Modellbasierte Systementwicklung für Smarte Sensoren

Dr. Benjamin Schwabe, Andrea Hollenbach

26.05.2017

Agenda

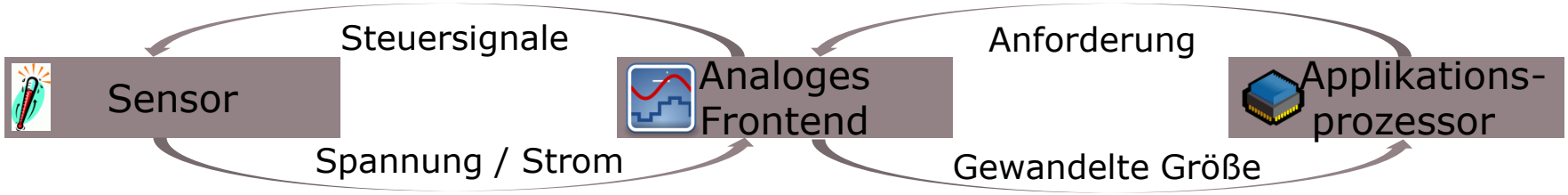
1. Einführung Smarte Sensoren
2. Entwicklung der Auswertealgorithmen und Code Generierung
3. Erweiterungen des Modellbasierten Ansatzes
4. Zusammenfassung

Agenda

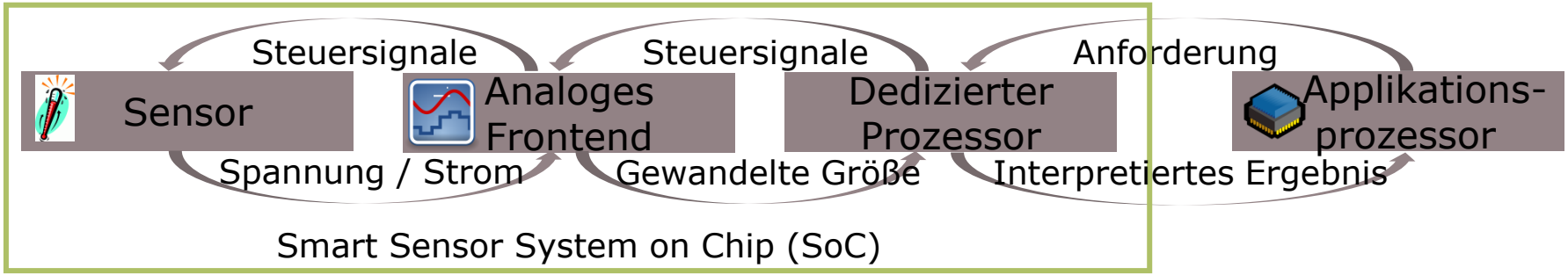
1. Einführung Smarte Sensoren
2. Entwicklung der Auswertealgorithmen und Code Generierung
3. Erweiterungen des Modellbasierten Ansatzes
4. Zusammenfassung

Was macht Sensoren smart?

› Klassische Sensoren:

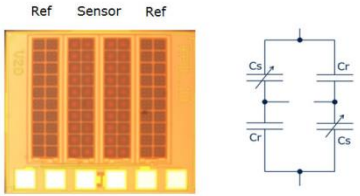


› Smarte Sensoren:

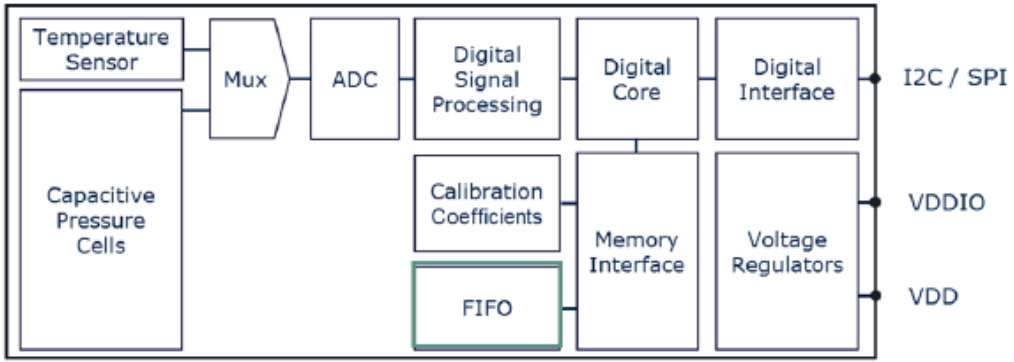


Beispiel: Digital Pressure Sensor DPS310

- Kapazitiver Drucksensor:



- In einem System on Chip:



- Für Wearables, Multicopter,…

Quelle: http://www.infineon.com/dgdl/Infineon-DPS310-DS-v01_00-EN.pdf?fileId=5546d462576f34750157750826c42242

Welche Vorteile bieten smarte Sensoren?

- **Spart Platz:**
  weniger Komponenten auf dem Board, Bussystem
- **Flexibler Einsatz**:
  SoC angepasst an die Aufgabe
- **Energiesparend**:
  Applikationsprozessor muss nicht auf das gewandelte Ergebnis warten
- **Mehrwert:**
  schnellere Systementwicklung beim Kunden

Neue Herausforderungen…

- System on Chip Architektur (Mixed Signal+ digital + Firmware)
- Embedded Firmware Design
- Hardware + Firmware Verifikation
- Software für Evaluierung und Design-In (bspw. GUIs, Debugger)

Wie bauen wir die nötigen Kompetenzen rasch auf?

Was tun mit sich ändernden Spezifikationen?

Wie beschleunigen wir die SoC Entwicklung?

Wie schaffen wir ein günstiges SoC?

Risikominimierung einer SoC Entwicklung?

…unsere Antwort: Modellbasierte Entwicklung & Code Generierung

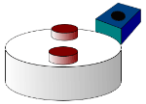
- **Systemsimulationen** (Wünsch Dir Was)
  - Modellbasierte Konzeptanalyse
  - Implementierung des Konzepts ist offen
- **FPGA** und ähnliche Implementierungen (erste Einschränkungen)
  - Hardware in the Loop
  - Rapid Prototyping
- **Dediziertes Produkt** in Silizium (geringe Flexibilität)
  - Etwa: nur noch Parametrisierung möglich, fixe Firmware
  - Enge Speichergrenzen

Agenda

1. Einführung Smarte Sensoren
2. Entwicklung der Auswertealgorithmen und Code Generierung
3. Erweiterungen des Modellbasierten Ansatzes
4. Zusammenfassung

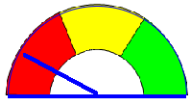
Aspekte bei der Entwicklung der Algorithmen zur Auswertung der Rohdaten vom Sensor

- Umrechnung kompliziert:



Sensordaten
(Spannung, Strom,…)

- Nichtlinearer Zusammenhang
- Differentiale / Integration
- Signal vs. Rauschen (Filtern)



Messgröße
(Druck, Gesten,…)

- Kostenoptimiertes System on Chip:
  - ARM M0 o.ä. ohne native Fließpunktarithmetik
  - Speicherverbrauch
- Metriken zur Code Analyse:
  - Speicherverbrauch
  - Rechenaufwand: Zyklen
  - Lesbarkeit des Codes: Codezeilen

Beispiel: Lösen linearer Systeme (LR / QR Zerlegung)

› Lineares Gleichungssystem: $K\,x = b$

Gauss Zerlegung (LU/LR)

K = LR mit

› L: untere Dreiecksmatrix
› R: obere Dreiecksmatrix

$$x = U^{-1}L^{-1}\,b$$

Orthogonalisierung (QR)

K = QR mit

› Q: orthogonale Matrix
› R: obere Dreiecksmatrix

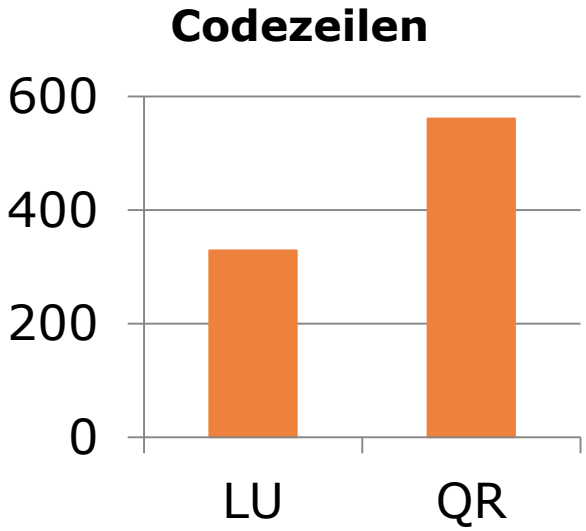
$$x = R^{-1}Q^{T}\,b$$

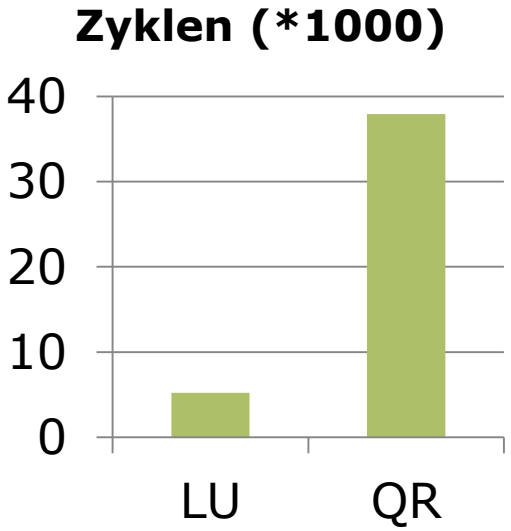
Keine echte Invertierung ➔ Vorwärts - /Rückwärtssubstitution

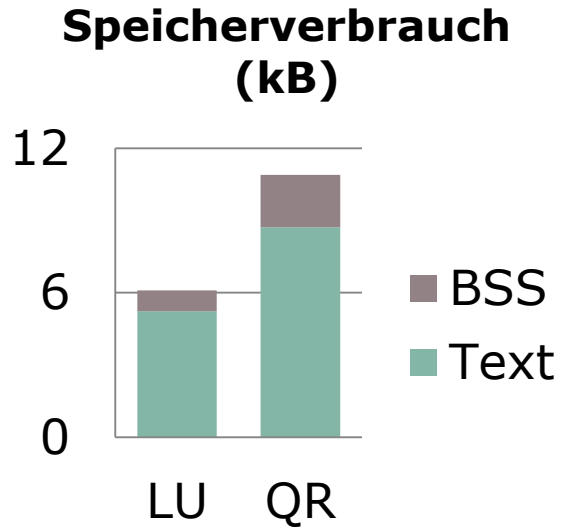
Vergleich der Zerlegungsansätze auf einem ARM M4: Wie erwartet – QR Zerlegung ist viel zu teuer!

Target: Infineon XMC4500 (Relax Kit)

- 4x4 Matrix, Fließpunktarithmetik (einfache Rechengenauigkeit),
- Gauss-Zerlegung basierend auf LU Simulink Block
- Orthogonalisierung basierend auf QR Simulink Block



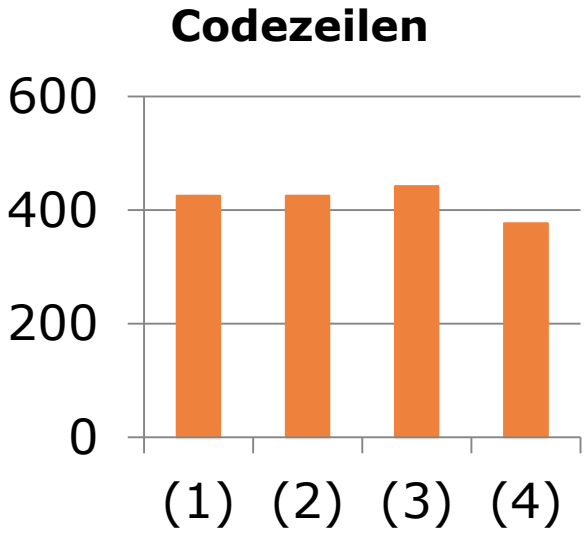


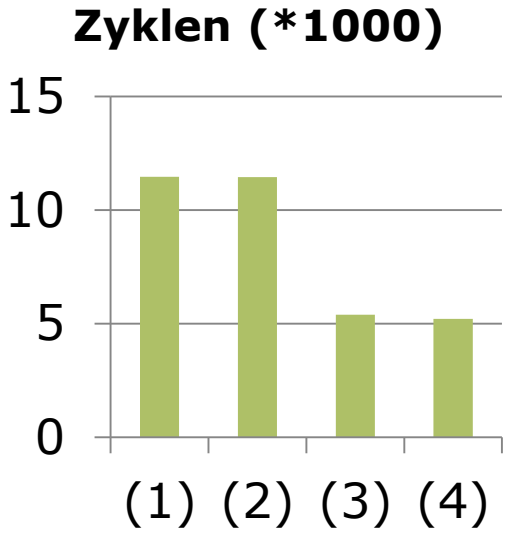


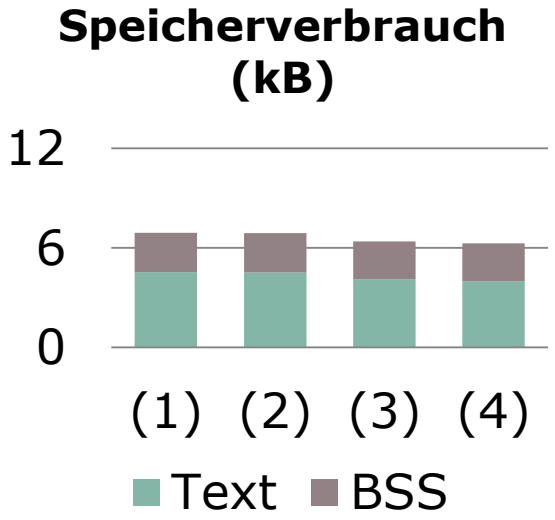
Vergleich verschiedener Implementierungen der Substitutionen: fertige Funktionen konkurrenzfähig

Target: Infineon XMC4500 (Relax Kit)

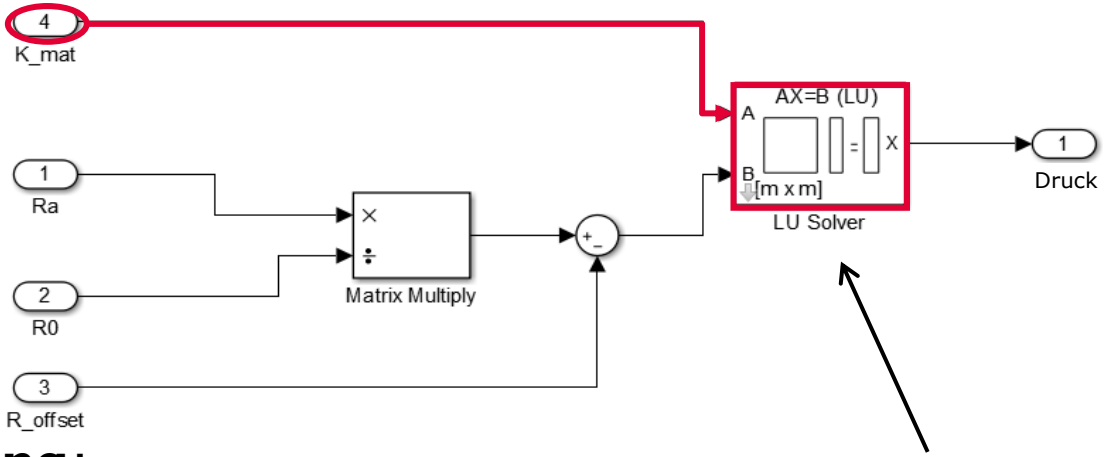
(1) LU solve ohne Permutation (nutzt intern doppelte Rechengenauigkeit)
(2) LU solve mit Permutation (nutzt intern doppelte Rechengenauigkeit)
(3) linsolve
(4) selbst geschriebene Substitutionen







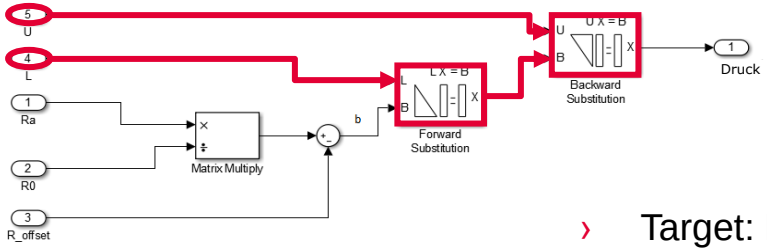
Einsparpotenzial: Konstante Matrix



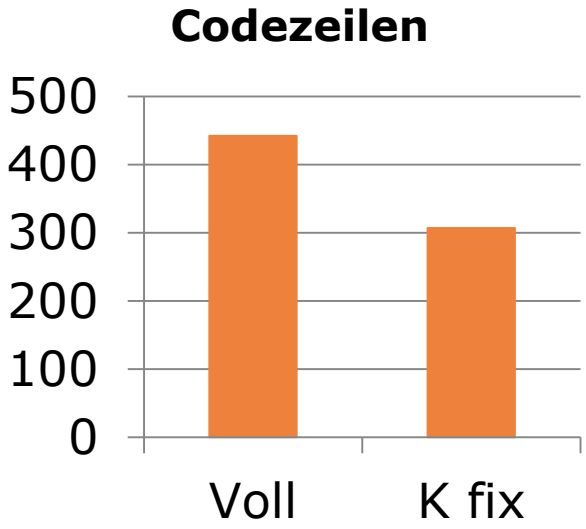
Beobachtung:

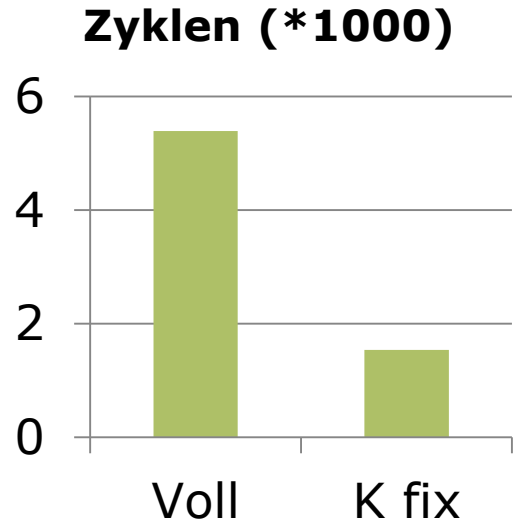
K konstant ➔ Zerlegung nur einmal berechnen

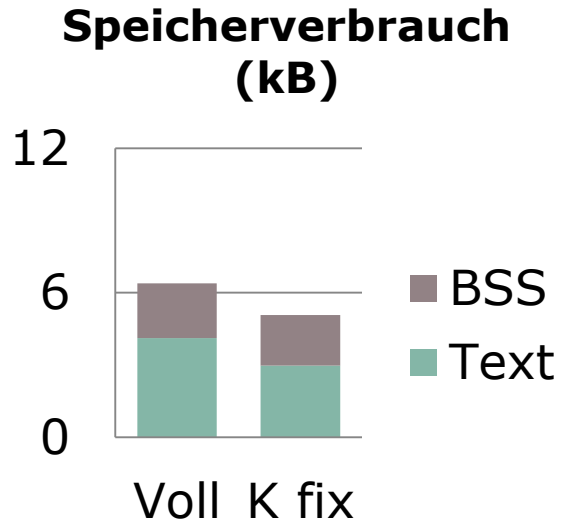
Deutlich weniger Rechenaufwand dank zerlegter Matrix



› Target: Infineon XMC4500 (Relax Kit)



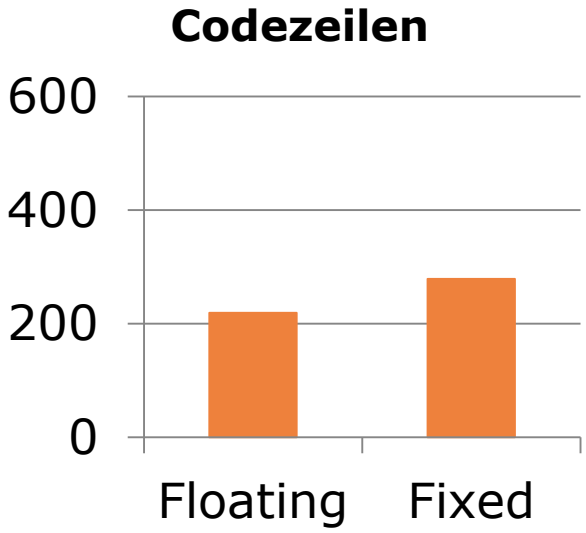


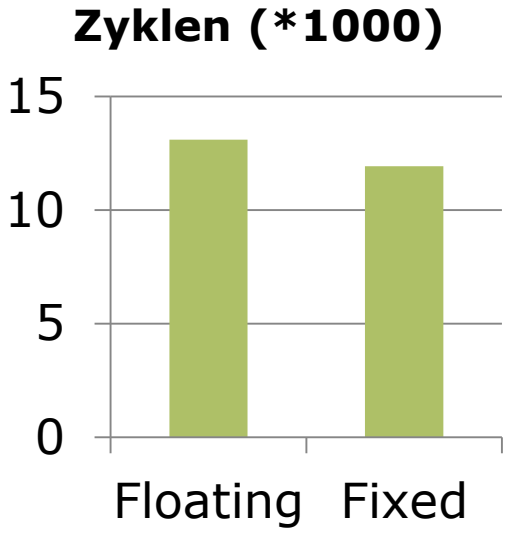


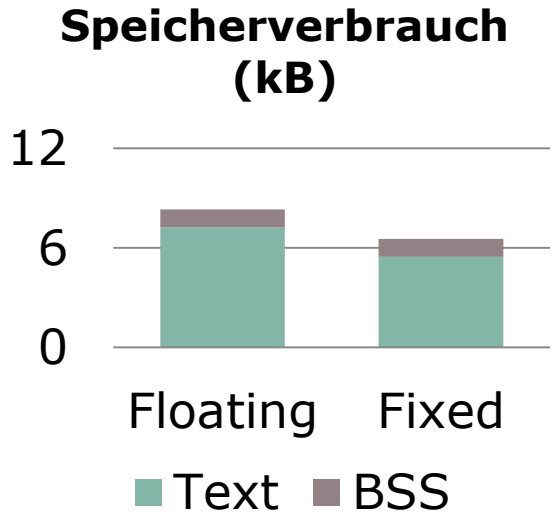
Konvertierung in Fixpunkt Arithmetik: Noch genau genug und kleiner!

Target: Infineon XMC1100 ARM M0 (XMC™™ 2GO)

› Alle Variablen : 32-bit Fixpunkt Datentypen (mit Vorzeichen)

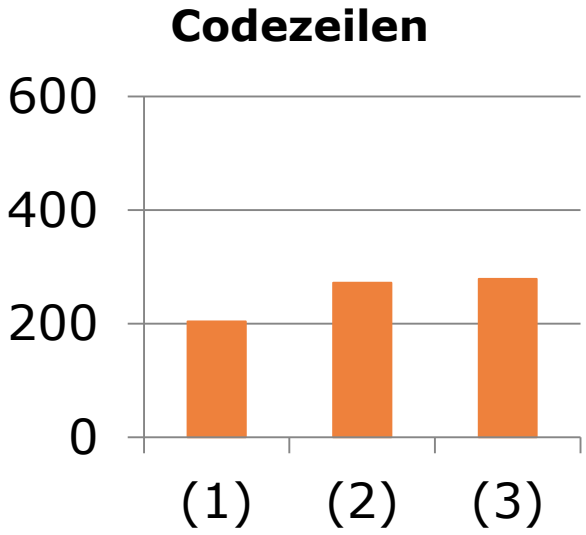


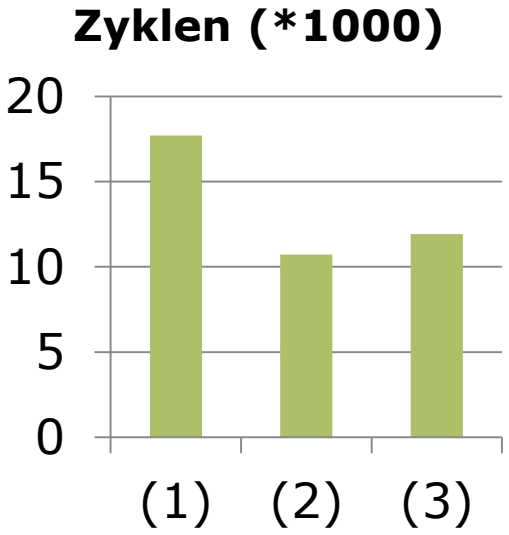


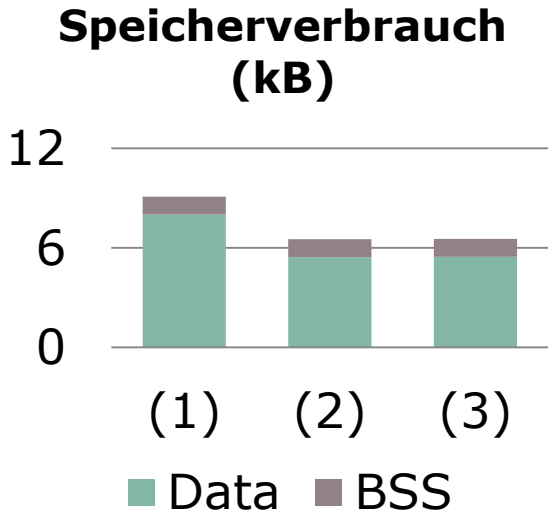


Die besten Kandidaten:
Aufgrund der teuren Division kann QR sich lohnen…

Target: ARM M0 XMC1100 (XMC™™ 2GO)
(1) QR Zerlegung mit Fließpunktarithmetik (emuliert)
(2) QR Zerlegung mit Fixpunkt Datentypen
(3) LR Zerlegung mit Fixpunkt Datentypen



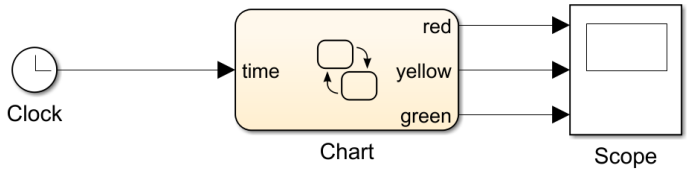


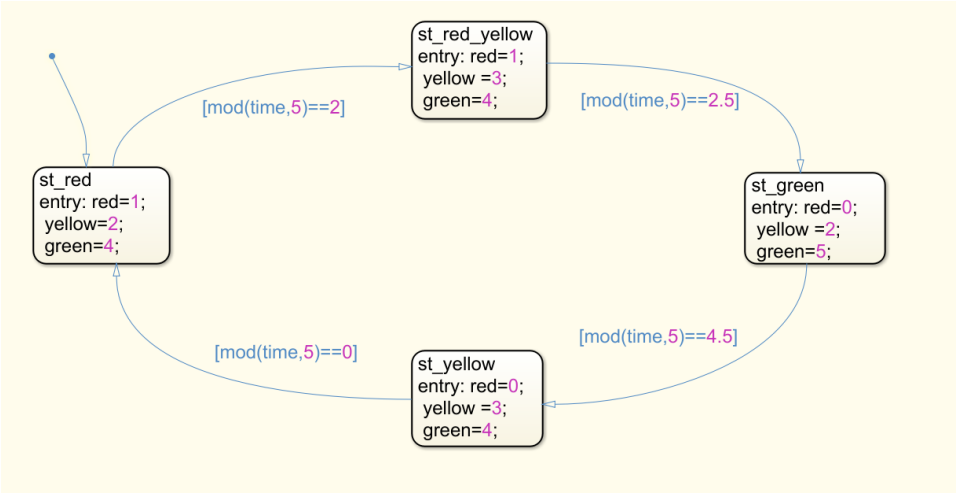


Agenda

1. Einführung Smarte Sensoren
2. Entwicklung der Auswertealgorithmen und Code Generierung
3. Erweiterungen des Modellbasierten Ansatzes
4. Zusammenfassung

Behandlung von Zustandsautomaten: Einfache Modelle können zu unerwünschtem Polling führen

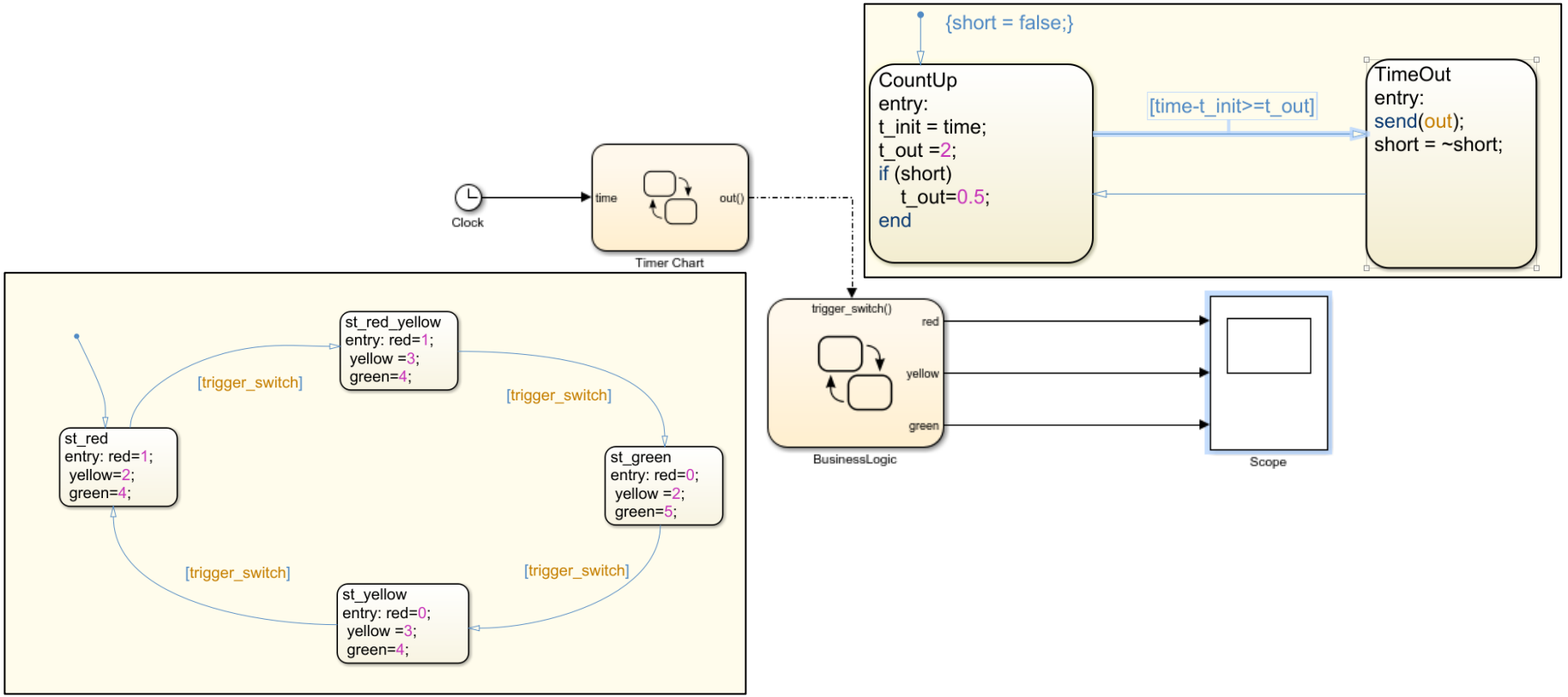





```
int_T main(int_T argc, const char *argv[])
{
  /* Unused arguments */

 /* Initialize model */
 TrafficLightChart_initialize();

 /* Attach rt_OneStep to a timer or interrupt service routine with
  * period 0.1 seconds (the model's base sample time) here.  The
  * call syntax for rt_OneStep is
  *
  *  rt_OneStep();
  */
```

```
/* Model step function */
void TrafficLightChart_step(void)
{
  real_T Clock;

  /* Clock: '<Root>/Clock' */
  Clock = TrafficLightChart_M->Timing.t[0];

  /* Chart: '<Root>/Chart' */
  /* Gateway: Chart */
  /* During: Chart */
  if (TrafficLightChart_DW.is_active_c3_TrafficLightChart == 0U) {
    /* Entry: Chart */
    TrafficLightChart_DW.is_active_c3_TrafficLightChart = 1U;

    /* Entry Internal: Chart */
    /* Transition: '<S1>:10' */
    TrafficLightChart_DW.is_c3_TrafficLightChart = TrafficLightChart_IN_st_red;

    /* Entry 'st_red': '<S1>:1' */
    /* '<S1>:1:1' red=1; */
```

Lösung: Ereignis getriggerter Ansatz bereits bei Modellierung ➔ direkt verwendbarer Code



Agenda

1. Einführung Smarte Sensoren
2. Entwicklung der Auswertealgorithmen und Code Generierung
3. Erweiterungen des Modellbasierten Ansatzes
4. Zusammenfassung

Zusammenfassung

- Modellbasierter Ansatz ermöglicht schnelle Entwicklung Smarter Sensoren
- Code Generation = Brücke zwischen Konzept und Firmware
  - schneller Vergleich vieler Varianten
- Passende Bibliotheksfunktionen liefern effizienten Code
  - gute Kenntnisse der Bibliotheken sehr hilfreich
- Schneller Transfer von Fließpunkt- zu Fixpunktarithmetik
- Endergebnis für lineare Gleichungssysteme überrascht
- Erweiterung aus Zustandsautomaten möglich, bedarf aber hinreichender Modellierungskenntnisse in Simulink

Part of your life. Part of tomorrow.

Infineon