Präsentation der Disseration

Softwaremethoden zur Senkung der Verlustenergie in Microcontrollersystemen

Ralf Hildebrandt

Technische Universität Dresden

16. Juli 2007

Inhalt

Problemstellung

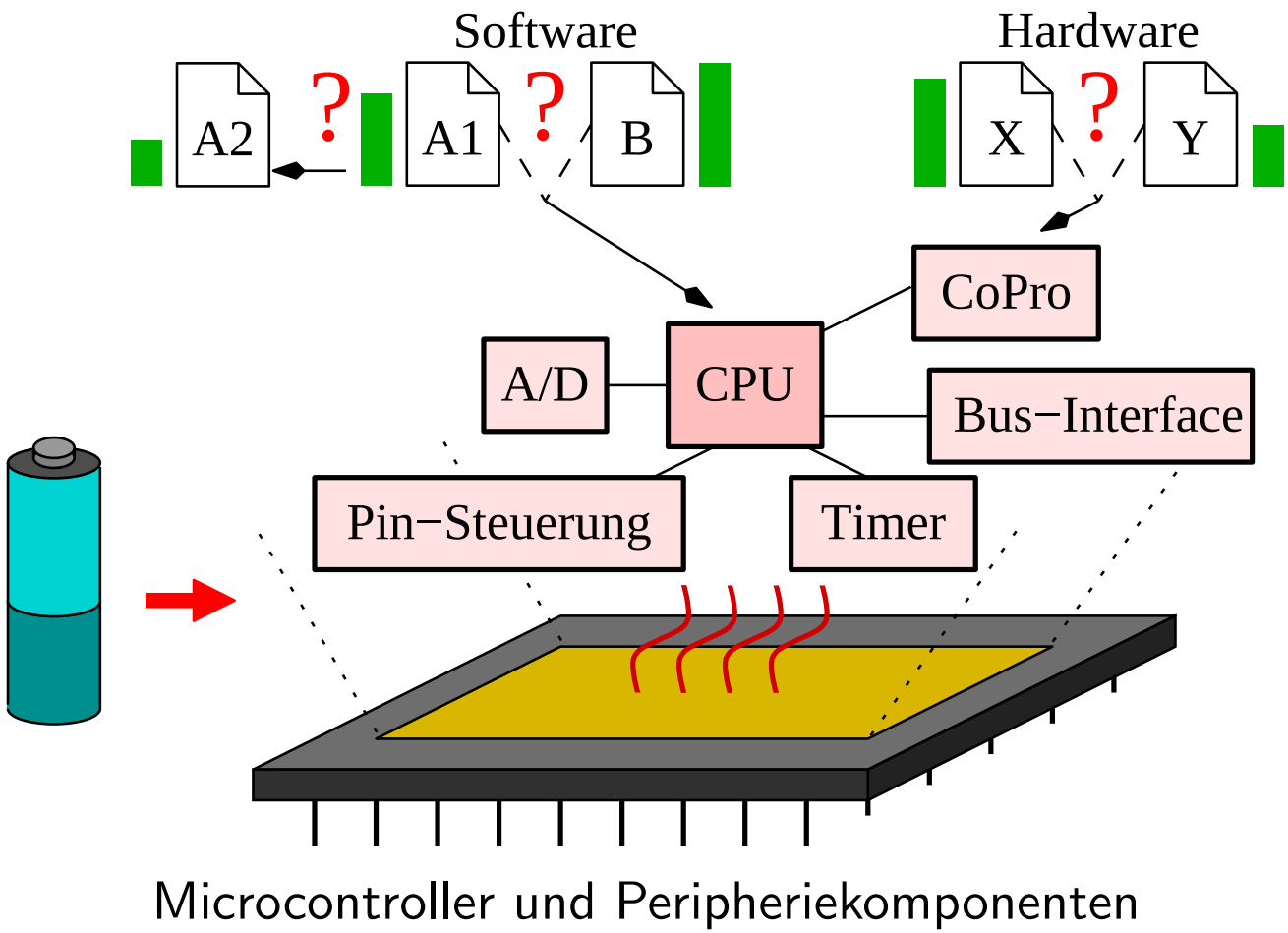
Verlustabschätzung mit VHDL

Darstellung von Softwarequelltext in der Hardwaresimulation

Erweiterte Softwareanalyse

Zusammenfassung

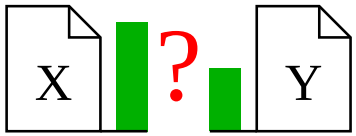
Problemstellung



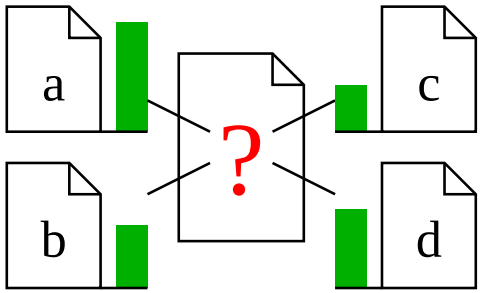
Microcontroller und Peripheriekomponenten

Entwurfsprozess zur Senkung der Verlustenergie

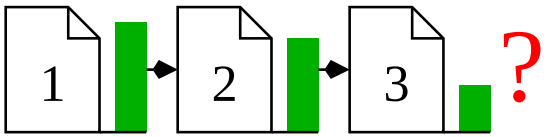
1. Verlustabschätzung
   - absolute Verlustwerte
   - relative Vergleiche zwischen Varianten



2. Lokalisierung der Quellen der größten Verluste
   - Hardwarekomponenten (Baugruppen, Elemente)
   - Softwareroutinen (Funktionen, Befehle)



3. Aufdecken der Ursachen für die Verluste
   - Je abstrakter die Beschreibungsform, desto stärker sind die Ursachen für Verluste verschleiert. (Hochsprache beim Softwareentwurf)



Problemstellung

Verlustabschätzung mit VHDL

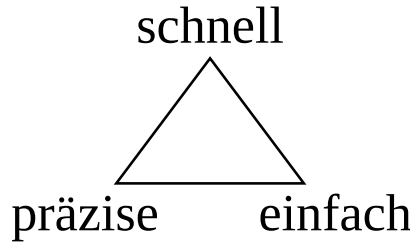
Darstellung von Softwarequelltext in der Hardwaresimulation

Erweiterte Softwareanalyse

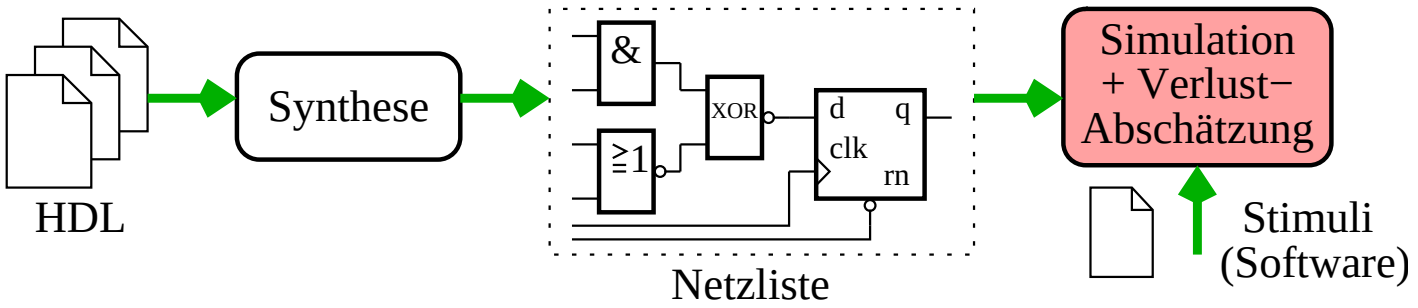
Zusammenfassung

Verlustabschätzung I

- Wünsche:

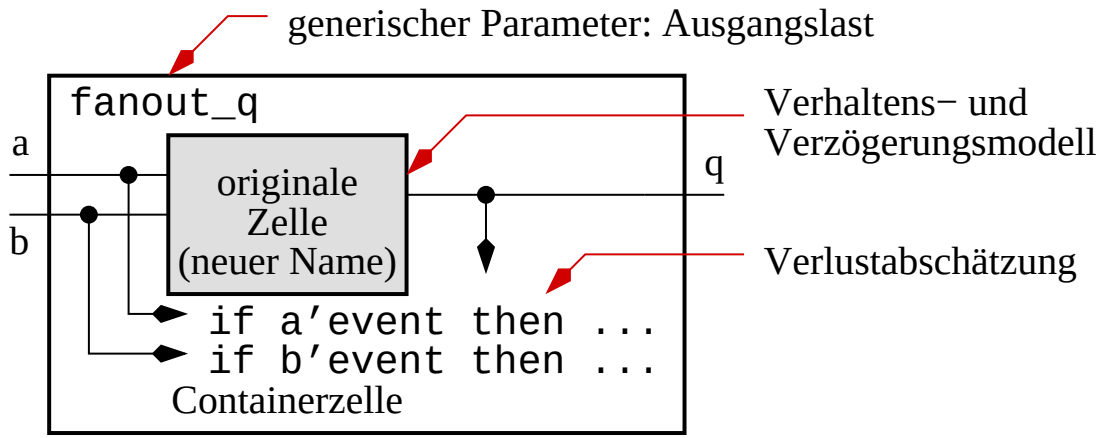


- eine Möglichkeit: Abschätzung basierend auf Hardwarebeschreibung
  - einsetzbar während des Entwurfs von Hardwarebaugruppen
  - Simulation des realen Verhaltens (keine Fehler abstrakter Modelle)
  - Untersuchung von Software



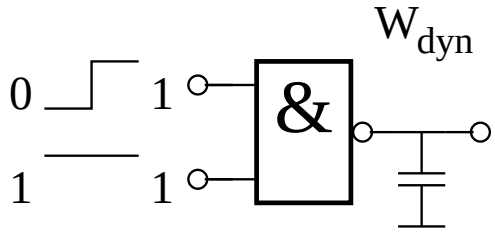
Verlustabschätzung II

- Integration der Abschätzung in die VHDL-Modelle der digitalen Zellen
  - keine externen Werkzeuge, Nutzung der normalen Simulationsumgebung
  - Verwendung der Stimuli vorhandener Testbenches
  - reduzierter Aufwand für Vorbereitung und Durchführung (das Setup)
  - einfache Zuordnung von Verlusten und logischem Verhalten



Methode der Verlustabschätzung

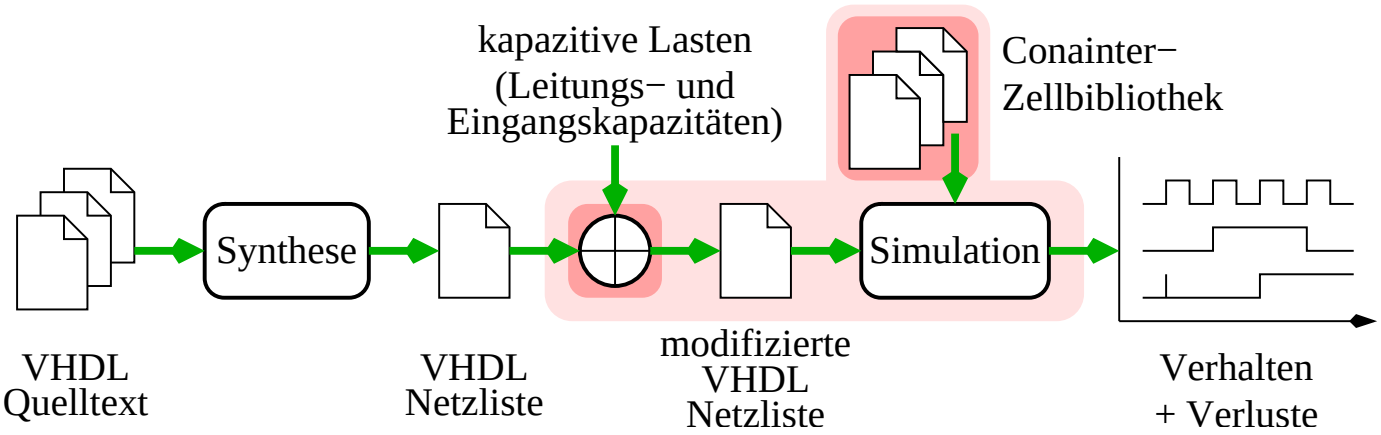
- Berechnung der dynamischen Verlustenergie W_{dyn} im Voraus (Spectre)
  - für alle Zellen einer Zellbibliothek
  - für alle Eingangssignalbelegungen und gespeicherte Zustände
  - für alle möglichen Einzelsignaländerungen
  - für ausgewählte Lasten (Fanout)



- Abspeichern der vorausberechneten Werte in einer Datenbasis (in den Containerzellen)
- Bestimmung der Verluste während der Hardwaresimulation aus der Datenbasis heraus

Verlustabschätzung mit VHDL

- Berücksichtigung der kapazitiven Lasten über modifizierte Netzliste



- relativer Fehler $\leq 21\%$ gegenber Spectre-Simulationen
- Simulationszeit 3fach größer als bei normaler Netzlistensimulation
- sehr geringer Aufwand für das Aufsetzen der Simulation

Problemstellung

Verlustabschätzung mit VHDL

Darstellung von Softwarequelltext in der Hardwaresimulation

Erweiterte Softwareanalyse

Zusammenfassung

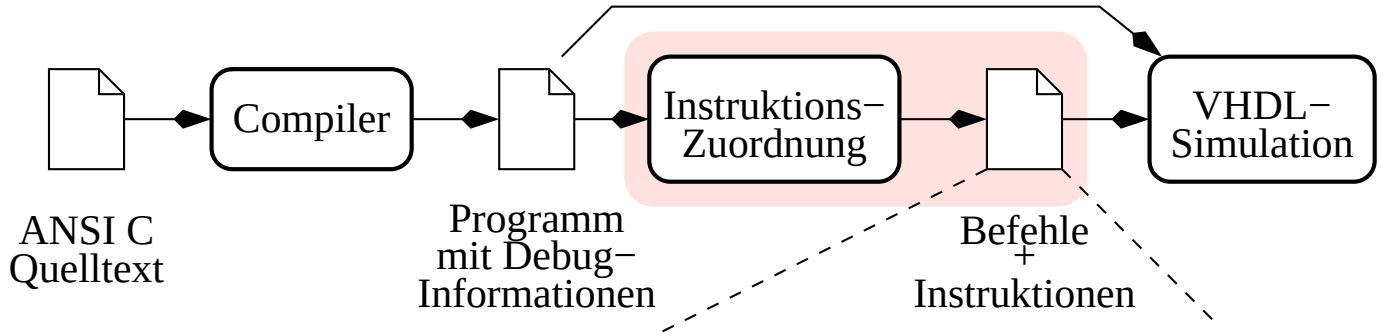
Verlustlokalisierung beim Softwareentwurf

- Grundsatz: längere Rechenzeit, konstanter Takt $\Rightarrow$ höhere Verlustenergie
- Verlustabschätzung bzw. Hardwaresimulation dennoch sinnvoll bei
  - Einsatz von Software und Hardware zur Problemlösung
  - Entwicklung von Hardwarebaugruppen (HDL-Simulation) / Software für noch nicht existierende Hardwarebaugruppen
- gemeinsamer / gleichzeitiger Hardware- und Softwareentwurf
  - Simulation innerhalb der HDL-Simulationsumgebung (kein Debugger)
  - Logische Zuordnung von HDL-Simulation und Software nur über Programmcounter und Disassembler möglich

Softwareanalyse in Hardwaresimulationsumgebung muss vereinfacht werden!

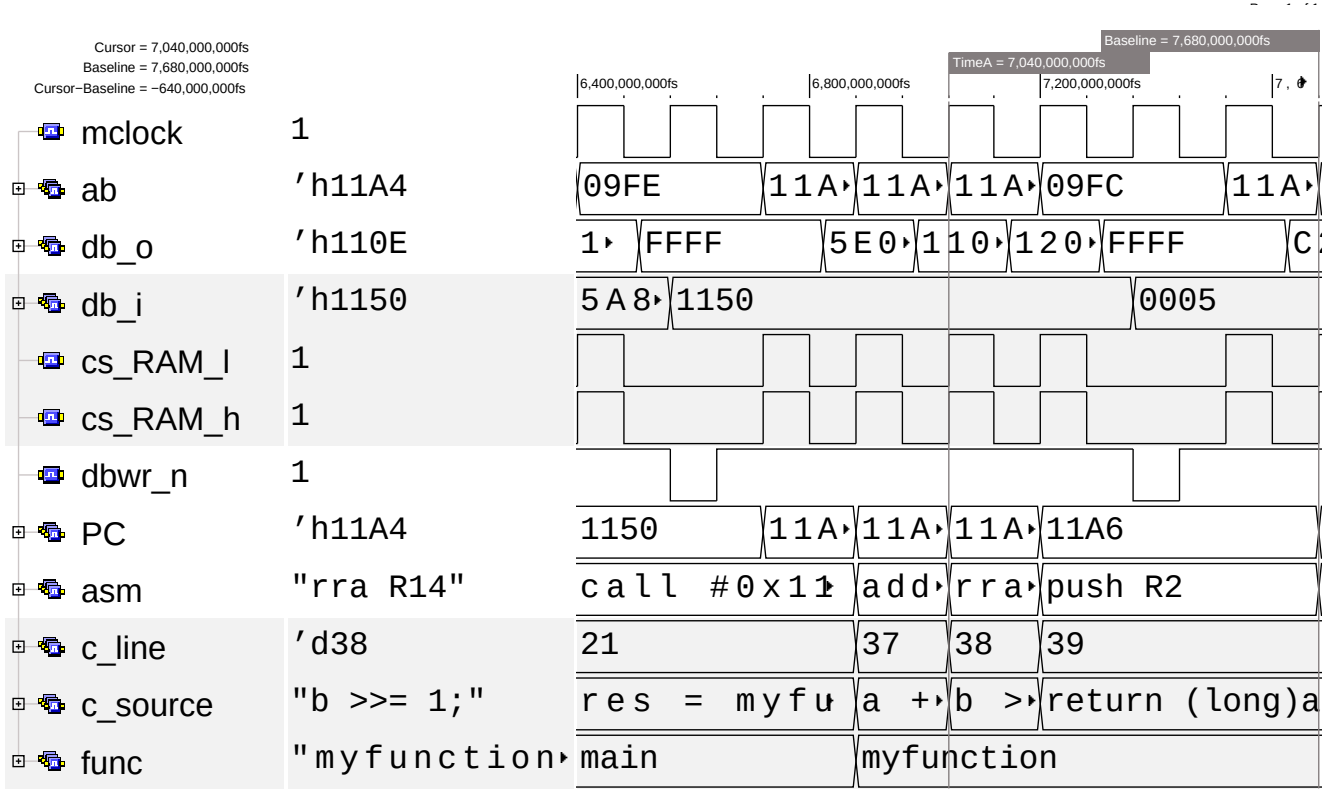
Softwareanalyse bei der Hardwaresimulation

Darstellung von Softwarequelltext in der Hardwaresimulationsumgebung:



0x2418	Adresse der Maschineninstruktion
mov R8,R4	disassemblierte Maschineninstruktion
75	Zeilennummer in der Hochsprache
b = (unsigned long)a;	Befehlszeile in der Hochsprache
main	aktuelle Funktion

Darstellung in der Hardwaresimulation



Printed on Wed Apr 05 06:58:58 MEST 2006

Printed by SimVision from Cadence Design Systems, Inc.

Problemstellung

Verlustabschätzung mit VHDL

Darstellung von Softwarequelltext in der Hardwaresimulation

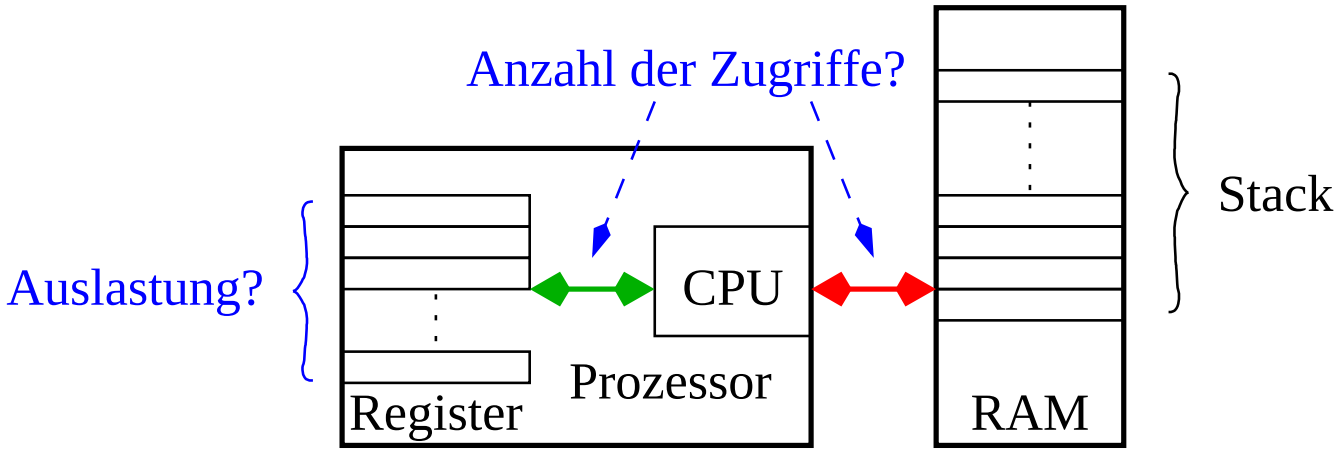
Erweiterte Softwareanalyse

Zusammenfassung

Bewertung von Software bezüglich Verlusten

- Untersuchung von Software mit Analysemethoden für Hardware ist aufwändig und zeitintensiv.
- Problemstellungen für hoch optimierte Software:
  - Welche Routinen sind am kostenintensivsten und welche Befehle darin sind am aufwändigsten?
  - Was sind die Gründe für die beobachtete Ausführungszeit?
- Laufzeitanalyse von Software (Profiling) liefert bisher zu grobkörnige Informationen. (z. B. Ausführungszeit von Funktionen)
- Schlussfolgerung: Profiling muss mehr und zielgerichtetere Informationen liefern!

Grundlagen feinkörniger Softwareoptimierung



- Voraussetzung: einfacher Algorithmus
- vorhandene Register effizient nutzen, RAM-Zugriffe minimieren
- Wo sind Variablen abgelegt und wie häufig wird auf sie zugegriffen?

Erweiterte Programmanalyse

1. detailliertes Profiling nicht nur auf Funktionsebene, sondern auf Befehlsebene: Lokalisierung der Kostenschwerpunkte
   - Taktanzahl + Anzahl der Befehlsausführungen
   - Instruktionsgröße (Programmspeicherbedarf)
   - Anzahl der Speicherzugriffe (RAM, ROM)
2. Variablenanalyse: Aufdeckung der Gründe für die Kosten
   - Typanalyse aller Variablen, Funktionsparameter und Funktionsresultate
   - Angabe der Speicherposition (Register oder Stack)
   - Zugriffszähler für Stackvariablen, Zugriffszähler für Register
3. automatisierte Vorschläge zur Optimierung (in Ansätzen möglich)

Damit sind feinkörnige Softwareoptimierungen ganz ohne Assemblerkenntnisse auf Hochsprachenebene möglich. Hauptschwerpunkt ist die effiziente Auslastung von Registern.

Erweiterte Programmanalyse

1. detailliertes Profiling nicht nur auf Funktionsebene, sondern auf Befehlsebene: Lokalisierung der Kostenschwerpunkte
   - Taktanzahl + Anzahl der Befehlsausführungen
   - Instruktionsgröße (Programmspeicherbedarf)
   - Anzahl der Speicherzugriffe (RAM, ROM)
2. Variablenanalyse: Aufdeckung der Gründe für die Kosten
   - Typanalyse aller Variablen, Funktionsparameter und Funktionsresultate
   - Angabe der Speicherposition (Register oder Stack)
   - Zugriffszähler für Stackvariablen, Zugriffszähler für Register
3. automatisierte Vorschläge zur Optimierung (in Ansätzen möglich)

Damit sind feinkörnige Softwareoptimierungen ganz ohne Assemblerkenntnisse auf Hochsprachenebene möglich. Hauptschwerpunkt ist die effiziente Auslastung von Registern.

Erweiterte Programmanalyse

1. detailliertes Profiling nicht nur auf Funktionsebene, sondern auf Befehlsebene: Lokalisierung der Kostenschwerpunkte
   - Taktanzahl + Anzahl der Befehlsausführungen
   - Instruktionsgröße (Programmspeicherbedarf)
   - Anzahl der Speicherzugriffe (RAM, ROM)
2. Variablenanalyse: Aufdeckung der Gründe für die Kosten
   - Typanalyse aller Variablen, Funktionsparameter und Funktionsresultate
   - Angabe der Speicherposition (Register oder Stack)
   - Zugriffszähler für Stackvariablen, Zugriffszähler für Register
3. automatisierte Vorschläge zur Optimierung (in Ansätzen möglich)

Damit sind feinkörnige Softwareoptimierungen ganz ohne Assemblerkenntnisse auf Hochsprachenebene möglich. Hauptschwerpunkt ist die effiziente Auslastung von Registern.

Erweiterte Programmanalyse

1. detailliertes Profiling nicht nur auf Funktionsebene, sondern auf Befehlsebene: Lokalisierung der Kostenschwerpunkte
   - Taktanzahl + Anzahl der Befehlsausführungen
   - Instruktionsgröße (Programmspeicherbedarf)
   - Anzahl der Speicherzugriffe (RAM, ROM)
2. Variablenanalyse: Aufdeckung der Gründe für die Kosten
   - Typanalyse aller Variablen, Funktionsparameter und Funktionsresultate
   - Angabe der Speicherposition (Register oder Stack)
   - Zugriffszähler für Stackvariablen, Zugriffszähler für Register
3. automatisierte Vorschläge zur Optimierung (in Ansätzen möglich)

Damit sind feinkörnige Softwareoptimierungen ganz ohne Assemblerkenntnisse auf Hochsprachenebene möglich. Hauptschwerpunkt ist die effiziente Auslastung von Registern.

Erweiterte Programmanalyse

1. detailliertes Profiling nicht nur auf Funktionsebene, sondern auf Befehlsebene: Lokalisierung der Kostenschwerpunkte
   - Taktanzahl + Anzahl der Befehlsausführungen
   - Instruktionsgröße (Programmspeicherbedarf)
   - Anzahl der Speicherzugriffe (RAM, ROM)
2. Variablenanalyse: Aufdeckung der Gründe für die Kosten
   - Typanalyse aller Variablen, Funktionsparameter und Funktionsresultate
   - Angabe der Speicherposition (Register oder Stack)
   - Zugriffszähler für Stackvariablen, Zugriffszähler für Register
3. automatisierte Vorschläge zur Optimierung (in Ansätzen möglich)

Damit sind feinkörnige Softwareoptimierungen ganz ohne Assemblerkenntnisse auf Hochsprachenebene möglich. Hauptschwerpunkt ist die effiziente Auslastung von Registern.

Beispiel zur Optimierung mit Erweiterter Programmanalyse

```
int myfunction1(struct mystructtype mystruct) {
        int a,b,c;
        ...
        a = mystruct.element + 123;
        b = mystruct.element * mystruct.vector[5];
        c = b - mystruct.element;
```

```
int myfunction2(struct mystructtype *mystruct) { // Zeiger
        int a,b,c;                               // Registervariablen
        ...
        a = mystruct->element;                   // lokale Kopie
        b = a * mystruct->vector[5];
        c = b - a;
        a += 123;
```

Beispiel zur Optimierung mit Erweiterter Programmanalyse

```
int myfunction1(struct mystructtype mystruct) {
        int a,b,c;
        ...
        a = mystruct.element + 123;
        b = mystruct.element * mystruct.vector[5];
        c = b - mystruct.element;
```

```
int myfunction2(struct mystructtype *mystruct) { // Zeiger
        int a,b,c;                                 // Registervariablen
        ...
        a = mystruct->element;                     // lokale Kopie
        b = a * mystruct->vector[5];
        c = b - a;
        a += 123;
```

Zusammenfassung

1. Verlustabschätzung mit VHDL mit im Voraus berechneten Werten
   - schnell, ausreichend genau, einfaches Setup
2. Darstellung von Softwarequelltext in der Hardwaresimulation
   - erhebliche Vereinfachung der Softwareanalyse
3. Erweiterte Softwareanalyse
   - detaillierte Lokalisierung der Kostenschwerpunkte
   - ermöglicht Softwareoptimierungen auf Hochsprachenniveau ohne Assemblerkenntnisse

Vielen Dank für die Aufmerksamkeit