Developing AUTOSAR Compliant Embedded Software

Senior Application Engineer
Sang-Ho Yoon
 Agenda

- AUTOSAR Compliant Code Generation
  - AUTOSAR Workflows
  - Starting from Software Component Descriptions in ARXML files (Top-Down Approach)
  - Starting from Simulink (Bottom-up Approach)
- Verification of AUTOSAR ASWC with Model
- Verification of AUTOSAR ASWC for Generated Code
AUTOSAR Compliant Code Generation
What is AUTOSAR?

- AUTOSAR® (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture

- Partnership consisting of more than 140 companies from the global automotive industry

Source: AUTOSAR, status 04.01.2011
What is AUTOSAR?

3-layered Architecture

Target-independent application development through an RTE and standardized interfaces

Behavior Modeling & Code Generation

BSW Configuration & RTE Generation

Software Architecture Definition
MathWorks AUTOSAR Approach

No separate AUTOSAR Blockset needed

- Code-generation through Mapping

AUTOSAR Software Component Approach with Simulink

- Simulink for developing behavior
- Import and Export of SW Component Description Files (ARXML)

Simultaneous generation of C-code and ARXML-Files

- Consistency between C-code and ARXML SW-C description files
Support for AUTOSAR Workflows

- Export ARXML
- Import/Update Simulink model
- Generate
- Integrate

ARXML  C Code
Getting Started

- Start with ARXML files containing AUTOSAR Component descriptions (Top-Down approach)

- Start with an existing Simulink model (Bottom-Up Approach)
Starting from Software Component Descriptions in ARXML files

Top-Down Approach
Top Down Workflow

ARXML

Import as new Simulink model

Update existing Simulink model
%Import ARXML Files
importerObj = arxml.importer('rtwdemo_autosar_multirunnables.arxml')

%Create new model with interfaces
model = importerObj.createComponentAsModel('/pkg/swc/ASWC');
Design Controller from Requirement Case Study : Seat Belt Reminder
SBR Algorithm Model

Functional Requirements

SBR Model
Tracing Requirements ↔ Model
Simulink Verification and Validation

- Creating links between textual documents and model objects
Requirements Traceability – Report
Simulink Verification and Validation

- Requirements Report provides screenshots of the model and lists all the associated requirements.
Simulate and Verify Algorithms
%Import ARXML Files
importerObj = arxml.importer('rtwdemo_autosar_multirunnables.arxml')

%Create new model with interfaces and internal behavior
model = importerObj.createComponentAsModel('/pkg/swc/ASWC', ... 
    'CreateInternalBehavior',true);
Updating Existing Models from ARXML

V1.arxml  Updated to V2.arxml
Update Existing Models from ARXML

%Import ARXML Files
importerObj = arxml.importer('rtwdemo_autosar_multirunnables_v2.arxml')

%Update existing model
importerObj.updateModel('ASWC')
How about Legacy Code?

- Use of Legacy Code Tool for introduction of existing C code on Simulink models

External C Function

MATLAB code specification
Starting from Simulink
Bottom-up Approach
Overview: Model AUTOSAR Components

Application Layer

AUTOSAR Software Component 1

... 

AUTOSAR Software Component n

Virtual Functional Bus (VFB)

This model generates AUTOSAR compliant code and software component XML files.

Configure model as AUTOSAR

Generate code (double-precision model)
Overview: Mapping Simulink to AUTOSAR

Data Elements of AUTOSAR S/R Ports Implemented with Simulink I/O
Mapping Simulink to AUTOSAR

- Single Periodic Runnable-Entity
- Implement with Top-level model
Overview: Mapping Multiple Entry points

- Runnable-Entity
- Maps to Function-Call Inputs
Step 0: Design and Simulate Algorithms

Functional Requirements

SBR Model
Step 1: Select AUTOSAR Target
Step 2: Launch AUTOSAR Configuration

>> rtwdemo_autosar_counter
Step 3: Map Between Simulink-AUTOSAR with Mapping Editor

- View/Edit AUTOSAR Properties
- Perform mapping of Simulink Entities to AUTOSAR in the Simulink Mapping view
Editing AUTOSAR Properties

- Add/Remove AUTOSAR Entities
- Edit Properties of AUTOSAR Entities
- Configure ARXML options
Mapping Inports to AUTOSAR Receiver Ports
Map Outports to AUTOSAR Sender Ports
Map Entry Point Functions for a Model
Step 4: Validate Configuration
Step 5: Generate Code

- Build model
- Generates both C & ARXML
- Code uses RTE APIs for Sender Receiver ports
Flexibility to specify Packages
Verification of AUTOSAR ASWC with Model
Test Cases to Signal Builder

1. TC01 - Passing Maneuver
   This test case is tested with 50% throttle value to check passing maneuver and steady gear number increase.

2. TC02 - Gradual Acceleration
   This test case tests gear change behavior with gradual speed increase with steady increase of throttle pedal position.

![Test Cases](image1.png)

![Signal Builder](image2.png)
Module Test-Harness

Test Cases (Signal Builder)

SBR Model (Model Block)

Output Check (Assertion Block)
Coverage metrics identifies untested portions of your model
Improving Test Suite
Simulink Design Verifier

- Generating tests to reach coverage criteria

Test generation from model

Test inputs that ensure complete coverage
How about Coverage for Legacy Code

- Software Component includes S-Function for Legacy Code
Coverage for C-code S-Functions

S-Function Instances

Included instances: slxSFcnSLDVExample/Legacy code S-Function

Tests

Test 1


Summary

<table>
<thead>
<tr>
<th>File Contents</th>
<th>D1</th>
<th>C1</th>
<th>MCDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. categorize.c</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>2. ... categorize_input</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Details

1. File categorize.c

Functions: categorize_input (line 3)

C code

/* Legacy code example */
signed char result;

if(input > 0.0 && input < threshold) {
    result = 0;
} else if(input >= threshold) {
    result = 1;
} else if(input < -threshold) {
    result = -1;
} else {
    result = 0;
}

return result;
Verification of AUTOSAR ASWC for Generated Code
Software-in-the-Loop (SIL) Testing:
Verify Production Controller with Software-in-the-loop

Execution
- Host/Host
- Nonreal-time

Compiled C Code
S-Function
(Windows DLL)
Processor-in-the-Loop Testing:
Verify Production Controller with Processor-in-the-loop

Execution
• Host/Target
• Nonreal-time
Verification with SIL/PIL

- Support for PIL with AUTOSAR target
- Profile code and measure execution time on target
Formal Verification of Generated Code with Polyspace

Can you prove absence of run-time errors?
Runtime Errors

- **Arithmetic errors**
  - Found in (code)
    - Overflows, division by zero, bit-shifts, square root of negative numbers
  - Caused by (model)
    - Faulty scaling, changes in or unknown calibrations, untested data ranges coming out of a subsystem into an arithmetic block

- **Memory corruption**
  - Found in (code)
    - Out of bound array indexes
    - Pointer arithmetic
  - Caused by (model)
    - Array manipulation in Stateflow
    - Hand-written look-up table functions

- **Data truncation**
  - Found in (code)
    - Overflows
    - Wrap around
  - Caused by (model)
    - Saturations leading to unexpected data flow inside the generated code

- **Coding errors**
  - Found in (code)
    - Non initialized data
    - Dead code leading to unreachable transitions or states
  - Caused by (model)
    - Faulty Stateflow programming
CATEGORIES OF STATIC ANALYSIS OF USING TOOLS

- Compiler Warnings
- Code Metrics, and Coding Rules
- Error Prevention
- Error Detection
- Bug Findings (False negative)
- Formal Methods (No False negative)
Polyspace PRODUCTS

Compiler Warnings
Code Metrics, and Coding Rules

Polyspace Bug Finder
Polyspace Code Prover

Error Prevention

Error Detection

Bug Findings
(False negative)

Formal Methods
(No False negative)

Polyspace Bug Finder
Polyspace Code Prover
Polyspace
Formal Methods based Static Code Analysis

- Exhaustively verify code
  - Detect and prove absence of runtime errors
  - Precisely determines and propagates variable ranges

- Languages supported
  - C, C++, and Ada

- Verify SW robustness
  - Analyze for full range operating conditions
  - Specified ranges of parameters and inputs

```
static void pointer_arithmetic (void) {
  int array[100];
  int *p = array;
  int i;
  for (i = 0; i < 100; i++) {
    *p = 0;
    p++;
  }
  if (get_bus_status() > 0) {
    if (get_oil_pressure() > 0) {
      *p = 5;
    } else {
      i++;
    }
  }
  i = get_bus_status();
  if (i >= 0) {
    (*p - i) = 10;
  }
}
```

- Green: reliable safe pointer access
- Red: faulty out of bounds error
- Gray: dead unreachable code
- Orange: unproven may be unsafe for some conditions
- Purple: violation MISRA-C/C++ or JSF++ code rules

Range data tool tip
Fixing the Runtime Error in the Design
Trace and Fix Using PolySpace Model Link SL

May Overflow
(establish traceability between PolySpace analysis and model)
Formal Verification in Design phase with Simulink Design Verifier

Is it possible to verify early in “Design” phase?
Can you prove your design always satisfies requirements?
Identifying Design Errors Early
Simulink Design Verifier

- Static Run-Time Error Detection
  - Automatic identification of hard-to-find design inconsistencies in the model without running simulation

- Supported detecting types of errors
  - Dead logic
  - Integer overflow
  - Division by zero
  - Range violation
  - Assertion violation
  - Out of bound array access
Verifying Design Against Requirements
Simulink Design Verifier

1. Functional and Non-Functional Requirement
2. Formal model (System)
3. Formal Property

Improve algorithm, or requirements
Example of Properties

- Simple True / False
  - Simple logic blocks
  - Truth table

- Temporal logic
  - Temporal Logic blocks (part of SLDV demo)
    - Example of using basic templates as building blocks for complex property.
  - Imply blocks (part of SLDV blockset)
    - \(~A\) or \(B\)
  - Stateflow / Embedded MATLAB Functions
Summary

- Simulink and Embedded Coder provide extensive AUTOSAR capabilities out-of-the-box.
- Use one AUTOSAR workflow (Top-Down/Bottom-Up) that best support your workflow and AUTOSAR concepts
- Take advantage of Production Code Generation to accelerate your AUTOSAR projects while reducing risk and improving quality.
- Various Model-Based Verification methods can also be applied to AUTOSAR ASWC development.
Change the world by

**Accelerating the pace**

of discovery, innovation, development, and learning

in engineering and science