Studienarbeit

Messung des Energieverbrauchs von Caches am Beispiel des StrongARM-Prozessors

Gregory Sapsford

10. Juli 2001

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mobile und eingebettete Systeme werden zunehmend mit leistungsfähigen Prozessoren ausgestattet. Diese Systeme werden unter anderem in der Telekommunikation eingesetzt.

In Mobiltelefonen werden durch den wachsenden Funktionsumfang zunehmend leistungsfähigere Prozessoren eingesetzt, so dass sie unter anderem auch als Organizer und Internetzugang Verwendung finden. Eine der wichtigsten Forderungen an diese Systeme ist eine möglichst lange, vom Stromnetz unabhängige Bereitschaftszeit, die jedoch nicht immer die der Herstellerangaben erreicht. Die Gründe hierfür liegen vor allem im Nutzungsprofil des jeweiligen Anwenders. So wird die Standby-Zeit unter anderem durch häufigere Verwendung des integrierten Organizers oder lange Telefonate drastisch reduziert, da in der Regel nur Mustertelefonate und die Empfangsbereitschaft in die Berechnung der Standby-Zeit einfließen.

Damit ein System hinsichtlich des Energieverbrauchs optimiert werden kann, muss zunächst ermittelt werden, wie hoch der Energiebedarf der einzelnen Komponenten ist. Wenn sich dieser Energiebedarf in einen hardware- und einen softwareabhängigen Teil trennen lässt, können an beiden Stellen Optimierungen durchgeführt werden.

Für den Fall der softwareabhängigen Energieoptimierung wird an der Universität Dortmund am Fachbereich Informatik, Lehrstuhl 12, ein Compiler für die Hochsprache C entwickelt, der aus Programmen, die in C entworfen wurden, einen energieoptimierten Maschinencode generieren kann. Die Forschung konzentriert sich auf eine spezielle Familie von RISC-Prozessoren, die für geringen Energiebedarf entwickelt wurden und vorwiegend in eingebetteten Systemen eingesetzt werden. Für den Compiler ist es unter anderem wichtig zu wissen, wieviel Energie der Cache des Prozessors benötigt, um energieoptimierte Speicherbelegungen und Instruktionsplatzierungen zu erreichen. Diese Studienarbeit wird sich mit diesem Thema beschäftigen und den Einfluss der Cachebenutzung auf den Energieverbrauch untersuchen.

1.1 Motivation

Die Prozessorarchitekturen vieler Hersteller sind nur anhand von Datenblättern mit Blockschaltbildern und groben Übersichten erhältlich, da aufgrund des Urheberrechts und zum Schutz des eigenen Know-Hows hardwarenahe Architekturbeschreibungen in VERILOG oder VHDL nicht veröffentlicht werden. Somit sind Simulationen, die Energieverbrauchswerte für den generierten Code liefern könnten, nicht möglich, weil sie auf diesen Beschreibungen basieren.

Aufgrund dieser Tatsache wurde versucht, mit Hilfe des Maschinencodes eines Prozessors zu ermitteln, welchen Einfluss die Verwendung des Caches auf den Energieverbrauch des Prozessors hat. Die Ergebnisse sollen zukünftig in die Entwicklung des Low-Power-Compilers einfließen.

1.2 Ähnliche Arbeiten

Theokaridis [THE00] hat den Energieverbrauch von Prozessor-Instruktionen unter Einbeziehung der Energiekosten des externen Speichers untersucht. Zwei wichtige Ergebnisse sind, dass auf externe Speicherzugriffe nach Möglichkeit verzichtet werden soll, da dort die meisten Energiekosten anfallen, und dass bestimmte Befehlsgruppen möglichst vermieden werden sollen, da sie die Energieaufnahme des Prozessors erhöhen.

In seiner Diplomarbeit hat Theokharidis den ARM7TDMI zur Messung der Energiekosten von Instruktionen und den Seiteneffekten wie Inter-Instruction-Effekte und Speicherzugriffskosten verwendet.

1.3 Ziel dieser Arbeit

Das Ziel dieser Arbeit ist es, das Energiemodell von Theokharidis um den Energiekostenanteil des Caches zu erweitern. Da der ARM7TDMI, den er verwendet hat, keinen Cache besitzt, wird für diese Studienarbeit ein StrongARM SA1100 eingesetzt. Die Messmodule werden auf die Anforderungen der Energiekosten des Caches angepasst. Mit ihnen sollen Energiemessungen durchgeführt und diese anschliessend bewertet werden.

Kapitel 2

Einführung in Low Power

„Low Power" bezeichnet ein Gebiet, in dem sowohl hardware- als auch softwareseitig Methoden zur Energieoptimierung von Systemen erforscht werden. Besonders in Systemen, in denen ein geringer Energieverbrauch eine Voraussetzung für deren Einsatz ist, sind die Ergebnisse interessant. Diese Systeme zeichnen sich unter anderem durch folgende Eigenschaften aus:

- Mobile Geräte: Bei akkubetriebenen Geräten ist die Betriebsdauer ein wichtiges Kriterium. Diese wird durch die Akkukapazität begrenzt und kann durch optimierten Energieverbrauch verlängert werden.
- Hohe Wärmeentwicklung: Hohe Energieaufnahme geht mit einer hohen Wärmeentwicklung einher. Um diese abzuführen, werden Kühlkörper eingesetzt. Aufgrund des begrenzten Raumes in eingebetteten Systemen sind dort dann aktive Kühlmaßnahmen erforderlich, die ihrerseits ebenfalls Energie verbrauchen. Hier kann durch Optimierung das Kühlsystem verkleinert oder gänzlich unnötig werden.
- Hohe Zuverlässigkeit: Die Lebensdauer von Halbleiterbausteinen reduziert sich durch erhöhte Temperaturen und Stromstärken. Diese kann durch Energieoptimierung verlängert werden.

2.1 Elektrotechnische Grundlagen

Zunächst werden die elektrischen Größen Leistung und Energie definiert, die als Grundlage der Optimierungsmethoden im Low Power-Bereich dienen. Sie wurden aus [FDL89][HMS92] übernommen.

2.1.1 Leistung

Aus der Spannung U und dem Strom I errechnet sich die elektrische Leistung P allgemein nach:

$$P = U \cdot I$$

P hat die Einheit $1W\,(Watt) = 1V\,(Volt) \cdot 1A\,(Ampere)$. Mit der Betriebsspannung $V_{dd} = U$ ist daher die elektrische Leistung eines Prozessors:

$$P = V_{dd} \cdot I$$

2.1.2 Energie

Die verbrauchte Energie E, oder die Arbeit, eines Systems errechnet sich aus der Leistung P, die während Dauer T aufgenommen wurde.

$$E = P \cdot T$$

E hat die Einheit $1J\,(Joule) = 1Ws = 1VAs$. Die Zeit T in Sekunden entspricht der Ausführungszeit eines Programmes auf einem Prozessor. Sie setzt sich aus der Zahl N aller Taktzyklen, die zum Durchlaufen eines Programms nötig sind, und der Dauer t eines Taktzyklus zusammen:

$$T = N \cdot t$$

Daraus ergibt sich die Definition des Energieverbrauchs eines Programms nach:

$$E = V_{dd} \cdot I \cdot N \cdot t$$

Nimmt man die Versorgungsspannung V_{dd} und die Länge eines Taktzyklus t als konstant an, so hängt die aufgenommene Energie proportional von der Zyklenzahl N und dem Strom I ab.

2.2 Energieverbrauch aktiver Schaltungen

Die Energiekosten der in CMOS-Technologie realisierten Schaltkreise eines Prozessors setzen sich aus drei verschiedenen Typen zusammen [SYN96]:

1. **Switching Power**

   Switching Power bezeichnet die Energie, die zum Umladen der Lastkapazitäten benötigt wird. Die Umschaltleistung P_c hat bei aktiven CMOS-Zellen einen Anteil von 70 bis 90 Prozent an den Gesamtenergiekosten und errechnet sich aus:

$$P_c = \frac{V_{dd}^2}{2} \sum_{\forall nets(i)} (C_{Load_i} \cdot TR_i)$$

   C_{Load_i} stellt dabei die gesamte kapazitive Last des Netzes i dar, das von der Zelle angesteuert wird, TR_i bezeichnet die Umschaltrate des Netzes i, also die Umladungen pro Sekunde.

2. **Short Circuit Power**

   Wird in der Schaltung in Bild 2.1 das Eingangssignal von niedrigem zu hohem Potential umgeschaltet, dann schaltet sich der N-Kanal Transistor ein und der P-Kanal Transistor aus. Während dieses Zustandswechsels leiten für einen kurzen Moment beide Transistoren und ein Kurzschlussstrom I_{SC} fließt zwischen V_{CC} und GND, der die Kurzschluss-Verlustleistung P_{SC} verursacht. In Schaltungen mit kurzen Umschaltzeiten ist diese Kurzschlussenergie gering, bei Schaltungen mit längeren Umschaltzeiten kann diese Energie allerdings bis zu 30 Prozent des gesamten Energieverbrauchs ausmachen.

$$P_{SC} = f[C_{Load}, WeightAvg_{(Trans)}] \cdot TR_i$$

$WeightAvg_{(Trans)}$ist dabei die durchschnittliche Umschaltzeit des Ausganges.

3. Leakage Power

Auch im statischen Zustand, wenn die CMOS-Zelle nicht schaltet, fliesst ein Strom, der sogenannte Leckstrom, und verursacht einen gewissen Energieverbrauch.

$$P_{LeakageTotal} = \sum_{\forall cells(i)} (P_{CellLeakage_i})$$

$P_{CellLeakage_i}$ bezeichnet die Leckleistung einer inaktiven Zelle. In Schaltungen, die die meiste Zeit aktiv sind, macht die Leckenergie weniger als ein Prozent des gesamten Energieverbrauchs aus. Für Schaltungen, die sich hauptsächlich im statischen Zustand befinden, ist es jedoch wichtig, auch die Leckenergie zu berücksichtigen.

In Abbildung 2.1 sind die Ströme zu erkennen, die den Leckstrom I_{LK}, Kurzschlussstrom I_{SC} und den Umladestrom I_{SW} verursachen.
Durch optimierte Algorithmen und Architekturen kann die Schalthäufigkeit von Gattern und damit der Energieverbrauch gesenkt werden.
Weitere hardwareseitige Ansätze zur Reduzierung der Energiekosten sind:

- Absenkung der Versorgungsspannung. Sie geht quadratisch in die Berechnung des Umschaltstromes ein.

- Abschalten unbenutzter Einheiten. Durch ein Powermanagement können ungenutzte Einheiten abgeschaltet werden.

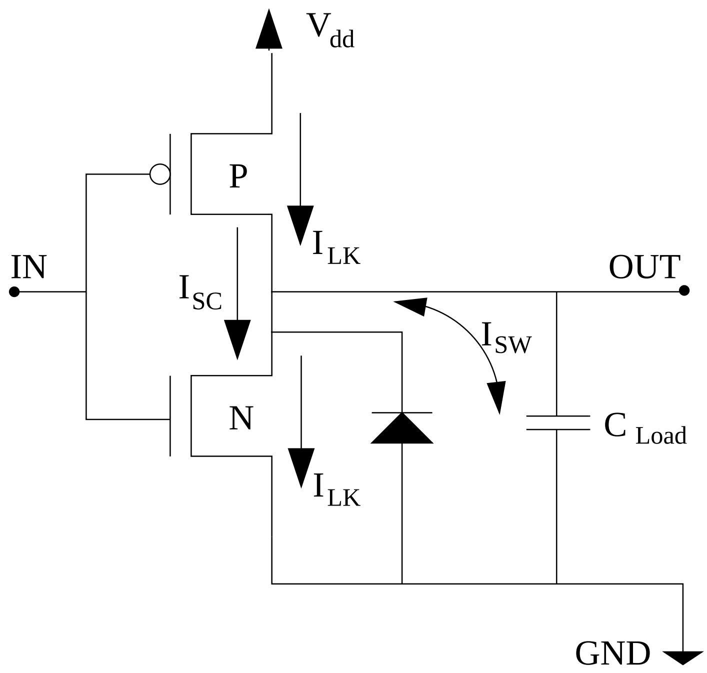

Abbildung 2.1: Ströme im CMOS Inverter [SYN96]

2.3 Energieoptimierter Compilerbau

Im Compilerbau besteht die Möglichkeit, durch Softwareoptimierungen den Energieverbrauch eines Programms zu reduzieren. Die Methoden können unter anderem sein:

-Daten neu berechnen

Da externe Speicherzugriffe sehr hohe Energiekosten verursachen, sollte der Compiler versuchen, benötigte Daten aus energetisch günstigeren Speicherbereichen zu verwenden oder nach Möglichkeit diese aus den vorhandenen Registerinhalten neu zu berechnen.

-Minimierung externer Speicherzugriffe

Durch Compileroptimierungen kann versucht werden, benötigte Daten oder Instruktionen in einen energietechnisch günstigeren Teil des Speichers zu verlegen. So könnten zum Beispiel kurze, häufig durchlaufene

Schleifen so platziert werden, dass diese aus einem Cache geladen werden können.

Kapitel 3

StrongARM SA1100-Prozessor

Für die Messung der Energieaufnahme des prozessorinternen Caches wurde der StrongARM SA1100-Prozessor ausgewählt. Der SA1100 ist ein hochintegrierter Mikrokontroller, der neben einem 32-Bit StrongARM-RISC-Prozessorkern eine System-Kontrolleinheit, mehrere Kommunikationskanäle, einen LCD-Kontroller, einen PCMCIA-Kontroller und universelle I/O-Ports hat. Wegen seiner hohen Verarbeitungsgeschwindigkeit von 180 MIPS und geringer Energieaufnahme von 430 mW bei 160 MHz [INT99] wird er in eingebetteten Systemen wie z. B. Handheld Computer oder Mobiltelefone eingebaut.

Neben dem eigentlichen StrongARM-Kern befinden sich ein Instruktionscache von 16 kByte, ein 8 kByte write-back Datencache und die jeweils dazugehörigen MMUs sowie ein Lese- und ein Schreib-Puffer im Prozessorkern.

3.1 Eigenschaften des SA1100

Der SA1100-Prozessor hat folgende Eigenschaften:

- 32-BIT-RISC CPU
- 31 physikalische 32-BIT-Register
- Load-/ Store-Architektur

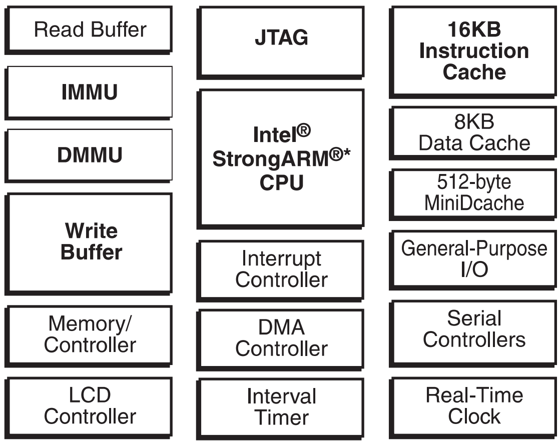

Abbildung 3.1: Eigenschaften des SA1100 [INT99]

- 16 kByte Instruktionscache
- 8 kByte write-back Datencache
- MMU
- 32-BIT-Daten- und 26-BIT-Adressbus
- USB-Schnittstelle
- 28 universelle I/O-Ports
- LCD-Controller
- PCMCIA-Controller
- Speicher-Controller
- JTAG-Schnittstelle

- UART-Controller
- USB-Controller
- IrDA-Controller

3.1.1 Caches

Die interne Verarbeitungsgeschwindigkeit des StrongARM und auch vieler anderer moderner Prozessoren ist wesentlich höher als die Datentransferrate zum externen Speicher. Das bedeutet, dass eine CPU bei einem Speicherzugriff relativ viele Wartezyklen einschieben muss, bis die benötigten Daten oder Befehle übertragen sind. Hier setzen Caches an, die, wenn sie auf dem selben Chip wie die CPU untergebracht sind, durch ihre kurzen Zugriffszeiten und hohen Datenraten die benötigten Daten wesentlich schneller zur CPU übertragen können. Die Größe eines Caches liegt, genau wie seine Lage in der Speicherhierarchie im System, in der Regel zwischen der des Hauptspeichers und der in der CPU, also den Registern [HP94]. In Tabelle 3.1 ist eine allgemeine Speicherhierarchie eingetragen. Da nur ein kleiner Teil des

Name	Realisierung	Zugriffszeit (ca.)	Gültig-Prüfung	Austausch durch
Tertiärspeicher	Bänder	$3 \cdot 10^{1}\,s$	BS	Operator
Sekundärspeicher	Platten	$1 \cdot 10^{-2}\,s$	BS	BS
(Platten-Cache)	DRAM	$1 \cdot 10^{-7}\,s$	BS	BS
Primärspeicher	DRAM	$5 \cdot 10^{-8}\,s$	HW	BS
Cache	SRAM	$1 \cdot 10^{-8}\,s$	HW	HW
Register	SRAM	$1 \cdot 10^{-9}\,s$	Compiler	Compiler

Tabelle 3.1: Allgemeine Speicherhierarchie [MAR00]

Bedeutung der Abkürzungen:

- BS: Betriebsystem
- HW: Hardware

Hauptspeichers im Cache abgelegt werden kann, sind verschiedene Strategien zur Datenübertragung zwischen dem Speicher und dem Cache entwickelt worden, die auf der Faustregel aufbauen, dass Programme 90% ihrer Zeit für 10% ihres Codes verbrauchen.

Cacheorganisation

Ein Cache ist in der Regel in verschiedene Blöcke unterteilt, damit er verschiedene Teile des Speichers vorhalten kann. Um einen Block aus dem Speicher im Cache platzieren zu können, müssen mehrere Speicherblöcke zusammen einem Block im Cache zugeordnet werden. Dies geschieht, indem der höherwertige Teil der Adresse die Blockadresse und der Rest die Block-Offset-Adresse angibt. Die Blockadresse bestimmt die Platzierung im Cache und hat eine Länge von $Adresse - \log_2(Blockgroesse)$. Wo genau ein Block im Cache zu finden ist, hängt von der Cache-Organisation ab:

- Bei einem vollassoziativen (fully associative) Cache kann jeder Block an jeder Stelle im Cache platziert werden.

- Werden innerhalb eines Caches mehrere Blöcke zu einer Gruppe zusammengefasst, so spricht man von einem mehrfach assoziativen Cache. Beim Laden eines Speicherblocks wird dieser zuerst einer Gruppe (Set) zugeordnet und anschließend erst in einem Block platziert. Ein Set errechnet sich gewöhnlich nach:

  $$Blockadresse\, modulo\, (Zahl\, der\, Sets\, im\, Cache)$$

  Bei n Blöcken im Set spricht man von n-fach assoziativem (n-way-set associative) Cache.

- Wenn jeder Block nur in einem bestimmten Cacheblock zugeordnet wird, nennt man ihn direct-mapped, oder einfach assoziativer Cache. Die Zuordnung (mapping) geschieht gewöhnlich nach:

  $$Blockadresse\, modulo\, (Anzahl\, der\, Cachebloecke)$$

Cachezugriffe

Bei einem Speicherzugriff wird zunächst geprüft, ob der entsprechende Block schon im Cache ist. Ist dies nicht der Fall (Fehlzugriff), muss dieser nachgeladen und der entsprechende Block im Cache ersetzt werden. Im einfachsten Fall, dem direct-mapped Cache, wird der Block direkt in den korrespondierenden Cacheblock geladen. Bei einem Fehlzugriff auf einen voll- oder mehrfach assoziativen Cache stehen mehrere Blöcke zur Auswahl, die über verschiedene Strategien ausgewählt werden:

- Zufall (random) - Der zu ersetzende Block wird zufällig ausgewählt.
- Am längsten ungenutzt (least recently used, LRU) - Hier wird der Block ersetzt, der am längsten nicht mehr angesprochen wurde. Das verringert die Gefahr, dass häufig verwendete Daten überschrieben werden.
- Zyklisch (round robin) - Die Blöcke innerhalb eines Sets werden zyklisch ersetzt.

Ein weiteres Unterscheidungsmerkmal ist der schreibende Zugriff. Die beiden grundlegenden Möglichkeiten sind:

- Write-through - Daten werden sowohl in den Block des Caches als auch im entsprechenden Block in den Speicher geschrieben.
- Write-back - Die Daten werden zunächst nur in den Cacheblock geschrieben. Der veränderte Block wird aber erst in den Speicher zurückgeschrieben, wenn er ersetzt wird.

3.1.2 Memory-Map

Der Speicherkontroller des SA1100 unterstützt ROM, Flash, DRAM, SRAM und PCMCIA Kontrollsignale. Sein Adressraum ist in 4 Partitionen zu je 1 GByte aufgeteilt [INT99]. Die Aufteilung ist in Abbildung 3.2 eingetragen.

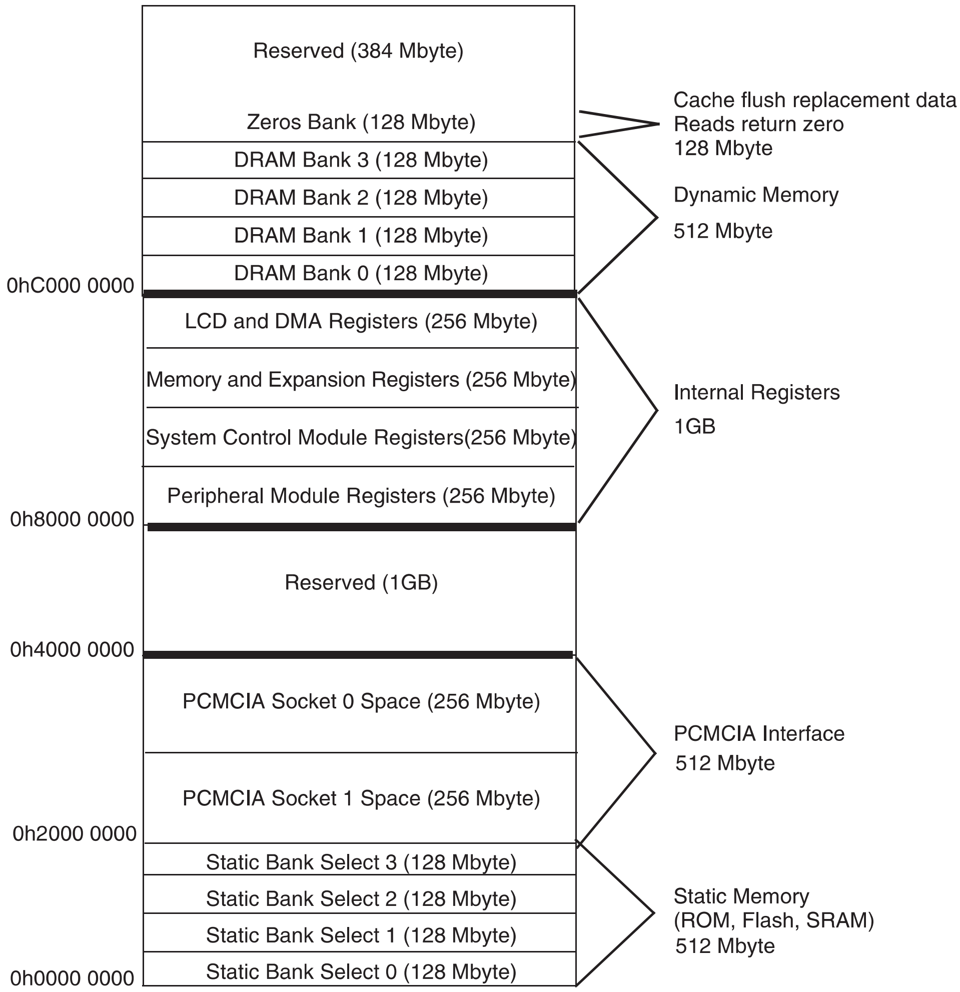

Abbildung 3.2: Memory-Map des SA1100 [INT99]

Daraus folgt, dass sich Programmcode und Daten in den folgenden Adressbereichen befinden können:

- Partition 1:
  Der Bereich von 0h0000 0000 bis 0h1FFF FFFF ist für statischen Speicher vorgesehen und für ROM, SRAM und Flash Speicher geeignet. Aus diesem Grund ist dieser Bereich vorwiegend für Bootcode und sonstige Programme geeignet.

- Partition 4:
  Zwischen 0hC000 0000 und 0hFFFF FFFF ist der Bereich für DRAM

Speicher. Hier können Programme und Daten abgelegt werden. Ebenfalls kann Programmcode aus der ersten Partition hierhin kopiert und ausgeführt werden, um den Programmablauf zu beschleunigen.

Kapitel 4

Energiemodell

Theokharidis hat in seiner Diplomarbeit [THE00] ein Energiemodell aufgestellt, das zur Messung der Energieaufnahme von Prozessor-Instruktionen dient.

4.1 Energiekosten

In Abbildung 4.1 sind diejenigen Energiekosten aufgeführt, die bei der Abarbeitung eines Programms entstehen.

Basiskosten

Die Basiskosten stellen die Energiekosten dar, die durch die Schaltkreisaktivitäten bei der Abarbeitung einer Instruktion verursacht werden.

$$E_{Base} = \sum_i (B_i \times N_i)$$

Die Basiskosten E_{Base} eines Programms errechnen sich aus den Basiskosten einer Instruktion B_i und der Anzahl ihrer Aufrufe N_i .

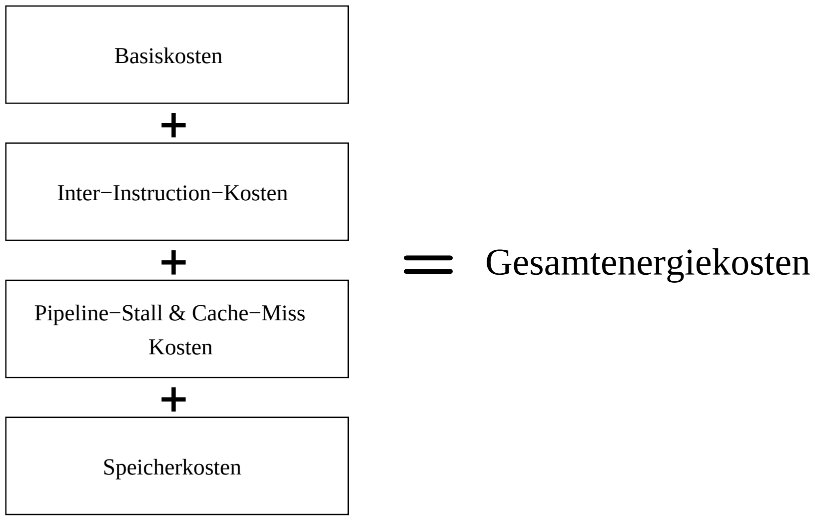

Abbildung 4.1: Energiemodell nach Theokharidis [THE00]

Inter-Instruction-Kosten

Neben den Basiskosten müssen auch die sogenannten Inter-Instruction-Kosten berücksichtigt werden. Diese entstehen zwischen zwei Befehlen, also beim Umschalten zwischen verschiedenen Ausführungseinheiten, die von den Instruktionen angesprochen werden und durch Änderung der Busbelegung. Dabei ist zu beachten, dass diese Kosten bei ähnlichen Instruktionen, die aufeinander folgen, wesentlich geringer ausfallen, als bei unterschiedlichen, da im letzten Fall die alte Einheit deaktiviert und eine andere Einheit aktiviert wird, im ersten Fall aber mit der gleichen Einheit weitergearbeitet werden kann. Desweiteren spielen hier auch die Schaltkreisaktivitäten auf dem Daten- und Adressbus eine Rolle, weil die Leitungen zum Teil neu belegt werden müssen. Je mehr Bits dabei umgeschaltet werden müssen, desto grösser ist dieser Effekt. Die Kosten errechnen sich aus den zusätzlichen Kosten $O_{i,j}$ eines Instruktionspaares i, j und der Häufigkeit $N_{i,j}$ ihres Auftretens:

$$E_{InterInstruktion} = \sum_{i,j} (O_{i,,j} \times N_{i,j})$$

Pipeline-Stalls und Cache-Misses

Im Gegensatz zu den beiden vorgenannten Energiekostenarten sind die durch Pipelines und Caches verursachten Kosten dynamisch vom Programmdurchlauf abhängig. Ein Fehlzugriff auf den Cache, ein Cache-Miss, führt zu einem teuren externen Speicherzugriff, da die benötigten Daten aus dem Speicher in den Cache nachgeladen werden. Für Pipeline-Stalls gibt es drei Gründe:

- Resource Hazard: Wenn verschiedene Instruktionen auf dieselbe Hardwareeinheit zugreifen müssen, dann wird die Pipeline angehalten, bis die entsprechende Einheit wieder zur Verfügung steht.
- Daten Hazard: Wenn eine Instruktion das Ergebnis einer noch nicht beendeten Instruktion benötigt, dann wird ein Pipeline-Stall ausgeführt.
- Kontroll Hazard: Nach Sprüngen tritt gewöhnlich ein Kontroll-Hazard auf, da aufgrund der Pipelinearchitektur die Adresse des Sprungzieles noch nicht vorliegt, wenn die nächste Instruktion gelesen werden soll.

Ein Pipeline-Stall bewirkt, dass wartende Befehle in ihrer jeweiligen Stufen bleiben und bis zur Bereinigung des Hazards Wartezyklen eingelegt werden.

$$E_{PipelineCache} = \sum_i (E_{Cache_i} + E_{Pipeline_i})$$

Die Energiekosten jedes Pipelinestalls $E_{Pipeline_i}$ und Cache-Misses E_{Cache_i} müssen berücksichtigt werden.

4.2 Cachekosten

Die Cachekosten werden durch die Abarbeitung der Befehle verursacht. Bei einem Cache-Miss ist jeweils zu berücksichtigen, dass nicht nur das angeforderte Datum oder die Instruktion geladen wird, sondern immer ein ganzer Cache-Block, in diesem Versuchsaufbau 32 Bytes.

4.2.1 Instruktionscache

Bei jedem Aufruf einer Instruktion wird auf den Instruktionscache zugegriffen, falls dieser eingeschaltet ist. Zu den Energiekosten, die dieser Aufruf verursacht, müssen bei einem Cache-Miss noch zusätzliche Kosten berücksichtigt werden. Die Energiekosten berechnen sich nach:

$$E_{ICache} = \sum_i (D_i \cdot N_i) + \sum_j (D_{Miss_j} \cdot N_j)$$

D_i bezeichnet die Kosten für den Aufruf einer Instruktion aus dem Cache und D_{Miss_i} die zusätzlichen Kosten, die bei einem Cache-Miss anfallen. N_i ist die Häufigkeit, mit der eine Instruktion i aufgerufen wird.

4.2.2 Datencache

Bei einem Aufruf von Load- und Store-Befehlen wird nicht nur auf den Instruktionscache zugegriffen, sondern diese Befehle führen ihrerseits einen Zugriff auf den Datencache aus. Bei der Energiebetrachtung dieser Befehle müssen daher neben den Kosten für einen Cachezugriff und einem eventuellen Cache-Miss im Instruktionscache auch die Energiekosten für den Zugriff auf den Datencache, mit einem möglichen Cache-Miss, berücksichtigt werden. Die Energiekosten des Datencaches berechnen sich nach:

$$E_{DCache} = \sum_i (Z_i \cdot N_i) + \sum_j (Z_{Miss_j} \cdot N_j)$$

Die Kosten für den Zugriff auf den Datencache werden mit Z_i bezeichnet, die zusätzlichen Kosten bei einem Cache-Miss mit Z_{Miss_i}. Die Häufigkeit des Auftretens wird mit N_i bezeichnet.

4.3 Energiemodell eines Prozessors

Das vollständige Energiemodell eines Prozessors zur Bestimmung des Energieverbrauchs eines Programms ergibt sich, wenn alle anfallenden Kostenarten aufsummiert werden.

$$E_{CPU} = \sum_i (B_i N_i) + \sum_{i,j} (O_{i,j} N_{i,j}) + \sum_i (E_{Pipeline_i} + E_{Cache_i}) + \sum_i (D_i \cdot N_i) +$$

$$\sum_j (D_{Miss_j} \cdot N_j) + \sum_i (Z_i \cdot N_i) + \sum_j (Z_{Miss_j} \cdot N_j)$$

4.4 Umsetzung des Energiemodells auf den SA1100

Aus dem aufgestellten Energiemodell werden jetzt Testmuster zur Messung der Energiekosten des SA1100-Caches auf dem gewählten Versuchsaufbau generiert. Dabei ist darauf zu achten, dass möglichst nur die Energiekosten des Caches variiert werden.

4.4.1 Wahl der Testmuster

Zur Ermittlung der Cachekosten werden Schleifen generiert, die sich aus Befehlen zusammensetzen, die über Speicherzugriffe den Cache direkt ansprechen können. Dazu zählen die Load- und Store-Befehle, die beide Caches ansprechen, und die Branch-Befehle, die ausschließlich den Instruktions-Cache ansprechen. Bei den Load- und Store-Befehlen ist darauf zu achten, dass diese entweder aus dem Instruktions-Cache oder direkt aus dem Hauptspeicher abgearbeitet werden und daher in beiden Caches Kosten verursachen können. Branch-Befehle hingegen verursachen keine Kosten im Datencache, da sie nicht auf Daten zugreifen, sondern nur Verzweigungen im Programm bewirken.

4.4.2 Berechnung der Energiekosten

Wie in Kapitel 5.1.1 bereits angedeutet, lässt sich die Stromaufnahme des Caches nicht isoliert betrachten, weil er mit dem ARM-Kern zusammen auf dem Chip liegt. Daher sind die berechneten Energiekosten immer die Summe aus den Cachekosten und den Kosten des ARM-Kerns.

Die aufgenommene Energie des SA1100-Prozessorkerns berechnet sich, wie in Kapitel 2.1.2 ausgeführt wurde, nach:

$$E = V_{dd} \cdot I \cdot N \cdot t$$

Der Prozessorkern wird mit einer Spannungen V_{dd} von 1,5 V versorgt, die Taktfrequenz ist auf 162,2 MHz eingestellt. Die Dauer t eines Taktzyklus beträgt:

$$t = \frac{1}{162,2 \cdot 10^6} s$$

Daraus ergeben sich mit der Zahl der Taktzyklen N, die ein Befehl zum Abarbeiten benötigt, die Energiekosten, die eine Instruktion verursacht durch:

$$E_{Instruktion} = \frac{1,5 \cdot I \cdot N}{162,2 \cdot 10^6} \cdot Vs$$

Kapitel 5

Messaufbau

Zur Energiemessung des Caches wurden ein Board mit SA1100-Prozessor, ein Digitalmultimeter und ein PC eingesetzt. Die Abbildung 5.1 zeigt den Messaufbau. Das Board wird vom PC aus gesteuert, somit können Messprogramme auf das Board übertragen und dort ausgeführt werden. Das Digitalmultimeter wird zur Messung der Stromaufnahme des Prozessorkerns verwendet.

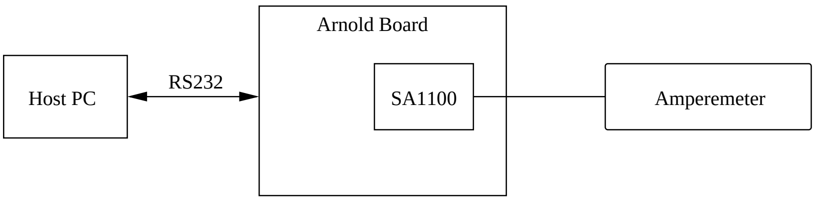

Abbildung 5.1: Versuchsaufbau

5.1 Arnold Board

Als Plattform wurde das „Arnold“ Board von der Keith & Koep GmbH eingesetzt. Das Board hat folgende Eigenschaften:

- SA1100-Microcontroller
- 2 MByte Flash, enthält das Bootimage, das den Angel startet

- 8 MByte Flash, frei verfügbar für Anwendungen
- 32 MByte EDO DRam
- PCMCIA-Slot
- JTAG-Schnittstelle
- RS232-Schnittstelle
- LCD-Controller
- Touch-Schnittstelle
- Audio-Schnittstelle
- 32-Bit-Datenbus und 26-Bit-Adressbus
- Angel-Debug-Monitor

Das Blockschaltbild in Abbildung 5.2 zeigt die Ausstattungsmerkmale des Arnold Boards. Das Board wird über eine 12-Volt Spannungsquelle versorgt. Die für die I/O-Einheit des Prozessors sowie den Speicher und den Flash benötigten 3,3 Volt Versorgungsspannung werden von einem getakteten Schaltregler erzeugt, die vom Prozessorkern benötigten 1,5 Volt durch einen Linearregler. Die interne Taktrate des Prozessors ist auf 162,2 MHz eingestellt. Die externe Ansteuerung des Boards über einen PC kann entweder über die JTAG Schnittstelle erfolgen oder, wie hier, über die RS232-Schnittstelle.

5.1.1 Strommessung am Prozessor

Wie in Abbildung 5.3 zu erkennen ist, sind der Cache des SA1100 und der StrongARM-Kern feste Bestandteile im Prozessorkern und werden daher von der gleichen Spannungsquelle versorgt. Aus diesem Grund lässt sich die Stromaufnahme des Caches nicht isoliert von der des StrongARM-Kerns messen. Die Messung der Stromaufnahme erfolgt, indem in die Ausgangslei-

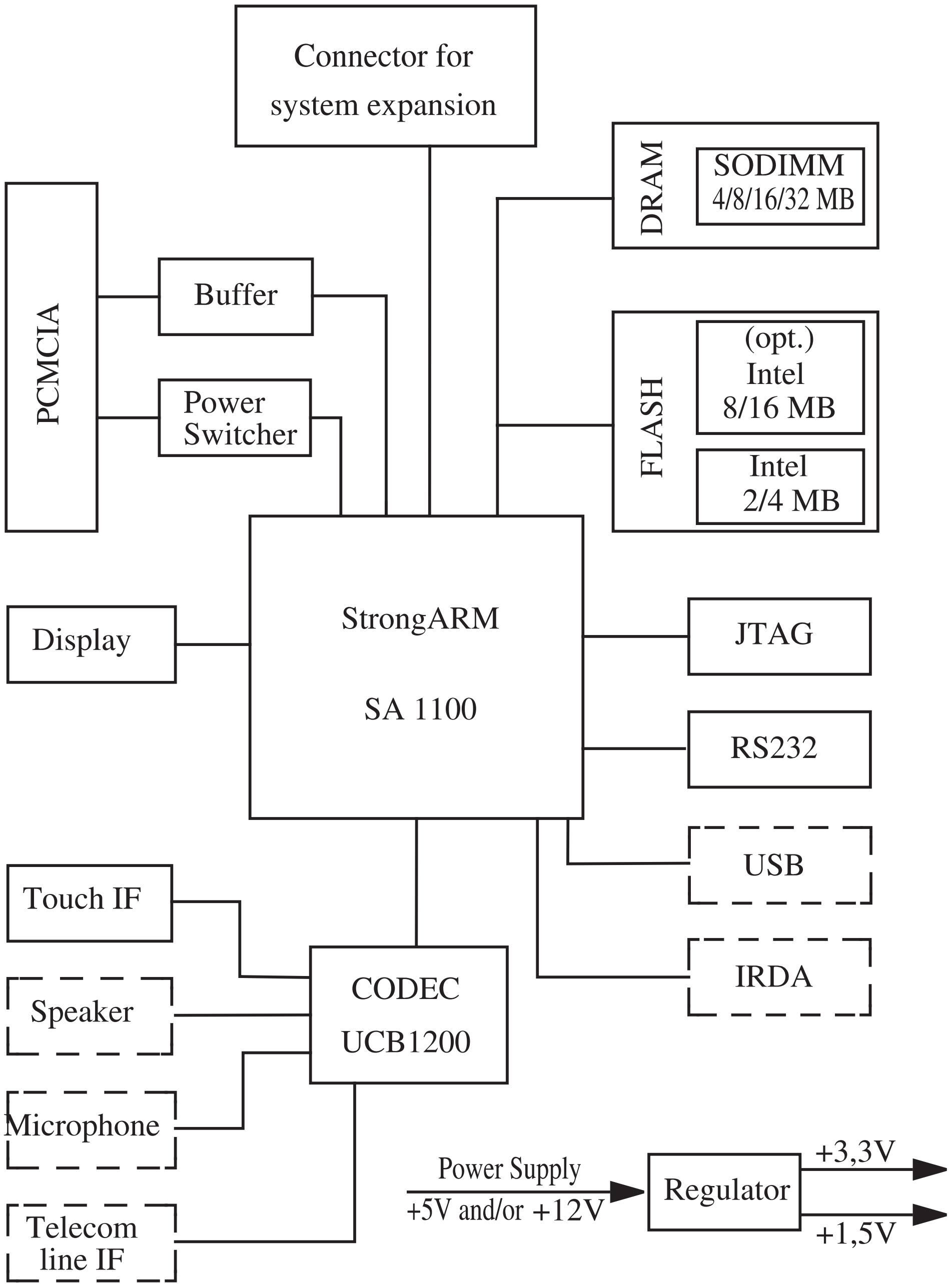

Abbildung 5.2: Blockschaltbild vom Arnold Board [KK00]

tung des 1,5 Volt Linearreglers, ein MIC29204BM, ein Messgerät eingeschleift wird. Dazu wird Pin 1 des Reglers von der Platine abgelötet und jeweils eine Leitung an das Pad auf der Platine und an den Pin angelötet. Diese

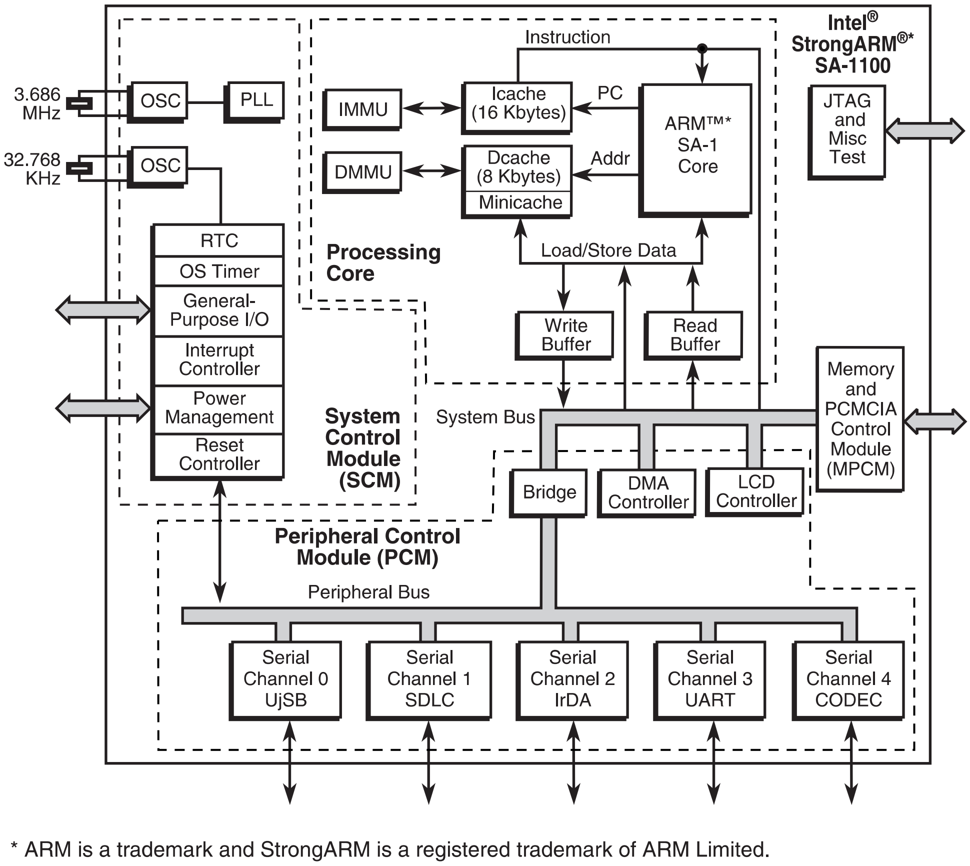

Abbildung 5.3: Blockschaltbild des SA1100 [INT99]

Leitungen werden an das Messgerät angeschlossen.

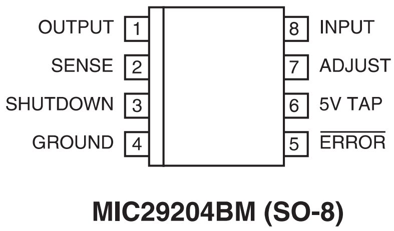

Abbildung 5.4: 1,5 V Spannungsregler [MIC98]

5.1.2 Speicheraufteilung

Die Memory-Map des SA1100 kann Tabelle 3.2 entnommen werden. Der Speicherbereich des Arnold Boardes, der sich innerhalb dieser Memory-Map befindet, gliedert sich in drei Teile:

1. Unterer Flash Speicher
   Dieser Speicherbereich beginnt an der Adresse 0x0000.0000. Er bildet Bank 0 des statischen Speichers und ist 2 MByte groß. Dieser Speicher ist bootfähig, daher befindet sich hier der Primary-Boot-Loader (PBL), der nach dem Anlegen der Betriebsspannung als erstes ausgeführt wird und anschliessend den Angel Remote Debug Agent startet.

2. Oberer Flash Speicher
   Dieser Bereich beginnt bei Adresse 0x0800.0000 und ist 8 MByte groß. Er bildet die Bank 1 des statischen Speichers und steht für Code zur freien Verfügung.

3. EDO RAM
   Dieser Speicher bildet die Bank 0 des dynamischen Speichers. Er beginnt an der Adresse 0xC000.0000 und ist 32 MByte gross.

5.1.3 Angel-Debugger

Angel ist ein System, das die Kommunikation zwischen Host-PC und Arnold-Board ermöglicht. Es besteht aus zwei Teilen, dem Angel-Kernel, der auf dem Board läuft, und dem Debugger auf dem Host-PC. Dieses System ermöglicht die Ausführung und das Debugging auf dem Board sowie die Übertragung von Programmen.

Angel-Kernel

Der Angel-Kernel ist im Flashspeicher des Arnold Boards enthalten. Nach einem Reset wird der Angel in den schnelleren Hauptspeicher kopiert und durch Interrupts angesprochen. Dabei überwacht er die serielle Schnittstelle und führt Anfragen vom Host-PC aus. Zu seinen Aufgaben gehören unter anderem:

- Programme vom Host laden
- Programme starten und stoppen
- Breakpoints setzen
- Tracen von Programmen
- Lesen und Schreiben von Prozessorregistern
- Lesen und Schreiben von Speicheradressen

Ausgaben von Programmen werden an den Host-PC geschickt und an der Konsole ausgegeben, Tastatureingaben in dieser Konsole werden zum Board geschickt.

5.2 Multimeter Escort 95

Die Messungen werden mit dem Digitalmultimeter Escort 95 durchgeführt. Für dieses Gerät gibt der Hersteller einen mittleren Fehler von <1%. Die wichtigsten technischen Daten sind in Tabelle 5.1 aufgelistet.

Grundgenauigkeit	0,06 %
DC V	$10\,\mu V - 1000\,V$
DC A	$100\,nA - 10\,A$
Widerstand	$0,1\,\Omega - 40\,M\Omega$
Messintervall	20 Mess./Sek.

Tabelle 5.1: Technische Daten zum verwendeten Messgerät ESCORT 95 [CCM]

5.3 Messsoftware

Im Rahmen dieser Studienarbeit wurde eine Messsoftware entwickelt, mit deren Hilfe die Stromaufnahme des Prozessorkerns des SA1100 bei unterschiedlichen Betriebszuständen des Caches gemessen werden kann. Die Messsoftware besteht aus zwei Teilen:

Die Initialisierung des Prozessors:

Hier wird die MMU des Prozessors, der Daten und der Instruktionscache je nach Bedarf durch Zugriff auf das Register 1 (Control) des Coprozessor 15 über die Instruktion MCR an- oder ausgeschaltet [INT99]. Der Instruktionscache wird durch Bit 12, der Datencache durch Bit 2 des Registers 1 im Coprozessor ein- und ausgeschaltet. Dieser Teil wurde hauptsächlich in C implementiert. Der Tabelle 5.2 können die Zustände der Caches entnommen werden, bei denen die verschiedenen Messungen durchgeführt wurden.

	Instruktions Cache	Daten Cache
Zustand 1	aus	an
Zustand 2	an	aus
Zustand 3	aus	aus
Zustand 4	an	an

Tabelle 5.2: Zustände des Caches

Die eigentliche Messschleife:

Die Energieaufnahme des Caches wird mit unterschiedlichen Befehlen gemessen, indem mit dem betreffenden Befehl jeweils eine Schleife aufgebaut wird. Load- und Store-Instruktionen werden in Schleifen eingesetzt, in denen die Instruktion 50 mal wiederholt wird, mit anschliessendem Sprung an den Schleifenanfang. Die Schleife mit dem Branchbefehl ist 32 Befehle lang.

5.3.1 Auszüge aus den Messschleifen

Mit Branch-Befehlen können Cache-Misses im Instruktionscache forciert werden, da hier keine Datenspeicherzugriffe stattfinden sondern direkt Befehle angesprungen werden. Der Datencache wird nur bei Load- und Store-Befehlen angesprochen. Aus diesen Gründen wurden drei verschiedene Module mit Messschleifen implementiert.

Modul 1 (Load)

Es erfolgen lesende Zugriffe auf unterschiedliche Speicherbereiche. Dabei werden die Messungen für jeden Cachezustand in Tabelle 5.2 wiederholt. In den unterschiedlichen Durchläufen werden nur die Basisadresse in r0 und der Offset verändert. Eine Messroutine sieht wie folgt aus:

```
        MOV     r2, #0x190000    ; Schleifenzaehler
                                 ; initialisieren
    Loop
        LDR     r1, [r0,#0]
        LDR     r1, [r0,#4]
                .
                .
        LDR     r1, [r0,#196]    ; 50 Instruktionen
        SUBS    r2,#1
        BNE     Loop
```

Der Adressbereich, auf den die Schleife zugreift, wird durch r0 festgelegt. Durch Variation des Offset kann festgelegt werden, ob bei eingeschaltetem Cache alle Daten aus diesem geholt werden können, oder ob nur auf ein Set zugegriffen wird und daher bei jeder Instruktion ein externer Speicherzugriff erfolgt. Bei einem Offset von 256 Bytes wird immer auf dasselbe Set zugegriffen. Die Länge der Schleife wurde auf 50 Befehle festgelegt, weil sich gezeigt hat, dass bei einer größeren Länge keine Energieunterschiede mehr messbar sind, bei kürzeren jedoch der Stromverbrauch durch den Sprungbefehl verändert wird.

Modul 2 (Store)

Hier wird schreibend auf verschiedene Adressen zugegriffen. Auch hier werden die Messungen für jeden Cachezustand in Tabelle 5.2 wiederholt. Die Schleife ist wie Modul 1 aufgebaut, r1 wird vor dem Einsprung in die Schleife mit einem definierten Wert initialisiert.

Modul 3 (Branch)

Im Gegensatz zu den anderen beiden Modulen wird hier mit Branch-Befehlen gearbeitet, da diese nicht den Datencache ansprechen. Durch unterschiedliche Sprungweiten kann hier eingestellt werden, ob bei eingeschaltetem Cache die gesamte Schleife im Cache vorliegt, oder ob die Befehle bei jedem Aufruf erneut aus dem Speicher geladen werden müssen. Aus diesem Grund besteht eine Schleife aus mindestens 33 Branch-Instruktionen, da ein Set im Cache aus 32 Blöcken besteht. Bei einer Sprungweite von 128 Befehlen findet jedes Mal ein Fehlzugriff statt, da dann immer auf dasselbe Set zugegriffen wird.

```
        MOV        r2, #C80000        ; Schleifenzaehler
                                      ; initialisieren
    Loop
        B          l1                 ; 1. Sprungbefehl
        NOP                           ; die Anzahl der NOPs
        .                             ; bestimmt die Sprungweite
        .
        NOP
    l1
        B          l2
        NOP
        .
        .
        NOP
    l2
        B          l3
        NOP
        .
        .
        .
        .
        NOP
    l32
```

```
SUBS        r2, #1
BNE         Loop                ;33. Sprungbefehl
```

5.3.2 Einsatz der Messmodule

Die Messmodule sind kleine Programme, die je nach Bedarf in ein Hauptprogramm eingebunden werden können. Für die unterschiedlichen Messungen werden im Hauptprogramm die Einstellungen für den Cachecontroller vorgenommen und das entsprechende Messmodul eingebunden. Zur Untersuchung verschiedener Basisadressen und Offsets in den Modulen 1 und 2 oder unterschiedlichen Sprungweiten in Modul 3 müssen diese entsprechend angepasst werden. Anschließend wird das Programm kompiliert, an das Board übertragen und dort gestartet.

5.4 Entwicklungsumgebung ARM-SDT

Die Entwicklung der Messsoftware und die anschließenden Messungen wurden mit Hilfe des ARM-SDT 2.50 (ARM Software Development Toolkit) durchgeführt. Das ARM-SDT ist eine Entwicklungsumgebung, die aus zwei getrennten grafischen Benutzeroberflächen, dem APM (ARM Project Manager) und dem ARMSD (ARM Symbolic Debugger), sowie den Entwicklungstools ARMCC, ARMASM und ARMLINK besteht. Die Messmodule und das Hauptprogramm wurden unter der grafischen Benutzeroberfläche APM (ARM Project Manager) erstellt.

ARMCC

Diese beiden C-Compiler übersetzen den ARM-Sourcecode. Der ARMCC erzeugt aus ANSI-C Quellcode 32-Bit-ARM-Maschinencode.

ARMASM

Der ARM-Assembler übersetzt ARM-Code in linkfähigen Objektcode.

ARMLINK

Der Linker erzeugt aus dem generierten Objektcode Programme, die auf dem StrongARM lauffähig sind.

ARMSD

Der ARMSD ist der Simulator und Debugger des ARM-SDT. Er hat unter anderem folgende Funktionen:

-Armulator:

Mit dem Armulator kann der Prozessor und der Arbeitsspeicher des Zielsystems emuliert und so ein Programm getestet werden, bevor es auf ein reales System übertragen und ausgeführt wird.

-Remote Access:

Mit Remote-Access können Programme an das Zielsystem übergeben und dort gestartet werden, um sie dort zu debuggen. Die Kommunikation geschieht mit Hilfe des Angel-Monitorprogramms auf dem Zielsystem.

Kapitel 6

Energiemessung und Auswertung

In diesem Kapitel wird gezeigt wie der Stromverbrauch des Prozessorkerns mit dem Betriebszustand des Caches und der Cachebenutzung zusammenhängt. Die gewonnenen Ergebnisse lassen darauf schliessen, dass sich in den durchgeführten Messreihen der Zustand des Datencaches nicht verändern ließ.

6.1 Berechnete Zyklenzahl

Um die Energiekosten des Caches zu ermitteln, die durch einen Befehl erzeugt werden, muss die Zyklenzahl bekannt sein, die für die Abarbeitung der Instruktion nötig ist. Dies ist insbesondere dann wichtig, wenn ein Cache-Miss vorliegt, da zunächst ein ganzer Block, also 32 Byte, aus dem Speicher übertragen werden und dieser Speicherzugriff zusätzliche Energiekosten verursacht. Gemessen wurden 3 verschiedene Schleifentypen. Die Typen sind Modul 1, Modul 2 und Modul 3 (Abschnitt 5.3.1). Alle Module werden jeweils für jeden Cachezustand in Tabelle 5.2 ausgeführt.

Die errechnete Zyklenzahl, die eine Instruktion benötigt, ist in Tabelle 6.1 aufgeführt. Der Offset und die Sprungweite sind in Bytes angegeben.

Bei der Messung mit den Modulen 1 und 2 wird deutlich, dass der Zustand des Datencaches nicht verändert wird, da jeweils in Zustand 1 und Zustand 3 sowie in Zustand 2 und Zustand 4 die gleiche Zyklenzahl benötigt wurde,

Modul	Offset/ Sprungweite	Zyklen Zustand 1	Zyklen Zustand 2	Zyklen Zustand 3	Zyklen Zustand 4
1	4	55	26	55	26
2	4	51	23	51	23
3	4	56	2	56	2
3	128	58	163	58	163

Tabelle 6.1: Taktzyklen der Befehle in jedem Cachezustand

obwohl das Kontrollbit des Datencaches in Zustand 2 und 3 aus, und in Zustand 1 und 4 eingeschaltet ist.

6.2 Gemessene Ströme

Der Strom des Prozessorkerns wird, wie in Kapitel 4.4 beschrieben, gemessen. Bei den Messwerten ist die Messgenauigkeit des Messgerätes zu berücksichtigen.

In Tabelle 6.2 sind die Ergebnisse der Strommessungen von Modul 1 (Kapitel 5.3.1) für verschiedene Offsets eingetragen. In Tabelle 6.3 sind die Messergebnisse von Modul 2 eingetragen.

Offset in Byte	Zustand 1 (mA)	Zustand 2 (mA)	Zustand 3 (mA)	Zustand 4 (mA)
4	66,3	89,0	66,5	88,8
32	66,0	88,2	66,3	88,0

Tabelle 6.2: Stromaufnahme bei Load-Befehlen

Offset in Byte	Zustand 1 (mA)	Zustand 2 (mA)	Zustand 3 (mA)	Zustand 4 (mA)
4	67,6	96,9	67,6	97,1
32	67,4	97,1	67,5	96,8

Tabelle 6.3: Stromtabelle bei Store-Befehlen

Die gemessenen Ströme in den Tabellen 6.2 und 6.3 zeigen, ebenso wie die Zyklenzahlen in Tabelle 6.1, durch die jeweils annähernd identischen Werte in

Zustand 1 und 3 sowie in Zustand 2 und 4, dass der Zustand des Datencaches nicht verändert wird.

In Tabelle 6.4 sind die Ergebnisse der Strommessungen von Modul 3 (Kapitel 5.3.1) eingetragen. Die große Differenz in den gemessenen Strömen zwischen eingeschaltetem und abgeschaltetem Instruktionscache liegen darin bergündet, dass in den ersten beiden Fällen die Messschleife komplett im Cache abgearbeitet wurde. Dies stimmt auch mit den Messungen in Tabelle 6.1 überein.

Sprungweite in Byte	Zustand 1 (mA)	Zustand 2 (mA)	Zustand 3 (mA)	Zustand 4 (mA)
4	62,5	224,4	62,3	224,5
8	63,0	236,3	63,2	236,2
128	63,6	63,3	63,6	63,3

Tabelle 6.4: Stromaufnahme bei Branch-Befehlen

6.3 Errechnete Energie pro Befehl

Die Energiekosten, die ein Befehl verursacht, werden aus dem aufgenommenen Strom und der Zeit berechnet, die der Befehl zum Abarbeiten benötigt. Hier fließen auch die Kosten für den Speicherzugriff ein.

Energie der Load- und Store-Befehle

Die von den Load- und Store-Befehlen verursachten Kosten hängen vom Zustand des Instruktionscaches ab, nicht jedoch vom Zustand des Datencaches. Die Messwerte liegen im Rahmen der Messgenauigkeit. Die Energiewerte in Tabelle 6.5 wurden mit den Formeln in Kapitel 4.4.2 aus den Messwerten errechnet. Da die Werte nicht vom Zustand des Datencaches abhängen, wird nur der Zustand des Instruktionscaches berücksichtigt.

Instruktion	Instruktionscache	Energie ($nW \cdot s$)
Load	an	21,2
Load	aus	33,6
Store	an	20,6
Store	aus	31,8

Tabelle 6.5: Energie von Load- und Store-Befehlen

Energie des Branch-Befehls

Die Energiekosten des Branch-Befehls in Tabelle 6.6, die aus den Messergebnissen errechnet wurden, berücksichtigen auch die jeweilige Sprungweite.

Sprungweite	Instruktionscache	Energie ($nW \cdot s$)
4	on	4,15
4	off	32,4
128	on	34,1
128	off	95,4

Tabelle 6.6:

Kapitel 7

Zusammenfassung der Ergebnisse

Aus den gemessenen Werten und den daraus errechneten Zykluszeiten und Energiekosten der Befehle ist erkennbar, dass durch die Verwendung des Instruktionscaches eine wesentliche Energieersparnis erreicht werden kann. Dies wird umso deutlicher, wenn die Energiekosten eines Befehles zusammen mit seiner Durchlaufzeit betrachtet werden. Dabei ist jedoch zu Berücksichtigen, dass ein Cache zusätzliche Energie und Chipfläche benötigt. Bei Verzweigungsbefehlen ist jedoch auf die Sprungweite zu achten, da sich bei ungünstigen Werten der Energieverbrauch vervielfachen kann.

Ausblick

Diese Studienarbeit Beschäftigte sich mit der Untersuchung des Energieverbrauchs des Caches. Dabei wurde immer der Angel als Hilfsmittel verwendet. Weitere Untersuchungen könnten dahingehend stattfinden, dass der Angel abgeschaltet oder Ersetzt wird, um mögliche Einflüsse durch seinen Einsatz zu eliminieren. Die Methode und Ergebnisse dieser Arbeit können dabei als Grundlage dienen.

Anhang A

Dieser Anhang enthaelt den Sourcecode des Hauptprogramms.

```
#include <stdio.h>
#include ''encache.h''
#include ''data_cachebench.h''
// Definition der Bits des MMU-Kontrollregisters
#define MMU_I 0x1000     //Instruktionscache
#define MMU_D 0x0004     //Datencache
////////////////////////////////
//            MAIN            //
////////////////////////////////
int main()
{
        int i,id, testdata, startbase;
//      Basisadresse fuer Speicherzugriffe
        startbase=0x0C0000000;
//      Datenwort fuer Schreibzugriffe
        testdata=0xFFFFFFFF;
//      auslesen des Kontrollregisters
        __asm { MRC P15, 0, id, c1, c0; }
//      die Bits 4 und 12 werden geloescht
        id=(id & (0xFFFFEFFB));
//      und durch MMU_D und MMU_I bei Bedarf gesetzt.
        id=(id | (MMU_D + MMU_I));
//      Kontrollregister neu setzen
        __asm { MCR P15, 0, id, c1, c0; }
//      das Messmodul aufrufen
        i=call_data_cache(startbase,testdata);
        return 1;
}
```

Literaturverzeichnis

[CCM] Cosinus Computermesstechnik GmbH: *Digitale Handmultimeter Escort-95/97 Bedienungsanleitung*

[FDL89] Möschwitzer A.: *Formeln der Elektrotechnik und Elektronik*, 2. Auflage, VEP Verlag Technik, 1989

[HMS92] Hering E.; Martin R.; Stohrer M.: *Physik für Ingenieure,* 4. Auflage, VDI-Verlag GmbH, 1992

[HP94] Hennessy, John L.; Patterson, David A.: *Rechnerarchitektur* ,Vieweg Verlag, 1994

[INT99] Intel Corporation: *Intel StrongARM SA-1100 Microprocessor Developer's Manual*, Dokument-Nr:278088-004, August 1999

[KK00] Keith & Koep GmbH: *Arnold Board Dokumentation 3.0,* Januar 2000

[MAR00] Marwedel, P: *Skript zur vorlesung Rechnerarchitektur,* Universität Dortmund, Fakultät Informatik, Lehrstuhl 12, Technische Informatik, April 2000

[MIC98] Micrel: Datenblatt zum *MIC2920A/29201/29202/29204,* Januar 1998

[SYN96] Synopsys, Inc.: *Power Products Reference Manual*, Version 3.5, September 1996

[THE00] Theokharidis, Michael: *Diplomarbeit*, *Energiemessung von ARM7TDMI Prozessor-Instruktionen*, Universität Dortmund, Fakultät Informatik, Lehrstuhl 12, Technische Informatik, November 2000

Erklärung

Ich versichere, dass ich diese wissenschaftliche Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen sind, wurden in jedem einzelnen Fall durch Angabe der Quelle als Entlehnung kenntlich gemacht. Das gleiche gilt auch für beigegebene Skizzen und Darstellungen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, den 30. März 2001

Einwilligung

Hiermit erkläre ich mich damit einverstanden, dass diese wissenschaftliche Arbeit nach den Bestimmungen des §6 Absatz 1 des Gesetzes über Urheberrecht vom 9.9.1965 in die Bereichsbibliothek aufgenommen und damit für Leser der Bibliothek öffentlich zugänglich gemacht wird.

Ferner bin ich damit einverstanden, dass gemäß §54 Absatz 1 Satz 1 dieses Gesetzes Leser zu persönlichen wissenschaftlichen Zwecken Kopien aus der Arbeit anfertigen dürfen.

Dortmund, den 30. März 2001