EMSBench:
Benchmark und Testumgebung für reaktive Systeme

Florian Kluge, Theo Ungerer

Institut für Informatik
Universität Augsburg

Echtzeit 2015
12. November 2015

Echtzeit-Benchmarks

- Test, Evaluierung & Vergleich von
  - Komponenten/Mechanismen
  - Systemen
  - Werkzeugen

- Beispiele:
  - Mälardalen-Suite: WCET-Analyse
  - PARSEC: Parallele Programme

- Typische Eigenschaften:
  - geringe Komplexität
  - kein reaktives Verhalten

  $\rightarrow$ kein umfassender Test einer Systemarchitektur möglich

Gewünschte Eigenschaften für Systembenchmark

- Komplexität
  - → Zusammenspiel mehrerer Programmmodule
- Reaktivität
  - → Interaktion mit Umgebung
- Parallelität
  - → Einsatz in Mehrkernprozessoren

→ Nutze Software aus realem Anwendungsfall

→ Ermögliche realitätsnahe Ausführung

→ EMSBench als Benchmark und Testumgebung

Überblick

Motivation

FreeEMS & EMSBench.ems

Trace-Erzeugung

Einsatz

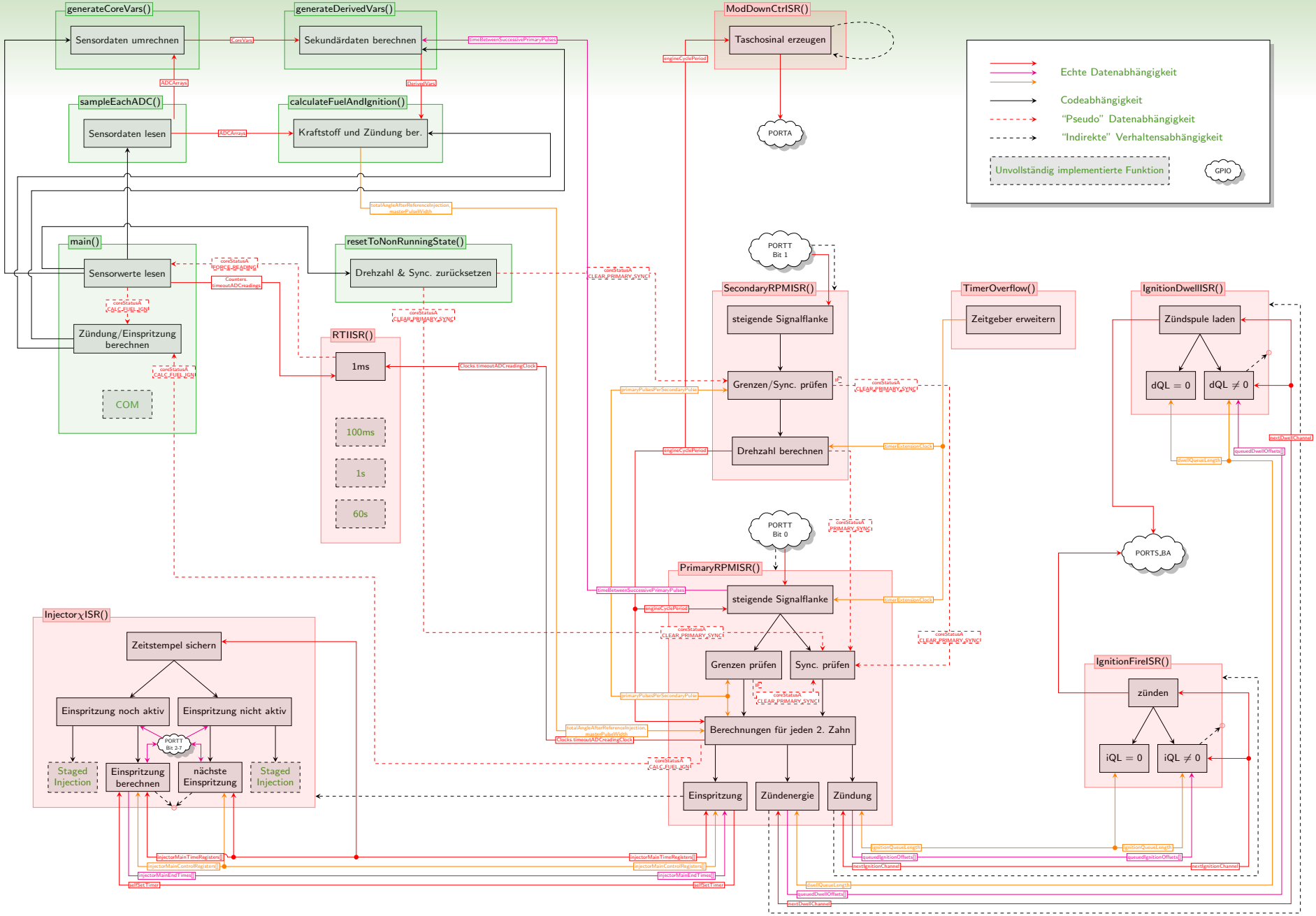
Zusammenfassung & Ausblick

FreeEMS

- Open Source Engine Management System
- 16-Bit Freescale µC
- erfolgreich getestet auf über 20 Verbrennungsmotoren

- Basis für diese Arbeit: Version 0.1.1

- Eigenschaften:
  - Umfangreicher ereignisgesteuerter Code (ISRs)
  - `main()`-Schleife für niedrigpriore Aufgaben


generateCoreVars()
Sensordaten umrechnen
generateDerivedVars()
Sekundärdaten berechnen
ModDownCtrISR()
Taschosinal erzeugen
Echte Datenabhängigkeit
Codeabhängigkeit
"Pseudo" Datenabhängigkeit
"Indirekte" Verhaltensabhängigkeit
Unvollständig implementierte Funktion
GPIO
sampleEachADC()
Sensordaten lesen
calculateFuelAndIgnition()
Kraftstoff und Zündung ber.
PORTA
main()
Sensorwerte lesen
resetToNonRunningState()
Drehzahl & Sync. zurücksetzen
Zündung/Einspritzung berechnen
COM
RTIISR()
1ms
100ms
1s
60s
PORTT Bit 1
SecondaryRPMISR()
steigende Signalflanke
Grenzen/Sync. prüfen
Drehzahl berechnen
TimerOverflow()
Zeitgeber erweitern
IgnitionDwellISR()
Zündspule laden
dQL = 0
dQL ≠ 0
PORTT Bit 0
PrimaryRPMISR()
steigende Signalflanke
Grenzen prüfen
Sync. prüfen
Berechnungen für jeden 2. Zahn
Einspritzung
Zündenergie
Zündung
PORTS_BA
IgnitionFireISR()
zünden
iQL = 0
iQL ≠ 0
InjectorχISR()
Zeitstempel sichern
Einspritzung noch aktiv
Einspritzung nicht aktiv
PORTT Bit 2-7
Staged Injection
Einspritzung berechnen
nächste Einspritzung
Staged Injection

FreeEMS

Wichtige Funktionen & Hardware-Abhängigkeiten

- 2 Input-Capture-Kanäle für Kurbel-/Nockenwellensensor:
  - `PrimaryRPMISR`: Stellen von Einspritz- und Zündzeitpunkten.
  - `SecondaryRPMISR`: Sicherstellung der Synchronität zwischen EMS und Motor
- $\leq$ 6 Output-Compare-Kanäle:
  - `InjectorXISRs`
  - Einstellen von Einspritz-Ende bzw. nächstem Einspritzvorgang
- Steuerung der Zündung über 2 Interval Timer:
  - `IgnitionDwellISR`: Laden der Zündspule
  - `IgnitionFireISR`: Entladen der Zündspule
- Periodische Aufgaben: `RTIISR` (1 ms, 10 ms, 100 ms, ...)
- `main`-Schleife: Lesen von ADCs, Neuberechnung der Einspritz- und Zündzeiten.

Anpassungen in EMSBench.ems

- Abstraktion von Sensordaten
  - Sensor-Addressen zeigen auf regulären Speicher
  - Sinnvolle Initialisierung
- Beibehaltung von Kurbel-/Nockenwellensensor
  - Zentral für Ablaufverhalten!
- Abstraktionsschicht (HAL)
  - Portierbarkeit auf beliebige (eingebettete) Plattformen
  - Aktuell: STM32F4-Discovery (ARM Cortex-M4), eigenes FPGA-µC-Design (Altera Nios-2)

Trace-Erzeugung

- Ziel: realitätsnahe Ausführung von EMSBench.`ems`
  - z.B. für Performance-Evaluierung unter vergleichbaren Bedingungen

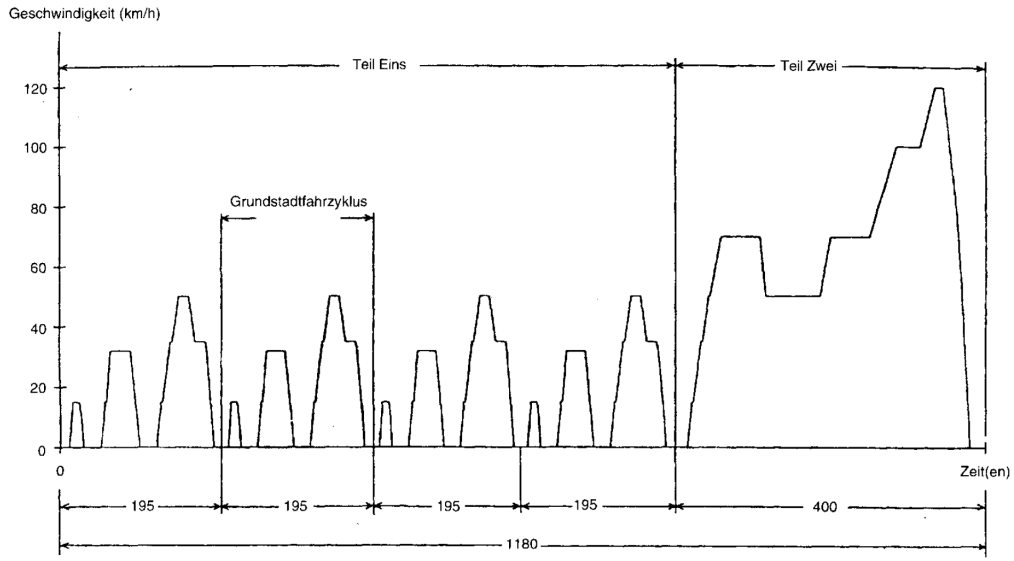
$\rightarrow$ Eingabe-Traces für Kurbel-/Nockenwellensensor-Eingänge!

  - basierend auf Fahrzyklen

- Benötigte Tools:
  - EMSBench.`tgpp`: Umwandlung Fahrzyklus $\rightarrow$ Kurbel-/Nockenwellenzyklus
  - EMSBench.`tg`: Emulation des Kurbel-/Nockenwellenverhaltens

Fahrzyklus

- z.B. „Neuer europäischer Fahrzyklus“ (NEFZ)
- Abfolge von Betriebszuständen
- Aufbau Betriebszustand:
  - Anfangs- und Endgeschwindigkeit
  - Beschleunigung
  - Dauer
  - verwendeter Gang bzw. Gangwechsel



Quelle: Richtlinie des Rates vom 20. März 1970 zur Angleichung der Rechtsvorschriften der Mitgliedstaaten über Maßnahmen gegen die Verunreinigung der Luft durch Emissionen von Kraftfahrzeugen. Fassung vom 01.01.2007.

EMSBench.tgpp

Schnittstelle

- Eingabe:
  - Fahrzyklus
  - Fahrzeugdaten (Bereifung, Getriebe, Leerlaufdrehzahl)
- Ausgabe:
  - Kurbel-/Nockenwellenzyklus (Phasen)
  - Phase: Dauer Δt, Beschleunigung α
- Ausführung auf PC

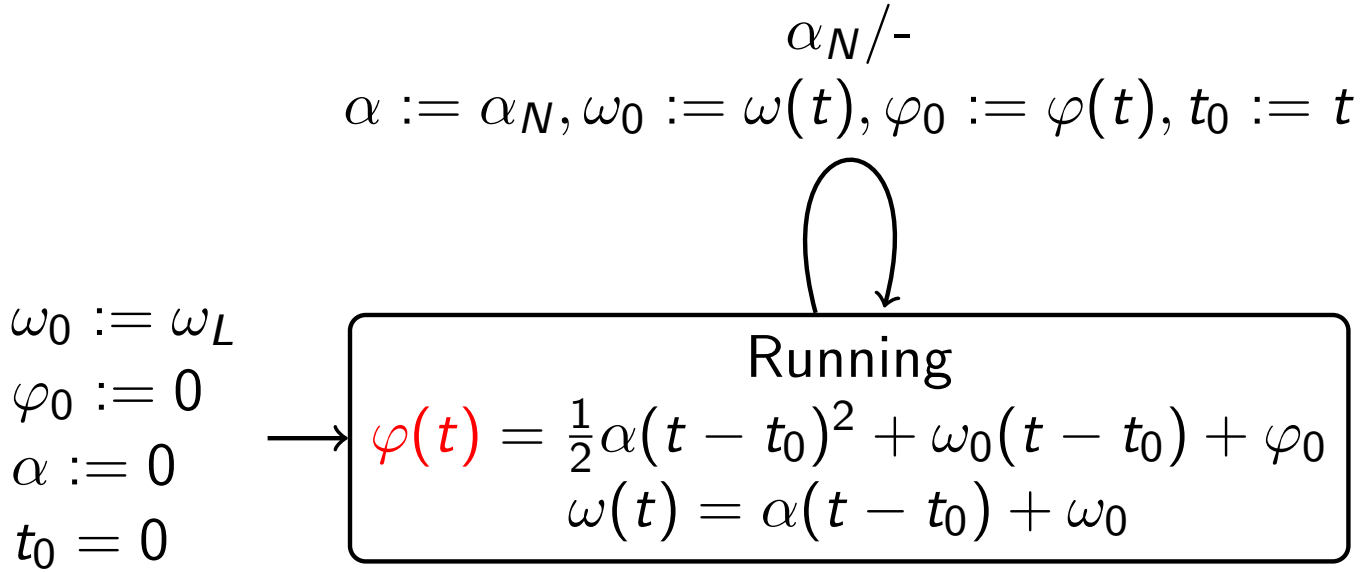
- Direkte Abbildung Betriebszustand $\rightarrow$ Phase nur bei konstanter Beschleunigung mit festem Gang

- Betriebszustände mit mehreren Phasen:
  - Anfahren aus Stillstand
  - Gangwechsel
  - Verzögerung bei geöffneter Kupplung

$\rightarrow$ Angleichung an/von Leerlauf nötig

EMSBench.tg

Verhalten der Kurbel-/Nockenwelle



Eingabe:

$\alpha_N \in \mathbb{R} \cup \{\text{absent}\}$

Ausgabe:

$\omega(t) \in \mathbb{R}$

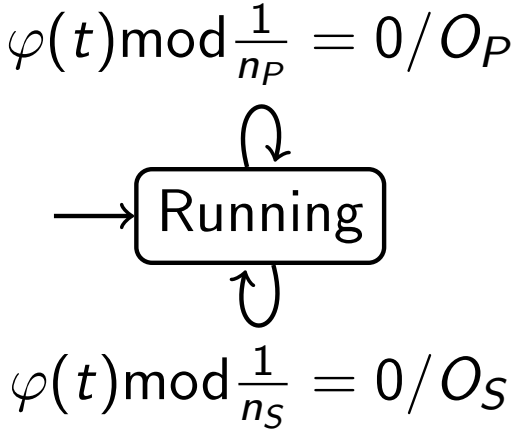
$\varphi(t) \in \mathbb{R}^+$

Variablen:

$\alpha \in \mathbb{R}$

$\omega_0, \varphi_0, t_0 \in \mathbb{R}^+$

Verhalten des Kurbel-/Nockenwellensensors



Eingabe:

$\varphi(t) \in \mathbb{R}^+$

Ausgabe:

$O_P \in \{\text{absent}, \text{present}\}$

$O_S \in \{\text{absent}, \text{present}\}$

Parameter:

n_P, n_S: Anzahl Primär-/Sekundärzähne des Kurbel-/Nockenwellensensors

- Auf eingebetteter Plattform
- Benötigt 2 Output-Compare-Kanäle
- Hardware-Abstraktionsschicht für Portierbarkeit
- Aktuell: STM32F4-Discovery (ARM Cortex-M4), eigenes FPGA-µC-Design (Altera Nios-2)

Einsatzmöglichkeiten

- WCET-Analyse
  - Einfluss von Unterbrechungen
  - Cache-Analyse
- Schedulability-Analyse
  - Eignung einer Plattform (Hardware + OS)
- Beispiel-/Testprogramm
  - Test und Evaluierung von z.B. Betriebssystemmechanismen

Zusammenfassung

- Klassische Benchmark-Suiten für umfassende System-Evaluierung nur eingeschränkt geeignet!

- EMSBench:
  - Testbett und Benchmark
  - Benchmark EMSBench.`ems` basiert auf FreeEMS Motorsteuerung
  - Eingabe-Trace-Erzeugung mit EMSBench.`tgpp` und EMSBench.`tg`

- Eigenschaften:
  - Komplexität: Interagierende nebenläufige Module
  - Reaktivität: Unterbrechungsgetriebener Code

- Download: `https://github.com/unia-sik/emsbench`

Ausblick

- Parallelisierung:
  - Geringes Potential in FreeEMS/EMSBench.`ems` selbst
  - Ergänze ISRs um künstlichen parallelen Code
    z.B. Signalverarbeitung (modelliert Behandlung von Motorklopfen)
  - Alternativ: Verteilung der ISRs auf verschiedene Kerne

- Einsatz als Benchmark zur WCET-Analyse: in Arbeit