Analyse und Entwicklung einer µ-Controller-basierten Echtzeit-Steuerung für Sensorsysteme mit Ada-Raven

Dipl.-Inf. Matthias Götz
ci-Tec GmbH Karlsruhe

Dr.-Ing. Jörg Matthes,
Dr.-Ing. Hubert B. Keller,
Dipl.-Math. Rolf Seifert
Institut für Angewandte Informatik
Forschungszentrum Karlsruhe GmbH / KIT

Inhalt

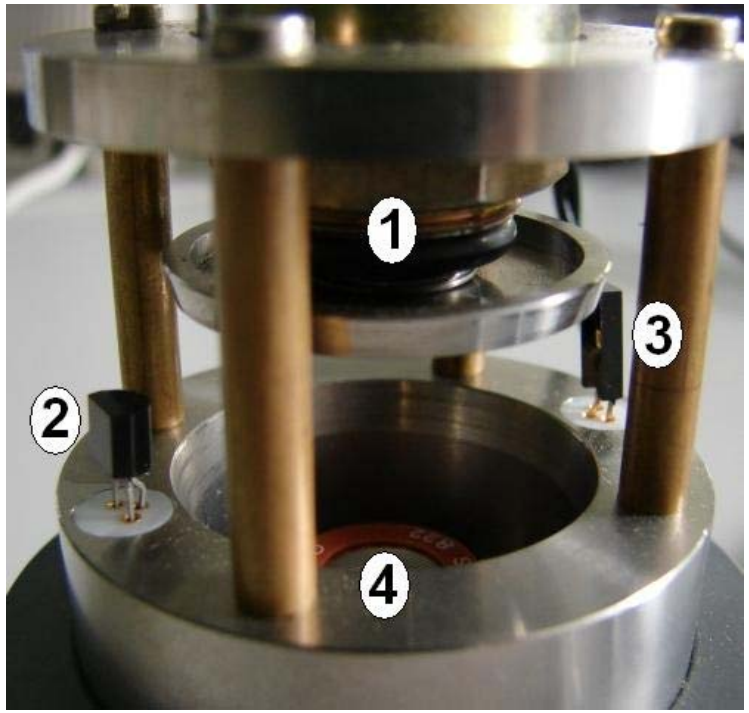
- Einführung
- Architektur
- Tasking-Modell
- Scheduling Analyse
- Einsatz des Ravenscar-Profils
- Erweiterungen
- Zusammenfassung

Einführung – Elektronische Nase

- Sensorsystem zur Detektion und Konzentrationsbestimmung von Gaskomponenten

- Einsatz für Leckage-Überwachung (Gefahrenstoffe), Qualitätskontrollen oder Umwelt-Monitoring

- Aufbau:
  - Sensorkammer mit Hubmagnet (1)
  - Temperatursensor (2)
  - Feuchtesensor (3)
  - beheizter gassensitiver Sensor (4)



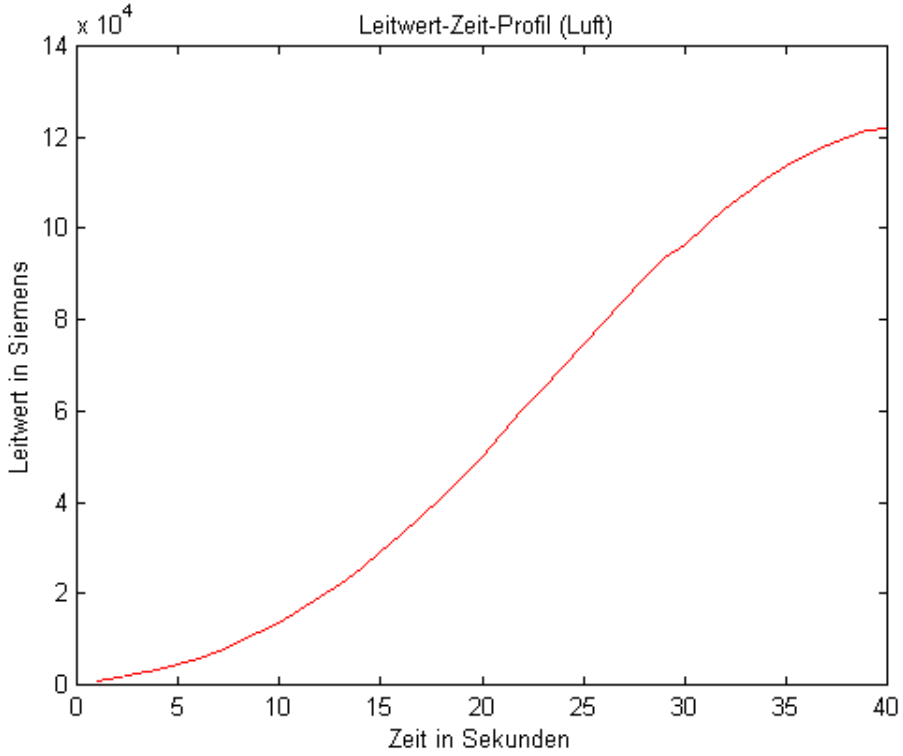
Einführung – Elektronische Nase

Messprinzip

- zyklische Variation der Sensortemperatur durch Beheizung
  → Temperatur-Zeit-Profil

- temperaturabhängiges Absorptions-verhalten der Gasmoleküle an Sensor-Oberfläche führt zu Leitwertänderung
  → Leitwert-Zeit-Profil

- Musteranalyse auf Basis Leitwert-Zeit-Profil
  → Klassifikation
  → Konzentrationsbestimmung



Einführung – Elektronische Nase

Typischer Aufbau eines Messzyklus‘

- Phase Abkühlung (0 s – 15 s)
- Phase Vorbereitung (15 s – 25 s)
- Phase Messung (25 s – 65 s)

Zeit/s	Hubmagnet	Heizung	Temperatur/Feuchtigkeit	Leitwert
0	Auf	Aus	Keine Messung	Keine Messung
5				
10				
15		An (30%)	Messung	
20				
25	Zu	An (50%)	Keine Messung	Messung
30				
35				
40				
45				
50				
55				
60				
65		An (100%)		

Zu Beginn 5 Sekunden Initialisierung dann 1. Zyklus

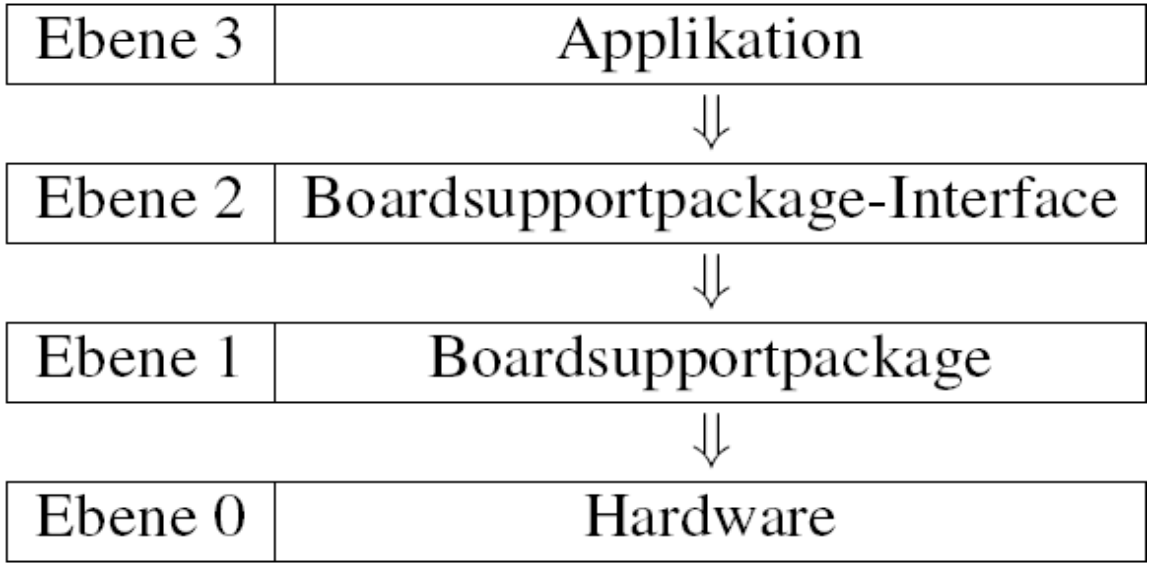
Einführung – Elektronische Nase

Aufgaben der Software

- zyklische Ansteuerung des Hubmagneten zur Steuerung der einzelnen Phasen des Messzyklus‘,
- kontinuierliche Ansteuerung der Sensorheizung, zur Generierung eines Temperatur-Zeit-Profils (z.B. stetig ansteigende Sensortemperatur),
- kontinuierliche Erfassung des Sensorleitwerts,
- zyklische Erfassung der Umgebungstemperatur und –feuchte,
- Musteranalyse des Leitwert-Zeit-Profils am Ende oder bereits während eines Messzyklus‘,
- Kommunikation mit einem Sensor-Managementsystem zur zentralen Datenverarbeitung z.B. per TCP/IP
- …

Architektur

Ebenenarchitektur zur Trennung von hardware- und anwendungsspezifischen Aspekten



Architektur

Hardware

- PHYTEC phyCORE Motorola MPC 565
- leistungsfähiger Prozessor für spätere Musteranalyse
- Ethernet für zukünftige Anbindung an Sensor-Management-System

Architektur

Board-Support-Package BSP (extern bereit gestellt)

bietet Funktionen zur

- Analog-Digital-Umsetzung
- Pulsweitenmodulation
- Digitalen Ein-/Ausgabe

Benutzung der Funktionen ist sehr hardware-spezifisch

(kaum Abstraktion)

Architektur

Board-Support-Package-Interface soll hardware-spezifische Funktionen des BSP abstrahieren und der Anwendungsschicht zu Verfügung stellen

Ziele:

- hardware-unabhängige, komfortable Funktionen für Applikationsentwicklung
- Wechsel der Hardware → nur das Anpassen der BSP-Interface-Schicht auf das spezifische BSP nötig (Applikationsschicht bleibt unverändert)
- kapselt gemeinsame Hardware-Ressourcen für Exklusiv- bzw. unterbrechungsfreie Nutzung (Beispiel später)

Architektur

Applikationsschicht: nebenläufige Aktivitäten der Steuerungssoftware werden auf Ada-Tasks abgebildet.

Vorteile:

- Änderungen im Aufbau des Messzyklus erfordern i.Allg. nur Änderung der Zeitkonstanten
- Einfacher Aufbau der einzelnen Tasks

Basisvariante besitzt:

- Initialisierungs-,
- Leitwertmessungs-,
- Heizungssteuerungs-,
- Temperatur- und Feuchtemessungs- sowie
- Hubmagnetsteuerungs-Task

Architektur – Ada Task Beispiel

```
package body Hubmagnet is

    --Hubmagnet Task--
    task body Hubmagnet_Task is

        --Zeitvariable--
        Next : Ada.Real_Time.Time := Zeit_Konstanten.Zyklus_Start;

    begin

        --Initialisierung abwarten--
        delay until Next;

        --Zyklusschleife--
        loop

            --Open--
            Pin_Konstanten.Write(Pin_Konstanten.Hubmagnet_Pin_Name, True);

            --Chill & Prepare abwarten--
            Next := Ada.Real_Time."+"(Next, (Ada.Real_Time."+"(Zeit_Konstanten.Chill_Time, Zeit_Konstanten.Prepare_Time)));
            delay until Next;

            --Close--
            Pin_Konstanten.Write(Pin_Konstanten.Hubmagnet_Pin_Name, False);

            --Measure abwarten--
            Next := Ada.Real_Time."+"(Next, Zeit_Konstanten.Measure_Time);
            delay until Next;

        end loop;

    end Hubmagnet_Task;

    --Task instanziieren--
    Hubmagnet : Hubmagnet_Task;

end Hubmagnet;
```

Tasking-Modell

- Gemäß dem Ravenscar-Profil kommt ein preemptives, prioritätengesteuertes Scheduling zum Einsatz (FIFO within priorities)

- durch das preemptive Scheduling können niederpriore Tasks während der Nutzung einer gemeinsamen Ressource (im kritischen Abschnitt) durch höherpriore Tasks verdrängt werden

- um dabei entstehende Probleme auszuschließen kommt für gemeinsam genutzte Ressourcen das Ceiling-Locking zum Einsatz
  - verhindert das Problem der Prioritätsumkehr (priority inversion)
  - verhindert Verklemmungen (deadlocks)
  - verhindert Blockierungsketten (chained blocking) (maximale Blockierungszeit = Dauer eines kritischen Abschnitts einer anderen Task)

Tasking-Modell

- gemeinsam genutzte Ressourcen werden im BSP-Interface durch Ada-Protected-Objects gekapselt

- durch das Ada-Laufzeitsystem wird die Exklusivnutzung des Protected-Objects sicher gestellt

- den Protected-Objects werden Prioritäten zugewiesen, die als Ceiling-Prioritäten genutzt werden

  → nutzt eine Task eine gemeinsame Ressource, dann erhält sie in dieser Zeit die Ceiling Priorität der Ressource

- bei der Vergabe der Prioritäten sind zwei Fälle zu unterscheiden

Tasking-Modell

1. Die Ressource repräsentiert gemeinsame genutzte Daten, für deren Konsistenzsicherung ein exklusiver Zugriff notwendig ist (Standardfall)

   → Ceiling-Priorität = maximale Priorität aller Tasks, die diese Ressource nutzen

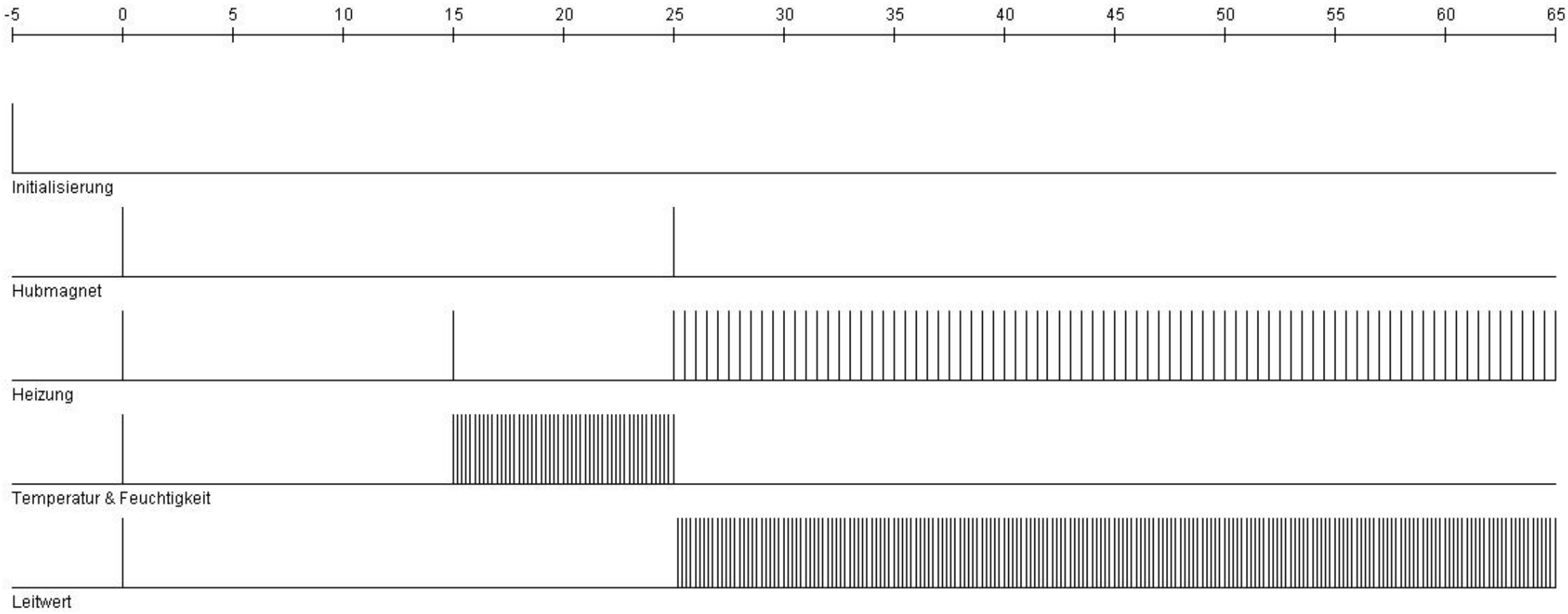
2. Die Ressource repräsentiert eine gemeinsam genutzte Hardware-Komponente, deren Aktivität nicht unterbrochen werden darf (z.B. Analog-Digital-Umsetzung darf nicht durch irgendeine andere Task unterbrochen werden, sonst falsche Werte)

   → Ceiling-Priorität ≥ maximale Priorität aller Tasks (z.B. maximal mögliche Priorität)

   → kritischer Abschnitt (z.B. A/D-Wandlung) wird niemals unterbrochen

Scheduling-Analyse

Aktivitäten der einzelnen Tasks während eines Messzyklus



Hinweis: die einzelnen Linien repräsentieren bereits die maximalen Ausführungszeiten
Aufgrund der Leistungsfähigkeit des Prozessors ist kleinste Linienstärke bereits zu stark.

Scheduling-Analyse

Vergabe der Task-Prioritäten nach dem Deadline-Monotonic-Verfahren

(relative Deadlines D_i ergeben sich aus den technischen Anforderungen)

Task-Name	Periodendauer T_i	max. Ausführungszeit C_i	Deadline D_i	Priorität
Temperatur/ Feuchtigkeit	200 ms	0.06 ms	20 ms	254
Leitwert	200 ms	0.09 ms	20 ms	254
Heizung	500 ms	0.01 ms	50 ms	253
Hubmagnet	25000 ms	0.02 ms	100 ms	252

max. Ausführungszeit C_i beinhaltet bereits die maximale Dauer eines kritischen Abschnitts einer anderen Task (maximale Blockierung)

Scheduling-Analyse

Nachweis der Ausführbarkeit der Prozessmenge

- hier trivial: Summe aller maximalen Ausführungszeiten ist kleiner als die kleinste relative Deadline (ohne Musteranalyse-Task)

- allgemein: Nachweis über Rate-Monotonic-Analyse
  - Berechnung der Prozessorauslastung (C_i inkl. Blockierung)
  - Anstelle der Periodendauer werden die relativen Deadlines gesetzt

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{1/n} - 1\right)$$

  - Für n=4 Tasks ergibt sich hier 0.0079 < 0.76

Einsatz des Ravenscar-Profils

- Das Ravenscar-Profil ist ein definiertes Subset von Ada für die Entwicklung Hoch-Sicherheitskritischer Systeme

- Die Restriktionen durch das Profil erlauben den statischen Nachweis bestimmter Eigenschaften wie Vorhersagbarkeit, Ausführbarkeit und Speicherbeschränktheit

- Ravenscar-Tasking-Modell lässt sich auf kleines, effizientes Laufzeitsystem abbilden

→ Ziele im Projekt:
  - Erfahrung mit Ravenscar und Ada sammeln
  - höhere Zuverlässigkeit der Software

Einsatz des Ravenscar-Profils

Einige Ravenscar-Restriktion → Konsequenz für die Steuerungssoftware

- *FIFO_Within_Priorities* + *Ceiling_Locking* + *No_Dynamic_Priorities* → Task- und Ceiling-Prioritäten statisch vergeben, dadurch einfache Scheduling-Analyse
- *No_Task_Allocators, No_Task_Hierarchy* → Tasks stehen ohnehin bei Programmstart fest und stehen auf einer Ebene
- *Max_Task_Entries =>0* → Direkte Kommunikation zwischen den Tasks (Synchronisierung) wurde nicht benötigt, Datenaustausch und Hardware-Zugriffskontrolle über Protected-Objects
- *No_Relative_Delay* → Nur delay-until-Statements werden verwendet

Erweiterungen

- Task zur Auswertung der Leitwert-Zeit-Profile (Musteranalyse)
- niedrige Priorität
- Daten werden von Mess-Tasks über Protected-Objects bereit gestellt (Sicherstellung der Datenkonsistenz)
- „Abholen“ der Daten über entry mit Barrier-Condition „New_Data“
  → Auswerte-Task ruft entry auf und wird aktiviert, sobald Barrier-Condition erfüllt ist und keine höher priore Task aktiv ist
- Task zur Datenübermittlung an Sensormanagementsystem über TCP/IP (ähnliche Struktur wie Datenauswerte-Task)

Zusammenfassung

- Entwicklung einer µ-Controller-Sensorsteuerungssoftware mit Ada-Raven
- Ebenenmodell erlaubt Abstraktion von spezifischer Hardware und gute Portierbarkeit der Applikation
- Nebenläufige Aktivitäten durch Ada-Tasks direkt und übersichtlich realisierbar
- Preemptives, prioritäten-gesteuertes Scheduling
- Kontrolle gemeinsamer Ressourcen über Protected-Objects mit Ceiling-Locking
- Vergabe der Prioritäten mit Deadline-Monotonic-Verfahren
- Nachweis der Ausführbarkeit mit Rate-Monotonic-Analyse

→ Ravenscar-Profil stellt auch in Hinblick auf Erweiterungen keine echte Einschränkung dar (Vorteile überwiegen)