EB tresos AutoCore and Polyspace

Measuring AUTOSAR software with a view towards ISO 26262

Alexander Much, 20110518
Topics

• About Elektrobit and AUTOSAR
  – definition of the complexity for measurement
• Polyspace usage
• ISO/DIS 26262 and AUTOSAR
  – method selection and decisions
• Tool collaboration
• Further steps
About Elektrobit
EB (Elektrobit) overview

- EB’s customers are the leading companies in the automotive and wireless markets
- EB provides demanding embedded software and hardware solutions
- EB’s technical core competences are:
  - Automotive-grade software
  - Wireless technologies and solutions
  - System and software architectures
- Net sales of MEUR 161.8 in 2010
- Listed on OMX Nordic Exchange Helsinki
- Approximately 1550 employees in 7 countries, 3 continents
EB Automotive Software

Figures

- Software products and services for Electronic Control Units, (Body, Comfort, Chassis, Powertrain), Infotainment and Driver Assistance

- Net Sales in 2010: MEUR 80.1

- Employees: ~1000
EB Automotive Software

Engineering competence close to our customers

- USA: Novi (Detroit), Bothell (Seattle)
- France: Paris
- Austria: Vienna
- Germany: Erlangen, Munich, Gaimersheim (Ingolstadt), Böblingen (Stuttgart), Braunschweig (Wolfsburg), Konstanz
- China: Beijing, Shanghai
- Japan: Tokyo
Selected Customers

Audi
BMW
Daimler
Fiat Group
Ford
General Motors
PSA
Renault
Porsche
VW Group

Autoliv
Bosch
Continental
Delphi
Hella
United Navigation (Falk)
Johnson Controls
Magneti Marelli
Medion
Mettler Toledo
Panasonic
Valeo
Visteon
ZF

Key Partners

Freescale
Fujitsu
Infineon
Intel
Windows Embedded Gold Partner
Renesas
ST Microelectronics
Texas Instruments
Solutions for the automotive world

**ECU Software**
- EB provides software modules for BMW standard core
- Leading AUTOSAR technical partner for JasPar in Japan
- Supplier of FlexRay solutions for BMW X5

**Infotainment Software**
- Audi A6 and Q7 Generic User Interface developed by EB
- Navigation and speech dialog software for the new Audi A1
- Application development (911 Assist, Vehicle Health Report and AppLink) as well as system integration partner for Ford SYNC™ platform

**Driver Assistance**
- ADTF distribution partner for AEV (Audi Electronics Venture)
- Extensions to ADTF and accompanying engineering services
- Additional algorithms and applications

Software Engineering, Product Customization, System Integration, Consulting
ECU Software

EB tresos® – One product line for ECU software development

- **EB tresos Designer**
  Automotive network design tool

- **EB tresos Studio**
  Basic software configuration tool

- **EB tresos AutoCore**
  AUTOSAR compliant basic software

- **EB tresos WinCore**
  AUTOSAR compliant basic software core that runs on Win32

- **EB tresos Debug & Trace**
  AUTOSAR runtime analysis

- **EB tresos Inspector**
  FR, CAN and LIN measurement and analysis tool

- **EB tresos Busmirror**
  FlexRay and CAN cluster emulation tool

- **EB 61x0, EB 2100, EB 5100**
  Bus interfaces for cluster emulation and measurement
EB and AUTOSAR
EB tresos AutoCore for VAG

- OS
  - OS SchM
  - ComM
  - EcuM
  - Nm

- Mode Management
  - Wdglf

- Memory
  - NvM
  - Crc

- Firmware
  - Wdglf
  - Mmu
  - (Eep)
  - (Fls)
  - (CanTrcv)
  - (FrTroc)
  - Fr*

- COM Services
  - Com
  - PduR

- FlexRay
  - FrSm*
  - FrNm*
  - FrTp*

- CAN
  - CanSm
  - CanNm
  - CanTp

- LIN
  - Lin
  - LinSm

- SDG
  - SDS
  - UDS

- DEH
  - Dem-Wrapper

- BAP
  - BAP-Wrapper

- KS Slave
  - KS-Wrapper

- FlexRay
  - FrSm*
  - FrNm*
  - FrTp*

- Module with VAG extensions

- VW module with AutoCore Wrapper

*VW extensions planned

EB AC module already incl. VAG extensions

Module with VAG extensions
EB tresos AutoCore for Daimler

Application Layer
- DemSc
- RoElite

Runtime Environment

Abstraction Layer
- Mode Management
- Diagnostic
- Memory
- COM Services
- Firmware

Service Layer
- CAN
- FlexRay
- LIN

ECU Abstraction Layer
- IPduM

Microcontroller Abstraction Layer
- Spi, RamTst
- Eep, FIs
- CanTcv
- Can

Daimler specific AUTOSAR modules

Generic AUTOSAR 3.0 modules
EB tresos Safety

AutoCore for safety related projects
EB tresos Safety AutoCore for safety related projects

Benefits
- Freedom from interference between safety and non-safety related software
- Protected communication and execution
- Compliant to ISO 26262
- Support of projects up to ASIL-D
- Supports use of QM black-box SW in ASIL-D ECU

Solution
- Fully integrated in EB tresos AutoCore
- Safety features according AS 4.0

Hardware Support
- Freescale Leopard MPC5643L
- Further MCUs on request
EB tresos Safety Product Overview

EB tresos AutoCore can be extended by **EB tresos Safety products** to meet requirements of safety-related projects up to ASIL-D

### Data Protection
- **EB tresos Safety OS**
  - Memory Protection
  - Register Protection
- **EB tresos AutoCore RTE Partitioning**
  - Memory Protection

### Communication Protection
**EB tresos Safety E2E Protection**
- End-To-End Protection of Inter or Intra-ECU-Communication

### Execution Protection
**EB tresos Safety Time Protection**
- Deadline Monitoring
- Control Flow Monitoring
- optional link with Challenge-response

*) complementing engineering services offered for verification
Polyspace usage
Polyspace & AutoCore

• Introduced for EB tresos AutoCore in Q1/2010

• Used in two different ways
  – static analysis: MISRA-C and metrics
  – dynamic analysis: formal verification

• Both analysis methods are performed for each module individually

• Current status:
  – All AUTOSAR modules (> 60) are verified
    • Statically in continuous testing (every day)
    • Dynamically for each iteration (4 weeks)
  – Polyspace is fully integrated in automated build environments
Static analysis & Code metrics

• Polyspace is used as **static analysis tool**
  – e.g. for checking MISRA-C:2004 compliance
  – EB is in contact with MathWorks providing feedback to Polyspace R&D about capabilities of static analysis and its suitability for automotive applications.

• Polyspace is used for measuring **code metrics**
  – based on the metrics published by the HIS (HerstellerInitiative Software)
  – see [http://portal.automotive-his.de/images/pdf/SoftwareTest/his-sc-metriken.1.3.1.pdf](http://portal.automotive-his.de/images/pdf/SoftwareTest/his-sc-metriken.1.3.1.pdf)

• The analysis
  – is fully integrated into the build environment
    • i.e. “make polys”
  – is accumulated over multiple configurations to a final result
Formal verification

- Code verification is an automated process, but the analysis is still manual.

- what has been found
  - missing range checks, e.g. for array indices
  - overflows, missing or incorrect initialization of variables

- Experience after one year
  - number of violations due to implementation errors has dropped
  - due to constant feedback to development
  - most modules are now consistently >97% green, even during development

- Suggestion:
  - like with MISRA-C: use analysis tooling often and use it as feedback to development to avoid errors
  - otherwise you constantly have a high verification and re-work effort (no learning effect)
  - fully integrate Polyspace in the build environment
Effort for automation

• Insertion into the build environment:
  – ca. 2d

• Creating correct and configurable configurations
  – ca 1d

• Static analysis:
  – Interpreting Polyspace output files into internal formats and placing the results in metric databases: ca 1 week
  – All output files are open formats and XML-based and can be processed with open tools, e.g. xslt

• Dynamic analysis:
  – Creating the configuration: ca 1d
  – Analysis results are not further automated (manual effort)

• Overall effort spent for automation: ca 2 weeks
Activation scenario and critical sections

- Usually a software module has three activation types
  - called by services ("from above")
  - called by the e.g. the hardware, possibly in interrupt context
  - periodically triggered to perform some work

- The activation depends on the configuration in the architecture
  → identifying shared global data becomes a complicated task

- The results are then used for performing optimization of the amount of critical sections necessary
  - of course: this can also be configured
Hints for verification

- There needs to be a good architecture which
  - clearly defines software modules
  - and all interactions between them
- In the first step (SW development):
  - verify units
  - create activation scenarios from the architecture
  - use defensive programming (e.g. value range checks) at each “border point”
- Second step (SW integration):
  - build “stacks” of modules
  - create activation scenarios
  - again: use defensive programming at each “border point”
- This approach fits into the life-cycle of most development projects.
- Usually the whole ECU software is too large to be checked with Polyspace
Polyspace and ISO/DIS 26262
Polyspace and ISO 26262

• With the advent of ISO/DIS 26262 some development and verification methods will be more widely used.

• Polyspace can cover parts of ISO/DIS 26262, e.g.:
  – enforcing coding guidelines by using static analysis
  – ensuring compliance with design and implementation principles:
    • initialization of variables
    • use and justification of global variables
  – supporting control and data flow analysis
  – detecting unintended functionality, e.g. “dead code”
## ISO/DIS 26262-6, Table 1

<table>
<thead>
<tr>
<th>Topics</th>
<th>Polyspace support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Enforcement of low complexity</td>
<td>Fully (HIS-metrics)</td>
</tr>
<tr>
<td>1b Use of language subsets</td>
<td>Fully (MISRA-C:2004)</td>
</tr>
<tr>
<td>1c Enforcement of strong typing</td>
<td>Partly (MISRA + formal verification)</td>
</tr>
<tr>
<td>1d Use of defensive implementation techniques</td>
<td>Partly</td>
</tr>
<tr>
<td>1e Use of established design principles</td>
<td>Partly (HIS-metrics)</td>
</tr>
<tr>
<td>1f Use of unambiguous graphical representation</td>
<td>None</td>
</tr>
<tr>
<td>1g Use of style guides</td>
<td>None</td>
</tr>
<tr>
<td>1h Use of naming conventions</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 1: Topics to be covered by modelling and coding guidelines
### ISO/DIS 26262-6, Table 9

<table>
<thead>
<tr>
<th>Topics</th>
<th>Polyspace support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a One entry and one exit point in subprograms and functions</td>
<td>Fully (MISRA 14.7)</td>
</tr>
<tr>
<td>1b No dynamic objects or variables, or else online test during their creation</td>
<td>Fully (MISRA 20.4)</td>
</tr>
<tr>
<td>1c Initialization of variables</td>
<td>Fully (MISRA 9.1 and formal verification using special configuration)</td>
</tr>
<tr>
<td>1d No multiple use of variable names</td>
<td>Fully (MISRA rules 5.* and formal verification)</td>
</tr>
<tr>
<td>1e Avoid global variables or else justify their usage</td>
<td>Partly (formal verification using activation scenarios)</td>
</tr>
<tr>
<td>1f Limited use of pointers</td>
<td>Fully (MISRA 11.1, 11.2, 11.3, 11.5, 16.7, 17.3, 17.4, 17.5, 17.6 and formal verification)</td>
</tr>
<tr>
<td>1g No implicit type conversions</td>
<td>Partly (MISRA 10.1, 10.2)</td>
</tr>
<tr>
<td>1h No hidden data flow or control flow</td>
<td>None</td>
</tr>
<tr>
<td>1i No unconditional jumps</td>
<td>Fully (MISRA 14.4, 20.7)</td>
</tr>
<tr>
<td>1j No recursions</td>
<td>Fully (MISRA and formal verification)</td>
</tr>
</tbody>
</table>

Table 9 — Design principles for software unit design and implementation
<table>
<thead>
<tr>
<th>Topics</th>
<th>Polyspace support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Informal verification</td>
<td>See table 11</td>
</tr>
<tr>
<td>1b Semi-formal verification</td>
<td>None</td>
</tr>
<tr>
<td>1c Formal verification</td>
<td>None</td>
</tr>
<tr>
<td>1d Control flow analysis</td>
<td>Partly (call graph)</td>
</tr>
<tr>
<td>1e Data flow analysis</td>
<td>Partly (data dictionary, all read/write accesses)</td>
</tr>
<tr>
<td>1f Static code analysis</td>
<td>Fully</td>
</tr>
<tr>
<td>1g Semantic code analysis</td>
<td>Fully</td>
</tr>
</tbody>
</table>

Table 10: Methods for the verification of software unit design and implementation
## ISO/DIS 26262-6, Table 11

<table>
<thead>
<tr>
<th>Topics</th>
<th>Polyspace support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Inspection of the software unit design</td>
<td>None</td>
</tr>
<tr>
<td>1b Walkthrough of the software unit design</td>
<td>None</td>
</tr>
<tr>
<td>1c Model inspection</td>
<td>None</td>
</tr>
<tr>
<td>1d Model walkthrough</td>
<td>None</td>
</tr>
<tr>
<td>1e Inspection of the source code</td>
<td>Supporting (Review mode)</td>
</tr>
<tr>
<td>1f Walkthrough of the source code</td>
<td>Supporting (Review mode)</td>
</tr>
</tbody>
</table>

Table 11: Methods for the informal verification of software unit design and implementation
EB tresos and Polyspace

An outlook
Polyspace and AUTOSAR

- There could be additional support of Polyspace for AUTOSAR projects, e.g.:
  - special sub-sets of MISRA rules that cannot be used
  - knowing and checking AUTOSAR data types:
    - the type `boolean` can be limited to 0/1 and its use can be checked
      - could then also used for static analysis
    - the types `uint8_least` types could be range-checked
      - which will be interesting also for ISO-C:99 `_fast` types
  - some coding guidelines from EB and from AUTOSAR could be enforced by Polyspace
Integration: Polyspace and Tresos

• There are many integration possibilities of AUTOSAR, EB tresos and Polyspace

• EB tresos is eclipse-based and there already is a Polyspace plugin
  – How about a “check this project with Polyspace”-button?
  – List of files and inclusions are generated by the build environment by using the project configuration from EB tresos Studio

• Even more interesting: generating the configuration
  – Scheduling and dynamics are taken from AUTOSAR configuration (possible in 4.0)
  – Definition of global variables and their ranges can also be generated from the AUTOSAR configuration
  – This also applies to the external interfaces of customer applications
Next steps
Next steps

• Further cooperation with MathWorks:
  – on static analysis
  – on tool integration
  – in the SQO group

• Further roll-out is planned for:
  – OEM specific AUTOSAR versions and modules
  – Development of complex device drivers

• Experience needs to be gathered:
  – for specific ECU projects
  – e.g. the complete basic software of an ECU can be checked for individual customer projects
Thank you!