Using the Benefits of Model-Based Design to Develop AUTOSAR Basic Software Modules

Mohamed Soliman & Amjad Elshenawy

Mathworks Automotive Conference 2016
Stuttgart, Sep., 21\textsuperscript{st} 2016
Agenda

1. Why use MBD for Developing AUTOSAR BSW Modules?
2. CAN State Manager (CanSM)
3. Challenges Encountered in Developing CanSM using MBD
4. Results of Our Experiment
Agenda

1. Why use MBD for Developing AUTOSAR BSW Modules?
2. CAN State Manager (CanSM)
3. Challenges Encountered in Developing CanSM using MBD
4. Results of Our Experiment
AUTOSAR Embraces Complexity

Number of Basic SW Modules

- Number of Basic SW Modules

<table>
<thead>
<tr>
<th>Release Version</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.0</td>
<td>2005</td>
</tr>
<tr>
<td>R2.0</td>
<td>2006</td>
</tr>
<tr>
<td>R2.1</td>
<td>2006</td>
</tr>
<tr>
<td>R3.0</td>
<td>2007</td>
</tr>
<tr>
<td>R3.1</td>
<td>2008</td>
</tr>
<tr>
<td>R3.2</td>
<td>2011</td>
</tr>
<tr>
<td>R4.0</td>
<td>2011</td>
</tr>
<tr>
<td>R4.1</td>
<td>2014</td>
</tr>
<tr>
<td>R4.2</td>
<td>2015</td>
</tr>
</tbody>
</table>

Number of Basic SW Modules:
- 1X in 2005
- 33 in 2006
- 45 in 2006
- 51 in 2007
- 53 in 2008
- 53 in 2008
- 55 in 2011
- 80 in 2011
- 89 in 2014
- 98 in 2015

R4.2 release has 98 modules, a 3X increase from R1.0.
AUTOSAR Embraces Complexity

Number of Requirements

- 1X
- 2.525
- 5.311
- 6.006
- 8.626
- 8.928
- 9.811
- 19.569
- 27.357
- 31.759

R1.0 2005
R2.0 2006
R2.1 2006
R3.0 2007
R3.1 2008
R3.2 2011
R4.0 2011
R4.1 2014
R4.2 2015

12X
Characteristics of AUTOSAR Basic Software Modules

- **Pre-compile Configuration**: Enabling/disabling optional functionality
- **Link-time Configuration**: Configuration of modules that are only available as object code (e.g. IP protection)
- **Post-build Configuration**: Change the configuration after building the code (in the run time)

**Highly Configurable**
Characteristics of AUTOSAR Basic Software Modules

- Standard Interfaces
- and Standard Types
Motivations for using MBD for Developing AUTOSAR BSW Modules

In our case MBD is selected to provide the following benefits:

- Shorter development time
- Better re-usability and maintainability of design / model.
- Improvement of the product quality
Agenda

1. Why use MBD for Developing AUTOSAR BSW Modules?
2. CAN State Manager (CanSM)
3. Challenges Encountered in Developing CanSM using MBD
4. Results of Our Experiment
CAN State Manager

- One of the basic software communication stack modules.
- Responsible for managing the states of the Can networks.
CAN State Manager

Module Complexity

- 280 requirements.
- 26 Configuration parameters.
- 18 Provided Interfaces.
Agenda

1. Why use MBD for Developing AUTOSAR BSW Modules?
2. CAN State Manager (CanSM)
3. Challenges Encountered in Developing CanSM using MBD
4. Results of Our Experiment
SWS_BSW_00029: If the BSW Module contains optional functionality, then this functionality shall be enabled (STD_ON) or disabled (STD_OFF) by a Pre-compile time configuration parameter.
Pre-compile Configuration

- Using “Variant Subsystem” to generate pre-compile configuration
Pre-compile Configuration

- Generate preprocessor conditional for with variant model blocks.
SRS_Can_01142: The CAN State Manager shall offer a network abstract API to upper layer

Example scenario: "Network status change upon Communication Manager module (ComM) request"

```
Std_ReturnType:=CanSM_RequestComMode(NetworkHandle,ComM_Mode:=COMM_FULL_COMMUNICATION)
CanSM_RequestComMode(...)=E_OK
```

COMM_FULL_COMMUNICATION indication
Standard Interfaces

CanSM_MainFunction

Function parameters:
- Function packaging: Nonreusable function
- Function name options: User specified
- Function name: CanSM_MainFunction
- File name options: Auto
- Function interface: void
- Memory section for initialize/terminate functions: Inherit from model
- Memory section for execution functions: Inherit from model

Subsystem
Select the settings for the subsystem block. To enable parameters on the Code Generation tab, on the Main tab, select "Treat as atomic unit."
Code Duplication

SWS_BSW_00127: The BSW Module implementation shall avoid duplication of code.
Code Duplication

- Using library of atomic sub-chart to avoid code duplication.
Compliance with MISRA C Rules

SWS_BSW_00115: If the BSW Module implementation is written in C language, then it shall conform to the MISRA C 2004 Standard

- Source complexity (Cyclomatic Complexity): Number of linearly independent paths should not exceed a certain limit.

- Implicit and explicit type conversions (Casting). Example: casting from integer to pointer is prohibited.

- Parentheses “(” and “)” should be used to emphasis expressions.

- The final clause of a switch statement shall be the default clause.
Compliance with MISRA C Rules

Cyclomatic Complexity control by separating atomic parts in separate functions
Compliance with MISRA C Rules

- Implicit and explicit type conversions (Casting)
- Parentheses level
- The final clause of a switch statement
Non-Functional Requirements

- Maintainability
- Reusability
- Reliability
- Efficiency (Execution Time, Memory consumption, …)
Non-Functional Requirements

- Execution time: Switch Case Vs If Else

Expression

- Case #1: Statement #1
- Case #2: Statement #2
- Case #n: Statement #n

Expression

- Exp #1: Statement #1
- Exp #2: Statement #2
- Exp #n: Statement #n

Default Statement
Non-Functional Requirements

- Execution time optimization: Code generation with Switch Case instead of If Else
Smoke Testing

- Smoke testing is non-exhaustive software testing, ascertaining that the most crucial functions of a program work, but not bothering with finer details.

- Smoke testing is not a substitute for traditional testing mechanism.
Smoke Testing

- Attaching Microsoft Visual Studio to Matlab process.
Smoke Testing

- Debugging in the Model and the manual code.
Agenda

1. Why use MBD for Developing AUTOSAR BSW Modules?
2. CAN State Manager (CanSM)
3. Challenges Encountered in Developing CanSM using MBD
4. Results of Our Experiment
Results of The Provided Solution

- Development time is about 18% less than the other manually developed modules with similar size.

- Bug fixing is about 34% shorter than the other manually developed modules with similar size.

- Number of issues found during testing phase is about 30% less than the other manually developed modules with similar size.