Masterarbeit MA705

Verilog Based Power Simulation

Matthias Seiß

Institut für Elektronik
Technische Universität Graz
Leiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Bösch

Begutachter: Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser
Betreuer (TU Graz): Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser
Betreuer (Infineon): Dipl.-Ing. Helmut Koroschetz

Graz, im April 2012

Diese Masterarbeit wurde unterstützt von
Infineon Technologies Austria AG
Development Center Graz
Abteilung Chipcard - Contactless Innovation
Leitung: Dipl.-Ing. Dr.techn. Walter Kargl

Kurzfassung

Die Leistungsfähigkeit von integrierten Schaltungen hat sich in den letzten Jahren enorm gesteigert. Immer mehr Transistoren werden auf immer kleiner werdenden Flächen integriert. Mit der gesteigerten Rechenleistung wird auch der Energieverbrauch eines ICs zu einem immer wichtiger werdenden Faktor, speziell bei mobilen Anwendungen wie RFID. Um die erhöhte Leistungsaufnahme in Grenzen zu halten, werden neben konventionellen Methoden wie Verkleinerung der Lastkapazitäten oder Verringerung der Schaltfrequenz auch dynamische Methoden wie z.B. *Clock Gating* angewandt. Dabei werden Logikelemente, die im aktuellen Taktzyklus nicht aktiv sind, gezielt vom Takt getrennt, sollte der aktuelle Energieverbrauch des Chips bzw. des betreffenden Moduls zu hoch sein. Somit wird die Leistungsaufnahme des Blocks - und damit auch des gesamten Chips - reduziert.

Vor dieser Arbeit konnten zuverlässige Energie-Abschätzungen nur für konstante Stimuli, die aus einer vorhergehenden Digitalsimulation gewonnen werden mussten, durchgeführt werden. Das bedeutet, dass dynamische Effekte wie z.B. die Abschaltung von Clocks aufgrund eines zu hohen Stromverbrauchs, nicht, oder nur in unzureichender Genauigkeit, simuliert werden konnten. Ziel dieser Masterarbeit war es, diese dynamischen Verläufe der Energieaufnahme eines Chips während der normalen Digitalsimulation mitverfolgen zu können. Dazu müssen zuerst mit Hilfe der Netzkapazitäten und der Dauer der Signalübergänge die entsprechenden Energien für jeden möglichen Signalwechsel einer jeden Zelle berechnet und in der Netzliste des Designs als Parameter hinterlegt werden. Um diese Energiewerte dann zu verarbeiten, müssen die funktionalen Modelle der Zelle entsprechend angepasst werden. Diese Anpassungen sollen vollautomatisch von einem Perl-Skript durchgeführt werden. Die Ergebnisse aus den Simulationen sollen für verschiedene Testfälle mit den Ergebnissen aus dem Power-Analyse-Tool PrimeTimePX® verglichen und damit verifiziert werden.

Durch diese Arbeit wurde es möglich, die aktuelle Energieaufnahme des Chips modulfein für jeden einzelnen Takt zu bestimmen und auf diese auch entsprechend reagieren zu können. Mit Hilfe der bereitgestellten Stromaufnahme kann auch ein eventueller Einbruch der Versorgungsspannung berechnet werden, dem durch einen Regler entgegengewirkt werden kann.

Schlagwörter: Leistungs-Simulation, Verilog, Stromverbrauch, dynamische Leistung, statische Leistung

Abstract

The performance of integrated circuits is highly increasing in the last few years. The number of transistors is doubled every 1.5 years, the integration level is growing fast. Due to this fact, power dissipation of an IC has become more and more important, especially in mobile applications like RFID. To control und reduce the power dissipation of an integrated circuit, several techniques are used. Besides conventional methods like decreasing load capacitances and reducing switching frequency, other (more complex) methods like *Clock Gating* are used. If the actual current consumption of the chip is too high, Clock Gating suppresses particular clocks connected to logical cells. So the concerning block is deactivated and the dynamic power of this cells is nearly extincted.

Until this master thesis, authentic power estimation could only be done for constant stimuli, which were gained from a preceding digital simulation run. This means, that dynamic effects like disabling clocks due to too high current consumption could not, or only in insufficient accuracy, be simulated. The goal of this work is to create a possibility to simulate the actual power dissipation of the chip simultaneously to the normal digital simulation. To achieve this, all energy values for every possible input state and every occuring output transition have to be calculated for all cells in the design. This values should be patched into the netlist to use it in the simulation. To process the values in the simulation, the functional models of the standard cells also have to be modified. All this calculation und patching should be done with an Perl script, which gets the load capacitances and the input transition times of the current design as input. To verify the correctness of the power simulator, the results should be compared with reports from the commercial power estimation tool PrimeTimePX®.

This work enables the possibility to simulate the current power dissipation of the chip with very high accuracy. It is now possible to recognize a supply voltage drop because of too high current consumption. The information of the falling voltage can be fed into a control system which can regulate and compensate the drop.

Keywords: Power-Simulation, Verilog, dynamic power, leakage power

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Datum	Ort	Unterschrift

Danksagung

Diese Arbeit wurde bei der Infineon Technologies Austria AG in Graz durchgeführt. Deshalb gilt mein erster Dank dem gesamten Team der CL-Inno-Abteilung, welches mir immer mit freundlichen und hilfreichen Antworten auf meine Fragen Unterstützung geboten hat.

Bedanken möchte ich mich auch bei meinen Betreuern Dipl.-Ing. Helmut Koroschetz bei Infineon und Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser auf Seiten der TU Graz.

Mein größter Dank gilt meiner gesamten Familie, allen voran meinen Eltern, die mir das Studium an der Technischen Universität Graz erst ermöglicht haben. Nicht erst seit meiner Studienzeit boten sie mir immer Rückhalt und Unterstützung in jedweder Art und Weise. Einen großen Dank möchte ich auch meinem Bruder aussprechen, welcher mir während meiner gesamten Ausbildung immer zur Seite gestanden hat und das auch hoffentlich im weiteren Verlauf unserer beider Lebenswege machen wird. Ohne diese großartige Unterstützung seitens meiner Familie wäre ein Studium in dieser Form nicht möglich gewesen. Danke!

Zum Schluss möchte ich noch meinen Freunden danken, durch die mein Studium an der TU Graz sowohl innerhalb als auch außerhalb der Lehrsäle zu einer wunderbaren und ereignisreichen Zeit geworden ist.

Matthias Seiß

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Diese Masterarbeit ist bei der Infineon Technologies Austria AG im Wintersemester 2011/12 in Zusammenarbeit mit dem Institut für Elektronik der Technischen Universität Graz entstanden.

In diesem Projekt soll eine bestehende Simulationsumgebung rund um ModelSim® so erweitert werden, dass auch die Stromverbräuche des Chips bzw. der einzelnen Module der Netzliste mitsimuliert werden können. Bei Infineon werden Power-Analysen vorwiegend mit der kommerziellen Software PrimeTimePX® von Synopsys® durchgeführt. Diese erfordert aber konstante Stimuli, die vorher aus einer abgeschlossenen Digitalsimulation gewonnen werden müssen. Somit ist es derzeit nicht möglich, die Auswirkungen eines hohen Stromverbrauchs auf die Takterzeugung während einer Simulation entsprechend zu berücksichtigen. Techniken wie Clock Gating (Takte werden ausgeblendet bzw. unterdrückt) können nur unzureichend analysiert werden. Dies ist jedoch ein wichtiger Punkt, um das Hochlaufverhalten des Chips zu simulieren.

Um diesen *Fast Power Simulator (FPS)* umzusetzen ist es erforderlich, die bestehenden funktionalen Modelle der Standardzellen-Bibliothek mit Power-Informationen zu ergänzen und die Modelle zu erweitern. Die Leistungswerte der einzelnen Zellen sind unterschiedlich und sie müssen für jede Zelle einzeln berechnet und in der Verilog-Netzliste hinterlegt werden. Die Berechnung der Leistungsverbräuche erfolgt mit Hilfe der Daten aus dem Synopsys-Libfile der Standardzelle, den Lastkapazitäten der Knoten und den Übergangszeiten an den Eingängen. Diese Berechnungen und das Modifizieren der Netzliste bzw. das Adaptieren der funktionalen Modelle soll durch ein Perl-Skript erfolgen.

Die einzelnen Energiewerte für die Module werden in Form von globalen Variablen bereitgestellt, aufgeschlüsselt in Internal Power, Switching Power und Leakage Power.

1.1 Motivation

Durch die immer weiter voranschreitende Miniaturisierung und den stark steigenden Rechenleistungen von integrierten Schaltungen, wird der Aspekt des Energiemanagements immer wichtiger. Speziell bei mobilen Anwendungen wie z.B. RFID, spielt die genaue Kenntnis über den Energiebedarf eine tragende Rolle. Einerseits um diesen Bedarf im

Falle einer Überbelastung des Chips zu regulieren, andererseits aber auch, um zukünftige Neuentwicklungen hinsichtlich Leistungsverbrauch besser optimieren zu können.

Diese Masterarbeit soll es ermöglichen, bereits während der Digitalsimulation den Energiebedarf einzelner Module genau zu verfolgen und darauf zu reagieren. Durch eine hohe Granularität in Verbindung mit einer sehr guten Genauigkeit, können effiziente Analysen hinsichtlich Stromverbrauch durchgeführt werden. Dadurch, dass die Leistungsaufnahmen des Chips in Form von globalen Variablen bereitgestellt werden, können sowohl die Hardware-Module selbst, als auch eine Testbench von außen auf diese zugreifen und darauf reagieren. Eine geplante Anwendung des FPS ist es, mit einem Reglermodell auf einen eventuellen Einbruch der Versorgungsspannung (engl.: voltage drop) aufgrund eines zu hohen Stromverbrauchs zu reagieren und diesen auszuregeln. Die Detektion eines solchen Einbruchs wird mit dieser integrierten Power-Simulation möglich.

Durch die Aufschlüsselung in Internal und Switching Power wird die Betrachtung noch einmal verfeinert. Damit kann auch die Funktionsweise einzelner Module genau untersucht und optimiert werden.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines funktionierenden Ablaufs, welcher eine Simulation von modulfeinen Leistungsaufnahmen ermöglicht und diese Werte für weitere Analysen bereitstellt.

Folgende Punkte sind umzusetzen:

- Entwicklung eines Tcl-Skripts zum Auslesen der Kapazitäten und Übergangszeiten aus PrimeTime
- Berechnung der einzelnen Energiewerte für jede Zelle, aufgeschlüsselt in Internal, Switching und Leakage Power
- Erweitern der Netzliste um diese Energiewerte
- Modifizieren der funktionalen Verilog-Modelle, um die Event-basierenden Energien zu verarbeiten
- Aufschlüsselung in Clock-Tree / Non-Clock-Tree Zellen
- Vergleich der Simulationsergebnisse mit PrimeTimePX Reports
- Dokumentation / Präsentation

1.3 Gliederung der Arbeit

In Kapitel 1 werden in allgemeinen Worten die Grundzüge der Arbeit beschrieben. Ebenso wird in der Einleitung ein kurzer Überblick über die vorhandenen Rahmenbedingungen gegeben, sowie der Anreiz der Arbeit beschrieben und die zu erreichenden Ziele formuliert.

Kapitel 2 beschäftigt sich mit den theoretischen Grundlagen zu CMOS, mit besonderem Blick auf die Leistungsaufnahme einer CMOS-Zelle. Dazu wird die grundlegende Funktionsweise von MOSFETs erklärt und deren Verwendung in der CMOS-Technologie beschrieben. Weiters werden einige Techniken zur Reduzierung des Energiebedarfs eines integrierten Schaltkreises gezeigt.

In Kapitel 3 wird die Entwicklung der Software beschrieben, welche die Energiewerte berechnet und die Netzliste bzw. die funktionalen Modelle modifiziert. Es wird ein Überblick gegeben, wie die genaue Berechnung der Werte aus dem Synopsys-Libfile funktioniert und wie die dafür benötigten Lastkapazitäten und Übergangszeiten aus PrimeTime ausgelesen werden.

Die Simulationsergebnisse werden in Kapitel 4 behandelt. Hier werden Resultate von ersten Versuchen bis zu vollständigen Chip-Simulationen gezeigt und fortlaufende Vergleiche mit PrimeTimePX und dessen Ergebnisse diskutiert.

Kapitel 5 schließt die Arbeit mit einer Zusammenfassung der entwickelten Lösung und den erzielten Ergebnissen. Dazu wird ein Ausblick gegeben, wie die Arbeit fortgeführt werden kann.

2 Technologische Grundlagen

In diesem Kapitel wird eine Übersicht über die technischen Eigenschaften von Complementary Metal Oxide Semiconductor (CMOS) gegeben, mit Schwerpunkt auf die Leistungsaufnahme einer solchen Zelle.

Um die auftretenden Leistungen in einer CMOS-Zelle zu verstehen, wird zuerst die Funktionsweise eines MOSFETs im Allgemeinen erläutert. Anschließend werden die einzelnen Komponenten, aus denen sich der gesamte Leistungsverbrauch einer Zelle zusammensetzt, diskutiert: Internal Power, Switching Power und Leakage Power. Weiters werden Techniken gezeigt, um den Energiebedarf eines integrierten Schaltkreises zu senken, wie z.B. *Clock Gating*, *Dynamic Voltage Scaling* oder *Dynamic Frequency Scaling*.

2.1 MOSFET

Durch die geringen Herstellungskosten in Verbindung mit einer sehr hohen möglichen Integrationsdichte werden in modernen digitalen Schaltkreisen zu einem Großteil Metall-Oxid Halbleiter Feldeffekttransistoren (MOSFETs) verwendet.

Ein MOSFET verfügt wie ein normaler Feldeffekttransistor über die Anschlüsse *Drain (D)*, *Source (S)* und *Gate (G)*. Hinzu kommt mit *Bulk (B)* noch ein vierter Anschluss, welcher eine Verbindung zum Substrat darstellt. In der Regel wird *Bulk* mit dem *Source*-Anschluss des Transistors verbunden.

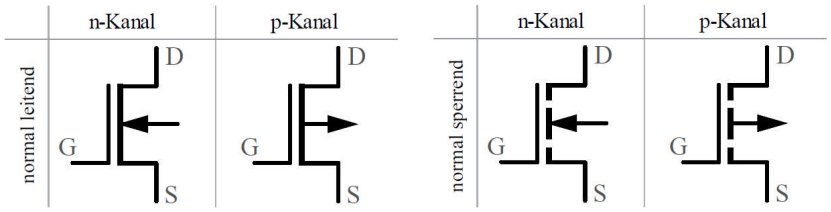

Abbildung 2.1: Schaltsymbole von MOSFET-Transistoren [9]

Grundsätzlich kann man zwischen zwei Typen unterscheiden:

- n-Kanal-MOSFET (NMOS)
- p-Kanal-MOSFET (PMOS)

Diese Unterteilung ist analog zu den bipolaren Transistoren, ein n-Kanal-MOSFET entspricht hierbei einem bipolaren npn-Transistor, ein PMOS einem pnp-Typ. Innerhalb dieser Typen wird auch noch zwischen Anreicherungstyp (selbstsperrend, engl.: enhancement) und Verarmungstyp (selbstleitend, engl.: depletion) unterschieden. In der Regel werden in digitalen Schaltungen Anreicherungstypen verwendet.

Die Funktionsweise eines MOSFETs wird anhand eines selbstsperrenden n-Kanal-MOSFETs erläutert (Abbildung 2.2).

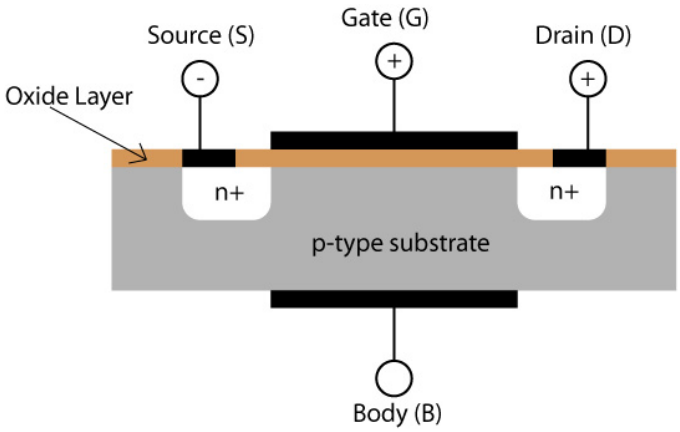

Abbildung 2.2: Struktur eines n-Kanal-MOSFETs [8]

Dieser besteht aus einem schwach p-dotiertem Kristall (Substrat, Bulk, Body), in das zwei stark n-dotierte Bereiche eingebracht sind. Diese Bereiche stellen Drain und Source des Transistors dar. Da zwischen diesen n-dotierten Gebieten das p-dotierte Substrat liegt, entsteht hierbei eine npn-Struktur, ähnlich einem bipolaren npn-Transistor. Über diesem Kanal wird nun eine Elektrode angebracht (Gate), welche mit Hilfe einer dünnen Schicht aus Siliziumdioxid (SiO_2) vom Kanal isoliert wird. Durch diese Isolationsschicht erreicht man, dass vom Gate kein (bzw. nur ein sehr geringer) Strom in den Kanal fließt, da die Steuerung der Drain-Source-Strecke nur über die Spannung am Gate passiert (Gate-Source-Spannung V_{GS}).

Damit gehört der MOSFET zur Gruppe der MISFETs (metal insulator semiconductor FETs) bzw. IGFETs (insulated gate FETs). Liegt keine Spannung am Gate an, kann kein Stromfluss zwischen den Anschlüssen Drain und Source stattfinden, da die n-Bereiche durch das p-dotierte Substrat getrennt sind. Legt man eine positive Spannung zwischen

Gate und Substrat an, entsteht im Substrat ein elektrisches Feld, welches Elektronen aus dem Substrat in Richtung Gate anzieht. Löcher wandern entgegen den Elektronen in Richtung Substrat. Dadurch entsteht mit größer werdender Spannung ein Überschuss an Elektronen unter dem Gate und damit ein n-leitender Kanal zwischen den Drain/Source-Gebieten. Unter dem Gate herrscht *Inversion*, d.h. die Anzahl der Minoritätsträger (bei p-dotiertem Substrat Elektronen) übersteigt jene der Majoritätsträger (Löcher). Wie

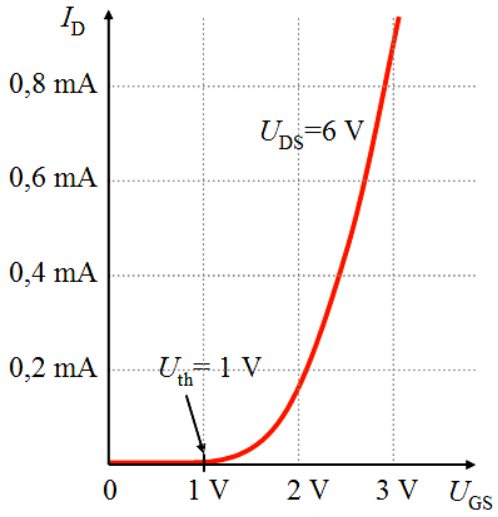

Abbildung 2.3: MOSFET-Eingangskennlinie [5]

gut die Leitfähigkeit des Kanals ist, kann nun mit der Spannung V_{GS} gesteuert werden. Bei Erhöhung der Spannung werden mehr Elektronen in Richtung des Gates gezogen und der Kanal zwischen Drain und Source wird größer und damit leitfähiger. Wenn die Spannung am Gate unter eine bestimmte Schwellenspannung (engl.: threshold voltage V_{th}) fällt, werden nicht mehr genug Elektronen unter dem Gate gesammelt um einen leitenden Kanal auszubilden. Der Transistor befindet sich dann im gesperrten Zustand (Abbildung 2.3).

Das Ausgangskennlinienfeld in Abbildung 2.4 zeigt das Verhältnis von Drain-Source-Spannung V_{DS} zum Strom I_{DS} durch den Kanal bei einer bestimmten Gate-Source-Spannung V_{GS}. In den Abbildungen 2.3 und 2.4 werden die verschiedenen Spannungen mit U bezeichnet.

Die Kennlinie ist nur in einem bestimmten Bereich linear (Widerstandsbereich), darüber hinaus kommt der Effekt der Sättigung zum Tragen. Bei zu hoher Spannung zwischen Drain und Source und der damit einhergehenden Feldstärke kommt es zu einem keilförmigen Abschnüren des Kanals. Damit steigt der Stromfluss bei Erhöhung der Spannung kaum noch an.

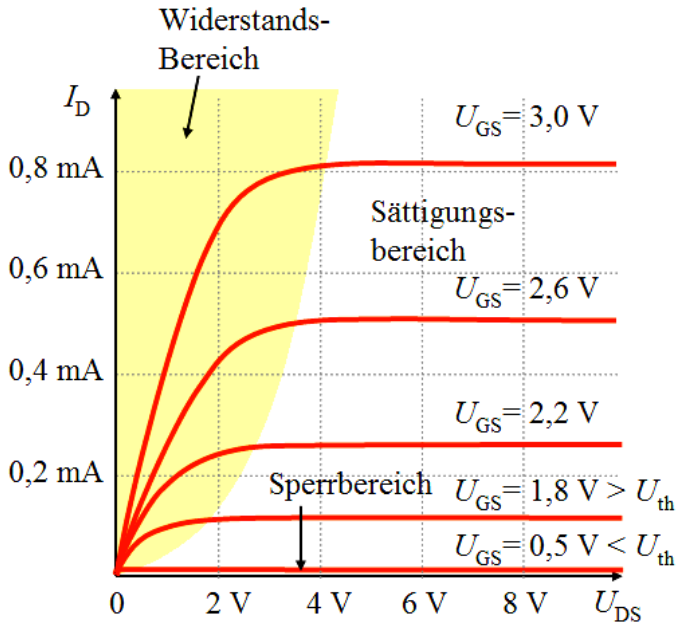

Abbildung 2.4: MOSFET-Ausgangskennlinie [5]

Linearer Bereich:

$$(V_{GS} - V_{th}) > V_{DS} \tag{2.1.1}$$

Sättigungs-Bereich:

$$(V_{GS} - V_{th}) = V_{sat} \leq V_{DS} \tag{2.1.2}$$

Beim p-Type-MOSFET befinden sich - gegenteilig zum NMOS - zwei stark p-dotierte Gebiete in einem schwach n-dotierten Substrat. Um den Kanal zwischen Drain und Source aufzubauen, muss am Gate eine negative Spannung (bezogen auf Substrat) angelegt werden. Dadurch werden Löcher zum Gate hingezogen und es bildet sich ein leitender Kanal zwischen den p-dotierten Bereichen.

2.2 CMOS

Complementary Metal Oxide Semiconductor (CMOS) ist eine Halbleitertechnik, bei der n-Kanal-MOSFETs und p-Kanal-MOSFETs gemeinsam verwendet werden. Die komplementären Transistoren werden speziell zusammengeschaltet und mit der gleichen Steuerspannung angesteuert. Der einfachste Fall eines CMOS-Bausteins ist ein Inverter:

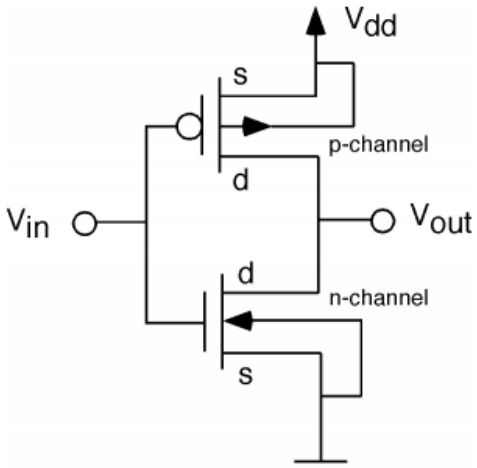

Abbildung 2.5: CMOS Inverter [15]

Im Pull-Up-Pfad, also der Verbindung des Ausgangs zur Spannung V_{DD}, befindet sich ein p-Kanal-MOSFET, im Pull-Down-Pfad (Verbindung zu GND) ein n-Kanal-MOSFET. Bei einer positiven Spannung am Eingang baut der NMOS eine leitende Verbindung zwischen Drain und Source auf und zieht damit V_{out} auf Masse. Durch die positive Spannung am Eingang wird gleichzeitig der PMOS gesperrt. Somit stellt sich am Ausgang der definierte Zustand *LOW* ein. Bei einer negativen Spannung an V_{in} wird hingegen der PMOS leitend und zieht dadurch den Ausgang auf V_{DD}. Der NMOS sperrt, der Ausgang liegt somit auf *HIGH*.

Durch diese Anordnung wird also erreicht, dass immer ein Transistor sperrt, während der andere leitend ist. Es kommt also in der Theorie zu keinem statischen Stromverbrauch, weder beim Ausgangszustand *LOW* (PMOS sperrt, NMOS leitet), noch beim Zustand *HIGH* (PMOS leitet, NMOS sperrt). Lediglich während dem Umschaltmoment, in dem beide Transistoren kurz leitend sind, tritt ein Stromfluss auf.

Die Transistoren, die für die Funktion des Inverters benötigt werden, werden auf dem gleichen Substrat realisiert. Dadurch werden sowohl der NMOS als auch der PMOS den gleichen äußeren Einflüssen wie z.B. der Temperatur ausgesetzt. Der schematische Aufbau eines CMOS-Inverters in n-Wannen-Technik ist in Abbildung 2.6 dargestellt. Das Substrat ist wie bei einem normalen n-Kanal-MOSFET schwach p-dotiert. Darin liegen die n-dotierten Bereiche für die Realisierung des NMOS. Um auch den PMOS am gleichen Träger zu verwenden, wird eine n-dotierte Wanne in das Substrat eingebracht, in der dann die stark p-dotierten Gebiete für die Realisierung des p-Kanal-MOSFETs eingelassen werden.

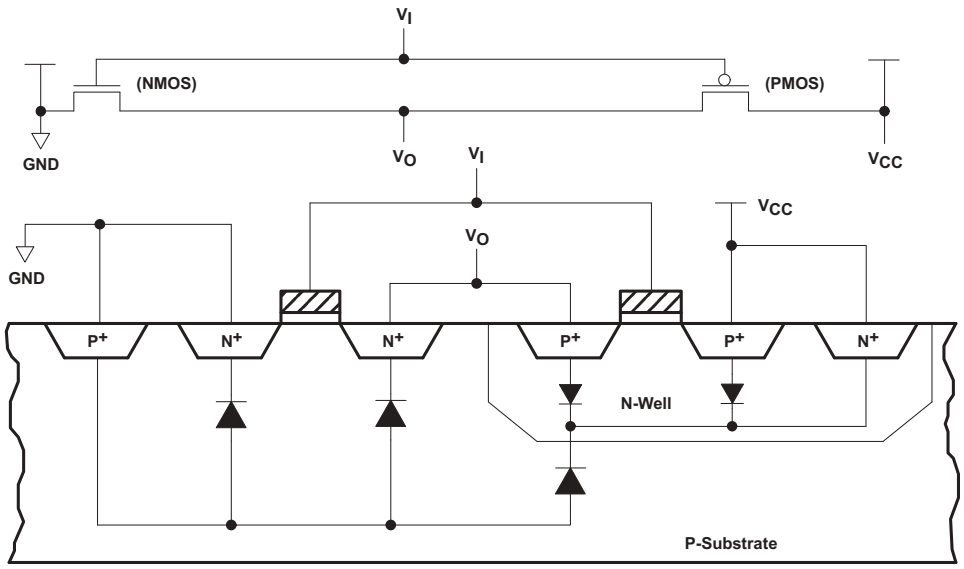

Abbildung 2.6: CMOS Inverter in n-Wannen-Technik [13]

Der umgekehrte Weg, also eine p-Wanne für den NMOS in einem n-dotierten Substrat wird dementsprechend p-Wannen-Technik genannt. Neben diesen beiden Fertigungsmethoden gibt es auch noch die Zwei-Wannen-Technik, bei der sowohl NMOS als auch PMOS in einer eigenen Wanne realisiert werden. Dadurch können die Dotierungen der beiden Transistoren unabhängig voneinander gewählt werden, was eine wesentliche Erhöhung der Flexibilität im Designprozess ermöglicht.

Durch diesen gemeinsamen Aufbau entstehen allerdings auch mehrere parasitäre Dioden zwischen den verschiedenen Dotierungsgebieten, welche einen gewissen Teil zum statischen Leckstrom einer CMOS-Zelle beitragen (siehe Kapitel 2.3.3).

2.3 CMOS-Leistungsverbrauch

2.3.1 Internal Power

Die Leistungsaufnahme einer CMOS-Zelle gliedert sich prinzipiell in zwei Gruppen:

- Dynamischer Verbrauch
- Statischer Verbrauch

Der dynamische Energieverbrauch setzt sich aus den internen Strömen in der Zelle, die während einer Zustandsänderung an den Eingängen auftreten, und dem Umladestrom für die Netzkapazitäten, der bei einer Ausgangsänderung auftritt, zusammen.

$$P_{dyn} = P_{internal} + P_{switching} \tag{2.3.1}$$

Alle diese auftretenden Energien werden anhand eines CMOS-Inverters (Abb. 2.5) erklärt.

Da eine auftretende Eingangsflanke an einer CMOS-Zelle nie ein infinitesimaler Sprung sein kann, sondern immer eine endliche Zeit in Anspruch nimmt, sind für einen kurzen Moment sowohl der PMOS als auch der NMOS leitend. Dadurch entsteht im Umschaltmoment eine direkte Verbindung zwischen V_{DD} und Masse. Es kommt für diesen kurzen Augenblick zu einem Stromfluss I_{SC} (engl.: *short-circuit current*).

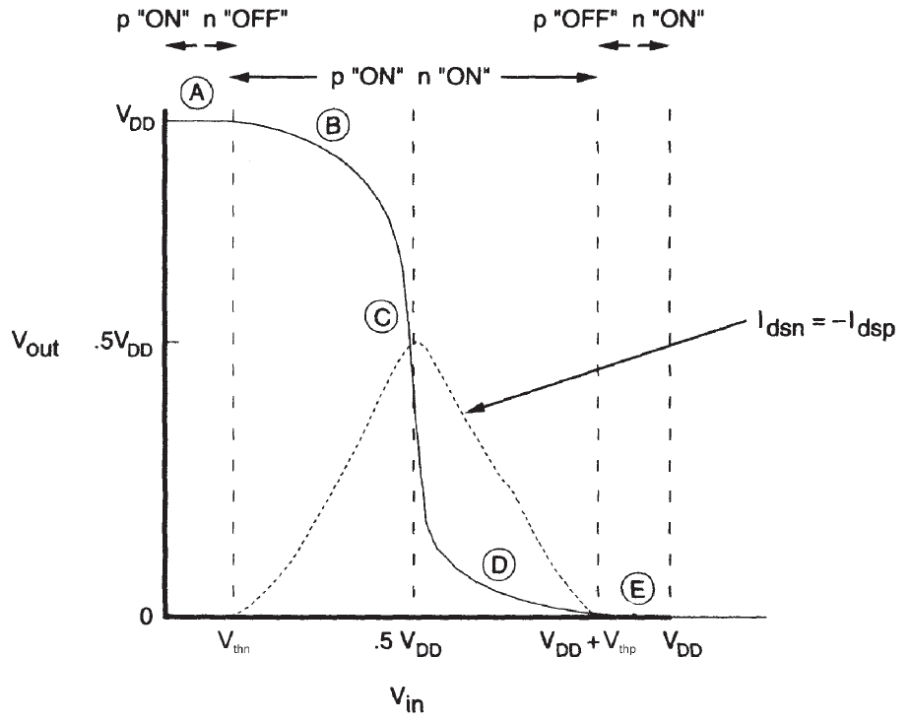

Abbildung 2.7: Strom im Umschaltmoment [14]

Dieser Übergang ist in Abbildung 2.7 dargestellt, eingeteilt in 5 verschiedene Bereiche. Zu Beginn liegt am Eingang des Inverters keine Spannung an, damit leitet der PMOS-Transistor gegen VDD und der NMOS-Transistor sperrt. Am Ausgang liegt das Potenzial V_{DD} (Bereich Ⓐ). Bei Erhöhung der Eingangsspannung fängt neben dem PMOS auch

der NMOS-Transistor an, einen leitenden Kanal aufzubauen. Beide Transistoren sind zu diesem Zeitpunkt also leitend, damit kommt es zum Stromfluss zwischen V_{DD} und Ground (Bereich Ⓑ). Im Punkt Ⓒ sind beide Transistoren gleich leitend, der Strom I_{SC} (in der Abbildung 2.7 mit I_{dsn} bezeichnet) erreicht sein Maximum. Mit der steigenden Spannung verliert der PMOS kontinuierlich seine Leitfähigkeit, damit wird der auftretende Stromfluss wieder gemindert (Bereich Ⓓ). Wenn die Eingangsspannung hoch genug ist, sperrt der PMOS-Transistor komplett gegen V_{DD}, der NMOS-Transistor ist maximal leitend zu Ground. Der Stromfluss I_{SC} verschwindet und der Umschaltvorgang ist abgeschlossen (Bereich Ⓔ). [14]

Um zu ermitteln, wie viel Energie dieser Stromfluss im Umschaltmoment benötigt, wird der Verlauf aus Abbildung 2.7 mit einer Dreiecksfunktion approximiert. Die Energie ist proportional der Fläche unter der Stromkurve, multipliziert mit der Versorgungsspannung V_{DD}. Dadurch ergibt sich für eine steigende Flanke mit der Dauer t_r und dem Strom I_{max}

$$E_{short-circuit} = V_{DD} \cdot \frac{I_{max} \cdot t_r}{2} \tag{2.3.2}$$

Die Energiebedarf erhöht sich also mit steigendem Strom I_{max} und längerer Übergangszeit t_r. Der maximal auftretende Strom I_{max} wird durch den Sättigungsstrom der Transistoren begrenzt. Dieser hängt von der Größe, dem Herstellungsverfahren, der Temperatur und anderen Faktoren ab. Der tatsächlich auftretende Strom hängt stark von dem Verhältnis der Eingangs- zur Ausgangs-Übergangszeit ab. Dieses Verhältnis steht in direktem Zusammenhang mit der Lastkapazität C_L. Diese Kapazität setzt sich im Allgemeinen aus der Leitungskapazität und den parasitären Kapazitäten der MOSFETs (hierbei vor allem die Gate-Kapazität des nachfolgenden Gatters) zusammen. [11]

Um den Einfluss der Lastkapazität auf den internen Strom zu bestimmen, wird die Schaltung eines CMOS-Inverters im Umschaltmoment als Serienschaltung von zwei Widerständen inklusive der Lastkapazität C_L dargestellt (Abbildung 2.9).

Bei einer fallenden Flanke am Eingang sinkt die Leitfähigkeit des NMOS-Transistors gegen Masse, der Widerstand in der Ersatzschaltung vergrößert sich also mit der Zeit t. Dieser Widerstand begrenzt den Strom I_{SC}, dargestellt mit dem blauen Pfeil. Die Größe dieses Stromes hängt nun von der Spannung $v_o(t)$ ab, welche sowohl am Widerstand als auch an der Lastkapazität anliegt. Der Verlauf dieser Spannung im Umschaltmoment ist also definiert durch die Ladekurve der Kapazität am Ausgang.

$$I_{SC}(t) = \frac{v_o(t)}{R\uparrow(t)} = \frac{V_{DD}(1 - e^{\frac{-t}{R\downarrow(t)C}})}{R\uparrow(t)} \tag{2.3.3}$$

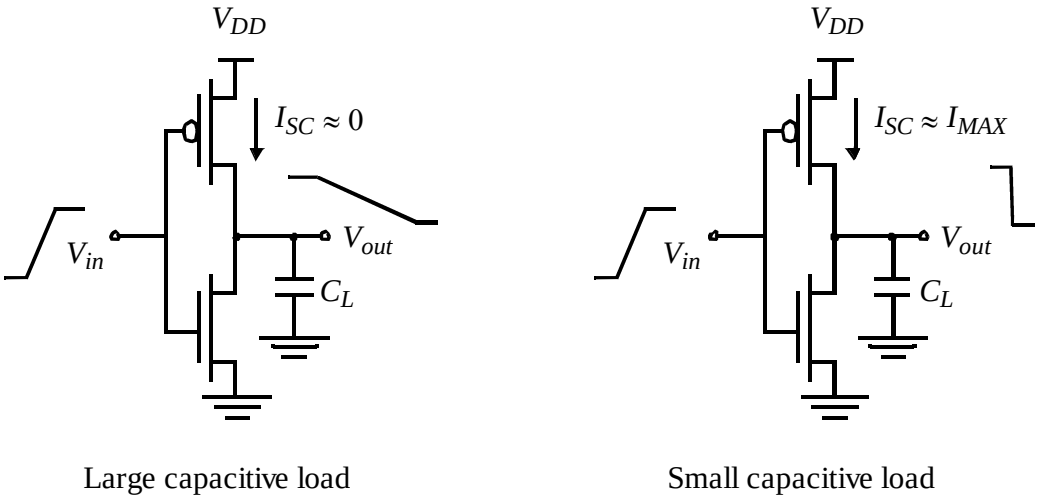

Abbildung 2.8: Ausgangs-Übergangszeiten für große und kleine Lastkapazität [11]

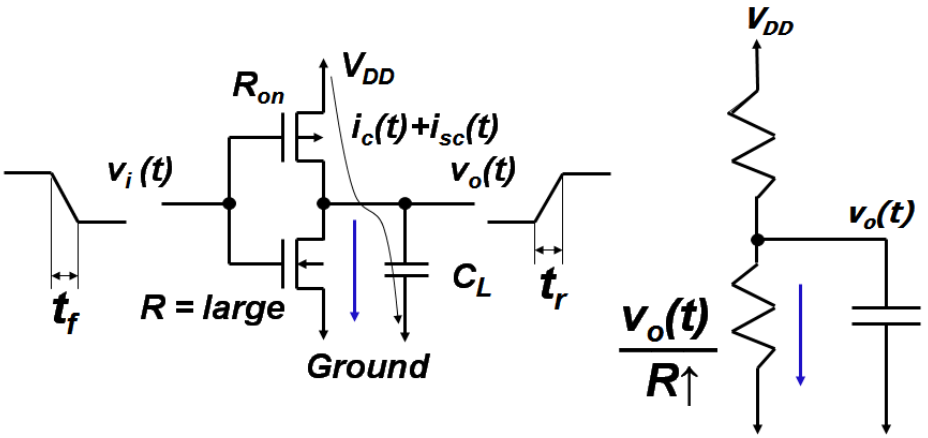

Abbildung 2.9: Ersatzschaltung eines CMOS-Inverters im Umschaltmoment [1]

Wie schnell die Spannung am Ausgang ansteigt hängt nun davon ob, wie groß die Kapazität ist, die es zu laden gilt. Eine kleine Kapazität wird schnell geladen, das bedeutet, dass die Spannung am Ausgang (und damit am Ersatzwiderstand des NMOS) sehr schnell ansteigt. In dieser kurzen Zeit ist allerdings der zeitabhängige Widerstands-Wert noch relativ klein, d.h. durch die große Spannung $v_o(t)$ und den geringen Widerstand R(t) stellt sich ein hoher Strom I_{SC} ein. Die Verläufe dieser Werte ist in Abbildung 2.9 dargestellt.

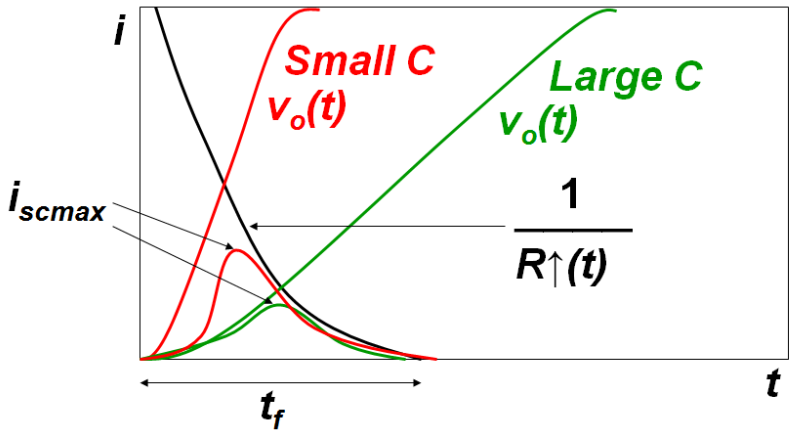

Abbildung 2.10: Verlauf der Ausgangsspannung und des Stromes I_{SC} [1]

Eine große Kapazität wird hingegen relativ langsam geladen, ergo stellt sich eine flachere Spannungskurve am Ausgang ein. Durch die geringere Spannung im Bereich des steigenden Widerstandes am NMOS fließt hier auch ein kleinerer Strom I_{SC}. Sobald der NMOS komplett sperrt (d.h. $R(t) \approx \infty$) wird der Stromfluss abgeschnürt.

Eine kleine Lastkapazität führt also zu größeren internen Verlusten der Zelle. Beispielhaft ist in Abbildung 2.11 der Verlauf des auftretenden Stroms für drei verschiedene Lastkapazitäten dargestellt. Man sieht gut, dass mit steigender Kapazität C_L der maximale Wert des Stroms immer weiter sinkt. Deswegen sollten die Eingangs- und Ausgangs-Übergangszeiten nach Möglichkeit aneinander angepasst werden (*Slope Matching*).

Neben dem Strom I_{SC} kommt es in der Zelle selbst zu Strömen, die benötigt werden, um die internen Kapazitäten zu laden. Diese Ströme sind vor allem bei komplexeren CMOS-Zellen mit vielen Transistoren von Bedeutung.

Die gesamte Internal Power ergibt sich also zu

$$P_{internal} = P_{SC} + P_{C_{int}} \tag{2.3.4}$$

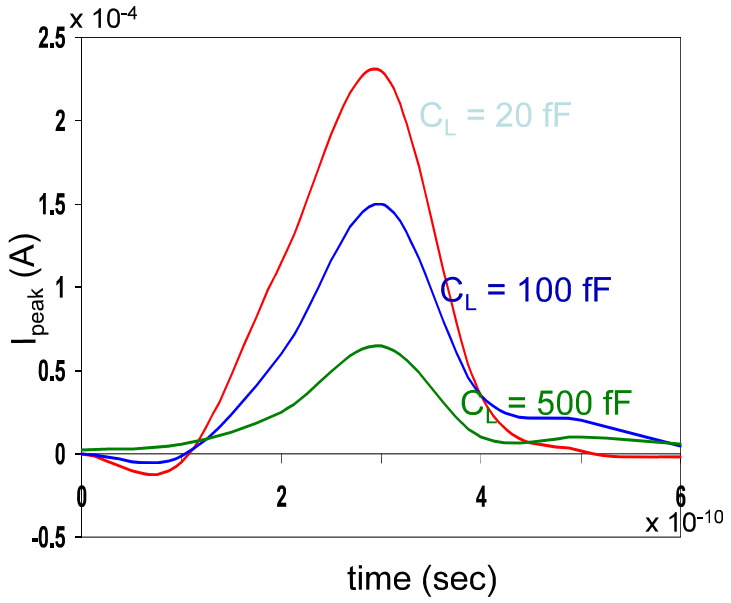

Abbildung 2.11: Strom als Funktion der Lastkapazität [7]

2.3.2 Switching Power

Der zweite große Bereich der dynamischen Leistungsaufnahme ist die Switching Power. Wie bereits in Kapitel 2.3.1 beschrieben, befindet sich an jedem Knoten einer digitalen Schaltung eine Lastkapazität C_L, die sich aus der Leitungskapazität und den parasitären Kapazitäten der MOSFETs zusammensetzt. Bei einer Zustandsänderung an einem Knoten, muss diese Kapazität also entweder auf- oder entladen werden.

Bei einer *steigenden* Flanke am Ausgang, muss die Kapazität auf die Versorgungsspannung V_{DD} (logischer Zustand HIGH) aufgeladen werden. Für diesen Vorgang wird eine bestimmte Menge Energie aus der Versorgung benötigt.

$$v(t) = V_{DD}[1 - e^{\frac{-t}{RC_L}}] \tag{2.3.5}$$

$$i(t) = C\frac{dv(t)}{dt} = \frac{V_{DD}}{R}e^{\frac{-t}{RC_L}} \tag{2.3.6}$$

$$E_{0\to1} = \int_0^\infty V_{DD} \cdot i(t)\, dt = \int_0^\infty \frac{V_{DD}^2}{R}e^{\frac{-t}{RC_L}}\, dt = V_{DD}^2 C_L \tag{2.3.7}$$

Die Hälfte dieser Energie wird beim Ladevorgang im PMOS verbraucht, die andere Hälfte im Kondensator C_L gespeichert.

Bei einer *fallenden* Flanke am Ausgang muss die in der Kapazität gespeicherte Energie wieder entladen werden. Hierbei wird allerdings keine Energie aus der Versorgung benötigt, da das Entladen nur mit Hilfe des gegen Masse geschalteten NMOS passiert. Die gespeicherte Energie im Kondensator wird also verbraucht.

$$E_{1\to 0} = 0 \tag{2.3.8}$$

Ein Energiebedarf aus der Versorgung entsteht somit nur bei einer steigenden Flanke am Ausgang. Daher ist auch das Verhältnis $\alpha_{0\to 1}$ von steigenden Flanken zu der Gesamtzahl aller Übergange in die Berechnung für die Switching Power mit einzubeziehen [3]. Um von der Energie noch auf die Leistung zu kommen, wird die Frequenz f_{clk} in die Formel eingebracht. Damit ergibt sich die Formel für die Switching Power zu

$$P_S = V_{DD}^2 \cdot C_L \cdot f_{clk} \cdot \alpha_{0\to 1} \tag{2.3.9}$$

Wenn der Fokus der Betrachtung der Switching Power nicht im direkten Energiebedarf aus der Versorgung liegt, sondern im tatsächlichen physikalischen Umsetzen der Energie, kann die Formel auch mit dem konstanten Multiplikator 1/2 und einem Aktivitäts-Faktor α angegeben werden:

$$P_S = \frac{V_{DD}^2}{2} \cdot C_L \cdot f_{clk} \cdot \alpha \tag{2.3.10}$$

Der Faktor 1/2 kommt daher, dass bei einer steigenden Flanke am Eingang die Energie $V_{DD}^2 C_L/2$ im PMOS und bei einer fallenden Flanke die gespeicherte Energie im Kondensator (ebenso $V_{DD}^2 C_L/2$) im NMOS-Transistor verbraucht wird. Der Aktivitätsfaktor gibt an, wie oft ein Bereich pro Taktzyklus aktiv ist. Ein Clock-Generator hat z.B. einen Faktor von $\alpha = 1$, da er bei jedem Takt-Zyklus aktiv ist.

2.3.3 Leakage Power

Neben den dynamischen Leistungsaufnahmen gibt es in einer CMOS-Zelle auch einen statischen Leckstrom (engl.: leakage current), der auch ohne Schaltvorgänge an Ein- oder Ausgängen auftritt. Dieser Strom wird großteils durch 3 Effekte bestimmt:

- Sub-threshold leakage
- Leckstrom durch parasitäre Dioden
- Elektronen-Durchtunnelung des Gate-Oxids

Durch den großen Einfluss der Versorgungsspannung V_{DD} auf den Leistungsverbrauch wird diese in digitalen Schaltungen häufig gezielt verringert. Dadurch kommt es aber auch zu einem Performance-Verlust, da die Umladevorgänge der Netzkapazitäten länger dauern (siehe Kapitel 2.4.2 Dynamic Voltage Scaling). Um die effektive Geschwindigkeit des Schaltkreises wieder zu erhöhen, wird oftmals auch die Schwellenspannung V_{th} reduziert. Durch diese geringeren Schwellen kann es allerdings passieren, dass die Transistoren nicht mehr komplett gesperrt werden können, und somit bei einer Spannungsdifferenz zwischen Drain und Source ein Stromfluss stattfinden kann. Dieser ungewollte Verluststrom wird als *Sub-threshold leakage* bezeichnet und kann durch die Formel aus [12] beschrieben werden:

$$I_{sub-threshold} = \mu_0 C_{OX} (\frac{W_{eff}}{L_{eff}}) V_T^2 \cdot e^{1.8} \cdot e^{\frac{(V_{GS}-V_{th})}{nV_T}} \tag{2.3.11}$$

μ_0 ist hierbei die Ladungsträgermobilität, C_{OX} die Gate-Oxid Kapazität pro Einheitsfläche, W_{eff} und L_{eff} die effektive Weite und Länge des Gates, V_T die Temperaturspannung und n ein Technologieparameter.

Wie in Abbildung 2.6 ersichtlich, bilden sich zwischen den verschiedenen Dotierungen im Substrat parasitäre Dioden. Diese Dioden sind zwar in Sperrrichtung geschalten, lassen jedoch trotzdem einen kleinen Leckstrom passieren. Dieser Strom wird mit der Formel aus [10] beschrieben:

$$I_{reverse-bias} = A \cdot J_s \cdot (1 - e^{-\frac{V_{RB}}{nV_T}}) \tag{2.3.12}$$

Hierbei ist A die Fläche des pn-Übergangs, J_s die maximale Sättigungs-Stromdichte, n der Emissions-Koeffizient (üblicherweise n=1), V_{RB} die Spannung in Sperrrichtung und V_T die Temperaturspannung bei der Temperatur T.

In Summe sind diese Dioden-Ströme im Vergleich zu den anderen Komponenten der statischen Stromaufnahme relativ gering. In neueren Fertigungsprozessen wird der Dioden-Leckstrom fast zur Gänze eliminiert.

Da sich in den letzten Jahren der Fertigungsprozess von CMOS immer weiter in Richtung Miniaturisierung entwickelt hat, muss im Bereich der statischen Energieaufnahme auch der Effekt der Durchtunnelung von Elektronen durch das isolierende Gate-Oxid mitberücksichtigt werden. Der Tunneleffekt beschreibt die Wahrscheinlichkeit, mit der ein quantenmechanisches Teilchen eine Potentialbarriere durchbrechen kann, obwohl es das aus Sicht der klassischen Physik eigentlich nicht könnte. Durch diesen Tunneleffekt überwinden Elektronen vom Gate das eigentlich isolierende Siliziumdioxid und dringen in das Substrat ein. Damit entsteht ein Stromfluss $I_{tunneling}$, der von der Dicke der

SiO_2-Schicht abhängig ist und somit bei neueren Technologien ($\leq$ 45 nm) immer mehr an Beachtung finden muss.

Die gesamte statische Stromaufnahme ergibt sich damit zu:

$$I_{static} = I_{sub-threshold} + I_{reverse-bias} + I_{tunneling} \tag{2.3.13}$$

2.4 Methoden zur Reduktion der Leistungsaufnahme

2.4.1 Clock Gating

In den letzten Jahren hat sich Clock Gating zu einer weit verbreiteten Methode, um den Stromverbrauch eines ICs zu senken, entwickelt. Bei dieser Technik wird der Energiebedarf dadurch reduziert, dass bei Überbelastung verschiedene Logikelemente gezielt vom Takt getrennt werden. Dadurch wird die dynamische Stromaufnahme deutlich gesenkt.

Um das Abschalten bzw. das Unterdrücken des Taktes zu ermöglichen, müssen spezielle Zellen und Kontrollsignale in das Design eingearbeitet werden. Es gibt prinzipiell zwei verschiedene Methoden, um ein effektives Clock Gating zu realisieren:

- Latch-Free Clock Gating
- Latch-Based Clock Gating

Bei der Latch-Free Methode wird das Taktsignal nicht direkt mit der nachfolgenden Logik verbunden, sondern es wird vorher mit einem Steuersignal UND-verknüpft. Somit

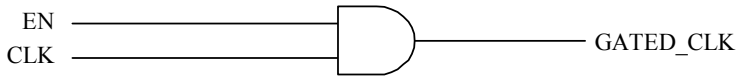

Abbildung 2.12: Latch-Free Clock Gating [6]

kann die nachfolgende Schaltung mit einem einfachen Low-Signal auf der Steuerleitung komplett vom Takt getrennt werden. Wenn statt einem einfachen UND-Gatter andere Logikelemente verwendet werden, können auch komplexere Steuerbefehle mit mehreren Steuerleitungen verarbeitet werden.

Die Latch-Based Clock Gating Methode verwendet hingegen zusätzlich sequentielle Logik, um das Steuersignal während einer Clock-Periode stabil zu halten. Dazu wird das Steuersignal, bevor es zur UND-Verknüpfung mit dem Takt kommt, in ein Latch geschrieben. Dieses Latch wird mit demselben Clock verbunden, der über das UND-Gatter auch die nachfolgende Schaltung versorgt. Damit wird sichergestellt, dass das Steuersignal

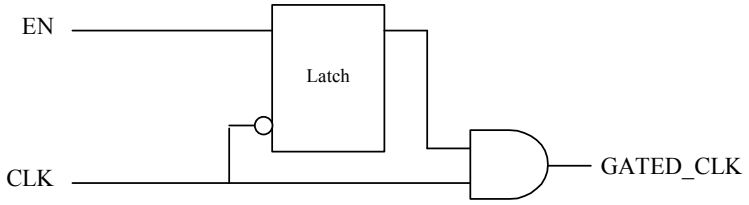

Abbildung 2.13: Latch-Based Clock Gating [6]

während einer Takt-Periode auf einem stabilen Wert bleibt und sich nur gleichzeitig mit einer Flanke des Taktes ändern kann.

2.4.2 Dynamic Voltage Scaling

Eine weitere Methode, um die Leistungsaufnahme eines Schaltkreises zu verringern, ist das Dynamic Voltage Scaling (DVS). Hierbei wird - ebenso wie bei Clock Gating - versucht, die *dynamische* Stromaufnahme zu senken. Als Grundlage dieser Methode dient die Formel für die Switching Power aus [3]:

$$P_S = V_{DD}^2 \cdot C_L \cdot f_{clk} \cdot \alpha_{0\to1} \tag{2.4.1}$$

Dynamic Voltage Scaling nutzt die Tatsache, dass die Versorgungsspannung V_{DD} in dieser Gleichung als quadratischer Faktor vorkommt. Das bedeutet, dass durch eine geringe Senkung der Versorgungspannung eine deutliche Verringerung der Switching Power erreicht werden kann (*Undervolting*).

DVS versucht dabei, örtliche und zeitliche Inhomogenitäten in der Systemauslastung, wie als Beispiel in Bild 2.14 gezeigt, auszunutzen.

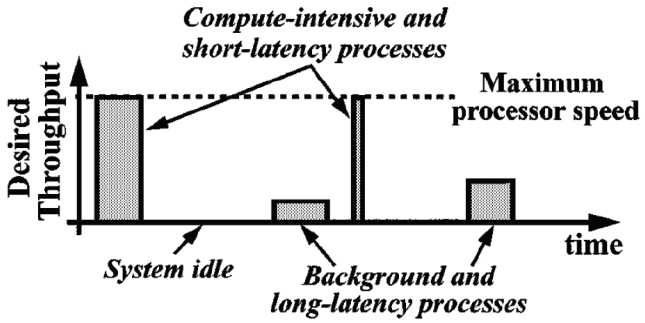

Abbildung 2.14: Beispielhafte Systemauslastung eines Prozessors [2]

In den Phasen, in denen der IC keine Aufgabe (*idle*), oder nur Aufträge mit geringem Ressourcenverbrauch bearbeitet, wird die Versorgungsspannung der betreffenden Module auf einen geringeren Level gesenkt. Bei einer angenommenen Spannungs-Senkung von 10% ergibt sich als Beispiel

$$P_{S100\%} \propto V_{DD}^2 \tag{2.4.2}$$

$$P_{S90\%} \propto (0,9 * V_{DD})^2 \tag{2.4.3}$$

$$Energieersparnis = 1 - \frac{P_{S90\%}}{P_{S100\%}} = 1 - \frac{(0,9 * V_{DD})^2}{V_{DD}^2} = 1 - 0,9^2 = 19\% \tag{2.4.4}$$

D.h. man erreicht eine Energieersparnis von 19% gegenüber dem Betrieb mit normaler Versorgungsspannung. Allerdings steigt mit fallender Versorgungsspannung auch die Zeit, die für einen Umladezyklus der Lastkapazität nötig ist. Damit wird die maximale Schaltgeschwindigkeit der CMOS-Schaltung verringert und es kommt zu einem deutlichen Performance-Verlust in der Rechenleistung (Abbildung 2.15). [4]

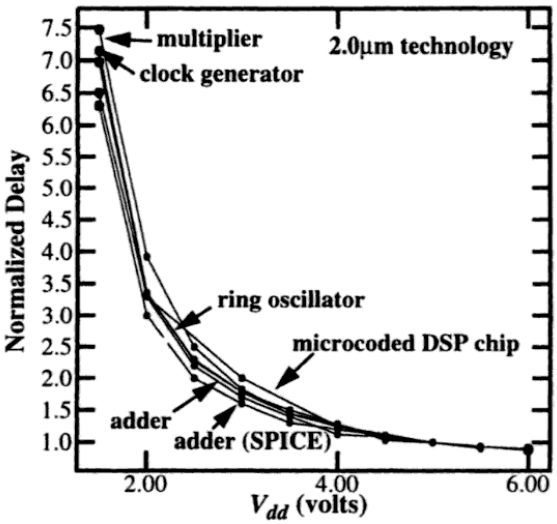

Abbildung 2.15: Verzögerungszeit in Abhängigkeit von V_{DD} [4]

Deshalb muss die Methode des Dynamic Voltage Scalings immer genau an die aktuell geforderte Rechenleistung des Chips angepasst werden. Beim *Overvolting* wird dieses Prinzip in die entgegengesetzte Richtung angewendet. Das bedeutet, dass die Versorgungsspannung über die normale Grenze hinaus erhöht wird, um höhere Taktraten zu ermöglichen (→ *overclocking*).

2.4.3 Dynamic Frequency Scaling

Die Methode des Dynamic Frequency Scalings funktioniert nach einem ähnlichen Prinzip wie Dynamic Voltage Scaling. Auch hier wird versucht, die dynamische Energieaufnahme im Bereich der Switching Power zu verringern. In diesem Fall wird dazu die Taktfrequenz verringert.

$$P_S = V_{DD}^2 \cdot C_L \cdot f_{clk} \cdot \alpha_{0\to1} \tag{2.4.5}$$

Da die Frequenz f_{clk} nicht wie die Spannung V_{DD} mit einem Quadrat in die Gleichung eingeht, ist die Einsparung bei Verringerung des Taktes dementsprechend geringer als bei einer Senkung der Versorgungsspannung.

Durch die gesenkte Frequenz werden weniger Befehle in einem bestimmten Zeitintervall abgearbeitet, ergo ergibt sich auch hier ein Performance-Verlust. Deswegen wird die Methode des Dynamic Frequency Scalings auch nur dann angewendet, wenn die anstehenden Aufgaben dies erlauben, bzw. wenn der IC keine Operationen ausführen muss.

In den meisten Fällen wird diese Methode gemeinsam mit Dynamic Voltage Scaling angewendet, um die höchstmögliche Energieeinsparung zu erzielen.

2.5 Allgemeiner Aufbau eines integrierten Schaltkreises

In modernen Mikrochips wird zum Großteil CMOS-Technologie verwendet. Um ein konsistentes Chip-Design zu erhalten und das Risiko von Fehlfunktionen zu minimieren, werden in vielen Fällen Standardzellen verwendet. Diese Standardzellen sind Logik-Gatter, die in Funktion und Layout bereits vor der eigentlichen Entwicklung eines neuen Chips fertig definiert sind. Standardzellen reichen von einfachen Zellen wie Invertern oder UND-Gattern bis hin zu komplizierter sequentieller Logik wie Flip-Flops mit mehreren Ein- und Ausgängen. Alle Zellen, die das gleiche Layout bzw. die gleiche Fertigungs-Technologie inklusive deren Eigenschaften haben, werden in speziellen Bibliotheken zusammengefasst. In der Regel haben alle Standardzellen einer Bibliothek eine einheitliche Höhe, damit sie im Layout passgenau nebeneinander platziert und leicht miteinander verbunden werden können. Dadurch wird das Risiko von Fehlfunktionen und Störungen in der Schaltung erheblich minimiert, sowie der Entwicklungs- und Produktions-Aufwand eines Chips deutlich gesenkt. Für jede einzelne Zelle gibt es eine Beschreibung in der Bibliothek, in der die genaue Funktionalität, das Timing-Verhalten oder auch eine eventuelle Modellierung des Leistungsverbrauchs definiert sind.

Neben diesen Standardzellen gibt es auch fertige *Makros*, die eine komplexere Funktionalität abdecken, nach außen aber wie eine "Black Box" wirken. Beispiele für solche Makros sind vor allem Speicherelemente wie RAM, ROM oder NVM (*Non-Volatile Memory*, nichtflüchtiger Speicher). Genaue Leistungs-Modelle sind für solche Elemente in der Regel eher schwierig zu definieren, da diese nicht nur von Pin-Zuständen und Kapazitäten bzw. Übergangszeiten abhängen, sondern auch vom Betriebsstatus des entsprechenden Moduls (Read-Phase, Write-Phase, Idle, etc.).

2.6 Digitalsimulation

In der Entwicklung einer integrierten Schaltung gibt es prinzipiell zwei wichtige Phasen bzw. Ebenen einer digitalen Hardware-Simulation. Die erste Simulation erfolgt auf Register-Transfer-Level. Auf dieser Ebene der Abstraktion wird das Verhalten der Schaltung mit Hilfe von Registern und dem Signalfluss zwischen diesen beschrieben und so die Funktion der Schaltung entwickelt. Die Beschreibung erfolgt in einer *Hardware Description Language (HDL)* wie z.B. VHDL (engl.: *Very High Speed Integrated Circuit Hardware Description Language*) oder Verilog.

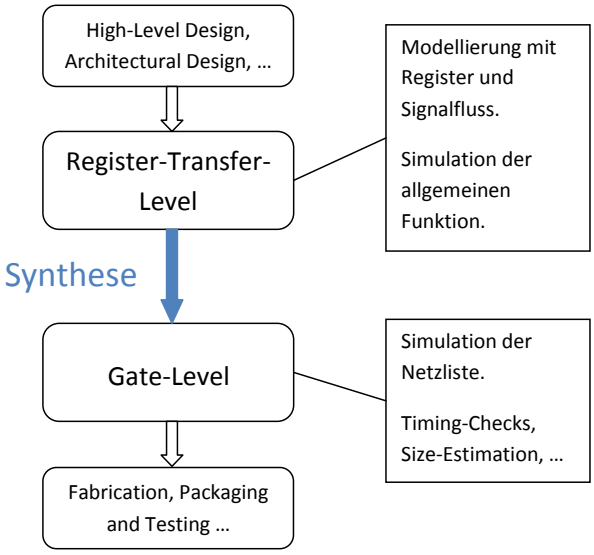

Abbildung 2.16: Phasen in der Digitalsimulation

Wenn die Funktion der Schaltung auf RTL-Ebene beschrieben und verifiziert wurde, wird aus der HDL-Beschreibung eine Netzliste erzeugt. In dieser Netzliste stehen nur mehr einzelne Logik-Gatter, Speicherelemente bzw. vordefinierte Makros (siehe auch

Kapitel 2.5). Dieser Schritt wird *Synthese* genannt, die Schaltung befindet sich nun auf Gate-Level. Diese Netzliste wird wiederum simuliert um die korrekte Funktion der Schaltung auch auf Gate-Level zu überprüfen. Im Gegensatz zur Simulation auf RTL-Ebene werden hier auch die Verzögerungszeiten der einzelnen Gatter sowie die allgemeine Ausbreitungsgeschwindigkeit der Signale berücksichtigt.

Um das funktionale Verhalten der einzelnen Gatter zu beschreiben, brauchen alle Zellen Modelle, welche die Informationen über die genaue Funktionalität und andere Parameter für die Simulation bereitstellen. Diese Modelle werden vor der eigentlichen Simulation kompiliert und zusammen mit der Netzliste und anderen Dateien in die Simulationsumgebung mit eingebunden. Im vorliegenden Design-Flow sind sowohl die Netzliste als auch die funktionalen Modelle in Verilog geschrieben. Da in der Netzliste auf Gate-Level nur mehr die einzelnen Standardzellen mit deren Ein- und Ausgängen stehen, wird hier mit der Entwicklung der Simulation des Leistungsverbrauchs angesetzt. Dazu werden sowohl die Netzliste als auch die erwähnten funktionalen Verilog-Modelle der Zellen mit Power-Informationen adaptiert.

3 Entwicklung der Power Simulation

In diesem Kapitel wird der Ansatz der Lösung beschrieben, sowie die Umsetzung und Implementierung der Software erläutert. Um den Ansatz zu verstehen, wird ein Überblick über den Ablauf einer konventionellen, digitalen Power-Simulation beschrieben. Die entwickelte Lösung wird in den folgenden Kapiteln als *Fast Power Simulator (FPS)* bezeichnet.

3.1 Konventionelle Leistungs-Abschätzung

Eine Abschätzung der genauen Stromaufnahme eines integrierten Schaltkreises ist im Allgemeinen nur über Umwege, d.h. mit Hilfe spezieller Power-Analysis-Tools möglich. Dazu muss zuerst eine normale Digitalsimulation auf Gate-Level-Ebene durchgeführt werden. Erst danach kann mit Hilfe von speziellen Programmen der Leistungsverbrauch für die vorangegangene Simulation ermittelt werden. Beispielhaft wird hier der Ablauf für die kommerzielle Software PrimeTimePX® beschrieben, welcher aber für die meisten Power-Analysis-Tools ähnlich ist. PrimeTime*PX* ist eine Erweiterung von PrimeTime, mit der neben den Standard-Funktionalitäten wie Timing-Checks oder Parasitäten-Bestimmung auch verschiedene Power-Analysen möglich sind. PrimeTimePX wird bei Infineon als Standard für Leistungs-Abschätzungen genutzt.

Für eine vollständige Analyse der Leistungsaufnahme braucht man im Wesentlichen vier Bestandteile (Abbildung 3.1):

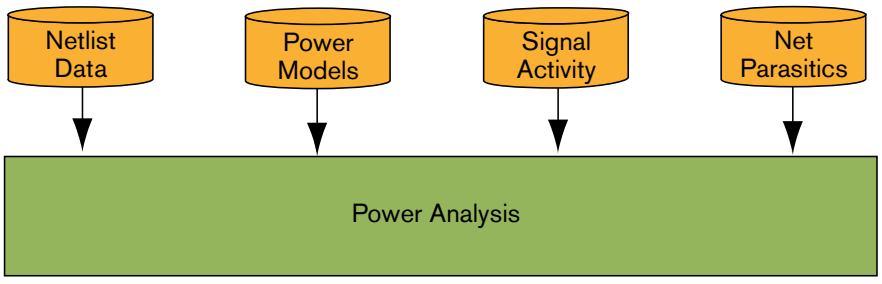

Abbildung 3.1: Bestandteile einer Power-Analyse [7]

- Netlist Data

Der wichtigste Teil der Leistungsabschätzung ist die Netzliste des Designs. In ihr stehen alle instanzierten Zellen mit ihren jeweiligen Zelltypen und deren Verbindung untereinander.

```
SLL10L100_USBUFX120 I0004 (.A ( net0024 ) , .Z ( net0010 ) ) ;
SLL10L100_UBSIVX120 I0003 (.A ( net0021 ) , .Z ( net0013 ) ) ;
SLL10L100_USLNDX020 I0012 (.B ( net0026 ) , .Z ( net0029 ) , .A ( net0025 ) ) ;
SLL10L100_USAN2X040 I0016 (.B ( net0017 ) , .Z ( net0020 ) , .A ( net0025 ) ) ;
```

Abbildung 3.2: Auszug aus einer Netzliste

- Power Models

Um eine genaue Abschätzung des Leistungsverbrauchs zu ermöglichen, braucht man für jede Art von Standardzelle eine Beschreibung der möglichen Leistungswerte. Dazu gehört neben der Leakage Power vor allem die Internal Power, welche für jede mögliche Pin-Stellung und für jeden möglichen Signalübergang an der Zelle definiert sein muss. Diese Power Models sind in den Bibliotheken der Zelltypen modelliert (siehe Kapitel 3.4.2). Für die Switching Power wird kein Modell benötigt, sie wird mit Hilfe der Netzkapazitäten, der Spannung und der Frequenz berechnet. Für Makros, also fertig zusammengestellte Module wie z.B. Speicher (RAM, ROM) müssen eigene allgemeine Power-Models definiert sein, die PrimeTimePX zur Berechnung der Energie heranzieht. Sollten keine Modelle existieren, wird eine grobe Schätzung des Leistungsverbrauchs durchgeführt.

- Signal Activity

Die Aktivität einer Zelle beeinflusst vor allem die *dynamische* Leistungsaufnahme enorm. Deshalb ist es nötig, Informationen über auftretende Signalübergänge an den Pins der Standardzellen bereitzustellen, um eine möglichst genaue Abschätzung der Leistung vornehmen zu können. Dazu werden aus der vorangegangenen Digitalsimulation die Aktivitäten der Zellen aus der Netzliste aufgezeichnet und in einer Datenstruktur abgelegt. Dafür gibt es zwei verschiedene Formate:

1. VCD (Value Change Dump)
2. SAIF (Switching Activity Interchange Format)

In einer VCD-Datei werden alle Änderungen einer Leitung zu einem bestimmten Zeitpunkt aufgezeichnet. Es gibt also für jede auftretende Flanke einen eigenen Eintrag in der Datei, der sowohl den Namen der Leitung als auch den genauen Zeitpunkt des Wechsels speichert. Dadurch hat die VCD-Methode den Vorteil einer sehr großen Genauigkeit bzw. einer feinen Granularität. Allerdings können hierbei für längere Zeitintervalle sehr schnell enorm große Dateien entstehen, was die Handhabung dieser wesentlich beeinträchtigt. Der Aufbau einer VCD-Datei ist in Abbildung 3.3 dargestellt.

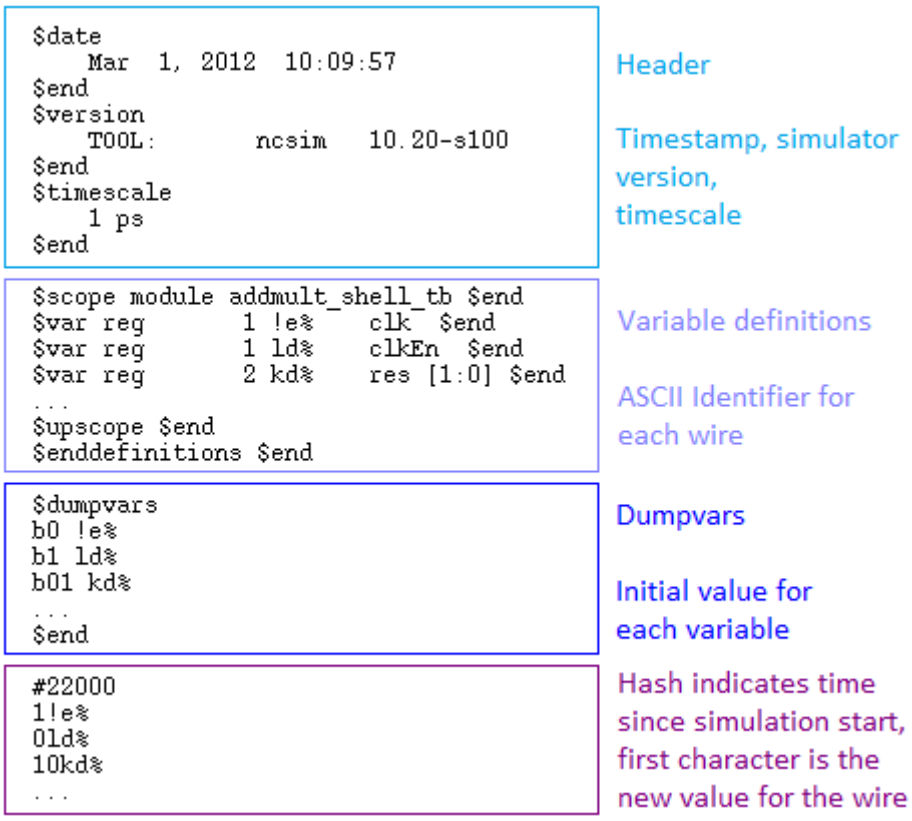

Abbildung 3.3: Aufbau einer VCD-Datei

Bei der SAIF-Methode werden nicht alle 0/1-Wechsel einzeln protokolliert, sondern über ein gewisses Zeitintervall aufsummiert. Man hat also die Anzahl von steigenden und fallenden Flanken in einem bestimmten Zeitfenster, nicht aber die genauen Zeitpunkte, wann diese aufgetreten sind. Dadurch ist die Größe einer SAIF-Datei im Allgemeinen sehr viel kleiner als die einer VCD-Datei, bietet allerdings nicht deren Genauigkeit.

Für eine Abschätzung des durchschnittlichen Stromverbrauchs eines Chips in einem zeitlich festgelegten Bereich, ist die Methode mit SAIF ausreichend. Für eine genauere und zeitlich höher aufgelöste Analyse braucht man allerdings ein VCD-file, welches alle einzelnen Übergänge inklusive deren Zeitpunkte protokolliert.

Innerhalb von PrimeTimePX kann mit dem Befehl *read_vcd* eine VCD-Datei, mit *read_saif* eine SAIF-Datei eingelesen werden. Welcher Zeitabschnitt analysiert werden soll, wird dem read-Befehl als Parameter *-time* übergeben.

```
read_vcd /home/userx/design_sim.vcd -time {2000 3000}
```

Die Zeit wird hierbei in 10^{-9} s angegeben. Der obere Befehl beschreibt also ein Zeitfenster von 2 µs bis 3 µs.

- Net Parasitics

Hierin stehen die Kapazitäten, die Widerstände und die Induktivitäten der Eingangs- und Ausgangspins aller Zellen. Mit diesen Daten können die Kapazitäten der Knoten, sowie die Verzögerungszeiten bei Signalübergängen berechnet werden. Diese Daten sind wichtig, um die Internal und Switching Power zu berechnen.

Aus diesen vier Teilen erstellt PrimeTimePX dann entweder eine Durchschnitts-Leistungs Abschätzung (*Average Power Report*), oder eine zeitlich aufgelöste Analyse, die auch als grafischer Verlauf ausgegeben werden kann (*Time-Based Power Report*). Für diese Reports werden aus den Signal-Activity-Files alle Übergänge, welche in dem zu analysierenden Zeitfenster aufgetreten sind, bestimmt und der entsprechenden Zelle zugeordnet. Bei jeder dieser Flanken tritt in der Schaltung ein gewisser Energiebedarf auf, dessen Größe abhängig von den Werten aus den Net Parasitics ist (siehe Kapitel 2.3). Alle Energien, die bei einem bestimmten Zustandswechsel auftreten, werden berechnet und aufsummiert bzw. gespeichert. Mit diesen Daten kann PrimeTimePX anschließend eine komplette Leistungs-Abschätzung für die durchgeführte Simulation erstellen.

Mit dem Befehl *report_power* wird ein Report für den Gesamtverbrauch des Chips ausgegeben. Neben der Aufteilung in Internal, Switching und Leakage werden die Leistungen noch in verschiedene Bereiche wie z.B. Register, kombinatorische Zellen oder sequentielle Zellen aufgeschlüsselt (Abbildung 3.4).

```
****************************************
Report : Averaged Power
Design :
Version: D-2010.06-SP3-2
Date   : Mon Apr 23 13:50:02 2012
****************************************

  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000 5.110e-05 1.200e-04 1.711e-04 ( 2.85%)
clock_network           2.819e-03 1.485e-03 6.171e-06 4.311e-03 (71.76%)  i
register                9.694e-05 3.936e-05 1.849e-06 1.382e-04 ( 2.30%)
combinational           3.268e-04 9.649e-04 8.260e-05 1.374e-03 (22.88%)
sequential              1.137e-05 1.613e-06 6.481e-08 1.305e-05 ( 0.22%)

  Net Switching Power  = 2.542e-03   (42.32%)
  Cell Internal Power  = 3.254e-03   (54.17%)
  Cell Leakage Power   = 2.107e-04   ( 3.51%)
                         ---------
Total Power            = 6.007e-03  (100.00%)
```

Abbildung 3.4: PrimeTimePX-Report der Leistungsaufnahme eines Chips

Neben dem gesamten Chip können auch einzelne Module oder Zellen aufgelistet werden (Abbildung 3.5). Dazu kann man entweder das komplette Design bis zu einer angegebenen Hierarchie-Tiefe auflisten (Parameter *-hier* mit bestimmter Tiefe *-level*), oder die gewünschten Elemente direkt angeben. Auch hier wird die Energie in die drei Bereiche Internal, Switching und Leakage aufgeteilt. In Abbildung 3.5 wird beispielhaft eine hierarchische Aufschlüsselung bis in die zweite Ebene gezeigt.

```
report_power -hier -level 2
```

```
****************************************
Report : Averaged Power
        -hierarchy
        -levels 2
Design :
Version: D-2010.06-SP3-2
Date   : Mon Apr 23 13:51:00 2012
****************************************
                                    Switch   Int      Leak      Total
Hierarchy                           Power    Power    Power     Power    %
------------------------------------------------------------------------------
 gact_prod                          2.54e-03 3.25e-03 2.11e-04  6.01e-03 100.0
  sh_clk_gate_pre_n (sh_clk_gate_pre_0_1_1_11)
                                    1.98e-05 3.18e-05 1.41e-07  5.18e-05   0.9
    clk_gate (tc_clk_gate_pre_11)   1.98e-05 3.18e-05 1.41e-07  5.18e-05   0.9
  gact_inst (2_20)                  2.39e-03 3.08e-03 2.09e-04  5.68e-03  94.5
    gact_sysctrl_inst (core9_sysctrl)
                                    4.30e-06 9.38e-06 8.47e-08  1.38e-05   0.2
    gac_inst (c_4_13_32)            1.07e-05 2.43e-05 1.59e-07  3.51e-05   0.6
    fmi_inst (fmi)                  1.01e-03 1.12e-03 1.44e-04  2.27e-03  37.9
    arm_core_inst (arm_wrapper)     1.33e-03 1.92e-03 6.37e-05  3.31e-03  55.2
    ppb_ict_inst (ppb_ict_2_20_0_1_0)
                                    3.47e-09 2.41e-09 1.99e-07  2.05e-07   0.0
  sh_clk_gate_pre_p (sh_clk_gate_pre_0_1_1_12)
                                    6.13e-05 4.72e-05 2.20e-07  1.09e-04   1.8
    clk_gate (tc_clk_gate_pre_12)   6.13e-05 4.72e-05 2.20e-07  1.09e-04   1.8
```

Abbildung 3.5: Aufschlüsselung der Module bis in die zweite Ebene

Die Einheit aller Werte ist Watt. Diese Reports werden im Laufe der Entwicklung des Fast Power Simulators für verschiedene Simulationen immer wieder als Referenz bzw. Vergleich herangezogen. Um dabei die maximale Leistung eines Chips zu simulieren, werden spezielle Benchmark-Simulationen angewendet. Oft verwendet wird hier der Dhrystone-Test, welcher eine gute Abschätzung des Leistungsverbrauchs liefert.

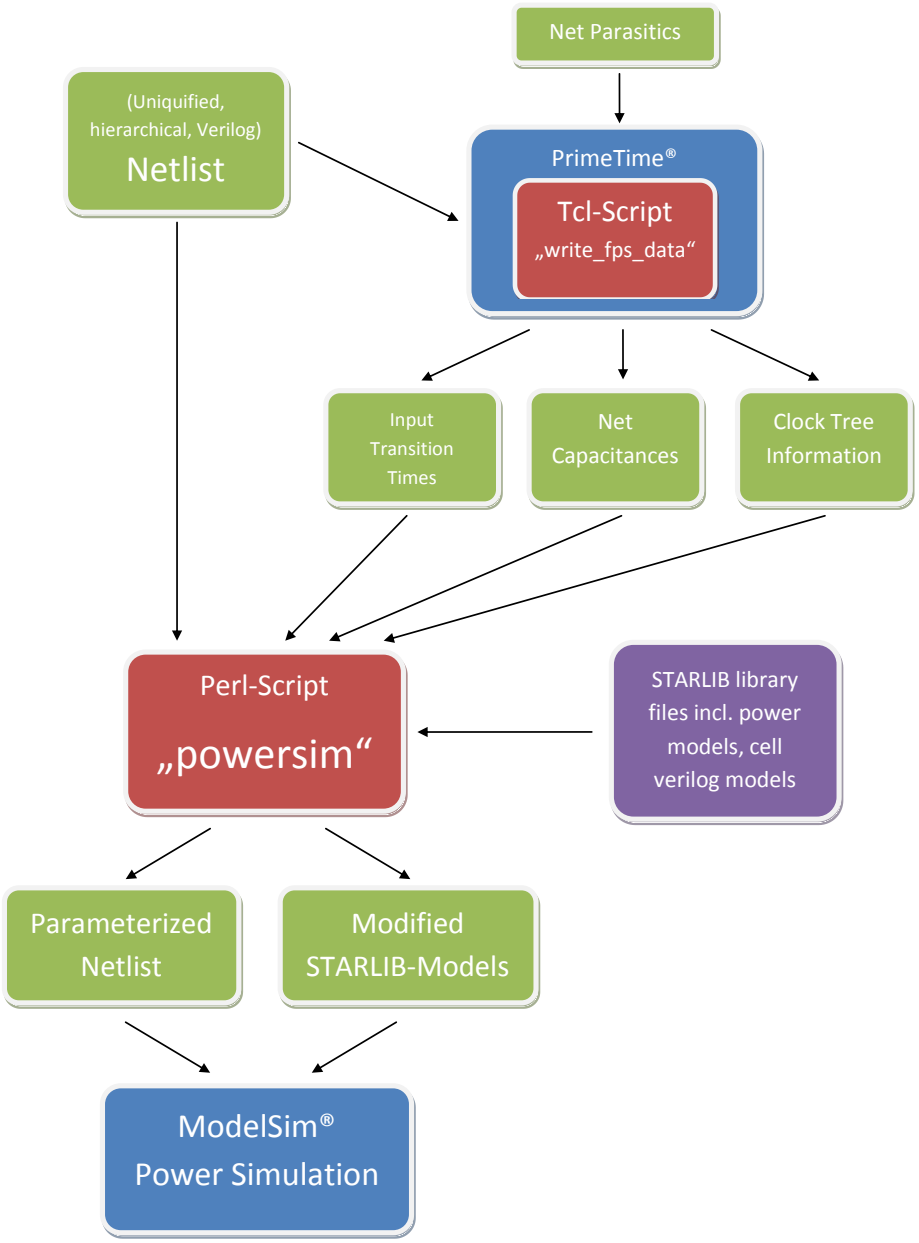

Abbildung 3.6: Flowchart FPS

3.2 Konzept der neuen Lösung

Der Ansatz dieser Arbeit besteht nun darin, bereits während der Digitalsimulation den aktuellen Leistungsverbrauch mitzusimulieren. Das Prinzip ist, bei jeder auftretenden Flanke in der Digitalsimulation einen bestimmten Wert aufzusummieren, welche der Energie entspricht, die an dieser Zelle für diesen Übergang aufgebracht werden muss. Wie in Kapitel 2 beschrieben, hängen diese Energien vor allem von den Lastkapazitäten der Knoten (relevant für Internal und Switching Power) und der Dauer der Signalübergänge (Internal Power) ab. Dazu kommt noch der Einfluss der Versorgungsspannung, sowie bei einer Leistungsbetrachtung noch jener der Taktfrequenz.

Um alle auftretenden Flanken an einer Zelle abzudecken, müssen bereits vor der Simulation alle möglichen Energiewerte, die auftreten könnten, berechnet und der Simulation übergeben werden. Abhängig von der Pin-Stellung und Art der auftretenden Flanke (steigende oder fallende Flanke) wird dann ein bestimmter Energiewert ausgewählt und aufsummiert. Die Werte werden von einem Perl-Skript im Vorhinein berechnet und in die Netzliste als Parameter eingefügt. Um diese dann während der Simulation zu verarbeiten, werden auch die funktionalen Verilog-Modelle der Zellen adaptiert und mit den Parametern und diversen Abfragen ergänzt. Mit dieser adaptierten Netzliste und den modifizierten funktionalen Modellen wird die Simulation durchgeführt. Dieser Ablauf ist in Abbildung 3.6 grafisch beschrieben.

3.3 Ermittlung der Kapazitäten und Übergangszeiten

Bevor mit der Berechnung der Energiewerte und dem Modifizieren der Netzliste und der funktionalen Modelle begonnen werden kann, müssen zuerst alle Kapazitäten der Knoten, sowie die Eingangs-Übergangszeiten der einzelnen Gatter der Netzliste bestimmt werden. Dies passiert mit Hilfe von PrimeTime, welches diese Werte aus den Parasitic-Files und den Modellen der Leitungsverbindungen berechnet. Um diese Werte auszulesen und zu verwenden, wird ein Tcl-Skript geschrieben, welches in PrimeTime ausgeführt werden kann.

Die Kapazität an einem Knoten der Netzliste setzt sich aus der back-annotierten Leitungskapazität und der Summe der Pin-Kapazitäten aller angeschlossenen Pins zusammen. Die Kapazität an einem Pin einer Zelle ist für steigende und fallende Flanken leicht unterschiedlich, deshalb müssen jeweils 2 Kapazitätswerte ermittelt werden. Die PrimeTime-Attribute *ba_capacitance_max* und *pin_capacitance_max_rise* bzw. *pin_capacitance_max_fall* repräsentieren diese Werte.

Listing 3.1: Tcl-Skript zum Auslesen und Berechnen der Kapazitäten

```
foreach_in_collection one_cell $cell_list {
    set outputs [filter_collection
        [get_pins -of_objects $one_cell] "direction==out"]
    foreach_in_collection output $outputs {
        set net [get_nets -quiet -of_objects $output]
        set cap_ba [get_attribute $net ba_capacitance_max]
        set cap_pin_rise [get_attribute $net pin_capacitance_max_rise]
        set cap_pin_fall [get_attribute $net pin_capacitance_max_fall]
        set cap_rise [expr "$cap_pin_rise+$cap_ba"]
        set cap_fall [expr "$cap_pin_fall+$cap_ba"]
    }
}
```

Die Eingangs-Übergangszeiten der Zellen in der Netzliste stehen in den Attributen *actual_rise_transition_max* für steigende und *actual_fall_transition_max* für fallende Flanken.

Listing 3.2: Tcl-Skript zum Auslesen der Eingangs-Übergangszeiten

```
foreach_in_collection one_cell $cell_list {
    set inputs [filter_collection
        [get_pins -of_objects $one_cell] "direction==in"]
    foreach_in_collection input $inputs {
        set trans_rise [get_attribute
            [get_pins $input] actual_rise_transition_max]
        set trans_fall [get_attribute
            [get_pins $input] actual_fall_transition_max]
    }
}
```

Die Kapazitäten und die Übergangszeiten werden in zwei verschiedene Ausgabedateien *caps.txt* und *trans.txt* geschrieben. Beide enthalten jeweils den vollen Hierarchie-Namen zur vollständigen Identifizierung der Zelle sowie die jeweiligen Werte mit der Unterscheidung zwischen steigender (RISE) und fallender (FALL) Flanke (Abbildung 3.7).

In die Dateien für die Kapazitäten und Übergangszeiten wird noch ein Header geschrieben, in dem die Bedingungen, für die die Werte generiert wurden, aufgelistet werden. Jeder dieser *Corner* wird durch Temperatur, Versorgungsspannung und Fertigungsprozess der Transistoren beschrieben. Der angegebenen Corner *max* im Auszug aus Abbildung 3.7 hat also eine Spannung von 1,62 V, eine Temperatur von 125°C und einen Transistor-Fertigungsfaktor von 1,4.

Für spätere Analysen wird auch noch eine dritte Datei *clock_tree.txt* generiert, in der gespeichert wird, ob sich eine Zelle im *Clock Tree* (bzw. allgemeiner: *Clock Distribution Network*) befindet, oder nicht. Dadurch kann genau ermittelt werden, wie viel Leistung bei der Generierung und Verteilung der einzelnen Taktsignale im Vergleich zur restlichen Logik verbraucht wird. Diese Unterscheidung, ob es sich um eine solche Zelle handelt

```
// ****************************************************************
// Conditions:
//
// operating_condition_name          max
// process                           1.4
// temperature                       125
// voltage                           1.62
// ****************************************************************

 i_pad_frame/icc_place7/A 0.330675 (RISE)
 i_pad_frame/icc_place7/A 0.205734 (FALL)
 i_pad_frame/CTS_PSYN3/A 0.541674 (RISE)
 i_pad_frame/CTS_PSYN3/A 0.351828 (FALL)
 ...
```

Abbildung 3.7: Auszug aus der Transitions-Datei

oder nicht, steht in dem boolschen Attribut *is_clock_network* und wird ähnlich wie die Übergangszeit bzw. die Kapazität ausgelesen.

Mit diesen Daten und der Netzliste kann nun mit der Berechnung der Energiewerte begonnen werden.

3.4 Ablauf des Skripts

3.4.1 Verarbeiten der Netzliste

Für den Fast Power Simulator wird eine hierarchische Netzliste benötigt, welche von einem Perl-Skript eingelesen und sequentiell verarbeitet wird. Ein solcher Durchlauf kann für verschiedene Corner durchgeführt werden. Die wichtigsten davon sind:

- "nom" : Nominal, Temp.: 27°C, Spannung: 1,35 V, Fertigungsprozess: Typical (1,0)
- "max" : Maximum, Temp.: 125°C, Spannung: 1,15 V, Fertigungsprozess: Slow (1,4)
- "min" : Minimum, Temp.: -40°C, Spannung: 1,5 V, Fertigungsprozess: Fast (0,7)

Für jeden Corner sind eigene Libfiles definiert, welche die jeweiligen Informationen über den Energieverbrauch bei den definierten Bedingungen beinhalten. Neue Corner können bei Bedarf leicht hinzugefügt werden, jedoch müssen für jeden neuen Corner auch entsprechende Libfiles zur Verfügung gestellt werden. Für welchen dieser Eckpunkte die Berechnung durchgeführt werden soll, muss dem Skript zu Beginn mit dem Parameter *-corner* übergeben werden (e.g. *powersim -corner max*).

Sollten alle erforderlichen Daten vorhanden sein, kann mit dem Berechnen der Werte und dem Patchen der Netzliste begonnen werden. Die genauen Dateinamen von Input und

Output-Files werden in Abbildung 3.8 aufgelistet. Alle Pfade zu diesen Dateien werden am Anfang des Skripts definiert.

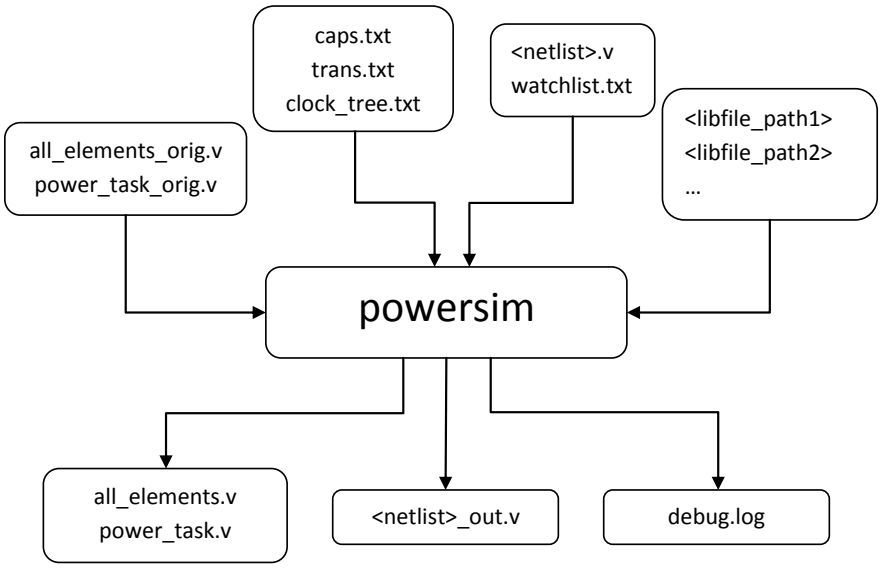

Abbildung 3.8: Input- und Output-Dateien des "powersim"-Skripts

Die Netzliste wird zu Beginn des Skripts eingelesen und in einem Array gespeichert, welches dann zeilenweise nach Standardzellen durchsucht wird. Diese Zellen haben einen bestimmten Prefix, mit denen sie von anderen Modulen bzw. Makros unterschieden werden können. Bei der zu simulierenden Technologie (l90) ist dieser Prefix "LL10".

Wenn das Skript auf eine bekannte Standardzelle trifft, muss zuerst der vollständige Hierarchie-Pfad des Gatters ermittelt werden. Nur mit diesem Pfad kann die Zelle eindeutig definiert werden, um anschließend die entsprechenden Kapazitäts- und Übergangswerte zu bestimmen.

Eine beispielhafte schematische Darstellung einer hierarchischen Netzliste ist in Abbildung 3.9 dargestellt. Der komplette Pfad für die Standardzelle LL10_X3 wäre hier beispielsweise "module_a/module_d/LL10_X3". Für die Zelle auf oberste Ebene ist der Pfad gleich dem Instanznamen dieser, also "LL10_X1".

Um diese Information nun aus der Netzliste herauszulesen, wird eine Sub-Funktion *search_module* aufgerufen, welche als Parameter die Zeilennummer der gefundenen Zelle in der Netzliste erhält (Abbildung 3.10).

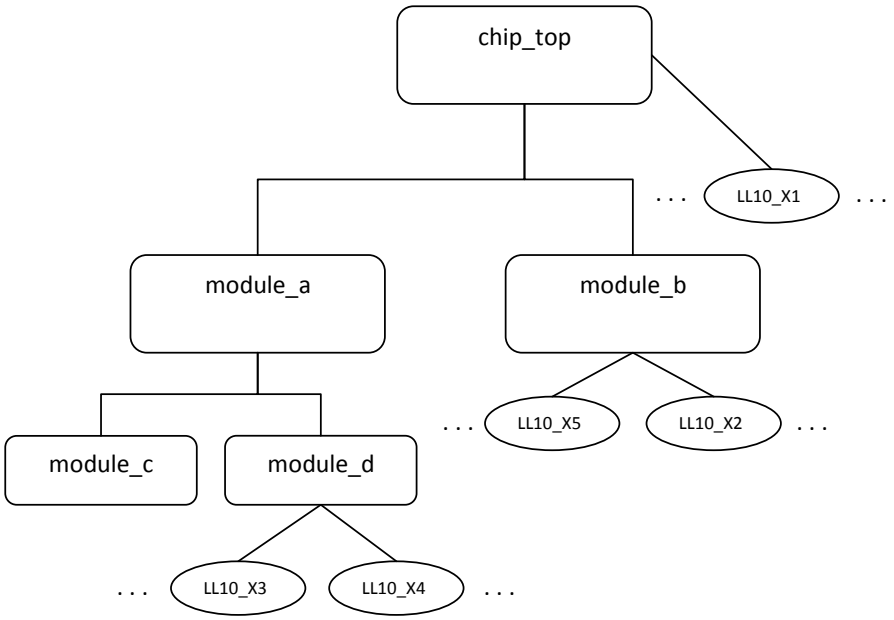

Abbildung 3.9: Schematischer Aufbau einer hierarchischen Netzliste

Diese Funktion sucht von der Instanzierung der Zelle aufwärts nach der Definition des Moduls, in welchem sich die Zelle befindet. Dazu wird in der Netzliste nach dem Keyword *module* gesucht und der Modulname abgespeichert. Um die aktuelle Hierarchie-Ebene zu bestimmen, wird die Netzliste nach der Instanzierung des gefundenen Moduls durchsucht und der Instanzname als erste Ebene des Hierarchie-Pfades gespeichert. Von hier aus wird wieder die Netzliste aufwärts nach der Modul-Definition und anschließend nach dessen Instanzierung durchsucht. Dieser Vorgang wird so oft wiederholt, bis keine Instanzierung des zuletzt ermittelten Moduls mehr gefunden werden kann. Die Funktion ist somit bei der Top-Ebene des Chips angelangt. Alle Instanznamen der Ebenen werden zusammen mit dem Instanznamen der Zelle selbst in einer Variable gespeichert. Dadurch ist die Zelle durch Typenbezeichnung und Hierarchie-Pfad vollständig definiert.

Mit der Hierarchie können nun die Kapazitäten an den Ausgängen sowie die Übergangszeiten an den Eingängen der Zelle aus den vorher generierten Dateien *caps.txt* und *trans.txt* ermittelt werden. Mit diesen Werten können nun alle möglich auftretenden Energien für die aktuelle Zelle berechnet werden. Dazu wird auch hier eine Sub-Funktion *search_power* verwendet, welche aus dem passenden Synopsys-Libfile dieser Zelle die Werte berechnet. Als Eingabe erhält diese Funktion den Namen der Bibliothek, in der die Zelle definiert ist, die Typenbezeichnung selbst und den Hierarchie-Pfad.

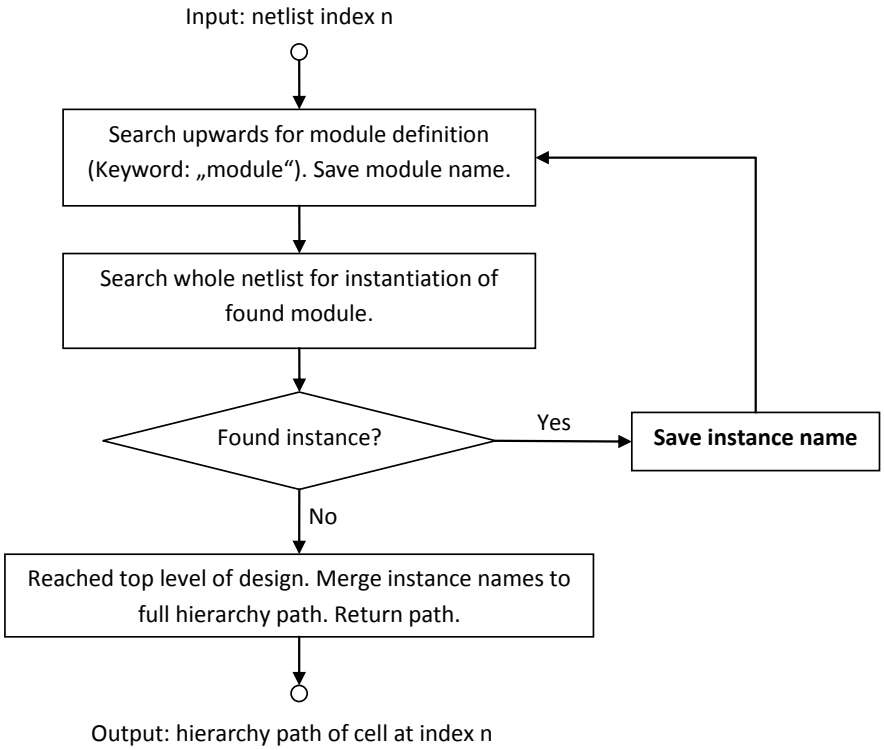

Abbildung 3.10: Programmablaufplan der Subfunktion *search_module*

Diese Funktion sucht zuerst nach der Definition der Zelle im Libfile. Wurde diese gefunden, wird nach der Definition der Internal Power gesucht. Wie diese aussieht, wird im nächsten Punkt beschrieben.

3.4.2 Berechnen der Energiewerte

Im Synopsys-Libfile der Zelle sind alle relevanten Daten wie die Funktionsbeschreibung oder Timing-Informationen enthalten. Ebenso gibt es eine Beschreibung über den allgemeinen Leckstrom der Zelle, sowie Informationen über die interne Energieaufnahme bei einer auftretenden Flanke. Für diese Internal Power sind für jede mögliche Pin-Stellung des Gatters Werte definiert, welche in einem Array aufgelistet sind. Welche Energie nun bei einem Signalwechsel auftritt, hängt im Falle eines Ausgangs von der Eingangs-Übergangszeit und der Ausgangskapazität, im Falle eines Eingangs nur von der Übergangszeit ab.

Dieser Bereich der Internal Power wird im Libfile mit *internal_power()* abgegrenzt. Innerhalb diesem gibt es noch die Unterteilung in die Blöcke *rise_power()* und *fall_power()*, sowie einen Bezeichner, der festlegt, ob es sich um die Energie am VDD-Pin oder am VSS-Anschluss handelt. Bei anderen Technologien wie z.B. C65 (Chips mit 65 nm Strukturbreite) gibt es noch einen extra definierten Internal Power-Bereich für den Anschluss VDD_BULK (siehe auch Kapitel 3.7). Alle Energiewerte im Libfile haben eine Einheit von 10^{-12} J.

Da die Energie für Eingänge nur von den Rise- und Fall-Zeiten abhängt, gibt es für diese ein eindimensionales Array, in dem die definierten Werte aufgelistet sind. Der Index dieses Arrays besteht aus verschiedenen Werten für die Übergangszeit, welche als 10^{-9} s definiert sind. Damit sind für bestimmte Punkte der Eingangs-Übergangszeit die auftretenden Energien definiert.

```
internal_power() {
     related_pg_pin : VDD;
     fall_power(power1d) {
     index_1(" 0.02       ,  0.6        ");
     values( " 0.00357149 ,  0.00357352 ")
     }
     rise_power(power1d) {
     index_1(" 0.002      ,  0.02       ,  0.06       ,  0.2        ,  0.28       ,  0.4        ,  0.6        ");
     values( " 0.000478339,  0.000349717,  0.000206285,  9.98454e-05,  7.88448e-05,  5.86658e-05,  3.70632e-05")
     }
     when              : "A'*C'*D'";
}
```

Abbildung 3.11: Wertetabelle der internen Leistungsaufnahme für Eingänge

Um nun den passenden Wert für eine bestimmte Eingangs-Übergangszeit t zu finden, wird eine Interpolation zwischen den verfügbaren Werten im Array durchgeführt. Für das

eindimensionale Array der Eingänge wird eine einfache lineare Interpolation verwendet.

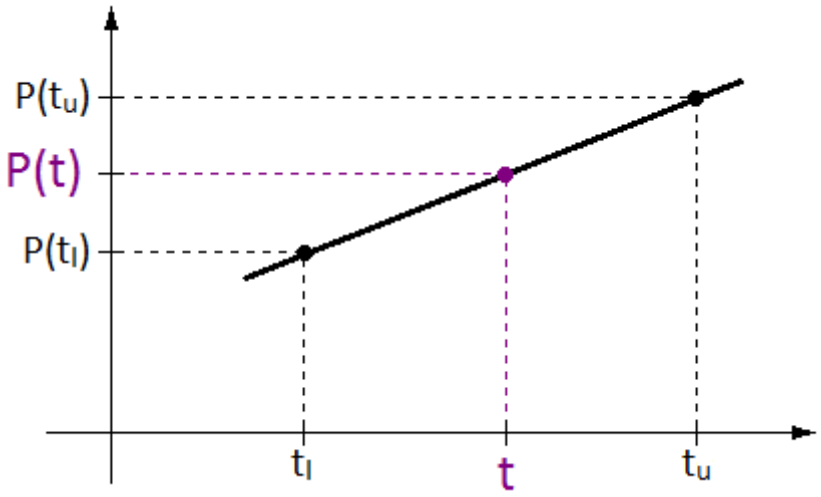

Abbildung 3.12: Prinzip der linearen Interpolation

Dabei werden die Punkte oberhalb (Index u, *upper*) und unterhalb (Index l, *lower*) des gesuchten Wertes mit einer Geraden verbunden, und mit der Eingangs-Übergangszeit t multipliziert.

$$\begin{aligned} k &= \frac{P(t_u) - P(t_l)}{t_u - t_l} \\ d &= P(t_l) - (k \cdot t_l) \\ P(t) &= k \cdot t + d \end{aligned} \tag{3.4.1}$$

Eine angenommene Rise-Transition von 0,03 ns liegt in der Tabelle aus Abbildung 3.11 zwischen den Werten 0,02 und 0,06. Diese beiden Werte fungieren als t_l und t_u. Damit muss der resultierende Wert für die Energie zwischen 0,000349717 und 0,000206285 liegen, mit der Formel 3.4.1 für die lineare Interpolation würde folgender Wert für die steigende Flanke berechnet:

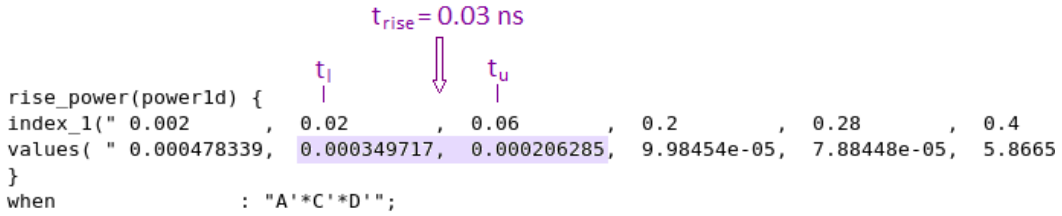

Abbildung 3.13: Eckpunkte der linearen Interpolation

$$
\begin{aligned}
t_u &= 0,06 \\
t_l &= 0,02 \\
P(t_u) &= 0,000206285 \\
P(t_l) &= 0,000349717 \\
k &= \frac{P(t_u) - P(t_l)}{t_u - t_l} = \frac{0,000206285 - 0,000349717}{0,06 - 0,02} = -0,0035858 \\
d &= P(t_l) - (k \cdot t_l) = 0,000349717 - (-0,0035858 \cdot 0,02) = 0,000421433 \\
P(t) &= k \cdot t + d = -0,0035858 \cdot 0,03 + 0,000421433 = 0,000313859 = 313,859\ pW
\end{aligned}
\tag{3.4.2}
$$

Für welchen Zustand dieser Wert gilt, steht unter der Tabelle, festgelegt in der Zeile *when*. Hier sind die Pins der Zelle aufgelistet, welche untereinander UND-verknüpft sind. Ein einfaches hochgestelltes Komma ' zeigt eine Negation an. Dementsprechend gilt der Energiewert aus der Beispiel-Berechnung für den Zustand A=0, C=0 und D=0, bei einer steigenden Flanke an Pin B.

Für Ausgänge sind im Libfile zweidimensionale Arrays definiert, in denen die Eingangs-Übergangszeit und die Ausgangs-Kapazität als Indizes fungieren.

Der erste Index entspricht wieder der Übergangszeit und gibt die Zeile des Arrays an. Der zweite Index repräsentiert die Ausgangs-Kapazität in 10^{-12} F. Er gibt die passende Spalte an. Um auch hier die besten Werte für bestimmte Zeit t und Kapazität C zu finden, wird in diesem Fall eine bilineare Interpolation verwendet.

Diese besteht im ersten Schritt aus zwei linearen Interpolationen zwischen den Punkten $P(Q_{12})$ und $P(Q_{22})$ bzw. zwischen $P(Q_{11})$ und $P(Q_{21})$. Anschließend werden diese beiden Punkte wiederrum mit einer Geraden verbunden und der endgültige Wert berechnet. Diese Schritte können in einfacher mathematischer Form in einer Formel ausgedrückt werden:

```
internal_power() {
        related_pg_pin : VDD;
        rise_power(power2d) {
        index_1(" 0.002      ,  0.06       ,  0.2        ,  0.4        ,  0.6        ");
        index_2(" 0.004      ,  0.016      ,  0.064      ,  0.128      ");
        values( " 0.0163152  ,  0.0163499  ,  0.0164019  ,  0.0164545  ", \
                " 0.0160071  ,  0.0160362  ,  0.0161719  ,  0.0162804  ", \
                " 0.0179274  ,  0.0173624  ,  0.0167471  ,  0.0165829  ", \
                " 0.0221056  ,  0.0208708  ,  0.0189473  ,  0.0180662  ", \
                " 0.0266402  ,  0.0249414  ,  0.0218922  ,  0.0202694  ");
        }
        fall_power(power2d) {
        index_1(" 0.002      ,  0.02       ,  0.06       ,  0.2        ,  0.28       ,  0.4        ,  0.6 ");
        index_2(" 0          ,  0.004      ,  0.016      ,  0.032      ,  0.064      ,  0.128      ");
        values( " 0.00246483 ,  0.00247387 ,  0.00248067 ,  0.00248322 ,  0.00248439 ,  0.00248491 ", \
                " 0.00216616 ,  0.00222363 ,  0.00230031 ,  0.00233662 ,  0.00236287 ,  0.00237919 ", \
                " 0.0021661  ,  0.00218291 ,  0.00223522 ,  0.00227772 ,  0.00231838 ,  0.00234857 ", \
                " 0.00391771 ,  0.00363211 ,  0.00320673 ,  0.00296645 ,  0.00276587 ,  0.00261918 ", \
                " 0.00543735 ,  0.00497693 ,  0.00422529 ,  0.00374963 ,  0.00331644 ,  0.00297794 ", \
                " 0.00800449 ,  0.00732352 ,  0.00611017 ,  0.00526136 ,  0.00442489 ,  0.00372801 ", \
                " 0.0126222  ,  0.0116765  ,  0.00982427 ,  0.008384   ,  0.00682427 ,  0.00542367 ");
        }
        related_pin        : "D";
        when               : "A'*B";
}
```

Abbildung 3.14: Wertetabelle der internen Leistungsaufnahme für Ausgänge

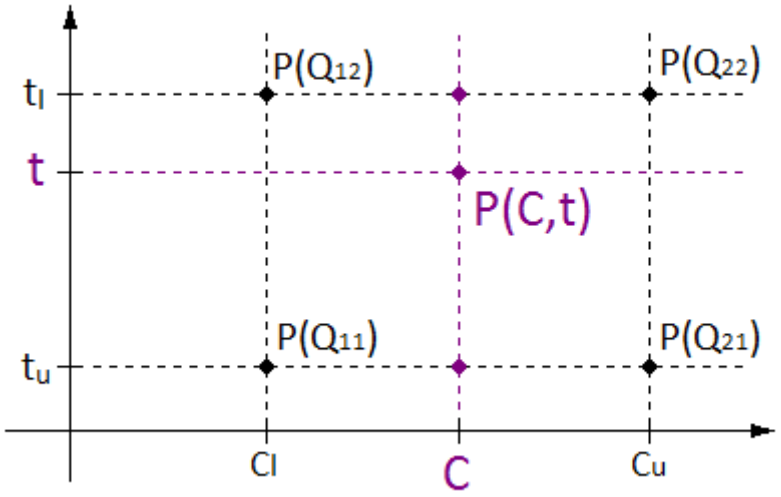

Abbildung 3.15: Prinzip der bilinearen Interpolation

$$
\begin{aligned}
P(C,t) \approx & \frac{P(Q_{11})}{(C_u - C_l)(t_l - t_u)}(C_u - C)(t_l - t) \\
& + \frac{P(Q_{21})}{(C_u - C_l)(t_l - t_u)}(C - C_l)(t_l - t) \\
& + \frac{P(Q_{12})}{(C_u - C_1)(t_l - t_u)}(C_u - C)(t - t_u) \\
& + \frac{P(Q_{22})}{(C_u - C_1)(t_l - t_u)}(C - C_l)(t - t_u)
\end{aligned}
\tag{3.4.3}
$$

Beispielberechnung für eine Fall-Transition-Time von 0,1 ns und einer Lastkapazität von 0,02 pF:

Die Übergangszeit von 0,1 ns liegt zwischen den Punkten 0,06 (=t_l) und 0,2 (=t_u) im ersten Index. Da der erste Index die Zeile des zweidimensionalen Arrays festlegt, wird der Bereich des zu berechnenden Wertes durch die Zeilen 3 und 4 begrenzt (0,06 ist der dritte Wert im Index, 0,2 der vierte). Der Kapazitätswert von 0,02 pF liegt im zweiten Index zwischen den Werten 0,016 (=c_l) und 0,032 (=c_u), also zwischen der dritten und vierten Spalte im Array. Dadurch ist der Bereich, in dem sich der Wert befindet, definiert und es kann mit der Interpolation begonnen werden.

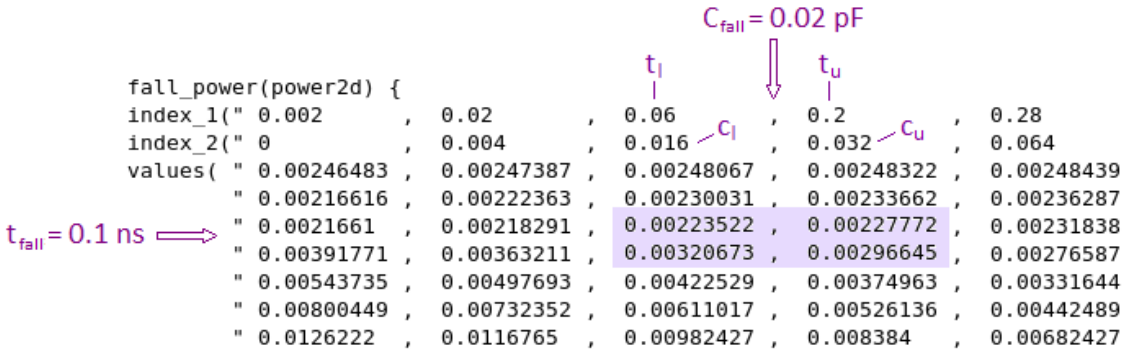

Abbildung 3.16: Eckpunkte der bilinearen Interpolation

$$
\begin{aligned}
P(Q_{11}) &= 0,00320673,\ P(Q_{12}) = 0,00223522 \\
P(Q_{21}) &= 0,00296645,\ P(Q_{22}) = 0,00227772 \\
t_u &= 0,2,\ t_l = 0,06 \\
C_u &= 0,032,\ C_l = 0,016 \\
P(C,t) &\approx \frac{0,00320673}{(0,032-0,016)(0,06-0,2)}(0,032-0,02)(0,06-0,1) \\
&\quad + \frac{0,00223522}{(0,032-0,016)(0,06-0,2)}(0,02-0,016)(0,06-0,1) \\
&\quad + \frac{0,00296645}{(0,032-0,016)(0,06-0,2)}(0,032-0,02)(0,1-0,2) \\
&\quad + \frac{0,00227772}{(0,032-0,016)(0,06-0,2)}(0,02-0,016)(0,1-0,2)
\end{aligned}
\tag{3.4.4}
$$

$$P(C,t) \approx 0,00284272 = 2,84272\ nW$$

Auch bei Ausgängen wird mit *when* festgelegt, für welchen Zustand der Wert gilt. Zusätzlich wird mit *related_pin* angezeigt, welcher Eingangspin für die Änderung am Ausgang verantwortlich war. Das ist nötig, weil eine Flanke am Ausgang, welche beispielsweise durch eine Änderung an Pin A hervorgerufen wurde, eine andere interne Energie verbraucht als eine Flanke, welche durch Pin B initiiert worden ist. Dementsprechend ist der Wert aus obiger Berechnung gültig für eine fallende Flanke am Ausgang Z, die von einer Änderung an Pin D hervorgerufen wurde (related_pin : “D”) , bei einem Zustand von A=0 und B=1.

Wenn alle Werte berechnet wurden, gibt die Funktion diese in einem Array an das Hauptprogramm zurück. Danach erfolgt die Berechnung der Switching Energy. Dafür wird entsprechend Formel 2.3.7 die Kapazität des Ausgangs der Zelle mit der quadrierten Versorgungsspannung multipliziert. Sollte eine Zelle mehrere Ausgänge besitzen, werden für alle Ausgänge einzelne Energien berechnet.

Als letztes wird noch die statische Leistungsaufnahme (Leakage Power) aus der Bibliothek der Zelle ermittelt. Diese steht in der Sektion *cell_leakage_power* und ist - im Gegensatz zur Internal Power - bereits als Leistung und nicht als Energie definiert. Das Synopsys-Format des Libfiles würde - ähnlich der Internal Power - auch eine Definition eines zustandsabhängigen Leckstroms erlauben, dieser ist jedoch in den zur Verfügung stehenden Bibliotheken noch nicht vorhanden. Deshalb wurde bei der aktuellen Berechnung nur der zustands*un*abhängige Leckstrom ermittelt.

Diese Methode zum Ermitteln der Energiewerte kann nur für Standardzellen vorgenommen werden. Für Makros wie Speicherelemente gibt es keine Bibliothek, in der die Energien

entsprechend definiert sind. Zusätzlich sind bei komplizierteren Makros wie NVM die Energieverbräuche nicht immer gleich, sondern hängen vom Betriebsmodus und anderen Faktoren ab.

3.4.3 Patchen der Netzliste

Um die berechneten Energiewerte während der Simulation verarbeiten zu können, müssen diese den funktionalen Verilog-Modellen der Zellen übergeben werden. Dies geschieht mit Hilfe von Parametern, die in die funktionalen Verilog-Modelle der Zellen eingefügt und bei der Instanzierung in der Netzliste mit den Werten gefüllt werden.

Dazu werden die Parameter-Werte in der Netzliste zwischen Zelltyp und Instanzname hineingeschrieben (Abbildung 3.17). Da in den Parametern keine Kommazahlen übergeben werden können, müssen die Energiewerte skaliert werden. Hierbei muss darauf geachtet werden, dass die Skalierung nicht zu groß gewählt wird, da dadurch die Zählerwerte während der Simulation zu stark ansteigen und Überläufe produzieren könnten. Ebenso darf die Skalierung nicht zu klein sein, da sonst die einzelnen Werte zu klein werden um eine ausreichende Genauigkeit zu garantieren. Als Kompromiss zwischen diesen Faktoren wird ein Skalierungswert von 10^5 gewählt. Da die Energiewerte in den Bibliotheken schon als 10^{-12} J definiert sind, ergibt sich somit als Einheit für die Zähler 10^{-17} J.

```
SLL10L100_USBUFX120 #("i_dcore___i_auth",1,1825,2151,171) I0004 (.A ( net0024 ) , .Z ( net00 ...
SLL10L100_UBSIVX120 #("i_dcore___i_auth",0,3042,2363,3935) I0003 (.A ( net0021 ) , .Z ( net0 ...
SLL10L100_USLNDX020 #("i_dcore___i_auth",0,5613,-527,709,-668,684,1673,252,2092,252) I0012 ( ...
SLL10L100_USAN2X040 #("i_dcore___i_auth",0,5643,-391,584,-533,547,1489,2989,1707,3594) I0016 ...
```

Abbildung 3.17: Parameterübergabe in der Netzliste

Die Reihenfolge der Parameter ist nach folgendem Prinzip festgelegt:

#("<hierarchy-path>",<clock_tree_true/false>,<switching_energy>,
<internal_energy_1>, ..., <internal_energy_N>)

Beim Aufruf des "powersim"-Skripts kann man mit dem Parameter *-depth* angeben, ob eine Aufschlüsselung in Module stattfinden soll oder nicht. Wird kein Wert übergeben, wird als Tiefe 0 angenommen und es wird nur der Leistungsverbrauch des gesamten Chips simuliert. Je größer der Parameter *-depth* gewählt wird, desto tiefer wird das Design analysiert. Wenn bspw. eine Analyse bis zur zweiten Ebene unter der Top-Ebene gewünscht wird, muss als Parameter 2 übergeben werden (siehe auch Abbildung 3.9). Durch diese Option wird jeder Zelle ein Hierarchie-Pfad übergeben, der bis in die angegebene Tiefe reicht, um die Zellen jenen Modulen zuzuordnen, in denen sie sich befinden. Die Trennung zwischen zwei Hierarchie-Ebenen findet hierbei mit einem dreifachen Unterstrich statt.

Im Netzlisten-Auszug aus Abbildung 3.17 wurde eine Tiefe von 2 angegeben. Die Zellen befinden sich also im Modul *i_auth*, welches wiederum im Modul *i_dcore* liegt.

Da man in der Regel nicht immer alle Module einer kompletten Hierarchie-Ebene untersuchen will, wurde auch eine Möglichkeit entwickelt, nur bestimmte Module aufzulisten. Dies geschieht mit einer externen Datei *watchlist.txt*, in welcher einzelne Pfade zu Modulen angegeben werden können, für die der Energieverbrauch ermittelt werden soll. In der Datei müssen die Hierarchien im normalen Format für Hierarchie-Pfade angegeben werden, d.h. die einzelnen Ebenen müssen mit einem einfachen Slash getrennt werden.

Die Watchlist kann auch gemeinsam mit der Anweisung *-depth* verwendet werden. Sollte sich eine Instanz in der Watchlist in einer Ebene befinden, die schon durch die depth-Anweisung abgedeckt ist, wird dieser Eintrag in der externen Datei ignoriert.

3.5 Modifizieren der funktionalen Modelle

In den funktionalen Modellen werden die Parameter am Beginn der jeweiligen Zelle eingefügt und mit dem Wert 0 initialisiert. Diese Werte werden dann entsprechend den auftretenden Events behandelt. Eine komplette modifizierte Zelle eines Inverters ist in Abbildung 3.18 dargestellt.

Die Benennung der Parameter erfolgt nach einem speziellen Schema:

$$\overbrace{A}^{Pin}\_\underbrace{r}_{rise/fall}\_\overbrace{B1\_C0\_D1}^{Eingangszustand} \tag{3.5.1}$$

Der erste Buchstabe gibt den aktuellen Pin an, an dem die Änderung auftritt. Ob es sich bei der Änderung um eine steigende oder fallende Flanke handelt, wird mit dem zweiten Buchstaben angegeben. Hierbei steht r für eine steigende, f für eine fallende Flanke. Am Ende wird die Stellung der Eingangspins beschrieben, für die der Energiewert gültig ist. Das obere Beispiel zeigt also den Parameter für eine steigende Flanke an Pin A, bei den Eingangszuständen B=1, C=0 und D=1.

Für Ausgänge gibt es noch einen zusätzlichen Eintrag:

$$Z\_f\_\overbrace{rel\_A}^{Related\ Pin}\_B0\_C1\_D1 \tag{3.5.2}$$

Mit dem *Related Pin* wird angezeigt, welcher Eingangspin für die Änderung am Ausgang verantwortlich war. Der obere Fall beschreibt die Energie, die eine fallende Flanke am

```
module LL10L100_UBSIVX010 (A, Z);

output  Z;
input   A;

not #0.001 not1 (Z,A);

specify
  specparam def_del  = 0.01;

// Delays
  (A  => Z) = def_del;
endspecify

endmodule
```

```
module LL10L100_UBSIVX010 (A, Z);

        parameter hier=0;
        parameter clock_tree=0;
        parameter switching_energy=0;
        parameter Z_r=0;
        parameter Z_f=0;

output  Z;
input   A;

not #0.001 not1 (Z,A);

specify
  specparam def_del  = 0.01;

// Delays
  (A  => Z) = def_del;
endspecify

reg [47:0] counter_internal=0;
reg [47:0] counter_switching=0;

`include "powersim_starlib/verilog/power_task.v"
always @(posedge Z)
begin
        power_task(hier,clock_tree,Z_r,switching_energy);
end
always @(negedge Z)
begin
        power_task(hier,clock_tree,Z_f,0);
end

endmodule
```

Abbildung 3.18: Links: normales Inverter-Modell, Rechts: modifiziertes Modell

Ausgang Z, welche durch eine Änderung an Pin A hervorgerufen wurde, verbraucht, wenn die Eingänge auf B=0, C=1 und D=1 liegen.

Alle möglichen Energiewerte, die auftreten können, werden also mit Hilfe von Parametern an das Modul übergeben. Auf diese kann das Modul selbst während der Simulation zugreifen und sie entsprechend verarbeiten.

Dazu werden für jeden Pin des Modells zwei always-Blöcke definiert, die auf die steigende bzw. fallende Flanke des Pins reagieren. Innerhalb dieser Blöcke gibt es if-, case- bzw. casex-Abfragen, um die aktuell vorherrschenden Signale an den relevanten Pins der Zelle abzufragen. Sollten nicht alle Pins relevant sein, werden diese im jeweiligen casex-Block mit x für *dont care* behandelt.

Als Bespiel wird die Parameter-Auswahl für eine angenommene Flanke an einem UND-Gatter mit 3 Eingängen gezeigt. Die aktuelle Stellung der Eingänge B und C sei 1 und 0, während eine steigende Flanke an Pin A auftritt. Auf diese Flanke reagiert der "always @(posedge A)"-Block und setzt als erstes den integer-Wert *recent* auf 0 (Abbildung 3.19). Mit dieser Variable wird immer die als letzte auftretende Flanke an den Eingängen gespeichert, um bei einer eventuellen Ausgangs-Änderung feststellen zu können, welche Pin-Änderung am Eingang dafür verantwortlich war.

```
always @(posedge A) begin
        recent = 0;
        case ( {B,C} )
                2'b00 :  power_task(hier,0,A_r_B0_C0,0);
                2'b10 :  power_task(hier,0,A_r_B1_C0,0);
                2'b11 :  power_task(hier,0,A_r_B1_C1,0);
        endcase
end
```

Abbildung 3.19: always-Block für eine steigende Flanke an Pin A

Danach wird die aktuelle Stellung der Pins B und C abgefragt. Im Beispiel ist B auf 1 und C auf 0, das bedeutet, dass der zweite Fall im Case-Konstrukt zutrifft. Das Aufsummieren selbst übernimmt ein Task, welcher als Übergabeparameter den entsprechenden Energiewert (A_r_B1_C0), den Switching Energy Wert (in diesem Fall ist dieser 0, da es sich um eine Flanke an einem Eingang handelt), ein Clock Tree true/false Bit (hier 0, also keine Clock Tree Zelle) und im Falle einer modulfeinen Aufschlüsselung noch den Hierarchie-Pfad der Zelle erhält.

Wie bereits erwähnt, kann beim Aufruf des "powersim"-Skripts mit dem Parameter *depth* angegeben werden, bis auf welche Ebene des Designs die Aufschlüsselung der Module durchgeführt werden soll. Wird keine Tiefe angegeben, wird 0 angenommen. Damit wird nur die Energieaufnahme des gesamten Chips simuliert, eine Aufschlüsselung in Module

findet nicht statt.

Für die Ermittlung des gesamten Leistungsverbrauchs des Chips bzw. einzelner Module, werden globale Zähler benötigt. In das Design wird dazu ein neues Modul *global_power_counter* eingebracht.

```
module global_power_counter ();

        parameter leakage=0;

        //*** Unit is e-12 Watt ***
        reg [63:0] leakage_power=leakage;

        integer enable=0;

        //*** Unit is e-17 Joule ***
        reg [63:0] energy=0;
        reg [63:0] internal_energy=0;
        reg [63:0] switching_energy=0;

        reg [63:0] clock_tree_internal_energy=0;
        reg [63:0] clock_tree_switching_energy=0;

        real old_energy_internal, old_energy_switching, temp_int, temp_switch;
        real e_diff_internal_J, e_diff_switching_J;

        //*** counter definition ***

        reg [47:0] i_dcore_internal=0;
        reg [47:0] i_dcore_switching=0;
        reg [47:0] i_dcore=0;
        reg [47:0] i_dcore___i_auth_internal=0;
        reg [47:0] i_dcore___i_auth_switching=0;
        reg [47:0] i_dcore___i_auth=0;
        //*** Unit is e-17 Joule ***

        //*** end counter definition ***
```

Abbildung 3.20: Eingefügtes Modul - globale Zähler

Dieses beinhaltet die globalen Variablen, in denen die Internal und die Switching Power aufsummiert werden, ein Register mit der gesamten Leakage Power der Netzliste, sowie Zähler für Clock Tree-Zellen. Wenn im Perl-Skript mit dem Parameter *depth* eine Aufschlüsselung in einzelne Module gewählt wurde, stehen hier auch die einzelnen Zähler für die jeweiligen Module. Für jedes Element gibt es hier eigene Zähler für Internal und Switching, sowie einen Zähler mit der gesamten Energie des Moduls. Selbiges gilt, wenn in der externen Datei *watchlist.txt* Hierarchien angegeben wurden, für die man die Energie separat simulieren möchte.

Um nicht nur die verbrauchte Energie, sondern auch den aktuellen Stromverbrauch der Schaltung zu bestimmen, wurde eine Berechnung der aktuellen Leistung hinzugefügt (Abbildung 3.21). Dazu wird die Energiezunahme in einem bestimmten Zeitintervall t_{int} ermittelt und durch diese Zeit dividiert. Da es sich hierbei nicht um die mathematisch korrekte Berechnung der Leistung handelt, sondern nur eine Annäherung der Ableitung

```
always begin
        #1000 begin

                temp_int=$signed(`PATH_TOP.internal_energy)-old_energy_internal;
                temp_switch=$signed(`PATH_TOP.switching_energy)-old_energy_switching;

                e_diff_internal_J=temp_int*0.00000000000000001;
                e_diff_switching_J=temp_switch*0.00000000000000001;

                old_energy_internal=$signed(`PATH_TOP.internal_energy);
                old_energy_switching=$signed(`PATH_TOP.switching_energy);

        end
end
```

Abbildung 3.21: Eingefügtes Modul - Leistungsberechnung

durchgeführt wird, wird das Resultat nicht als $P(t)$ sondern als $E_{diff}(t)$ bezeichnet.

$$E_{diff}(t) = \frac{E(t) - E(t - t_{int})}{t_{int}} \tag{3.5.3}$$

Je kürzer das Zeitintervall t_{int} gewählt wird, desto genauer wird die Leistung berechnet. Entsprechend dem festgelegten Wert im timescale-Attribut ist das Zeitintervall in Abbildung 3.21 1000 ns, also findet hier eine Leistungsberechnung mit einem Intervall t_{int} von 1 µs statt. Dadurch, dass die Energiekurve nicht stetig, sondern aus kleinen Sprüngen zusammengesetzt ist, kann das Zeitintervall t_{int} hierbei aber nicht beliebig klein gemacht werden.

Wie bereits erwähnt, übernimmt das Aufsummieren der Zähler ein Power-Task (Abbildung 3.22). Ein Verilog-Task ist eine Art Sub-Funktion, welche innerhalb eines Moduls aufgerufen und ausgeführt werden kann. Dieser Task kann in einer externen Datei definiert und mit einem include-Befehl in das Modul, in dem der Task aufgerufen werden soll, eingebunden werden. Im Gegensatz zu einer Verilog-Funktion muss ein Task keinen Rückgabewert liefern, der Power-Task greift direkt auf die globalen Variablen im global_power_counter zu. Im Beispiel wird eine Aufschlüsselung der Energie für die Module *i_dcore* und *i_dcore/i_auth* gegeben.

Am Beginn des Tasks werden die Eingänge und lokalen Variablen definiert (Abbildung 3.22).

Der erste Eingangswert ist die Hierarchie, welche als String an den Task übergeben wurde. Ein einzelnes ASCII-Zeichen braucht zur Speicherung ein Register mit 8 Bit. Dementsprechend ist das Register *hier* auf 128 Zeichen begrenzt. Die weiteren übergebenen Parameter werden als Integer-Variablen abgespeichert. Dazu kommen noch verschiedene Register und Variablen, die für Zwischenergebnisse und andere Berechnungen im Task verwendet werden.

```
task power_task;

        input reg [1023:0] hier;
        input integer clock_tree;
        input integer energy_internal_value, energy_switching_value;

        reg signed [63:0] energy_internal_total;
        reg signed [63:0] energy_switching_total;

        real energy_;
```

Abbildung 3.22: Power Task - Teil 1

Danach findet das eigentliche Aufsummieren der Zähler statt. Die Zähler für Internal und Switching Power werden separat erhöht, ebenso wie der Zähler für die gesamte Energie.

```
energy_internal_total = $signed(`PATH_TOP.internal_energy) + $signed(energy_internal_value);
`PATH_TOP.internal_energy = energy_internal_total;

energy_switching_total = $signed(`PATH_TOP.switching_energy) + $signed(energy_switching_value);
`PATH_TOP.switching_energy = energy_switching_total;

`PATH_TOP.energy = `PATH_TOP.internal_energy+`PATH_TOP.switching_energy;

if ( clock_tree == 1 )
begin
        `PATH_TOP.clock_tree_internal_energy = $signed(`PATH_TOP.clock_tree_internal_energy)
                                             + $signed(energy_internal_value);
        `PATH_TOP.clock_tree_switching_energy = $signed(`PATH_TOP.clock_tree_switching_energy)
                                              + $signed(energy_switching_value);

end

counter_internal = counter_internal+energy_internal_value;
counter_switching = counter_switching+energy_switching_value;
```

Abbildung 3.23: Power Task - Teil 2

Für den Fall, dass es sich um eine Clock Tree-Zelle handelt, werden auch noch die speziellen Zähler *clock_tree_internal_energy* und *clock_tree_switching_energy* erhöht. Zusätzlich besitzt jede Zelle zwei eigene Zähler *counter_internal* und *counter_switching*, welche nur die Energie der Zelle selbst aufsummieren. Damit ist eine Analyse des Leistungsverbrauchs bis in die tiefste Ebene (also einzelne Standardzellen) möglich.

Am Beginn wurden die einfacheren Zellen wie Inverter, Buffer oder UND- bzw. ODER-Gatter mit wenigen Eingängen modelliert. Da die Funktion dieser Zellen relativ einfach ist und nur wenig verschiedene Zustände auftreten können, wurde die Modellierung dieser Typen manuell durchgeführt. Für Zellen mit komplexeren Funktionen und mehr Eingängen wurde ein Parser (ebenfalls in Perl) entwickelt, welcher aus dem entsprechenden Libfile der Zelle alle möglichen Zustände der Pins ausliest und die Parameter automatisch benennt und erstellt. Dieser Parser ist vor allem für größere Standardzellen von Vorteil, da

```
energy_ = $signed(energy_internal_value) + $signed(energy_switching_value);

//*** counter case ***

case ( hier )
 "i_dcore" : begin
 `PATH_TOP.i_dcore_internal=$signed(`PATH_TOP.i_dcore_internal) + energy_internal_value;
 `PATH_TOP.i_dcore_switching=$signed(`PATH_TOP.i_dcore_switching) + energy_switching_value;
 `PATH_TOP.i_dcore=$signed(`PATH_TOP.i_dcore_switching) + $signed(`PATH_TOP.i_dcore_internal);
 end
 "i_dcore___i_auth" : begin
 `PATH_TOP.i_dcore_internal=$signed(`PATH_TOP.i_dcore_internal) + energy_internal_value;
 `PATH_TOP.i_dcore_switching=$signed(`PATH_TOP.i_dcore_switching) + energy_switching_value;
 `PATH_TOP.i_dcore=$signed(`PATH_TOP.i_dcore_switching) + $signed(`PATH_TOP.i_dcore_internal);
 `PATH_TOP.i_dcore___i_auth_internal=$signed(`PATH_TOP.i_dcore___i_auth_internal) + energy_internal_value;
 `PATH_TOP.i_dcore___i_auth_switching=$signed(`PATH_TOP.i_dcore___i_auth_switching) + energy_switching_value;
 `PATH_TOP.i_dcore___i_auth=$signed(`PATH_TOP.i_dcore___i_auth_switching) + $signed(`PATH_TOP.i_dcore___i_auth_internal);
 end
endcase
```

Abbildung 3.24: Power Task - Teil 3

dort bis zu 3800 verschiedene Zustände auftreten können und damit auch 3800 Parameter berechnet und übergeben werden müssen.

Alle Verilog-Modelle der Zellen werden in einer Datei *all_elements_orig.v* zusammengefasst, um die Handhabung dieser zu erleichtern. Diese Datei wird eingelesen und die Zellen entsprechend modifiziert. Die fertigen Modelle mit den Parametern und den if- bzw. case-Abfragen werden in eine Ausgabedatei *all_elements.v* geschrieben. Auch für den Power-Task gibt es eine eigene Datei mit dem Namen *power_task.v*.

3.6 Simulationsumgebung

Als Simulationsumgebung für den Test des Fast Power Simulators wird ModelSim® von MentorGraphics® verwendet. ModelSim ist bei Infineon der Standard-Simulator für Entwicklungen in der l90-Technologie, welche für aktuelle Chips verwendet wird. Deshalb wird auch für die Entwicklung des FPS dieser Simulator verwendet.

Die Dateien, welche nach dem erfolgreichen Durchlauf des “powersim”-Skripts erzeugt werden, müssen in die bestehende ModelSim-Umgebung eingebunden werden, um diese entsprechend zu verwenden. Im ersten Schritt werden dazu die Umgebungsvariablen MODELSIM und MGC_LOCATION_MAP gesetzt.

```
setenv MODELSIM powersim_starlib/modelsim.ini
setenv MGC_LOCATION_MAP powersim_starlib/mgc_location_map
```

In modelsim.ini stehen Daten zur Initialisierung des Simulators, sowie verschiedene Variablen und Pfade. Die mgc_location_map beinhaltet die Pfade zu diversen Bibliotheken, die für die Simulation benötigt werden.

Um die modifizierten funktionalen Verilog-Modelle der Zellen zu verwenden, müssen diese zuerst kompiliert werden. Als ersten Schritt muss hierzu das Verzeichnis, in dem die später erzeugten Kompilate der Modelle liegen sollen, festgelegt werden. Das geschieht mit dem Befehl *vlib*.

```
vlib powersim_starlib/MSobj_6.1
```

Das Kompilieren selbst erfolgt mit dem Befehl *vlog*. Als Parameter erhält dieser Befehl das allgemeine Verzeichnis der neuen Modell-Bibliothek (als Variable *powersim_starlib*, festgelegt in modelsim.ini), den direkten Pfad zu der Datei, in welcher der Verilog-Code der Modelle liegt (*all_elements.v*) sowie eine Datei mit timescale-Informationen für die Simulation (*starlib_ts_ms.v*):

```
vlog -work powersim_starlib starlib_ts_ms.v
          powersim_starlib/verilog/all_elements.v
```

Damit sind die Modelle fertig kompiliert und können in der Simulation verwendet werden. Dazu muss aber noch die adaptierte Netzliste kompiliert werden, um auch die Initialisierung der Parameter mit den Energiewerten zu ermöglichen und das Modul *global_power_counter* zu initialisieren. Auch das Kompilieren der Netzliste passiert mit dem Befehl *vlog*.

```
vlog -work test_design ts_ms.v home/usr/routing/test_design_hier.v
```

Wenn sowohl das Kompilieren der funktionalen Modelle, als auch das der Netzliste erfolgreich war, kann die Simulation mit *vsim* gestartet werden.

3.7 Transfer auf C65-Technologie

Nachdem der Fast Power Simulator für die l90-Technologie implementiert und getestet wurde, wurde der Wunsch geäußert, den FPS auch für die Technologie "C65" vorzubereiten und zu verwenden. Die C65-Technologie wird bei Infineon in Zukunft für neu entwickelte Chips verwendet, die mit einer Strukturbreite von 65 nm gefertigt werden. Da sich diese Generation von Chips noch in einem frühen Stadium der Entwicklung befindet, sind hier genaue Simulationen des Energieverbrauchs natürlich von großem Interesse und bieten einen erheblichen Vorteil im Hinblick auf Entwicklung und Verbesserung der Schaltungen.

Die Unterschiede für den Fast Power Simulator zwischen l90 und C65 sind relativ gering, müssen aber dennoch berücksichtigt werden. Allen voran gibt es leicht unterschiedliche Standardzellen mit neuen Bibliotheken. Diese Zellen besitzen keinen einheitlichen Prefix mehr, sondern haben je nach Struktur einen eigenen Bezeichner:

- H10/H12: High threshold voltage
- R10/R12: Regular threshold voltage

Also muss beim Durchlauf einer Netzliste nicht nur nach einem bestimmten Prefix gesucht werden, sondern nach allen oben genannten. Zu den alten Zellen mit neuem Prefix kommen auch komplett neue Standardzellen hinzu, bei denen wieder die funktionalen Modelle entsprechend adaptiert werden müssen.

Ein weiterer Unterschied liegt in der Definition der Internal Power-Werte in den Bibliotheken der Zellen. Zu den Blöcken für die Energien aus dem Anschluss VDD kommen noch Informationen für VDD_BULK hinzu. Der absolut auftretende Energieverbrauch setzt sich aus der Summe von VDD und VDD_BULK zusammen, dementsprechend müssen im ersten Schritt die Energien wie bisher für den *related_pg_pin: VDD* ausgelesen werden. Danach müssen noch zusätzlich die Energie-Werte aus dem Bereich *related_pg_pin: VDD_BULK* ermittelt und mit den vorherigen addiert werden. Erst dann hat man die korrekte Energiemenge, die anschließend als Parameter in die Netzliste geschrieben werden kann.

Für die Technologie C65 hat man sich bei Infineon auch für einen Wechsel auf eine neue Simulator-Umgebung festgelegt. Statt dem bisher verwendeten ModelSim von MentorGraphics wird in Zukunft Incisive® (in vielen Fällen oft nur *NCSim* genannt) von Cadence® verwendet. Deshalb muss auch das Integrieren vom Fast Power Simulator in die Simulationsumgebung angepasst werden.

In dieser Umgebung müssen die Netzliste und die Verilog-Modelle der Zellen nicht im Vorhinein kompiliert werden. Dies geschieht beim Start des Simulators automatisch, deswegen werden die Pfade zur Datei *all_elements.v* und zur Netzliste direkt beim Aufruf mit angegeben. Mit dem Befehl *irun* wird der Simulator gestartet:

```
irun -design_top test_design \
     /home/user/all_elements.v -ALLOWREDEFINITION \
     /home/user/test_design_netlist.v \
     /home/user/test_design_tb.sv \
     -gui
```

Sollte das Kompilieren ohne Fehler durchgeführt worden sein, startet die grafische Umgebung *SimVision* und die Simulation mit dem FPS kann begonnen werden.

4 Simulationsergebnisse

4.1 Erste Ergebnisse

Bei den ersten Simulationen wurde nur ein einziger Inverter modelliert (siehe Abbildung 3.18) und dessen Parameter mit relativ großen Werten beschrieben. Dadurch sollte schnell ersichtlich werden, ob das Aufsummieren der Energien aus den Parametern richtig funktioniert.

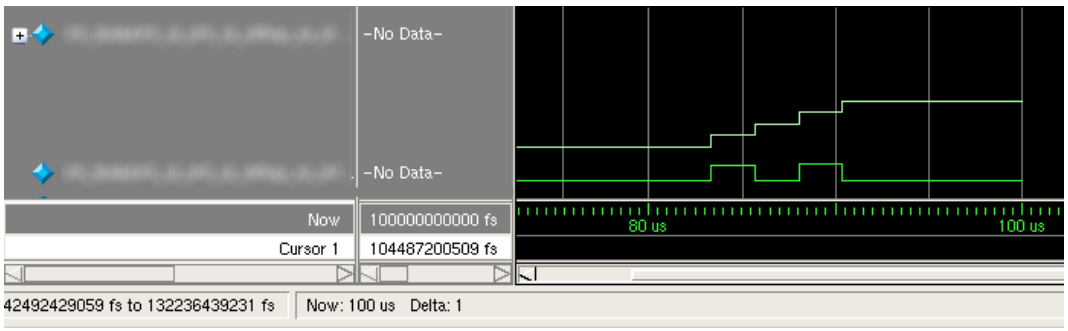

Abbildung 4.1: Erste Simulation

Das untere Signal stellt den Ausgang des Inverters dar, das obere Register ist der globale Zähler, welcher die interne Energie der Zelle aufsummiert. Man erkennt gut, dass bei jeder Änderung des Ausgangs der Zähler um einen gewissen Energiewert erhöht wird. In diesem Fall verbraucht eine fallende Flanke eine geringere Energie als eine steigende, deshalb sind die Sprünge im Zähler bei einer fallenden Flanke am Ausgang des Inverters etwas kleiner, als jene bei der steigenden Flanke.

Mit dieser Simulation konnte bereits zu einem frühen Zeitpunkt das Konzept geprüft und mit der Adaptierung und Modellierung aller Zellen begonnen werden.

4.2 Diverse Simulationen

Nachdem ein Teil der Zellen modelliert wurde, musste nach einigen Simulationen festgestellt werden, dass die aufsummierte Energie tendenziell etwas zu groß ist. Nach kurzer

Suche der Ursache wurde festgestellt, dass bereits die Initialisierung eines Signales zu Beginn der Simulation als eine reguläre Flanke gewertet wurde, auch wenn es sich um eine Flanke von x auf 1 oder von x auf 0 handelt. Diesen Effekt kann man in Abbildung 4.2 gut erkennen.

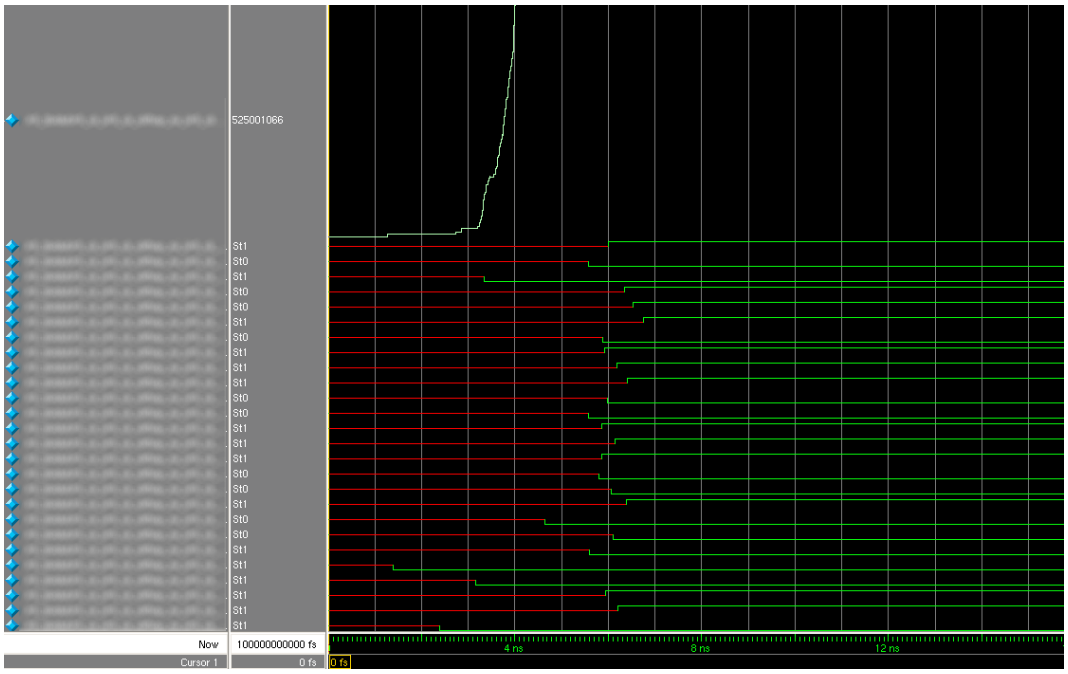

Abbildung 4.2: Initialisierung der Werte

Um das zu verhindern, wurde eine Variable *enable* eingeführt. Nur wenn diese auf den Wert 1 gesetzt wird, beginnen die Zähler des FPS zu laufen. Diese Variable hat auch den Vorteil, dass die Laufzeit einer FPS-Simulation in Bereichen, in denen die Leistungsaufnahme nicht relevant ist, deutlich beschleunigt werden kann (siehe dazu auch Kapitel 4.3).

In Abbildung 4.3 sieht man gut, wie die Erhöhung der Zähler und die damit verbundene Berechnung der Leistung nur in jenen Zeitabschnitten aktiv ist, in denen die *enable*-Variable auf 1 gesetzt ist. Die Zählerstände werden bei Deaktivierung des FPS nicht gelöscht, sondern nur "eingefroren".

Da die Leistung proportional dem Strom ist, kann durch eine Division der Leistung durch die Versorgungsspannung V_{DD} auf den Strom geschlossen werden.

Mit der Information über den Strom, welcher im jeweiligen Zeitpunkt verbraucht wird, kann mit Hilfe der Stützkapazität auf die aktuell vorherrschende Versorgungsspannung geschlossen werden. Bei einem zu hohen Stromverbrauch wird zu viel Ladung aus der

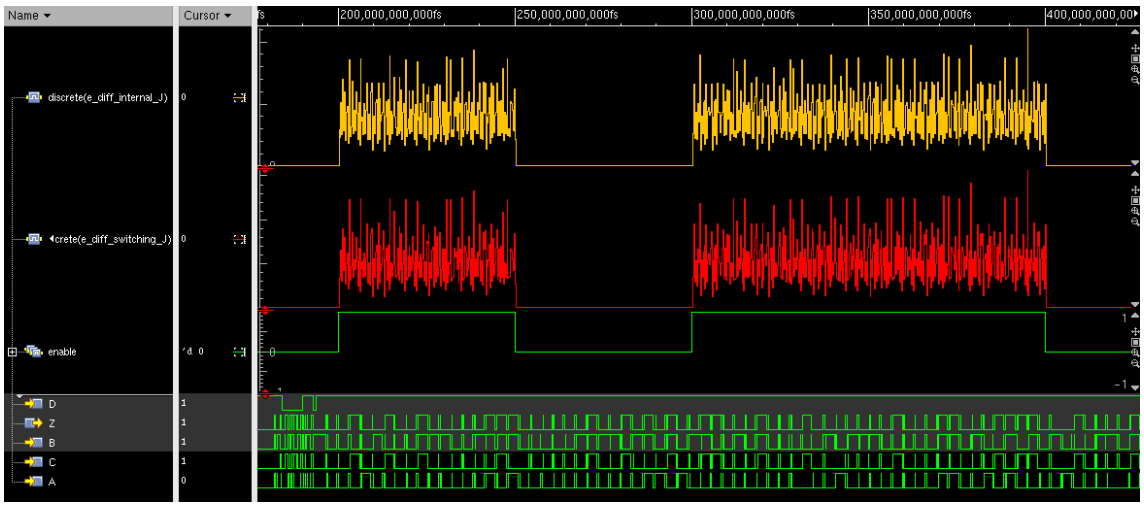

Abbildung 4.3: Aktivierung des FPS mit dem Signal *enable*

Kapazität gezogen, dadurch bricht die Spannung leicht ein. Diese Information über einen eventuellen *voltage drop* kann in einem Regler verwendet werden, der auf diesen Einbruch reagieren kann. Ein solches Reglermodell ist einer der Hauptzwecke, für den der Fast Power Simulator entwickelt werden sollte.

Abbildung 4.4 zeigt eine Leistungskurve mit einem Zeitintervall von 100 ns. Man erkennt gut, dass der Chip ab ca. 75 μs Simulationszeit zu arbeiten beginnt und die Leistungsaufnahme dadurch sprunghaft ansteigt. Nach der ersten Spitze pendelt sich die Leistung auf einen relativ konstanten Wert ein, unterbrochen durch sehr kurze Leistungsspitzen im Abstand von ca. 20 μs. Wenn solche Spitzen zu groß werden, können diese einen Einbruch der Versorgungsspannung verursachen, da diese den geforderten Strom nicht mehr liefern kann.

In Abbildung 4.5 wird ein Chip während der Simulation in den Sleep-Mode versetzt. Das bedeutet, dass der IC in eine Art Ruhezustand gebracht und dadurch die Stromaufnahme deutlich gesenkt wird. Dieser Effekt kann in der Leistungskurve zwischen den Zeitpunkten 265 μs und 335 μs gut beobachtet werden. Durch diese Information ist es schon zu einem frühen Zeitpunkt der Entwicklung möglich, den Stromverbrauch im Sleep-Mode zu minimieren.

Durch die modulfeine Aufteilung ist es auch möglich, die Leistungskurven einzelner Module während der Simulation zu verfolgen. Im Beispiel aus Abbildung 4.6 kann man erkennen, dass im Modul “no_r_clf” zu zwei Zeitpunkten relativ große Stromspitzen auftreten, die in weiterer Folge eventuell einen Einbruch der Versorgungsspannung bewirken könnten.

Mit dem FPS wäre es nun auch möglich, das “no_r_clf”-Modul weiter aufzuschlüsseln

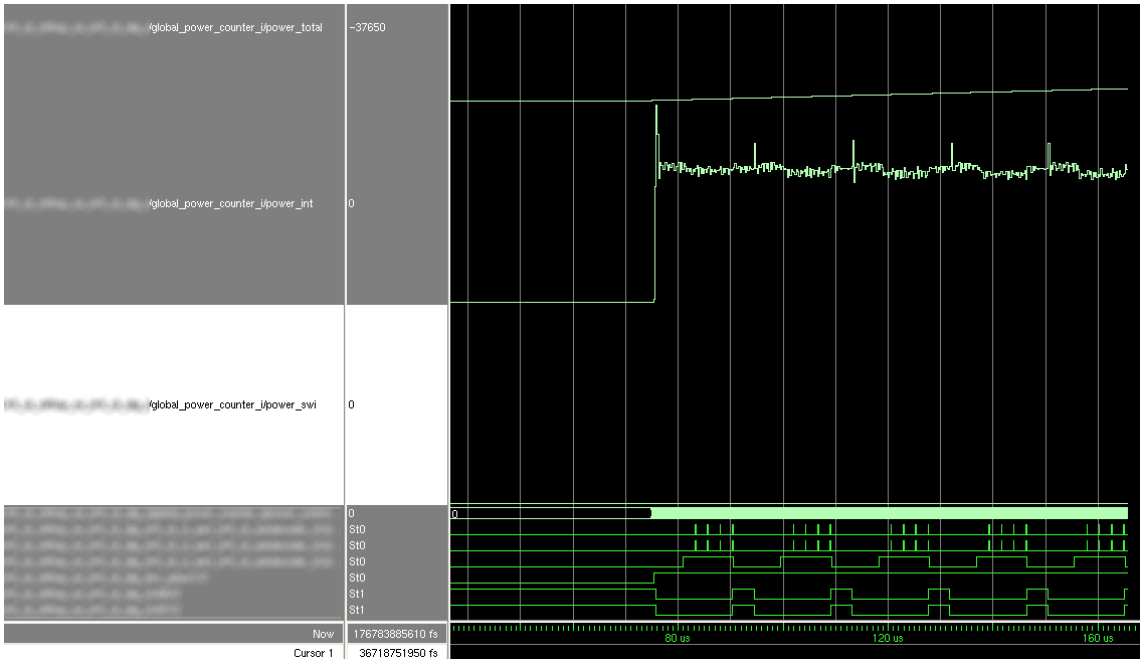

Abbildung 4.4: Energiezähler und angenäherte Leistung

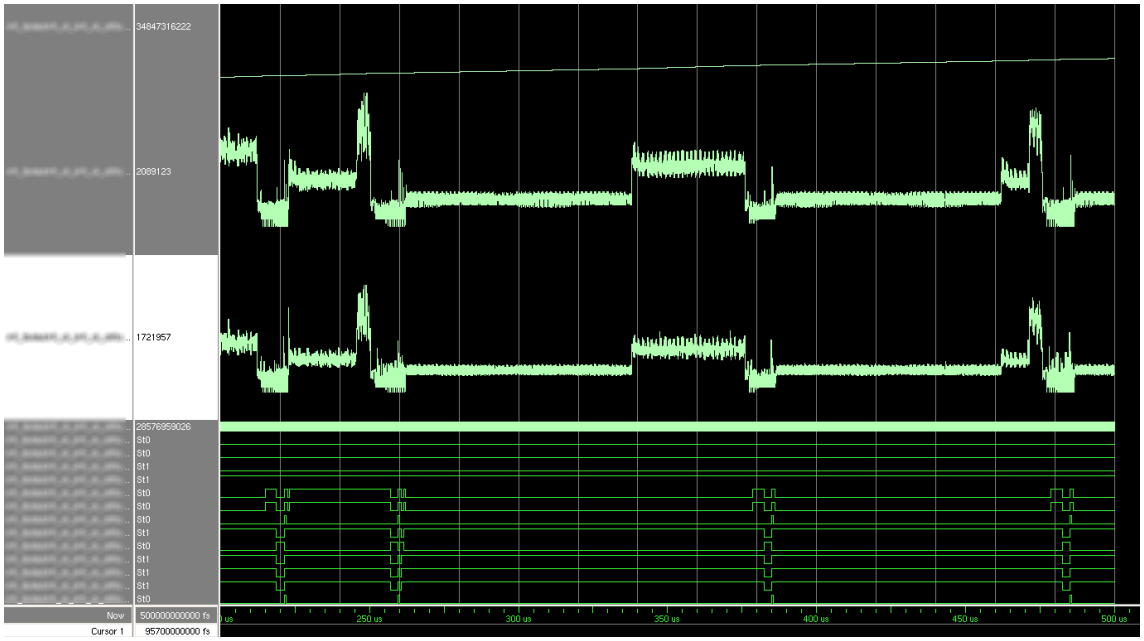

Abbildung 4.5: Leistungsaufnahme Sleep-Mode

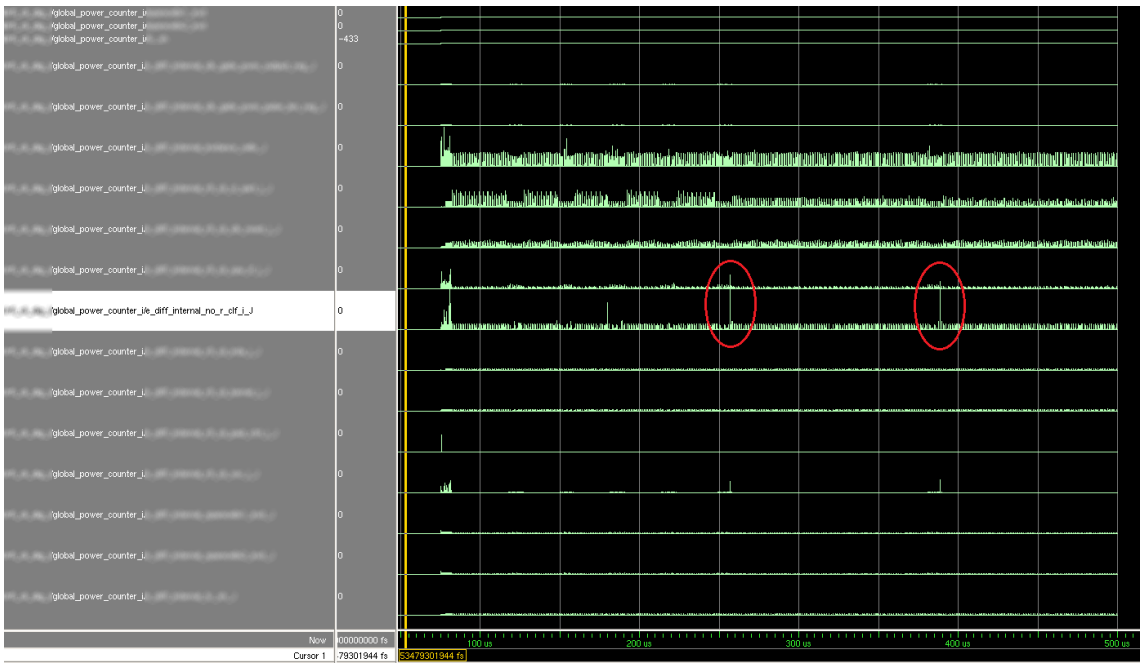

Abbildung 4.6: Leistungsaufnahme einzelner Module

(bis hin zu den einzelnen Standardzellen auf tiefster Ebene) um herauszufinden, welches Modul bzw. welche Zelle für diesen hohen Strom verantwortlich ist.

4.3 Laufzeit und Genauigkeit

Dadurch, dass bei jedem Signalwechsel an jeder Zelle ein Wert ausgesucht und aufsummiert werden muss, ist ein deutlicher Anstieg der Simulationszeit erkennbar. Neben anderen Faktoren hängt diese Zeit auch signifikant von der Tiefe der Analyse ab. Das bedeutet, dass eine reine Simulation des gesamten Chips deutlich schneller abläuft, als eine Aufschlüsselung in einzelne Module. Dieser Anstieg der Simulationszeit wird aber in Anbetracht der Vorteile des FPS in Kauf genommen.

Durch die Variable *enable* kann der Power Simulator auch erst zu einem späteren Zeitpunkt aktiviert werden, sofern die Leistungsaufnahme nur in einem bestimmten Bereich der Simulation von Interesse ist. Dadurch kann in den Phasen, in denen der FPS nicht gebraucht wird, die Simulation wieder deutlich beschleunigt werden. Je nach Tiefe und Komplexität im Bereich der Leistungsberechnung der Power-Simulation kann mit einem Deaktivieren der FPS-Zähler die Simulationszeit um 40-60% verringert werden.

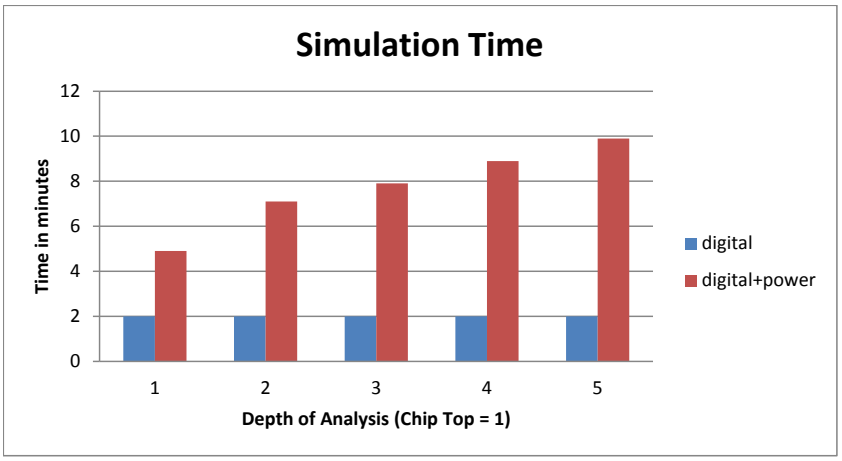

Abbildung 4.7: Dauer der Simulation mit steigender Simulationstiefe

Die Genauigkeit der Ergebnisse wurde im Laufe der Entwicklung durch verschiedene Schritte immer weiter erhöht (Abbildung 4.8):

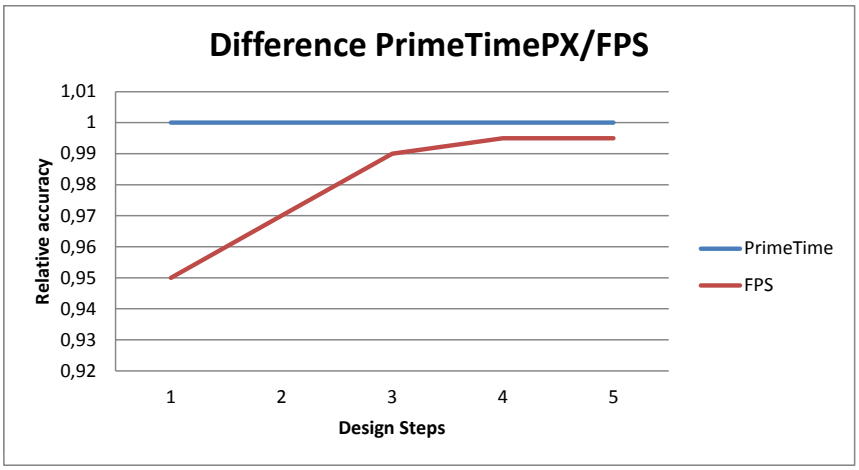

Abbildung 4.8: Genauigkeit der Simulation (Referenz: PrimeTimePX)

1. Erste Simulationen
2. Verbesserte Interpolation bzw. Extrapolation
3. Erhöhung der internen Genauigkeit von PrimeTime
4. Unterscheidung Rise/Fall-Kapazitäten
5. Verbesserungen im Tcl-Skript für PrimeTime

Als Referenz wurden für verschiedene Simulationen VCD-Dateien erstellt, welche dann mit PrimeTimePX verarbeitet wurden. Die für verschiedene Corner generierten Reports wurden anschließend mit den Simulationsergebnissen des FPS verglichen. Für den Gesamt-Verbrauch eines Chips wurde so eine durchschnittliche Genauigkeit von ca. 0,5% erreicht. Der Wert für die Internal Power hatte im Schnitt mit ca. 1% die größte Abweichung, die Switching und Leakage Power erreichten eine Genauigkeit von ca. 0,1 bis 0,2%

Für einzelne Module wurde teilweise eine deutlich höhere Abweichung festgestellt als für den gesamten Leistungsverbrauch des Chips.

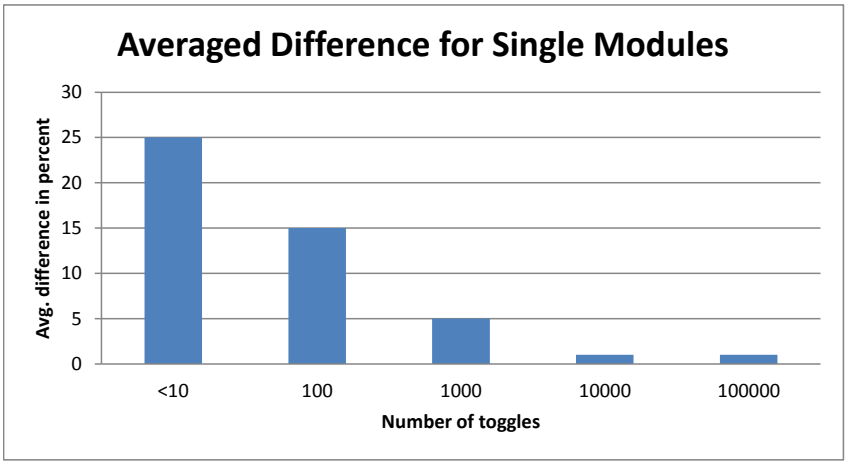

Abbildung 4.9: Durchschnittliche Abweichung einzelner Module

Abbildung 4.9 zeigt die durchschnittliche Abweichung einzelner Module, gruppiert nach Anzahl der Signalübergänge in der Simulation. Hierbei wird deutlich, dass die Abweichung für eine geringere Anzahl an Flanken mit ca. 25% relativ groß ist, diese jedoch mit steigender Anzahl an Übergängen rapide abnimmt. Grundsätzlich gilt also: Je weniger Übergänge auftreten, desto ungenauer ist der FPS-Wert im Vergleich zum Wert aus PrimeTimePX.

4.4 Weiterführende Analysen

Durch die Aufteilung aller Energien in Internal und Switching Power, sowie weitere mögliche Unterteilungen wie z.B. in Clock-Tree und Non-Clock-Tree Zellen, können mit dem FPS diverse Analysen und Simulationen durchgeführt werden.

Eine hilfreiche Analyse zur Bestimmung der Stabilität des Chips ist die Simulation der eigentlichen Stützkapazität des Chips. Diese setzt sich im Prinzip aus den einzelnen

Lastkapazitäten an jenen Knoten zusammen, die den logischen Zustand 1 besitzen. An diesen Knoten ist die Lastkapazität geladen und es kann aus ihr ein eventueller Strombedarf bei einem Signalwechsel abgedeckt werden. Die Bestimmung der gesamten Kapazität aller Knoten kann mit Hilfe der übergebenen Switching Energy und der Spannung leicht durchgeführt werden. Das Prinzip ist, bei einer positiven Flanke an einem Ausgang einen globalen Zähler mit dem Wert der Kapazität an diesem Ausgang zu erhöhen, bei einer negativen zu verringern. Diese Schritte werden im power_task durchgeführt. Damit kann der Verlauf der Stützkapazität während der Simulation durchgehend verfolgt werden.

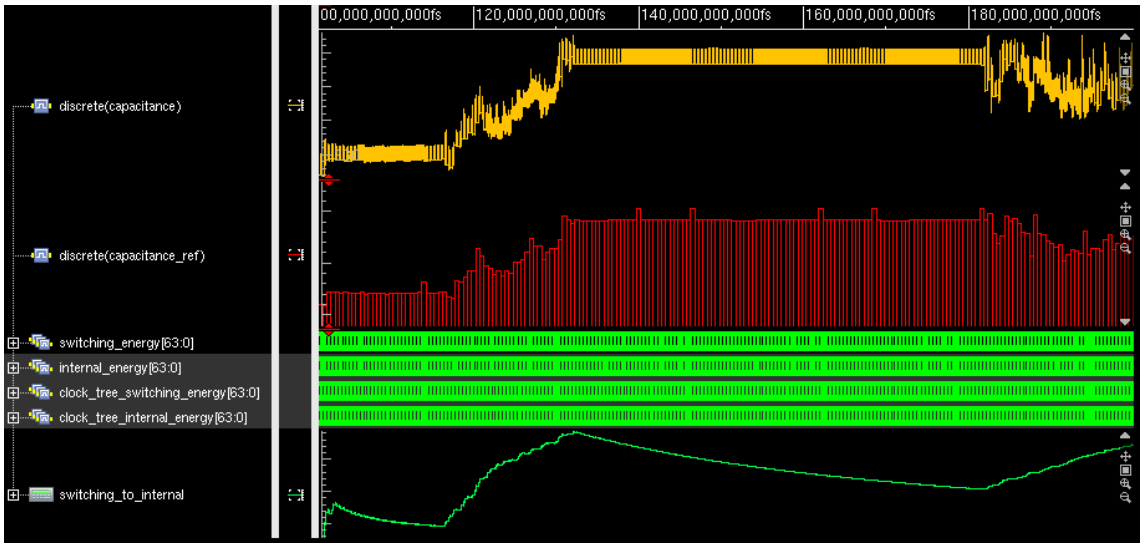

Abbildung 4.10: Verlauf der eigentlichen Stützkapazität während der Simulation

Der erste Verlauf in Abbildung 4.10 ist die Stützkapazität, die mit der vorher beschriebenen Methode ermittelt wird. Dabei wird bei jeder Pin-Änderung an einem Ausgang entweder ein Wert addiert oder subtrahiert. Die Kurve ist also kontinuierlich. Um diesen Wert zu verifizieren, wurde in jedes Zell-Modell ein weiterer always-Block eingefügt, der in einem bestimmten Zeit-Intervall die Zustände der vorhandenen Ausgangs-Pins abfragt und im Falle eines logisch-1-Zustandes einen weiteren globalen Zähler um den entsprechenden Kapazitätswert erhöht. Dieser Referenz-Zähler ist das zweite Signal in Abbildung 4.10. Man erkennt deutlich, dass die beiden Verläufe der Signale gleich sind, die korrekte Funktionsweise des kontinuierlichen Zählers wurde damit bestätigt. Weiters sieht man in dieser Simulation das Verhältnis von Switching zu Internal Power (unterstes Signal). Hierbei wird die Korrelation zwischen Switching Power und dem Verlauf der Stützkapazität deutlich. In den Bereichen, in denen sich viele Ausgänge ändern, gibt es eine hohe Switching Power und gleichzeitig große Änderungen im Verlauf der Kapazität.

Um einen Startwert für diesen kontinuierlichen Kapazitäts-Zähler zu bekommen, werden zu Beginn alle Initialisierungen der Ausgänge auf logisch 1 ausgewertet und deren Kapazitäten aufsummiert (Abbildung 4.11). Übergänge vom undefinierten Wert x auf logisch 0 werden bei der Initialisierung nicht berücksichtigt, da es sich dabei nicht um wirkliche negative Flanken von 1 auf 0 handelt und somit auch noch keine Ladung umgeladen wird.

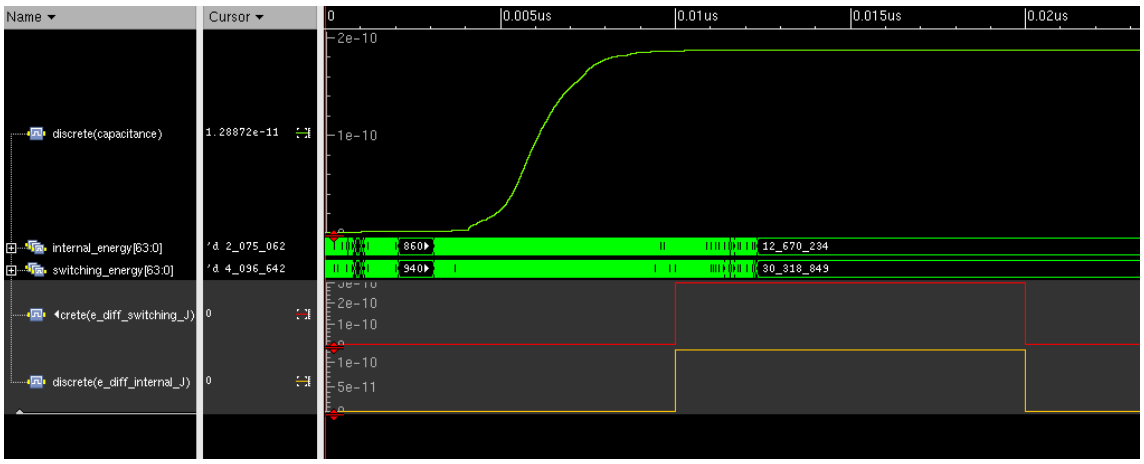

Abbildung 4.11: Aufsummieren der Kapazitäten aller Knoten mit Zustand 1

In einem weiteren Schritt wird dieser Kapazitätswert mit der Variable *enable* gekoppelt. Nur wenn diese auf dem logischen Zustand 1 ist, werden Kapazitäts-Werte hinzugefügt oder abgezogen. Um auch in diesem Fall einen korrekten Startwert zu bekommen, wird in das funktionale Verilog-Modell jeder Zelle ein zusätzlicher Block eingefügt.

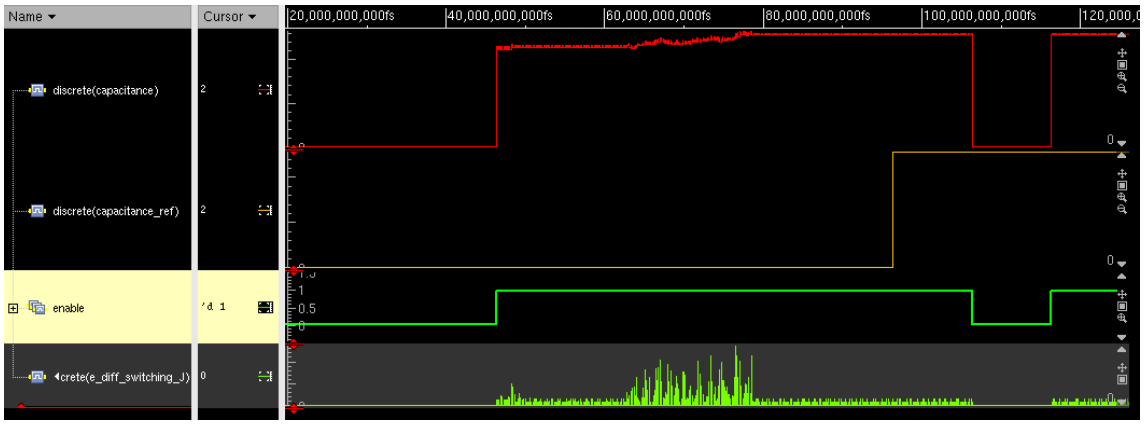

Abbildung 4.12: Kapazitätszähler mit *enable*-Signal

Hier wird auf die steigende Flanke von *enable* reagiert. Zum Zeitpunkt des Übergangs von 0 auf 1 berechnet jede Zelle seine Kapazitäten an den Ausgängen aus den übergebenen Switching-Energien und die logischen Zustände der Ausgänge werden abgefragt. Wenn ein Ausgang auf 1 liegt, wird die vorher berechnete Kapazität dem globalen Zähler *capacitance* hinzuaddiert. Da alle Zellen auf die *enable*-Flanke reagieren und ihre Kapazitäten in die globale Variable schreiben, bekommt man zu diesem Zeitpunkt den genauen Kapazitätswert über die komplette Schaltung. Dieses Verhalten ist in Abbildung 4.12 gut zu erkennen. Bei einer fallenden Flanke von *enable* wird der Wert resetiert.

Eine weitere mögliche Untersuchung wäre z.B. das Verhältnis von Flankenanzahl zu verbrauchter Leistung. Dadurch kann leicht ermittelt werden, an welchen Leitungen eine Kapazitäts-Reduktion am meisten Energieersparnis bringen würde. Wie in Kapitel 2.3.1 beschrieben, führt eine kleine Lastkapazität aber auch zu einem höheren internen Stromverbrauch der Zelle. Um einen optimalen Kapazitätswert zu finden kann daher auch das Verhältnis von Internal zu Switching Power mit Hinblick auf die Anzahl der Flanken an Ein- und Ausgängen herangezogen werden. Sollte es an einer Zelle viele Flanken an den Eingängen, aber nur sehr wenige Übergänge an den Ausgängen geben, würde eine höhere Lastkapazität trotz dem damit verbundenen Ansteigen der Switching Power womöglich eine noch größere Einsparung im Bereich der Internal Power bringen.

Alle diese Analysen sind mit dem FPS sehr leicht durchführbar und auch beliebig erweiterbar, da alle Variablen global verfügbar sind und die Umgebung ein leichtes Hinzufügen von neuen Registern und Berechnungen erlaubt.

5 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, innerhalb einer herkömmlichen Digitalsimulation auch den aktuellen Energie- bzw. Leistungsverbrauch zu simulieren. Bisher konnte der genaue Leistungsverbrauch von Chips und deren Modulen nur nach einer vollständig absolvierten Digitalsimulation durchgeführt werden. Dazu wurden spezielle Programme wie PrimeTimePX verwendet, welche aus den Signalübergängen in der Simulation und den Power-Models in den jeweiligen Bibliotheken der Zellen die Leistung berechnet. Dadurch, dass diese Leistungs-Abschätzung aber erst nach einer fertigen Digitalsimulation durchgeführt werden kann, konnten verschiedene Mechanismen, die auf einen zu hohen Stromverbrauch reagieren (z.B. Clock Gating, Dynamic Voltage Scaling, etc.) nur unzureichend simuliert werden.

Mit dem Fast Power Simulator wurde es nun möglich, eine sehr genaue und zeitlich fein aufgelöste Simulation des aktuellen Leistungs- bzw. Stromverbrauchs während der normalen Digitalsimulation durchzuführen. Das wurde dadurch realisiert, dass bereits vor der eigentlichen Simulation alle auftretenden Energien mit einem Perl-Skript berechnet und in die Netzliste als Parameter eingefügt wurden. Diese Energien wurden mit Hilfe der Lastkapazitäten an den Knoten, sowie den Eingangs-Übergangszeiten berechnet. Um diese Energiewerte dann während der Simulation verarbeiten zu können, wurden auch die funktionalen Verilog-Modelle aller Standardzellen adaptiert und mit Abfragen und Berechnungen erweitert. Neben dem Energieverbrauch des gesamten Chips können auf Wunsch auch die einzelnen Hierarchie-Ebenen aufgeschlüsselt und simuliert werden. Es ist auch möglich, in einer externen Datei Pfade zu bestimmen Modulen anzugeben, welche in der Simulation berücksichtigt werden sollen. Weiters besitzt auch jede einzelne Zelle eigene Zähler, die nur die Energien, welche die Zelle selbst verbraucht, aufsummieren. Damit kann eine lückenlose Aufschlüsselung aller im Design enthaltenen Module und Zellen durchgeführt werden. Alle genannten Energien sind in Internal und Switching Power aufgetrennt, damit auch hier genaue Untersuchungen durchgeführt werden können. Die Zähler sind als globale Variablen in einem eigenen Modul definiert, wodurch alle anderen Module und Zellen leicht auf diese zugreifen können. Damit ist es auch möglich, ein Regler-Modell in einem Modul zu implementieren, welches auf einen zu hohen Strom im Chip und damit auf einen eventuellen Einbruch der Versorgungsspannung (*voltage drop*) reagieren und diesen ausregeln kann. Durch den Aufbau des Fast Power Simulators sind auch weitere Analysen wie z.B. das Verhältnis von steigenden zu fallenden Flanken oder das Verhältnis von Switching Power zu Flankenanzahl leicht durchzuführen und

auch beliebig zu erweitern. Die Genauigkeit im Vergleich zum bisherigen Verfahren mit PrimeTimePX liegt im Bereich von ca. 0,5% für den gesamten Chip. Die Laufzeit verlängert sich im Vergleich zu einer reinen Digitalsimulation je nach Einstellungen des FPS um den Faktor 3 bis 5. Die verlängerte Laufzeit ist im Vergleich zu den Vorteilen und Möglichkeiten, die der Fast Power Simulator bringt, aber von geringer Bedeutung.

Weiters diente diese Arbeit auch als Machbarkeitsstudie für eine weiterführende Entwicklung des Fast Power Simulators, welche mit externen C-Funktionen, die mit Hilfe des von Verilog zur Verfügung gestellten *Program Language Interfaces (PLI)* in die Simulation eingebunden werden, das Patchen der Netzliste hinfällig macht. Bei dieser Methode werden die Energiewerte im Vorhinein mit PrimeTimePX berechnet und in einer externen Datei abgespeichert. Diese Datei wird dann während der Simulation mit Hilfe des C-Codes von den Modulen eingelesen und entsprechend den aktuell vorherrschenden Pin-Zuständen aufsummiert. Die neu entwickelte Methode bietet den Vorteil, dass die originale Netzliste nicht mehr modifiziert werden muss, allerdings braucht man für das Berechnen der Werte die PX-Erweiterung von PrimeTime, welche bei dem in der Arbeit gezeigten Verfahren nicht notwendig ist. Erste Versuche mit der neuen Methode haben gezeigt, dass die Laufzeit und die Genauigkeit bei beiden Ansätzen fast ident ist. Zusätzlich wird auch an genauen Stromverbrauchs-Modellen von Makros gearbeitet, um auch diese in die Power Simulation mit einzubeziehen.

Das neue Verfahren wird nach Fertigstellung in den Standard-Workflow bei Infineon integriert und in Zukunft für Power-Simulationen und andere Analysen verwendet werden.

Literaturverzeichnis

[1] Agrawal, V. D. und S. Ravi: Low-Power Design and Test. online. http://www.eng.auburn.edu/~vagrawal/COURSE/SUM_07_HYD/lp_hyd_2.ppt.

[2] Burd, T. D., T. A. Pering, A. J. Stratakos und R. W. Brodersen: Dynamic Voltage Scaled Microprocessor System. IEEE Journal of Solid-State Circuits, Vol. 35, No. 11, 2000.

[3] Chandrakasan, A. P. und R. W. Brodersen: Minimizing power consumption in digital CMOS circuits. Proceedings of the IEEE, Vol. 83, No. 4, 1995.

[4] Chandrakasan, A. P., S. Sheng und R. W. Brodersen: Low-Power CMOS Digital. IEEE Journal of Solid-State Circuits, Vol. 27, No. 4, 1992.

[5] Eggenberger, O.: Halbleiter-Bauelemente. online, 2012. Universität Stuttgart. http://www.iris.uni-stuttgart.de/lehre/eggenberger/eti/09_Halbleiter/Kennlinien.htm.

[6] Emnett, F. und M. Biegel: Power Reduction Through RTL Clock Gating. Synopsys Users Group Conference, San Jose, 2000.

[7] Haider, M. R.: CMOS Inverter Power Dissipation. online, 2009. http://www.sonoma.edu/users/h/haider/courses/ces522_vlsi/lecture05.pdf.

[8] Knowles, K. M. und C. Dunleavy: Introduction to Semiconductors. online, 2006. DoITPoMS - University of Cambridge. http://www.doitpoms.ac.uk/tlplib/semiconductors/mosfet.php.

[9] Nordmann, A.: Schaltsymbole von MISFET-Transistoren. online, 2008. http://upload.wikimedia.org/wikipedia/commons/thumb/2/2f/MISFET-Transistor_Symbole.svg/450px-MISFET-Transistor_Symbole.svg.png.

[10] Pedram, M.: Minimizing Leakage Power in CMOS: Technology Issues. online, 2008. Dept. of Electrical Engineering University of Southern California. http://atrak.usc.edu/~massoud/Talks/Pedram-LeakageTutorial-epfl08.pdf.

[11] Rabaey, J. M.: Low Power Design in CMOS. online, 1995. http://bwrc.eecs.berkeley.edu/classes/icbook/slides/slides4a.pdf.

[12] Rao, R., A. Srivastava, D. Blaauw und D. Sylvester: Statistical Analysis of Subthreshold Leakage Current for VLSI Circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 2, 2004.

[13] Sarwar, A.: CMOS Power Consumption and Cpd Calculation. online, 1997. Texas Instruments. http://www.ti.com/lit/an/scaa035b/scaa035b.pdf.

[14] Weste, N. H. E. und K. Eshraghian: Principles of CMOS VLSI Design. Addison Wesley, 1993.

[15] Wilson, B.: CMOS Logic. online, 2008. Connexions. http://cnx.org/content/m1029/latest/.