

# Echtzeitsysteme

Übungen zur Vorlesung

Analyse von Ausführungszeiten

**Simon Schuster    Peter Wägemann**

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)  
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)  
<https://www4.cs.fau.de>

09.11.2018



## 1 Rekapitulation: Worst-Case Execution Time

## 2 Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

## 3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT



## 1 Rekapitulation: Worst-Case Execution Time

## 2 Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

## 3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT



# Worst-Case Execution Time

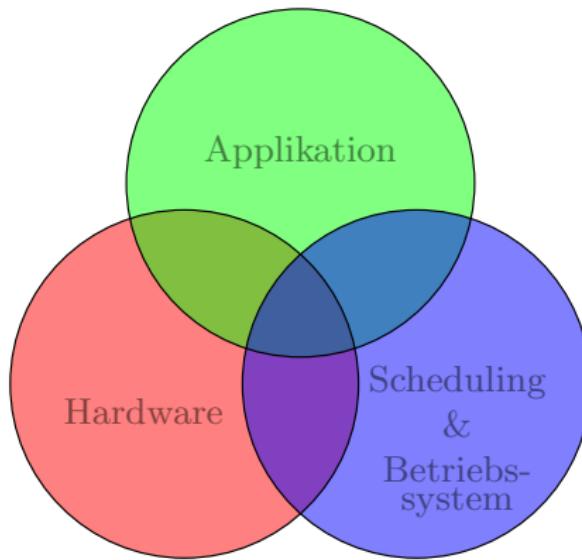


- Eine entscheidende Größe für:
  - Statische Ablaufplanung
  - Planbarkeitsanalyse
  - Übernahmeprüfung
  - ...

☞ Es geht um den **schlimmsten Fall** (engl. *worst case*)  
→ Obere Schranke für **alle** Fälle



# Wiederholung: Einflüsse auf die Ausführungszeit



- 1 **Applikation:** Eingabedaten, ...
- 2 **Hardware:** Caches, Pipelining, ...
- 3 **Scheduling:** Höherpriore Aufgaben, Interrupts, Overheads, ...



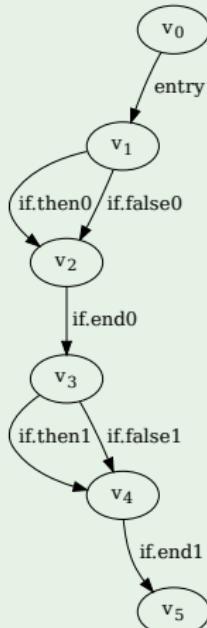
# WCET-Analyse – Flusssensitive Informationen

```
1 void func(int a) { // entry
2     if (a % 2) {
3         f(); // if.then0
4     }
5     ++a; // if.end0
6
7     if(a % 2) {
8         g(); // if.then1
9     }
10    ... // if.end1
11 }
```

Beispiel

- T-Graph aus Kontrollflussgraph abgeleitet
- Worst Case == maximaler Fluss durch T-Graph
- Nebenbedingungen des Flussproblems:
  - $\text{freq}(\text{entry}) = \text{freq}(\text{if.then0}) + \text{freq}(\text{if.false0})$
  - $\text{freq}(\text{if.then0}) + \text{freq}(\text{if.false0}) = \text{freq}(\text{if.end0})$
  - ...  
...
- Nebenbedingungen werden für Integer Linear Program (ILP) verwendet:  
Zielfunktion:

$$\max : \text{cost}(\text{entry}) \cdot \text{freq}(\text{entry}) + \text{cost}(\text{if.then0}) \cdot \text{freq}(\text{if.then0}) + \dots$$

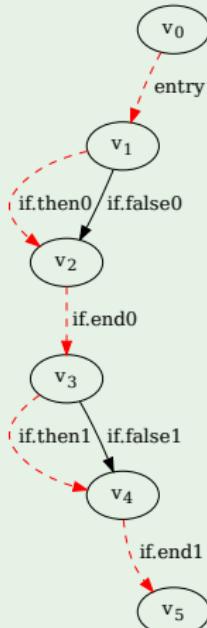


# WCET-Analyse – Flusssensitive Informationen

```
1 void func(int a) { // entry
2     if (a % 2) {
3         f(); // if.then0
4     }
5     ++a; // if.end0
6
7     if(a % 2) {
8         g(); // if.then1
9     }
10    ... // if.end1
11 }
```

Pessimistische Annahmen der IPET

- Für jeden Basis Block: WCET notwendig
- Schleifengrenzen notwendig
- Struktureller Ansatz: *nicht kontextsensitiv*
- Im Beispiel: *beide Pfade* aufgenommen  
⇒ **pessimistische Annahme**
- *Nachträgliche* Reduktion dieser Überabschätzung  
☞ **abstrakte Interpretation** ∼ VEZS





## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
    link    a6,#0          // 16 Zyklen  
    moveml #0x3020 ,sp@-   // 32 Zyklen  
    movel   a6@(8) ,a2      // 16 Zyklen  
    movel   a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [2]

- Ergebnis:  $e_{\text{getop}} = 80$  Zyklen
- Annahmen:
  - Obere Schranke für jede Instruktion
  - Obere Schranke der Sequenz durch Summation



## Äußerst pessimistisch und zum Teil falsch

- Falsch für Prozessoren mit Laufzeitanomalien
  - WCET der Sequenz > Summe der WCETs aller Instruktionen
- Pessimistisch für moderne Prozessoren
  - Pipeline, Cache, Branch Prediction, Prefetching, ... haben großen Anteil an der verfügbaren Rechenleistung heutiger Prozessoren
  - Blanke Summation einzelner WCETs ignoriert diese Maßnahmen



- ☞ Hardware-Analyse teilt sich in verschiedene Phasen
  - Aufteilung ist nicht dogmenhaft festgeschrieben
- **Integration** von Pfad- und Cache-Analyse
  - 1 Pipeline-Analyse
    - Wie lange dauert die Ausführung der Instruktionssequenz?
  - 2 Cache- und Pfad-Analyse sowie WCET-Berechnung
    - Cache-Analyse wird direkt in das Optimierungsproblem integriert
- **Separate** Pfad- und Cache-Analyse
  - 1 Cache-Analyse
    - Kategorisiert Speicherzugriffe mit Hilfe einer Datenflussanalyse
  - 2 Pipeline-Analyse
    - Ergebnisse der Cache-Analyse werden anschließend berücksichtigt
  - 3 Pfad-Analyse und WCET-Berechnung



1 Rekapitulation: Worst-Case Execution Time

2 Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT





**Cache:** ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:

Treffer (engl. *hit*), Daten/Instruktion sind im Cache  $\sim e_h$

Fehlschlag (engl. *miss*), Daten/Instruktion sind nicht im Cache  $\sim e_m$



**Hits** sind schneller als **Misses**:  $e_m \gg e_h$

→ Strafe liegt schnell bei > 100 Taktzyklen

- Eigenschaften von Caches mit Einfluss auf deren Analyse

Typ

- Cache für Instruktionen
- Cache für Daten
- kombinierter Cache für Instruktionen und Daten

Auslegung

- direkt abgebildet (engl. *direct mapped*)
- vollassoziativ (engl. *fully associative*)
- satz- oder mengenassoziativ (engl. *set associative*)

Seitenersetzungsstrategie

- engl. (*pseudo*) *least recently used*, (Pseudo-)LRU
- engl. (*pseudo*) *first in first out*, (Pseudo-)FIFO



# Ergebnisse der Cache-Analyse

- Wissen ob eine Instruktion / ein Datum im Cache ist, oder nicht:

must, die Instruktion ist **garantiert im Cache**

- man kann immer die schnellere Ausführungszeit  $e_h$  annehmen
- wird für die Vorhersage von Treffern verwendet

may, die Instruktion ist **vielleicht im Cache**

- ist dies nicht der Fall, muss man die Ausführungszeit  $e_m$  annehmen
- wird für die Vorhersage von Fehlschlägen verwendet

**persistent**, die Instruktion **verbleibt im Cache**

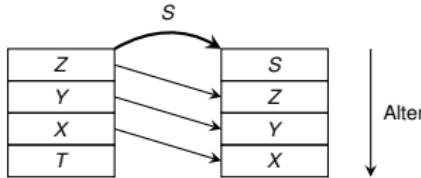
- erster Zugriff ist ein Fehlschlag, alle weiteren sind Treffer
- erster Zugriff:  $e_m$ , weitere Zugriffe:  $e_h$ 
  - ist besonders für Schleifen interessant, die den Cache „füllen“



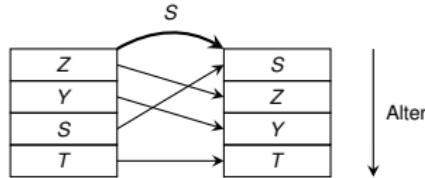
# Beispiel: LRU-Cache, 4-fach assoziativ

LRU = „least recently used“ – Das älteste Element fliegt raus!

Cache Miss



Cache Hit



- Caches werden häufig in **Sätze** (engl. *cache set*) unterteilt
  - Ein *n*-fach assoziativer Cache besitzt pro Satz *n* Cache-Blöcke
  - Aufnahme von *n* konkurrierende Speicherstellen pro Satz möglich
  - Inhalt und Verwaltungsinformation (bei LRU das Alter des Blocks) werden sowohl bei Treffern als auch bei Fehlschlägen aktualisiert
- Konkrete Semantik des Caches



must-Analyse und may-Analyse approximieren diese konkrete Semantik:

must Obergrenze des Alters  $\leadsto$  Unterapproximation des Inhalts

- Obergrenze  $\leq$  Assoziativität  $\leadsto$  garantiert im Cache

may Untergrenze des Alters  $\leadsto$  Überapproximation des Inhalts

- Untergrenze  $>$  Assoziativität  $\leadsto$  garantiert nicht im Cache

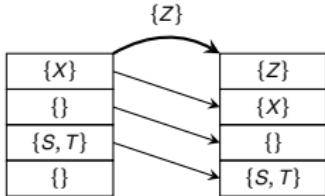


# Beispiel: LRU-Cache, Zugriff auf eine Speicherstelle

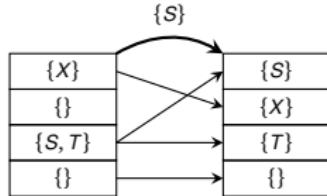
- Annäherung des Cache-Verhaltens durch must- und may-Approximation:  
Aktualisierung von Inhalt und Verwaltungsinformation

must-  
Approximation

Potential Cache Miss

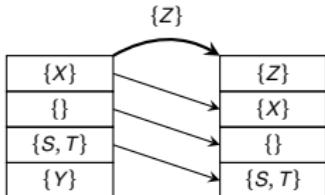


Definitive Cache Hit

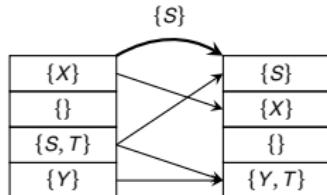


may-  
Approximation

Definitive Cache Miss

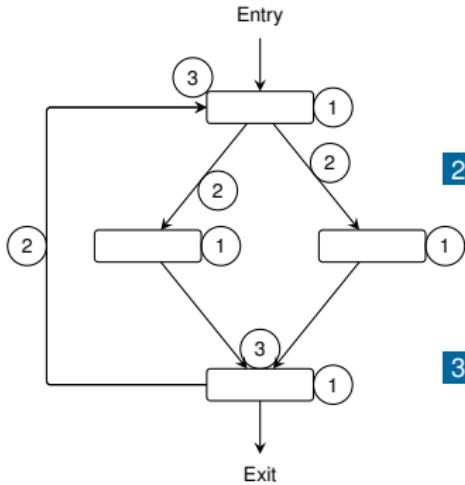


Potential Cache Hit



# Wie funktioniert nun die Cache-Analyse?

- Die Analyse ist eine Datenflussanalysen [1, Kapitel 8]



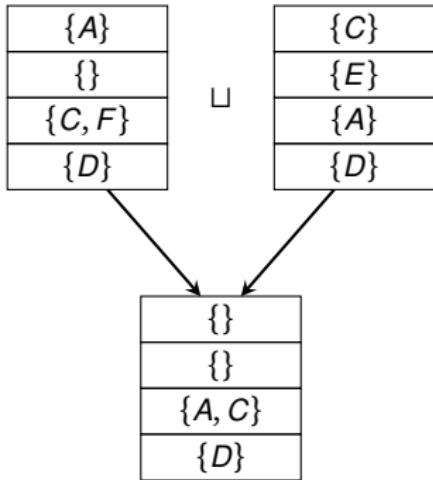
- 1 sammle Information in den Grundblöcken
  - Speicherzugriffe (s. Folie V/14)
  - man bestimmt die Übertragungsfunktion (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
  - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
- 3 fließt der Kontrollfluss wieder zusammen, wird auch die Information verschmolzen  
→ Verschmelzungsoperatoren

- Verschmelzungsoperatoren für must- und may-Analyse

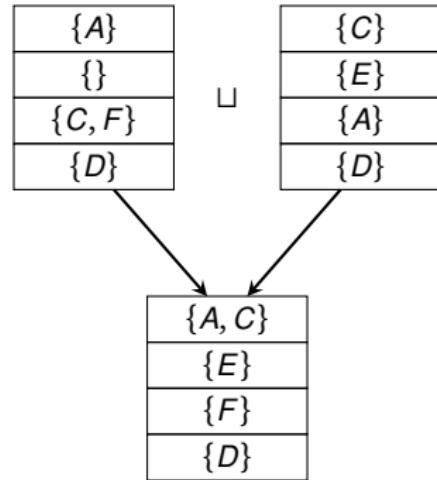


# Verschmelzungsoperatoren für must- und may-Analyse

must-Analyse



may-Analyse

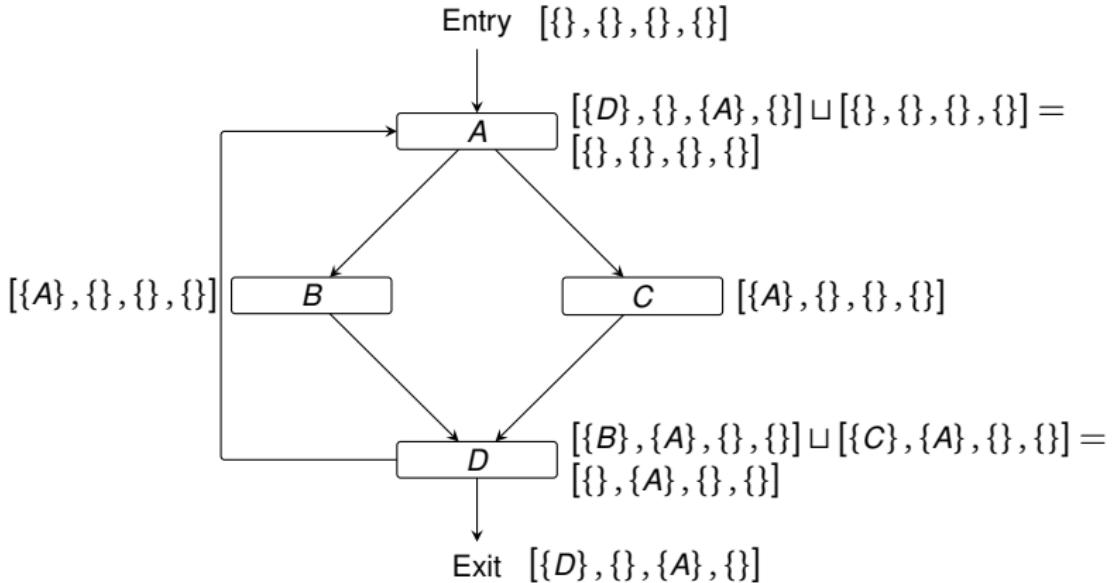


„Schnittmenge + max. Alter“

„Vereinigungsmenge + min. Alter“



# Beispiel: must-Analyse für LRU



☞ Hier ist leider keine Vorhersage von Treffern möglich 😞



# Praxisrelevante Cache-Implementierungen

- ☞ Cache-Analyse mithilfe einer Datenflussanalyse funktioniert für **mengenassoziative Caches mit LRU** sehr gut
  - Zugriffe auf unterschiedliche Cache-Zeilen beeinflussen sich nicht
  - Beispiel TriCore: 2-fach assoziativer LRU-Cache
- ⚠ Es kommen auch andere Strategien zum Einsatz:
  - Im Durchschnitt ähnliche Leistung wie LRU, **weniger vorhersagbar**
  - **Pseudo-LRU**
    - Cache-Zeilen werden als Blätter eines Baums verwaltet
    - must-Analyse **eingeschränkt brauchbar**, may-Analyse **unbrauchbar**
    - Beispiel: PowerPC 750/755
  - **Pseudo-Round-Robin**
    - 4-fach mengenassoziativer Cache mit **einem** 2-bit Ersetzungszähler
    - must-Analyse **kaum**, may-Analyse **überhaupt nicht brauchbar**
    - Beispiel: Motorola Coldfire 5307

☞ Keine belastbaren Aussagen zum STM32F411



1 Rekapitulation: Worst-Case Execution Time

2 Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

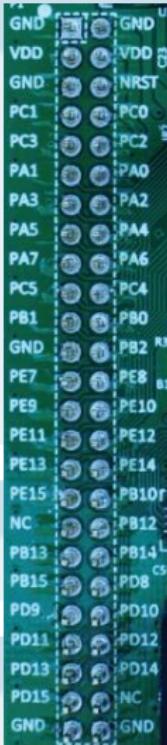
3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT



## General Purpose Input/Output

- Pins eines Mikrochips zur *freien Verwendung*
- Konfigurierbar als Ein-/Ausgang
- Teilweise pegelfest bis 5 V  
    ~ Mikrocontroller-Handbuch lesen ☺
- Zugriff über
  - spezielle Speicheradressen
  - Spezialanweisungen



## Ansteuerung

☞ void ezs\_gpio\_set(bool) //PD12

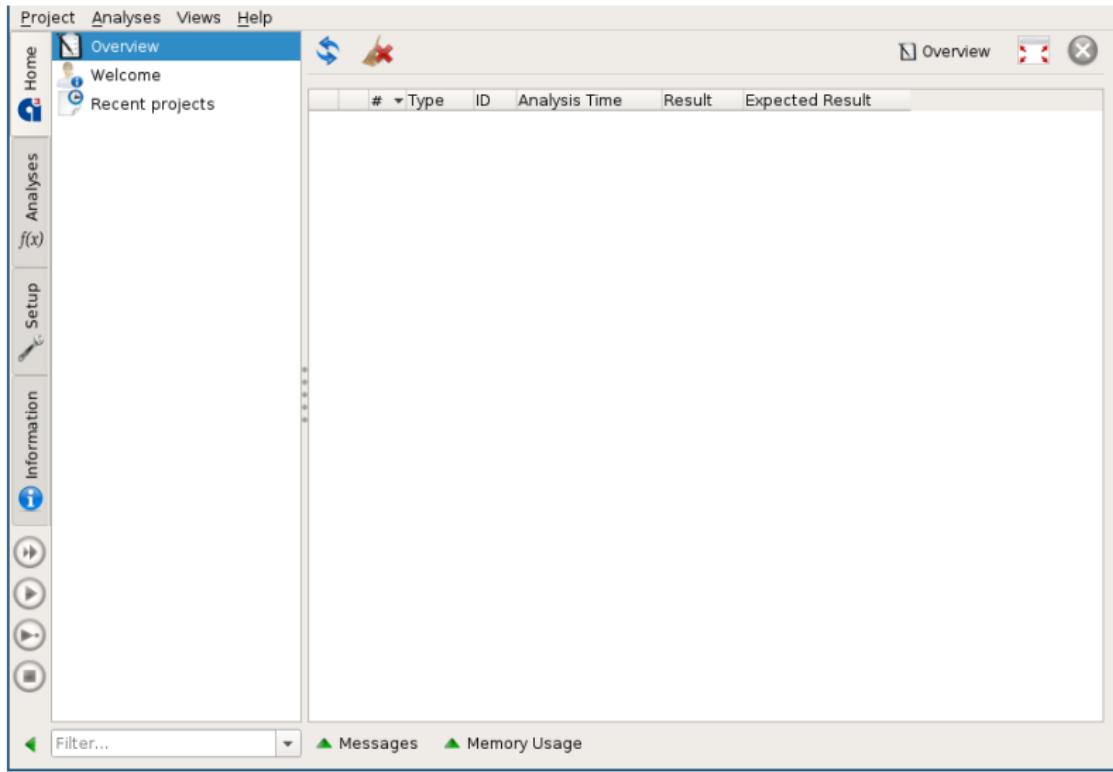
## Auswertung

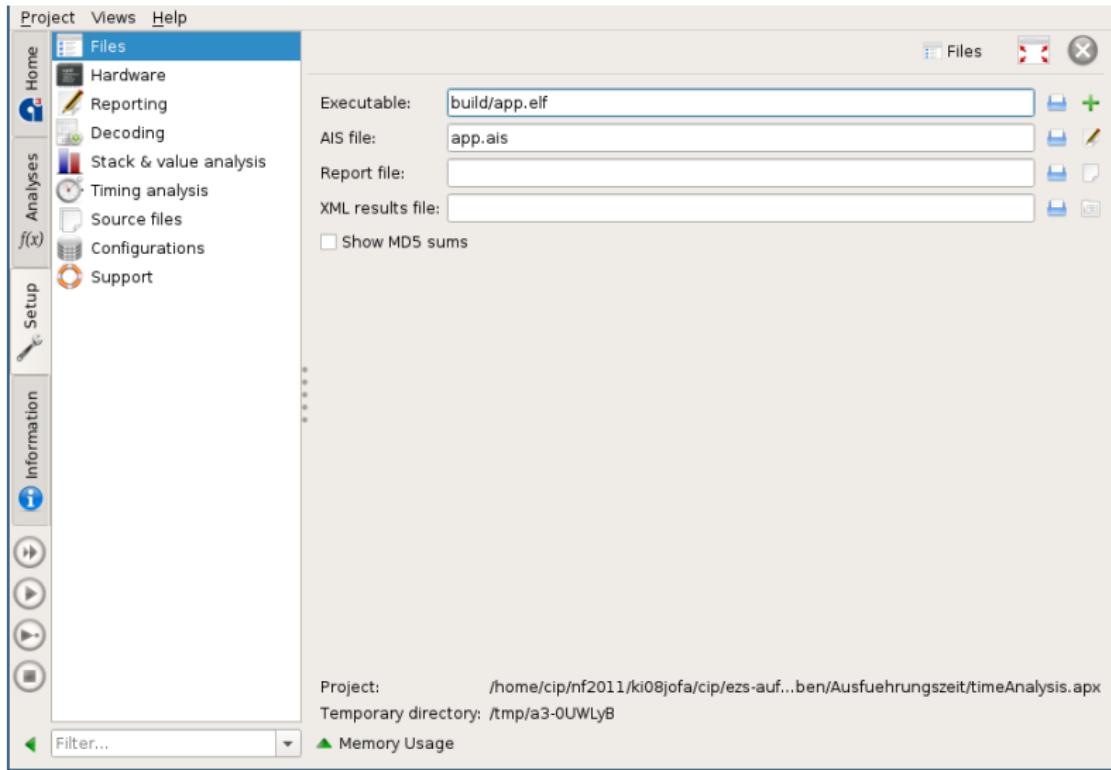
☞ Oszilloskop



# Startansicht

AbsInt aiT





# Projektdateien annotieren

AbsInt aiT

The screenshot shows the AbsInt aiT software interface. On the left is a vertical toolbar with icons for Home, Analyses (f(x)), Setup, and Information. The main window has a menu bar with Project, File, Edit, Views, and Help. A toolbar at the top includes icons for opening, saving, and zooming. The central area displays an AIS (Annotations for the Compiler) file named 'app.ais'. The code content is as follows:

```
1 # enter AIS annotations here or use the AIS Wizard from the context menu
2 # compiler
3 ais2 { compiler: "arm-gcc"; }
4 # clock rate
5 ais2 { clock: 84 MHz; }

7
8 # memory area 0x20000000 to 0x2001ffff (sram)
9 ais2 {
10     area 0x20000000 to 0x2001ffff {
11         readable: true; writable: true;
12         contains data;
13         access time: 1 cycles;
14     }
15 }
16
17 # context specification
18 ais2 {
19     mapping {
20         max length: inf;
21         max unroll: 2;
22         default unroll: 2;
23     }
24 }
25
```

At the bottom, there is a filter field labeled 'Filter...', a memory usage indicator 'Memory Usage', and a status message 'Line 1, column 1: app.ais'.



# Neue Analyse anlegen

AbsInt aiT

The screenshot shows the AbsInt aiT software interface. At the top, there is a menu bar with 'Project', 'Views', and 'Help'. Below the menu is a toolbar with icons for 'Home' (selected), 'Analyses' (highlighted in blue), 'f(x)', 'Setup', and 'Information'. To the right of the toolbar is a 'Create' button with a dropdown arrow. A tooltip for 'Create' says: 'You can also use the **Symbols** or **DWARF** view to create multiple analyses of the same type by selecting the analysis entries and using the **Create analyses** action from the toolbar or context menu.' Below the 'Analyses' icon, there are five analysis options listed:

- Control-Flow Visualizer**: Visualization of control-flow graph.
- aIT**: Safe WCET analysis. This item has a tooltip: 'Safe WCET analysis'.
- StackAnalyzer**: Stack usage analysis.
- ValueAnalyzer**: Program value analysis.
- TimingProfiler**: Profile the performance of your application.

At the bottom of the window, there is a 'Filter...' search bar and a 'Memory Usage' indicator.



# Neue Analyse anlegen

AbsInt aiT

The screenshot shows the AbsInt aiT software interface with the title "Neue Analyse anlegen". The main window displays a configuration dialog for a new analysis. The "ID" field is set to "aiT". The "Comment" field is empty. The "Configuration" field is set to "Default Configuration". The "Dependencies" field is empty. The "Analysis start" field is highlighted in red and has a green edit icon to its right. The "AIS file" field is empty. The "Report file" field is empty. The "XML report file" field is empty. The "HTML report file" field is empty. The "GDL output" field is empty. The "Expected result" field is empty and has a dropdown menu showing "cycles". The "Result" field is labeled "n/a". On the left side, there is a vertical toolbar with icons for Home, Analyses (highlighted), Setup, Information, and various navigation buttons. At the bottom, there is a toolbar with "Filter...", "Messages", and "Memory Usage" buttons.

# Neue Analyse anlegen

AbsInt aiT

The screenshot shows the AbsInt aiT tool interface. The main window title is "Analysis Start". A checkbox labeled "Regular expression" is checked. Below it is a table with two columns: "Address" and "Name". The table lists 786 functions. The first few entries are:

Address	Name
0x08009d8d	_adddf3
0x0800a6ed	_addsf3
0x0800b139	_aeabi_atexit
0x0800a5b1	_aeabi_cdcmpeq
0x0800a5b1	_aeabi_cdcmpeq
0x0800a5a1	_aeabi_cdcmpeq
0x0800a651	_aeabi_d2lz
0x0800a6a1	_aeabi_d2uz
0x08009d8d	_aeabi_dadd
0x0800a5c1	_aeabi_dcmpeq
0x0800a5fd	_aeabi_dcmpge
0x0800a611	_aeabi_dcmpgt
0x0800a5e9	_aeabi_dcmpeq
0x0800a5d5	_aeabi_dcmplt
0x0800a625	_aeabi_dcmprn
0x0800a345	_aeabi_ddiv
0x0800a0f1	_aeabi_dmul
0x08009d81	_aeabi_drsub
0x08009d89	_aeabi_dsub
0x0800a049	_aeabi_f2d
0x0800a6ed	_aeabi_fadd
0x0800aa65	_aeabi_fdiv
0x0800a0fd	_aeabi_fnanl

Below the table, it says "786 functions". At the bottom right of the dialog are "Cancel" and "OK" buttons. The "Analyses" tab is selected in the left sidebar.

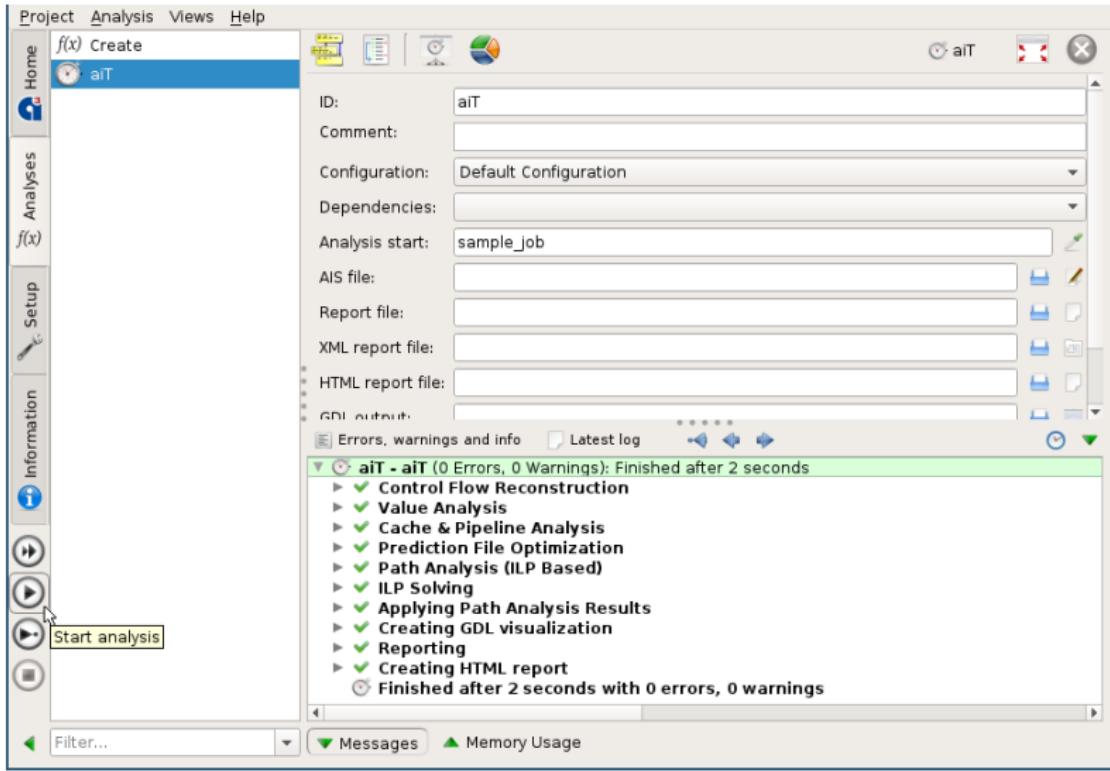


The screenshot shows the AbsInt aiT software interface with the following details:

- Project:** app.ais
- Views:** Project, Views, Help
- Left Sidebar:** Home, Analyses (Timing analysis is selected), f(x), Setup, Information.
- Timing Analysis Panel:**
  - Cache Analysis:** Instruction cache: Normal, Data cache: Normal.
  - Pipeline Analysis:** WCET computation mode: Global worst-case (applies only to aiT analyses), Threshold for applying default memory regions: 1024 kB,  Enable widening for cache states,  Skip timing analysis for main entry if additional starts are defined,  Detect timing anomalies (requires "Path analysis variant" to be "Prediction file based"),  Generate pipeline basic block statistics.
  - Path Analysis:** Default loop bound: 4, Default recursion bound: 4, Path analysis variant: ILP based, ILP solver: clpsolve.
- Bottom:** Filter... and Memory Usage.

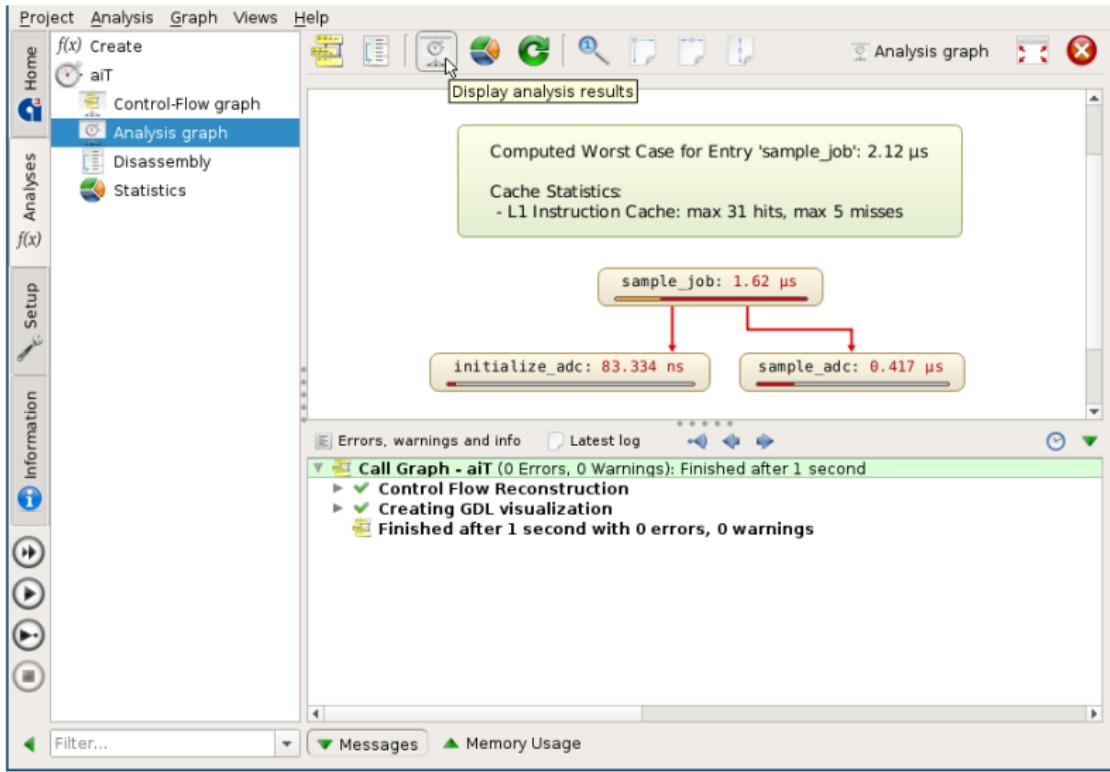
# Analyse starten

AbsInt aiT



# Analyse untersuchen

AbsInt aiT



# Analyse untersuchen

AbsInt aiT

The screenshot shows the AbsInt aiT software interface with the following details:

- Menu Bar:** Project, Analysis, Statistics, Views, Help.
- Toolbars:** f(x) Create, Home, aIT, Control-Flow graph, Analysis graph, Disassembly, Statistics (selected).
- Left Sidebar:** Analyses, Setup, Information.
- Central Area:**
  - WCET:** Submenu with options: WCET (context), Variable usage, Variable usage (context), Object size, Infeasibility, Sources.
  - Display analysis statistics:** Shows a table of analysis results:

Routine	Calls	Self [cycles]	Self [ns]	Self [ms]
sample_job	1	136	1619.05	1.61905
sample_adc	1	35	416.67	0.41667
initialize_adc	1	7	83.33	0.08333
  - Logs:** Errors, warnings and info, Latest log, Call Graph - aIT (0 Errors, 0 Warnings): Finished after 1 second (details: Control Flow Reconstruction, Creating GDL visualization, Finished after 1 second with 0 errors, 0 warnings).
- Bottom Navigation:** Filter..., Messages, Memory Usage.

- aiT gibt Zeitmessungen zunächst nur in Takten aus
- Taktrate angeben  $\leadsto$  tatsächliche Zeit

## Beispiele:

```
clock: 84MHz;  
clock: 83.95 .. 84.05 MHz;
```



- Manche Codestücke sind nicht analysierbar
  - Ausführungszeit annotieren
- ⚠ Natürlich nur sinnvoll, wenn WCET bereits bekannt

## Beispiel:

```
routine "even"{
    not analyzed;
    takes: 150 cycles;
}

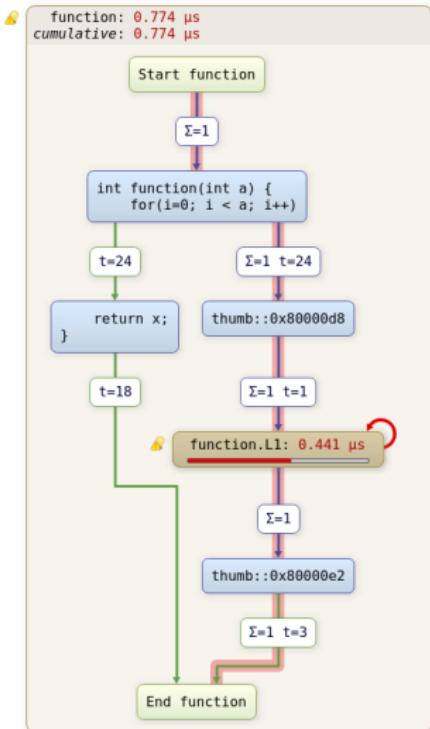
# exclude code as far as specified program points
instruction ProgramPoint snippet {
    continue at: ProgramPoint1 , PP2 , ... , PPn;
    not analyzed;
    takes: 10 cycles;
}
```



- Genaue Anzahl von Schleifendurchläufen zu bestimmen ist teuer
- ☞ aiT versucht standardmäßig nur zwei Durchläufe zu interpretieren
- Lohnt sich jedoch manchmal
- ☞ aiT mehr Freiheiten für die Analyse geben

## Beispiel:

```
loop "function.L1" mapping {  
    default unroll: 100;  
}
```



- Grenzen von Hand spezifizieren

Beispiele:

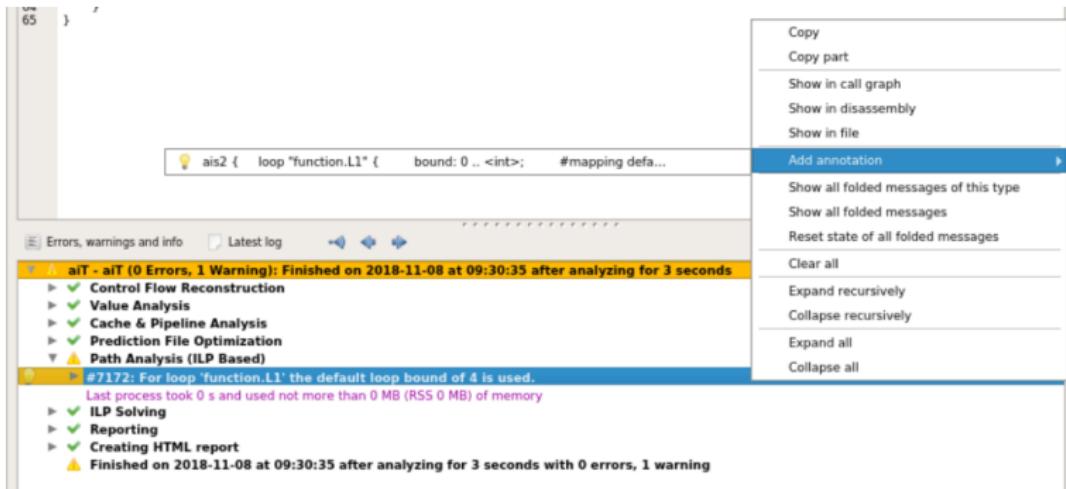
```
loop "function.L1" { bound: 0 .. 10 end; }
loop "function.L1" { bound: 10 begin; }
loop "function.L1" { bound: 10 .. inf end; }
loop "function.L1" { takes 20 ms; }
```

- Grenzen in Abhängigkeit von Registern spezifizieren

Beispiele:

```
loop "function.L1" {
    bound: 0 .. floor((reg("r0") - reg("r1")) / 4);
}
```





☞ Das Herausforderung ist nicht die Syntax, sondern das Finden  
(präziser) Schleifengrenzen



### Problem:

```
if (C) {  
    A(); // Vorbedingungen für Schleife in R()  
    R();  
} else {  
    B(); // Andere Vorbedingungen für R()  
    R();  
}
```

### Lösung:

```
// Annotations-"Variable" rmax definieren  
routine "A" { enter with: user("rmax") = 10; }  
routine "B" { enter with: user("rmax") = 20; }  
// "Variable" in Annotation nutzen  
loop "R.L1" { bound: 0 .. user("rmax"); }
```



aiT ist oft nicht in der Lage  
Rekursionen zu analysieren

→ Grenzen von Hand spezifizieren

## Problem:

```
int fib(int n) {  
    if (n <= 1)  
        return n;  
    return fib(n-1)  
        + fib(n-2);  
}
```

## Beispiele:

```
routine "fib" { recursion bound: 0 .. 10; }  
routine "fib" { recursion bound: 10; }  
routine "fib" { recursion bound: 5 .. 10; }
```

- Weitere Annotationen im Hilfe-Menü des aiT  
→ „AIS2 quick reference“



# Literaturverzeichnis

---

- [1] Steven S. Muchnick.

*Advanced compiler design and implementation.*

Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

- [2] Peter Puschner.

*Zeitanalyse von Echtzeitprogrammen.*

PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1993.

- [3] Reinhard Wilhelm.

Embedded systems.

[http:](#)

//react.cs.uni-sb.de/teaching/embedded-systems-10-11/lecture-notes.html,  
2010.

Lecture Notes.

