Echtzeitsysteme - Übungen

Analyse von Ausführungszeiten

Sommersemester 2024

Eva Dengler Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

Friedrich-Alexander-Universität
Technische Fakultät

Übersicht

Rekapitulation: Worst-Case Execution Time

Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT

Übersicht

Rekapitulation: Worst-Case Execution Time

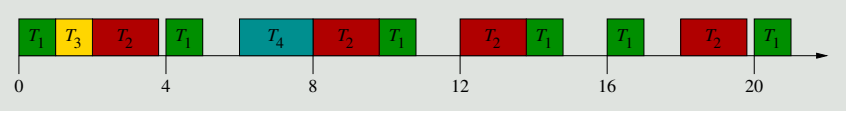
Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT

Worst-Case Execution Time

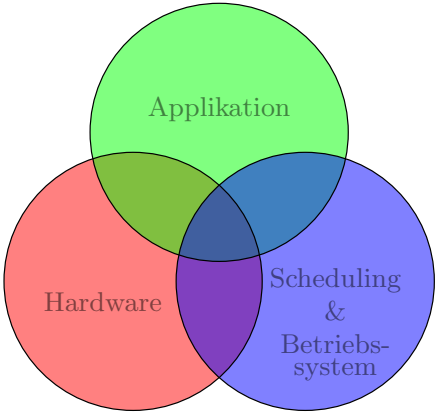


- Eine entscheidende Größe für:
  - Statische Ablaufplanung
  - Planbarkeitsanalyse
  - Übernahmeprüfung
  - …

☞ 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

Beispiel

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

- T-Graph aus Kontrollflussgraph abgeleitet
- Worst Case == maximaler Fluss durch T-Graph

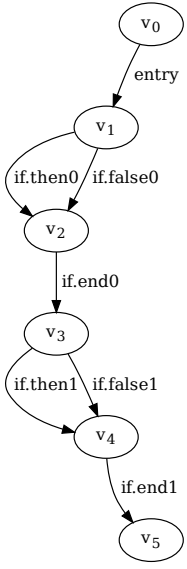
WCET-Analyse – Flusssensitive Informationen

Beispiel

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

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

$$max : cost(entry) \cdot freq(entry) + cost(if.then0) \cdot freq(if.then0) + \ldots$$

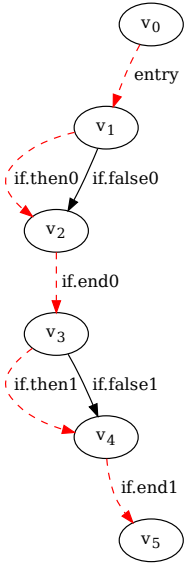


WCET-Analyse – Flusssensitive Informationen

Pessim. Annahmen der IPET

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

- 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



WCET eines Basis Blocks?

☞ 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 [3]

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

WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

- Falsch für mit Laufzeitanomalien behaftete Systeme
  - (intuitive) Annahmen über Worst-Case-Verhalten verletzt
  - Lokales Maximum führt nicht zwingend zu globalem Maximum
- 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

WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

Laufzeitanomalie: Beispiel

- Initialer Cachezustand zu Beginn der untersuchten Aufgabe?

WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

Laufzeitanomalie: Beispiel

- Initialer Cachezustand zu Beginn der untersuchten Aufgabe?
- Leerer Cache?

WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

Laufzeitanomalie: Beispiel

- Initialer Cachezustand zu Beginn der untersuchten Aufgabe?
- Leerer Cache?
  - ⚠ Abhängig von konkretem Cacheverhalten
  - ⚠ Falsche Annahme bei FIFO-Cache [1]

WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

- Falsch für mit Laufzeitanomalien behaftete Systeme
  - (intuitive) Annahmen über Worst-Case-Verhalten verletzt
  - Lokales Maximum führt nicht zwingend zu globalem Maximum
- 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

WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

Kein Pipelining:

WCET eines Basis Blocks?

☞ Grundproblem: Ausführungszyklen von Instruktionen zählen

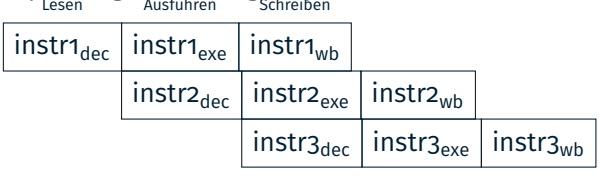
```
_getop:
  link     a6,#0          // 16 Zyklen
  moveml   #0x3020,sp@-   // 32 Zyklen
  movel    a6@(8),a2      // 16 Zyklen
  movel    a6@(12),d3     // 16 Zyklen
```

Quelle: Peter Puschner [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

Pipelining: (dreistufige Pipeline)



WCET eines Basis Blocks?

☞ Grundproblem: Ausführungszyklen von Instruktionen zählen

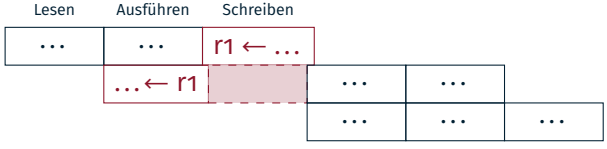
```
_getop:
  link     a6,#0          // 16 Zyklen
  moveml   #0x3020,sp@-   // 32 Zyklen
  movel    a6@(8),a2      // 16 Zyklen
  movel    a6@(12),d3     // 16 Zyklen
```

Quelle: Peter Puschner [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

Pipelining mit Hazards:



WCET eines Basis Blocks?

☞ 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 [3]

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

⚠ Äußerst pessimistisch und zum Teil falsch

- Falsch für mit Laufzeitanomalien behaftete Systeme
  - (intuitive) Annahmen über Worst-Case-Verhalten verletzt
  - Lokales Maximum führt nicht zwingend zu globalem Maximum
- 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

- ☞ Hardware-Analyse teilt sich in verschiedene Phasen
  - Aufteilung ist nicht dogmenhaft festgeschrieben

Hardware-Analyse

- ☞ 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

Hardware-Analyse

- ☞ 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

Übersicht

Rekapitulation: Worst-Case Execution Time

Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT

Grundlagen der Cache-Analyse [4, Kapitel 22]

- ☞ Cache: ein kleiner, schneller Zwischenspeicher
  - Zugriffszeiten variieren je nach Zustand des Caches enorm:

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

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

Grundlagen der Cache-Analyse [4, Kapitel 22]

☞ Cache: ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:

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

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

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

→ Strafe liegt schnell bei > 100 Taktzyklen

Grundlagen der Cache-Analyse [4, Kapitel 22]

☞ Cache: ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:
  - **Treffer** (engl. *hit*), Daten/Instruktion sind im Cache $\rightsquigarrow e_h$
  - **Fehlschlag** (engl. *miss*), Daten/Instruktion sind nicht im Cache $\rightsquigarrow e_m$

⚠ 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*)
  - **Zeilenersetzungsstrategie**
    - 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

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

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

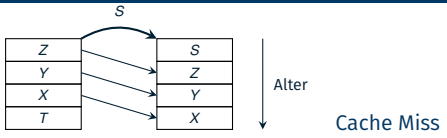
  may, die Instruktion ist vielleicht im Cache

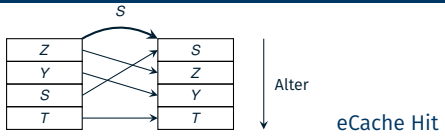
  → ist dies nicht der Fall, muss man die Ausführungszeit e_m annehmen
  - wird für die Vorhersage von Fehlschlägen verwendet

  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

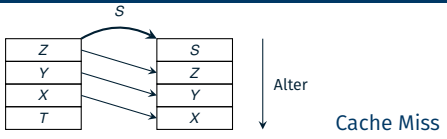


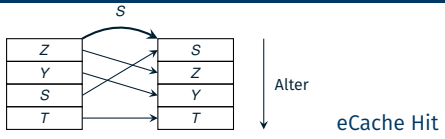


- Caches werden häufig in Sätze (engl. *cache set*) unterteilt
  - Ein *n*-fach assoziativer Cache besitzt pro Satz *n* Cache-Blöcke
  - Aufnahme von *n* konkurrierende Speicherstellen pro Satz möglich
  - Inhalt und Verwaltungsinformation (bei LRU das Alter des Blocks) werden sowohl bei Treffern als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

  → Konkrete Semantik des Caches

Beispiel: LRU-Cache, 4-fach assoziativ





- 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- und may-Analyse approximieren diese konkrete Semantik:

must Obergrenze des Alters $\rightsquigarrow$ Unterapproximation des Inhalts
- Obergrenze $\leq$ Assoziativität $\rightsquigarrow$ garantiert im Cache

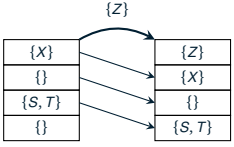
may Untergrenze des Alters $\rightsquigarrow$ Überapproximation des Inhalts
- Untergrenze $>$ Assoziativität $\rightsquigarrow$ garantiert nicht im Cache

Beispiel: LRU-Cache, Zugriff auf eine Speicherstelle

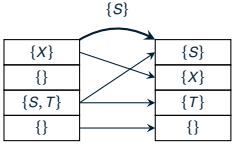
- Annäherung des Cache-Verhaltens durch must- und may-Approximation: Aktualisierung von Inhalt und Verwaltungsinformation

must-Approximation

Potential Cache Miss

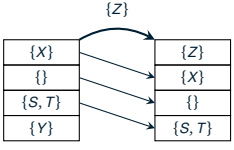


Definitive Cache Hit

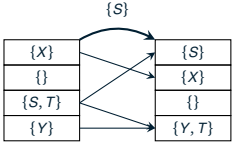


may-Approximation

Definitive Cache Miss



Potential Cache Hit

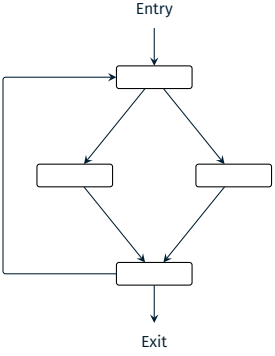


Wie funktioniert nun die Cache-Analyse?

- ☞ Die Analyse ist eine Datenflussanalysen [2, Kapitel 8]

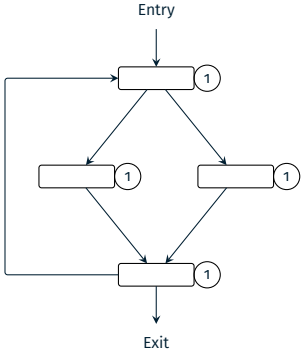
Wie funktioniert nun die Cache-Analyse?

☞ Die Analyse ist eine Datenflussanalysen [2, Kapitel 8]



Wie funktioniert nun die Cache-Analyse?

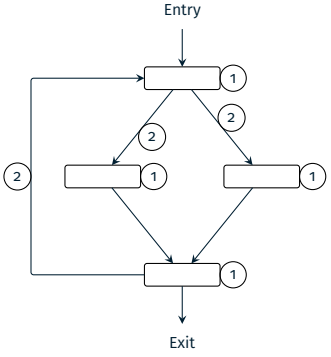
☞ Die Analyse ist eine Datenflussanalysen [2, Kapitel 8]



1. sammle Information in den Grundblöcken
   - Speicherzugriffe (s. Folie 13)
   - man bestimmt die Übertragungsfunktion (engl. *transfer function*) des Grundblocks

Wie funktioniert nun die Cache-Analyse?

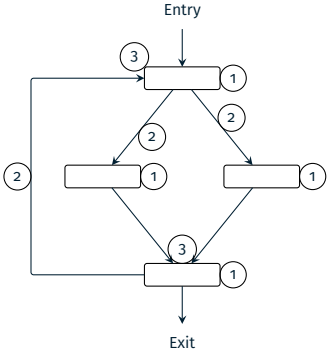
☞ Die Analyse ist eine Datenflussanalysen [2, Kapitel 8]



1. sammle Information in den Grundblöcken
   - Speicherzugriffe (s. Folie 13)
   - man bestimmt die Übertragungsfunktion (engl. *transfer function*) des Grundblocks
2. die Information wird über ausgehende Kanten weiterverteilt
   - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke

Wie funktioniert nun die Cache-Analyse?

☞ Die Analyse ist eine Datenflussanalysen [2, Kapitel 8]

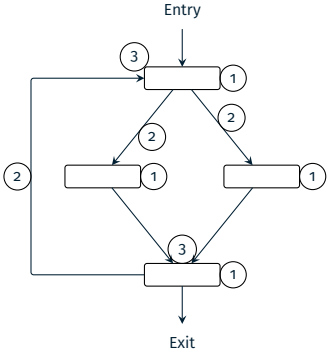


1. sammle Information in den Grundblöcken
   - Speicherzugriffe (s. Folie 13)
   - man bestimmt die Übertragungsfunktion (engl. *transfer function*) des Grundblocks
2. die Information wird über ausgehende Kanten weiterverteilt
   - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
3. fließt der Kontrollfluss wieder zusammen, wird auch die Information verschmolzen

   → Verschmelzungsoperatoren

Wie funktioniert nun die Cache-Analyse?

☞ Die Analyse ist eine Datenflussanalysen [2, Kapitel 8]



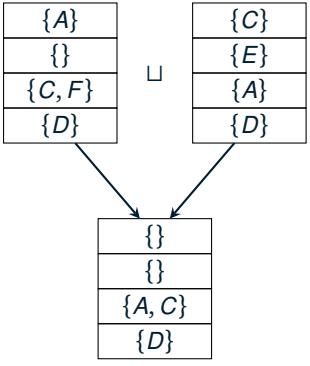
1. sammle Information in den Grundblöcken
   - Speicherzugriffe (s. Folie 13)
   - man bestimmt die Übertragungsfunktion (engl. *transfer function*) des Grundblocks
2. die Information wird über ausgehende Kanten weiterverteilt
   - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
3. fließt der Kontrollfluss wieder zusammen, wird auch die Information verschmolzen

   → Verschmelzungsoperatoren

☞ Verschmelzungsoperatoren für must- und may-Analyse

Verschmelzungsoperatoren für must- und may-Analyse

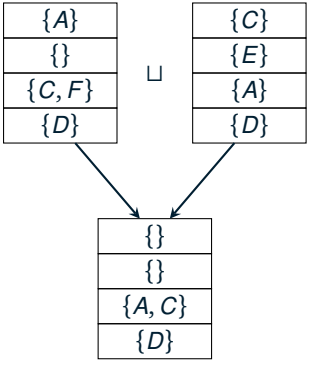
must-Analyse



„Schnittmenge + max. Alter“

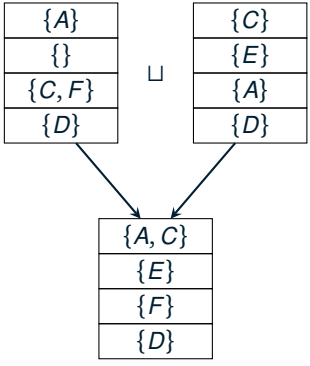
Verschmelzungsoperatoren für must- und may-Analyse

must-Analyse



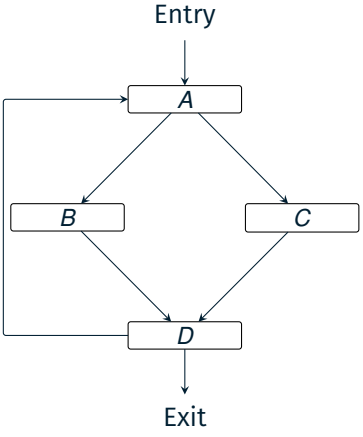
„Schnittmenge + max. Alter“

may-Analyse

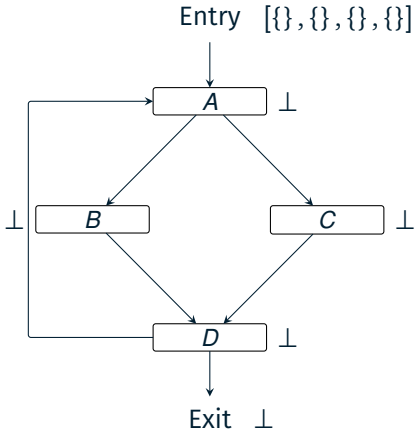


„Vereinigungsmenge + min. Alter“

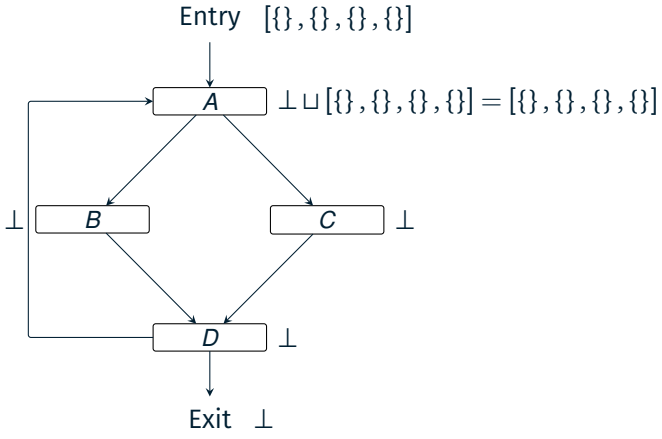
Beispiel: must-Analyse für LRU



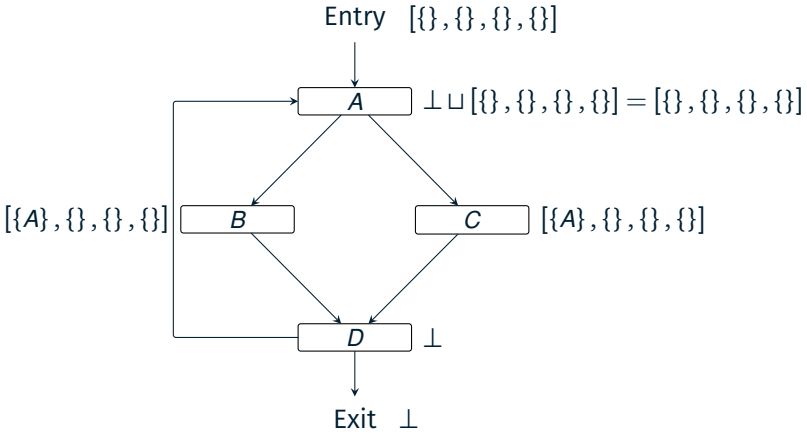
Beispiel: must-Analyse für LRU



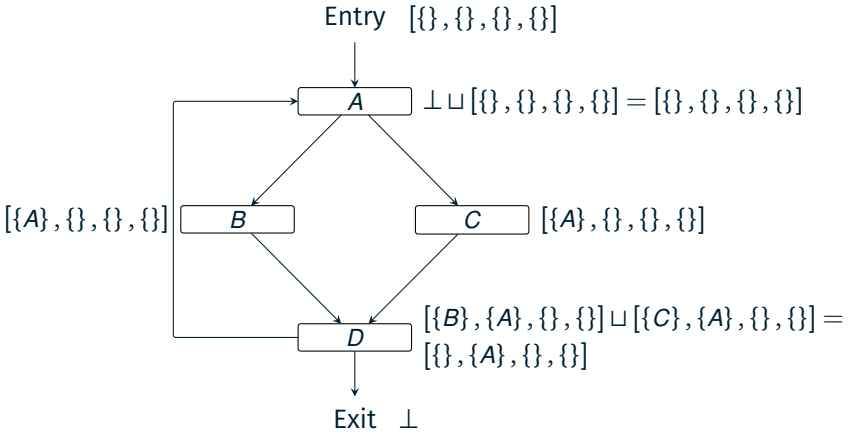
Beispiel: must-Analyse für LRU



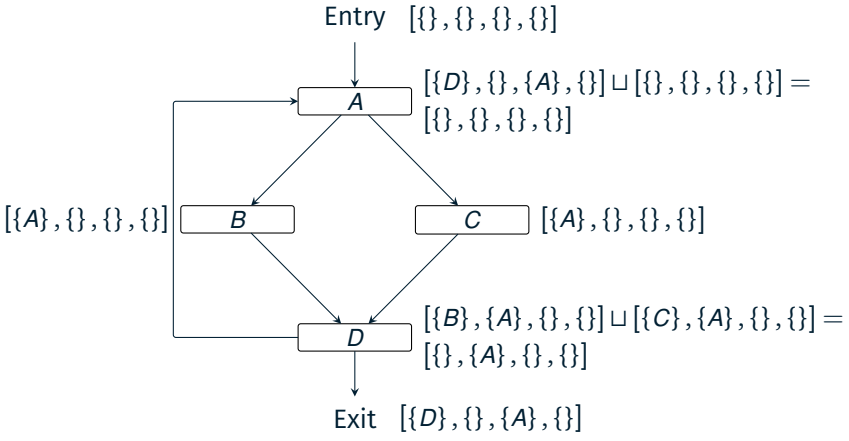
Beispiel: must-Analyse für LRU



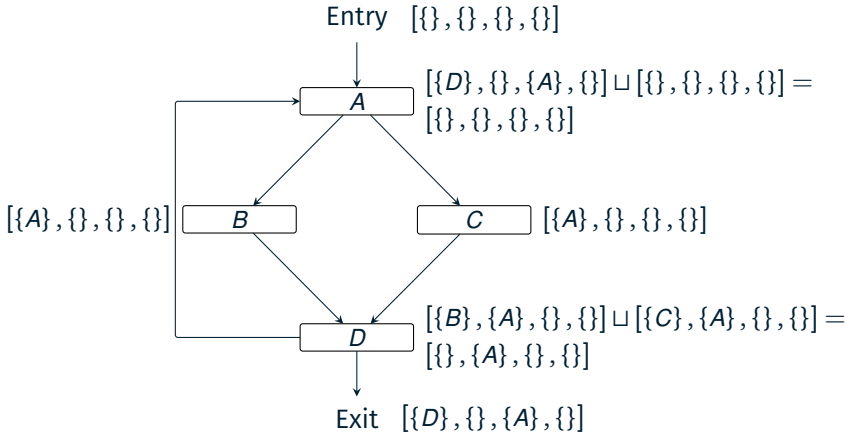
Beispiel: must-Analyse für LRU



Beispiel: must-Analyse für LRU



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

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

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 STM32F429

Übersicht

Rekapitulation: Worst-Case Execution Time

Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT

GPIO

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

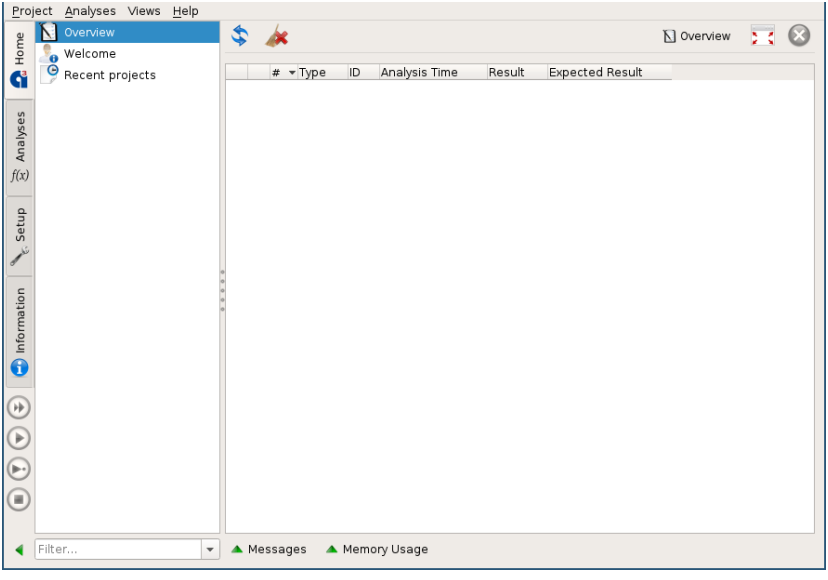
Ansteuerung

☞ `void ezs_gpio_set(bool) //PD12`

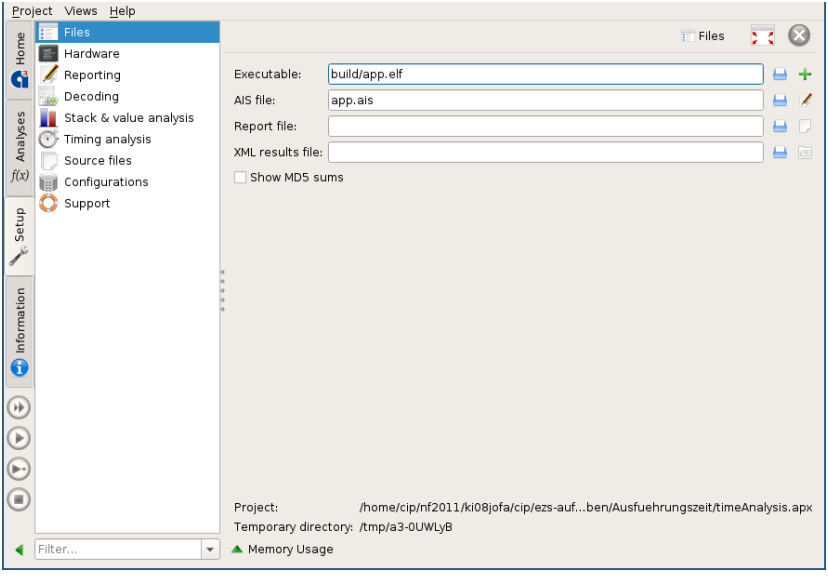
Auswertung

☞ Oszilloskop

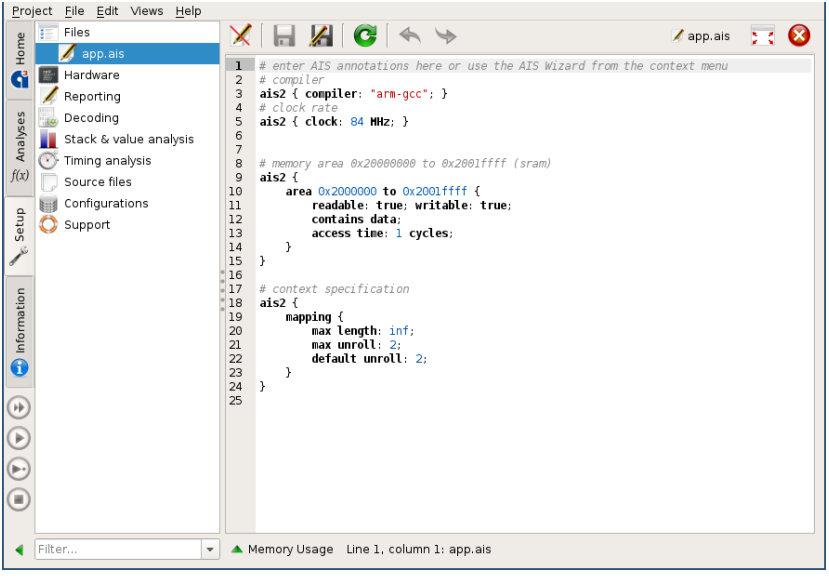
Startansicht



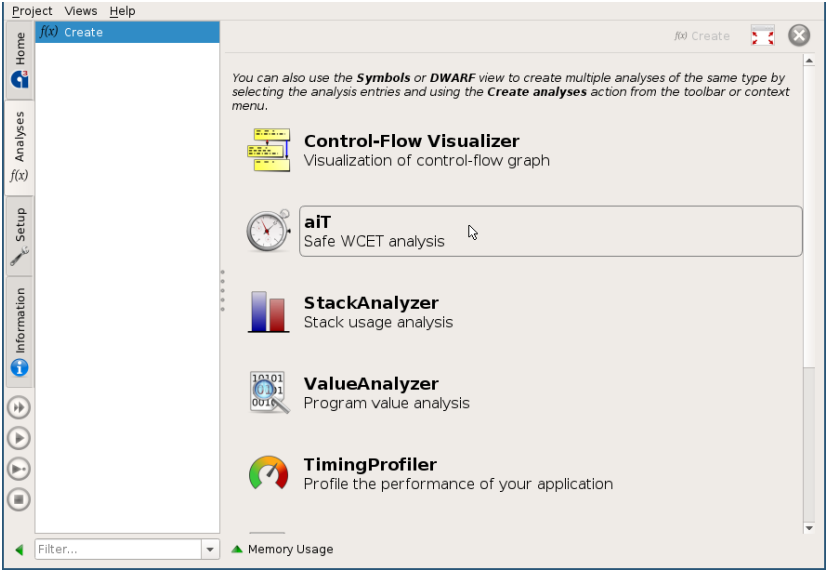
Projektdateien



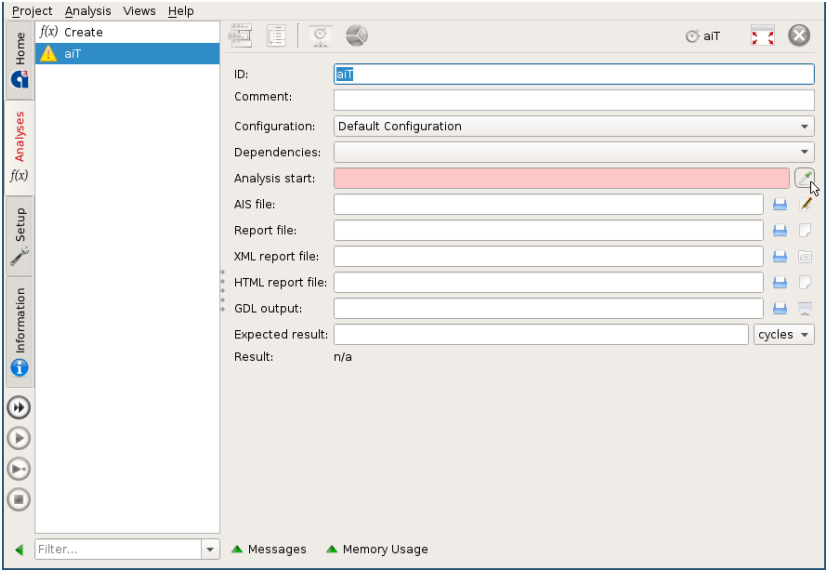
Projektdateien annotieren



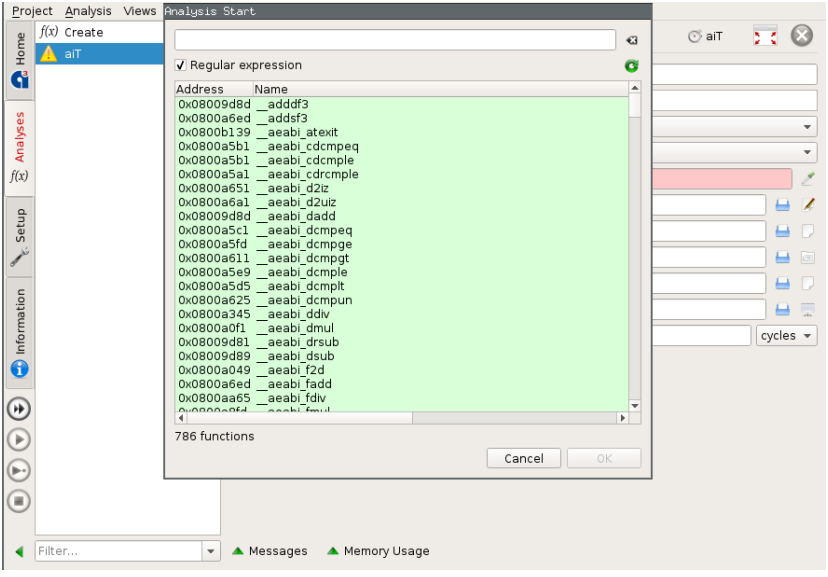
Neue Analyse anlegen



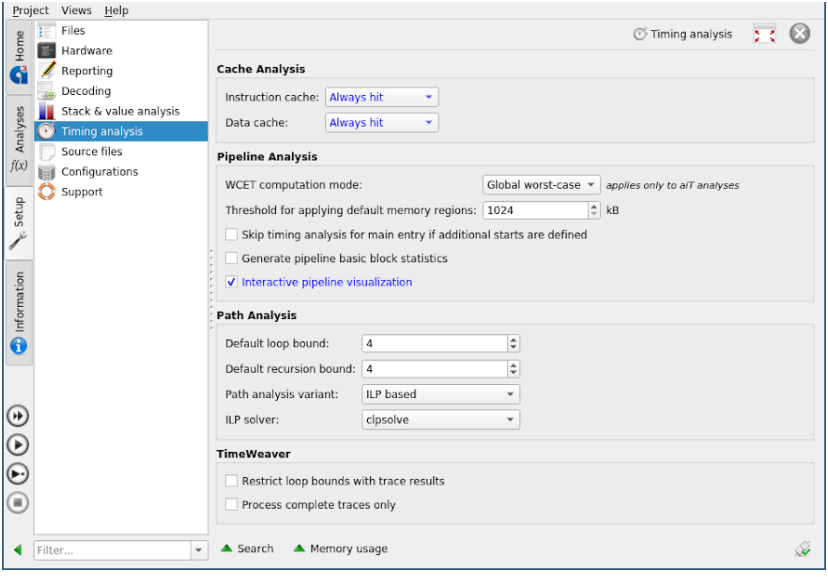
Neue Analyse anlegen



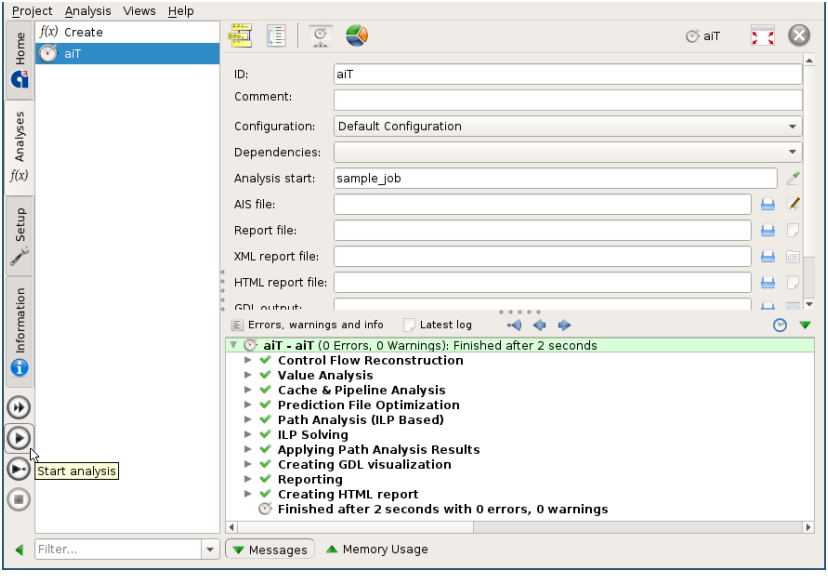
Neue Analyse anlegen



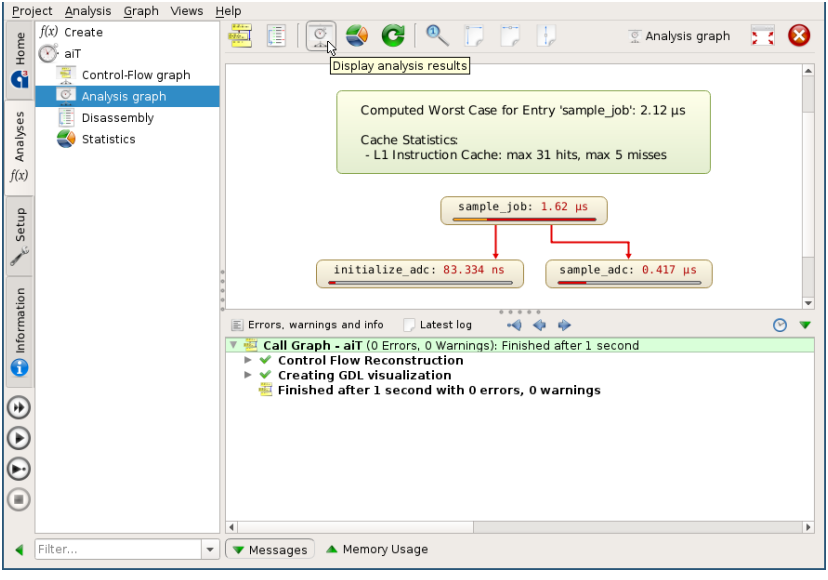
Analyseparameter



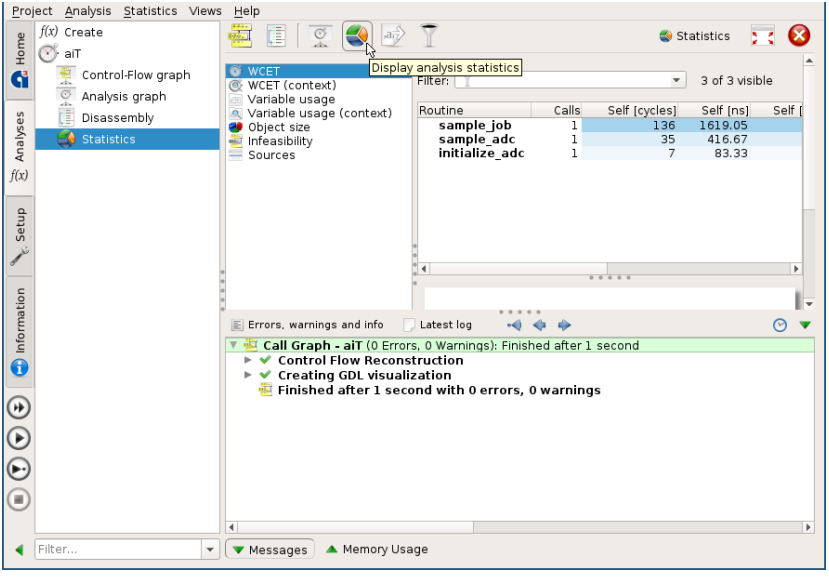
Analyse starten



Analyse untersuchen



Analyse untersuchen



Annotationen

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

Beispiele:

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

Annotationen

- 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;
}
```

Annotationen

- Genaue Anzahl von Schleifendurchläufen zu bestimmen ist teuer

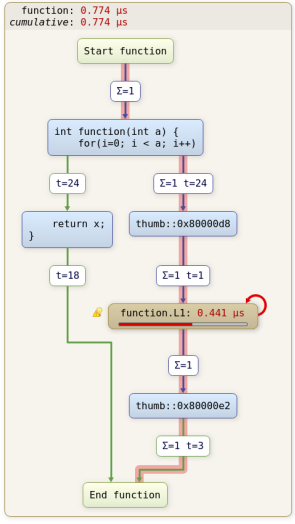
☞ aiT versucht standardmäßig nur zwei Durchläufe zu interpretieren

- Lohnt sich jedoch manchmal

☞ aiT mehr Freiheiten für die Analyse geben

Beispiel:

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



- Grenzen von Hand spezifizieren

Beispiele:

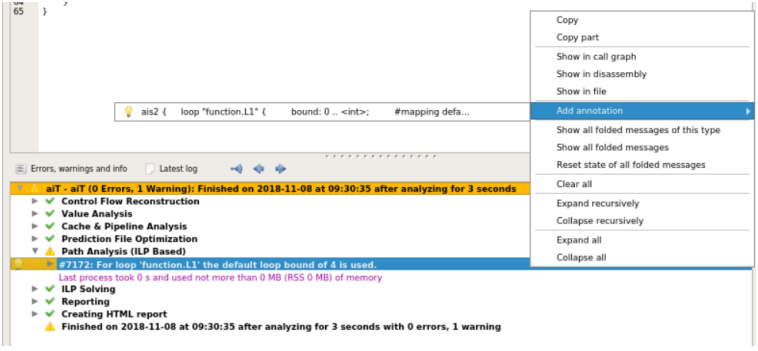
```
loop "function.L1" { bound: 0 .. 10 end; }
loop "function.L1" { bound: 10 begin; }
loop "function.L1" { bound: 10 .. inf end; }
loop "function.L1" { takes 20 ms; }
```

- Grenzen in Abhängigkeit von Registern spezifizieren

Beispiele:

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

Annotationen



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

Problem:

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

Annotationen

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“*

Besprechung der Übungsaufgabe

„Ausführungszeit“

Literaturverzeichnis (1)

[1] Franck Cassez, René Rydhof Hansen, and Mads Chr Olesen.
What is a timing anomaly?
In *Proceedings of the 12th International Workshop on Worst-Case Execution Time Analysis (WCET '12)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

[2] Steven S. Muchnick.
Advanced compiler design and implementation.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[3] Peter Puschner.
Zeitanalyse von Echtzeitprogrammen.
PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1993.

Literaturverzeichnis (2)

[4] Reinhard Wilhelm.
Embedded systems.
http://react.cs.uni-sb.de/teaching/embedded-systems-10-11/lecture-notes.html, 2010.
Lecture Notes.