SCHAEFFLER

Sichere Software im Antriebsstrang

Gastvortrag FAU SS 2017 – Verläßliche Echtzeitsysteme, 2017-07-13
Dr. Georg Drenkhahn, Schaeffler Technologies
Georg.Drenkhahn@schaeffler.com

Inhalt

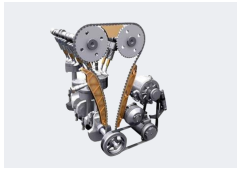
SCHAEFFLER

Schaeffler auf einen Blick

Zwei Sparten – Automotive und Industrie

SCHAEFFLER

Automotive (Systeme)


Motorsysteme


Getriebesysteme


Fahrwerksysteme


Hybride und elektrische Antriebssysteme

Industrie (Sektorencluster)


Wind

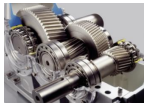

Aerospace


Offroad


Rail


Two-Wheelers


Raw Materials


Power Transmission


Industrial Automation

Strategie „Mobilität für morgen“

E-Mobilität – Lösungen für alle Fahrzeugklassen und Märkte

SCHAEFFLER

Komponenten und Systeme für Einstiegs-Hybridisierung

- E-Clutch für Komfort und CO_2- Einsparungen im Handschalter
- Optimierte Komponenten für 12V- und 48V-BAS-Systeme (Belt-Alternator-Starter) wie z. B. (schaltbarer) Riemenscheibenentkoppler und Pendelspanner


Pendelspanner

Hybridmodule für 48V- und Hochvoltanwendungen

- Positioniert zwischen Motor und Getriebe (P2)
- Ermöglicht je nach Auslegung und Zyklus CO_2-Einsparungen von ca. 15 % bis > 60 %
- Koaxiale oder achsparallele Bauform
- 48V (Mild HEV) oder Hochvolt (FHEV/PHEV)
- Integrierbar als Add-On oder optional mit integriertem Anfahrelement (Drehmomentwandler oder Doppelkupplung)


P2- Hybridmodul

E-Achsen für Hybrid- und reinelektrische Fahrzeuge

- Durch modularen Aufbau des Antriebselements an spezifischen Kundenanforderungen anpassbar
- Integrierbar als Achsgetriebe in 1- oder 2-gängiger Ausführung
- Koaxiales oder achsparalleles Design für optimale Bauraumnutzung
- Optional am Antriebsmodul montierte Elektromaschine ermöglicht Torque Vectoring


E-Achse

Radnabenantriebe

- Elektrischer Motor, Leistungselektronik, Bremse und Kühlsystem in der Felge verbaut
- Direkte Kraftübertragung auf die Straße und integrierte Regelung für mehr Agilität und Sicherheit
- Erhöhter Lenkeinschlag erlaubt Einparken in kleinste Parklücken
- Ermöglicht gänzlich neue Fahrzeugkonzepte


E-Wheel Drive

Inhalt

SCHAEFFLER

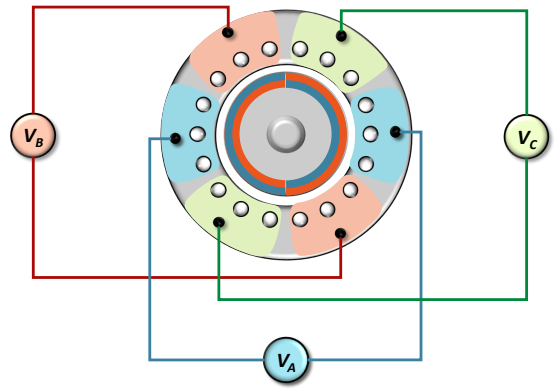
Funktionsweise eines Elektromotors

SCHAEFFLER

Fundamentale Bestandteile eines modernen Elektromotors:

- Rotor mit oder ohne Permanentmagnete
- Dreiphasige Windung im Stator

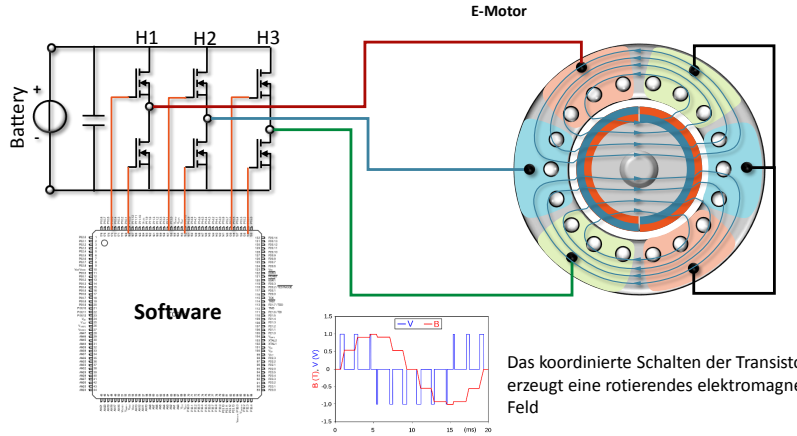
Wenn man diesen Motor an eine Batterie anschließt, wird er sich kaum drehen …



Software für die Ansteuerung der Leistungselektronik

SCHAEFFLER

Verwendung einer B6 Brücke mit 6 Leistungstransistoren

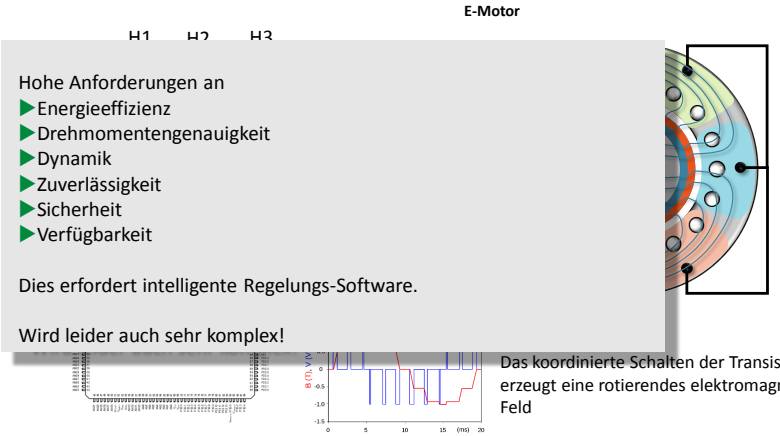


Das koordinierte Schalten der Transistoren erzeugt eine rotierendes elektromagnetisches Feld

Software für die Ansteuerung der Leistungselektronik

SCHAEFFLER

Verwendung einer B6 Brücke mit 6 Leistungstransistoren



Hohe Anforderungen an
- Energieeffizienz
- Drehmomentengenauigkeit
- Dynamik
- Zuverlässigkeit
- Sicherheit
- Verfügbarkeit

Dies erfordert intelligente Regelungs-Software.

Wird leider auch sehr komplex!

Das koordinierte Schalten der Transistoren erzeugt eine rotierendes elektromagnetisches Feld

Inhalt

SCHAEFFLER

Funktionale Sicherheit

SCHAEFFLER

Die **Funktionale Sicherheit** ist das Nichtvorhandensein von **inakzeptablen Risiken** verursacht durch **Gefahren für Leib und Leben** aus **fehlerhaften Systemfunktionen.**

Z.B. eine Bremse, die versagt, oder ein Lenkung, die blockiert.

Dies schließt den sicheren Betrieb auch bei

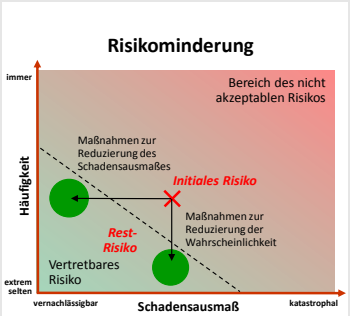
- Bedienfehlern
- Hard- und Software Fehlern
- veränderten Umgebungsbedingungen

ein.

FuSi ist nicht : elektrische Sicherheit, Brandschutz, … (wenn eigentliche Funktion nicht beeinflusst ist)

Wichtigste Normen:

- IEC 61508 "Functional Safety of [..] electronic safety related Systems": Sehr allgemein für alle Bereiche: Bahn, Industrie, Lust/Raumfahrt, AKWs, ….
- ISO 26262 "Road Vehicles – Functional Safety": für sicherheitsrelevante elektrische/elektronische Systeme in KFZ.



Ungewollte Beschleunigung

SCHAEFFLER

Funktionen des elektrischen Antriebs im Fahrzeugantriebsstrang:

- Erzeuge Antriebsmoment
  (Wandle elektrische Leistung in mechanische Leistung)
- Erzeuge Bremsmoment
  (Wandle mechanische Leistung in elektrische Leistung)

Gefahren und Risiken für das Fahrzeug:

- Unerwünschte Beschleunigung
- Unerwünschtes Bremsen
- Beschleunigung in falsche Richtung
- Antriebs oder Bremsmoment zu klein
- Antriebs- oder Bremsmoment zu spät
- …

Aus den **Gefahren** werden **Sicherheitsziele und Anforderungen** abgeleitet und einer **Sicherheitsfunktion** zugewiesen. Die Sicherheitsfunktionen können auch in der **Software** realisiert werden.

Den Gefahren wird auf Basis des **Schadensausmaßes**, der **Häufigkeit** und der **Beherrschbarkeit** ein **Safety Integrity Level** zugewiesen.

Ungewollte Beschleunigung ist kein exotisches Problem für Autos!

Auszug aus Wikipedia:
https://en.wikipedia.org/wiki/Sudden_unintended_acceleration

Reported incidents [edit source]

Reported incidents of sudden acceleration, include:

- 1987: The 1982-1987 Audi 5000s sales in the United States fell after recalls linked to sudden unintended acceleration. There were 700 accidents and 6 deaths.
- 1988: 1986 Honda Accords were documented to have had sudden acceleration incidents due to the Vehicle Speed Control component, as reported to the NHTSA.[17]
- 1997: Sudden acceleration in Jeep Cherokees and Jeep Grand Cherokees was reported by Diane Sawyer in a March 1997 ABC News *Primetime* segment.[18][19]
- 2000: Several Ford Explorers were reported about in the UK by a Channel 4 news program where the vehicle was already moving at speed and experienced sudden acceleration.[20]
- 2004: The National Highway Traffic Safety Administration (NHTSA) sent Toyota a chart showing that Toyota Camrys

…. (lange Liste)

Früher wurde das Drehmoment durch die Drosselklappenstellung bestimmt. Sicherheitsrelevante Elemente waren die mechanische Teile (Pedal, Zug, Drosselklappe, …)

Software als Ursache für Ungewollte Beschleunigung

SCHAEFFLER

Moderne Fahrzeuge: Drehmomentanforderungen werden in (zentralen) Steuergeräten ermittelt und über digitale Busse übertragen (Drive-By-Wire)
→ **Software**, Elektrik und Sensorik sind hochgradig **sicherheitsrelevant**

Prominentes Beispiel: Unfall mit Toyota Camry 2007 mit einem Todesopfer, Schadenersatzklage "Bookout vs Toyota"

Im Rahmen des Prozesses konnten Experten (NASA Engineering and Safety Center, M. Barr, P. Koopman) den Quellcode des der "Electronic Throttel System" Software einsehen.

Gefundene Mängel im Electronic Throttle Control System (ETCS):

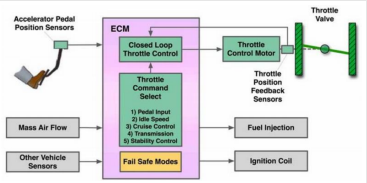
HW Plattform:

- RAM der CPU (V850) war nicht durch EDAC geschützt

SW Architektur, Design und Implementierung:

- Keine durchgängige redundante oder abgesicherte Speicherung kritischer Variablen
- Unentdeckter Stack Overflow möglich, kein Safety OS
- Nur 11 von 127 Regeln aus den MISRA Coding Guidelines wurden angewendet.
- Sehr viele Verletzungen im Code: unsichere Type Casts, Seiteneffekte, nicht initialisierte Variablen, …
- Funktionen mit Zyklomatischer Komplexität > 50
- Globale Variablen mit konkurrierenden Zugriffen, Rekursion

http://embeddedgurus.com/barr-code/2013/10/an-update-on-toyota-and-unintended-acceleration



Wahrscheinliche Ursache:

1. Durch Rekursion ausgelöster Stack Overflow
2. Überschreiben von OSEK OS Daten zur Task Aktivierung
3. Deaktivierung des zyklischen Tasks, in dem die Berechnung des Drosselventilwinkels **und** das Auslösen der Fail Safe Reaktionen umgesetzt war.
4. Watchdog wurde durch Interrupt bedient → fehlende Detektion des ausgefallenen Task
5. Drosselklappe bleibt weit geöffnet stehen

Common Cause Fehler (Speicher Korruption) & Kein Fault Containment

Finanzielle Kosten für Toyota für alle "Ungewollten Beschleunigung" Ereignisse:

Kosten für Gerichtsverfahren + Kosten für Fahrzeugrückrufe + Strafzahlungen wegen Verstöße gegen US Gesetze zur Fahrzeugsicherheit + Schadenersatz für Opfer + Vergleich mit dem US Dep. of Justice zur Einstellung weiterer Untersuchungen

→ mehr als 1.2 Milliarden $

Inhalt

Wie macht Software das Antriebssystem sicher?

SCHAEFFLER

System- und SW-Architektur

Redundanz-Pattern: Zu teuer im Automobil

Systeme im Fahrzeug sind meistens Fail-Safe. D.h. im ausgeschalteten Zustand ungefährlich.

Z.B. Ausfall des Antriebsmotors: Fahrer kann an den Straßenrand rollen und (mechanisch) bremsen. (Meist) beherrschbare und sichere Situation.

Funktionale Sicherheit wird erreicht, wenn sicherheitsrelevante Abweichungen erkannt / diagnostiziert werden können und die Systemfunktion ausgeschaltet werden kann:

- Zuverlässige Fehlererkennung
- Zuverlässige Abschaltmechanismen

Nicht FuSi relevant aber trotzdem wichtig: keinen unnötigen Abschaltungen. Verfügbarkeit!

Übliches Architektur Muster: **Monitor – Aktuator Pattern**

Normalerweise Aktuator und Monitor auf unterschiedlichen Prozessoren. In der Autoindustrie hat sich ein abgewandeltes (billigeres) Muster etabliert, das
E-Gas Konzept, ein standardisiertes Überwachungskonzept für Benzin und Diesel Motorsteuerungen

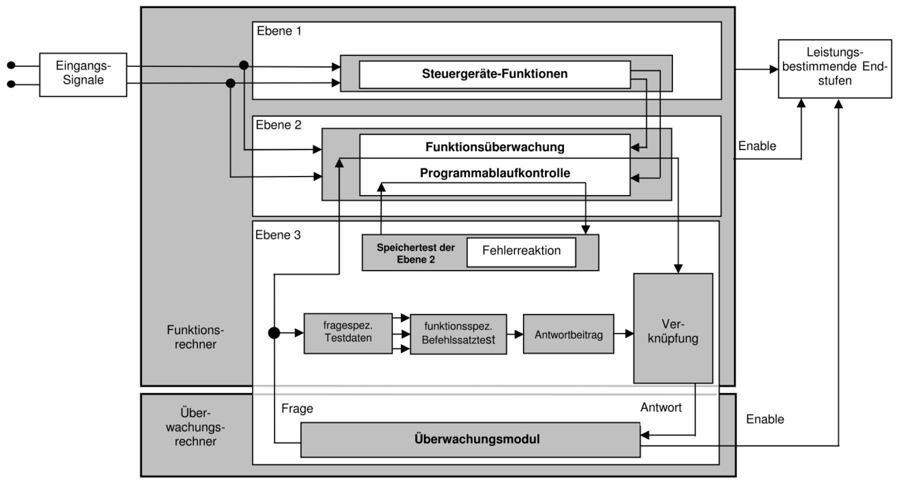
Erstellt von Arbeitskreis von VW, Audi, Daimler, BMW, Porsche + Zulieferer, erstes Release 2002

Ausgelegt für den Ersatz des Gaspedal-Seilzugs durch eine elektronische Lösung (elektrische Drosselklappe) in Verbrennungsmotoren. Kann aber konzeptionell auf die Überwachung von Elektromotoren übernommen werden.

Öffentlich verfügbar:
https://www.iav.com/publikationen/technische-veroeffentlichungen/e-gas-monitoring-concepts

Ebenenstruktur des E-Gas Konzeptes

SCHAEFFLER



Sicherheits-Integritätslevel

SCHAEFFLER

Einstufung von E/E oder SW Systemen bezüglich **Sicherheitsfunktionen** und **Zuverlässigkeit**

Industrienormen wie ISO26262 "Road vehicle – Functional safety " definieren Kriterien zur Beurteilung von Integritätsleveln:

- Zunehmende Kritikalität: QM → ASIL A → ASIL B → ASIL C → ASIL D

Gefährdungsbeurteilung für jede identifizierte Fehlfunktion (z.B. ungewollte Beschleunigung) basierend auf

- Betriebssituationen und deren Häufigkeiten bestimmen (z.B. Fahrt auf Landstraße → häufig = E4)
- Reaktionsmöglichkeit und deren Beherrschbarkeit durch den Fahrer bewerten (z.B. Bremsen → beherrschbar = C1)
- Bewertung des Schadensausmaß wenn keine Reaktion erfolgt bzw. möglich (z.B. Schwerer Unfall mit lebensbedrohlichen Verletzungen möglich = S3)

(siehe Vorlesung Teil 3)

Schadens-ausmaß-klasse	Häufigkeits-klasse Betriebs-Situation	Kontrollierbarkeits-/Beherrschbarkeitsklasse		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Eine zentrale system- und SW-architektonische Maßnahme zur Erreichung eines hohen ASIL ist die **Zerlegung** des Systems in mehrere **redundante** und **rückwirkungsfreie** Teilsysteme, die das gleiche Sicherheitsziel sicherstellen. (siehe Vorlesung Teil 4)

z.B. Gesamtsystem mit ASIL B → Teilsystem #1 mit ASIL A + Teilsystem #2 mit ASIL A

Rückwirkungsfreiheit erfordert **räumliche** und **zeitliche Isolation:**

Sicherzustellen durch Trennung in HW, z.B. Aufteilung auf verschiedene Cores und Schutz durch MPU, und/oder in SW durch Safety Betriebssysteme

→ kostenmotivierte Integration von mixed-ASIL SW auf gleiche Rechnerkerne

Sicherheitsmaßnahmen auf Design- und Implementierungsebene

Etablierte **Sicherheitsmaßnahmen** und **Design Muster** für die Erkennung von HW- und SW-Fehlern.

Teil der ISO 26262 Norm und damit "Stand der Technik". Im Rahmen der Produkthaftung zwingend zu betrachten.

- Watchdog mit separater Zeitbasis und Zeitfenster
- Überwachung des logischen Programmflusses
- Erkennung von Informationsänderungen im unveränderlichen Speicher
  - Blocknachbildung
  - Speichersignaturen auf Blöcken
- Erkennen von Fehlern beim Adressieren, Schreiben, Speichern und Auslesen des variablen Speichers
- Erkennen von Ausfällen in der Informationsübertragung
  - Vollständige Hardware-Redundanz
  - Übertragungsredundanz
  - Informationsredundanz
  - Frame-Zähler
  - Timeout-Überwachung
- Stapel-Over-/Under-Flow-Erkennung
- Integrierte Hardware-Konsistenzüberwachung
- Selbsttest durch Software
- Selbsttest unterstützt durch Hardware
- HW-Redundanz (z.B. Dual-Core)
- Konfigurationsregistertest
- RAM-Musterprüfung

Praktische Umsetzung von Sicherheitsmaßnahmen (1)

Sicherheitsmaßnahme	Konkretisierung
Watchdog mit separater Zeitbasis und Zeitfenster	CPU externer Watchdog, eigener Quarz und verschiedene Spannungsüberwachungen
Überwachung des logischen Programmflusses	Definition von Scheduling Randbedingungen, Überwachung der Task Aktivierungsreihenfolge durch einen Safety Watchdog in SW
Erkennung von Informationsänderungen im unveränderlichen Speicher	
Blocknachbildung	Wird meist schon auf Ebene des NVRAM Stacks umgesetzt
Speichersignaturen auf Blöcken	CRC Prüfsummen, im NVRAM Stack oder in Anwendung
Erkennen von Fehlern beim Adressieren, Schreiben, Speichern und Auslesen des variablen Speichers	MPU basierter Speicherschutz, Einsatz von ECC RAM (ohne HW-Unterstützung sehr schwer zu realisieren)

Praktische Umsetzung von Sicherheitsmaßnahmen (2)

Sicherheitsmaßnahmen	Konkretisierung
Erkennen von Ausfällen in der Informationsübertragung	
Vollständige Hardware-Redundanz	z.B. über 2 Busse, oder Sensorwertübertragung über SPI Bus (digital) und analog und ADC
Übertragungsredundanz	Zyklische und mehrfache Übertragung der gleichen Werte
Informationsredundanz	Unterschiedlich kodierte Kopien von Werten, CRC (siehe Vorlesung Teil 5)
Frame-Zähler	Übertragung eines Zählers zusammen mit der Payload zur Erkennung von doppelten oder fehlenden Nachrichten
Timeout-Überwachung	Überwachung der erwarteten Zykluszeit bei periodischen Nachrichten

Praktische Umsetzung von Sicherheitsmaßnahmen (3)

Sicherheitsmaßnahmen	Konkretisierung /Kommentar
Stapel-Over-/Under-Flow-Erkennung	Überwachung durch das Betriebssystem, High Water Marks oder per MPU Hardware; erkennt aber keine Speicherschreiber innerhalb des Stacks
Integrierte Hardware-Konsistenzüberwachung	z.B. Traps bei Illegalen Op Codes oder Div By Zero
Selbsttest durch Software	Z.B. Testweise Auslösung eines externen Wdg Bausteins, um Funktionsfähigkeit des Abschaltpfads zu prüfen. Testweises Auslösen von EDC Speicherschutz.
Selbsttest unterstützt durch Hardware	Build In Self Test (BIST) durch Safety Prozessoren, wird beim Prozessor Start durchgeführt. Testet z.B. RAM, ROM, ALU, …
HW-Redundanz (z.B. Dual-Core)	Lock Step Cores, Mehrfache Sensoren, Peripherie, … (Kosten!) (siehe Vorlesung Teil 5)
Konfigurationsregistertest	Konfiguration der MPU/Wdg/andere kritische Peripherie beim Startup, Zyklisches rücklesen der Werte und Vergleich mit Erwartungswert. Verhindert unabsichtliches abschalten von Überwachungsfunktionen.
RAM-Musterprüfung	Schreiben und Lesen von Testmustern im Hintergrund, Detektierung von Stuck-at Fehlern im RAM; notwendig für nicht-EDC RAM

Praktische Umsetzung

SCHAEFFLER

In der Paxis: **Don't reinvent the wheel!**

Wenn sicherheitsrelevante SW Funktionen umgesetzt werden:

- Verwendung einer **qualifizierten/zertifizierten Prozessor HW Plattform**
  EDC RAM, Memory protection unit (MPU), lock step cores, Peripherie mit elektrischen Selbsttests.
  Alle großen Automotive Prozessorhersteller haben Prozessoren mit den üblichen HW Schutzmechanismen im Angebot.
- Nutzung von Software des Halbleiterherstellers für die "Selbsttest durch Software". Nur Aufwand für die sichere Integration dieser SW.
- Nutzung von zum Prozessor gehörigen **Überwachungs-ICs**
  Realisiert Spannungsüberwachung, ext. Watchdog, Abschalttests
- Nutzung von **Basic Software für sicherheitsrelevanten Einsatz**:
  - Betriebssystemen für Safety Anwendungen
    Realisiert isolierte Ausführung von Software, zeitliche und räumliche Überwachung
  - Watchdog Stack: zeitlichen und logischen Ablaufüberwachung
  - End-2-End Protection: Absicherung von Datenkommunikation über CRC, Zähler, Timeout Überwachung

Design von zuverlässiger Embedded Software ist immer ein **Hardware-Software-Codesign** und setzt ein gutes **Systemverständnis** bei SW Entwicklern voraus.

ISO26262: Maßnahmen in Requirements, Architektur und Design

SCHAEFFLER

Requirements

Im Wesentlichen abgesichert durch Schritte im Entwicklungsprozess, u.a.:

- Regeln für die saubere sprachliche Formulierung
- Validierung durch Reviews

Architektur uns Design

- Natürlichsprachliche Beschreibung
- (Halb-)Formale Notation (z.B. mit UML Modellen)
  Verhindert Missinterpretation von informellen Grafiken
- Modularität und Kapselung
  - Hierarchisch gegliedert
  - Kohäsion und lose Kopplung
- Einfachheit
  - Überschaubare Größen von Komponenten und Interfaces
  - Einfaches Scheduling und wenige Interrupts

Komplexität bedeutet

- Fehleranfälligkeit
- Nachweis der Korrektheit sehr schwierig, zB für die Zeitanalyse (siehe Vorlesung Teil 9)

SW Design

Mechanismen zur **Fehlererkennung** einbauen: Wie kann man erkennen, das etwas schief läuft?

- Design by Contract und Prüfung der Kontrakte zur Laufzeit z.B. Range checks in Pre- und Post-Conditions (siehe Vorlesung Teil 10)
- Plausibilitätsprüfungen für Daten (zeitlicher Verlauf, gegen Redundante Informationen)
- Bedingungen für das Control Flow Monitoring ermitteln
- Diversität (Leute, Tools, Programmiersprachen, ...; sehr teuer)

Mechanismen zur **Fehlerbehandlung**: Einfach abschalten gehen nicht immer

- Recovery, z.B. durch Wiederholung
- Degradierung, Aufrechterhaltung der nicht betroffenen Teilfunktionen
- Parallele Redundanz: Wenn 2 SW Teile die gleiche Funktion erfüllen
- Error Correcting Codes. Datenredundanz nutzen.

Methoden für die **SW Design Verifikation**

- Reviews, Walk-Through, Inspektion
- Simulation, Insbesondere bei Modellbasierter SW
- Prototyping
- Formale Verifikation (sehr aufwändig → siehe Vorlesung Teil 8)
- Daten- und Controllflussanalyse (Design muss sehr detailliert sein, auch sehr aufwändig)

Große Kunst im Projekt: eine **sinnvolle Auswahl und Kombination** der Maßnahmen zusammenstellen. So wenig wie möglich, so viel wie nötig!

Design/Implementierung

SCHAEFFLER

SW Implementierung

- Funktionen mit nur einem Eingang und Ausgang
- Keine dynamischen Objekte (kein Heap)
- Variablen Initialisierungen
- Namensregeln
- Vorsichtiger Einsatz von globale Variablen
- Limitierter Gebrauch von Pointern
- Typen Konvertierung, keine Casts
- Keine Sprünge ("gotos")
- Keine Rekursion

DER Codierrichtlienestandard in der Automobilindustrie:

MISRA-C Guidelines for the use of C language in critical systems

- Verbietet Konstrukte, die anfällig sind für Programmierfehler.
- Regelsatz für C Programmierer und für C Code Generatoren.

Die meisten Regeln können durch kommerziellen Tools gefunden werden

- gehört zur statischen Verifikation (ohne Code Ausführung)
- Automatisierbar und messbar

Abschnitte MISRA Regeln
8.1 A standard C environment
8.2 Unused code
8.3 Comments
8.4 Character sets and lexical conventions
8.5 Identifiers
8.6 Types
8.7 Literals and constants
8.8 Declarations and definitions
8.9 Initialization
8.10 The essential type model
8.11 Pointer type conversions
8.12 Expressions
8.13 Side effects
8.14 Control statement expressions
8.15 Control flow
8.16 Switch statements
8.17 Functions
8.18 Pointers and arrays
8.19 Overlapping storage
8.20 Preprocessing directives
8.21 Standard libraries
8.22 Resources

Beispiel für Programmierfehler und MISRA Regel:

```
int a = 111;
int b = 011;
int c = 001;
printf("%d", a+b+c);
```

Wie lautet das Ergebnis?

Ergebnis: 121
011 == 9

Rule 7.1 Octal constants shall not be used

Noch ein Beispiel aus den MISRA Coding Rules

SCHAEFFLER

```
int            siA = -9;
unsigned char  ucB =  6u;
unsigned short usB =  6u;
unsigned int   uiB =  6u;

if (siA + ucB < 4)
  printf("-9 + 6 = -3 < 4\n");
else
  printf("What the f***?\n");

if (siA + usB < 4)
  printf("-9 + 6 = -3 < 4\n");
else
  printf("What the f***?\n");

if (siA + uiB < 4)
  printf("-9 + 6 = -3 < 4\n");
else
  printf("What the f***?\n");
```

Wie lautet das Ergebnis?

Ergebnis auf **32bit** Plattform:

-9 + 6 = -3 < 4
-9 + 6 = -3 < 4
What the f***?

Ergebnis auf **16bit** Plattform:

-9 + 6 = -3 < 4
What the f***?
What the f***?

Kurzerklärung:

Fall 1: (int) + (unsigned char)
1. Integer promotion → (int) + (int)
2. Usual arithmetic conversions → (int) + (int)
3. Kleiner-Als Operator: (int) < (int)

Fall 2: (int) + (unsigned short)
1. Integer promotion → 16bit: (int) + (unsigned int); 32bit: (int) + (int)
2. Usual arithmetic conversions → 16bit: (unsigned int) + (unsigned int); 32bit: (int) + (int)
3. Kleiner-Als Operator:
   16bit: (unsigned int) < (int) → (unsigned int) < (unsigned int); 32bit: (int) < (int)

Fall 3: (int) + (unsigned int)
1. Integer promotion → (int) + (unsigned int)
2. Usual arithmetic conversions → (unsigned int) + (unsigned int)
3. Kleiner-Als Operator: (unsigned int) < (int) → (unsigned int) < (unsigned int)

Details sind im C Standard unter "Integer promotion" und "Usual artithmetic conversion" zu finden. (verwirrend und kompliziert)

Rule 10.4 Both operands of an operator in which the *usual arithmetic conversions* are performed shall have the same *essential type category*

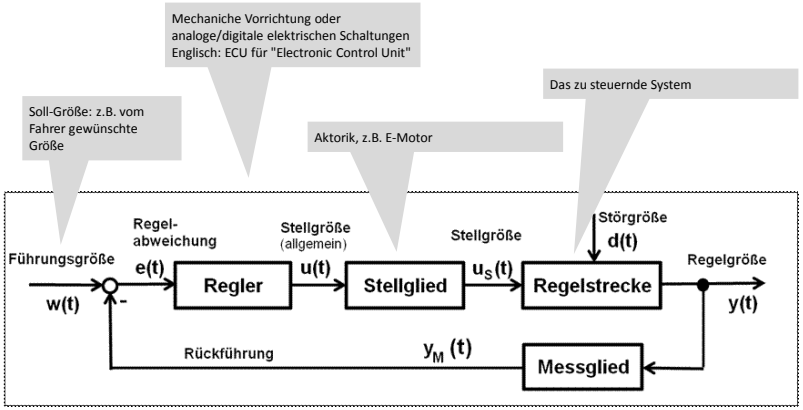
Inhalt

SCHAEFFLER

Steuergeräte (ECUs)

SCHAEFFLER

Steuergeräte in mechatronischen Systemen führen meist Aufgaben der Regelungtechnik aus



Applikationsmodellierung

SCHAEFFLER

Simulink ermöglicht die Erstellung von **Blockschaltbildern** = Graphische Standardnotation für regelungstechnische Aufgaben

Graphische Notation ist Repräsentation von konkreten mathematischen Operationen

- Blöcke/Subsysteme stehen für (mathematische) Operationen/Funktionen
- können hierarchisch tief verschachtelt sein

Ein/Ausgangsdatenflüsse werden durch Linien und Pfeile dargestellt.

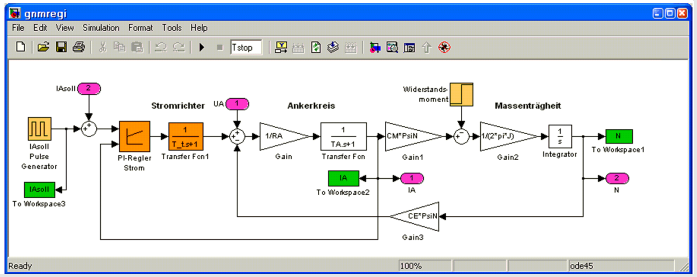
Werkzeug wandelt Blockdiagramm in Differenzen/Differentialgleichungssystem um, das numerisch integriert wird.

Simulation von Sensoren, Aktoren, Regler und Regelstrecke selbst

→ Modelle können ausgeführt werden: viele Analysemöglichkeiten durch die komplette Simulation.

Dominantes Quasi-Standard Werkzeug in der modellbasierten ECU Software Entwicklung:
MATLAB/Simulink/Stateflow
(The MathWorks)

Unter "Modell" wird in diesem Kontext fast immer ein Simulink Modell verstanden.



Modellbasiere SW Entwicklung

SCHAEFFLER

Neben der Analyse durch Simulation auf PC kann der Regler selbst für die Codegenerierung verwendet werden.

Notwendige Zusatzinformationen für die Codeerzeugung

- Definition der Funktionen (Namen, Funktionsprototypen)
- Definition der Datenspeicher (z.B. globale Variablen oder Zugriffsfunktionen)
- Spezifische Einstellungen, z.B. für Optimierungen

Modellbasierte Entwicklung verdrängt zunehmend Source Code Erstellung von Hand im regelungstechnischen Bereich.
Das Modell ist mehr als eine "ausführbare Spezifikation".

Die Regeln für die sichere SW Entwicklung müssen entsprechend angepasst werden:

- Coding Guidelines → Modeling Guidelines
  Coding Guidelines bleiben wichtig für die Hersteller von Code Generatoren
- Quell-Code Metriken → Modell Metriken
  Quellcode muss nicht mehr händisch gepflegt werden und Code Komplexität ist kein Problem für Code Generatoren
- Line/Branch/MCDC Abdeckung → Modell Abdeckung

Modeling Guidelines

- Vermeidung von fehleranfälligen Konstrukte
- Wartbarkeit
- Lesbarkeit
- Testbarkeit
- Geeignet zur Code Generierung

verschiedene Guideline Quellen, z.B. vom "MathWorks Automotive Advisory Board"
https://de.mathworks.com/solutions/automotive/standards/maab.html

Modell-Metriken

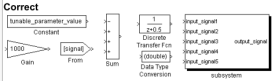
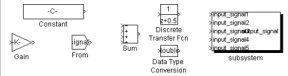
- Grundlage: Halstead-Metrik: #Operatoren, #Operanden, #Unterschiedliche Operatoren, #Anzahl unterschiedliche Operanden
- Für Modelle: #Blöcke, #Blockeingänge, #Blocktypen, #Blockausgänge

Junges Forschungsfeld: Know-How und Werkzeuge haben keine vergleichbar lange Historie wie für handgeschriebene SW

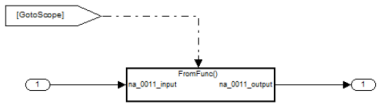
Modeling Guidelines --- Beispiele

SCHAEFFLER

7.1.6. jm_0002: Block resizing

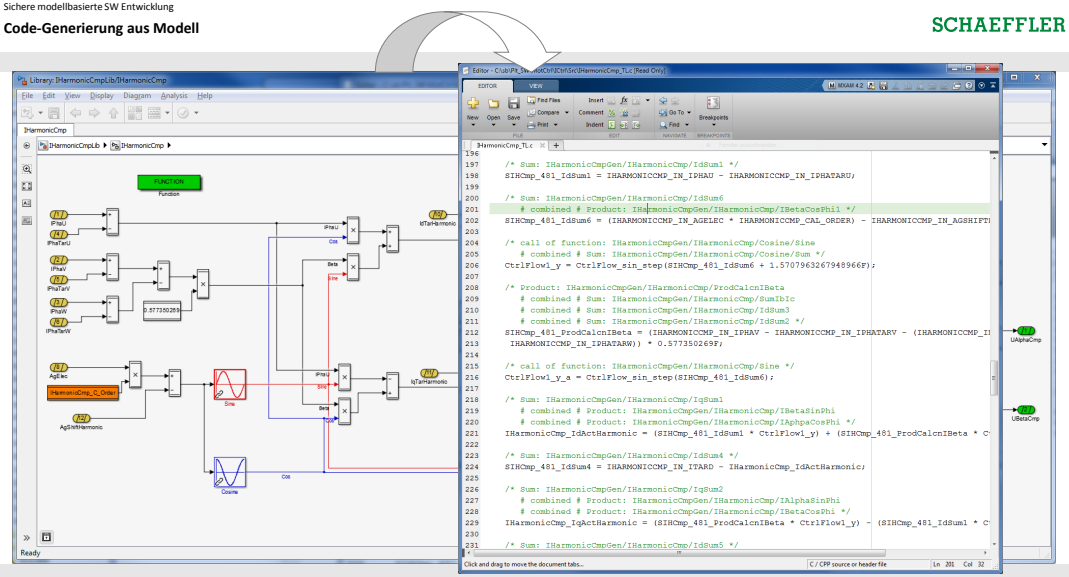
ID: Title	jm_0002: Block resizing
Priority	mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	All blocks in a model must be sized such that their icon is completely visible and recognizable. In particular, any text displayed (for example, tunable parameters, filenames, or equations) in the icon must be readable. This guideline requires resizing of blocks with variable icons or blocks with a variable number of inputs and outputs. In some cases, it may not be practical or desirable to resize the block icon of a subsystem block so that all of the input and output names within it are readable. In such cases, you may hide the names in the icon by using a mask or by hiding the names in the subsystem associated with the icon. If you do this, the signal lines coming into and out of the subsystem block should be clearly labeled in close proximity to the block. **Correct**  **Incorrect** 
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

7.3.5. na_0011: Scope of Goto and From blocks

ID: Title	na_0011: Scope of Goto and From blocks
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	For signal flows, the following rules apply: • From and Goto blocks must use local scope. Note: Control flow signals may use global scope. Control flow signals are output from: • Function-call generators • If and Case blocks
	
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☑ Simulation

Code-Generierung aus Modell

SCHAEFFLER



Zusammenfassung

SCHAEFFLER

- Software im Antriebsstrang übernimmt anspruchsvolle und sicherheitsrelevante Funktionen
- Schlechte Software ist teuer und gefährlich
- Betrachtungen zur Zuverlässigkeit und Sicherheit verlangt von SW-Architekten Verständnis für das Gesamtsystem und für die ECU Hardware
- Architektur und Design von verlässlichen softwareintensiven Systemen ist immer ein HW/SW-Codesign
- Der "Stand der Technik" an etablierten SW Maßnahmen zur Realisierung von Sicherheitsmaßnahmen in Automotive SW ist in den letzten Jahren stark gestiegen.
- Modellbasierte SW Entwicklung erfordert angepasstes Vorgehen zur Verifikation von ECU SW

SCHAEFFLER

Danke!