgelieferte Programmbefehle)
EBNF	Erweiterte Backus Naur Form
EDA	Electronic Design Automation (Entwurfsautomatisierung)
f.	folgend
ff.	folgende
FPA	Function-Point-Analysis (Funktionspunktanalyse)
GB	Gigabyte
HDL	Hardware Description Language (Hardwarebeschreibungssprache)
IC	Integrated Circuit (Integrierter Schaltkreis)
IEC	International Electrotechnical Comission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
k	kilo (Tausend)
LoC	Lines of Code (Quellcodezeilen)

MIPS	Microprocessor without Interlocked Pipeline Stages
MPU	Micro Processor Unit (Mikroprozessor)
OP	Object Points (Objektpunkte)
PM	Personenmonate
PRICE	Parametric Review of Information for Costing and Evaluation
PT	Personentage
PWM	Pulse-Width Modulation (Pulsweitenmodulation)
RISC	Reduced Instruction Set Computer
RTL	Register-Transfer-Level (Register-Transfer-Ebene)
S.	Seite(n)
SEER	System Evaluation and Estimation of Resources
SLIM	Software-Lifecycle-Management (Softwarelebenszyklusmodell)
SLoC	Source Lines of Codes (Quellcodezeilen)
SOG	Syntax Object Graph (Syntaxobjektgraph)
UFP	Unadjusted Function Points (Unangepasste Funktionspunkte)
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Anmerkung: Nicht aufgeführt sind Abkürzungen, welche ausschließlich als zusätzliche Informationen an einer einzigen Stelle verwendet werden (beispielsweise: Electronic Numerical Integrator and Computer – ENIAC) und bekannte Eigennamen (beispielsweise: International Business Machines Corporation – IBM) sowie Abkürzungen, welche ausschließlich im Literaturverzeichnis für die Publikationsdaten verwendet werden (beispielsweise: American Institute of Electrical Engineers – AIEE).

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Literaturverzeichnis

[1] Abran, Alain und Robillard, Pierre, *Function Points – A Study of their Measurement Processes and Scale Transformations*, in: Journal of Systems and Software, Band 25, Nummer 2, Seiten 171 – 184, Elsevier Science, New York, NY, USA, 1994.

[2] Aho, Alfred et al., *Compiler – Prinzipien, Techniken und Werkzeuge*, 2. Auflage, Pearson Studium, München, Deutschland, 2008.

[3] Albrecht, Allan, *Measuring Application Development Productivity*, in: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, Band 10, Seiten 83 – 92, IBM, Monterey, CA, USA, 1979.

[4] Albrecht, Allan und Gaffney, John, *Software Function, Source Lines of Code, and Development Effort Prediction – A Software Science Validation*, in: IEEE Transactions on Software Engineering, Band 9, Nummer 6, Seiten 639 – 648, IEEE, New York, NY, USA, 1983.

[5] Altera Corporation (Hrsg.), *Quartus II Subscription Edition*, dl.altera.com/download/software/quartus-ii-se/12.1 (Abgerufen am 10. Jan. 2016), San Jose, CA, USA, 2012.

[6] Altera Corporation (Hrsg.), *Quartus Prime Design Software*, www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html (Abgerufen am 13. Mai 2016), San Jose, CA, USA, 2016.

[7] Ammenwerth, Elske et al., *IT-Projektmanagement im Gesundheitswesen*, 2. Auflage, Schattauer, Stuttgart, Deutschland, 2015.

[8] Angelini, Chris, *Intel Core i7-3960X Review – Sandy Bridge-E and X79 Express*, www.tomshardware.com/reviews/core-i7-3960x-x79-sandy-bridge-e,3071.html (Abgerufen am 24. Dez. 2015), 2014.

[9] Armel, Kate, *Data-Driven Estimation, Management Lead to High Quality*, in: Software Quality Professional, Band 15, Nummer 2, Seiten 35 – 47, Quantitative Software Management, McLean, VA, USA, 2013.

[10] Armel, Kate, *The QSM Project Database*, in: QSM Software Almanac, Lungu, Angela (Hrsg.), Seiten 9 – 12, Quantitative Software Management, McLean, VA, USA, 2014.

[11] Aron, Joel, *Estimating Resources for Large Programming Systems*, in: Software Engineering Techniques, Buxton, John und Randell, Brian (Hrsg.), Seiten 68 – 79, NATO Science Committee, Rom, Italien, 1970.

[12] Aspinall, Willy, *A Route to More Tractable Expert Advice*, in: Nature, Band 463, Seiten 294 – 295, Nature Publishing Group, London, UK, 2010.

[13] Badstübner, Frank et al., *Quantitative Productivity Measurement in IC Design*, in: Proceedings of the Design, Automation and Test in Europe 2008, Seiten 934 – 935, ACM, München, Deutschland, 2008.

[14] Banker, Rajiv et al., *An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment*, in: Journal of Management Information Systems, Band 8, Nummer 3, Seiten 127 – 150, Taylor & Francis, Abingdon, UK, 1992.

[15] Banker, Rajiv et al., *Automating Output Size and Reuse Metrics in a Repository-Based Computer-Aided Software Engineering (CASE) Environment*, in: IEEE Transactions on Software Engineering, Band 20, Nummer 3, Seiten 169 – 187, IEEE, Piscataway, NJ, USA, 1994.

[16] Bazeghi, Cyrus et al., *μComplexity – Estimating Processor Design Effort*, in: Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture, Seiten 10 – 19, IEEE, Washington, DC, USA, 2005.

[17] Bazeghi, Cyrus et al., *System and Processor Design Effort Estimation*, in: VLSI-SoC – Advanced Topics on Systems on a Chip, Band 291, Seiten 249 – 269, Springer, Boston, MA, USA, 2009.

[18] Beckett, Donald, *An Analysis of Function Point Trends*, in: QSM Software Almanac, Lungu, Angela (Hrsg.), Seiten 59 – 68, Quantitative Software Management, McLean, VA, USA, 2014.

[19] Behrens, Charles, *Measuring the Productivity of Computer Systems Development Activities with Function Points*, in: IEEE Transactions on Software Engineering, Band 9, Nummer 6, Seiten 648 – 652, IEEE, Piscataway, NJ, USA, 1983.

[20] Beierlein, Thomas und Hagenbruch, Olaf, *Taschenbuch Mikroprozessortechnik*, Fachbuchverlag Leipzig, München, Deutschland, 1999.

[21] Berlinger, Eli, *An Information Theory Based Complexity Measure*, in: Proceedings of the 1980 National Computer Conference, Seiten 773 – 779, ACM, New York, NY, USA, 1980.

[22] Black, David et al., *SystemC – From the Ground Up*, 2. Auflage, Springer, New York, NY, USA, 2010.

[23] Blaschke, Jana et al., *ASIC Design Project Management Supported by Multi Agent Simulation*, in: Artificial Intelligence Applications and Innovations III, Band 296, Iliadis, Lazaros et al. (Hrsg.), Seiten 87 – 93, Springer, Boston, MA, USA, 2009.

[24] Blaschke, Jana et al., *Using Genetic Algorithms for Planning of Asic Chip-Design Project Flows*, in: IEEE Congress on Evolutionary Computation 2009, Seiten 1881 – 1888, IEEE, Piscataway, NJ, USA, 2009.

[25] Boehm, Barry, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[26] Boehm, Barry et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, Upper Saddle River, NJ, USA, 2000.

[27] Bossert, Martin, *Einführung in die Nachrichtentechnik*, Oldenbourg Verlag, München, Deutschland, 2012.

[28] Bossert, Martin, *Kanalcodierung*, 3. Auflage, Oldenbourg Verlag, München, Deutschland, 2013.

[29] Brand, Hans-Jürgen et al., *Referenzsystem zur Messung der Produktivität beim Entwurf nanoelektronischer Systeme*, Schlussbericht, edacentrum, Hannover, Deutschland, 2009.

[30] Brockman, Jay, *A Schema-Based Approach to CAD Task Management*, Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[31] Brown, Bernice, *Delphi Process – A Methodology Used for the Elicitation of Opinions of Experts*, www.rand.org/pubs/papers/P3925.html (Abgerufen am 13. Aug. 2015), RAND Corporation, Santa Monica, CA, USA, 1968.

[32] Brunnberg, Josef, *Optimale Lagerhaltung bei ungenauen Daten*, Verlag Dr. Gabler, Wiesbaden, Deutschland, 1970.

[33] Burghardt, Manfred, *Projektmanagement – Leitfaden für die Planung, Überwachung und Steuerung von Projekten*, 8. Auflage, Publicis Publishing, Erlangen, Deutschland, 2008.

[34] Burghardt, Manfred, *Einführung in Projektmanagement – Definition, Planung, Kontrolle, Abschluss*, 6. Auflage, Publicis Publishing, Erlangen, Deutschland, 2013.

[35] Cadence Design Systems (Hrsg.), *Cadence Tools*, www.cadence.com/products/Pages/all_products.aspx (Abgerufen am 5. Nov. 2015), San Jose, CA, USA.

[36] Cadence Design Systems (Hrsg.), *EDA Software and Verification Tools – Cadence Design Systems – The Leader in System Design Enablement*, www.cadence.com (Abgerufen am 5. Nov. 2015), San Jose, CA, USA.

[37] Cadence Design Systems, *System Development Suite*, www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification.html (Abgerufen am 18. Sep. 2016), Berkshire, UK, 2016.

[38] Carnap, Rudolf, *Induktive Logik und Wahrscheinlichkeit*, Springer, Wien, Österreich, 1959.

[39] Cobourn, Thomas, *Resource Management for CAD Frameworks*, Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[40] Common Software Measurement International Consortium (Hrsg.), *The COSMIC Functional Size Measurement Method – Version 3.0.1 – Deutsche Fassung*, 2011.

[41] Common Software Measurement International Consortium (Hrsg.), *The COSMIC Functional Size Measurement Method – Version 4.0*, 2014.

[42] Cook, Curtis, *Information Theory Metric for Assembly Language*, in: Proceedings of the Third Annual Oregon Workshop on Software Metrics, Silver Falls, OR, USA, 1991.

[43] Cooke, Roger, *Experts in Uncertainty – Opinion and Subjective Probability in Science*, in: Environmental Ethics and Science Policy, 11. Serie, Shrader-Frechette, Kristin (Hrsg.), Oxford University Press, New York, NY, USA, 1991.

[44] Cooke-Yarborough, Edmund, *Some Early Transistor Applications in the UK*, in: Engineering Science and Education Journal, Band 7, Nummer 3, Seiten 100 – 106, IET, Herts, UK, 1998.

[45] Dajsuren, Yanja et al., *Simulink Models are also Software – Modularity Assessment*, in: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, Seiten 99 – 106, ACM, Vancouver, Kanada, 2013.

[46] Dalkey, Norman und Helmer, Olaf, *An Experimental Application of the Delphi Method to the Use of Experts*, in: Management Science, Band 9, Nummer 3, Seiten 458 – 467, INFORMS, Catonsville, MD, USA, 1963.

[47] Davis, Boyd, *Intel Previews Intel Xeon Nehalem-EX Processor*, Intel Corporation (Hrsg.), www.intel.com/pressroom/archive/releases/2009/20090526comp.htm (Abgerufen am 24. Dez. 2015), Santa Clara, CA, USA, 2009.

[48] Davis, Joseph et al., *Creating Shared Information Spaces to Support Collaborative Design Work*, in: Information Systems Frontiers, Band 3, Nummer 3, Seiten 377 – 392, Kluwer Academic Publishers, Hingham, MA, USA, 2001.

[49] Eiselt, Hans-Peter, *Verfahren zur Programmierzeit-Schätzung*, in: 5. Jahrbuch der EDV 1976, Heilmann, Heidi (Hrsg.), Seiten 141 – 155, Forkel-Verlag, Stuttgart, Deutschland, 1976.

[50] El Arbi, Fedi und Ahlemann, Frederik, *Einleitung*, in: Strategisches Projektmanagement, Ahlemann, Frederik und Eckl, Christoph (Hrsg.), Seiten 1 – 21, Springer, Berlin, Deutschland, 2013.

[51] El-Aawar, Haissam, *Hardware Complexity of Microprocessor Design According to Moore's Law*, in: 3rd International Conference on Advanced Computer Science and Information Technology, Seiten 1 – 7, Zürich, Schweiz, 2014.

[52] El-Haik, Basem und Shaout, Adnan, *Software Design for Six-Sigma – A Roadmap for Excellence*, John Wiley & Sons, Hoboken, NJ, USA, 2011.

[53] Feess, Eberhard, *Komplexität*, Springer (Hrsg.), wirtschaftslexikon.gabler.de/Archiv/5074/komplexitaet-v8.html (Abgerufen am 10. März 2016), Gabler Wirtschaftslexikon, Wiesbaden, Deutschland, 2016.

[54] Fenstermaker, Stephen et al., *METRICS – A System Architecture for Design Process Optimization*, in: Proceedings of the 37th Annual Design Automation Conference, Seiten 705 – 710, ACM, Los Angeles, CA, USA, 2000.

[55] Fornaciari, William et al., *Early Estimation of the Size of VHDL Projects*, in: 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Seiten 207 – 212, ACM, Newport Beach, CA, USA, 2003.

[56] Gajski, Daniel, *Principles of Digital Design*, Prentice Hall, Upper Saddle River, NJ, USA, 1997.

[57] Gajski, Daniel und Kuhn, Robert, *New VLSI Tools*, in: Computer, Band 16, Nummer 12, Seiten 11 – 14, IEEE, Los Alamitos, CA, USA, 1983.

[58] Gajski, Daniel et al., *High-Level Synthesis – Introduction to Chip and System Design*, Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[59] Gajski, Daniel et al., *Specification and Design of Embedded Systems*, Prentice Hall, Englewood Cliffs, NJ, USA, 1994.

[60] Galorath Incorporated (Hrsg.), *SEER for Hardware, Electronics and Systems*, www.galorath.com/products/hardware/cost-estimating-software-hardware-projects (Abgerufen am 31. Okt. 2015), El Segundo, CA, USA.

[61] Galorath Incorporated (Hrsg.), *SEER for Hardware, Electronics & Systems*, Datenblatt, www.galorath.com/wp-content/uploads/2014/08/SEERforHardware2.pdf (Abgerufen am 31. Okt. 2015), El Segundo, CA, USA, 2011.

[62] Geirhos, Matthias, *IT-Projektmanagement*, Galileo Press, Bonn, Deutschland, 2014.

[63] Genz, Christian et al., *Visualization of SystemC Designs*, in: Proceedings of IEEE International Symposium on Circuits and Systems 2007, Seiten 413 – 416, IEEE, New Orleans, LA, USA, 2007.

[64] Goldstine, Herman und Goldstine, Adele, *The Electronic Numerical Integrator and Computer (ENIAC)*, in: Mathematical Tables and Other Aids to Computation, Band 2, Nummer 15, Seiten 97 – 110, American Mathematical Society, Providence, RI, USA, 1946.

[65] Großjohann, Rolf, *Kostenschätzung von IT-Projekten*, in: Software-Metriken in der Praxis – Einführung und Anwendungen von Software-Metriken in der industriellen Praxis, Ebert, Christof und Dumke, Reiner (Hrsg.), Seiten 117 – 141, Springer, Berlin, Deutschland, 1996.

[66] Gu, Jifa und Zhu, Zhichang, *Knowing Wuli, Sensing Shili, Caring for Renli – Methodology of the WSR Approach*, in: Systemic Practice and Action Research, Band 13, Nummer 1, Seiten 11 – 20, Kluwer Academic Publishers, Hingham, MA, USA, 2000.

[67] Halstead, Maurice, *Elements of Software Science – Operating and Programming Systems Series*, Elsevier North-Holland, New York, NY, USA, 1977.

[68] Harrison, Warren, *An Entropy-Based Measure of Software Complexity*, in: IEEE Transactions on Software Engineering, Band 18, Nummer 11, Seiten 1025 – 1029, IEEE, Piscataway, NJ, USA, 1992.

[69] Hartley, Ralph, *Transmission of Information*, in: The Bell System Technical Journal, Band 7, Nummer 3, Seiten 535 – 563, AT&T, Bedminster, NJ, USA, 1928.

[70] Hassine, Amir, *Ein Ansatz zur formalen Beschreibung und Simulation von Chipdesignprozessen*, Verlag Dr. Hut, München, Deutschland, 2009.

[71] Hassine, Amir und Barke, Erich, *Measure your Design Value to Improve it*, in: Proceedings of the 2005 IEEE International Engineering Management Conference, Band 2, Seiten 668 – 672, IEEE, Austin, TX, USA, 2005.

[72] Hassine, Amir und Barke, Erich, *On Modeling and Simulating Chip Design Processes – The RS Model*, in: IEEE International Engineering Management Conference 2008, IEEE, Estoril, Portugal, 2008.

[73] Hassine, Amir und Barke, Erich, *Towards Simulation of Chip Design Processes – The Request Service Model*, in: Proceedings of the 19th IASTED International Conference on Modelling and Simulation, Seiten 193 – 198, ACTA Press, Anaheim, CA, USA, 2008.

[74] Häusler, Stefan et al., *Modellierung von Komplexität und Qualität als Faktoren von Produktivität in Design-Flows für integrierte Schaltungen*, in: Tagungsband edaWorkshop 07, Seiten 59 – 64, Hannover, Deutschland, 2007.

[75] Häusler, Stefan et al., *Real-Time Quality Estimation to Enable Process Evaluation in Integrated Circuit Development*, in: IEEE International Engineering Management Conference 2008, Seiten 1 – 5, IEEE, Estoril, Portugal, 2008.

[76] Heil, Oskar, *Improvements in or relating to Electrical Amplifiers and other Control Arrangements and Devices*, Patent: GB 19350006815, Anmeldedatum: 02. März. 1934, London, UK, 1934.

[77] Heinrich, Lutz und Lehner, Franz, *Informationsmanagement – Planung, Überwachung und Steuerung der Informationsinfrastruktur*, 8. Auflage, Oldenbourg Verlag, München, Deutschland, 2005.

[78] Heintze, Nevin, *Set Based Program Analysis*, Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[79] Henze, Ernst, *Einführung in die Informationstheorie*, 2. Auflage, VEB Deutscher Verlag der Wissenschaften, Berlin, Deutschland, 1967.

[80] Hinrichs, Neele, *Performance-Management in Chipdesignprojekten unter Verwendung von Task-Graphen zur Prozessoptimierung*, Shaker Verlag, Aachen, Deutschland, 2012.

[81] Hinrichs, Neele und Barke, Erich, *Applying Performance Management on Semiconductor Design Processes*, in: IEEE International Conference on Industrial Engineering and Engineering Management 2008, Seiten 278 – 281, Singapore, 2008.

[82] Hinrichs, Neele et al., *Building up a Performance Measurement System to Determine Productivity Metrics of Semiconductor Design Projects*, in: IEEE International Engineering Management Conference 2007, Seiten 327 – 329, IEEE, Austin, TX, USA, 2007.

[83] Hinrichs, Neele et al., *An Approach for Analyzing and Evaluating Semiconductor Design Projects*, in: Proceedings of the World Congress on Engineering and Computer Science Vol II, Seiten 1061 – 1066, San Francisco, CA, USA, 2009.

[84] Hinrichs, Neele et al., *Optimization of Chip Design Processes using Task Graphs*, in: 2nd International Conference on Software Technology and Engineering, Band 1, Seiten 116 – 120, IEEE, San Juan, Puerto Rico, 2010.

[85] Höher, Peter, *Grundlagen der digitalen Informationsübertragung – Von der Theorie zu Mobilfunkanwendungen*, Springer, Wiesbaden, Deutschland, 2011.

[86] Horsch, Jürgen, *Innovations- und Projektmanagement – Von der strategischen Konzeption bis zur operativen Umsetzung*, Springer, Wiesbaden, Deutschland, 2013.

[87] Hosagrahara, Arvind, *Measuring Productivity and Quality in Model-Based Design*, Patent: US 7613589, Anmeldedatum: 27. Juli. 2005, Boston, MA, USA, 2005.

[88] Hosagrahara, Arvind und Smith, Paul, *Measuring Productivity and Quality in Model-Based Design*, in: Transactions Journal of Passenger Cars – Electronic and Electrical Systems, SAE, Warrendale, PA, USA, 2005.

[89] Hummel, Oliver, *Aufwandsschätzungen in der Software- und Systementwicklung kompakt*, Spektrum Akademischer Verlag, Heidelberg, Deutschland, 2011.

[90] IBM-Deutschland (Hrsg.), *Handbuch für die EDV-Organisation*, IBM-Form 71 523–2, Sindelfingen, Deutschland, 1968.

[91] IEEE (Hrsg.), *ANSI/IEEE Std 91-1984 – IEEE Standard Graphic Symbols for Logic Functions*, New York, NY, USA, 1984.

[92] IEEE (Hrsg.), *IEEE Std 1076-1987 – IEEE Standard VHDL Language Reference Manual*, New York, NY, USA, 1988.

[93] IEEE (Hrsg.), *ANSI/IEEE Std 91a-1991 – IEEE Standard Graphic Symbols for Logic Functions*, New York, NY, USA, 1991.

[94] IEEE (Hrsg.), *IEEE Std 1164-1993 – IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)*, New York, NY, USA, 1993.

[95] IEEE (Hrsg.), *IEEE Std 1076-2008 – IEEE Standard VHDL Language Reference Manual*, New York, NY, USA, 2009.

[96] Intel Corporation (Hrsg.), *Excerpts from a Conversation with Gordon Moore – Moore's Law*, ftp://download.intel.com/museum/Moores_Law/Video-transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf (Abgerufen am 9. Aug. 2015), Santa Clara, CA, USA, 2005.

[97] Intel Corporation (Hrsg.), *Microprocessor Quick Reference Guide*, www.intel.com/pressroom/kits/quickreffam.htm (Abgerufen am 24. Dez. 2015), Santa Clara, CA, USA, 2008.

[98] Intel Corporation und Micron Technology (Hrsg.), *Intel, Micron Introduce 25-Nanometer NAND – The Smallest, Most Advanced Process Technology in the Semiconductor Industry*, www.intel.com/pressroom/archive/releases/2010/20100201comp.htm (Abgerufen am 30. Nov. 2015), Santa Clara, CA, USA, 2010.

[99] International Function Point Users Group (Hrsg.), *Function Point Counting Practices Manual – Release 4.0*, Princeton Junction, NJ, USA, 1994.

[100] International Function Point Users Group (Hrsg.), *Function Point Counting Practices Manual – Release 4.3.1*, Princeton Junction, NJ, USA, 2010.

[101] International Function Point Users Group (Hrsg.), *The IFPUG Guide to IT and Software Measurement*, Princeton Junction, NJ, USA, 2012.

[102] International Organization for Standardization (Hrsg.), *ISO/IEC 20968:2002 – Software Engineering – MK II Function Point Analysis – Counting Practices Manual*, Genf, Schweiz, 2002.

[103] International Organization for Standardization (Hrsg.), *ISO/IEC 20926:2009 – Software and Systems Engineering – Software Measurement – IFPUG Functional Size Measurement Method*, Genf, Schweiz, 2009.

[104] International Organization for Standardization (Hrsg.), *ISO/IEC 19761:2011 – Software Engineering – COSMIC – A Functional Size Measurement Method*, Genf, Schweiz, 2011.

[105] Irvine, Maury, *Early Digital Computers at Bell Telephone Laboratories*, in: IEEE Annals of the History of Computing, Band 23, Nummer 3, Seiten 22 – 42, IEEE, Piscataway, NJ, USA, 2001.

[106] Jacome, Margarida, *Design Process Planning and Management for CAD Frameworks*, Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1993.

[107] Jakoby, Walter, *Projektmanagement für Ingenieure*, Springer, Wiesbaden, Deutschland, 2010.

[108] Jakoby, Walter, *Intensivtraining Projektmanagement – Ein praxisnahes Übungsbuch für den gezielten Kompetenzaufbau*, Springer, Wiesbaden, Deutschland, 2015.

[109] Jansen, Dirk, *Hardware/Software Co-Design*, in: The Electronic Design Automation Handbook, Jansen, Dirk (Hrsg.), Seiten 172 – 198, Springer, Dordrecht, Niederlande, 2007.

[110] Jansen, Dirk, *Introduction*, in: The Electronic Design Automation Handbook, Jansen, Dirk (Hrsg.), Seiten 22 – 32, Springer, Dordrecht, Niederlande, 2007.

[111] Jiang, Rong, *Research and Measurement of Software Complexity Based on Wuli, Shili, Renli (WSR) and Information Entropy*, in: Entropy, Band 17, Nummer 4, Seiten 2094 – 2116, MDPI, Basel, Schweiz, 2015.

[112] Johannesson, Rolf, *Informationstheorie – Grundlage der (Tele-)Kommunikation*, Studentlitteratur, Lund, Schweden, 1992.

[113] Johnson, Eric, *Analysis and Refinement of Iterative Design Processes*, Dissertation, University of Notre Dame, Notre Dame, IN, USA, 1996.

[114] Joyner, Claude et al., *Pros, Cons, and Alternatives to Weight Based Cost Estimating*, in: Proceedings of the 47th AIAA/ASME/SAE/ASEE Joint Propulsion Conference, San Diego, CA, USA, 2011.

[115] Kahng, Andrew und Mantik, Stefanus, *A System for Automatic Recording and Prediction of Design Quality Metrics*, in: International Symposium on Quality Electronic Design 2001, Seiten 81 – 86, IEEE, San Jose, CA, USA, 2001.

[116] Kanellos, Michael, *Moore's Law to Roll On for Another Decade*, news.cnet.com/2100-1001-984051.html (Abgerufen am 9. Aug. 2015), CNET, San Francisco, CA, USA, 2003.

[117] Kesel, Frank und Bartholomä, Ruben, *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs – Einführung mit VHDL und SystemC*, 3. Auflage, Oldenbourg Verlag, München, Deutschland, 2013.

[118] Koppe, Roland et al., *Impact Estimation for Design Flow Changes*, in: Tagungsband edaWorkshop 11, Seiten 69 – 74, Dresden, Deutschland, 2011.

[119] Kunkler, Edward, *A Cooperative Industry Study on Software Development/Maintenance Productivity*, Xerox Corporation, Rochester, NY, USA, 1983.

[120] Kuusi, Osmo, *Expertise in the Future Use of Generic Technologies – Epistemic and Methodological Considerations Concerning Delphi Studies*, Report, VATT-Research – Government Institute for Economic Research, Helsinki, Finnland, 1999.

[121] Lang, Bernhard, *Kombinatorische Schaltungen*, in: Taschenbuch Digitaltechnik, 3. Auflage, Siemers, Christian und Sikora, Axel (Hrsg.), Seiten 121 – 138, Carl Hanser Verlag, München, Deutschland, 2014.

[122] Lange, Walter und Bogdan, Martin, *Entwurf und Synthese von Eingebetteten Systhemen – Ein Lehrbuch*, Oldenbourg Verlag, München, Deutschland, 2013.

[123] Laranjeira, Luiz, *Software Size Estimation of Object-Oriented Systems*, in: IEEE Transactions on Software Engineering, Band 16, Nummer 5, Seiten 510 – 522, IEEE, Piscataway, NJ, USA, 1990.

[124] Lavington, Simon, *A History of Manchester Computers*, British Computer Society, London, UK, 1998.

[125] Leppelt, Peter et al., *An Approach to Make Semiconductor Design Projects Comparable*, in: Asian Pacific Industrial Engineering and Management Systems Conference 2006, Seiten 2067 – 2074, Bangkok, Thailand, 2006.

[126] Levy, Sean et al., *An Overview of the n-dim Environment*, White Paper, n-dim group, Pittsburgh, PA, USA, 1993.

[127] Lilienfeld, Julius, *Electric Current Control Mechanism*, Patent: CA 272437, Anmeldedatum: 22. Okt. 1925, New York, NY, USA, 1925.

[128] Lilienfeld, Julius, *Method and Apparatus for Controlling Electric Currents*, Patent: US 1745175, Anmeldedatum: 08. Okt. 1926, Brooklyn, NY, USA, 1926.

[129] Lipsett, Roger et al., *VHDL – Hardware Description and Design*, 15. Auflage, Kluwer Academic Publishers, Norwell, MA, USA, 1989.

[130] Litke, Hans-Dieter, *IT-Projekte richtig strukturieren und systematisch planen*, in: Handbuch IT-Projektmanagement, Tiemeyer, Ernst (Hrsg.), Seiten 175 – 226, Carl Hanser Verlag, München, Deutschland, 2010.

[131] Markowsky, George, *Information Theory*, www.britannica.com/topic/information-theory (Abgerufen am 13. Nov. 2015), Encyclopædia Britannica, Chicago, IL, USA, 2015.

[132] Marwedel, Peter, *Eingebettete Systeme*, Springer, Heidelberg, Deutschland, 2008.

[133] Mastretti, Mirella, *VHDL Quality – Synthesizability, Complexity and Efficiency Evaluation*, in: Proceedings of the Conference on European Design Automation 1995, Seiten 482 – 487, IEEE, Los Alamitos, CA, USA, 1995.

[134] McCabe, Thomas, *A Complexity Measure*, in: IEEE Transactions on Software Engineering, Band 2, Nummer 4, Seiten 308 – 320, IEEE, Piscataway, NJ, USA, 1976.

[135] McKenzie, John, *TX-0 Computer History*, Technischer Bericht, Massachusetts Institute of Technology: Research Laboratory of Electronics, Cambridge, MA, USA, 1974.

[136] McKinsey & Company (Hrsg.), *IC Industry Database*, www.mckinsey.com/client_service/semiconductors/tools_and_solutions/ic_industry_database (Abgerufen am 27. Okt. 2015), New York, NY, USA.

[137] McKinsey & Company (Hrsg.), *IC Project Planner*, www.mckinsey.com/client_service/semiconductors/tools_and_solutions/ic_project_planner (Abgerufen am 27. Okt. 2015), New York, NY, USA.

[138] McKinsey & Company (Hrsg.), *Numetrics*, www.numetrics.com (Abgerufen am 27. Okt. 2015), New York, NY, USA.

[139] McKinsey & Company (Hrsg.), *Numetrics*, www.mckinseysolutions.com/solutions/numetrics.aspx (Abgerufen am 27. Okt. 2015), New York, NY, USA.

[140] McKinsey & Company (Hrsg.), *Numetrics, McKinsey R&D Analytics Solutions*, www.mckinsey.com/client_service/semiconductors/tools_and_solutions (Abgerufen am 27. Okt. 2015), New York, NY, USA.

[[141]] Menhorn, Benjamin und Slomka, Frank, *Entwurfsentropie – Ein Maß im Schaltungsentwurf*, in: 7th GI/GMM/ITG-Workshop Multi-Nature-Systems, Günzburg, Deutschland, 2009.

[[142]] Menhorn, Benjamin und Slomka, Frank, *Project Management Through States*, in: International Conference on Engineering Management and Service Sciences, IEEE, Piscataway, NJ, USA, 2009.

[[143]] Menhorn, Benjamin und Slomka, Frank, *Design Entropy Concept – A Measurement for Complexity*, in: ESWEEK 2011 Compilation Proceedings, Seiten 285 – 294, ACM, New York, NY, USA, 2011.

[[144]] Menhorn, Benjamin und Slomka, Frank, *States and Complexity*, in: Coping with Complexity, Dumitrescu, Dan et al. (Hrsg.), Seiten 68 – 88, Cluj-Napoca, Ungarn, 2011.

[[145]] Menhorn, Benjamin und Slomka, Frank, *Confirming the Design Gap*, in: Advances in Computational Science, Engineering and Information Technology, Band 225, Nagamalai, Dhinaharan et al. (Hrsg.), Seiten 281 – 292, Springer, Cham, Schweiz, 2013.

[[146]] Menhorn, Benjamin und Slomka, Frank, *Quantitative Analysis of Software Code by States*, in: Proceedings of the 8th IASTED International Conference on Advances in Computer Science, Assawinchaichote, Wudhichai et al. (Hrsg.), ACTA Press, Calgary, Kanada, 2013.

[[147]] Menhorn, Benjamin und Slomka, Frank, *VHDL-Komplexitätsanalyse*, Technischer Bericht, Universität Ulm, Ulm, Deutschland, 2015.

[[148]] Menhorn, Benjamin et al., *Digital Hardware Projects – A New Tool for Automated Complexity Analysis*, in: Proceedings of the 8th IEEE International Symposium on Industrial Embedded Systems, IEEE, Piscataway, NJ, USA, 2013.

[149] Meyer Media (Hrsg.), *Pentium 4 Die Wallpaper*, www.deskpicture.com/DPs/Technology/p4_die_1.html (Abgerufen am 30. Nov. 2015), 2011.

[150] Micron Technology (Hrsg.), *NAND Flash Memory – MT29F8G08FABWP*, Datenblatt, www.micron.com/parts/nand-flash/mass-storage/mt29f8g08fabwp (Abgerufen am 10. Aug. 2016), Boise, ID, USA, 2004.

[151] Micron Technology (Hrsg.), *NAND Flash Memory – MT29F8G08ADBDAH4*, Datenblatt, www.micron.com/parts/nand-flash/mass-storage/mt29f8g08adbdah4 (Abgerufen am 11. Aug. 2016), Boise, ID, USA, 2009.

[152] Milutis, Michael (Hrsg.), *Focus on Lawrence Putnam – A CAI State of the Practice Interview*, Computer Aid, Allentown, PA, USA, 2006.

[153] Mollick, Ethan, *Establishing Moore's Law*, in: IEEE Annals of the History of Computing, Band 28, Nummer 3, Seiten 62 – 75, IEEE, Piscataway, NJ, USA, 2006.

[154] Moore, Gordon, *Cramming More Components onto Integrated Circuits*, in: Electronics, Band 38, Nummer 8, Seiten 114 – 117, IEEE, Piscataway, NJ, USA, 1965.

[155] Moore, Gordon, *Progress in Digital Integrated Electronics*, in: International Electron Devices Meeting, Band 21, Seiten 11 – 13, IEEE, Piscataway, NJ, USA, 1975.

[156] Müller, Silvia und Paul, Wolfgang, *The Complexity of Simple Computer Architectures*, Springer, Berlin, Deutschland, 1995.

[157] Mueller, Silvia und Paul, Wolfgang, *Computer Architecture – Complexity and Correctness*, Springer, Berlin, Deutschland, 2010.

[158] Narendra, Siva et al., *Through the Looking Glass Continued (III) – Update to Trends in Solid-State Circuits and Systems from ISSCC 2014*, in: IEEE Solid-State Circuits Magazine, Band 6, Nummer 1, Seiten 49 – 53, IEEE, Piscataway, NJ, USA, 2014.

[159] n-dim group (Hrsg.), *n-dim group*, www.ndim.edrc.cmu.edu (Abgerufen am 31. Okt. 2015), Pittsburgh, PA, USA.

[160] Norden, Peter, *Resource Usage and Network Planning Techniques*, in: Operations Research in Research and Development, Dean, Burton (Hrsg.), Seiten 149 – 169, John Wiley & Sons, London, UK, 1963.

[161] Noth, Thomas und Kretzschmar, Mathias, *Aufwandschätzung von DV-Projekten – Darstellung und Praxisvergleich der wichtigsten Verfahren*, 2. Auflage, Springer, Berlin, Deutschland, 1986.

[162] Numetrics Management Systems (Hrsg.), *Measuring IC and ASIC Design Productivity*, White Paper, Santa Clara, CA, USA, 2000.

[163] Nyquist, Harry, *Certain Factors Affecting Telegraph Speed*, in: Journal of the AIEE, Band 3, Nummer 2, Seiten 324 – 346, AT&T, Bedminster, NJ, USA, 1924.

[164] Parr, Terence, *The Definitive ANTLR Reference – Building Domain-Specific Languages*, The Pragmatic Bookshelf, Raleigh, NC, USA, 2007.

[165] Parr, Terence und Harwell, Sam, *ANTLR v4*, www.antlr.org (Abgerufen am 20. Nov. 2015), San Francisco, CA, USA.

[166] Patterson, David und Hennessy, John, *Rechnerorganisation und -entwurf – Die Hardware/Software-Schnittstelle*, 3. Auflage, Elsevier, München, Deutschland, 2005.

[167] Pfleeger, Shari et al., *Software Cost Estimation and Sizing Methods – Issues and Guidelines*, Rand Corporation, Santa Monica, CA, USA, 2005.

[168] Poston, Bill und Dury, Joe, *Semiconductor Product Lifecycle Management – Industry Adoption, Benefits and the Road Ahead*, White Paper, Kalypso, Beachwood, OH, USA, 2006.

[169] Potemski, Andrew und Srinivasan, Ravi, *Setting up a Versatile Flow and Environment to Improve Design Productivity*, White Paper, Synopsys, Mountain View, CA, USA, 2006.

[170] PRICE Systems (Hrsg.), *Hardware Estimating Model for TruePlanning – Version 3.0*, White Paper, www.pricesystems.com/resources/article/Hardware-Estimating-Model-for-TruePlanning.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2010.

[171] PRICE Systems (Hrsg.), *A Built-In Path to Data-Driven Estimating*, Datenblatt, www.pricesystems.com/resources/article/Services-TrueFindings.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[172] PRICE Systems (Hrsg.), *A Comprehensive Data-Driven Approach to Cost Management*, Datenblatt, www.pricesystems.com/resources/article/Overview-Brochure-Capability-Brief.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[173] PRICE Systems (Hrsg.), *A Predictive Cost Modeling System for Better Decision-Making*, Datenblatt, www.pricesystems.com/resources/article/Product-TruePlanning%C2%AE.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[174] PRICE Systems (Hrsg.), *Data-Driven Credibility in Life Cycle Cost Management*, Datenblatt, www.pricesystems.com/resources/article/Services-ESI-Overview.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[175] PRICE Systems (Hrsg.), *Harness the Power of Historical Data*, Datenblatt, www.pricesystems.com/resources/article/Services-Data-Mining-Analysis.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[176] PRICE Systems (Hrsg.), *Map Estimate Results to Practical Uses, More Easily*, Datenblatt, www.pricesystems.com/resources/article/Services-TrueMapper.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[177] PRICE Systems (Hrsg.), *PRICE H Hardware and PRICE HL Hardware Life Cycle Cost Models*, Datenblatt, www.pricesystems.com/resources/article/Cost-Model-Hardware.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[178] PRICE Systems (Hrsg.), *Specialized Cost Models for Military Structure Alternatives*, Datenblatt, www.pricesystems.com/resources/article/Cost-Model-Concept.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2013.

[179] PRICE Systems (Hrsg.), *Plug-and-Play Cost Objects for Estimating Space Science Missions*, Datenblatt, www.pricesystems.com/resources/article/Cost-Model-Space-Misisons.aspx (Abgerufen am 26. Okt. 2015), Mount Laurel, NJ, USA, 2015.

[180] Putnam, Lawrence, *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*, in: IEEE Transactions on Software Engineering, Band 4, Nummer 4, Seiten 345 – 361, IEEE, Piscataway, NJ, USA, 1978.

[181] Putnam, Lawrence, *Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System*, in: The IEEE Computer Society's 2nd International Computer Software and Applications Conference, Seiten 827 – 832, IEEE, Piscataway, NJ, USA, 1978.

[182] Quantitative Software Management (Hrsg.), *Quantitative Software Management*, www.qsm.com (Abgerufen am 4. Nov. 2015), McLean, VA, USA.

[183] Rechenberg, Peter, *Zum Informationsbegriff der Informationstheorie*, in: Informatik-Spektrum, Band 26, Nummer 5, Seiten 317 – 326, Springer, Berlin, Deutschland, 2003.

[184] Reich, Yoram et al., *Building Agility for Developing Agile Design Information Systems*, in: Research in Engineering Design, Band 11, Nummer 2, Seiten 67 – 83, Springer, London, UK, 1999.

[185] Reichardt, Jürgen und Schwarz, Bernd, *VHDL-Synthese – Entwurf digitaler Schaltungen und Systeme*, 5. Auflage, Oldenbourg Verlag, München, Deutschland, 2009.

[186] Reichardt, Jürgen und Schwarz, Bernd, *VHDL-Synthese – Entwurf digitaler Schaltungen und Systeme*, 6. Auflage, Oldenbourg Verlag, München, Deutschland, 2013.

[187] Rhoads, Steve, *Plasma – Most MIPS I(TM) Opcodes*, www.opencores.org/project, plasma (Abgerufen am 31. Juli 2016).

[188] Riedlinger, Reid et al., *A 32 nm 3.1 Billion Transistor 12-Wide-Issue Itanium Processor for Mission-Critical Servers*, in: IEEE International Solid-State Circuits Conference, Seiten 84 – 86, IEEE, Piscataway, NJ, USA, 2011.

[189] Robinson, Michael, *Improve SoC Design Productivity by Performing Quality Checks on IP Timing Constraints*, White Paper, Synopsys, Mountain View, CA, USA, 2006.

[190] Rojas, Raùl, *How to make Zuse's Z3 a Universal Computer*, in: IEEE Annals of the History of Computing, Band 20, Nummer 3, Seiten 51 – 54, IEEE, Piscataway, NJ, USA, 1998.

[191] Rost, Manfred und Wefel, Sandro, *Elektronik für Informatiker – Von den Grundlagen bis zur Mikrocontroller-Applikation*, Oldenbourg Verlag, München, Deutschland, 2013.

[192] Roth, Charles und John, Lizy, *Digital Systems Design Using VHDL*, 2. Auflage, Thomson Learning, Toronto, Kanada, 2008.

[193] Sackman, Harold, *Delphi Assessment – Expert Opinion, Forecasting, and Group Process*, www.rand.org/pubs/reports/R1283.html (Abgerufen am 13. Aug. 2015), RAND Corporation, Santa Monica, CA, USA, 1974.

[194] Safonov, Vladimir, *Trustworthy Compilers*, John Wiley & Sons, Hoboken, NJ, USA, 2010.

[195] Scarpazza, Daniele, *A Development Effort and Size Estimation Method for Partially and Fully Specified VDHL Projects*, Abschlussarbeit, Politecnico Di Milano, Mailand, Italien, 2002.

[196] Schatten, Alexander et al., *Best Practice Software-Engineering – Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*, Springer, Heidelberg, 2010.

[197] Schönfeld, Dagmar et al., *Informations- und Kodierungstheorie*, 4. Auflage, Springer, Wiesbaden, Deutschland, 2012.

[198] Schütz, Alfred, *The Concept of Electronic Design Automation*, in: The Electronic Design Automation Handbook, Jansen, Dirk (Hrsg.), Seiten 33 – 49, Springer, Dordrecht, Niederlande, 2007.

[199] Seidl, Helmut et al., *Übersetzerbau – Analyse und Transformation*, Band 3, Springer, Berlin, Deutschland, 2010.

[200] Semiconductor Industry Association (Hrsg.), *2015 Factbook*, www.semiconductors.org/semiconductors/2015_sia_factbook (Abgerufen am 5. Aug. 2015), Washington, DC, USA, 2015.

[201] Sergeichik, Vladimir et al., *Obfuscation and Watermarking of FPGA Designs based on Constant Value Generators*, in: 14th International Symposium on Integrated Circuits, Seiten 608 – 611, IEEE, Piscataway, NJ, USA, 2014.

[202] Shannon, Claude, *A Mathematical Theory of Communication*, in: The Bell System Technical Journal, Band 27, Nummer 3, Seiten 379 – 423, AT&T, Bedminster, NJ, USA, 1948.

[203] Shermon, Dale, *Systems Cost Engineering – Program Affordability Management and Cost Control*, Gower, Farnham, UK, 2009.

[204] Shrout, Ryan, *Intel Xeon E5-2600 v3 Processor Overview – Haswell-EP up to 18 Cores*, www.pcper.com/reviews/Processors/Intel-Xeon-E5-2600-v3-Processor-Overview-Haswell-EP-18-Cores (Abgerufen am 24. Dez. 2015), 2014.

[205] Sikora, Axel, *Sequenzielle Schaltungen*, in: Taschenbuch Digitaltechnik, 3. Auflage, Siemers, Christian und Sikora, Axel (Hrsg.), Seiten 139 – 174, Carl Hanser Verlag, München, Deutschland, 2014.

[206] Silbey, Alex und Johnston, Shannon, *Delivering During the Downturn – Engineering Management Strategies at Top IC Companies*, Webinar, Numetrics, Cupertino, CA, USA, 2009.

[207] Smith, Paul et al., *Best Practices for Establishing a Model-Based Design Culture*, in: Systems Engineering, SAE, Warrendale, PA, USA, 2007.

[208] Sneed, Harry, *Schätzung der Entwicklungskosten von objektorientierter Software*, in: Informatik-Spektrum, Band 19, Nummer 3, Seiten 133 – 140, Springer, Berlin, Deutschland, 1996.

[209] Sneed, Harry, *Aufwandsschätzung in IT-Projekten*, in: Handbuch IT-Projektmanagement – Vorgehensmodelle, Managementinstrumente, Good Practices, Tiemeyer, Ernst (Hrsg.), Seiten 267 – 306, Carl Hanser Verlag, München, Deutschland, 2010.

[210] Sneed, Harry, *Aufwandsschätzung in IT-Projekten*, in: Handbuch IT-Projektmanagement – Vorgehensmodelle, Managementinstrumente, Good Practices, 2. Auflage, Tiemeyer, Ernst (Hrsg.), Seiten 273 – 314, Carl Hanser Verlag, München, Deutschland, 2014.

[211] Sohnius, Richard et al., *An Approach for Assessing Design Systems – Design System Simulation and Analysis for Performance Assessment*, in: 9th International Conference on Enterprise Information Systems, Funchal, Portugal, 2007.

[212] Sohnius, Richard et al., *Holonic Simulation of a Design System for Performance Analysis*, in: Holonic and Multi-Agent Systems for Manufacturing, Band 4659, Mařík, Vladimír et al. (Hrsg.), Seiten 447 – 454, Springer, Berlin, Deutschland, 2007.

[213] Solka, Michael und Srinivasan, Ravi, *Measuring and Improving IC Design Productivity*, White Paper, Synopsys, Mountain View, CA, USA, 2006.

[214] Spallek, Rainer und Zabel, Martin, *Computer*, in: Taschenbuch der Informatik, 7. Auflage, Schneider, Uwe (Hrsg.), Seiten 79 – 118, Carl Hanser Verlag, Leipzig, Deutschland, 2012.

[215] Stellman, Andrew und Greene, Jennifer, *Applied Software Project Management*, O'Reilly Media, Sebastopol, CA, USA, 2006.

[216] Stensrud, Erik, *Estimating with Enhanced Object Points vs. Function Points*, in: Proceedings of the 13th COCOMO/SCM Forum, Los Angeles, CA, USA, 1998.

[217] Stiller, Andreas, *Prozessorgeflüster – i-Tüpfelchen von der ISSCC und dem IDF*, in: c't, Nummer 5, Seiten 16 – 20, Heise Medien, Hannover, Deutschland, 2000.

[218] Stockmayer, Friedemann und Kreutzer, Hans, *Design using Standard Description Languages*, in: The Electronic Design Automation Handbook, Jansen, Dirk (Hrsg.), Seiten 86 – 145, Springer, Dordrecht, Niederlande, 2007.

[219] Subrahmanian, Eswaran et al., *The n-dim Approach to Creating Design Support Systems*, in: Proceedings of Design Engineering Technical Conference, ASME, Sacramento, CA, USA, 1997.

[220] Sutton, Peter, *A Framework and Discipline Independent Approach to Design Process Management*, Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.

[221] Symons, Charles, *Software Sizing and Estimating – MK II FPA (Function Point Analysis)*, Wiley, Chichester, UK, 1991.

[222] Synopsys, *Synplify Pro and Premier*, www.synopsys.com/Tools/Implementation/FPGAImplementation/CapsuleModule/synplify-pro-premier.pdf (Abgerufen am 18. Sep. 2016), Mountain View, CA, USA, 2015.

[223] Tanenbaum, Andrew, *Computerarchitektur – Strukturen, Konzepte, Grundlagen*, 5. Auflage, Pearson Studium, München, Deutschland, 2006.

[224] Tapio, Petri, *Disaggregative Policy Delphi – Using Cluster Analysis as a Tool for Systematic Scenario Formation*, in: Technological Forecasting and Social Change, Band 70, Nummer 1, Seiten 83 – 101, Elsevier Science, New York, NY, USA, 2003.

[225] Teich, Jürgen und Haubelt, Christian, *Digitale Hardware/Software-Systeme*, 2. Auflage, Springer, Berlin, Deutschland, 2007.

[226] The Inquirer (Hrsg.), *Gordon Moore says Aloha to Moore's Law*, www.theinquirer.net/inquirer/news/1014782/gordon-moore-aloha-moore-law (Abgerufen am 9. Aug. 2015), London, UK, 2005.

[227] The International Technology Roadmap for Semiconductors (Hrsg.), *International Technology Roadmap for Semiconductors 1999 Edition – Design*, Report, www.itrs.net/reports.html (Abgerufen am 5. Juli 2008), 1999.

[228] The International Technology Roadmap for Semiconductors (Hrsg.), *International Technology Roadmap for Semiconductors 2007 Edition – Design*, Report, www.itrs.net/reports.html (Abgerufen am 2. März 2010), 2007.

[229] The International Technology Roadmap for Semiconductors (Hrsg.), *International Technology Roadmap for Semiconductors 2011 Edition – Design*, Report, www.itrs.net/reports.html (Abgerufen am 9. Juni 2015), 2011.

[230] The World Semiconductor Trade Statistics (Hrsg.), *WSTS has Published the Final Semiconductor Market Figures for 2014*, www.wsts.org/content/download/3548/24110 (Abgerufen am 5. Aug. 2015), San Jose, CA, USA, 2014.

[231] Turoff, Murray, *The Design of a Policy Delphi*, in: Technological Forecasting and Social Change, Band 2, Nummer 2, Seiten 149 – 171, Elsevier Science, New York, NY, USA, 1970.

[232] United Kingdom Software Metrics Association (Hrsg.), *MK II Function Point Analysis – Counting Practices Manual – Version 1.3.1*, Kent, UK, 1998.

[233] Urbanski, Klaus und Woitowitz, Roland, *Digitaltechnik – Ein Lehr- und Übungsbuch*, 4. Auflage, Springer, Berlin, Deutschland, 2004.

[234] Valerdi, Ricardo, *The Constructive Systems Engineering Cost Model*, Dissertation, University of Southern California, Los Angeles, CA, USA, 2005.

[235] Valerdi, Ricardo et al., *COSYSMO – A Constructive Systems Engineering Cost Model Coming of Age*, in: 13th Annual International Symposium of the International Council On Systems Engineering, Band 13, Nummer 1, Seiten 70 – 82, Washington, DC, USA, 2003.

[236] Valerdi, Ricardo et al., *Systems Engineering Sizing in the Age of Acquisition Reform*, in: 14th Annual International Symposium of the International Council on Systems Engineering, Band 14, Nummer 1, Los Angeles, CA, USA, 2004.

[237] Van Staa, Peter und Sebeke, Christian, *Can Multi-Agents Wake Us from IC Design Productivity Nightmare?*, in: Holonic and Multi-Agent Systems for Manufacturing, Band 4659, Mařík, Vladimír et al. (Hrsg.), Seiten 15 – 16, Springer, Berlin, Deutschland, 2007.

[238] Walston, Claude und Felix, Charles, *A Method of Programming Measurement and Estimation*, in: IBM Systems Journal, Band 16, Nummer 1, Seiten 54 – 73, IBM, Riverton, NJ, USA, 1977.

[239] Wecker, Dieter, *Prozessorentwurf – Von der Planung bis zum Prototyp*, 2. Auflage, De Gruyter, Berlin, Deutschland, 2015.

[240] Werner, Martin, *Information und Codierung – Grundlagen und Anwendungen*, Mildenberger, Otto (Hrsg.), Vieweg & Sohn, Braunschweig, Deutschland, 2002.

[241] Wheeler, David, *SLOCCount (Source Lines of Code Count)*, www.dwheeler.com/sloccount (Abgerufen am 29. Nov. 2015).

[242] Wildemann, Horst, *Beschleunigte Entwicklungsprozesse in der Elektroindustrie – Abschlussbericht*, TCW Transfer-Centrum Verlag, München, Deutschland, 2003.

[243] Wilhelm, Reinhard et al., *Übersetzerbau – Syntaktische und semantische Analyse*, Band 2, Springer, Berlin, Deutschland, 2012.

[244] Wirth, Niklaus, *Grundlagen und Techniken des Compilerbaus*, 2. Auflage, Oldenbourg Verlag, München, Deutschland, 2008.

[245] Wolf, Jürgen, *C von A bis Z – Das umfassende Handbuch*, 3. Auflage, Rheinwerk Verlag, Bonn, Deutschland, 2009.

[246] Wolverton, Ray, *The Cost of Developing Large-Scale Software*, in: IEEE Transactions on Computers, Band 23, Nummer 6, Seiten 615 – 636, IEEE, Washington, DC, USA, 1974.

[247] Xilinx, *Vivado Design Suite*, www.xilinx.com/products/design-tools/vivado.html (Abgerufen am 18. Sep. 2016), San Jose, CA, USA, 2016.

[248] Zaum, Daniel et al., *Automatic Data Extraction – A Prerequisite for Productivity Measurement*, in: IEEE International Engineering Management Conference 2008, Seiten 1 – 5, IEEE, Estoril, Portugal, 2008.

[249] Zhu, Zhichang, *Dealing with a Differentiated Whole – The Philosophy of the WSR Approach*, in: Systemic Practice and Action Research, Band 13, Nummer 1, Seiten 21 – 57, Kluwer Academic Publishers, Hingham, MA, USA, 2000.

[250] Zuse, Konrad, *Der Computer – Mein Lebenswerk*, 5. Auflage, Springer, Heidelberg, Deutschland, 2010.

Lebenslauf

Persönliche Daten

Name	Benjamin Menhorn
Anschrift	Ferdinand-Sauerbruch-Straße 16 89134 Blaustein
Telefonnummer	0179 7747119
E-Mail	benjamin@menhorn.eu
Geburtsdatum	26. September 1982
Geburtsort	Bietigheim-Bissingen

Werdegang

Seit		04/2016	Leitung Controlling bei der RSU GmbH in Ulm.
10/2008	–	03/2016	Wissenschaftlicher Mitarbeiter an der Universität Ulm am Institut für Eingebettete Systeme/Echtzeitsysteme.
10/2002	–	09/2008	Wirtschaftsphysik (Diplom) an der Universität Ulm mit der Note sehr gut (1,4) und dem Diplomarbeitsthema: „From product idea to product introduction: the example of a small international operating US company“.
10/2010	–	06/2014	Rechtswissenschaften (Bachelor) an der Fernuniversität in Hagen mit der Note sehr gut (1,4) und dem Bachelorarbeitsthema: „Rechtsfragen der Online-Übermittlung“.
09/1993	–	06/2002	Allgemeine Hochschulreife am Friedrich-Abel-Gymnasium in Vaihingen/Enz mit der Note 1,8.