Codierungsverfahren zur Reduktion des Energiebedarfs von Programmen

Markus Knauer

25. Juli 2001

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Seit den letzten Jahren werden vermehrt eingebettete Systeme mit Prozessoren eingesetzt. Die Einsatzgebiete solcher Systeme sind vielfältig. Sie werden beispielsweise als mobile Geräte in der Telekommunikation und im Multimediabereich eingesetzt. Ein großes Anwendungsfeld, in dem eingebettete Systeme für den Benutzer meist völlig unbemerkt bleiben, ist die Automobilelektronik. Betrachtet werden zwei Hauptprobleme solcher Systeme: Die begrenzte Energieversorgungsmöglichkeit und die Wärmeentwicklung.

Beim Einsatz von beispielsweise tragbaren MP3-Playern oder Handys gibt es das Problem, dass die Standby- und Nutzzeiten oft zu wünschen lassen. Mit dem Trend der Verkleinerung der Geräte steht auf der einen Seite immer weniger Platz für Akkukapazitäten zur Verfügung, auf der anderen Seite steigt mit größer werdendem Funktionsumfang der Geräte auch der Energiebedarf der Geräte an. Akkukapazitäten konnten in der Vergangenheit nicht dermaßen verbessert werden, wie der Energieverbrauch zugenommen hat.

Die bei steigendem Energieverbrauch zunehmende Wärmeentwicklung kann zu vorzeitigem Materialverschleiß führen. Außerdem ist es für den Anwender nicht angenehm, wenn er z.B. an einem 50 Grad heißen Notebook arbeitet. Kühltechniken haben in der Vergangenheit kaum nennenswerte Fortschritte gebracht.

Wegen dieser aufgezeigten Probleme möchte man erreichen, den Energieverbrauch von eingebetteten Systeme zu verringern. Möglich ist dies sowohl durch Hardware- als auch durch Software-Optimierungen.

Auf der Hardware-Seite könnte eine Energieverringerung beispielsweise durch Herabsetzen der Versorgungsspannung, die einen geringeren Stromverbrauch nach sich ziehen würde, erreicht werden.

Auf der Software-Seite kann zur Energieeinsparung an verschiedenen Stellen angesetzt werden. Das Betriebssystem kann durch Powermanagement-

Funktionen den Energieverbrauch senken, indem nach längerer Nichtbenutzung des Systems Peripheriegeräte wie z.B. Monitor und Festplatte abgeschaltet werden [BM99]. Ein Nachteil kann hierbei sein, dass das Wiederhochfahren der Geräte eine gewisse Zeit in Anspruch nimmt. Eine andere Möglichkeit der Energieeinsparung liegt in der Erstellung von Software. Die Auswahl geeigneter und effizienter Algorithmen trägt wesentlich zur Einsparung von Energie bei. Zum Anderen ist es möglich, durch bestimmte Compiler-Techniken und Optimierungen eine Energieeinsparung zu erreichen. Herkömliche Compiler optimieren nur bezüglich Programmlaufzeit und Codegröße. Allgemein lässt sich sagen, dass eine Optimierung der Laufzeit meist auch zur Verringerung des Energieverbrauchs führt. Darüber hinaus lassen sich aber auch spezielle Optimierungen durchführen, die den Programmcode gezielt in Richtung Energiereduktion verbessern. Als Beispiel sei ein Prozessor-System betrachtet, welches neben externem Arbeitsspeicher einen Prozessor mit einem kleinen energiesparenden On-Chip-Speicher besitzt. Durch Plazierung von häufig benutzten Daten im On-Chip-Speicher lassen sich Energieeinsparungen erreichen. Eine weitere Möglichkeit der Energiereduktion zeigt sich bei Schaltaktiväten im Prozessor und im Speicher. Schaltaktiväten in Funktionseinheiten und auf Bussen haben Einfluss auf den Energieverbrauch von Prozessor und Speicher. Durch Verringerung dieser Aktivitäten durch den Compiler lassen sich Programme bezüglich Energie optimieren [STD94a]. Hiermit beschäftigt sich diese Diplomarbeit.

1.2 Ziele dieser Arbeit

Zuerst ist ein Energiemodell zu entwickeln, welches insbesondere die Kosten beschreibt, die zur Reduktion des Energiebedarfs von Programmen genutzt werden sollen. Zu berücksichtigen sind Kosten, die sowohl im Prozessor als auch im Speicher anfallen. Als nächstes Ziel ist das Energiemodell auf den ARM7TDMI anzuwenden. Es ist aufzuzeigen, welche Energiekosten durch Softwareoptimierungen für Energieeinsparungen genutzt werden können. Darauf aufbauend sollen Verfahren vorgestellt werden, welche die Optimierungen durchführen.

1.3 Kapitelübersicht

Kapitel 2 beschreibt elektrotechnische Hintergründe und Effekte, auf denen diese Diplomarbeit aufbaut. Im wesentlichen sind dies Energieverluste die bei Schaltvorgängen in Halbleiterbausteinen und beim Umladen von Leitungskapazitäten anfallen. Es folgt eine Beschreibung der Untersuchungsumgebung, die für diese Arbeit verwendet wird.

In Kapitel 3 werden bisher entwickelte Energiemodelle vorgestellt und ein im Rahmen dieser Arbeit neu entwickeltes Modell beschrieben.

Kapitel 4 beschreibt die Anwendung des Energiemodells auf den ARM7TDMI-Prozessor. Es wird aufgezeigt, mit welchen Testmustern Messungen durchgeführt werden und wie die Daten anschließend für das Energiemodell aufbereitet werden. Zur Durchführung von Optimierungen im nächsten Kapitel findet eine Auswertung der gewonnenen Daten statt. Im letzten Teil von Kapitel 4 erfolgt die Beschreibung, welche Software im Rahmen dieser Diplomarbeit implementiert wurde.

Aufbauend auf den bisherigen Kapiteln werden in Kapitel 5 Optimierungen zur Energieeinsparung vorgestellt. Zur Anwendung der einzelnen Optimierverfahren wird eine Strategie entwickelt, in welcher Reihenfolge die Verfahren angewendet werden.

Kapitel 6 bildet in Form einer Zusammenfassung und eines Ausblicks auf zukünftige Forschungsarbeiten den Abschluss dieser Arbeit.

Kapitel 2

Grundlagen

2.1 Elektrotechnische Hintergründe

Es folgt eine Übersicht der wichtigsten grundlegenden Begriffe und Formeln, die im Zusammenhang mit Leistung und Energie auftreten [KSW95].

- Leistung

  Die Leistung P ist definiert als Produkt von Spannung und Strom. Die beiden Größen werden als konstante oder mittlere Werte angenommen.

  $P = U \cdot I$
  Einheit: 1 W (Watt) = 1 V (Volt) $\cdot$ 1 A (Ampere)

  Der Begriff der Leistung allein gibt noch keine Auskunft darüber aus, ob beispielsweise Software während der Ausführung „viel oder wenig verbraucht". Es bleibt unberücksichtigt, wie lange die Software läuft. Bei Hinzunahme der Zeit gelangt man zum Begriff der Energie.

- Energie

  Wenn Spannung und Strom bzw. Leistung als Mittelwerte über die Zeit gegeben sind, berechnet sich die Energie als

  $E = U \cdot I \cdot t = P \cdot t$
  Einheit: 1 J (Joule) = 1 VAs = 1 Ws.

  Sollten keine mittleren Werte bekannt sein, lässt sich die Energie als Integral über die Zeit von t_1 bis t_2 wie folgt berechnen:

  $$E = \int_{t_1}^{t_2} p(t)\mathrm{d}t = \int_{t_1}^{t_2} i(t) \cdot u(t)\mathrm{d}t$$

Für die in einer Kapazität C speicherbare Energie gilt:

$$E_{cap} = \frac{1}{2} C U^2 \tag{2.1}$$

2.2 Strom als Einflussgröße des Energieverbrauchs

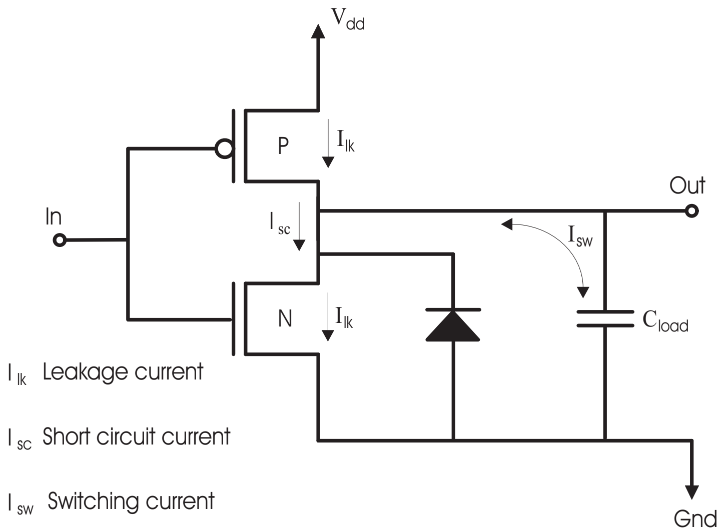


Abbildung 2.1: Inverter als Leitungstreiber in CMOS-Technik

Abbildung 2.1 stellt den Aufbau eines CMOS-Gatters nach [SYN96] dar. Das am Eingang angelegte Signal steht am Ausgang invertiert zur Verfügung. Am Ausgang kann beispielsweise die Leitung eines Busses angeschlossen sein. Der im Gatter fließende Strom lässt sich in drei Ströme aufteilen: Leckstrom I_{lk}, Kurzschlussstrom I_{sc} und Schaltstrom I_{sw}.

- Leckstrom (Leakage current)

  Der Leckstrom fließt unabhängig davon, ob das Gatter gerade schaltet. Während eines Schaltvorgangs macht der Leckstrom nur knapp 1 % des gesamten Entergieverbrauchs aus. Im inaktiven Zustand der Schaltung, d.h. am Eingang findet keine Signaländerung statt, ist dieser Strom der einzig fließende.

- Kurzschlussstrom (Short circuit current)

  Dieser Strom fließt während eines Schaltvorgangs, d.h. während der ansteigenden oder abfallenden Flanke des Eingangssignals. Die Transistoren P und N sind für kurze Zeit gleichzeitig leitend. Es fließt ein Strom von der Versorgungsspannung nach Masse. Die Verlustenergie ist abhängig von der

Schaltgeschwindigkeit des Gatters. Diese wiederum wird bestimmt durch die Technologie, in der das Gatter gefertigt ist. Der Energieverbrauch hat an der gesamten Schaltung einen Anteil von bis zu 30 %.

- Schaltstrom (Switching current)

  Der Schaltstrom fließt während des Ladens und Entladens von Kapazitäten am Ausgang des Gatters. Er führt zu 70 % bis 90 % Verlustenergie.

2.2.1 Schalthäufigkeiten

In aktiven CMOS-Schaltungen fallen durch Kurzschlussströme und insbesondere durch Schaltströme in nicht unerheblichem Maße Energieverluste an. Beiden Energien gemeinsam ist es, dass die Verluste linear mit der Anzahl der Schalthäufigkeiten zusammenhängen. Durch Schaltströme bedingte Energieverluste verhalten sich nach Formel 2.1 aus Abschnitt 2.1 proportional zu an den Gatter-Ausgängen angeschlossenen Kapazitäten. Große Kapazitäten treten vor allem bei externen Bussen auf. Um so länger ein Bus ist, um so höher ist auch seine Kapazität, die bei Schaltvorgängen mit entsprechend viel Energie umgeladen werden muß. Busse können in der Praxis aus vielen Leitungen bestehen. Dementsprechend ist es erfolgversprechend, durch Verringerung der Zustandswechsel auf Busleitungen (engl.: *switching activity*) Energieeinsparungen zu erreichen.

2.2.2 Anzahl von Einsen

Leckströme sind dafür verantwortlich, dass selbst bei keinem Wechsel des Eingangssignals Verlustströme auftreten. Je nach verwendeter Fertigungstechnologie der integrierten Schaltungen kann die Verlustenergie für eine am Eingang anliegende Eins oder Null (technisch entspricht dies einem High- oder Low-Pegel) unterschiedlich sein. Durch Ausnutzung dieses Effekts sind ebenfalls Energieeinsparungen möglich.

2.2.3 Funktionseinheiten

Funktionseinheiten wie z.B. ALU und Multiplizierer sind aus vielen Gattern aufgebaut. Bei Benutzung dieser Einheiten zwecks Ausführung einer Prozessorinstruktion finden erhöhte Schaltaktivitäten statt. Durch Messungen hat sich gezeigt [DAMT00], dass die Verwendung gleicher Funktionseinheiten hintereinander energetisch günstiger ist als die abwechselnde Nutzung. Zur erstmaligen Nutzung einer Funktionseinheit, die von der vorherigen Instruktion nicht benötigt wurde, fallen zusätzliche Aktivierungskosten an. Diese Eigenschaft kann zu Energieeinsparungen genutzt werden, indem Instruktionen so angeordnet werden, dass sie hintereinander gleiche Einheiten ansprechen.

2.3 Untersuchungsumgebung

2.3.1 ARM7TDMI-Prozessor

Als Arbeitsmittel für diese Diplomarbeit wurde der ARM7TDMI-Prozessor [ARM95b] des Herstellers *Advanced RISC Machines Ltd. (ARM)* eingesetzt. Es handelt sich hierbei um einen 32 Bit RISC (Reduced Instruction Set Computer) Prozessor. Er besitzt die 3 Pipelinestufen *Fetch*, *Decode* und *Execute*. Das Blockschaltbild ist in Abbildung 2.2 abgebildet.

Als Besonderheit verfügt der Prozessor über die beiden Instruktionssätze *ARM* und *THUMB*. Der wesentliche Unterschied besteht darin, dass Instruktionen im ARM-Modus 32 Bit und im THUMB-Modus 16 Bit breit sind. Zur Ausführung werden im *Thumb Instruction Decompressor* 16 Bit THUMB-Instruktionen ohne zusätzliche Prozessorzyklen in 32 Bit ARM-Instruktionen umgesetzt. Bei der Programmerstellung kann an beliegen Stellen zwischen beiden Befehlssätzen gewechselt werden. Der Vorteil des THUMB-Modus ist, dass im Durchschnitt der Code um 33 % kleiner ist als im ARM-Modus [ARM95a]. Dieser Vorteil wird mit dem Nachteil erkauft, dass der THUMB-Befehlssatz kleiner ist als der ARM-Befehlssatz. Beispielsweise gibt es weniger Adressierungsarten und begrenzte Sprungweiten. Wegen dieser Einschränkungen besteht THUMB-Code im Allgemeinen aus mehr Instruktionen.

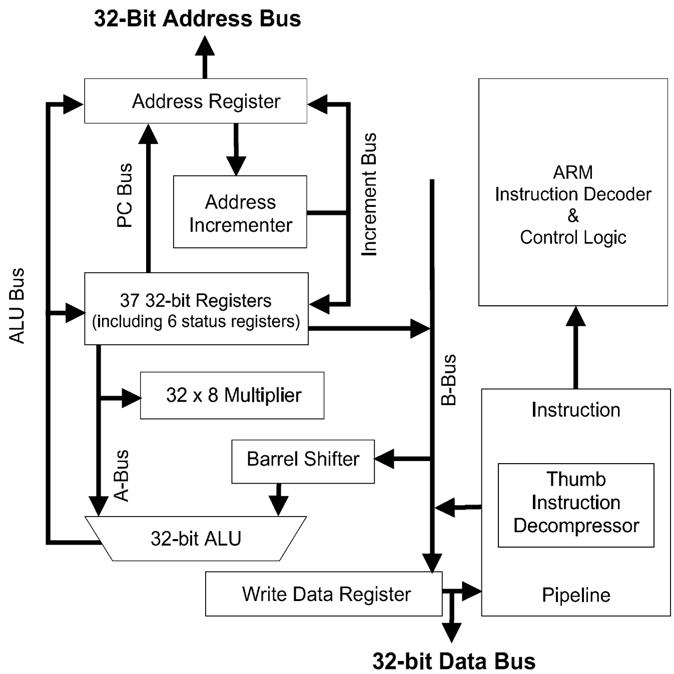


Abbildung 2.2: Aufbau des ARM7TDMI-Prozessors [ATM99a]

Der Prozessor besitzt insgesamt 37 32 Bit Register, wovon 6 als Statusregister genutzt werden. Es sind nicht alle Register gleichzeitig sichtbar, im *Usermode* sind für den ARM-Instruktionssatz nur die 16 Register R0 bis R15 und 2 Statusregister verfügbar (Tabelle 2.3). Wegen der begrenzten Instruktionsbreite im THUMB-Modus stehen dort nur die 8 Low Register R0 bis R7 zur Verfügung, auf die High Registers R8 bis R12 kann nur mittels spezieller Befehle zugegriffen werden. Register R13 bis R15 enthalten Stack Pointer, Link Register und Program Counter.

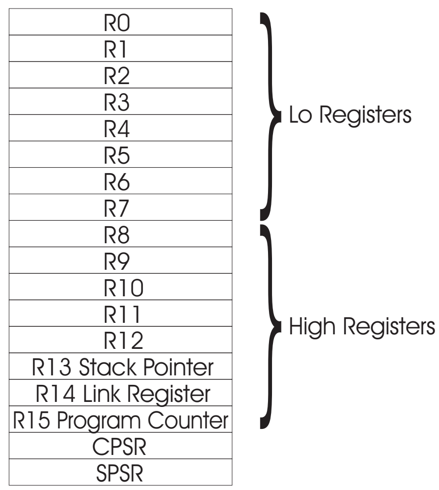


Abbildung 2.3: Registeraufbau des ARM7TDMI

Der Prozessor ist von ARM nicht als eigenständiger Chip erhältlich, sondern kann zur eigenen Verwendung als Core lizenziert werden.

Nachfolgend die wichtigsten Eigenschaften des ARM7TDMI Prozessors in Kurzform:

- 32 Bit RISC Architektur
- Von Neumann Load/Store Architektur
- 3-stufige Pipeline: Fetch, Decode und Execute
- 2 Instruktionssätze: ARM- und THUMB-Instruktionssatz
- 37 32 Bit Register
- 4 GB Adressraum
- 8 (byte), 16 (halfword) und 32 Bit (word) Datentypen
- Nur als Core erhältlich
- Niedriger Energieverbrauch
- 3,3 Volt Versorgungsspannung

2.3.2 Atmel Evaluationboard AT91EB01 und Software

Wie bereits erwähnt, ist der ARM7TDMI von ARM nicht als einzelner Chip erhältlich. Eingesetzt wird ein Evaluationboard *AT91EB01* der Firma *Atmel Corporation*. Der Core des ARM-Prozessors ist in Form des Microcontrollers *AT91M40400* zusammen mit 4 KB RAM und zusätzlicher Peripherie auf einem Chip vereint. Abbildung 2.4 zeigt den Aufbau. Strommessungen, die im Rahmen dieser Diplomarbeit durchgeführt wurden, konnten nicht allein am ARM-Prozessor durchgeführt werden, sondern beziehen sich immer auf den gesamten Microcontroller.

Das Evaluationboard enthält u.a. noch 512 KB RAM, 128 KB Flash-Speicher und Debugger-Hardware. Der Off-Chip-Speicher besitzt eine Datenbreite von 16 Bit, der On-Chip-Speicher von 32 Bit. Adress- und Datenbusse des 4 KB On-Chip- und des Off-Chip-Speichers sind getrennt.

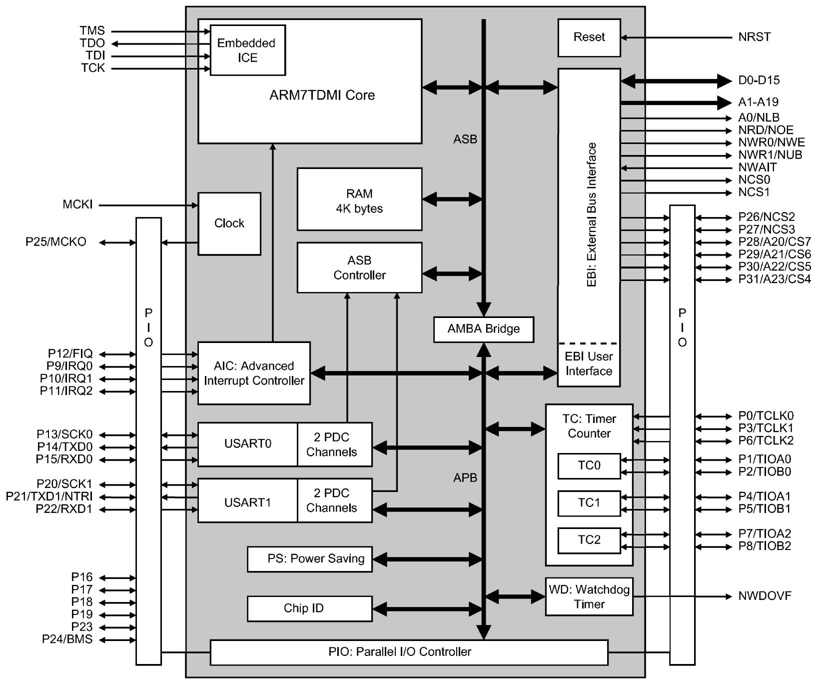


Abbildung 2.4: Microcontroller AT91M40400: ARM7TDMI-Core, 4 KB RAM und zusätzliche Peripherie [ATM99b]

Angeschlossen wird das Board über eine serielle Schnittstelle an einen Host-PC. Mit dem Programmsystem *Angel* auf dem Host-PC wird das Evaluationboard angesteuert, Programme werden auf das Board übertragen und dort ausgeführt. Bei Bedarf können die Programme im Zusammenspiel mit Angel und der Debugger-Hardware auf dem Board debugged werden.

Kapitel 3

Energiemodell

Zum Zweck von Energieoptimierungen und zur Energieabschätzung von Programmen wurden in der Forschung zahlreiche Energiemodelle für Prozessor-Systeme entwickelt, die jeweils auf unterschiedlichen Detail-Ebenen ansetzen.

Tiwari [TMW94],[TIW96] hat in seinen Arbeiten ein Modell vorgestellt, welches auf Instruktionsebene arbeitet. Er unterscheidet zwischen Base-Kosten und Inter-Instruction-Kosten. Die Base-Kosten werden erfasst, indem alle Instruktionen des Prozessors einzeln in einer Schleife gemessen werden. Inter-Instruction-Kosten sind zusätzliche Kosten, die zwischen der Ausführung zweier unterschiedlicher Instruktionen anfallen. Tiwaris Modell berücksichtigt keine Speicherkosten.

Sinevriotis et al. [SS99] haben Tiwaris Energiemodel auf den ARM7 Prozessor angewendet und Messungen durchgeführt. Basierend auf den Messungen werden Vorschläge für die Durchführung von Compiler-Optimierungen gegeben.

Benini [BHS98] hat ein Modell aufgestellt, welches auf Systemebene ansetzt. Betrachtet werden Komponenten wie Prozessor, Speicher, Display und Hintergrundbeleuchtung, z.B. eines Hand-Held-Systems. Die Komponenten werden durch Zustände mit unterschiedlichen Energieverbräuchen beschrieben.

Im folgenden wird ein Energiemodell vorgestellt, welches im Rahmen dieser Diplomarbeit entwickelt wurde. Es arbeitet auf Instruktions-Ebene und dient als Grundlage für Entscheidungen bei Compiler-Optimierungen in späteren Kapiteln. Ferner wird das Modell zur Berechnung des Energieverbrauchs von simulierten Programmen benutzt. Das neue Energiemodell basiert auf Tiwaris Modell. Erweitert wurde es um die Betrachtung von Speicherkosten. Bei der Anwendung des Modells auf den ARM7TDMI-Prozessor zeigt sich, dass die fehlende Berücksichtigung des Speichers zu ungenauen Ergebnissen führen kann. Zusätzlich werden Kosten von Daten-Codierungen auf Bussen berücksichtigt. Inspiriert wurde die Modellierung dieser Buskosten durch eine Arbeit von Chang, Kim und Lee [CKL00].

Weitere Ausführungen zu Energiemodellen finden sich bei Steinke et al. [SKWM01].

Anwendungsziele des Modells

1. Energieabschätzung von Programmen

   Die meisten Hardwarehersteller veröffentlichen nur sehr grobe Werte für den Energieverbrauch ihrer Prozessoren und Speicher. Es handelt es sich meist um Durchschnittswerte. Je nach Software verbrauchen Prozessoren im allgemeinen bei rechenintensiven Programmen, die im großen Maßstab die FPU benutzen (z.B. Videobearbeitungssoftware), mehr Energie als beispielsweise Textverarbeitungssoftware, die die meiste Zeit auf Benutzereingaben wartet.

   Das Modell kann genutzt werden, um von simulierten Programmen den Energieverbrauch zu berechnen. Die Energiewerte wiederum könnten zur Hilfe genommen werden, um mittels weiterer Berechnungen Rückschlüsse auf die mögliche Wärmeentwicklung zu ziehen und Maßnahmen zur Kühlung durchzuführen. Ebenso können Energieversorgungen entsprechend dimensioniert werden oder die Software auf Hochsprachenebene optimiert werden.

2. Entscheidungshilfen im Compiler für Energieoptimierungen

   Optimierungstechniken herkömmlicher Compiler optimieren nur bezüglich Codegröße oder Geschwindigkeit. Generell lässt sich sagen, dass Optimierungen der Geschwindigkeit auch meist zu einem geringeren Energieverbrauch führen. Trotzdem ist es wünschenswert, dass für eine weitergehende Energieoptimierung spezielle Compiler-Techniken entwickelt werden. Als Konsequenz ist das Energiemodell so zu erstellen, dass es Effekte beschreibt, die für Optimierungen genutzt werden können. Im Compiler wird dementsprechend eine Kostenfunktion für die Energie integriert.

3.1 Anforderungen

Das Energiemodell hat hinsichtlich der Anwendungsziele folgende Anforderungen [SKWM01]:

- Energiewerte für Instruktionen

  Je nach Mächtigkeit des Maschinen-Instruktionssatzes kann es für eine Anweisung unterschiedliche Instruktionen geben. Z.B. lässt sich die Multiplikation einer Zahl mit 2 auf verschiedene Arten realisieren. Entweder als normale Multiplikation im Multiplizierer oder als Addition mit sich selbst. Je nach Auswahlmöglichkeiten hat der Compiler die günstigste Instruktion zu wählen. Dazu müssen die Energiekosten für jede Instruktion bekannt sein.

- Reihenfolge von Instruktionen

  Bei der Ausführung von Instruktionen fallen für das Aktivieren oder Deaktivieren der Funktionseinheiten in Abhängigkeit von der Reihenfolge der Instruktionen unterschiedliche Energiekosten an. Mit einem Instruction-Scheduling kann der Compiler diese Kosten minimieren.

- Speicherhierarchie

  Wenn Prozessorsysteme verschiedene Speicher besitzen, die sich in Geschwindigkeit und Energie unterscheiden, ist vom Compiler eine entsprechende Platzierung von Instruktionen und Variablen vorzunehmen.

- Codierungen auf Bussen

  In Abhängigkeit von der Anzahl der Bit-Wechsel auf Busleitungen fallen unterschiedliche Energiekosten an. Diese Kosten für Bit-Wechsel müssen im Modell mit enthalten sein. Im Zusammenhang mit der Speicherhierarchie und den unterschiedlichen Bit-Wechseln auf Bussen ist vom Compiler/Linker ein energiegünstiges Speicherlayout für Instruktionen und Daten zu finden.

- Datenerhebung

  Zwecks Bestimmung der Modellparameter für einen bestimmten Prozessor ist es wünschenswert, dass Messungen für Datenerhebungen einfach durchzuführen sind, ohne dass über die internen Details des Prozessordesign Informationen vorhanden sein müssen.

- Übertragbarkeit

  Das Modell sollte für eine möglichst große Klasse von Prozessoren anwendbar sein.

3.2 Grundlagen

In den beiden nächsten Abschnitten werden *Einzelkosten* definiert und beschrieben. Einzelkosten erfassen einzelne Energieanteile auf Bussen und innerhalb von Komponenten wie Speicher und Funktionseinheiten. Das im Anschluss an die Grundlagen vorgestellte Modell setzt sich, strukturiert nach physikalischem Entstehungsort und logischer Gliederung, aus den nachfolgenden Einzelkosten zusammen.

3.2.1 Einzelkosten auf Bussen

In Abschnitt 2.2 wurde gezeigt, dass Schaltvorgänge auf Bussen und die Zahl der anliegenden Einsen den Energieverbrauch einer Schaltung beeinflussen. Zur

Beschreibung dieser Effekte werden nachfolgend die beiden Begriffe Hamming-Distanz und Ones eingeführt.

Definition: Hamming-Distanz

Die Hamming-Distanz gibt die Anzahl der unterschiedlichen Bits auf Bussen zu zwei aufeinander folgenden Zeitpunkten an. Die Hamming-Distanz zwischen zwei binären Codeworten a und b, bestehend aus n Bits, wird ausgedrückt durch die Formel $h(a, b)$:

$$h(a,b) = \sum_{i=0}^{n-1} a_i \text{ xor } b_i \quad , \text{mit } a = \sum_{i=0}^{n-1} 2^i \cdot a_i \text{ und } b = \sum_{i=0}^{n-1} 2^i \cdot b_i$$

Definition: Ones-Kosten

Die Anzahl der Einsen eines binären Codewortes a wird im folgenden als *Ones* bezeichnet. In der Literatur wurde keine spezielle Bezeichnung gefunden, deshalb wird dieser neue Name eingeführt. Berechnet wird die Anzahl der Einsen eines binären Codewortes a mit der Formel $w(a)$:

$$w(a) = \sum_{i=0}^{n-1} a_i \quad , \text{mit } a = \sum_{i=0}^{n-1} 2^i \cdot a_i$$

Approximation von Hamming-Distanz- und Ones-Kosten

Die Einflüsse von Hamming-Distanz- und Ones-Werten auf den Bussen auf die Energie wird als proportional aufgefasst. Die beiden Einzelkosten werden jeweils durch Hinzufügen von Proportionalitätskonstanten, beispielsweise α und β, berechnet: $E_{hamming}(a, b) = \alpha \cdot h(a, b)$ und $E_{ones}(a) = \beta \cdot w(a)$. Hierbei sind a und b die am Bus anliegenden Codeworte. Die Konstanten α und β sind für den jeweiligen Bus durch Messungen und mit statistischen Methoden empirisch zu bestimmen. Ist eine solche Linearisierung zu ungenau, sind entsprechend andere Approximationen zu benutzen.

Abhängig davon, in welcher Technologie Prozessor und Speicher hergestellt wurden und welche Art von Leitungstreibern für die Busse verwendet wird, kann der Energieverbrauch auch umgekehrt proportional zur Zahl der Einsen bzw. proportional zur Zahl der Nullen sein.

3.2.2 Einzelkosten von Komponenten

Definition: Base-Kosten

Base-Kosten einer Instruktion oder eines Speicherzugriffs sind die Kosten, die, unabhängig von anderen Kostenarten, pro Instruktion und Speicherzugriff anfallen. Sie werden allgemein bezeichnet mit

$$Base$$

Es folgt eine Unterteilung der Base-Kosten nach Instruktionen und Speicherzugriffen.

- Die Base-Kosten einer Instruktion fallen bei der Bearbeitung im Prozessor an. Sie sind spezifisch für jeden Opcode:

  $$BaseCPU(Opcode)$$

- Bei Speicherzugriffen sind die Base-Kosten für jeden getrennt ausgeführten Speicher mem_m zu bestimmen. Beispielsweise könnten Instruktionsspeicher $mem_0 = InstrMem$ und Datenspeicher $mem_1 = DataMem$ gegeben sein.

  Pro Speicher hängen die Base-Kosten ferner von der Richtung *Direction* und der Wortbreite $Word\_width$ des übertragenen Datums ab. Mögliche Werte für *Direction* sind *read* und *write*. Die Wortbreite bestimmt die Anzahl der übertragenen Bits und ist kleiner oder gleich der Breite des Busses.

  $$BaseMem(mem, Direction, Word\_width)$$

Definition: FUChange-Kosten

Diese Kostenart erfaßt für zwei aufeinander folgende Instruktionen $Instr_{i-1}$ und $Instr_i$ Kosten für den Wechsel von Funktionseinheiten. Wenn $Instr_i$ eine Funktionseinheit anspricht, die von $Instr_{i-1}$ nicht verwendet wurde, fallen für diese Funktionseinheit *Aktivierungskosten* an. Andersherum gibt es *Deaktivierungskosten*, wenn $Instr_{i-1}$ eine Funktionseinheit verwendet, die $Instr_i$ nicht benötigt.

Die Mehrkosten, die anfallen, wenn zu zwei aufeinander folgenden Zeitpunkten unterschiedliche Funktionseinheiten angesprochen werden, werden im folgenden ausgedrückt durch $FUChange(Instr_{i-1}, Instr_i)$:

Gegeben seien n Funktionseinheiten $F_1, \ldots, F_n$.

$$\begin{aligned} FUChange(Instr_{i-1}, Instr_i) = \sum_{j=1}^{n} \Big(\\ \gamma(F_j) \cdot \overline{FU(F_j, Instr_{i-1})} \cdot FU(F_j, Instr_i) + \\ \delta(F_j) \cdot FU(F_j, Instr_{i-1}) \cdot \overline{FU(F_j, Instr_i)} \Big) \end{aligned}$$

mit

$$FU(F, Instr) = \begin{cases} 1 & \text{, falls } Instr \text{ Funktionseinheit } F \text{ verwendet} \\ 0 & \text{, sonst} \end{cases}$$

Die Funktionen γ und δ liefern für die übergebene Funktionseinheit Aktivierungs- und Deaktivierungskosten.

Anmerkung: *FU* entspricht engl.: F̲unctional U̲nit.

In der Arbeit [DAMT00] wird der Begriff 'Inter-Instruction-Kosten' anstatt 'FUChange-Kosten' verwendet. *Inter-Instruction*-Kosten erfassen die Kosten zwischen zwei Instruktionen. Demnach gehören auch zusätzliche Buskosten, z.B. in Form von Hamming-Distanz-Kosten zwischen Instruction-Fetches, zu Inter-Instruction-Kosten. Zur besseren Unterscheidung wird in dieser Arbeit nicht der allgemeine Begriff Inter-Instruction verwendet, sondern anstelle dessen die jeweilige Kostenart.

3.3 Modellbeschreibung

Systembestandteile

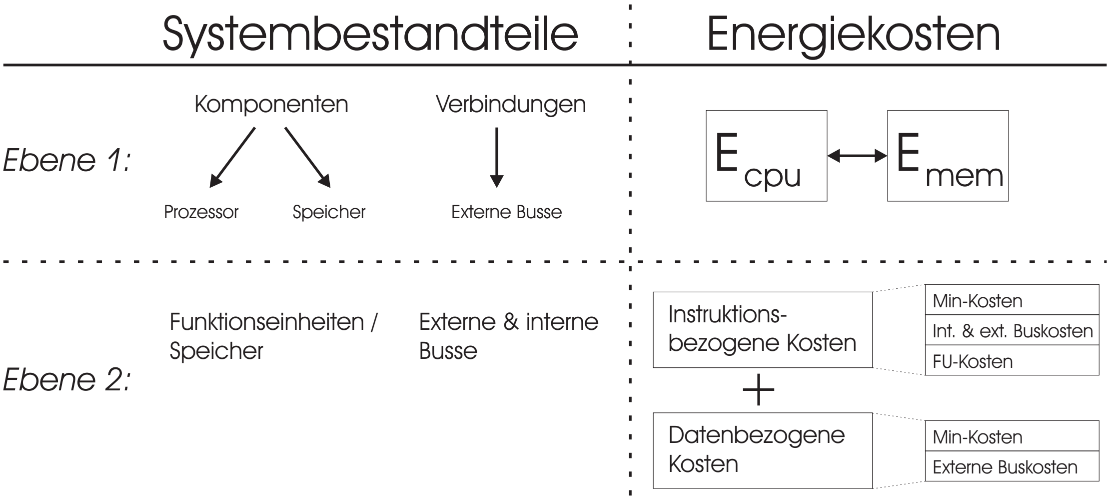


Abbildung 3.1: Systembestandteile und Energiekosten

In der ersten Ebene aus Abbildung 3.1, einer sehr vereinfachten Sicht, besteht das Modell aus den Komponenten Prozessor und Speicher, welche durch externe Busse miteinander verbunden sind. Entsprechend dem physikalisch getrennten Vorhandensein der Komponenten erfolgt die Erfassung der Energiekosten separat für Prozessor und Speicher. Auch wenn der Energieverbrauch der externen Busse nicht direkt gemessen werden kann, findet eine Erfassung trotzdem statt. Energiekosten fallen im Prozessor z.B. beim Anlegen einer Speicheradresse auf den Adressbus an. Im Speicher treten Buskosten beispielsweise auf, wenn zu übertragende Daten am Datenbus angelegt werden.

Bei Ebene 2 handelt es sich um eine detailliertere Sichtweise. Innerhalb des Prozessors werden zusätzlich die einzelnen Funktionseinheiten berücksichtigt. Die Weiterleitung der Daten zu den Funktionseinheiten innerhalb des Prozessors geschieht auf prozessorinternen Bussen. Bei der Erfassung der anfallenden Energiekosten gibt es neben der Aufteilung in Prozessor- und Speicherkosten zusätzlich eine Unterscheidung zwischen *datenbezogenen* und *instruktionsbezogenen* Kosten.

Datenbezogene Kosten erfassen alle Einzelkosten, die durch Zugriff auf einen Datenspeicher verursacht werden. Bei gemeinsamem Daten- und Instruktionsspeicher werden Kosten des Instruction-Fetches den datenbezogenen Kosten zugeordnet, bei getrenntem Speicher den instruktionsbezogenen Kosten.

Alle nicht datenbezogene Kosten werden den instruktionsbezogenen Kosten zugerechnet. Im einzelnen sind dies beispielsweise Kosten auf prozessorinternen Bussen und Kosten für die von der Instruktion verwendeten Funktionseinheiten.

Zur Bestimmung des Energieverbrauchs einer Programm-Sequenz werden Folgen von Instruktionen und Datenzugriffen betrachtet.

Instruktionen

Gegeben sei eine endliche Folge von N Prozessor-Instruktionen in der Reihenfolge ihrer Ausführung. Die Eigenschaften einer Instruktion $Instr_i,\ i \in (0, ..., N-1)$, sind gegeben durch:

- $IAddr_i$

  Dies ist die Adresse der Instruktion im Instruktionsspeicher. Der Wert wird auf dem Bus $IAddr$ angelegt.

- $IData_i$

  Über den Datenbus $IData$ des Instruktions-Datenbusses wird die Instruktion in den Prozessor geladen. Eine Instruktion setzt sich aus den drei Bestandteilen *Opcode*, *Immediate* und *Register* zusammen.

- $Opcode_i$

  Der Opcode einer Instruktion gibt den auszuführenden Befehl an. Er ergibt sich aus dem gesamten Instruktions-Wort abzüglich der Register- und Immediate-Codierungen.

- $Imm_{i,j}$

  Pro Instruktion gibt es, abhängig vom Opcode, j Immediate-Werte.

- $Reg_{i,k}$

  Eine Instruktion kann k Register-Angaben enthalten.

- $RegVal_{i,k}$

  Zu jedem Register existiert ein entsprechender Registerwert bzw. gleichbedeutend, ein Registerinhalt.

- $F_{i,l}$

  Eine Instruktion verwendet mindestens eine Funktionseinheit, allgemein jedoch l Stück.

Datenzugriffe

Gegeben sei eine endliche Folge von M Datenzugriffen in der Reihenfolge der Zugriffe. Es werden nur die Datenzugriffe auf den Datenspeicher $DataMem$ betrachtet. Zugriffe auf den Instruktionsspeicher $InstrMem$ wurden bereits einer Instruktion zugeordnet. Jeder Datenzugriff $DataAccess_i,\ i \in (0, ..., M-1)$, hat folgende Eigenschaften:

- $DAddr_i$

  $DAddr_i$ gibt die Adresse der zu lesenden oder schreibenden Speicherzellen an.

- $Direction_i$

  Angabe, ob der Speicherzugriff lesend oder schreibend ist, bzw. *read* oder *write*.

- $Word\_width_i$

  Die Wortbreite eines Datenzugriffs auf dem *Data*-Bus ist maximal so groß wie die Busbreite. Sie kann aber auch kleiner sein, wenn z.B. nur ein Halbwort- oder Byte-weiser Zugriff erfolgt.

- $Data_i$

  Dieser Parameter gibt das Datum an, welches auf dem Bus *Data* übertragen wird. Die Übertragung findet entsprechend den Werten von *Direction* und $Word\_width$ statt.

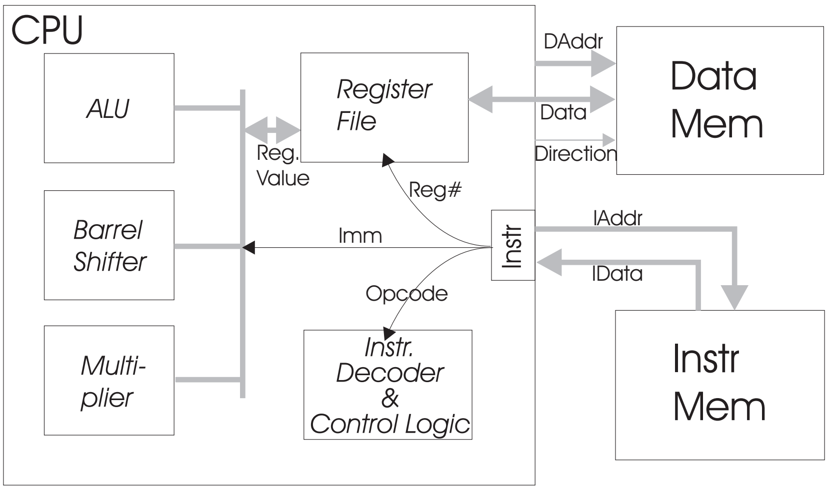


Abbildung 3.2: Blockschaltbild zum Modell eines RISC-Prozessorsystems

Funktionsweise des Modells

Das Modell des verwendeten Prozessorsystems ist in Abbildung 3.2 dargestellt. Typisch für eine Harvard-Architektur gibt es für Daten und Instruktionen jeweils einen eigenen Speicher mit physikalisch getrennten Bussen. Durch Anlegen einer Instruktions-Adresse *IAddr* bekommt der Prozessor auf dem Datenbus *IData* eine Instruktion. Die einzelnen Bestandteile *Opcode*, *Immediate* und *Register* der Instruktion werden zur Weiterverarbeitung an die entsprechenden Stellen weitergeleitet.

- Der *Opcode* wird im *InstructionDecoder* dekodiert. Die *ControlLogic* ist für die Koordination der Abarbeitung des Befehls zuständig. Sie aktiviert u.a. die von der Anweisung benötigten Funktionseinheiten. In diesem Modell gib es ALU, Barrel Shifter und Multiplier. Selbstverständlich sind auch weitere Funktionseinheiten möglich.

- *Immediate*-Werte werden an die entsprechende Funktionseinheit weitergeleitet.

- Entsprechend einer Load-/Store-Architektur greifen Funktionseinheiten nicht direkt auf den Datenspeicher zu, sondern über ein *RegisterFile*. Von dort werden die benötigten Werte aus den Registern an die Funktionseinheiten weitergeleitet. In der anderen Richtung werden Rückgabewerte in die entsprechenden Register im Register File wieder zurückgeschrieben. Das *RegisterFile* führt Zugriffe auf den Datenspeicher durch. Dazu wird über den Adressbus *DAddr* eine Speicherzelle adressiert. Die *Direction*-Leitung bestimmt, ob über den Datenbus *Data* lesender oder schreibender Speicherzugriff stattfinden soll.

Gemäß Abbildung 3.1 werden Einzelkosten im Prozessor und Speicher jeweils instruktions- und datenbezogen erfaßt. Es wird die Summe über alle Einzelkosten von N Instruktionen und M Datenzugriffen gebildet. Die Berechnung erfolgt in den beiden nächsten Abschnitten 3.3.1 und 3.3.2.

3.3.1 Prozessorkosten

Erfaßt werden alle im Prozessor anfallenden Einzelkosten. Zum einen sind dies instruktionsbezogene Kosten: Base-Kosten (Term 3.1), prozessorinterne Buskosten (3.2 bis 3.4 und 3.6 bis 3.8), Kosten zum Anlegen der Instruction-Fetch-Adresse (3.5 und 3.9) und FUChange-Kosten (3.10).

Zum anderen fallen datenbezogene Kosten an: Die Terme 3.11 bis 3.14 erfassen alle Buskosten zum Datenspeicher, die beim Anlegen von Datenadresse und Daten entstehen. Datenzugriffe über den Datenbus werden entweder lesend oder schreibend durchgeführt. Die beiden Terme 3.12 und 3.14 berücksichtigen dies, indem die Konstanten $\alpha_{6,dir}$ und $\beta_{6,dir}$ ein *dir* enthalten. Mögliche Werte für *dir* sind *read* oder *write*.

Instruktionsbezogene Kosten in der CPU

$$E_{cpu\_instr} = \sum_{i=0}^{N-1} \Big($$

$$BaseCPU(Opcode_i) + \tag{3.1}$$

$$\alpha_1 \cdot \sum_j w(Imm_{i,j}) + \tag{3.2}$$

$$\alpha_2 \cdot \sum_k w(Reg_{i,k}) + \tag{3.3}$$

$$\alpha_3 \cdot \sum_k w(RegVal_{i,k}) + \tag{3.4}$$

$$\alpha_4 \cdot w(IAddr_i) + \tag{3.5}$$

$$\beta_1 \cdot \sum_j h(Imm_{i-1,j}, Imm_{i,j}) + \tag{3.6}$$

$$\beta_2 \cdot \sum_k h(Reg_{i-1,k}, Reg_{i,k}) + \tag{3.7}$$

$$\beta_3 \cdot \sum_k h(RegVal_{i-1,k}, RegVal_{i,k}) + \tag{3.8}$$

$$\beta_4 \cdot h(IAddr_{i-1}, IAddr_i) + \tag{3.9}$$

$$FUChange(Instr_{i-1}, Instr_i)\Big) \tag{3.10}$$

Datenbezogene Kosten in der CPU

$$
\begin{aligned}
E_{cpu\_data} = \sum_{i=0}^{M-1} \Big(& \\
& \alpha_5 \cdot w(DAddr_i) + && (3.11)\\
& \alpha_{6,dir} \cdot w(Data_i) + && (3.12)\\
& \beta_5 \cdot h(DAddr_{i-1}, DAddr_i) + && (3.13)\\
& \beta_{6,dir} \cdot h(Data_{i-1}, Data_i)\Big) && (3.14)
\end{aligned}
$$

3.3.2 Speicherkosten

Die Funktionsweise des Speichers ist im Vergleich zum Prozessor wesentlich einfacher. Deshalb erweist sich die Erfassung der Einzelkosten als leichtere Aufgabe. Trotzdem wird zwischen instruktions- und datenbezogenen Kosten unterschieden. Wegen getrenntem Instruktions- und Datenspeicher werden die Base-Kosten für beide Speicher erfaßt (Terme 3.15 und 3.20), wobei auf ersteren Speicher nur lesend zugegriffen wird. Die Buskosten werden in den Termen 3.16 bis 3.19 und 3.21 bis 3.24 berücksichtigt. Ebenso wie bei datenbezogenen Kosten E_{cpu_data} in der CPU findet bei den datenbezogenen Kosten im Datenspeicher eine Unterscheidung zwischen lesendem oder schreibendem Zugriff statt. Die Konstanten $\alpha_{10,dir}$ und $\beta_{10,dir}$ (Terme 3.22 und 3.24) werden jeweils durch Angabe von $dir = read$ und $dir = write$ gemäß der Zugriffsrichtung unterschieden.

Instruktionsbezogene Kosten im Speicher InstrMem

$$
\begin{aligned}
E_{mem\_instr} = \sum_{i=0}^{N-1} \Big(& \\
& BaseMem(InstrMem, Direction = read, Word\_width_i) + && (3.15)\\
& \alpha_7 \cdot w(IAddr_i) + && (3.16)\\
& \alpha_8 \cdot w(IData_i) + && (3.17)\\
& \beta_7 \cdot h(IAddr_{i-1}, IAddr_i) + && (3.18)\\
& \beta_8 \cdot h(IData_{i-1}, IData_i)\Big) && (3.19)
\end{aligned}
$$

Datenbezogene Kosten im Speicher DataMem

$$E_{mem\_data} = \sum_{i=0}^{M-1} \Big($$
$$BaseMem(DataMem, Direction_i, Word\_width_i) + \quad (3.20)$$
$$\alpha_9 \cdot w(DAddr_i) + \quad (3.21)$$
$$\alpha_{10,dir} \cdot w(Data_i) + \quad (3.22)$$
$$\beta_9 \cdot h(DAddr_{i-1}, DAddr_i) + \quad (3.23)$$
$$\beta_{10,dir} \cdot h(Data_{i-1}, Data_i)\Big) \quad (3.24)$$

3.3.3 Gesamtkosten

Die Gesamtkosten eines Programmabschnitts berechnen sich als Summe von Prozessor- und Speicherkosten über alle betroffenen N Instruktionen und M Datenzugriffe.

$$E_{total} = E_{cpu\_instr} + E_{cpu\_data} + E_{mem\_instr} + E_{mem\_data}$$

3.4 Abschlussdiskussion

Jedes Modell versucht, mit Hinblick auf das Anwendungsziel, ein System unter gewissen Aspekten nachzubilden. Welche Eigenschaften und wie genau diese modelliert werden, ist stets als Kompromiss zwischen Detailtreue zum Originalsystem, Anwendungszweck und Einfachheit des Modells zu sehen. Aus diesem Grund ist es sinnvoll, ein Modell auf Vor- und Nachteile hin zu untersuchen.

Vorteile

- Keine Herstellerangaben notwendig

  Die vom Hersteller erhältlichen Angaben für den Energieverbrauch sind meist nur Durchschnittsangaben. Für eine Energieberechnung auf Transistorebene wäre die Kenntnis des Aufbaus, z.B. in Form einer Hardwarebeschreibungssprache, notwendig. Hersteller geben diese Informationen wegen Geheimhaltung meist nicht heraus. Das Modell kommt vollkommen ohne diese technischen Daten aus.

- Datenerhebungen einfach durchzuführen

  Eine Datenerhebung ist mit relativ einfachen Mitteln wie Strom- und Spannungsmessgeräten durchzuführen. Die Anpassung der Daten für das Energiemodell geschieht durch einfache Umrechnungen.

- Genauigkeit

  Von M. Theokharidis [DAMT00] wurde bereits ein Energiemodell aufgestellt, für welches sich Genauigkeitswerte von 1,7% gezeigt haben. Dieses Ergebnis beruht auf dem Unterschied zwischen realen Strommessungen während der Ausführung einer Instruktionsseuqenz und dem berechneten Stromwert der Sequenz durch eine Simulation. Das in dieser Diplomarbeit entwickelte Modell ist im Gegensatz zu dem Modell von Theokharidis feiner modelliert. Beispielsweise werden hier Base-Kosten nicht als Durchschnittswerte betrachtet, die sich durch Messungen mit verschiedenen Operanden ergeben, sondern es wird zwischen Base-Kosten und Kosten auf den einzelnen Bussen unterschieden. Durch eine detailliertere Modellierung in dieser Arbeit ist zu erwarten, dass das vorgestellte Modell genauere Ergebnisse bringt.

- Speicherhierarchie

  Viele Modelle berücksichtigen nur den Prozessor. Energieoptimierungen können aber auch unterschiedliche Kosten einzelner Speicher sowie Kosten für verschiedene Buscodierungen berücksichtigen. Deshalb wurde der Speicher explizit in das Modell aufgenommen.

- Geschwindigkeit

  Berechnungen, die für ein Modell durchgeführt werden, welches auf Transistorebene ansetzt, können schon für relativ kurze Programmsequenzen Tage dauern. Dieses auf Instruktionsebene arbeitende Energiemodell arbeitet mit wenigen und einfachen Rechenoperationen. Berechnungen, z.B. für Simulationen, dauern meist nur wenige Sekunden.

Nachteile

- Keine Berücksichtigung von Netzteil und zusätzlicher Peripherie-Geräte

  Das Modell berücksichtigt kein Netzteil, welches die notwendigen Spannungsversorgungen für das Prozessorsystem bereit stellt. Je nach Arbeitspunkt eines Netzteils kann es unterschiedliche Wirkungsgrade haben. Dies bedeutet, dass bei einem linearen Anstieg des Energieverbrauchs des Systems ohne Netzteil der gesamte Energieverbrauch des Systems mit Netzteil nicht mehr linear ansteigt. Somit ist kein direkter Rückschluss möglich, wieviel Energie ein System mit Netzteil verbraucht.

  Unbeachtet bleiben ebenso Peripheriegeräte wie Speichermedien, Anzeigen usw.

- Frequenz- und spannungsabhängig

  Als Annahme für das Modell sind Spannungsversorgungen und Taktfrequenzen von Prozessor und Speicher konstant. Einige moderne Prozessoren enthalten zur Energieeinsparung neue Techniken, bei denen während des Betriebs Frequenz oder Spannung verändert werden können. Ein System, welches bei einer anderen Spannung oder Frequenz betrieben wird, als wie Datenerhebungen durchgeführt wurden, hat völlig andere Energieverluste.

- Approximation von Bus-Kosten

  Die Einflüsse von Ones- und Hamming-Distanz-Kosten werden als linear vorausgesetzt. Bei Anwendung des Modell auf ein System ist zu überprüfen, ob die Approximation genau genug ist.

Kapitel 4

Anwendung des Modells auf den ARM7TDMI

Im vorherigen Kapitel wurde ein neues Energiemodell vorgestellt. Es wurde Wert darauf gelegt, dass es für eine möglichst große Klasse von Prozessorsystemen benutzbar ist. Nachfolgend wird das Modell beispielhaft auf den RISC-Prozessor ARM7TDMI angewandt. Eingesetzt wird das in Abschnitt 2.3.2 beschriebene Evaluationboard. Es folgt eine Beschreibung, wie mittels geeigneter Testfälle Daten erhoben werden. Die gewonnenen Daten werden mit Hilfe von statistischen Methoden, im wesentlichen mit der linearen Regressionsanalyse, in für das Modell passende Daten umgerechnet. Insgesamt lässt sich die Vorgehensweise in drei Schritte aufteilen:

1. **Durchführung von Messungen**

   Ziel der Durchführung der Messungen soll es sein, Daten zu erheben, die mittels weiterer Umrechungen die Parameter α_1 bis α_{10} und β_1 bis β_{10} des Energiemodells ergeben. Zur Durchführung der Versuche wird ein einfach zu realisierender Versuchsaufbau vorgestellt. Die Testanordnung unterteilt sich in einen Hardware- und einen Softwareteil. Einzelne Testfälle, mit denen Messungen durchgeführt werden sollen, werden in kleine Assembler-Programme eingebettet. Beim Ablauf der Programme werden mit einem Messgerät am Hardwareaufbau Daten gewonnen.

2. **Datenumrechnung**

   Die durchgeführten Messreihen beinhalten nicht immer nur die zu messenden Eigenschaften, sondern sind oft abhängig von anderen Einzelkosten. Die Herausrechnung dieser Abhängigkeiten ist Aufgabe dieses Schrittes. Sollten Daten nicht komplett messbar sein oder der Einfachheit wegen eine

Erfassung nicht gewünscht sein, ist eine Inter- oder Extrapolation vorhandener Messdaten notwendig. Denkbar wäre es, bei einem Speicherlayout, welches aus mehreren Speicherbausteinen besteht, Messungen an nur einem Baustein durchzuführen. Die Daten können anschließend auf die anderen Bausteine extrapoliert werden.

3. **Bestimmung der Parameter des Energiemodells**

   Im letzten Schritt werden aus den bearbeiteten Datenreihen die Parameter für das Energiemodell bestimmt.

4.1 Durchführung von Messungen

Die folgenden Abschnitte 4.1.1 bis 4.4.1 beschreiben für die Konstanten α_4 bis α_{10} und β_4 bis β_{10} des Energiemodells die Schritte der Datenerhebung, Umrechnung und Bestimmung der eigentlichen Konstanten. Angewendet werden die Verfahren auf den Prozessor und den externen Speicher.

4.1.1 Versuchsaufbau

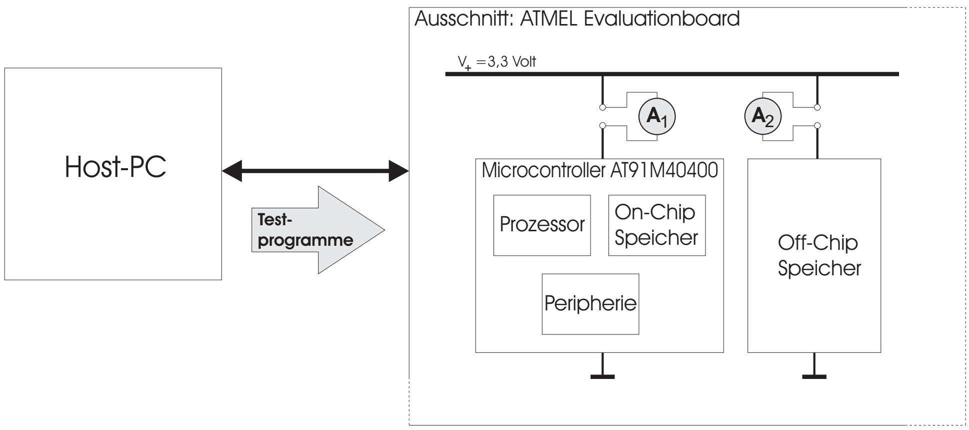


Abbildung 4.1: Versuchsaufbau

Hardware

Abbildung 4.1 zeigt den Aufbau der Versuchsanordnung. Es wird das Evaluationboard AT91EB01 von Atmel verwendet. Zur Gewinnung von Messdaten werden Strom-Messungen an Prozessor und externem Speicher durchgeführt. Dazu werden auf dem Evaluationboard die Leitungen der Versorgungsspannungen vom Prozessor-Chip und einem externen Speicherbaustein aufgetrennt und jeweils ein Amperemeter in Serie geschaltet. Zur Durchführung der Messungen wird das Digitalmultimeter 'Escort-95' der Firma ESCORT verwendet. Im Bereich der durchgeführten Strommessungen besitzt das Gerät eine Genauigkeit von ±0,2% [COS01].

Zu berücksichtigen ist, dass der Prozessor-Core nicht alleine auf einem Chip sitzt, sondern in einem Microcontroller mit On-Chip-Speicher sowie zusätzlicher Peripherie integriert ist. Somit können am Prozessor alleine keine isolierten Strom-Messungen vorgenommen werden, es wird automatisch der Stromverbrauch des gesamten Chips gemessen. Die Anteile des Stroms von On-Chip-Speicher und Peripherie werden dem Prozessor zugerechnet. Im folgenden bezieht sich bei Energiemessungen das Wort 'Prozessor' immer auf die Kombination von Prozessor und den zusätzlichen Einheiten.

Software

Zur Durchführung der Messungen werden Messmodule verwendet. Bei den Modulen handelt es sich um kleine Assembler-Programme, die für den THUMB-Modus des ARM7TDMI geschrieben sind. Mit den Messmodulen und den Amperemetern wird der durchschnittliche Stromverbrauch von kleinen Instruktionssequenzen gemessen, die aus maximal ca. 10 Befehlen bestehen können.

Die Module führen in einer Endlosschleife die zu messende Instruktionssequenz aus. Um die Messeinflüsse des Sprungbefehls am Ende der Endlosschleife möglichst gering zu halten, wird innerhalb der Schleife die Sequenz so oft wiederholt, dass die Schleife wenigstens aus 100 Instruktionen besteht. In der Diplomarbeit von M. Theokharidis [DAMT00] hat sich gezeigt, dass diese Form der Messungen genau genug ist. Bei der Ausführung von 100 im Vergleich zu 1000 gleichen Instruktionen innerhalb der Schleife ergeben sich keine Messunterschiede.

Bei der Ausführung der Module ergeben die an den beiden Multimetern angezeigten Werte jeweils einen Messwert für den Prozessor und einen Messwert für den Speicher.

Es folgt eine Beschreibung der beiden Messmodule zur Ermittlung von Ones- und Hamming-Distanz-Kosten.

Messmodul 1: Ones-Kosten

Mit dem nachfolgenden Messmodul werden Messreihen aufgestellt, die Ones-Kosten auf dem Daten- und Adressbus für *lesenden* Zugriff erfassen.

Im Abschnitt `header` werden Stackpointer und Datenbereich festgelegt. Anschließend findet ein Wechsel vom ARM-Modus in den THUMB-Modus statt. In `start` werden zu untersuchende Testmuster definiert, die im Register `r1` abgelegt sowie ab der Speicheradresse geschrieben werden, die im vorherigen Abschnitt festgelegt wurde. `loop` ist die Endlosschleife, in der die zu messende Instruktionssequenz steht. Hier handelt es sich um einen `LDRH`-Befehl, der Zugriffe auf den Adress- und Datenbus durchführt.

Zur Ermittlung der Messreihen werden die in Register `r2` und `r1` geladenen Werte geändert (gekennzeichnet durch Kommentarzeilen '`*** (1)`' und '`*** (2)`'). Zeile '`*** (1)`' definiert, welcher Wert auf dem Adressbus übertragen wird. Zeile '`*** (2)`' bestimmt den Wert auf dem Datenbus. Welche Testmuster hier jeweils verwendet werden, ist in Abschnitt 4.1.3 beschrieben.

Zur Ermittlung von Messreihen für *schreibenden* Zugriff auf dem Datenbus werden im Abschnitt `loop` die `LDRH`-Instruktionen durch `STRH`-Instruktionen ausgetauscht.

```
    AREA Benchlib, CODE, READONLY
    ENTRY

header
    MOV sp, #0x200      ; Stackpointer liegt im ON-CHIP RAM
    LDR r2, =0x02040000 ; *** (1) Datenbereich Off-Chip (Adressbus)
    MOV r3, #0x00000    ; Offset Datenbereich (immer 0 lassen)

    ADR r0, start +1    ; in THUMB-Modus wechseln
    BX r0

    CODE16

start
    LDR r1, =0x0000     ; *** (2) Verwendetes Testmuster (Datenbus)
    STR r1, [r2,r3]     ; Testmuster speichern

loop
    LDRH r1, [r2,r3]   ; 1. Instruktion
    LDRH r1, [r2,r3]   ; 2. Instruktion
...
    LDRH r1, [r2,r3]   ; 99. Instruktion
    LDRH r1, [r2,r3]   ; 100. Instruktion

    B loop
```

Messmodul 2: Hamming-Distanz-Kosten

Dieses Modul ist ähnlich dem vorherigen aufgebaut. Anstatt einer `LDRH`-Anweisung in der Endlosschleife werden zwei verwendet, die auf unterschiedliche Speicherbereiche zugreifen und verschiedene Daten auf dem Datenbus übertragen. Die Angabe von Adressbereichen und Daten geschieht in den vier Zeilen, die mit den Kommentaren '`*** (1)`' bis '`*** (4)`' markiert sind.

Das abgedruckte Modul führt *lesende* Zugriffe aus. Durch Ersetzung der `LDRH`-Instruktionen mit `STRH`-Instruktionen werden *schreibende* Zugriffe durchgeführt.

```
    AREA Benchlib, CODE, READONLY
    ENTRY

header
    MOV sp, #0x200      ; Stackpointer liegt im ON-CHIP RAM
    LDR r2, =0x02040000 ; *** (1) Datenbereich Off-Chip (Adressbus)
    LDR r4, =0x02040002 ; *** (2) 2. Datenbereich Off-Chip (Adressbus)
    MOV r3, #0x00000    ; Offset Datenbereich (immer 0 lassen)

    ADR r0, start +1    ; in THUMB-Modus wechseln
    BX r0

    CODE16

start
    LDR r1, =0x0000     ; *** (3) Verwendetes Testmuster (Datenbus)
    LDR r5, =0x0000     ; *** (4) 2. verwendetes Testmuster (Datenbus)
    STR r1, [r2,r3]     ; Testmuster speichern
    STR r5, [r4,r3]     ; 2. Testmuster speichern

loop
    LDRH r1, [r2,r3]   ; 1. Instruktion
    LDRH r5, [r4,r3]   ; 2. Instruktion
  ...
    LDRH r1, [r2,r3]   ; 99. Instruktion
    LDRH r5, [r4,r3]   ; 100. Instruktion

    B loop
```

Speicherorganisation

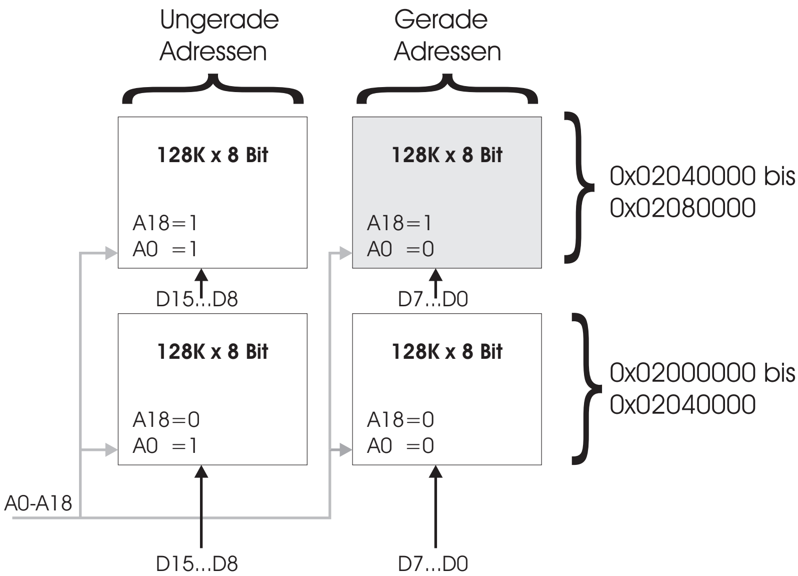


Abbildung 4.2: Speicherlayout des Off-Chip-Speichers

Der Datenbus zum Off-Chip-Speicher ist 16 Bit breit. Deshalb werden in den verwendeten Testmodulen lesende und schreibende Speicherzugriffe als 16 Bit breite halfword-Zugriffe realisiert, d.h. es werden die THUMB-Instruktionen `LDRH` und `STRH` benutzt. Die Größe des externen Speichers auf dem eingesetzten Atmel-Board beträgt 512 KByte. Standardmäßig wird der Speicher in den Bereich 0x02000000 bis 0x02080000 gemappt [ATM99c]. Physikalisch gesehen besteht der vorliegende Speicher aus vier gleichen Bausteinen. Bei der Speicherorganisation sind jeweils zwei Bausteine für die untere oder obere Hälfte der gesamten 512 KByte zuständig. Einer der jeweiligen Bausteine hält die Daten für gerade, der andere für ungerade Adressen. Messungen werden nur an einem Baustein durchgeführt. Gemessen wird am Chip, welcher für gerade Adressen im Bereich 0x02040000 bis 0x02080000 zuständig ist. Demnach ist das LSB bei 2 Byte aligned Adressen immer '0'.

4.1.2 Platzierung von Instruktions- und Datenspeicher

Um die Messwerte für Daten- und Adressbus möglichst unabhängig und isoliert bestimmen zu können, werden Instruktionen im On-Chip-Speicher abgelegt und Datenzugriffe im Off-Chip-Speicher durchgeführt. Somit wird vermieden, dass Instruktionsworte in der Instruction-Fetch-Phase die Zustände auf den externen Bussen verändern. Durch einen geringeren Energieverbrauch des internen Speichers gegenüber dem externen Speicher werden die Einflüsse des Instruction-Fetch minimal gehalten.

4.1.3 Testreihen

Im Abschnitt 4.1.1 wurden Messmodule für die Erfassung von Ones- und Hamming-Distanz-Kosten vorgestellt. Diese Module dienen als Ausgangsbasis für die Gewinnung von einzelnen Messwerten. Dazu werden im Messmodul die mit '`*** ( )`' gekennzeichneten Zeilen systematisch mit verschiedenen Testdaten, den sogenannten *Testmustern*, variiert. Bei Betrachtung aller Belegungsmöglichkeiten für einen Bus ergeben sich für die Anzahl von Testfällen folgende Komplexitäten:

Adressbus:	19 Bit Breite: $2^{19} = 524288$ Möglichkeiten
Datenbus:	16 Bit Breite: $2^{16} = 65536$ Möglichkeiten

Diese Anzahlen sowie deren Kombinationen untereinander würden zu zu vielen Testfällen führen. Zur Verringerung der durchzuführenden Datenerhebungen werden die Testreihen für jeden Bus in zwei Teile von Testfällen aufgeteilt:

1. Wertigkeit einer Busleitung

   Jede Busleitung wird einzeln auf '1' gesetzt. Die Daten werden so gewählt, dass im ersten Teil einer Testdatenreihe eine Eins vom LSB (Least Significant Bit) zum MSB (Most Significant Bit) wandert. Mit dieser Teilreihe wird die Wertigkeit einer jeden Busleitung erfaßt. Wenn davon ausgegangen wird, dass alle Busleitungen homogen aufgebaut sind, sollte im Idealfall für jede Leitung der gleiche Messwert herauskommen.

2. Anzahl gesetzter Busleitungen

   Im zweiten Teil der Testreihe wird für jeden Messwert, ausgehend vom LSB, in Richtung des MSBs eine Eins hinzugefügt. Diese Messwerte dienen zur Bestimmung der Koeffizienten α_i und β_i. Idealerweise sollten die Messwerte auf einer Geraden liegen, deren Steigung jeweils einem Koeffizienten entspricht.

Die Tabellen 4.1 bis 4.4 zeigen die Testdaten im einzelnen, wie sie in den beiden aufgeführten Messmodulen verwendet werden. Insgesamt gibt es für den Adressbus *35* Messwerte und für den Datenbus *33*. Es empfiehlt sich, die Testdaten in binärer Darstellung zu betrachten. Dort entspricht jede Bit-Position genau einer Leitung auf dem Bus.

Zu beachten ist, dass für die Testmuster in den Messmodulen das Speicherlayout berücksichtigt wird. Dazu werden Testmuster für den Adressbus erzeugt, die nur einen Speicherbaustein ansprechen. Jede Adresse ist um den Offset 0x02040000 erhöht. Zur Adressierung des Bausteins, welcher nur gerade Adressen aufnimmt, ist außerdem das LSB stets auf '0' gesetzt. In den beiden Tabellen 4.1 und 4.2 entsprechen die Nicht-kursiven Ziffern den Busleitungen an *einem* Speichermodul. Die Angaben in der Spalte 'w' beziehen sich nur auf diese Ziffern.

i	*Binär*	*Hexadezimal*	*w (nicht-kursive Ziffern)*
1	*00000010000001*00000000000000001*0*	0x02040002	1
2	*00000010000001*00000000000000010*0*	0x02040004	1
3	*00000010000001*00000000000000100*0*	0x02040008	1
4	*00000010000001*00000000000001000*0*	0x02040010	1
5	*00000010000001*00000000000010000*0*	0x02040020	1
6	*00000010000001*00000000000100000*0*	0x02040040	1
7	*00000010000001*00000000001000000*0*	0x02040080	1
8	*00000010000001*00000000010000000*0*	0x02040100	1
9	*00000010000001*00000000100000000*0*	0x02040200	1
10	*00000010000001*00000001000000000*0*	0x02040400	1
11	*00000010000001*00000010000000000*0*	0x02040800	1
12	*00000010000001*00000100000000000*0*	0x02041000	1
13	*00000010000001*00001000000000000*0*	0x02042000	1
14	*00000010000001*00010000000000000*0*	0x02044000	1
15	*00000010000001*00100000000000000*0*	0x02048000	1
16	*00000010000001*01000000000000000*0*	0x02050000	1
17	*00000010000001*10000000000000000*0*	0x02060000	1

Tabelle 4.1: Testmuster für den Adressbus: Wertigkeit einzelner Einsen

i	*Binär*	*Hexadezimal*	*w (nicht-kursive Ziffern)*
18	*00000010000001*00000000000000000*0*	0x02040000	0
19	*00000010000001*00000000000000001*0*	0x02040002	1
20	*00000010000001*00000000000000011*0*	0x02040006	2
21	*00000010000001*00000000000000111*0*	0x0204000E	3
22	*00000010000001*00000000000001111*0*	0x0204001E	4
23	*00000010000001*00000000000011111*0*	0x0204003E	5
24	*00000010000001*00000000000111111*0*	0x0204007E	6
25	*00000010000001*00000000001111111*0*	0x020400FE	7
26	*00000010000001*00000000011111111*0*	0x020401FE	8
27	*00000010000001*00000000111111111*0*	0x020403FE	9
28	*00000010000001*00000001111111111*0*	0x020407FE	10
29	*00000010000001*00000011111111111*0*	0x02040FFE	11
30	*00000010000001*00000111111111111*0*	0x02041FFE	12
31	*00000010000001*00001111111111111*0*	0x02043FFE	13
32	*00000010000001*00011111111111111*0*	0x02047FFE	14
33	*00000010000001*00111111111111111*0*	0x0204FFFE	15
34	*00000010000001*01111111111111111*0*	0x0205FFFE	16
35	*00000010000001*11111111111111111*0*	0x0207FFFE	17

Tabelle 4.2: Testmuster für den Adressbus: Anzahl der Einsen

i	*Binär*	*Hexadezimal*	w
1	0000000000000001	0x0001	1
2	0000000000000010	0x0002	1
3	0000000000000100	0x0004	1
4	0000000000001000	0x0008	1
5	0000000000010000	0x0010	1
6	0000000000100000	0x0020	1
7	0000000001000000	0x0040	1
8	0000000010000000	0x0080	1
9	0000000100000000	0x0100	1
10	0000001000000000	0x0200	1
11	0000010000000000	0x0400	1
12	0000100000000000	0x0800	1
13	0001000000000000	0x1000	1
14	0010000000000000	0x2000	1
15	0100000000000000	0x4000	1
16	1000000000000000	0x8000	1

Tabelle 4.3: Testmuster für den Datenbus: Wertigkeit einzelner Einsen

i	*Binär*	*Hexadezimal*	w
17	0000000000000000	0x0000	0
18	0000000000000001	0x0001	1
19	0000000000000011	0x0003	2
20	0000000000000111	0x0007	3
21	0000000000001111	0x000F	4
22	0000000000011111	0x001F	5
23	0000000000111111	0x003F	6
24	0000000001111111	0x007F	7
25	0000000011111111	0x00FF	8
26	0000000111111111	0x01FF	9
27	0000001111111111	0x03FF	10
28	0000011111111111	0x07FF	11
29	0000111111111111	0x0FFF	12
30	0001111111111111	0x1FFF	13
31	0011111111111111	0x3FFF	14
32	0111111111111111	0x7FFF	15
33	1111111111111111	0xFFFF	16

Tabelle 4.4: Testmuster für den Datenbus: Anzahl der Einsen

Die Testmuster für den Datenbus arbeiten auf der vollen 16 Bit Busbreite, welche das Evaluationboard zur Verfügung stellt.

Bezeichnungen

Nachfolgend wird beschrieben, welche Schreibweise für Messreihen und Messwerte verwendet wird.

Bezeichnung von Messreihen: $M_{unit,bus,ek,dir}$

Die einzelnen Indizes haben folgende Bedeutung:

- $unit \in \{cpu, mem\}$

  Bestimmt, ob die Messreihe für die CPU oder den Speicher ist.

- $bus \in \{data, addr\}$

  Gibt den Bus an.

- $ek \in \{w, h\}$

  Bestimmt, welche Einzelkosten die Messreihe enthält, entweder Ones- oder Hamming-Distanz-Kosten.

- $dir \in \{r, w\}$

  Richtung des durchgeführten Datenzugriffs auf dem Datenbus, lesend oder schreibend.

Für jeden der 4 Indizes gibt es 2 Möglichkeiten, also insgesamt $2^4 = 16$ Messreihen. In Abhängigkeit vom betrachteten Bus gehören zu jeder Messreihe $M_{unit,bus,ek,dir}$ entweder 35 oder 33 Messwerte. Eine gesamte Messreihe wird stets mit einem Großbuchstaben bezeichnet, die dazugehörigen Messwerte mit dem entsprechenden Kleinbuchstaben:

$$\text{Messwerte: } \begin{cases} m_{unit,bus,ek,dir_1}, \ldots, m_{unit,bus,ek,dir_{35}} & \text{für } bus = addr \\ m_{unit,bus,ek,dir_1}, \ldots, m_{unit,bus,ek,dir_{33}} & \text{für } bus = data \end{cases}$$

Aufgestellte Reihen

Abbildung 4.3 verdeutlicht, welche Messreihen erhoben und wie diese Reihen weiterberechnet werden.

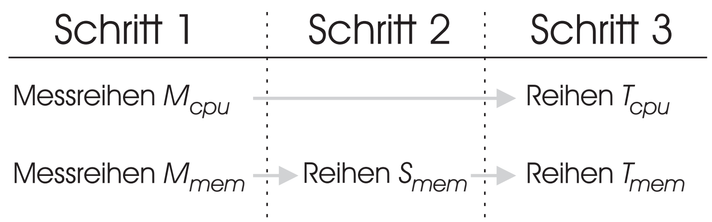


Abbildung 4.3: Bezeichnungen der Reihen

Im einzelnen werden folgende Schritte durchgeführt:

1. Zuerst werden an der CPU und am Speicher Messungen durchgeführt, die Messwerte bilden die Reihen M_{cpu} und M_{mem}.

2. Messungen am Speicher werden nur an einem Speichermodul vorgenommen. Bei einem Speicherzugriff sind aber stets zwei Module beteiligt. Die Umrechnung der Messreihen M_{mem} des Speichers auf zwei Speichermodule in Reihen S_{mem} ist Aufgabe dieses Schritts.

3. Die erhobenen Messreihen M_{cpu} und die errechneten Reihen S_{mem} beinhalten nicht immer nur die zu messenden Eigenschaften, sondern sind oft abhängig von anderen Einzelkosten. Zur Herausrechnung dieser Abhängigkeiten werden die Reihen M_{cpu} und S_{mem} in Reihen T umgerechnet.

Messung von Ones-Kosten

Kosten-Erfassung für	*(1) Datenbereich*	*(2) Testmuster Datenbus*
Adressbus	gem. Tabellen 4.1 und 4.2	0x0000
Datenbus	0x02040000	gem. Tabellen 4.3 und 4.4

Tabelle 4.5: Testmuster zur Messung von Ones-Kosten

Zur Gewinnung von Messdaten für Ones-Kosten auf dem Adress- und Datenbus werden die Werte aus Tabelle 4.5 in Messmodul 1 eingesetzt. Die Zahl in der Tabellenüberschrift gibt die Postion im Messmodul an. Pro Ausführung des Messmoduls wird jeweils ein Messwert für Prozessor und Speicher gewonnen. Alle Messungen sind für lesenden und schreibenden Datenzugriff durchzuführen. Dazu werden, wie bereits im Abschnitt zum Messmodul beschrieben, in der Endlosschleife alle `LDRH`-Befehle durch `STRH`-Befehle ersetzt. Erfasst werden die 8 Messreihen $M_{unit,bus,w,dir}$ mit $unit \in \{cpu, mem\}$, $bus \in \{data, addr\}$ und $dir \in \{r, w\}$. Die Messwerte sind in Anhang A.1 aufgelistet, graphisch dargestellt sind sie in den Abbildungen 4.4 bis 4.7.

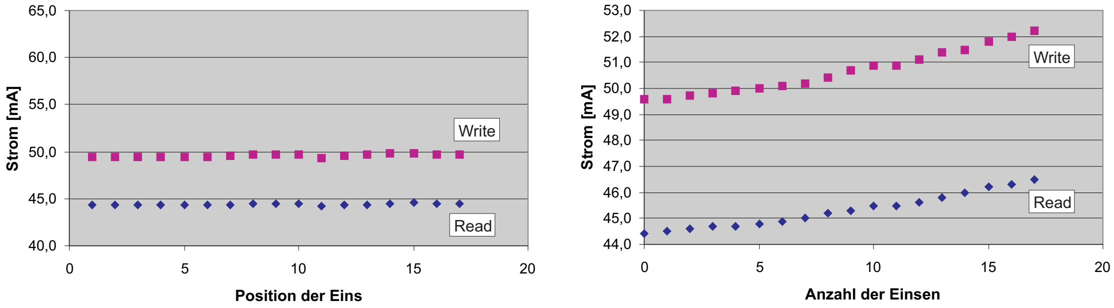


Abbildung 4.4: Messreihen M: CPU: Wertigkeit (*links*) und Anzahl '1'en (*rechts*) auf dem Adressbus

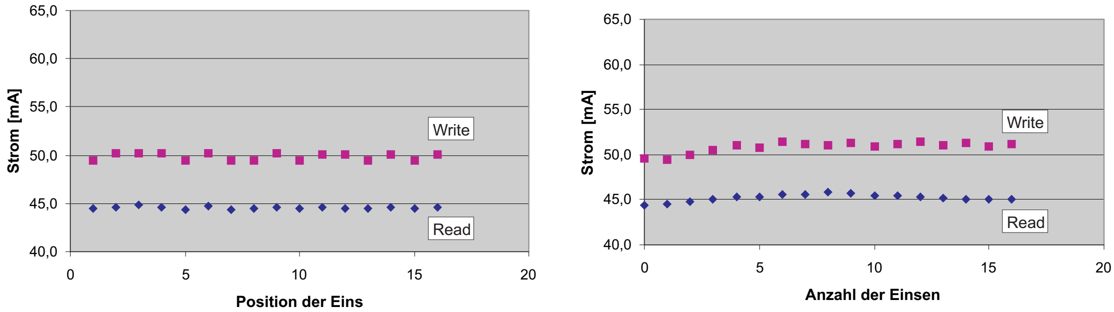


Abbildung 4.5: Messreihen M: CPU: Wertigkeit (*links*) und Anzahl '1'en (*rechts*) auf dem Datenbus

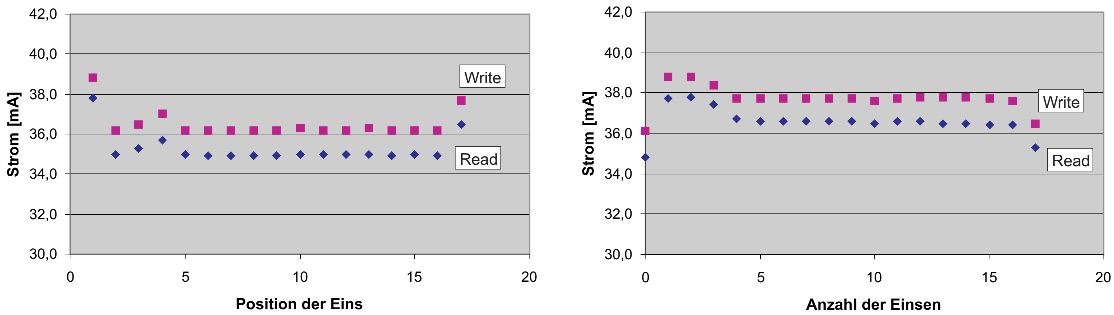
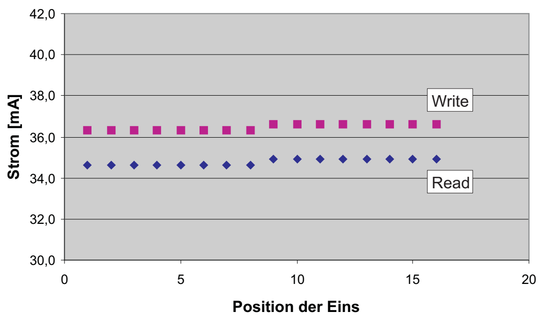


Abbildung 4.6: Messreihen M: Speicher: Wertigkeit (*links*) und Anzahl '1'en (*rechts*) auf dem Adressbus



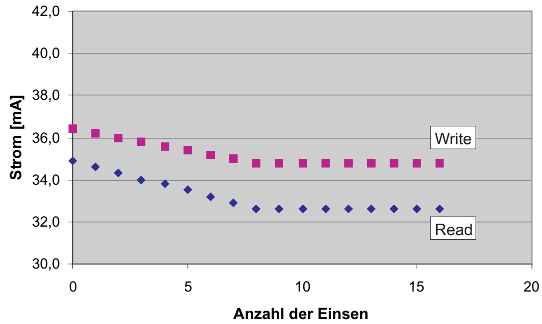


Abbildung 4.7: Messreihen M: Speicher: Wertigkeit (*links*) und Anzahl '1'en (*rechts*) auf dem Datenbus

Messung von Hamming-Distanz-Kosten

Kosten-Erfassung für	*(1) Datenbereich 1*	*(2) Datenbereich 2*	*(3) Testmuster 1 Datenbus*	*(4) Testmuster 2 Datenbus*
Adressbus	0x02040000	gem. Tabellen 4.1 und 4.2	0x0000	0x0000
Datenbus	0x02040000	0x02040002	0x0000	gem. Tabellen 4.3 und 4.4

Tabelle 4.6: Testmuster zur Messung von Hamming-Distanz-Kosten

Tabelle 4.6 zeigt die Testmuster für die Gewinnung von Messreihen für Hamming-Distanz-Kosten. Die Anwendung ist entsprechend dem vorherigen Unterabschnitt durchzuführen. Erfasst werden die 8 Messreihen $M_{unit,bus,h,dir}$ mit $unit \in \{cpu, mem\}$, $bus \in \{data, addr\}$ und $dir \in \{r, w\}$.

Die prinzipielle Vorgehensweise zur Erfassung von Hamming-Distanz-Effekten auf dem Adress- und Datenbus ist jeweils folgende: Eingesetzt wird das Messmodul 2. Innerhalb der Endlosschleife des Moduls werden mit dem ersten Testmuster nur Nullen auf dem entsprechenden Bus angelegt. Anschließend wird ein Testmuster aus den Tabellen 4.1 und 4.2 bzw. 4.3 und 4.4 verwendet. Da das erste Codewort Null ist, ist die Hamming-Distanz zwischen diesen beiden Codeworten gleich der Anzahl der Einsen des zweiten Codewortes.

Bei der Durchführung der Messungen für den Datenbus müssen zur Speicherung der Datenworte zwei unterschiedliche Adressen verwendet werden. Dadurch beinhalten die Messungen zusätzliche Kosten wegen der Hamming-Distanz auf dem Adressbus. Dies ist bei der Weiterverwendung der Daten entsprechend zu berücksichtigen.

Die Abbildungen 4.8 bis 4.11 zeigen die erhobenen Messreihen.

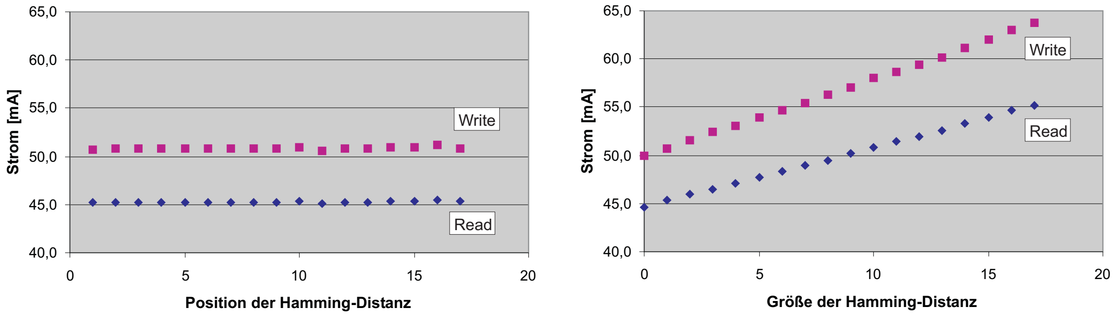


Abbildung 4.8: Messreihen M: CPU: Wertigkeit (*links*) und Größe Hamming-Distanz (*rechts*) auf dem Adressbus

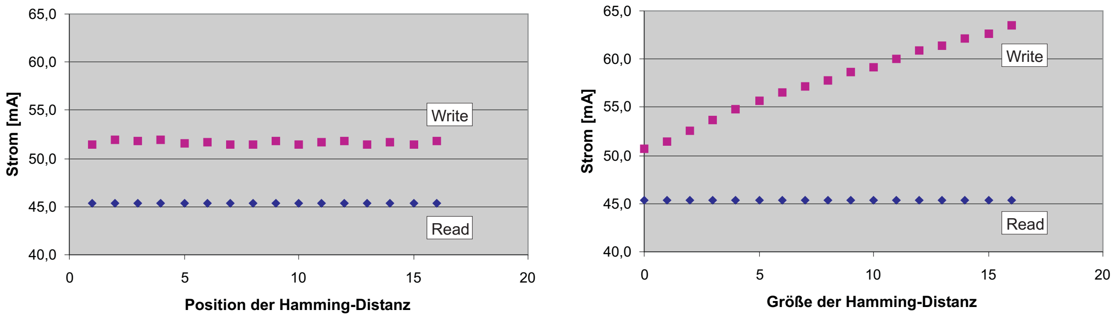


Abbildung 4.9: Messreihen M: CPU: Wertigkeit (*links*) und Größe Hamming-Distanz (*rechts*) auf dem Datenbus

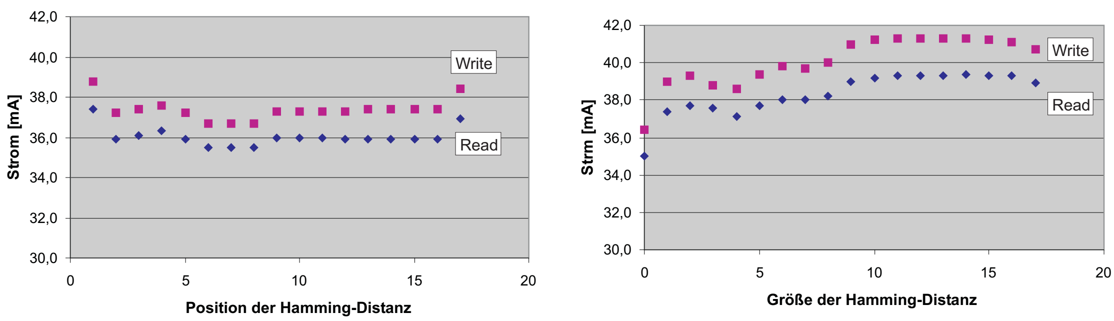


Abbildung 4.10: Messreihen M: Speicher: Wertigkeit (*links*) und Größe Hamming-Distanz (*rechts*) auf dem Adressbus

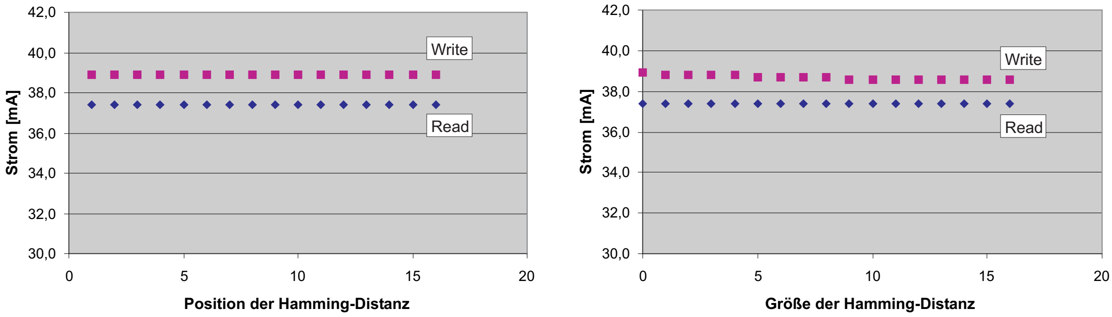


Abbildung 4.11: Messreihen M: Speicher: Wertigkeit (*links*) und Größe Hamming-Distanz (*rechts*) auf dem Datenbus

4.2 Datenumrechnung

4.2.1 Umrechnung der Speicherkosten auf zwei Speichermodule

Messungen am Speicher wurden nur an einem Speichermodul durchgeführt. Bei jedem Speicherzugriff sind aber stets zwei Module beteiligt, da ein 16 Bit breiter Datenbus gegeben ist und jedes Speichermodul 8 Bit Daten aufnimmt. Nachfolgend werden die Messreihen $M_{mem,bus,ek,dir}$ auf die Messreihen $S_{mem,bus,ek,dir}$ für zwei Module extrapoliert. Alle Umrechnungen sind gleichermaßen für $dir = r$ und $dir = w$ durchzuführen. Die Ergebnisse der Umrechnungen sind in Anhang A.2 aufgelistet.

Rechnungen für Ones- und Hamming-Kosten auf dem Adressbus

Für die Umrechnung der Messreihen des Adressbusses ist relevant, dass bei der Adressierung der beiden Speicherbausteine an beide RAMs die gleiche Adresse angelegt wird. Deshalb ist zu erwarten, dass bei der Verwendung gleicher Bausteine jeweils die gleichen Energien bzw. Ströme anfallen. Die einzelnen Messwerte werden demnach verdoppelt:

$$s_{mem,addr,w,dir_i} = 2 \cdot m_{mem,addr,w,dir_i}$$

$$s_{mem,addr,h,dir_i} = 2 \cdot m_{mem,addr,h,dir_i}$$

Rechnungen für Ones-Kosten auf dem Datenbus

Aus Abbildung 4.7 ist sehr gut erkennbar, dass bei Variation der Codierung auf dem Datenbus das Messgerät nur für die ersten 8 Bits bzgl. der Zahl der Einsen unterschiedliche Werte anzeigt. Zur Berechnung der neuen Datenreihen wird zu jedem Messwert i mit Hilfe der Tabelle A.2 aus dem Anhang derjenige Messwert addiert, dessen Codierung am zweiten Speichermodul anlag.

Durchführung für die Wertigkeit der Einsen (Indizes von 1 bis 8): Am zweiten Baustein lagen jeweils nur Nullen an, der zugehörige Index ist 17.

Durchführung für die Wertigkeit der Einsen (Indizes von 9 bis 16): Am zweiten Modul lagen jeweils diejenigen Codierungen an, die für $i = 1, \ldots, 8$ am ersten Modul anlagen.

Durchführung für die Anzahl der Einsen (Indizes von 17 bis 25): Am zweiten Baustein lagen jeweils nur Nullen an, der zugehörige Index ist 17.

Durchführung für die Anzahl der Einsen (Indizes von 26 bis 33): Am zweiten Modul lagen jeweils diejenigen Codierungen an, die für $i = 18, \ldots, 25$ am ersten Modul anlagen.

$$s_{mem,data,w,dir_i} = \begin{cases} m_{mem,data,w,dir_i} + m_{mem,data,w,dir_{17}} & \text{, für } i = 1, \ldots, 8 \\ m_{mem,data,w,dir_i} + m_{mem,data,w,dir_{i-8}} & \text{, für } i = 9, \ldots, 16 \\ \\ m_{mem,data,w,dir_i} + m_{mem,data,w,dir_{17}} & \text{, für } i = 17, \ldots, 25 \\ m_{mem,data,w,dir_i} + m_{mem,data,w,dir_{i-8}} & \text{, für } i = 26, \ldots, 33 \end{cases}$$

Rechnungen für Hamming-Kosten auf dem Datenbus

Abbildung 4.11 zeigt, dass bzgl. Hamming-Distanz für die ersten 8 Bits der Codierungen annähernd die gleichen Werte ermittelt wurden wie für die letzten 8 Bits. Die Messwerte der neuen Reihen ergeben sich durch Verdopplung der Messwerte:

$$s_{mem,data,h,dir_i} = 2 \cdot m_{mem,data,h,dir_i}$$

4.2.2 Umrechnungen zur Ermittlung von Einzelkosten

Die Messreihen M_{cpu} bzw. die berechneten Speicherreihen S_{mem} wurden ermittelt, um mit den Datenwerten der Reihen die Modellparameter zu bestimmen. Allerdings enthalten einzelne Reihen nicht nur der Reihe entsprechende Einzelkosten, sondern zusätzlich andere Einzelkosten E_{other}, die bei Durchführung der Messungen zwangsweise angefallen sind.

Als Beispiel, welche Einzelkosten E_{other} bei Messungen zwangsweise mit erfasst wurden, seien die Ones-Kosten bei Datenreihen zu Hamming-Distanz-Kosten erwähnt. Zur Messung von Hamming-Distanz-Kosten wurden pro Messung zwei binäre Codeworte benutzt, zum einen das Codewort '000...', zum anderen ein Codewort ungleich '000...', welches die Hamming-Distanz bildet. Da nur Effekte der Hamming-Distanz erfasst werden sollen, sind die Ones-Kosten des zweiten Codeworts zu subtrahieren.

Die Errechnung der Reihen T sind für Prozessor und Speicher identisch, deshalb erfolgt nachfolgend eine Erklärung der Einfachheit halber nur für die CPU.

Prozessorkosten

Die Erfassung von Ones-Kosten auf allen Bussen konnte isoliert durchgeführt werden. Es sind keine Umrechnungen erforderlich, die Werte für die neuen Reihen T werden von den Messreihen M direkt übernommen:

$$t_{cpu,addr,w,dir_i} = m_{cpu,addr,w,dir_i}$$

$$t_{cpu,data,w,dir_i} = m_{cpu,data,w,dir_i}$$

Bei den Messreihen für die Hamming-Distanz-Kosten wurden jeweils verschiedene Codierungen auf die Busse gelegt. Für jede Codierung, die ungleich der '000...'-Codierung ist, sind zusätzliche Mehrkosten in Form von Ones-Kosten angefallen. Diese *Mehr*kosten gegenüber den Kosten für das Codewort '000...' sind zu subtrahieren.

Die Instruktionen in der Endlosschleife der Messmodule für die Hamming-Distanz bestehen aus der mehrmaligen Wiederholung zweier Instruktionen. Jede Subtraktion von Ones-Kosten ist deshalb mit $\frac{1}{2}$ zu gewichten. Der Term 4.2 subtrahiert die Mehrkosten der Einsen auf dem Adressbus. Zur Berechnung der Mehrkosten wird die Differenz gebildet zwischen den Ones-Kosten $m_{cpu,addr,w,dir_i}$ des umzurechnenden Messwerts $m_{cpu,addr,h,dir_i}$ und dem Null-Codewort. Das Null-Codewort ergibt sich nach Tabelle A.1 als Messwert $m_{cpu,addr,w,dir_{18}}$ mit dem Index 18.

Term 4.4 subtrahiert die Mehrkosten der Einsen für den Datenbus. Der Index 17 des Messwerts $m_{cpu,data,w,dir_{17}}$ ergibt sich nach Tabelle A.2 als Messwert mit dem Testmuster der Null-Codierung.

Aus Tabelle 4.6, Seite 43 ist ersichtlich, dass bei den Messreihen für die Hamming-Distanz-Kosten auf dem Datenbus auf dem Adressbus eine Hamming-Distanz von 1 vorhanden war. Diese Hamming-Distanz fällt in der Endlosschleife in der 2er-Sequenz genau zweimal Mal an, die Mehrkosten sind demnach mit $\frac{2}{2} = 1$ zu gewichten (Term 4.5). Die zu subtrahierenden Mehrkosten ergeben sich aus der Differenz $t_{cpu,addr,h,dir_{19}} - t_{cpu,addr,h,dir_{18}}$. Dabei enstpricht nach Tabelle A.3 der Index 19 demjenigen Testmuster, welches bei der Durchführung der Messungen verwendet wurde. Der Index 18 ergibt sich wieder entsprechend dem Null-Codewort.

$$\begin{aligned} t_{cpu,addr,h,dir_i} &= m_{cpu,addr,h,dir_i} && (4.1) \\ &\quad -\frac{1}{2}(m_{cpu,addr,w,dir_i} - m_{cpu,addr,w,dir_{18}}) && (4.2) \end{aligned}$$

$$
\begin{aligned}
t_{cpu,data,h,dir_i} &= m_{cpu,data,h,dir_i} && (4.3)\\
&-\frac{1}{2}(m_{cpu,data,w,dir_i} - m_{cpu,data,w,dir_{17}}) && (4.4)\\
&-(t_{cpu,addr,h,dir_{19}} - t_{cpu,addr,h,dir_{18}}) && (4.5)
\end{aligned}
$$

Speicherkosten

Die Umrechnungen für die Reihen des Speichers sind entsprechend. Als Ausgangsreihen werden aber die S-Reihen verwendet:

$$t_{mem,addr,w,dir_i} = s_{mem,addr,w,dir_i}$$

$$t_{mem,data,w,dir_i} = s_{mem,data,w,dir_i}$$

$$
\begin{aligned}
t_{mem,addr,h,dir_i} &= s_{mem,addr,h,dir_i}\\
&-\frac{1}{2}(s_{cpu,addr,w,dir_i} - s_{cpu,addr,w,dir_{18}})
\end{aligned}
$$

$$
\begin{aligned}
t_{mem,data,h,dir_i} &= s_{cpu,data,h,dir_i}\\
&-\frac{1}{2}(s_{mem,data,w,dir_i} - s_{mem,data,w,dir_{17}})\\
&-(t_{mem,addr,h,dir_{19}} - t_{mem,addr,h,dir_{18}})
\end{aligned}
$$

Die berechneten Reihen T sind im Anhang A.3 aufgelistet, Grafiken folgen im Zusammenhang mit der linearen Regression in Kapitel 4.4.

4.3 Regressionsanalyse

Bisher wurden zur Datenerhebung Messungen durchgeführt. Die gewonnen Messwerte wurden weiteren Rechnungen unterzogen, so dass die einzelnen Reihen die Einzelkosten der entsprechenden Reihen sowie Base-Kosten beinhalten. Diese Werte können nicht als Parameter für das Energiemodell genutzt werden. Das Modell benutzt die zu bestimmenden Parameter $Base$, α_i und β_i in Termen der Form $Base + \alpha_i \cdot w(x)$ bzw. $Base + \beta_i \cdot h(x_1, x_2)$. Diese Terme sind nichts anderes als Geradengleichungen mit der y-Verschiebung $Base$ und der Steigung α_i bzw. β_i. Zur Ermittlung der Modellparameter sind die Datenwerte der Reihen durch Geraden zu approximieren. Die lineare Regressionsmethode führt diesen Schritt durch. Dabei werden die Geraden so bestimmt, dass sie eine gewisse Güte der Approximation an die Reihen sicherstellen.

Allgemeine Regressionsanalyse

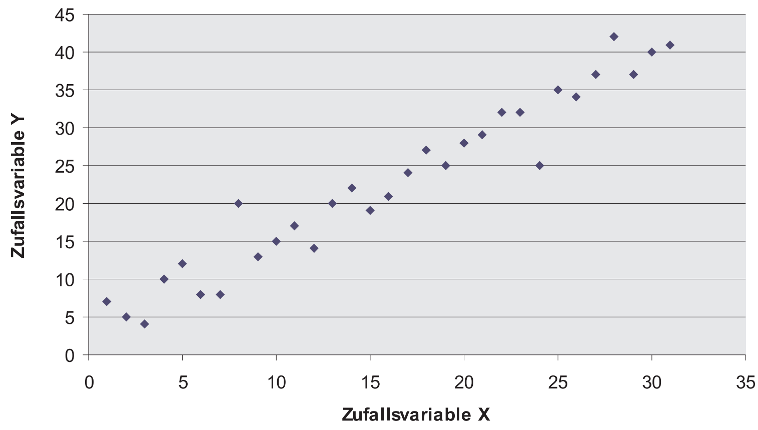


Abbildung 4.12: Beipiel einer Messreihe

Gegeben sei eine diskrete zweidimensionale Stichprobe mit einer Messreihe $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ zu einer zweidimensionalen Zufallsvariable (X, Y). Die einzelnen Stichprobenwerte lassen sich als Menge von i Punkten (x_i, y_i) in einer Ebene darstellen. Abbildung 4.12 zeigt eine solche Punktwolke einer Stichprobe. Ziel der Regressionsanalyse ist es, einen funktionalen Zusammenhang zwischen den x-Werten und y-Werten zu beschreiben. Man möchte die Stichprobe durch eine gut zu behandelnde Funktion f approximieren, etwa ein Polynom. Zur Beurteilung, wie gut die Funktion die gegebene Messreihe approximiert, wird der *quadratische Fehler* verwendet. Der Ausdruck

$$\delta_i = y_i - f(x_i)$$

gibt für jedes i den vertikalen Abstand vom Punkt (x_i, y_i) zur Funktion f an (Abbildung 4.13). Die Summe über alle quadrierten δ_i ergibt den quadratischen Fehler

$$\Delta^2 = \sum_{i=1}^{n} \delta_i^2.$$

Je kleiner der Wert ist, desto besser ist die Güte der Annäherung von f an die Stichprobe.

Lineare Regression

In dieser Diplomarbeit werden alle in Frage kommenden Messreihen durch eine Gerade, die *Regressionsgerade*, angenähert. Dieser Spezialfall wird als *Lineare Regression* bezeichnet. Gesucht ist jeweils ein Polynom ersten Grades, d.h. eine Funktion der Form $y = a + bx$, die den quadratischen Fehler minimiert. Zur

Herleitung, wie die *Regressionskoeffizienten* a und b bestimmt werden, sei beispielsweise auf die Quelle [GRTI96] verwiesen. Die Parameter lassen sich mit Hilfe der statistischen Kenngrößen *Arithmetisches Mittel*, *Stichprobenvarianz* s_x^2 und *Stichprobenkovarianz* s_{xy} berechnen:

$$b = \frac{s_{xy}}{s_x^2}$$

$$a = \bar{y} - \bar{x} \cdot \frac{s_{xy}}{s_x^2}$$

Abbildung 4.13 zeigt, wie für das vorherige Beispiel die Gerade der linearen Regression aussieht.

Anmerkung: Die vorgestellte lineare Regression wurde für den allgemeinen Fall beschrieben, dass eine zweidimensionale Zufallsvariable gegeben ist. In dieser Arbeit ist nur die y-Komponente eine stochastische Größe.

Berechnung verwendeter statistischer Kenngrößen

- Arithmetisches Mittel:

$$\bar{x} := \frac{1}{n} \cdot (x_1 + \ldots + x_n) = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

- Stichprobenvarianz:

$$s_x^2 := \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2$$

- Stichprobenkovarianz:

$$s_{xy} := \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

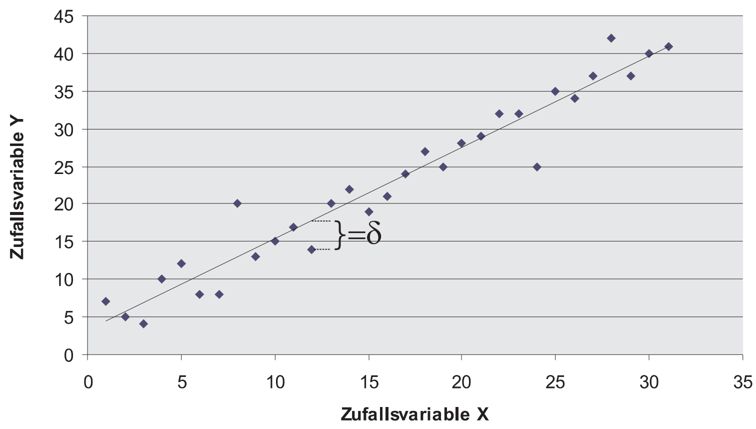


Abbildung 4.13: Messreihe mit Regressionsgerade f

4.4 Bestimmung der Parameter für das Energiemodell

Die lineare Regressionsanalyse wird auf die Werte der Reihen T angewendet, die die Anzahl gesetzter Busleitungen erfassen (Abschnitt 4.1.3). Es ergeben sich Werte gemäß Tabelle 4.7. Die Abbildungen 4.14 bis 4.17 zeigen Grafiken der Messreihen T sowie deren Regressionsgeraden. Da für spätere Compiler-Optimierungen die Summen von CPU- und Speicherkosten wichtig sind, wurden diese ebenfalls in den Grafiken dargestellt.

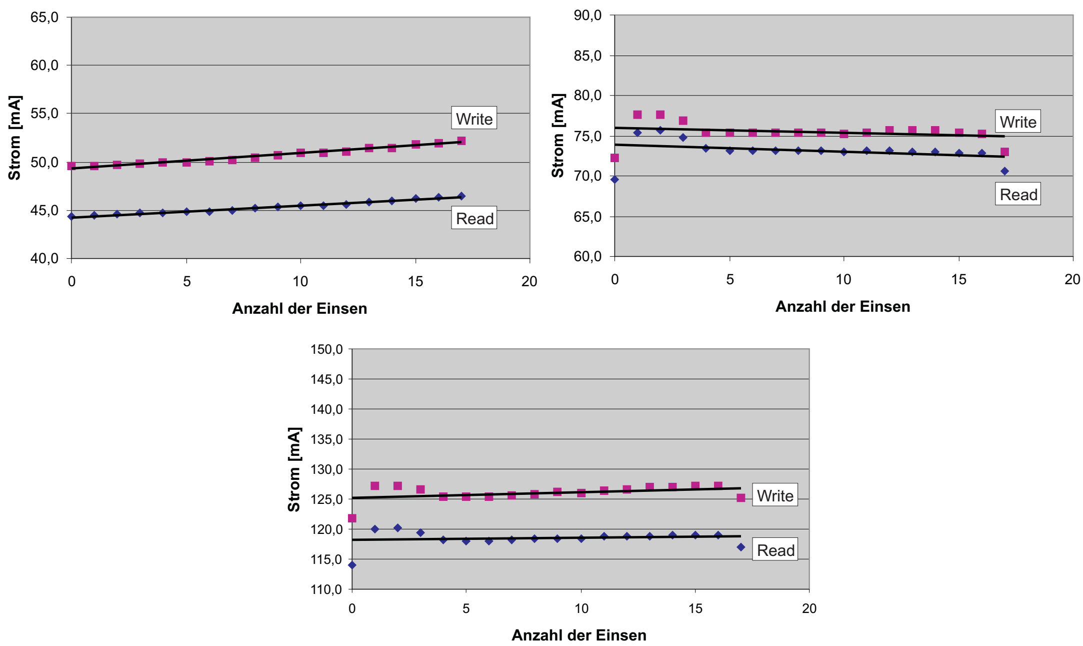


Abbildung 4.14: Reihen T: CPU: Anzahl '1'en CPU (*links*), Speicher (*rechts*) und Summe (*unten*) auf dem Adressbus

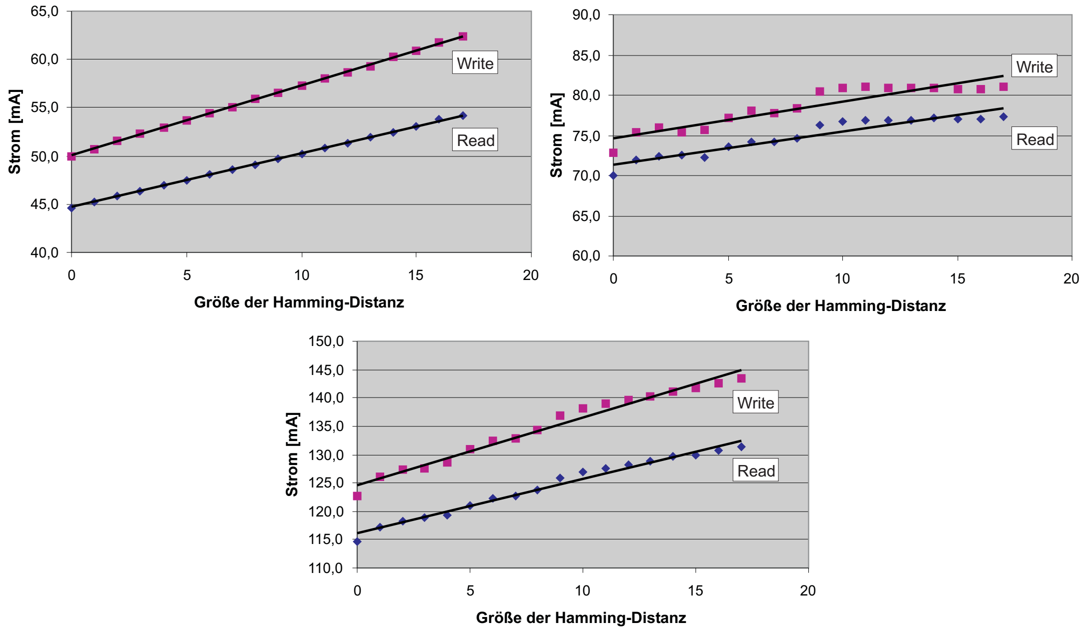


Abbildung 4.15: Reihen T: CPU: Größe Hamming-Distanz CPU (*links*), Speicher (*rechts*) und Summe (*unten*) auf dem Adressbus

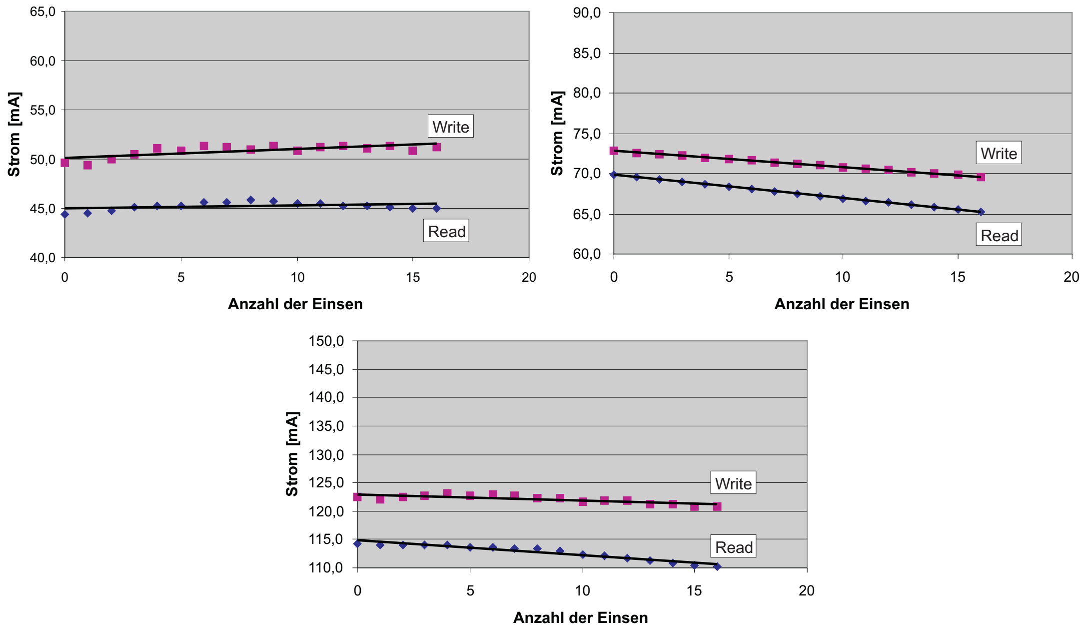


Abbildung 4.16: Reihen T: CPU: Anzahl '1'en CPU (*links*), Speicher (*rechts*) und Summe (*unten*) auf dem Datenbus

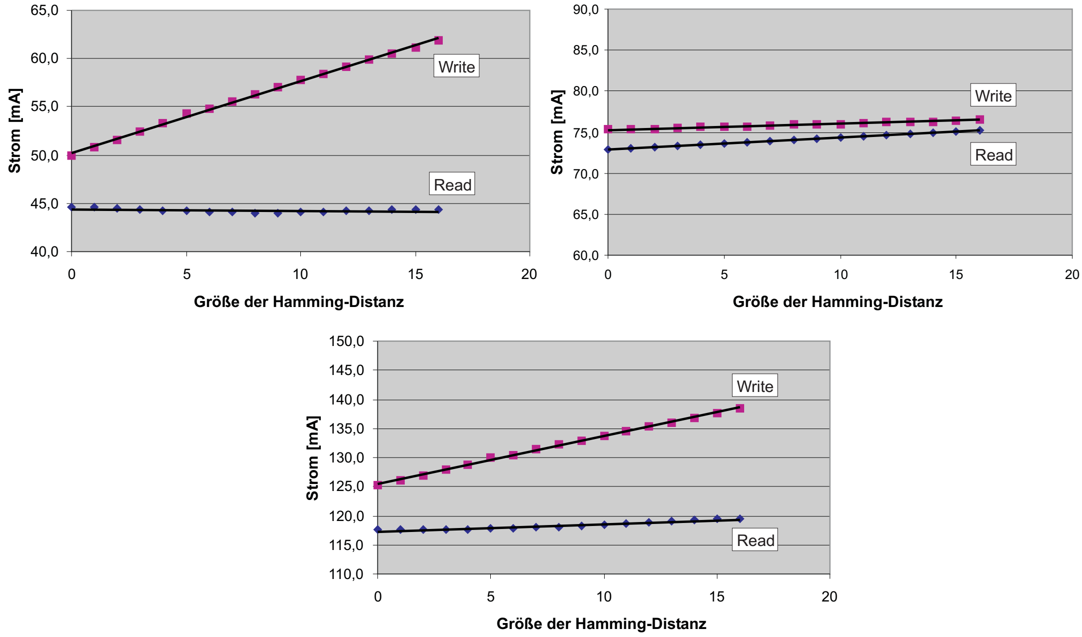


Abbildung 4.17: Reihen T: CPU: Größe Hamming-Distanz CPU (*links*), Speicher (*rechts*) und Summe (*unten*) auf dem Datenbus

Reihe	Read		Reihe	Write	
	a (mA)	b (mA)		a (mA)	b (mA)
$T_{cpu,addr,w,r}$	44,271	0,122	$T_{cpu,addr,w,w}$	49,306	0,159
$T_{cpu,addr,h,r}$	44,689	0,558	$T_{cpu,addr,h,w}$	50,025	0,728
$T_{cpu,data,w,r}$	45,000	0,027	$T_{cpu,data,w,w}$	50,143	0,087
$T_{cpu,data,h,r}$	44,350	-0,014	$T_{cpu,data,h,w}$	50,234	0,742
$T_{mem,addr,w,r}$	73,856	-0,085	$T_{mem,addr,w,w}$	75,963	-0,064
$T_{mem,addr,h,r}$	71,395	0,408	$T_{mem,addr,h,w}$	74,636	0,460
$T_{mem,data,w,r}$	69,796	-0,286	$T_{mem,data,w,w}$	72,800	-0,200
$T_{mem,data,h,r}$	72,902	0,143	$T_{mem,data,h,w}$	75,261	0,075

Tabelle 4.7: Regressionskoeffizienten a und b von $y = a + bx$ für alle Reihen T

Parameter für Ones- und Hamming-Distanz-Kosten

Die ermittelten Steigungswerte b enstprechen den Koeffizienten α_i und β_i. Das Energiemodell unterscheidet zwischen getrenntem Instruktions- und Datenspeicher. Die verwendete Versuchsumgebung besitzt aber einen gemeinsamen Speicher. Aus diesem Grund sind alle α_i und β_i, die sich auf den Instruktionsspeicher beziehen, mit den entsprechenden Parametern gleichzusetzen, die für den Datenspeicher gelten. Im einzelnen sind dies: $\alpha_4 = \alpha_5$, $\alpha_7 = \alpha_9$, $\alpha_8 = \alpha_{10,read}$, $\beta_4 = \beta_5$, $\beta_7 = \beta_9$ und $\beta_8 = \beta_{10,read}$.

Tabelle 4.8 zeigt die Parameter. Das Energiemodell benötigt die Parameter in Form von Energiewerten. Bisher liegen die Werte aber nur als Stromwerte vor. Nach der Formel für die Energie ($E = U \cdot I \cdot t$) verhalten sich Strom- und Energiewerte proportional zueinander. Zur Andeutung, dass die Werte noch in Energiewerte umzurechnen sind, werden die Stromwerte für Read-Zugriffe mit der Konstanten c_r multipliziert und die Werte für Write-Zugriffe mit der Konstanten c_w. Es wurden zwei verschiedene Konstanten gewählt, da zur Errechnung der Energiewerte für Read- und Write-Zugriffe unterschiedliche Zeiten zu berücksichtigen sind.

Parameter	Energie (J)		Parameter	Energie (J)	
	Read	Write		Read	Write
α_4, α_5	—	0,159 $\cdot c_w$	β_4, β_5	—	0,728 $\cdot c_w$
$\alpha_{6,dir}$	0,027 $\cdot c_r$	0,087 $\cdot c_w$	$\beta_{6,dir}$	-0,014 $\cdot c_r$	0,742 $\cdot c_w$
α_7, α_9	—	-0,064 $\cdot c_w$	β_7, β_9	—	0,460 $\cdot c_w$
α_8	-0,286 $\cdot c_r$	—	β_8	0,143 $\cdot c_r$	—
$\alpha_{10,dir}$	-0,286 $\cdot c_r$	-0,200 $\cdot c_w$	$\beta_{10,dir}$	0,143 $\cdot c_r$	0,075 $\cdot c_w$

Tabelle 4.8: Parameter für das ARM7TDMI Energiemodell

Parameter für Base-Kosten

Bei den ermittelten Koeffizienten der linearen Regression entsprechen die Werte a den Base-Kosten. Da mehrere Messreihen durchgeführt wurden, gibt es für die gleichen Base-Kosten mehrere a-Werte. Diese Werte müßten jeweils untereinander gleich sein, wegen Toleranzen der Messungen ergeben sich aber kleine Unterschiede. Deshalb werden bei der Ermittlung der Base-Kosten arithmetische Mittelwerte gebildet. Zur Andeutung, dass für das Energiemodell die Stromwerte noch in Energiewerte umzurechnen sind, werden die Werte wie zuvor mit einer Konstanten multipliziert.

$$BaseCPU(LDRH) = \frac{44,271 + 44,689 + 45,000 + 44,350}{4} \cdot c_r = 44,578 \cdot c_r \text{ [J]}$$

$$BaseCPU(STRH) = \frac{49,306 + 50,025 + 50,143 + 50,234}{4} \cdot c_w = 49,927 \cdot c_w \text{ [J]}$$

$$BaseMem(OffChip, read, 16) = \frac{73,856 + 71,395 + 69,796 + 72,902}{4} \cdot c_r = 71,987 \cdot c_r \text{ [J]}$$

$$BaseMem(OffChip, write, 16) = \frac{75,963 + 74,636 + 72,800 + 75,261}{4} \cdot c_w = 74,665 \cdot c_w \text{ [J]}$$

Strom vs. Energie

Alle Messungen und Berechnungen wurden mit Strom-Werten in der Einheit mA durchgeführt. Um die bisherigen Ergebnisse für das Energiemodell in Energiewerte umzurechnen, ist die Zeitdauer einer einzelnen Sequenz bzw. Instruktion im Messmodul zu berücksichtigen, bei der der jeweils gemessene Durchschnittsstrom geflossen ist. Inklusive Instruction-Fetch vom On-Chip-Speicher und Datenzugriff im Off-Chip-Speicher benötigen die verwendeten `LDRH`- und `STRH`-Instruktionen folgende Zyklen:

$$\text{Dauer LDRH: } 4\ Zyklen$$

$$\text{Dauer STRH: } 3\ Zyklen$$

Ein Zyklus entspricht bei 32,768 MHz Taktfrequenz des ARM7TDMI

$$t_C = \frac{1}{32,768 \cdot 10^6} s.$$

Die Versorgungsspannung des Prozessors beträgt 3,3V. Allgemein lassen sich die Energiewerte für die durchgeführten Strommessungen nach der Formel $E = U \cdot I \cdot t$ wie folgt berechnen:

$$E_{read} = 3,3V \cdot I_{read} \cdot 4 \cdot \frac{1}{32,768 \cdot 10^6} s$$

$$E_{write} = 3,3V \cdot I_{write} \cdot 3 \cdot \frac{1}{32,768 \cdot 10^6} s$$

Die beiden Tabellen 4.9 und 4.10 zeigen die Ergebnisse als Energiewerte.

Parameter	Energie ($10^{-12}J$)		Parameter	Energie ($10^{-12}J$)	
	Read	Write		Read	Write
α_4, α_5	—	48,0	β_4, β_5	—	219,9
$\alpha_{6,dir}$	11,0	26,4	$\beta_{6,dir}$	-5,5	224,1
α_7, α_9	—	-19,2	β_7, β_9	—	138,9
α_8	-115,3	—	β_8	57,7	—
$\alpha_{10,dir}$	-115,3	-60,4	$\beta_{10,dir}$	57,7	22,8

Tabelle 4.9: Parameter für das ARM7TDMI Energiemodell als Energiewerte

Base-Kosten	Energie ($10^{-9}J$)
BaseCPU(LDRH)	17,96
BaseCPU(STRH)	20,11
BaseMem(OffChip,read,16)	29,00
BaseMem(OffChip,write,16)	30,08

Tabelle 4.10: Base-Kosten für das ARM7TDMI Energiemodell als Energien

4.4.1 Speicherzugriffe unterschiedlicher Wortgrößen

Die ermittelten Parameter α_4 bis α_{10} und β_4 bis β_{10} gelten für alle Zugriffe, die auf den Off-Chip-Speicher durchgeführt werden. Es spielt keine Rolle, ob die Datenzugriffe z.B. von Load-/Store-Instruktionen stammen oder von Stack-Operationen. Die Datenerhebungen wurden für 16 Bit Datenzugriffe durchgeführt. Datenzugriffe, die die Busbreite von 16 Bit überschreiten, werden in mehrere aufeinander folgende Zugriffe aufgeteilt. Kosten für kleinere Wortbreiten werden nur für die entsprechende Breite des Datenzugriffs berechnet.

4.4.2 Immediate-Werte und Register

Die Parameter α_1, α_2, β_1 und β_2 werden mit Hilfe von MOVE-Operationen bestimmt. Die Vorgehensweise ist entsprechend derjenigen, die für die externen Buskosten angewendet wurde. α_3 und β_3, die Kosten in Abhängigkeit von Registerinhalten erfassen, werden mit ALU-Befehlen bestimmt.

4.4.3 Base-Kosten

Die Base-Kosten aller Instruktionen werden bestimmt, indem in Messmodulen hintereinander jeweils der gleiche Befehl ausgeführt wird. Dabei sollten eventuelle Operanden so gewählt werden, dass zwecks einheitlicher Base-Kosten alle Operanden nur Nullen enthalten. Zur Energieberechnung wird der gemessene durchschnittliche Strom mit der Versorgungsspannung und der Ausführungszeit einer Instruktion multipliziert.

4.4.4 FUChange-Kosten

Zur Bestimmung von FUChange-Kosten werden jeder Instruktion die Funktionseinheiten zugeordnet, die sie anspricht. Anschließend werden die Instruktionen zu Äquivalenzklassen gruppiert, so dass Befehle, die die gleichen Funktionseinheiten ansprechen, einer gemeinsamen Äquivalenzklasse angehören. Für das Energiemodell wird angenommen, dass für alle Instruktionen, die in einer gleichen Äquivalenzklasse liegen, die gleichen FUChange-Kosten anfallen. Um Messungen zur Gewinnung der Modellparameter durchführen zu können, werden

Aktivierungs- und Deaktivierungskosten nicht explizit einzeln erfasst, sondern beide Kosten werden als identisch betrachtet.

Zur Bestimmung aller FUChange-Kosten werden für jede Äquivalenzklasse die FUChange-Kosten zwischen einer Instruktion aus der jeweiligen Klasse und einer Instruktion $Instr_{BZ}$, die für alle FUChange-Kosten als Bezugspunkt dient, bestimmt. Als Bezugspunkt wird eine Instruktion gewählt, die nach Möglichkeit eine Funktionseinheit verwendet, die von allen anderen Befehlen auch benutzt wird. Für den ARM7TDMI wird die Instruktion $Instr_{BZ}$ =`MOV r0,#0x0` gewählt. Diese verwendet eine Funktionseinheit, die für Register-Transfers zuständig ist. Bis auf wenige Ausnahmen wie unbedingte Sprünge führen alle Instruktionen Register-Transfers durch. Zum Einen werden Register bei der Verwendung als Quelloperanden ausgelesen, zum Anderen schreiben Instruktionen Zielwerte in Register.

Für eine einzelne Äquivalenzklasse werden die FUChange-Kosten wie folgt bestimmt: Aus der Äquivalenzklasse wird willkürlich eine Instruktion gewählt. Die Operanden der Instruktion aus der Äquivalenzklasse sollten so gewählt werden, dass zwecks einheitlicher Base-Kosten alle Einzelkosten minimal sind. Für die Instruktion der Äquivalenzklasse und der Instruktion $Instr_{BZ}$ findet eine Strommessung statt. Zur Duchführung der Messung werden die beiden Instruktionen in der Endlosschleife eines Messmoduls wiederholt ausgeführt. Der durchschnittlich fließende Strom beider Instruktionen wird an der CPU gemessen. Die Wechselkosten ergeben sich aus der Differenz des gemessenen Stroms und dem theoretisch berechneten Strom, der bei der Ausführung beider Instruktionen fließen müsste. Der berechnete durchschnittlich fließende Strom lässt sich aus den Strömen der Base-Kosten $BaseCPU_I$ und eventuell anderen anfallenden Einzelkosten I_{other} berechnen:

$$I_{calc} = \frac{(BaseCPU_I(Opcode_1) \cdot n_1) + (BaseCPU_I(Opcode_2) \cdot n_2)}{n_1 + n_2} + I_{other}$$

n_1 und n_2 sind die Taktzyklen der Befehle zu $Opcode_1$ und $Opcode_2$. Der zusätzliche Strom $I_{FUChange}$, der durch die FUChange-Kosten anfällt, errechnet sich als:

$$I_{FUChange} = I_{1,2} - I_{calc}$$

$I_{1,2}$ ist der im Messmodul gemessene Strom der Sequenz beider Instruktionen.

Zur Ermittlung von Energiewerten für die Formel *FUChange* nach Unterkapitel 3.2.2 wird der Strom $I_{FUChange}$ zu gleichen Teilen in Aktivierungs- und Deaktivierungskosten umgerechnet. Die Gesamtenergie für beide Kosten ergibt sich als Produkt von Versorgungsspannung des Prozessors, dem Strom $I_{FUChange}$, der Summe der Taktzyklen beider Instruktionen und der Dauer eines Taktzyklus. Die Aktivierungs- und Deaktivierungskosten für Funktionseinheit F_i, die von der betrachteten Äquivalenzklasse benutzt wird, ergeben sich als:

$$\gamma(F_i) = \delta(F_i) = \frac{1}{2}(3,3V \cdot I_{FUChange} \cdot (n_1 + n_2) \cdot \frac{1}{32,768 \cdot 10^6} s)$$

4.5 Auswertungen

Auswertung der Wertigkeiten einzelner Busleitungen

Betrachtet werden die Messreihen, die für die Erfassung der Wertigkeiten einzelner Busleitungen aufgestellt wurden. Die Reihen sind auf Seite 42ff in den Abbildungen 4.4 (links) bis 4.11 links abgebildet. Zur Gewinnung der Messreihen wurden jeweils Testmuster verwendet, die auf dem entsprechenden Bus gezielt eine einzige Busleitung auf '1' setzen. Der erste Messwert setzt die erste Busleitung auf Eins, der zweite Messwert die zweite Busleitung usw. Es ist zu untersuchen, ob die Busleitungen pro Bus homogen aufgebaut sind und somit als gleichwertig gelten. Im Idealfall sollten alle Messwerte einer Messreihe identisch sein.

Die Abbildungen 4.4 und 4.5 (jeweils linke Grafiken) erfassen die Wertigkeiten von Einsen auf dem Adress- und Datenbus, die an der CPU gemessen wurden. Die einzelnen Werte pro Reihe zeigen nur kleine Schwankungen auf. Somit gelten alle Leitungen als gleichwertig.

Abbildung 4.6 beinhaltet die Wertigkeiten von Einsen auf dem Adressbus, die am Speicher erfasst wurden. Für die ersten 4 Messwerte sowie für den letzten Messwert zeigen sich Abweichungen von den übrigen Werten. Diese lassen sich nicht direkt begründen, da keine Informationen über den internen Aufbau des Speicherbausteins vorliegen, an dem die Messungen vorgenommen wurden.

Die am Speicher gemessenen Wertigkeiten von Einsen auf dem Datenbus zeigt Abbildung 4.7. Die ersten 8 Messwerte haben kleinere Ströme, als die letzten 8 Werte. Dies ist damit begründet, dass der Baustein, an dem gemessen wurde, nur die ersten 8 Bits des 16 Bit breiten Datenworts aufnimmt.

Die Abbildungen 4.8 bis 4.11 zeigen, im Gegensatz zu den 4 vorherigen Abbildungen, die Wertigkeiten von *Hamming-Distanzen* auf den einzelnen Busleitungen. Auffällig ist die Reihe, die Messwerte für die Hamming-Distanz auf dem Adressbus erfasst (Abbildung 4.10). Es gibt Ausreißer für die ersten 8 und den letzten Messwert. Diese Anomalien lassen sich ebensowenig begründen, wie die Unregelmäßigkeiten in Abbildung 4.6.

Abschließend lässt sich sagen, dass kleine Auffälligkeiten nur bei Messungen am Speicher für den Adressbus auftreten. Diese scheinen im internen Aufbau des Speichers begründet zu sein, über den aber keine weiteren Informationen verfügbar sind.

Maximale Anteile von Ones- und Hamming-Distanz-Kosten

In diesem Abschnitt wird analysiert, wie groß der Einfluss der einzelnen Buskosten auf den Gesamtenergieverbrauch von Instruktionen sein kann. Dazu wird berechnet, um welche maximalen prozentualen Werte Ones- und Hamming-Distanz-Kosten die Energiewerte von Load- und Store-Instruktionen auf dem Daten- und Adressbus erhöhen können. Zur Berechnung der prozentualen Änderungen werden jeweils zwei Energiewerte errechnet, ein minimaler und ein maximaler Wert.

Der prozentuale Wert gibt an, um welchen Anteil der größere Wert über dem kleineren liegt, d.h. um wiewiel Prozent die betrachten Einzelkosten auf dem jeweiligen Bus die Energie maximal erhöhen können. Durchgeführt wird dieses Verfahren für den Datenbus jeweils für Ones- und Hamming-Distanz-Kosten für lesenden und schreibenden Zugriff. Auf den Adressbus kann nur schreibend zugegriffen werden, somit werden Ones- und Hamming-Distanz-Kosten nur für lesenden Zugriff betrachtet.

Berechnung der Anteile von Ones-Kosten auf dem Datenbus

Es folgt die Ermittlung der prozentualen Änderung für Ones-Kosten auf dem Datenbus bei lesendem Zugriff. Zur Berechnung des ersten Energiewertes werden nur Nullen auf den Datenbus gelegt:

$$\begin{aligned}
&E1_{ones,data,read} = \\
&\quad BaseCPU(LDRH) + \alpha_{6,read} \cdot w('0000000000000000') + \\
&\quad BaseMem(OffChip, read, 16) + \alpha_{10,read} \cdot w('0000000000000000') \\
= &\quad 17,96 \cdot 10^{-9} J + 11,0 \cdot 10^{-12} J \cdot 0 + \\
&\quad 29,00 \cdot 10^{-9} J + (-115,3) \cdot 10^{-12} J \cdot 0 \\
= &\quad 46,96 \cdot 10^{-9} J
\end{aligned}$$

Für die Errechnung des zweiten Energiewertes werden nur Einsen auf den Bus gelegt:

$$\begin{aligned}
&E2_{ones,data,read} = \\
&\quad BaseCPU(LDRH) + \alpha_{6,read} \cdot w('1111111111111111') + \\
&\quad BaseMem(OffChip, read, 16) + \alpha_{10,read} \cdot w('1111111111111111') \\
= &\quad 17,96 \cdot 10^{-9} J + 11,0 \cdot 10^{-12} J \cdot 16 + \\
&\quad 29,00 \cdot 10^{-9} J + (-115,3) \cdot 10^{-12} J \cdot 16 \\
= &\quad 45,29 \cdot 10^{-9} J
\end{aligned}$$

Es erfolgt die Berechnung der prozentualen Änderung. Der kleinere der beiden berechneten Energiewerte wird als Bezugspunkt genommen. Zur Vereinfachung werden die Einheiten sowie die Zehnerpotenzen nicht hingeschrieben, beides kürzt sich im Bruch weg.

$$p_{ones,data,read} = \frac{46,96}{45,29} - 1 = 0,037$$

Ones-Kosten können die Energiewerte auf dem Datenbus bei lesendem Zugriff um maximal 3,7% erhöhen.

Die Berechnung der anderen Werte geschieht auf die gleiche Art.

$$\begin{aligned}
E1_{ones,data,write} =& \\
& BaseCPU(STRH) + \alpha_{6,write} \cdot w('0000000000000000') + \\
& BaseMem(OffChip, write, 16) + \alpha_{10,write} \cdot w('0000000000000000') \\
=& \ 20,11 \cdot 10^{-9} J + 26,4 \cdot 10^{-12} J \cdot 0 + \\
& 29,00 \cdot 10^{-9} J + (-60,4) \cdot 10^{-12} J \cdot 0 \\
=& \ 49,11 \cdot 10^{-9} J
\end{aligned}$$

$$\begin{aligned}
E2_{ones,data,write} =& \\
& BaseCPU(STRH) + \alpha_{6,write} \cdot w('1111111111111111') + \\
& BaseMem(OffChip, write, 16) + \alpha_{10,write} \cdot w('1111111111111111') \\
=& \ 20,11 \cdot 10^{-9} J + 26,4 \cdot 10^{-12} J \cdot 16 + \\
& 29,00 \cdot 10^{-9} J + (-60,4) \cdot 10^{-12} J \cdot 16 \\
=& \ 48,57 \cdot 10^{-9} J
\end{aligned}$$

$$p_{ones,data,write} = \frac{49,11}{48,57} - 1 = 0,011$$

Berechnung der Anteile von Hamming-Kosten auf dem Datenbus

$$\begin{aligned}
E1_{hamming,data,read} =& \\
& BaseCPU(LDRH) + \beta_{6,read} \cdot h('0000000000000000','000\ldots') + \\
& BaseMem(OffChip, read, 16) + \beta_{10,read} \cdot h('0000000000000000','000\ldots') \\
=& \ 17,96 \cdot 10^{-9} J + (-5,5) \cdot 10^{-12} J \cdot 0 + \\
& 29,00 \cdot 10^{-9} J + 57,7 \cdot 10^{-12} J \cdot 0 \\
=& \ 46,96 \cdot 10^{-9} J
\end{aligned}$$

$$\begin{aligned}
E2_{hamming,data,read} =& \\
& BaseCPU(LDRH) + \beta_{6,read} \cdot h('0000000000000000','111\ldots') + \\
& BaseMem(OffChip, read, 16) + \beta_{10,read} \cdot h('0000000000000000','111\ldots') \\
=& \ 17,96 \cdot 10^{-9} J + (-5,5) \cdot 10^{-12} J \cdot 16 + \\
& 29,00 \cdot 10^{-9} J + 57,7 \cdot 10^{-12} J \cdot 16 \\
=& \ 47,80 \cdot 10^{-9} J
\end{aligned}$$

$$p_{hamming,data,read} = \frac{47,80}{46,96} - 1 = 0,018$$

$$
\begin{aligned}
& E1_{hamming,data,write} = \\
& \quad BaseCPU(STRH) + \beta_{6,write} \cdot h('0000000000000000','000\ldots') + \\
& \quad BaseMem(OffChip, write, 16) + \beta_{10,write} \cdot h('0000000000000000','000\ldots') \\
& = 20,11 \cdot 10^{-9} J + 224,1 \cdot 10^{-12} J \cdot 0 + \\
& \quad 29,00 \cdot 10^{-9} J + 22,8 \cdot 10^{-12} J \cdot 0 \\
& = 49,11 \cdot 10^{-9} J
\end{aligned}
$$

$$
\begin{aligned}
& E2_{hamming,data,write} = \\
& \quad BaseCPU(STRH) + \beta_{6,write} \cdot h('0000000000000000','111\ldots') + \\
& \quad BaseMem(OffChip, write, 16) + \beta_{10,write} \cdot h('0000000000000000','111\ldots') \\
& = 20,11 \cdot 10^{-9} J + 224,1 \cdot 10^{-12} J \cdot 16 + \\
& \quad 29,00 \cdot 10^{-9} J + 22,8 \cdot 10^{-12} J \cdot 16 \\
& = 53,06 \cdot 10^{-9} J
\end{aligned}
$$

$$p_{hamming,data,write} = \frac{53,06}{49,11} - 1 = 0,080$$

Berechnung der Anteile von Ones-Kosten auf dem Adressbus

$$
\begin{aligned}
& E1_{ones,addr,write} = \\
& \quad BaseCPU(STRH) + \alpha_5 \cdot w('00000000000000000') + \\
& \quad BaseMem(OffChip, write, 16) + \alpha_9 \cdot w('00000000000000000') \\
& = 20,11 \cdot 10^{-9} J + 48,0 \cdot 10^{-12} J \cdot 0 + \\
& \quad 30,08 \cdot 10^{-9} J + (-19,2) \cdot 10^{-12} J \cdot 0 \\
& = 50,19 \cdot 10^{-9} J
\end{aligned}
$$

$$
\begin{aligned}
& E2_{ones,addr,write} = \\
& \quad BaseCPU(STRH) + \alpha_5 \cdot w('11111111111111111') + \\
& \quad BaseMem(OffChip, write, 16) + \alpha_9 \cdot w('11111111111111111') \\
& = 20,11 \cdot 10^{-9} J + 48,0 \cdot 10^{-12} J \cdot 17 + \\
& \quad 30,08 \cdot 10^{-9} J + (-19,2) \cdot 10^{-12} J \cdot 17 \\
& = 50,68 \cdot 10^{-9} J
\end{aligned}
$$

$$p_{ones,addr,write} = \frac{50,68}{50,19} - 1 = 0,010$$

Berechnung der Anteile von Hamming-Kosten auf dem Adressbus

$$\begin{aligned}
E1_{hamming,addr,write} = & \\
& BaseCPU(STRH) + \beta_5 \cdot h('00000000000000000','000\ldots') + \\
& BaseMem(OffChip, write, 16) + \beta_9 \cdot h('00000000000000000','000\ldots') \\
= & \ 20,11 \cdot 10^{-9}J + 219,9 \cdot 10^{-12}J \cdot 0 + \\
& 30,08 \cdot 10^{-9}J + 138,9 \cdot 10^{-12}J \cdot 0 \\
= & \ 50,19 \cdot 10^{-9}J
\end{aligned}$$

$$\begin{aligned}
E2_{hamming,addr,write} = & \\
& BaseCPU(STRH) + \beta_5 \cdot h('00000000000000000','111\ldots') + \\
& BaseMem(OffChip, write, 16) + \beta_9 \cdot h('00000000000000000','111\ldots') \\
= & \ 20,11 \cdot 10^{-9}J + 219,9 \cdot 10^{-12}J \cdot 17 + \\
& 30,08 \cdot 10^{-9}J + 138,9 \cdot 10^{-12}J \cdot 17 \\
= & \ 56,29 \cdot 10^{-9}J
\end{aligned}$$

$$p_{hamming,addr,write} = \frac{56,29}{50,19} - 1 = 0,122$$

Abbildung 4.18 zeigt zusammenfassend, welche prozentualen Mehrkosten durch Ones- und Hamming-Distanz-Kosten maximal anfallen.

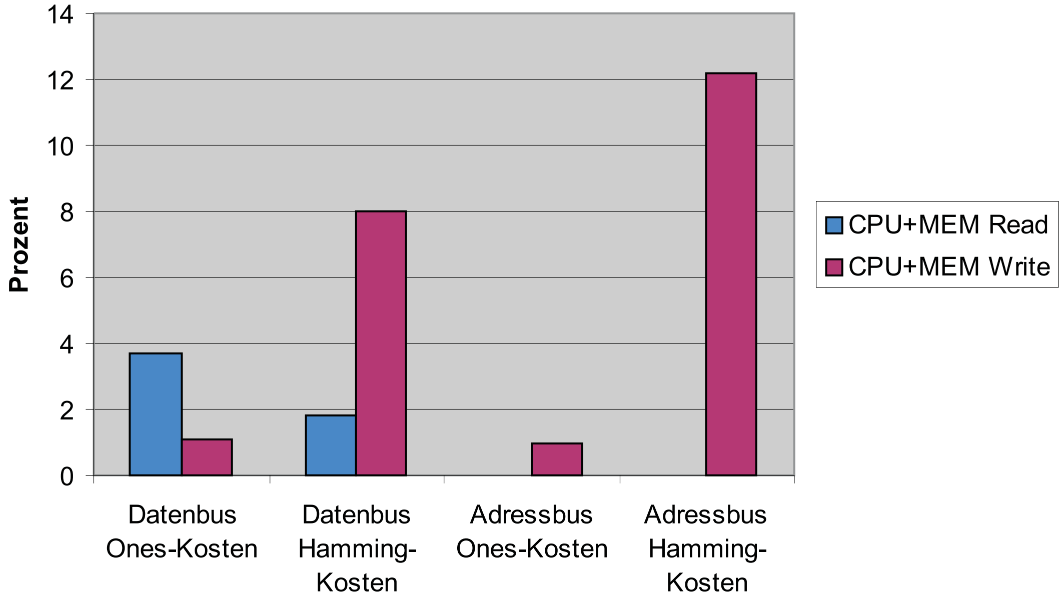


Abbildung 4.18: Vergleich der Buskosten

Es zeigt sich, dass Hamming-Distanzen auf dem Adressbus den Energieverbrauch um rund 12% erhöhen können. Zur Ausnutzung dieses Effekts für Energieoptimierungen ist es notwendig, die Hamming-Distanzen auf dem Adressbus zu verringern. Der Adressbus wird zum einen während des Instruction-Fetches genutzt, zum anderen zur Durchführung von Datenzugriffen. Die Ones-Kosten auf dem Datenbus fallen mit nur 1% sehr gering aus, es sind keine großen Energieeinsparungen zu erwarten.

Die Hamming-Dinstanz-Kosten führen für den Datenbus bei lesendem Datenzugriff im Extremfall zu knapp 2% mehr Energie, bei schreibendem Zugriff zu 8%. Dies kann ausgenutzt werden, um z.B. Energien bei Instruktionszugriffen zu verringern, indem durch Scheduling von Instruktionen die Hamming-Distanzen zwischen Instruktionsworten verkleinert werden. Die Ones-Kosten erhöhen den Energieverbrauch für lesende Datenzugriffe um knapp 4%, bei schreibenden Zugriffen um ca. 1%.

Bei der vorliegenden Versuchsumgebung zeigen sich z.B. für Hamming-Distanzen auf dem Datenbus bei lesendem Zugriff Energieerhöhungen von nur bis maximal 2%. Die erhobenen Messungen und die Ausmasse dieser Ergebnisse können nicht auf andere Systeme übertragen werden. Abhängig vom System können die auftretenden Effekte größer oder auch kleiner ausfallen.

Abschließend wird das Einsparpotential der Effekte zur Durchführung von Energiereduktionen in der Reihenfolge des Optimierpotentials diskutiert. Kosten auf dem Adressbus, verursacht durch Hamming-Distanzen, bieten das größte Einsparpotential für Optimierungen. Durch ein entsprechendes Speicherlayout können Energien auf dem Adressbus eingespart werden. Hamming-Distanz-Kosten haben auf dem Datenbus zwar bei schreibendem Zugriff mit 8% einen relativ großen Einfluss auf den Energieverbrauch, es zeigen sich aber keine relevanten Optimierungen auf. Dies ist damit begründet, dass Daten, die über den Datenbus geschrieben werden, während des Compiliervorgangs nicht bekannt sind. Informationen über die Daten müssten aber vorhanden sein, damit Optimierungen durchgeführt werden können. Das drittgrößte Einsparpotential bietet die Anzahl der Einsen auf dem Datenbus. Zur Ausnutzung dieses Effekts für Energieoptimierungen können z.B. Instruktionsworte durch eine entsprechende Registervergabe so verändert werden, dass häufig verwendete Instruktionsworte möglichst viele Einsen enthalten. Um so mehr Einsen auf dem Datenbus liegen, um so niedriger ist der Energieverbrauch. Mit knapp 2% ist das Einsparpotential bei Hamming-Distanzen auf dem Datenbus bei lesendem Zugriff zwar gering, sollte aber nicht unberücksichtigt bleiben. Durch Umsortieren der Instruktionsreihenfolge lassen sich Hamming-Distanzen auf dem Datenbus verringern. Für Ones-Kosten auf dem Datenbus ergeben sich für schreibenden Zugriff keine Energieeinsparmöglichkeiten, da keine Informationen über die Zusammensetzung der Daten gegeben sind. Ones-Kosten verursachen auf dem Adressbus mit rund 1% die geringsten Kosten. Dieser Effekt kann für Einsparungen durch Anwendung eines günstigen Speicherlayouts genutzt werden.

4.6 Durchgeführte Implementierungen

Am Lehrstuhl 12 des Fachbereichs Informatik ist eine Entwicklungsumgebung entstanden, die für Forschungsarbeiten im Bereich Low-Power eingesetzt wird. Abbildung 4.19 skizziert die wichtigsten Elemente dieser Umgebung.

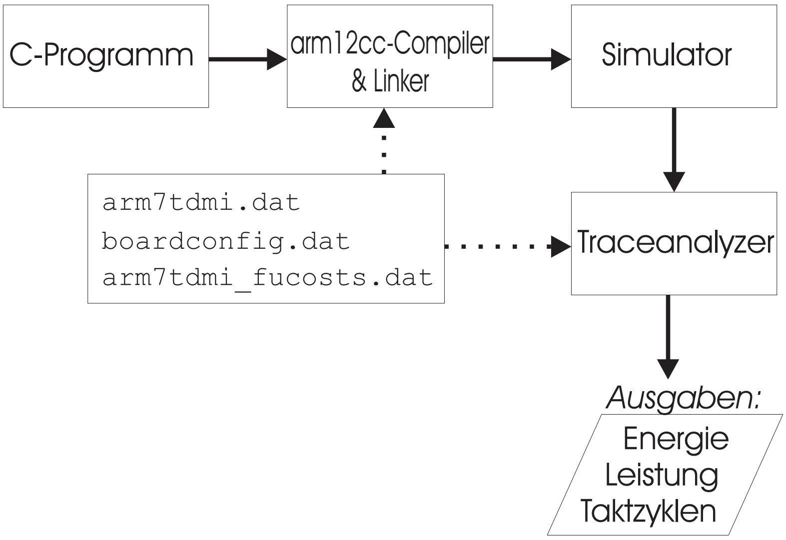


Abbildung 4.19: Software-Arbeitsumgebung

Ausgangspunkt für die Entwicklungsumgebung sind C-Programme, die im encc-Compiler in Assemblercode und anschließend in Maschinencode übersetzt werden. Der encc-Compiler wurde am Lehrstuhl 12 entwickelt. Neben standardmäßigen Optimierungen für Codegröße und Geschwindigkeit können verschiedene Energieoptimierungen durchgeführt werden, die im Laufe der Zeit am Lehrstuhl entwickelt wurden.

Der übersetzte Maschinencode wird mittels eines Linkers zu ausführbaren Programmen zusammengestellt und dient als Eingabe eines Simulators. Der Simulator simuliert die Ausführung des fertigen Codes und liefert als Ausgabe eine Trace-Datei. Diese Datei enthält Simulationsangaben des Programmablaufs. Die Angaben umfassen Informationen der ausgeführten Instruktionssequenzen sowie die durchgeführten Speicherzugriffe von Instruction-Fetches und Datenzugriffen.

Die Trace-Datei wird vom Traceanalyzer ausgewertet. Er berechnet Energie- und Leistungsverbrauch des simulierten Programms sowie dessen benötigte Taktzyklen. Die Software kann eingesetzt werden, um die berechneten Energien am realen Prozessor zu validieren sowie zur Bewertung von durchgeführten Compileroptimierungen.

Im Rahmen dieser Diplomarbeit wurde der Traceanalyzer dahingehend erweitert, dass er Energie- und Leistungswerte auch für Codierungskosten auf den Bussen sowie für FUChange-Kosten berechnet. In diesem Zusammenhang wurden vorhandene Konfigurationsdateien erweitert und neu hinzugefügt: Die Datei `arm7tdmi.dat`, welche u.a. den THUMB-Instruktionssatz des ARM7TDMI-Prozessors sowie die Leistungsdaten der einzelnen Instruktionen enthält, wurde um Angaben der von Instruktionen verwendeten Funktionseinheiten erweitert. Neu hinzugekommen ist die `arm7tdmi_fuchange.dat`-Datei. Diese enthält Angaben der FUChange-Kosten für die einzelnen Funktionseinheiten. Ferner gibt es die Datei `boardconfig.dat`. Diese beinhaltet, welche verschiedenen Speicher existieren und welche Base-Kosten für unterschiedliche Datenzugriffe anfallen. Erweitert wurde die Datei um Angabe der Parameter α_6 bis α_{10} und β_6 bis β_{10}. Die Konfigurationsdateien werden auch im encc-Compiler genutzt, um Optimierungen durchzuführen.

Kapitel 5

Optimierungen

5.1 Bisherige Arbeiten

Der überwiegende Teil bisheriger Forschungsarbeiten zu Energieoptimierungen konzentriert sich auf Codierungsverfahren, die mit Hilfe zusätzlicher Hardware durchgeführt werden. Ching-Long Su et al. [STD94a] stellen ein Verfahren namens *Cold Scheduling* vor, welches nur mit Softwareoptimierungen arbeitet. Der Name des Verfahren ist damit begründet, dass der Energieverbrauch des Prozessors und somit auch die Wärmeentwicklung verringert wird. Die Optimierung verringert die Bit-Wechsel auf dem Instruktions-Bus. Bei dem Verfahren handelt es sich um eine Heuristik, welche die Anzahl der Bit-Wechsel durch Scheduling der Instruktionen verringert. Optimiert werden die Hamming-Distanzen der Instruktionsworte untereinander.

Musoll et al. [MLC97] z.B. schlagen ein Optimierungsverfahren vor, welches Energieeinsparungen durch Hardwareänderungen erreicht. Musoll verringert durch sein Verfahren die Busaktivitäten auf dem Adressbus zum externen Datenspeicher und spart somit Energie ein. Das Verfahren geht von der Lokalität von Speicherreferenzen aus. Wenn z.B. eine Softwarefunktion mit drei Vektoren arbeitet, werden den Startadressen der Vektoren drei Arbeitsbereiche (engl.: *working zones*) zugewiesen. Die Startadressen dieser drei Arbeitsbereiche werden in der Nähe des Speichers, aber außerhalb von diesem, mittels zusätzlicher Hardware gepeichert. Die Energieeinsparung wird nun erreicht, indem zwecks Zugriff auf die Arbeitsbereiche der drei Vektoren nicht mehr die komplette Adresse auf dem Adressbus übertragen wird, sondern nur die Nummer des Arbeitsbreichs sowie eine Offsetadresse. Mit z.B. zwei Leitungen lassen sich vier Arbeitsbereiche ansprechen. Wenn für die Übertragung des Offsets z.B. 6 Bits verwendet werden, werden zur Übertragung der Adresse an den Speicher nur acht Busleitungen angesprochen. Zusätzliche Energiekosten fallen an, wenn auf eine Adresse zugegriffen wird, die nicht innerhalb eines Arbeitsbereichs liegt. Zur Codierung der Adresse in Arbeitsbereich und Offset wird für die CPU zusätzliche Hardware für einen

Encoder benötigt. Am Speicher wird die übertragene Codierung mittels eines Decoders wieder in eine reale Adresse umcodiert. Zur Energieeinsparung darf der zusätzliche Energiebedadrf für Encoder und Decoder die durch verringerte Busaktivitäten eingesparte Energie nicht wieder zunichte machen.

5.2 Grundlagen

5.2.1 Hamming-Distanzen aufeinander folgender Zahlen

Zur Ausführung einer linearen Instruktionssequenz ohne Sprünge werden am Adressbus fortlaufende Adressen angelegt. Bei der Adressierung unterschiedlicher Adressen unterscheiden sich deren binäre Codierungen mindestens um 1 Bit, d.h. die Hamming-Distanz beträgt mindestens 1. Abbildung 5.1 zeigt für 128 aufeinander folgende Adressen, beginnend bei Adresse 0, wie groß die Distanz zwischen Adresse $i-1$ und i ist. Es wird deutlich, dass es mit steigenden Adressen immer mal wieder ein Adressenpaar gibt, deren Hamming-Distanz um 1 größer ist, als die bisher größte Distanz war. Diese Adressenpaare liegen bei $(2^i-1, 2^i)$, für $i = 0, 1, 2, \ldots$. Außerdem treten bei höheren Adressen tendenziell größere Hamming-Distanzen auf, als bei niedrigen Adressen. Dies ist damit begründet, dass bei hohen Adressen mehr Bits wechseln können, als bei niedrigen: Wenn bei einer Zahl a, deren Codierung mit n Bits realisiert ist, alle $m \leq n$ niederwertigsten Bits '1' sind, so ist $h(a, a+1) = m+1$.

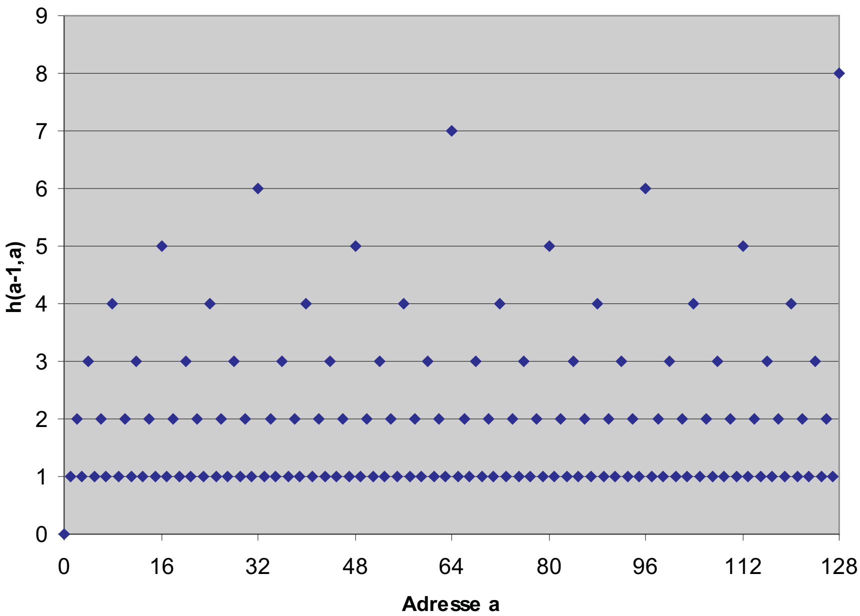


Abbildung 5.1: Hamming-Distanzen zwischen aufeinander folgenden Adressen

5.2.2 Gray Code

Zur Minimierung der Bit-Wechsel zwischen zwei aufeinander folgenden Zahlen wäre eine Codierung wünschenswert, bei der sich aufeinander folgende Zahlen nur um 1 Bit unterscheiden. Diese Eigenschaft hat der Gray Code. Beispielsweise würde eine Codierung auf dem Adressbus, welche alle Hamming-Distanzen zwischen verschiedenen Adressen zu Eins werden ließe, das Optimum sein. Formal lässt sich der Gray Code z.B. nach [STD94a] beschreiben als:

Der Gray Code ist eine Folge von Zahlen, die aus einer Kombination von Nullen und Einsen besteht. Aufeinander folgende Zahlen unterscheiden sich durch genau 1 Bit. Unter N_x sei im folgenden die Gray-Codierung mit x Bits verstanden.

Es sei $N_1 = 0, 1$ und $N_k = n_0, n_1, \ldots, n_{2^k-2}, n_{2^k-1}$. N_{k+1} wird erzeugt, indem allen Zahlen der Folge N_k am MSB eine '0' vorangestellt wird. Anschließend werden alle Zahlen von N_k in umgekehrter Reihenfolge wiederholt und eine '1' vorangestellt:

$N_{k+1} = 0n_0, 0n_1, \ldots, 0n_{2^k-2}, 0n_{2^k-1}, 1n_{2^k-1}, 1n_{2^k-2}, \ldots, 1n_1, 1n_0$

Beispiel: Erzeugung von N_3 aus $N_2 = 00, 01, 11, 10$:

$N_3 = 000, 001, 011, 010, 110, 111, 101, 100.$

5.2.3 Berechnung kumulierter Hamming-Distanzen von aufeinander folgenden Adressen

Zu Optimierungszwecken, z.B. zur Anordnung von Basisblöcken im Speicher, ist es notwendig, die Summe h_{sum} der Hamming-Distanzen zwischen aufeinander folgenden Adressen aus einem Adressbereich $[a, b]$, mit $a < b$, zu berechnen. a und b sind binäre Codeworte, bestehend aus n Bits. Mit Hilfe der Definition der Hamming-Distanz aus Abschnitt 3.2.1 sieht die Berechnung wie folgt aus:

Definition: Hamming-Distanz eines Adressbereichs

$$h_{sum}(a, b) = \sum_{i=a}^{b-1} h(i, i+1) = \sum_{i=a}^{b-1} \sum_{j=0}^{n-1} \Big(i_j \text{ xor } (i+1)_j \Big)$$

Hierbei ist i_j bzw. $(i+1)_j$ das j-te Bit der binären Darstellung von i bzw. von $(i+1)$.

Nachteilig ist an dieser Formel, dass die summierte Hamming-Distanz berechnet wird, indem die Hamming-Distanzen aller aufeinander folgenden Adressen einzeln berechnet werden und anschließend die gesamte Summe gebildet wird.

Besser wäre es, $h_{sum}(a, b)$ direkt aus a und b zu berechnen. Dazu wird die summierte Hamming-Distanz zwischen dem Null-Codewort '000...' und einem anderen Codewort a untersucht. Die Hamming-Distanz dieses Adressbereichs lässt sich relativ einfach berechnen. Dazu enthält Tabelle 5.1 alle Hamming-Distanzen für den Adressbereich $[0, a]$, für $a = 0, \ldots, 15$.

a dezimal	a binär	$h_{sum}(0,a)$
0	00000	0
1	00001	1
2	00010	3
3	00011	4
4	00100	7
5	00101	8
6	00110	10
7	00111	11
8	01000	15
9	01001	16
10	01010	18
11	01011	19
12	01100	22
13	01101	23
14	01110	25
15	01111	26
$\vdots$	$\vdots$	$\vdots$

Tabelle 5.1: Hamming-Distanzen für Adressbereiche von Adresse 0 bis Adresse a

Betrachtet werden die einzelnen Bits a_i (für $i = 0, \ldots, n-1$) von a in der binären Darstellung. Es wird untersucht, wie oft die einzelnen Bits im Adressbereich $[0, a]$, z.B. für $a = 15$, kippen. Bit b_0 wechselt an jedem Adressübergang, b_1 an jedem zweiten, b_2 an jedem vierten und b_3 an jedem achten Übergang. Somit kippt b_0 15-Mal, b_1 7-Mal, b_2 3-Mal und b_3 1-Mal. Verallgemeinert lässt sich sagen, dass ein Bit b_i, unter Berücksichtigung der Größe a des Adressbereichs sowie der Wertigkeit 2^i des Bits, ungefähr $\frac{a}{2^i}$ -Mal kippt. Innerhalb des Adressbereichs finden Bit-Wechsel aber nur bei Adressübergängen zu den Adressen $k \cdot 2^i$ (für $i \in N$) statt. Die genaue Zahl der Bitwechsel ergibt sich durch Abrunden der durchgeführten Division: $\left\lfloor \frac{a}{2^i} \right\rfloor$.

Die Summe aller einzelnen Bitwechselhäufigkeiten ergibt die Hamming-Distanz des Adressraums. Der Zusammenhang zwischen Adressbereich $[0, a]$ und dessen Hamming-Distanz lässt sich formal wie folgt schreiben:

$$h_{sum}(0, a) = \sum_{i=0}^{n-1} \left\lfloor \frac{a}{2^i} \right\rfloor$$

Die Hamming-Distanz des Adressraums $[a, b]$ kann nun mit Hilfe der vorherigen Formel berechnet werden als:

$$h_{sum}(a,b) = h_{sum}(0,b) - h_{sum}(0,a) = \sum_{i=0}^{n-1} \left\lfloor \frac{b}{2^i} \right\rfloor - \sum_{i=0}^{n-1} \left\lfloor \frac{a}{2^i} \right\rfloor \tag{5.1}$$

Diese Formel summiert nicht mehr über alle einzelnen Hamming-Distanzen des Adressraums, sondern berechnet die gesamte Hamming-Distanz direkt aus der Anfangsadresse a und der Endadresse b.

5.3 Problemformulierungen

Die Optimierungen arbeiten überwiegend auf den Effekten von Buskosten. Abbildung 5.2 verdeutlicht, wo diese anfallen. Ein Maschinencode-Programm besteht aus mehreren Basisblöcken. Ein Basisblock enthält eine Folge von Instruktionen, die sequenziell vom Prozessor ausgeführt werden. Am Ende eines Basisblocks kann ein bedingter, ein unbedingter oder gar kein Sprungbefehl stehen. Bei bedingten Sprüngen wird auf jeden Fall in einen anderen Basisblock gesprungen. Bei einem unbedingten Sprung wird entweder zu einem anderen Basisblock gesprungen (engl.: *branch taken*) oder der Kontrollfluss wird linear fortgesetzt (engl.: *branch not taken*), d.h. es wird in den nächsten Basisblock übergegangen.

Instruktionen sind als Instruktionsworte gegeben. Diese Instruktionsworte $IData$ werden in der Instruction-Fetch Phase über den Instruktionsbus vom Speicher in den Prozessor transportiert. Wenn der Speicher ein solches Instruktionswort auf den Bus legt, fallen zwischen diesem und dem vorherigen Instruktionswort, welches als letztes übertragen wurde, Hamming-Distanz-Kosten $h(IData1, IData2)$ an. Beim Holen einer Instruktion fallen nicht nur Kosten für Hamming-Distanzen auf dem Datenbus an, sondern entsprechend auch für den Adressbus.

Bei Datenzugriffen auf den Stack oder auf andere Datenbereiche fallen, ebenso wie bei Instruktionen, Kosten auf dem Adressbus ($h(DAddr1, DAddr2)$) und Datenbus an.

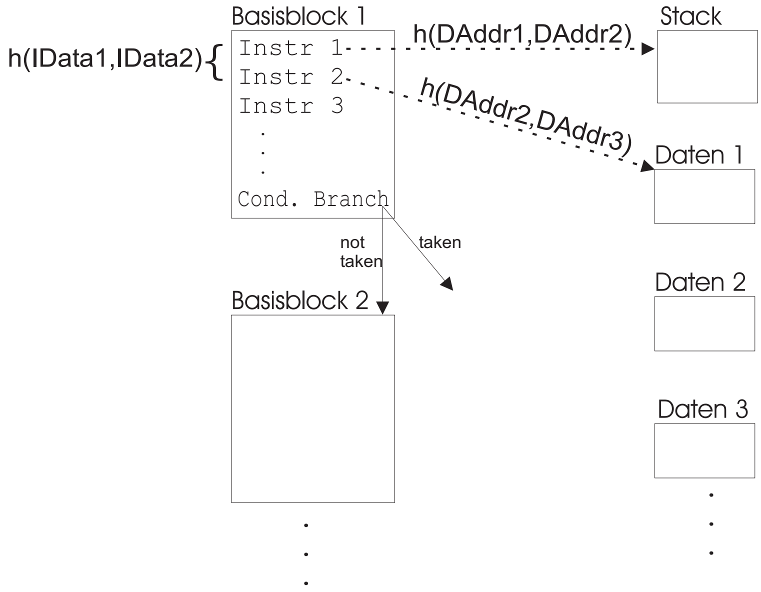


Abbildung 5.2: Speicherlayout und Buskosten

Zur Durchführung von Compileroptimierungen mit dem Ziel der Energieoptimierung werden nachfolgend 6 Möglichkeiten der Optimierung vorgestellt. Die einzelnen Optimierungen beruhen auf Energieeffekten, die in dieser Diplomarbeit im Rahmen des Energiemodells und der anschließenden Anwendung des ARM7TDMI-Prozessors erarbeitet und aufgezeigt wurden.

1. Innerhalb von Basisblöcken: Instruction Scheduling bzgl. Hamming-Distanzen von Instruktionsworten

2. Innerhalb von Basisblöcken: Instruction Scheduling bzgl. FUChange-Kosten

3. Anordnung von Basisblöcken im Adressraum

4. Anordnung von Funktionen im Adressraum

5. Adresszuordnung zu Variablen

6. Registerzuordnung auf Funktionsebene

Die Optimierungen arbeiten überwiegend auf den Effekten der Buskosten. Dazu werden die Codierungen von Instruktionen benötigt. Diese stehen erst nach allen anderen Optimierungen, Registerzuweisungen usw. zur Verfügung. Deshalb ist es sinnvoll, die nachfolgenden Optimierungen im Compiler als Post-Pass-Prozess durchzuführen, d.h. erst nach allen sonstigen Compiler-Optimierungen.

5.3.1 Scheduling bzgl. Hamming-Distanzen von Instruktionsworten

Der ausgenutzte Effekt beruht auf unterschiedlichen Energiewerten, die von Hamming-Distanzen auf dem Instruktionsbus zwischen den einzelnen Instruktionsworten stammen.

Zur Verringerung der Energien, die von Datenzugriffen auf dem Datenbus verursacht werden, müssten die übertragenen Daten bereits vor Ausführung des Programmcodes bekannt sein. Diese Information hängt aber von den Eingabedaten des Programms ab und ist während des Compiliervorgangs meist nicht vorhanden. Datenzugriffe werden deshalb zum Zweck der Optimierung nicht betrachtet.

Die Optimierung arbeitet auf den Codierungen von Instruktionsworten. Bei jedem Instruction-Fetch legt der Speicher ein Instruktionswort auf den Datenbus. Für diese Instruktion fallen Hamming-Distanz-Kosten an. Durch unterschiedliche Reihenfolgen der Instruktionsworte fallen verschiedene Hamming-Distanzen ab, die wiederum zu verschiedenen Energieverlusten führen.

Zur Optimierung wird die Reihenfolge der Instruktionsworte so geändert, dass die Summe aller Hamming-Distanzen zwischen den einzelnen Instruktionen minimal ist. Als Nebenbedingungen sind Datenabhängigkeiten zwischen den Instruktionen zu berücksichtigen. Diese schränken die Möglichkeiten des Schedulings ein. Ferner arbeitet diese Optimierung nur innerhalb von Basisblöcken, Instruktionen können also nicht über die Grenzen eines Blocks hinaus verschoben werden.

Bei einem gemeinsam vorhandenen Instruktions- und Datenspeicher können zwischen Instruction-Fetches auch noch Datenzugriffe stattfinden. Diese verändern den Zustand auf dem Bus in einen nicht bekannten Wert und bleiben unberücksichtigt. Im nachfolgenden wird von einem getrennten Instruktions- und Datenspeicher ausgegangen.

Formale Beschreibung

Ein Basisblock B führe n Instruction-Fetches durch. Die Instruktionsworte seien mit $IData_i$, für $i = 1, \ldots, n$, gegeben.

Unter Berücksichtigung von Datenabhängigkeiten ist mittels Instruction Scheduling folgende Kostenfunktion zu minimieren:

$$C_1 = \sum_{i=2}^{n} h(IData_{i-1}, IData_i)$$

n verschiedene Instruktionen lassen sich unter Berücksichtigung der Reihenfolge auf $n!$ verschiedene Arten anordnen. Somit beträgt der Aufwand, d.h. die Laufzeit für diese Optimierung $O(n!)$. In der Praxis ist aber die Anzahl n der Instruktionen, die umsortiert werden dürfen, auf Grund von Datenabhängigkeiten nicht sehr groß.

5.3.2 Scheduling bzgl. FUChange-Kosten

Für diese Optimierung wird der Effekt der FUChange-Kosten ausgenutzt. Abhängig davon, welche Instruktionen hintereinander ausgeführt werden und welche Funktionseinheiten diese ansprechen, fallen unterschiedliche FUChange-Kosten an. Diese Eigenschaft wird ausgenutzt, um innerhalb eines Basisblocks FUChange-Kosten zu minimieren. Unter Berücksichtigung aller Datenabhängigkeiten zwischen den Instruktionen werden diese zur Minimierung der Kosten umsortiert.

Formale Beschreibung

Ein Basisblock B bestehe aus n Instruktionen $Instr_i$ (für $i = 1, \ldots, n$). Unter Berücksichtigung von Datenabhängigkeiten ist mittels Instruction Scheduling die folgende Kostenfunktion zu minimieren:

$$C_2 = \sum_{i=2}^{n} FUChange(Instr_{i-1}, Instr_i)$$

Der Aufwand für diese Optimierung beträgt $O(n!)$, ist aber in der Praxis ebenso wie bei der vorherigen Optimierung meist kleiner.

5.3.3 Anordnung von Basisblöcken im Adressraum

Betrachtet werden hier unterschiedliche Energiekosten, die von Hamming-Distanzen auf dem Adressbus stammen. In Abhängigkeit von der Lage eines Basisblocks im Speicher sind die Hamming-Distanzen des Adressraums unterschiedlich. Wie bereits in Abschnitt 5.2.1 gezeigt wurde, sind die Größen von Hamming-Distanzen von aufeinander folgenden Adressen im Adressraum nicht homogen verteilt, sondern sie nehmen tendenziell mit größeren Adressen zu.

Zu Optimierungszwecken werden Basisblöcke im Speicher verschoben, so dass die Hamming-Distanzen des Adressbereichs, welche sich bei Instruction-Fetches auswirken, verkleinert werden. Bei der Durchführung des Verfahrens werden zusätzlich die Aufrufhäufigkeiten von Basisblöcken berücksichtigt, die durch eine statische Kontrollflussanalyse gewonnen werden können.

Am Ende eines Basisblocks ohne Sprung-Befehl oder mit bedingtem Sprung-Befehl kann der Kontrollfluss, ohne dass ein Sprung durchgeführt wird, in einen anderen Basisblock übergehen. Wenn durch Optimierung diese beiden Basisblöcke auseinander gerissen würden, müsste am Ende des ersten Basisblocks ein Sprungbefehl hinzugefügt werden. Es ist nicht zu erwarten, dass die Energiekosten für diesen zusätzlichen Befehl durch niedrigere Buskosten auf dem Adressbus wettgemacht werden können. Somit wird dieses Optimierverfahren nur auf Basisblöcke angewendet, die am Ende einen unbedingten Sprung (engl.: *unconditional branch*) durchführen.

Formale Beschreibung

Gegeben sei ein Adressraum A, bestehend aus n Adressen. Ferner seien k Basisblöcke B_i (für $i = 1, \ldots, k$) gegeben, deren letzte Anweisung ein unbedingter Sprung ist. Jeder Basisblock B_i besitze die Aufrufhäufigkeit g_i sowie die Start- und Endadresse $B_{i_{start}} \in A$ und $B_{i_{end}} \in A$. Durch Anordnung aller Basisblöcke im Adressraum A ist die folgende Kostenfunktion zu minimieren:

$$C_3 = \sum_{i=1}^{k} g_i \cdot h_{sum}(B_{i_{start}}, B_{i_{end}})$$

h_{sum} ist die in Abschnitt 5.2.3 eingeführte Gleichung 5.1. Zur Laufzeitabschätzung wird der worst case angenommen, bei dem alle k Basisblöcke aus nur einer Instruktion bestehen. Dann ergibt sich für den Aufwand dieser Optimierung, die k Basislöcken jeweils eine von n Adressen zuordnet: $O(n^k)$.

Anstatt nur Basisblöcke zu berücksichtigen, die mit einem unbedingten Sprung enden, können für die Optimierung auch Folgen von Basisblöcken mit einbezogen werden, wo genau der letzte Basisblock mit einem unbedingten Sprung endet. Hierbei muss allerdings die Aufrufhäufigkeit g_i entsprechend in Abhängigkeit der Einzelgewichte der Basisblöcke berechnet werden.

In Abschnitt 4.5 hat sich gezeigt, dass Ones-Kosten auf dem Adressbus zu unterschiedlichen Energien führen können. Dieser Effekt könnte bei dieser Optimierung sowie bei den drei nächsten Verfahren, die ebenfalls das Speicherlayout ändern, mit berücksichtigt werden.

5.3.4 Anordnung von Funktionen im Adressraum

Funktionen bestehen aus einem oder mehreren Basisblöcken. Funktionen werden aus anderen Basisblöcken aufgerufen und Übergabeparameter sowie die Rücksprungadresse der aufrufenden Stelle werden auf dem Stack gespeichert. Im Gegensatz zu Basisblöcken werden Funktionen verlassen, indem die Rücksprungadresse vom Stack direkt in den Program Counter geladen wird. Es findet also kein bedingter oder unbedingter Sprung statt. Zu Optimierungszwecken werden Funktionen als Ganzes im Speicher verschoben, so dass die Hamming-Distanzen, die bei Instruction-Fetches anfallen, verringert werden. Die Aufrufhäufigkeiten der Funktionen werden für die Optimierung mit berücksichtigt.

Formale Beschreibung

Gegeben sei ein Adressraum A, bestehend aus n Adressen. Ferner seien k Funktionen F_i (für $i = 1, \ldots, k$) gegeben. Jede Funktion F_i besitze die Aufrufhäufigkeit g_i sowie die Start- und Endadresse $F_{i_{start}} \in A$ und $F_{i_{end}} \in A$. Durch Anordnung aller Funktionen im Adressraum A ist die folgende Kostenfunktion zu minimieren:

$$C_4 = \sum_{i=1}^{k} g_i \cdot h_{sum}(F_{i_{start}}, F_{i_{end}})$$

Der Aufwand für diese Optimierung beträgt $O(n^k)$.

5.3.5 Adresszuordnung zu Variablen

Variablen können einerseits in Registern gespeichert werden, andererseits im Speicher. Für Optimierungen werden hier nur Variablen betrachtet, die im Speicher abgelegt werden. Zusätzlich findet eine Unterscheidung in lokale und globale Variablen statt. Lokale Variablen werden normalerweise auf einem Stack abgelegt, globale hingegen in einem anderen Datenbereich im Speicher.

Betrachtet wird ein System, bei dem Instruktions- und Datenspeicher separat sind. Zugriffe auf Variablen erfolgen durch Verwendung von Load-/Store und Push-/Pop-Operationen. Diese Operationen führen Datenzugriffe auf den Speicher durch, indem eine Adresse auf den Adressbus gelegt wird. Zu Optimierungszwecken werden innerhalb der Basisblöcke alle Hamming-Distanzen auf dem Adressbus addiert, die durch Datenzugriffe auf die Variablen zustande kommen. Diese Summe wird mit der Aufrufhäufigkeit gewichtet. Zur Durchführung der Optimierung sind alle gewichteten Summen durch eine Zuordnung von Adressen zu Variablen zu minimieren.

Formale Beschreibung

Gegeben sei ein Adressraum A mit m möglichen Adressen. Ferner existieren k Basisblöcke B_i (für i=1,...,k) mit den Aufrufhäufigkeiten g_i. Jeder Basisblock B_i führe Bn_i Datenzugriffe auf globale Variablen durch. Die Adressen der Datenzugriffe seien gegeben durch $B_{i,DAddr_j} \in A$, für $j = 1, \ldots, Bn_i$.

Durch Anordnung der Variablen im Adressraum A ist die Kostenfunktion C_5 zu minimieren.

$$C_5 = \sum_{i=1}^{k} g_i \cdot \sum_{j=2}^{Bn_i} h(B_{i,DAddr_{j-1}}, B_{i,DAddr_j})$$

Der Aufwand für diese Optimierung beträgt $O(m^k)$.

Wenn ein gemeinsamer Instruktions- und Datenspeicher gegeben ist, werden auf dem Adressbus nicht nur Hamming-Distanzen zwischen Datenzugriffen berücksichtigt, sondern zusätzlich noch Hamming-Distanzen auf dem Adressbus zwischen Daten- und Instruktionszugriffen.

5.3.6 Registerzuordnung auf Funktionsebene

Wie die erste Optimierung verringert auch dieses Verfahren die Bit-Wechsel auf dem Datenbus, welche durch Instruction-Fetches verursacht werden. Dieses Verfahren führt allerdings kein Instruction Scheduling durch, sondern verringert die Hamming-Distanzen der Instruktionsworte durch Registerzuordnung. Die Angabe der von einer Instruktion verwendeten Register ist Bestandteil des Instruktionswortes. Die optimale Vergabe von Registern kann auf der Ebene von Funktionen durchgeführt werden. Berücksichtigt werden die Basisblöcke der Funktionen sowie die Aufrufhäufigkeiten der Basisblöcke.

Formale Beschreibung

Eine Funktion bestehe aus k Basisblöcken B_i (für i=1,...,k) mit den Aufrufhäufigkeiten g_i. Jeder Basisblock B_i enthalte Bn_i Instruktionen. Die Instruktionsworte der Instruktionen seien bezeichnet mit $B_{i,IData_j} \in A$, für $j = 1, \ldots, Bn_i$.

Durch Vergabe von Registern ist folgende Kostenfunktion C_6 zu minimieren:

$$C_6 = \sum_{i=1}^{k} g_i \cdot \sum_{j=2}^{Bn_i} h(B_{i,IData_{j-1}}, B_{i,IData_j})$$

Wenn k Register zur Verfügung stehen, ist der Aufwand für diese Optimierung $O(k!)$.

Nach der Auswertung von Abschnitt 4.5 können Ones-Kosten auf dem Datenbus zu erhöhten Energiewerten führen. Dies kann zur Optimierung ausgenutzt werden, indem den Ones-Kosten entsprechend energiegünstige Register vergeben werden.

5.4 Kombination der Optimierungen

In Kapitel 5.3 wurden einzelne Optimierverfahren getrennt voneinander vorgestellt. Zur Anwendung der Optimierungen wäre es wünschenswert, alle Verfahren zusammenzufassen. In der Praxis ist es aber schwierig, ein solches gemeinsames Verfahren aufzustellen, welches effizient, d.h. mit einer akzeptablen Laufzeit, arbeitet. Deshalb werden die Optimierungen hintereinander ausgeführt. Dieses Vorgehen führt zwar nicht zwangsweise zu einem optimalen Ergebnis, aber die Ergebnisse werden als hinreichend genau genug erwartet. Da sich die Verfahren gegenseitig beeinflussen können, ist eine Strategie notwendig, in welcher Reihenfolge die Verfahren angewendet werden. Dazu erfolgt zuerst eine Betrachtung, wie sich die Verfahren gegenseitig beeinflussen.

Beeinflussung der Optimierverfahren untereinander

Abbildung 5.3 zeigt die Abhängigkeiten der Verfahren auf. Jedes Optimierverfahren ist durch einen Kreis dargestellt, die Abhängigkeiten sind durch gerichtete Pfeile dargestellt. Ein Pfeil von einem Verfahren a nach Verfahren b bedeutet, dass die Durchführung der Optimierung a Ergebnisse von Optimierung b beeinflussen kann.

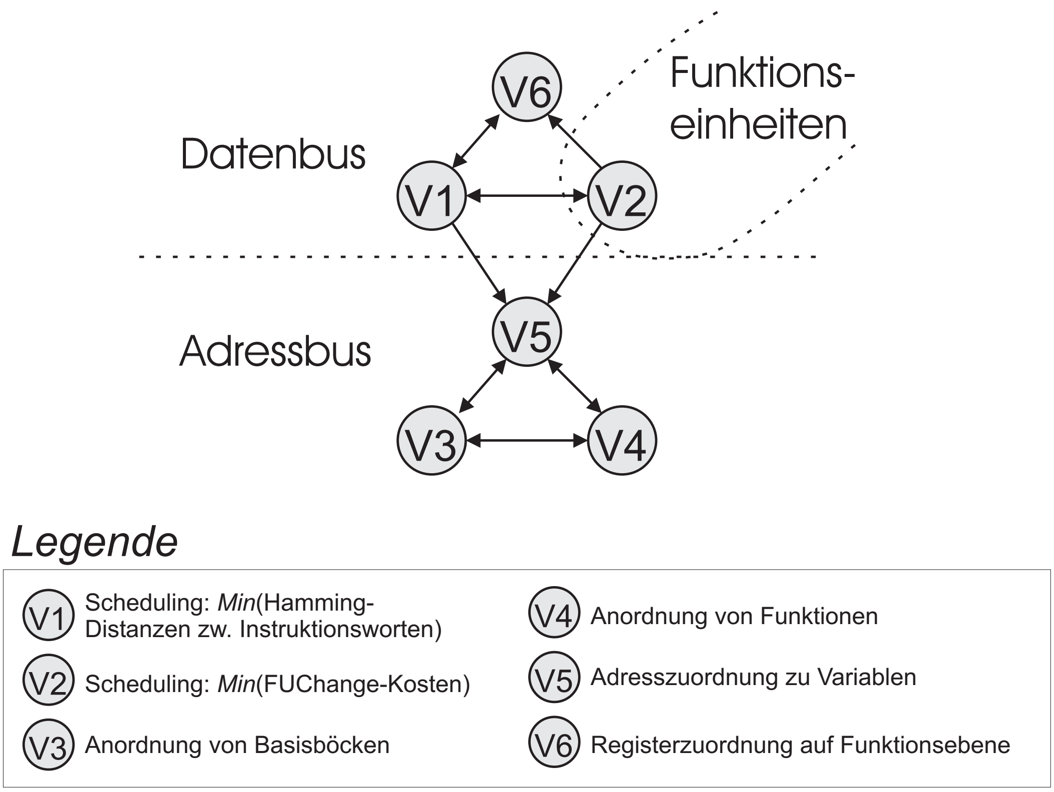


Abbildung 5.3: Abhängigkeiten der Optimierverfahren untereinander

Die Verfahren lassen sich danach klassifizieren, in welchen Einheiten die Energieoptimierungen zum Tragen kommen. Verfahren $V2$ optimiert innerhalb von Funktionseinheiten, $V1$ und $V6$ optimiert die Anzahl der Bit-Wechsel auf dem Datenbus und $V3$, $V4$ und $V5$ auf dem Adressbus.

V1 und V2 führen beide ein Instruction Scheduling durch und beeinflussen sich damit gegenseitig. $V6$ basiert darauf, die Hamming-Distanzen zwischen Instruktionsworten zu verringern. Dieses Verfahren wird vom Scheduling von $V1$ und $V2$ beeinflusst. Ebenso wie $V6$ arbeitet auch $V1$ dahingehend, die Hamming-Distanzen zwischen Instruktionsworten zu verkleinern. Somit beeinflussen sich $V6$ und $V1$ gegenseitig.

Die Verfahren $V3$ bis $V5$ beruhen darauf, dass sie Objekte innerhalb des Adressraums platzieren bzw. verschieben. Der Adressraum ist eine Resource mit beschränkter Grösse. Somit konkurrieren $V3$ bis $V5$ untereinander, es bestehen Abhängigkeiten zwischen diesen 3 Verfahren. Diese Verfahren verändern jedoch weder die Codierungen der Instruktionsworte, noch die Reihenfolge von Instruktionen innerhalb von Basisblöcken. Ferner haben sie keinen Einfluss auf die verwendeten Funktionseinheiten. Somit beeinflussen $V3$ bis $V5$ nicht die Verfahren $V1$, $V2$ oder $V6$. Allerdings kann das Scheduling von $V1$ und $V2$ die Hamming-Distanzen auf dem Adressbus zwischen Instruktions-Adresse und Datenadresse verändern. Deshalb haben $V1$ und $V2$ Einfluss auf Verfahren $V5$.

Strategie zur Ausführung der Optimierverfahren

Verfahren $V1$ und $V2$ arbeiten ähnlich und führen beide ein Instruction Scheduling durch. Das erste Verfahren optimiert bzgl. Hamming-Distanzen zwischen Instruktionsworten, das zweite Verfahren bzgl. FUChange-Kosten. Beide Schedulings arbeiten innerhalb von Basisblöcken. $V1$ und $V2$ lassen sich kombinieren und gemeinsam durchführen.

Auch die Funktionsweise von $V3$ und $V4$ ist ähnlich. Basisblöcke bzw. Funktionen werden im Speicher angeordnet, um die Hamming-Distanzen der jeweiligen Adressbereiche zu minimieren. $V4$ kann zusammen mit $V3$ durchgeführt werden, indem in Verfahren 4 eine Funktion nicht als komplette Einheit betrachtet wird, sondern als Menge von Basisblöcken, aus der sich die Funktion zusammensetzt. Abbildung 5.4 zeigt alle Zusammenfassungen sowie die neuen Beeinflussungen.

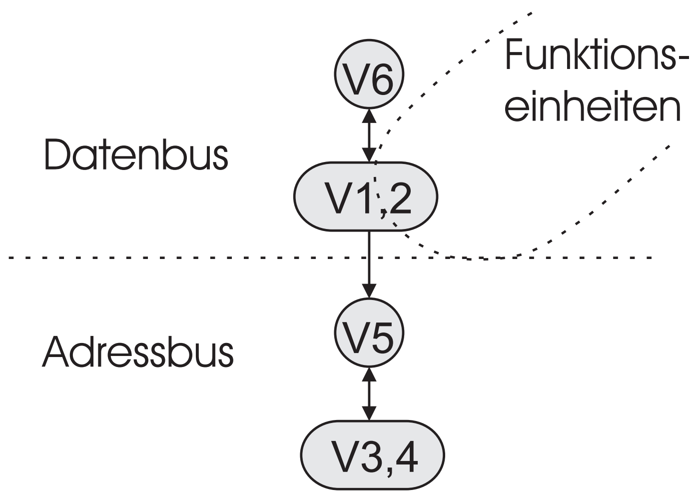


Abbildung 5.4: Zusammenfassung der Optimierverfahren und Abhängigkeiten

Es folgt die Betrachtung, in welcher Reihenfolge die Verfahren ausgeführt werden. Optimierungen $V1,2$ und $V6$ beeinflussen sich gegenseitig. Diese beiden Verfahren werden deshalb wiederholt ausgeführt. Gestartet wird mit Verfahren $V1,2$. Benötigt wird ein Abbruchkriterium, wann die Wiederholung beendet und zur nächsten Optimierung übergegangen wird. Beendet wird die Schleife, wenn sich bei $V6$ im Vergleich zur vorherigen Ausführung von $V6$ keine Verbesserung mehr zeigt. Diese Vorgehensweise des wechselseitigen Ausführens führt nicht zwangsweise zu einem optimalen Ergebnis. Vielmehr garantiert es schrittweise Verbesserungen, bis die Abbruchbedingung erreicht ist.

Die beiden nächsten Verfahren $V5$ und $V3,4$ optimieren, indem sie das Layout im Adressraum ändern. Dies hat keinen Einfluss auf die beiden vorherigen Optimierungen. Somit verbleibt die Aufgabe, eine Entscheidung der Reihenfolge für die beiden letzten Optimierungen zu treffen.

Die einzige Beeinflussung von $V5$, der Adresszuordnung von Variablen, auf Verfahren $V3,4$, welches Basisblöcke und Funktionen im Adressraum anordnet, besteht darin, dass beide Verfahren bzgl. der gleichen Resource „Speicher" konkurrieren. Demnach kann $V5$ nach der Anwendung von $V3,4$ zu keiner Verschlechterung der Optimierung $V3,4$ führen. Wenn hingegen erst $V5$ durchgeführt würde, könnte durch Verfahren $V3,4$ ein Großteil der gewonnenen Optimierung wieder zunichte gemacht werden. Es empfielt sich somit, erst $V3,4$ und dann $V5$ anzuwenden.

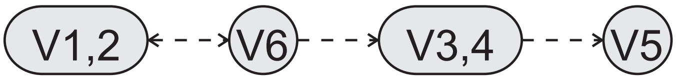


Abbildung 5.5: Reihenfolge der Optimierverfahren

Abbildung 5.5 zeigt die letztendliche Reihenfolge der Optimierungen. Bei der Abbildung ist zu beachten, dass die eingezeichneten Pfeile, im Gegensatz zu den beiden letzten Grafiken, keine Abhängigkeiten wiedergeben. Die Pfeile dienen lediglich zur Angabe der Reihenfolge.

Kapitel 6

Zusammenfassung und Ausblick

Zusammenfassung

Mit dem Ziel, in Prozessorsystemen durch Compileroptimierungen Energieeinsparungen durchzuführen, wurde als Grundstein in dieser Diplomarbeit ein neues Energiemodell entwickelt. Das Modell erfasst insbesondere Energiekosten, die durch Zustandswechsel auf Bussen anfallen. Vor allem externe Speicherbusse haben auf Grund ihrer Länge und der meist großen Busbreiten hohe Leitungskapazitäten. Das Umladen dieser Kapazitäten zum Zweck von Pegelwechseln führt zu hohen Schaltströmen, die einen großen Energieverlust nach sich ziehen. Verursacht durch unterschiedliche Leckströme erfasst das Energiemodell auch Buskosten, die nicht von Schaltvorgängen stammen. An Bussen anliegende Bitmuster verbrauchen, in Abhängigkeit von der Anzahl der Einsen, unterschiedliche Energien.

Ein Prozessorsystem enthält nicht nur externe Busse, sondern innerhalb des Prozessors auch interne Busse. Z.B. werden Registerwerte zwischen Register File und Funktionseinheiten auf solchen Bussen transportiert. Innerhalb des Prozessors fallen bei der Ausführung von Instruktionen zum Aktivieren und Deaktivieren von Funktionseinheiten zusätzliche Energien an. Diese Kosten wurden im Modell als FUChange-Kosten bezeichnet. Energien, die im Prozessor für die Ausführung einer Instruktion bzw. im Speicher für Speicherzugriffe anfallen, werden in Form von Base-Kosten erfasst. Base-Kosten enthalten keine Kosten, die durch andere Kostenarten anfallen. Alle anfallenden Energiekosten werden im Modell getrennt für Speicher und Prozessor beschrieben. Ferner findet eine Unterteilung zwischen instruktions- und datenbezogenen Kosten statt. Als Vorteil des Energiemodells ist zu sehen, dass es viele verschiedene Parameter der Energiekosten erfasst und somit eine hohe Detailgenauigkeit hat. Trotzdem sind die Datenerhebungen zur Bestimmung der Modellparameter einfach durchzuführen und es werden keine technischen Daten des Herstellers benötigt.

Das vorgestellte Energiemodell wurde auf den ARM7TDMI-Prozessor angewendet. Zur Bestimmung der Modellparameter waren Datenerhebungen durchzuführen. Messreihen wurden mit Hilfe von Software-Messmodulen und geeigneten Testdaten erhoben. Da die Daten in keiner geeigneten Form vorlagen, um direkt im Modell zu verwenden, waren Umrechnungen und die Anwendung von statistischen Methoden erforderlich. Die Buskosten wurden mit der linearen Regressionsmethode durch eine Gerade approximiert. Bei der Auswertung hat sich gezeigt, dass sich z.B. durch größere Hamming-Distanzen auf dem Adressbus der Energieverbrauch um bis zu 13% erhöht. Diese Energiekosten können duch geeignete Optimierungen verringert werden.

Aufbauend auf dem Energiemodell und den erhobenen Messwerten wurden Optimierungen zur Energiereduktion vorgeschlagen. Die Optimierungen arbeiten auf den Effekten von Buskosten und FUChange-Kosten. Im einzelnen wurden Kosten auf dem Adressbus und dem Daten- bzw. Instruktionsbus betrachtet. Durch Instruction Scheduling, Verändern des Speicherlayouts und der Registerzuordnung lassen sich Energieeinsparungen erreichen. Da sich die Optimierverfahren bei der Anwendung gegenseitig beeinflussen, wurde eine Strategie der Durchführung vorgestellt.

Ausblick

Das entwickelte Energiemodell wurde in dieser Arbeit auf den ARM7TDMI-Prozessor angewendet. Es stellt sich die Frage, auf welche Prozessorsysteme das Modell problemlos anwendbar ist oder ob es für andere Systeme verändert werden müsste.

In dieser Arbeit wurden Vorschläge gemacht, welche Energieoptimierungen durchgeführt werden können. Zur Umsetzung müssen effiziente Algorithmen ausgesucht oder entwickelt werden. In diesem Zusammenhang ist zu untersuchen, welche Energieeinsparungen in typischen Anwendungen erreicht werden können.

Die Ausmaße der in dieser Arbeit gemessenen Einzelkosten auf die Energie lassen sich nicht pauschal auf andere Systeme übertragen. Es könnten Untersuchungen angestellt werden, ob die Effekte der Einzelkosten in anderen Systemen zu größeren oder kleineren prozentualen Energieerhöhungen führen. Auch wäre es denkbar, dass die Auswirkungen von Buskosten auf die Energie bei bestimmten Prozessorklassen besonders groß sind, d.h. dass Optimierungen zu großen Energiereduktionen führen.

Anhang A

Tabellen

A.1 Messreihen

Es folgt eine Auflistung der durchgeführten Messreihen. Die Messreihen wurden im Zusammenhang mit Kapitel 4 gewonnen. Nachfolgend eine Erklärung der Spalten-Bezeichnungen der Tabellen:

Spalte 'i'

Nummer des Messwertes der entsprechenden Reihe. Entspricht dem Index in den Tabellen 4.1 bis 4.4

Testmuster auf Bus

Testmuster auf dem Adress- oder Datenbus zur Gewinnung des Messwertes.

Anzahl '1'

Anzahl der Einsen der binären Testmuster-Codierung.

Position '1'

Angabe, an welchen Stellen im binären Codewort Einsen stehen. Für den Adressbus sind gültige Werte 0 bis 16, für den Datenbus 0 bis 15.

Messreihen CPU und Mem

Name der Reihen und einzelne Messwerte des Stroms in der CPU oder im Speicher in der Einheit mA.

Hamming-Distanz

Die Hamming-Distanz bezieht sich auf das angegebene Testmuster und das Codewort '000...'.

Position Bit-Wechsel

Zum vorherigen Unterpunkt 'Hamming-Distanz' wird angegeben, an welchen Bit-Position Einsen stehen.

i	Testmuster auf Adressbus	Anzahl '1'	Position '1'	CPU Read $M_{cpu,addr,w,r}$	CPU Write $M_{cpu,addr,w,w}$	Mem Read $M_{mem,addr,w,r}$	Mem Write $M_{mem,addr,w,w}$
1	0x00001	1	0	44,4	49,5	37,8	38,8
2	0x00002	1	1	44,4	49,5	35,0	36,2
3	0x00004	1	2	44,4	49,5	35,3	36,5
4	0x00008	1	3	44,4	49,5	35,7	37,0
5	0x00010	1	4	44,4	49,5	35,0	36,2
6	0x00020	1	5	44,4	49,5	34,9	36,2
7	0x00040	1	6	44,4	49,6	34,9	36,2
8	0x00080	1	7	44,5	49,7	34,9	36,2
9	0x00100	1	8	44,5	49,7	34,9	36,2
10	0x00200	1	9	44,5	49,7	35,0	36,3
11	0x00400	1	10	44,2	49,3	35,0	36,2
12	0x00800	1	11	44,4	49,6	35,0	36,2
13	0x01000	1	12	44,4	49,7	35,0	36,3
14	0x02000	1	13	44,5	49,8	34,9	36,2
15	0x04000	1	14	44,6	49,8	35,0	36,2
16	0x08000	1	15	44,5	49,7	34,9	36,2
17	0x10000	1	16	44,5	49,7	36,5	37,7
18	0x00000	0	-	44,4	49,6	34,8	36,1
19	0x00001	1	0	44,5	49,6	37,7	38,8
20	0x00003	2	0 bis 1	44,6	49,7	37,8	38,8
21	0x00007	3	0 bis 2	44,7	49,8	37,4	38,4
22	0x0000F	4	0 bis 3	44,7	49,9	36,7	37,7
23	0x0001F	5	0 bis 4	44,8	50,0	36,6	37,7
24	0x0003F	6	0 bis 5	44,9	50,1	36,6	37,7
25	0x0007F	7	0 bis 6	45,0	50,2	36,6	37,7
26	0x000FF	8	0 bis 7	45,2	50,4	36,6	37,7
27	0x001FF	9	0 bis 8	45,3	50,7	36,6	37,7
28	0x003FF	10	0 bis 9	45,5	50,9	36,5	37,6
29	0x007FF	11	0 bis 10	45,5	50,9	36,6	37,7
30	0x00FFF	12	0 bis 11	45,6	51,1	36,6	37,8
31	0x01FFF	13	0 bis 12	45,8	51,4	36,5	37,8
32	0x03FFF	14	0 bis 13	46,0	51,5	36,5	37,8
33	0x07FFF	15	0 bis 14	46,2	51,8	36,4	37,7
34	0x0FFFF	16	0 bis 15	46,3	52,0	36,4	37,6
35	0x1FFFF	17	0 bis 16	46,5	52,2	35,3	36,5

Tabelle A.1: Messreihen M: Ones-Kosten auf dem Adressbus

i	Testmuster auf Datenbus	Anzahl '1'	Position '1'	CPU Read $M_{cpu,data,w,r}$	CPU Write $M_{cpu,data,w,w}$	Mem Read $M_{mem,data,w,r}$	Mem Write $M_{mem,data,w,w}$
1	0x0001	1	0	44,5	49,4	34,6	36,3
2	0x0002	1	1	44,6	50,2	34,6	36,3
3	0x0004	1	2	44,8	50,2	34,6	36,3
4	0x0008	1	3	44,6	50,2	34,6	36,3
5	0x0010	1	4	44,4	49,4	34,6	36,3
6	0x0020	1	5	44,7	50,2	34,6	36,3
7	0x0040	1	6	44,4	49,4	34,6	36,3
8	0x0080	1	7	44,5	49,4	34,6	36,3
9	0x0100	1	8	44,6	50,2	34,9	36,6
10	0x0200	1	9	44,5	49,4	34,9	36,6
11	0x0400	1	10	44,6	50,1	34,9	36,6
12	0x0800	1	11	44,5	50,1	34,9	36,6
13	0x1000	1	12	44,5	49,5	34,9	36,6
14	0x2000	1	13	44,6	50,1	34,9	36,6
15	0x4000	1	14	44,5	49,5	34,9	36,6
16	0x8000	1	15	44,6	50,1	34,9	36,6
17	0x0000	0	-	44,4	49,6	34,9	36,4
18	0x0001	1	0	44,5	49,4	34,6	36,2
19	0x0003	2	0 bis 1	44,8	50,0	34,3	36,0
20	0x0007	3	0 bis 2	45,1	50,5	34,0	35,8
21	0x000F	4	0 bis 3	45,3	51,1	33,8	35,6
22	0x001F	5	0 bis 4	45,3	50,8	33,5	35,4
23	0x003F	6	0 bis 5	45,6	51,4	33,2	35,2
24	0x007F	7	0 bis 6	45,6	51,2	32,9	35,0
25	0x00FF	8	0 bis 7	45,8	51,0	32,6	34,8
26	0x01FF	9	0 bis 8	45,7	51,3	32,6	34,8
27	0x03FF	10	0 bis 9	45,5	50,9	32,6	34,8
28	0x07FF	11	0 bis 10	45,5	51,2	32,6	34,8
29	0x0FFF	12	0 bis 11	45,3	51,4	32,6	34,8
30	0x1FFF	13	0 bis 12	45,2	51,1	32,6	34,8
31	0x3FFF	14	0 bis 13	45,1	51,3	32,6	34,8
32	0x7FFF	15	0 bis 14	45,0	50,9	32,6	34,8
33	0xFFFF	16	0 bis 15	45,0	51,2	32,6	34,8

Tabelle A.2: Messreihen M: Ones-Kosten auf dem Datenbus

i	Testmuster auf Adressbus	Hamming-Distanz	Position Bit-Wechsel	CPU Read $M_{cpu,addr,h,r}$	CPU Write $M_{cpu,addr,h,w}$	Mem Read $M_{mem,addr,h,r}$	Mem Write $M_{mem,addr,h,w}$
1	0x00001	1	0	45,2	50,7	37,4	38,8
2	0x00002	1	1	45,2	50,8	35,9	37,2
3	0x00004	1	2	45,2	50,8	36,1	37,4
4	0x00008	1	3	45,2	50,8	36,3	37,6
5	0x00010	1	4	45,2	50,8	35,9	37,2
6	0x00020	1	5	45,2	50,8	35,5	36,7
7	0x00040	1	6	45,2	50,8	35,5	36,7
8	0x00080	1	7	45,2	50,8	35,5	36,7
9	0x00100	1	8	45,2	50,8	36,0	37,3
10	0x00200	1	9	45,3	50,9	36,0	37,3
11	0x00400	1	10	45,1	50,6	36,0	37,3
12	0x00800	1	11	45,2	50,8	35,9	37,3
13	0x01000	1	12	45,2	50,8	35,9	37,4
14	0x02000	1	13	45,3	50,9	35,9	37,4
15	0x04000	1	14	45,4	50,9	35,9	37,4
16	0x08000	1	15	45,5	51,2	35,9	37,4
17	0x10000	1	16	45,3	50,8	36,9	38,4
18	0x00000	0	-	44,6	49,9	35,0	36,4
19	0x00001	1	0	45,3	50,7	37,4	39,0
20	0x00003	2	0 bis 1	46,0	51,6	37,7	39,3
21	0x00007	3	0 bis 2	46,5	52,4	37,6	38,8
22	0x0000F	4	0 bis 3	47,1	53,1	37,1	38,6
23	0x0001F	5	0 bis 4	47,7	53,9	37,7	39,4
24	0x0003F	6	0 bis 5	48,3	54,7	38,0	39,8
25	0x0007F	7	0 bis 6	48,9	55,4	38,0	39,7
26	0x000FF	8	0 bis 7	49,5	56,3	38,2	40,0
27	0x001FF	9	0 bis 8	50,2	57,1	39,0	41,0
28	0x003FF	10	0 bis 9	50,8	58,0	39,2	41,2
29	0x007FF	11	0 bis 10	51,4	58,7	39,3	41,3
30	0x00FFF	12	0 bis 11	51,9	59,4	39,3	41,3
31	0x01FFF	13	0 bis 12	52,6	60,2	39,3	41,3
32	0x03FFF	14	0 bis 13	53,3	61,2	39,4	41,3
33	0x07FFF	15	0 bis 14	53,9	62,0	39,3	41,2
34	0x0FFFF	16	0 bis 15	54,7	63,0	39,3	41,1
35	0x1FFFF	17	0 bis 16	55,2	63,7	38,9	40,7

Tabelle A.3: Messreihen M: Hamming-Kosten auf dem Adressbus

i	Testmuster auf Datenbus	Hamming-Distanz	Position Bit-Wechsel	CPU Read $M_{cpu,data,h,r}$	CPU Write $M_{cpu,data,h,w}$	Mem Read $M_{mem,data,h,r}$	Mem Write $M_{mem,data,h,w}$
1	0x0001	1	0	45,3	51,5	37,4	38,9
2	0x0002	1	1	45,3	51,9	37,4	38,9
3	0x0004	1	2	45,3	51,8	37,4	38,9
4	0x0008	1	3	45,3	52,0	37,4	38,9
5	0x0010	1	4	45,3	51,6	37,4	38,9
6	0x0020	1	5	45,3	51,7	37,4	38,9
7	0x0040	1	6	45,3	51,5	37,4	38,9
8	0x0080	1	7	45,3	51,5	37,4	38,9
9	0x0100	1	8	45,3	51,8	37,4	38,9
10	0x0200	1	9	45,3	51,4	37,4	38,9
11	0x0400	1	10	45,3	51,7	37,4	38,9
12	0x0800	1	11	45,3	51,8	37,4	38,9
13	0x1000	1	12	45,3	51,5	37,4	38,9
14	0x2000	1	13	45,3	51,7	37,4	38,9
15	0x4000	1	14	45,3	51,5	37,4	38,9
16	0x8000	1	15	45,3	51,8	37,4	38,9
17	0x0000	0	-	45,3	50,7	37,4	38,9
18	0x0001	1	0	45,3	51,5	37,4	38,8
19	0x0003	2	0 bis 1	45,3	52,6	37,4	38,8
20	0x0007	3	0 bis 2	45,3	53,7	37,4	38,8
21	0x000F	4	0 bis 3	45,3	54,8	37,4	38,8
22	0x001F	5	0 bis 4	45,3	55,7	37,4	38,7
23	0x003F	6	0 bis 5	45,3	56,5	37,4	38,7
24	0x007F	7	0 bis 6	45,3	57,2	37,4	38,7
25	0x00FF	8	0 bis 7	45,3	57,8	37,4	38,7
26	0x01FF	9	0 bis 8	45,3	58,7	37,4	38,6
27	0x03FF	10	0 bis 9	45,3	59,2	37,4	38,6
28	0x07FF	11	0 bis 10	45,3	60,0	37,4	38,6
29	0x0FFF	12	0 bis 11	45,3	60,9	37,4	38,6
30	0x1FFF	13	0 bis 12	45,3	61,4	37,4	38,6
31	0x3FFF	14	0 bis 13	45,3	62,2	37,4	38,6
32	0x7FFF	15	0 bis 14	45,3	62,6	37,4	38,6
33	0xFFFF	16	0 bis 15	45,3	63,5	37,4	38,6

Tabelle A.4: Messreihen M: Hamming-Kosten auf dem Datenbus

A.2 Umrechnung der Speicherkosten auf zwei Speichermodule

Nachfolgend stehen Tabellen, die die errechneten Werte zu den Reihen S enthalten.

i	Mem Read $S_{mem,addr,w,r}$	Mem Write $S_{mem,addr,w,w}$	i	Mem Read $S_{mem,addr,w,r}$	Mem Write $S_{mem,addr,w,w}$
1	75,6	77,6	18	69,6	72,2
2	70,0	72,4	19	75,4	77,6
3	70,6	73,0	20	75,6	77,6
4	71,4	74,0	21	74,8	76,8
5	70,0	72,4	22	73,4	75,4
6	69,8	72,4	23	73,2	75,4
7	69,8	72,4	24	73,2	75,4
8	69,8	72,4	25	73,2	75,4
9	69,8	72,4	26	73,2	75,4
10	70,0	72,6	27	73,2	75,4
11	70,0	72,4	28	73,0	75,2
12	70,0	72,4	29	73,2	75,4
13	70,0	72,6	30	73,2	75,6
14	69,8	72,4	31	73,0	75,6
15	70,0	72,4	32	73,0	75,6
16	69,8	72,4	33	72,8	75,4
17	73,0	72,4	34	72,8	75,2
			35	70,6	73,0

Tabelle A.5: Reihen S für den Speicher: Ones-Kosten auf dem Adressbus

i	Mem Read $S_{mem,data,w,r}$	Mem Write $S_{mem,data,w,w}$	i	Mem Read $S_{mem,data,w,r}$	Mem Write $S_{mem,data,w,w}$
1	69,5	72,7	17	69,8	72,8
2	69,5	72,7	18	69,5	72,6
3	69,5	72,7	19	69,2	72,4
4	69,5	72,7	20	68,9	72,2
5	69,5	72,7	21	68,7	72,0
6	69,5	72,7	22	68,4	71,8
7	69,5	72,7	23	68,1	71,6
8	69,5	72,7	24	67,8	71,4
9	69,5	72,7	25	67,5	71,2
10	69,5	72,7	26	67,2	71,0
11	69,5	72,7	27	66,9	70,8
12	69,5	72,7	28	66,6	70,6
13	69,5	72,7	29	66,4	70,4
14	69,5	72,7	30	66,1	70,2
15	69,5	72,7	31	65,8	70,0
16	69,5	72,7	32	65,5	69,8
			33	65,2	69,6

Tabelle A.6: Reihen S für den Speicher: Ones-Kosten auf dem Datenbus

i	Mem Read $S_{mem,addr,h,r}$	Mem Write $S_{mem,addr,h,w}$	i	Mem Read $S_{mem,addr,h,r}$	Mem Write $S_{mem,addr,h,w}$
1	74,8	77,6	18	70,0	72,8
2	71,8	74,4	19	74,8	78,0
3	72,2	74,8	20	75,4	78,6
4	72,6	75,2	21	75,2	77,6
5	71,8	74,4	22	74,2	77,2
6	71,0	73,4	23	75,4	78,8
7	71,0	73,4	24	76,0	79,6
8	71,0	73,4	25	76,0	79,4
9	72,0	74,6	26	76,4	80,0
10	72,0	74,6	27	78,0	82,0
11	72,0	74,6	28	78,4	82,4
12	71,8	74,6	29	78,6	82,6
13	71,8	74,8	30	78,6	82,6
14	71,8	74,8	31	78,6	82,6
15	71,8	74,8	32	78,8	82,6
16	71,8	74,8	33	78,6	82,4
17	73,8	76,8	34	78,6	82,2
			35	77,8	81,4

Tabelle A.7: Reihen S für den Speicher: Hamming-Kosten auf dem Adressbus

i	Mem Read $S_{mem,data,h,r}$	Mem Write $S_{mem,data,h,w}$	i	Mem Read $S_{mem,data,h,r}$	Mem Write $S_{mem,data,h,w}$
1	74,8	77,8	17	74,8	77,8
2	74,8	77,8	18	74,8	77,7
3	74,8	77,8	19	74,8	77,7
4	74,8	77,8	20	74,8	77,7
5	74,8	77,8	21	74,8	77,7
6	74,8	77,8	22	74,8	77,6
7	74,8	77,8	23	74,8	77,6
8	74,8	77,8	24	74,8	77,6
9	74,8	77,8	25	74,8	77,6
10	74,8	77,8	26	74,8	77,5
11	74,8	77,8	27	74,8	77,5
12	74,8	77,8	28	74,8	77,5
13	74,8	77,8	29	74,8	77,5
14	74,8	77,8	30	74,8	77,4
15	74,8	77,8	31	74,8	77,4
16	74,8	77,8	32	74,8	77,4
			33	74,8	77,4

Tabelle A.8: Reihen S für den Speicher: Hamming-Kosten auf dem Datenbus

A.3 Ermittlung von Reihen für Einzelkosten

Die folgenden Tabellen enthalten die Werte zu den Reihen T. Die Summe in den beiden letzten Spalte ist die jeweilige Summe von CPU-Wert und Mem-Wert.

i	CPU Read $T_{cpu,addr,w,r}$	CPU Write $T_{cpu,addr,w,w}$	Mem Read $T_{mem,addr,w,r}$	Mem Write $T_{mem,addr,w,w}$	$\sum$ Read	$\sum$ Write
1	44,4	49,5	75,6	77,6	120,0	127,1
2	44,4	49,5	70,0	72,4	114,4	121,9
3	44,4	49,5	70,6	73,0	115,0	122,5
4	44,4	49,5	71,4	74,0	115,8	123,5
5	44,4	49,5	70,0	72,4	114,4	121,9
6	44,4	49,5	69,8	72,4	114,2	121,9
7	44,4	49,6	69,8	72,4	114,2	122,0
8	44,5	49,7	69,8	72,4	114,3	122,1
9	44,5	49,7	69,8	72,4	114,3	122,1
10	44,5	49,7	70,0	72,6	114,5	122,3
11	44,2	49,3	70,0	72,4	114,2	121,7
12	44,4	49,6	70,0	72,4	114,4	122,0
13	44,4	49,7	70,0	72,6	114,4	122,3
14	44,5	49,8	69,8	72,4	114,3	122,2
15	44,6	49,8	70,0	72,4	114,6	122,2
16	44,5	49,7	69,8	72,4	114,3	122,1
17	44,5	49,7	73,0	75,4	117,5	125,1
18	44,4	49,6	69,6	72,2	114,0	121,8
19	44,5	49,6	75,4	77,6	119,9	127,2
20	44,6	49,7	75,6	77,6	120,2	127,3
21	44,7	49,8	74,8	76,8	119,5	126,6
22	44,7	49,9	73,4	75,4	118,1	125,3
23	44,8	50,0	72,2	75,4	118,0	125,4
24	44,9	50,1	72,2	75,4	118,1	125,5
25	45,0	50,2	72,2	75,4	118,2	125,6
26	45,2	50,4	72,2	75,4	118,4	125,8
27	45,3	50,7	72,2	75,4	118,5	126,1
28	45,5	50,9	73,0	75,2	118,5	126,1
29	45,5	50,9	72,2	75,4	118,7	126,3
30	45,6	51,1	72,2	75,6	118,8	126,7
31	45,8	51,4	73,0	75,6	118,8	127,0
32	46,0	51,5	73,0	75,6	119,0	127,1
33	46,2	51,8	72,8	75,4	119,0	127,2
34	46,3	52,0	72,8	75,2	119,1	127,2
35	46,5	52,2	70,6	73,0	117,1	125,2

Tabelle A.9: Reihen T zur Ermittlung von Einzelkosten: Ones-Kosten auf dem Adressbus

i	**CPU Read** $T_{cpu,data,w,r}$	**CPU Write** $T_{cpu,data,w,w}$	**Mem Read** $T_{mem,data,w,r}$	**Mem Write** $T_{mem,data,w,w}$	$\sum$ **Read**	$\sum$ **Write**
1	44,5	49,4	69,5	72,7	114,0	122,1
2	44,6	50,2	69,5	72,7	114,1	122,9
3	44,8	50,2	69,5	72,7	114,3	122,9
4	44,6	50,2	69,5	72,7	114,1	122,9
5	44,4	49,4	69,5	72,7	113,9	122,1
6	44,7	50,2	69,5	72,7	114,2	122,9
7	44,4	49,4	69,5	72,7	113,9	122,1
8	44,5	49,4	69,5	72,7	114,0	122,1
9	44,6	50,2	69,5	72,7	114,1	123,1
10	44,5	49,4	69,5	72,7	114,0	122,3
11	44,6	50,1	69,5	72,7	114,1	123,0
12	44,5	50,1	69,5	72,7	114,0	123,0
13	44,5	49,5	69,5	72,7	114,0	122,4
14	44,6	50,1	69,5	72,7	114,1	123,0
15	44,5	49,5	69,5	72,7	114,0	122,4
16	44,6	50,1	69,5	72,7	114,1	123,0
17	44,4	49,6	69,8	72,8	114,2	122,4
18	44,5	49,4	69,5	72,6	114,0	122,0
19	44,8	50,0	69,2	72,4	114,0	122,4
20	45,1	50,5	68,9	72,2	114,0	122,7
21	45,3	51,1	68,7	72,0	114,0	123,1
22	45,3	50,8	68,4	71,8	113,7	122,6
23	45,6	51,4	68,1	71,6	113,7	123,0
24	45,6	51,2	67,8	71,4	113,4	122,6
25	45,8	51,0	67,5	71,2	113,3	122,2
26	45,7	51,3	67,2	71,0	112,9	122,3
27	45,5	50,9	66,9	70,8	112,4	121,7
28	45,5	51,2	66,6	70,6	112,1	121,8
29	45,3	51,4	66,4	70,4	111,7	121,8
30	45,2	51,1	66,1	70,2	111,3	121,3
31	45,1	51,3	65,8	70,0	110,9	121,3
32	45,0	50,9	65,5	69,8	110,5	120,7
33	45,0	51,2	65,2	69,6	110,2	120,8

Tabelle A.10: Reihen T zur Ermittlung von Einzelkosten: Ones-Kosten auf dem Datenbus

i	**CPU Read** $T_{cpu,addr,h,r}$	**CPU Write** $T_{cpu,addr,h,w}$	**Mem Read** $T_{mem,addr,h,r}$	**Mem Write** $T_{mem,addr,h,w}$	$\sum$ **Read**	$\sum$ **Write**
1	45,2	50,8	71,8	74,9	117,0	125,7
2	45,2	50,9	71,6	74,3	116,8	125,2
3	45,2	50,9	71,7	74,4	116,9	125,3
4	45,2	50,9	71,7	74,3	116,9	125,2
5	45,2	50,9	71,6	74,3	116,8	125,2
6	45,2	50,9	70,9	73,3	116,1	124,2
7	45,2	50,8	70,9	73,3	116,1	124,1
8	45,2	50,8	70,9	73,3	116,1	124,1
9	45,2	50,8	71,9	74,5	117,1	125,3
10	45,3	50,9	71,8	74,4	117,1	125,3
11	45,2	50,8	71,8	74,5	117,0	125,3
12	45,2	50,8	71,6	74,5	116,8	125,3
13	45,2	50,8	71,6	74,6	116,8	125,4
14	45,3	50,8	71,7	74,7	117,0	125,5
15	45,3	50,8	71,6	74,7	116,9	125,5
16	45,5	51,2	71,7	74,7	117,2	125,9
17	45,3	50,8	72,1	75,2	117,4	126,0
18	44,6	49,9	70,0	72,8	114,6	122,7
19	45,3	50,7	71,9	75,3	117,2	126,0
20	45,9	51,6	72,4	75,9	118,3	127,5
21	46,4	52,3	72,6	75,3	119,0	127,6
22	47,0	53,0	72,3	75,6	119,3	128,6
23	47,5	53,7	73,6	77,2	121,1	130,9
24	48,1	54,5	74,2	78,0	122,3	132,5
25	48,6	55,1	74,2	77,8	122,8	132,9
26	49,1	55,9	74,6	78,4	123,7	134,3
27	49,8	56,6	76,2	80,4	126,0	137,0
28	50,3	57,4	76,7	80,9	127,0	138,3
29	50,9	58,1	76,8	81,0	127,7	139,1
30	51,3	58,7	76,8	80,9	128,1	139,6
31	51,9	59,3	76,9	80,9	128,8	140,2
32	52,5	60,3	77,1	80,9	129,6	141,2
33	53,0	60,9	77,0	80,8	130,0	141,7
34	53,8	61,8	77,0	80,7	130,8	142,5
35	54,2	62,4	77,3	81,0	131,5	143,4

Tabelle A.11: Reihen T zur Ermittlung von Einzelkosten: Hamming-Kosten auf dem Adressbus

i	**CPU Read** $T_{cpu,data,h,r}$	**CPU Write** $T_{cpu,data,h,w}$	**Mem Read** $T_{mem,data,h,r}$	**Mem Write** $T_{mem,data,h,w}$	$\sum$ **Read**	$\sum$ **Write**
1	44,6	50,8	73,1	75,4	117,7	126,2
2	44,6	50,8	73,1	75,4	117,6	126,2
3	44,5	50,7	73,1	75,4	117,5	126,1
4	44,6	50,9	73,1	75,4	117,6	126,3
5	44,7	50,9	73,1	75,4	117,7	126,3
6	44,5	50,6	73,1	75,4	117,6	126,0
7	44,7	50,8	73,1	75,4	117,7	126,2
8	44,6	50,8	73,1	75,4	117,7	126,2
9	44,6	50,7	73,1	75,3	117,6	126,0
10	44,6	50,7	73,1	75,3	117,7	126,0
11	44,6	50,7	73,1	75,3	117,6	125,9
12	44,6	50,8	73,1	75,3	117,7	126,0
13	44,6	50,8	73,1	75,3	117,7	126,0
14	44,6	50,7	73,1	75,3	117,6	125,9
15	44,6	50,8	73,1	75,3	117,7	126,0
16	44,6	50,8	73,1	75,3	117,6	126,0
17	44,7	49,9	72,9	75,3	117,6	125,2
18	44,6	50,8	73,1	75,3	117,7	126,1
19	44,5	51,6	73,2	75,4	117,7	127,0
20	44,3	52,5	73,4	75,5	117,7	128,0
21	44,2	53,3	73,5	75,6	117,7	128,9
22	44,2	54,3	73,6	75,6	117,8	129,9
23	44,1	54,8	73,8	75,7	117,8	130,5
24	44,1	55,6	73,9	75,8	118,0	131,4
25	44,0	56,3	74,1	75,9	118,0	132,2
26	44,0	57,1	74,2	75,9	118,2	133,0
27	44,1	57,8	74,4	76,0	118,5	133,8
28	44,1	58,4	74,5	76,1	118,6	134,5
29	44,2	59,2	74,6	76,2	118,8	135,4
30	44,3	59,9	74,8	76,2	119,0	136,1
31	44,3	60,6	74,9	76,3	119,2	136,9
32	44,4	61,2	75,1	76,4	119,4	137,6
33	44,4	61,9	75,2	76,5	119,6	138,4

Tabelle A.12: Reihen T zur Ermittlung von Einzelkosten: Hamming-Kosten auf dem Datenbus

Literaturverzeichnis

[ARM95a] Advanced RISC Machines Ltd.: *An Introduction to THUMB*, 1995

[ARM95b] Advanced RISC Machines Ltd.: *ARM7TDMI Data Sheet*, 1995

[ATM99a] Atmel Corporation: *Embedded RISC Microcontroller Core, ARM7TDMI*, 1999

[ATM99b] Atmel Corporation: *AT91 ARM Thumb Microcontrollers, AT91M40400*, 1999

[ATM99c] Atmel Corporation: *AT91EB01 Evaluation Board, User Manual*, 1998

[BHS98] L. Benini, R. Hodgson und P. Siegel: *System-level Power Estimation And Optimization*, International Symposium on Low Power Electronics and Design (ISLPED), 1998

[BM99] L. Benini und G. De Micheli: *System-Level Power Optimization: Techniques and Tools*, International Symposium on Low Power Electronics and Design (ISLPED), 1999

[CKL00] N. Chang, K. Kim und H. G. Lee: *Cycle-Accurate Energy Consumption Measurement and Analysis: Case Study of ARM7TDMI*, International Symposium on Low Power Electronics and Design (ISLPED), 2000

[COS01] COSINUS Computermesstechnik GmbH: *http://www.cosinus.de*, 2001

[DAMT00] M. Theokharidis: Diplomarbeit: *Energiemessung von ARM7TDMI Prozessor-Instruktionen*, Fachbereich Informatik der Universität Dortmund, 2000

[GRTI96] M. Greiner, G. Tinhofer: *Stochastik für Studienanfänger der Informatik*, Hanser Verlag, 1996

[KSW95] Kories, Schmidt-Walter: *Taschenbuch der Elektrotechnik*, Verlag Harri Deutsch, 1995

[MLC97] E. Musoll, T. Lang und J. Cortadella: *Exploiting the locality of memory references to reduce the address bus energy*, International Symposium on Low Power Electronics and Design (ISLPED), 1997

[SKWM01] S. Steinke, M. Knauer, L. Wehmeyer und Prof. Dr. P. Marwedel: *An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations*, Int. Workshop Power and Timing Modeling, Optimization and Simulation (PATMOS), 2001

[SS99] G. Sinevriotis und T. Stouraitis: *Power Analysis of the ARM 7 Embedded Microprocessor*, Int. Workshop Power and Timing Modeling, Optimization and Simulation (PATMOS), Oktober 1999

[STD94a] C.-L. Su, C.-Y. Tsui und A. M. Despain: *Low Power Architecture Design and Compilation Techniques for High-Performance Processors*, IEEE COMPCON, 15.02.1994

[STD94b] C.-L. Su, C.-Y. Tsui und A. M. Despain: *Saving Power in the Control Path of Embedded Processors*, IEEE Design & Test of Computers, Winter 1994

[SYN96] Synopsis: *Power Products Reference Manual, Version 3.5*, 1996

[TIW96] V. Tiwari: Dissertation: *Logic and System for Low Power Consumption*, Princeton University, November 1996

[TMW94] V. Tiwari, S. Malik und A. Wolfe: *Power Analysis of Embedded Software: A First Step towards Software Power Minimization*, IEEE Transactions on VLSI Systems, Dezember 1994