Implications of microcontroller software and tooling on safety-critical automotive systems

Kai Konrad (Dipl. Ing. (FH), MBA)
Infineon Technologies AG
April 2008

Kai.Konrad@Infineon.com
## Table of contents

- Functional safety applied to microcontrollers
- Software as a major issue for functional safety
- Functional safety software partitioning
  - Functional dependent safety
  - Functional independent safety
- Conclusion
Primary causes of system failure - Industry as example

- Specification 44.10%
- Design & Implementation 14.70%
- Operation & Maintenance 14.70%
- Design Changes 20.60%
- Installation & Commissioning 5.90%

Over 80% of all failures are caused before any user touches installations.

Only 15% of failures happen during operation and maintenance.

Source: HSE UK report 1999, based on industrial accidents based on 34 incidents.
A system is ‘Safe’ if there is a ‘Tolerable Risk’ when it is in operation

**Functional Safety**

“Part of the overall safety which depends on the correct functioning of safety-related systems for risk reduction. Functional safety is achieved, when every safety function is performed as specified”
Automotive system safety includes:

- Sensors
- Processing
  - Hardware
  - Software
- Actuators
- Interconnections
- Energy Supply
Process standardization as key to define functional safety future

- Industrial Standard for functional safety **IEC61508** emerging as a guideline for current automotive systems
  - Defining Safety Integrity Levels (SIL) 1 to 4
  - Interesting for Automotive are levels up to SIL3

- New Automotive Standard ISO26262 currently under preparation
  - Will be designed for automotive systems needs
  - Defining Automotive Safety Integrity Levels ASIL A to D

Motivation for standardization
- Common process over automotive industry
- Legal protection
- State-Of-The-Art definition
- Higher system quality
SIL3 as SYSTEM safety accreditation standard

Means:

- Probability of dangerous failure < $10^{-7}$ per hour of operation
  (100 FIT = 100 Failure In Time)

- Possible failure modes leading to a dangerous system state of < 1% of total system operational envelope
  (99% Safe Failure Fraction = SFF)

Requires:

- Detailed system and component fault analysis (FMEA)

- High component quality

- Strict design and integration methodology, documentation, application support

Example: A person who lives to 80 years old has 701280 hours of life before their ‘dangerous failure’

Equivalent to 0.7ppm  SIL 2
What does IEC 61508 SIL 3 mean when applied to a microcontroller?

**Microcontroller + Safety-Driver + Application Functional Safety Software** has to meet:

- Safe Failure Fraction (SFF) > 99%
- (Covers > 99% of used silicon!!!)
- Probability of failure per hour (PFH) <<10^-7

Infineon TriCore® TC1796

![State of the Art Microcontroller Diagram](image-url)
## Hardware errors in semiconductor as cause of dangerous systems faults

<table>
<thead>
<tr>
<th>Error Case</th>
<th>Statistical/Transient Error</th>
<th>Systematic/Static Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Behavior</td>
<td>Short Temporal Duration</td>
<td>Permanent Nature</td>
</tr>
<tr>
<td>Potential Causes</td>
<td>ESD</td>
<td>ESD, EMI</td>
</tr>
<tr>
<td></td>
<td>EMI</td>
<td>Electrical / Mechanical Overstress</td>
</tr>
<tr>
<td></td>
<td>Radiation</td>
<td>Specification Errors</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Hardware and Software Bugs (Common Mode Errors)</td>
</tr>
<tr>
<td>Measurement</td>
<td>FIT Rate Determination (e.g. Experimental)</td>
<td>PPM Rate Estimation (e.g. Field Experience)</td>
</tr>
</tbody>
</table>

### Potential Causes
- ESD
- EMI
- Radiation
- ...
# Table of contents

- Functional safety applied to microcontrollers
- Software as a major issue for functional safety
- Functional safety software partitioning
  - Functional dependent safety
  - Functional independent safety
- Conclusion
Software development as critical issue for system safety

- Is it possible to write software without bug’s???
- After initial coding you can expect one bug per 20 lines of code
- After thorough unit testing you can expect 1 bug per 1000 lines of code in the final release
  - 1 line ~5 bytes, so 1 bug per ~5KB

<table>
<thead>
<tr>
<th>Application</th>
<th>Microcontroller Type</th>
<th>Code Size</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steering Angle Sensor</td>
<td>8 Bit</td>
<td>32KB</td>
<td>7 Bugs</td>
</tr>
<tr>
<td>Low-end Sensor Cluster</td>
<td>16 Bit</td>
<td>128KB</td>
<td>26 Bugs</td>
</tr>
<tr>
<td>Airbag Controller</td>
<td>16/32 Bit</td>
<td>256KB</td>
<td>52 Bugs</td>
</tr>
<tr>
<td>EPS Controller</td>
<td>16/32 Bit</td>
<td>512KB</td>
<td>104 Bugs</td>
</tr>
<tr>
<td>Central Chassis Controller</td>
<td>32 Bit</td>
<td>1.5MB</td>
<td>308 Bugs</td>
</tr>
</tbody>
</table>
Today's automotive software partitioning as critical issue

AutoSAR* Operating System
Run-Time Environment
Drivers, Communication
Microcontroller Abstraction Layer

Microcontroller (e.g. TriCore®)

Safety Driver

Supplied by
OEM

Independent
Software
Company

Supplied by
TIER1

Independent
Software
Company

Independent
Software
Company

Copyright © Infineon Technologies 2008. All rights reserved.
Software compilation flow as critical issue

- Mathematic Model
- Auto code Generator
- C-Code
- C-Code
- C-Code
- Compiler
- Object Code
- Object Code
- Object Code
- Optimizer
- Linker
- Target Code
- Optimizer
- Final Target Code

Tool chain Overall Size: Several 100MB

Safe ???
### Additional Safety Driver requirements

- Coverage of transient computation faults
- Fault model for testing data and addresses of registers, caches, internal RAM, Flash, CSFRs
- Test for dynamic cross-over of memory cells or registers
- No, wrong or multiple addressing
- Testing of opcode decoding and execution including flag registers
- Test of watchdog, traps, ECC (Parity), ...
- Peripheral configuration and operation
- Testing of program counter and stack pointers
- Detection of Continuous interrupts, Crossover of interrupts, Unused Interrupts
- Task execution monitor for OS and critical tasks
- External ASIC covers common cause failure
  - Power supply, short circuit on chip
  - Temperature of chip
  - EMC System clock
Table of contents

- Functional safety applied to microcontrollers
- Software as a major issue for functional safety
- Functional safety software partitioning
  - Functional dependent safety
  - Functional impendent safety
- Conclusion
Feasible functional safety approach for microcontrollers:

- Functional safety of a microcontroller as part of the system has to be split into:

  - Microcontroller functional *dependent* safety
    - Must be considered in the application itself
    - Key competence of (automotive) ECU supplier
    - Concept support by semiconductor supplier

  - Microcontroller functional *independent* safety
    - Must be supported independent from application
    - Key competence of semiconductor supplier
      - Hardware support by microcontrollers
      - Supply pre accredited software and concepts to all customers
      - Supply maintenance and know how
      - Supply scalability over IEC61508
Example:
Infineon TriCore® PRO-SIL™ concept

Existing Infineon TriCore® products can fulfill IEC61508 SIL3 requirements without an additional safety microcontroller or dual core lockstep technology.
- E.g. TriCore TC1796 or TC1766
- Based on asynchronous / asymmetric dual core architecture

**TriCore PRO-SIL™ concept for functional dependent safety**
- Concept support for redundant or diverse calculation
- Software encapsulation schemes
- Task and OS execution monitoring

**TriCore PRO-SIL™ concept for functional independent safety**
- Supply accredited safety concepts
- Supply and maintain State-Of-The-Art safety driver
  - Written after CMMI standard
  - Application, operation system and runtime environment independent
  - Customer independent
  - Scalable common code set over many OEM and ECU suppliers for
    - greater quality
    - Interoperability
    - legal protection
Table of contents

- Functional safety applied to microcontrollers
- Software as a major issue for functional safety
- Functional safety software partitioning
  - Functional dependent safety
  - Functional independent safety
- Conclusion
Options for safe software development

Write and certify:
- All used Tools
- All application software to SIL3 Standards

Safe and Robust Code

Use Proven-In-Use or diverse tools, build redundancies and diversity into application
Requirements for safe computation

Coverage of Transient Errors
- Caused by e.g. Radiation
- PFH for usual microcontroller core system is not reaching SIL3 requirements (<10^-7)

Coverage of Static Errors
- Caused by soft- and hardware bugs
- Avoid common mode errors from hardware and software bugs

Safe and Robust Code

Redundant Calculation of critical software

Diverse Calculation of critical software
Software development and computation proposals

Every effort must be made to negate the need to qualify software and the tooling

- Qualification is expensive, limits configurations, freezes release levels, is difficult or impossible to prove
- **Currently there are full tool chains known to fulfill IEC61508 SIL 3 requirements**

<table>
<thead>
<tr>
<th>Transient Error Detection</th>
<th>Static Error Detection</th>
<th>Programming Model</th>
<th>Code Generator</th>
<th>Compiler /Linker</th>
<th>Libraries</th>
<th>Data /Structure s</th>
<th>Computing Cores (Hardware)</th>
<th>Method Proposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>Common</td>
<td>Common</td>
<td>Common</td>
<td>Common</td>
<td>Common</td>
<td>One Core</td>
<td>No Failure Consideration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common</td>
<td>Common</td>
<td>Common</td>
<td>Redundant</td>
<td>One Core (Double Calculation)</td>
<td>Calculate Same Algorithm Twice For Transient Errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common</td>
<td>Common</td>
<td>Common</td>
<td>Redundant</td>
<td>Redundant (e.g. Lockstep)</td>
<td>Calculate Algorithm Twice For Transient Errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common</td>
<td>Diverse</td>
<td>Diverse</td>
<td>Redundant</td>
<td>Common (Running Diverse Code Set)</td>
<td>Compile Code Twice With Different Optimization Levels For diversity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common</td>
<td>Diverse</td>
<td>Diverse</td>
<td>Redundant</td>
<td>Diverse (e.g. TriCore + PCP)</td>
<td>Use Asymmetric Core System With Two Different Tool Chains</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common</td>
<td>Diverse</td>
<td>Diverse</td>
<td>Redundant</td>
<td>Common (Running Diverse Code set)</td>
<td>Add Diverse Code Generation (e.g. Auto + Complex Code)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Diverse</td>
<td>Diverse</td>
<td>Diverse</td>
<td>Redundant</td>
<td>Diverse (e.g. TriCore + PCP)</td>
<td>Fully Diverse Development</td>
<td></td>
</tr>
</tbody>
</table>
Robust software partitioning as requirement for functional safety

Microcontroller e.g. TriCore with PCP

Safety Software Driver (Functional Independent)

Diversity

Application 1a Tasks

Application 1b Tasks

Application 2 (2x) Tasks

Redundancy

Application 3 Task 1 Task 2

... Task 1 Task 2

Unwatched Tasks

Application 4

AutoSAR* Operating System
Run-Time Environment
Drivers, Communication
Microcontroller Abstraction Layer

*AutoSAR - scalability class 4
Memory protection
Time protection
# Table of contents

- Functional safety applied to microcontrollers
- Software as a major issue for functional safety
  - Functional safety software partitioning
    - Functional dependent safety
    - Functional independent safety
- Conclusion
Example: TriCore & PCP as asymmetric dual core to support functional independent safety
Safety Driver Solution

PRO-SIL™ safety driver provides a scalable and state of the art solution – supplied by the silicon vendor
- Covering functional independent parts
- Supporting functional dependent parts

<table>
<thead>
<tr>
<th></th>
<th>Boot-Time or Shutdown</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Checksum</td>
<td>X</td>
<td>Slices</td>
</tr>
<tr>
<td>SRAM Tests</td>
<td>X</td>
<td>Slices</td>
</tr>
<tr>
<td>Interrupt System Tests</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Opcode Tests</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Program Flow Monitoring</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Task Execution Timing</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Internal Bus Tests</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Inter Core Comm. Tests</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Timer Tests</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Internal Watchdog Tests</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>External Watchdog Tests</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Analog Converter Tests</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Peripheral Tests</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CAN Monitoring</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>FlexRay Monitoring</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Some PRO-SIL™ safety driver mechanisms

- Op-Code check mechanism
  - Coverage of 99% of used silicon in TriCore and PCP
  - Test running within failure reaction time
- Usage of all TriCore build in safety features
- Task Execution Monitor for functional depended software
- Test Execution Monitor
- Error injection mechanisms
  - Test The Tester
- Operation and runtime environment independent
- Application independent
Table of contents

- Functional safety applied to microcontrollers
- Software as a major issue for functional safety
- Functional safety software partitioning
  - Functional dependent safety
  - Functional independent safety
- Conclusion
Conclusions

- Designing Applications following existing (IEC61508) and new (ISO26262) standards will be THE challenge for safety critical automotive systems
  - Process implementation as major effort
  - Quality requirements to be meet with current systems
- Several safety concepts to fulfill IEC61508 are existing or in preparation
- Software is the major issue for safe systems for all involved partners
  - OEM
  - ECU supplier
  - Semiconductor vendor
- Requirements to supply safe software do not depend on fully qualified tool chains
  - Safe Software can be done with in limits of nowadays existing software development processes
References:


We commit.
We innovate.
We partner.
We create value.