

# Ausführungszeiten

## Übung zur Vorlesung EZS

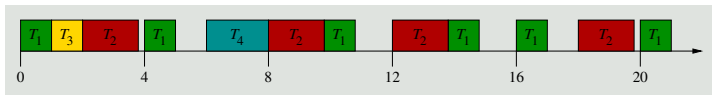
Florian Franzmann   Martin Hoffmann   Tobias Klaus  
Peter Wägemann

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

27. Oktober 2014

- 1 Rekapitulation: Worst-Case Execution-Time
- 2 Zeitmessungen
  - Zeitgeber
  - Oszilloskop
  - Diskussion
- 3 Statische Laufzeitanalyse
- 4 Handwerkszeug
  - libEVS
  - aiT

# Worst-Case Execution-Time



- Statische **Ablaufplanung**
- **Planbarkeitsanalyse**
- Später: Übernahmeprüfung
  
- **Worst-Case**
  - ↪ Obere Schranke für **alle** Fälle

## 1 Rekapitulation: Worst-Case Execution-Time

## 2 Zeitmessungen

- Zeitgeber
- Oszilloskop
- Diskussion

## 3 Statische Laufzeitanalyse

## 4 Handwerkszeug

- libEVS
- aiT

# Zähler in Mikrocontrollern

**Zähler** (*Counter*) zählen hardwarebasiert Ereignisse z.B. von:

- Externem Drehgeber (Radumdrehung)
- Externem Quarz (Real-Time Clock)
- Internem Prozessortakt (hohe Auflösung)

Äquidistante Ereignisse ermöglichen einen **Zeitgeber** (*Timer*) für

- Periodische Aktivierung
- **Messen von Zeitabständen**
- (Kontrolliertes Verbrennen von Prozessorzeit)

# Zähler Betriebsmodi

Zähler bzw. Zeitgeber bieten zwei Betriebsmodi:

**Abfragebetrieb** (Polling) Aktives Auslesen des Zählers, bis zum Erreichen eines vorgegebenen Wertes.

**Unterbrecherbetrieb** (Interrupt) Der Zähler unterbricht das laufende System beim Erreichen eines vorkonfigurierten Zählerstandes.

# Oszilloskop

- Tricore: Extrem guter Zeitgeber
  - ↳ 13.3 ns-Raster
    - nicht selbstverständlich
- Oszilloskope
  - meist schon vorhanden
  - hohe Zeitauflösung
  - einfache Bedienung

## Bedingung

Frei verfügbarer GPIO-Pin

# Diskussion

Welche Probleme können bei messbasierter Bestimmung der Ausführungszeiten auftreten?

## 1 Rekapitulation: Worst-Case Execution-Time

## 2 Zeitmessungen

- Zeitgeber
- Oszilloskop
- Diskussion

## 3 Statische Laufzeitanalyse

## 4 Handwerkszeug

- libEVS
- aiT

# Statische Laufzeitanalyse?

- Statisch?
  - ~> keine Ausführung des Programms
  
- Grundsätzliche Idee
  - Wie lange dauern die einzelnen Maschinenbefehle?
  - Wie sieht der längste Pfad durch das Programm aus?
  - ~> Addition aller Maschinenbefehle des längsten Pfades: **WCET!**
  
- Klingt doch einfach!?

# Wie lang dauert ein Maschinenbefehl?

- Cache
- Pipeline
- Branch-Prediction

# Wie lang dauert ein Maschinenbefehl?

- Cache
- Pipeline
- Branch-Prediction

Hochkomplex!

# Was ist der worst case?

- if/else
    - Einfach! Ein Pfad muss ja länger sein.
    - **ABER**: abhängig von Eingabe
  - Schleifen
    - Wie oft ausgeführt?  $\rightsquigarrow$  abhängig von Eingabe
    - Wie lange ist ein Durchlauf?  $\rightsquigarrow$  abhängig von Eingabe
- $\rightsquigarrow$  Wertanalyse: Welche Werte können Variablen annehmen?
- abstrakte Interpretation
  - Ausrollen von Schleifen: aiT-Projektparameter: max-unroll
  - oft manuelle Eingriffe erforderlich  $\rightsquigarrow$  Annotationen
- $\rightsquigarrow$  Optimierungsproblem

## 1 Rekapitulation: Worst-Case Execution-Time

## 2 Zeitmessungen

- Zeitgeber
- Oszilloskop
- Diskussion

## 3 Statische Laufzeitanalyse

## 4 Handwerkszeug

- libEVS
- aiT

# libEZS Überblick

## Plattformunabhängige Hilfsfunktionen

- Timer-Zugriff (Zeitmessung)
- DAC-Zugriff
- GPIO-Zugriff
- ...

```
aufgabe2
|-- CMakeLists.txt
|-- app.c
|-- ecos
`-- libEZS
    |-- include
    | |-- ezs_dac.h
    | |-- ezs_gpio.h
    | `-- ezs_stopwatch.h
    |-- src
    | `-- ezs_stopwatch.c
    `-- drivers
        `-- tc1796
            |-- ezs_dac.c
            |-- ezs_counter.c
            `-- ezs_gpio.c
```

Die *libEZS* wird ständig (auch von euch) erweitert.

# Zeitmessung `ezs_stopwatch.c/.h`

Für die Zeitmessung sollen zwei Funktionen implementiert werden:

```
void ezs_watch_start(cyg_uint32 *state);  
cyg_uint32 ezs_watch_stop(cyg_uint32 *state);
```

- Parameter: Zeiger auf (globale) Variable  
→ unabhängige Messzeitpunkte
- `ezs_watch_stop(cyg_uint32 *state)` gibt die Zeitdifferenz in Ticks zurück

## Hinweis

`ezs_counter_get()` in `drivers/include/ezs_counter.h`

# GPIO

## General Purpose Input/Output

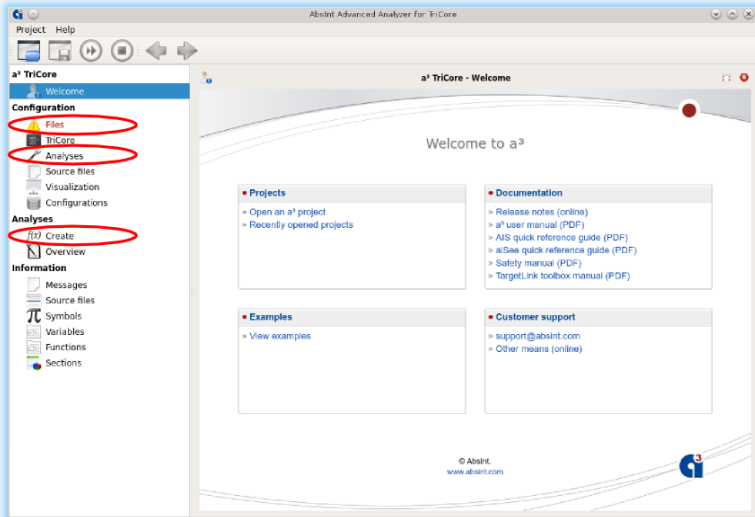
- Pins eines Mikrochips zur *freien Verwendung*
- Konfigurierbar als Ein-/Ausgang
- Oft auch Treiberstärke konfigurierbar
- Teilweise pegelfest bis 5 V
  - ~> Mikrocontroller-Handbuch lesen 😊
- Zugriff über
  - spezielle Speicheradressen
  - Spezialanweisungen

## Ansteuerung

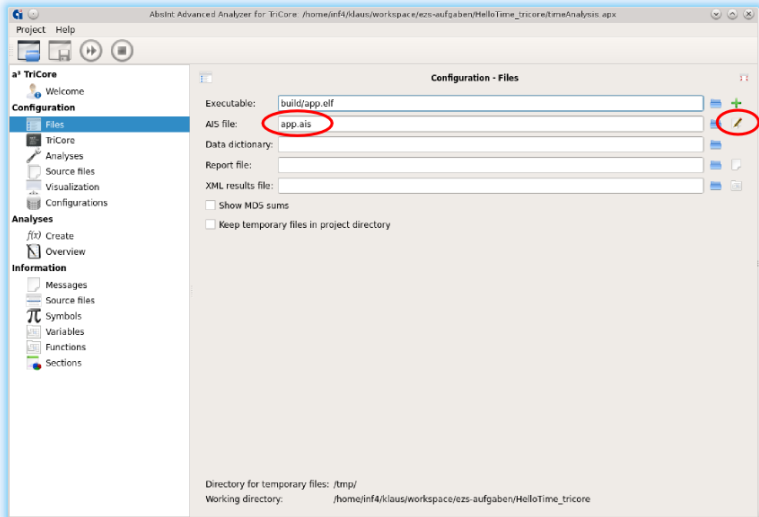
- Beim Tricore Ansteuerung per GPTA oder „von Hand“ möglich

~> `void ezs_gpio_set (bool)`

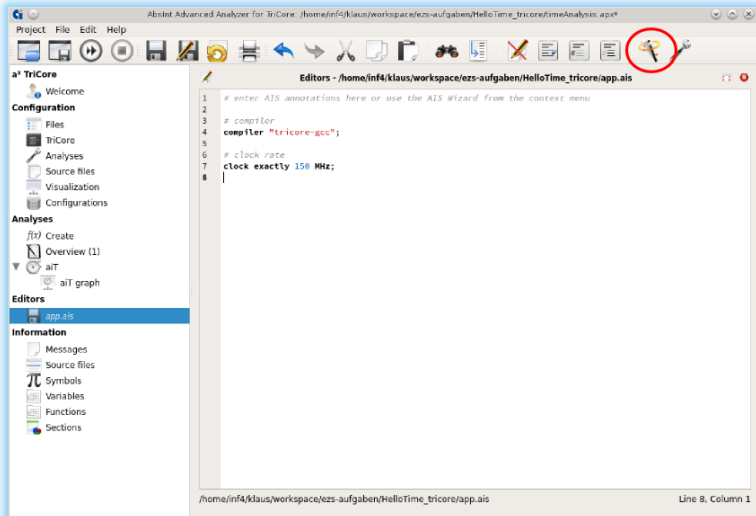
# Übersicht



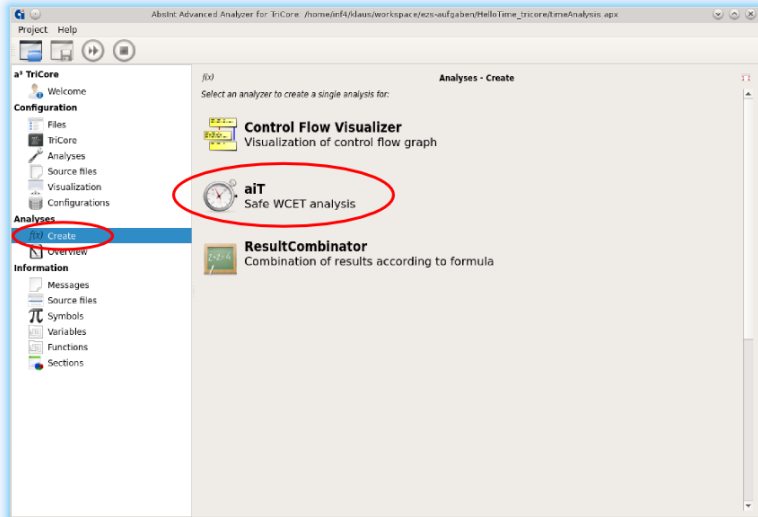
# Projektdateien



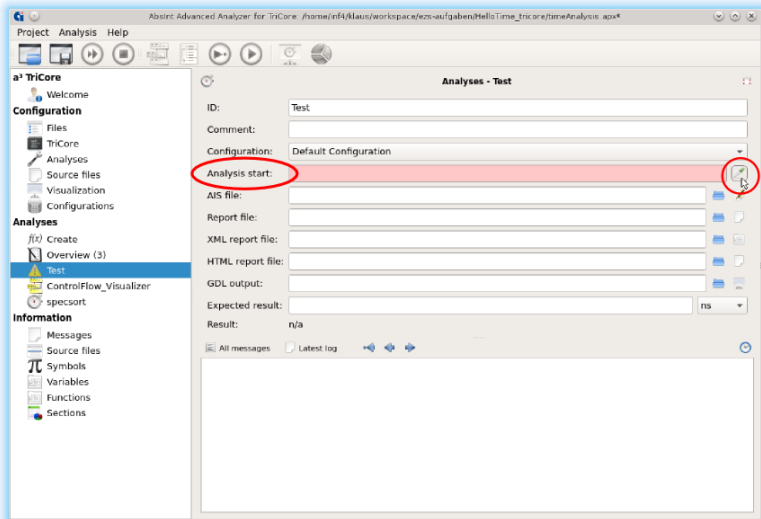
# Projektdateien bearbeiten



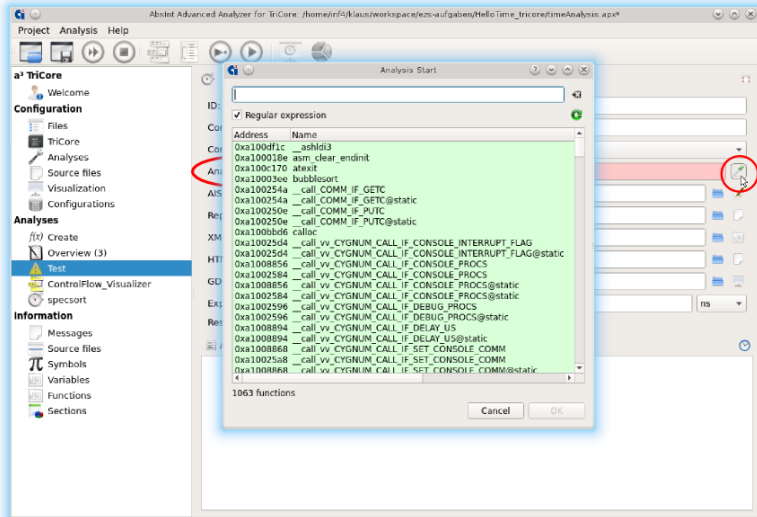
# Neue Analyse anlegen



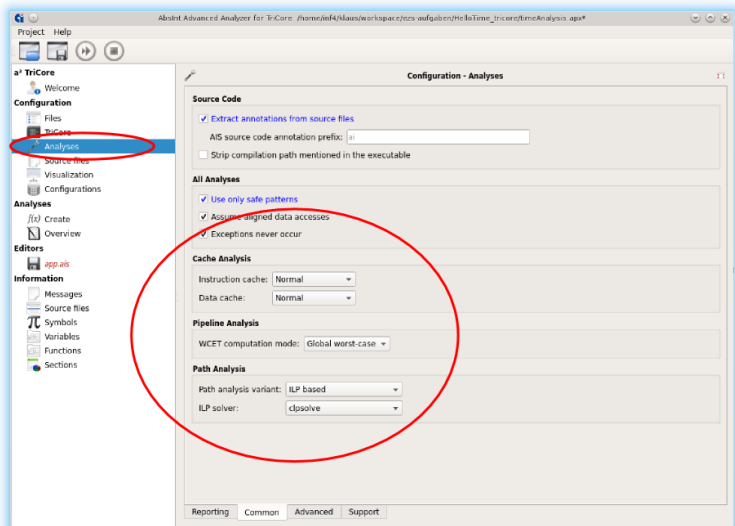
# Neue Analyse anlegen



# Neue Analyse anlegen



# Analyse Parameter



# Analyse starten

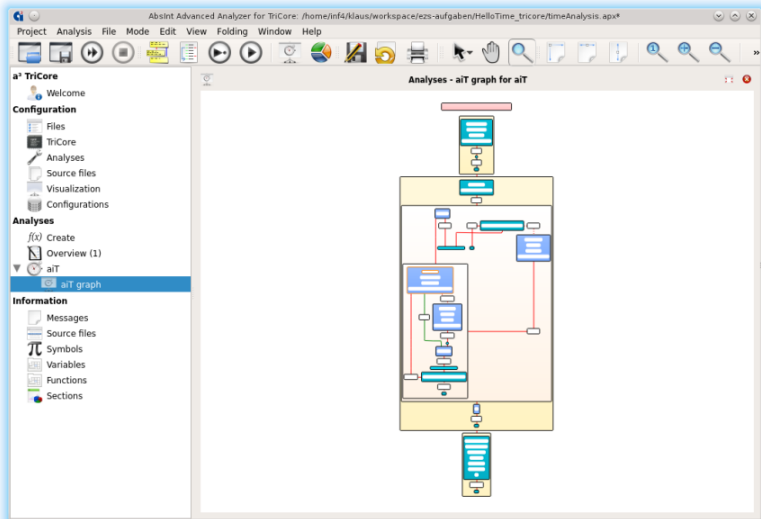
The screenshot displays the AbsInt Advanced Analyzer for TriCore interface. The window title is "AbsInt Advanced Analyzer for TriCore: /home/inf4/klaus/workspace/ezs-aufgaben/HelloTime\_tricore/timeAnalysis.apx". The interface is divided into several sections:

- Project Analysis Help**: A menu bar at the top.
- Configuration**: A sidebar on the left containing "Files", "TriCore", "Analyses", "Source files", "Visualization", and "Configurations".
- Analyses**: A sidebar on the left containing "Create", "Overview (1)", and "aiT".
- Editors**: A sidebar on the left containing "app.ais".
- Information**: A sidebar on the left containing "Messages", "Source files", "Symbols", "Variables", "Functions", and "Sections".
- Analyses - aiT**: The main configuration panel on the right, showing fields for ID (aiT), Comment, Configuration (Default Configuration), Analysis start (test\_job), AIS file, Report file, XML report file, HTML report file, GDL output, Expected result, and Result (2267922 cycles = 15.33 ms).
- Messages**: A log area at the bottom showing the execution results of the aiT analysis.

The "aiT" analysis configuration is highlighted with a red circle. The execution results are also highlighted with a red circle. The results show that the analysis completed successfully with 0 errors and 0 warnings.

```
aiT - aiT [interactive] (0 Errors, 0 Warnings): Finished after 31 seconds
├─ Control Flow Reconstruction
├─ Value Analysis
├─ Cache & Pipeline Analysis
├─ Prediction File Optimization
├─ Path Analysis (ILP Based)
├─ ILP Solving
├─ Applying Path Analysis Results
├─ Reporting
├─ Visualization
└─ Finished after 31 seconds with 0 errors, 0 warnings
```

# Analyse untersuchen



# Fragen?