MathWorks Vision for Systematic Verification and Validation

Bill Aldrich
Senior Development Manager
Simulink Verification and Validation, Simulink Design Verifier
Growing Complexity of Automotive Controls

- Emergency Braking
- Body Control Module
- Voice Recognition
- Power Window
- Vehicle-to-Infrastructure
- Power Liftgate
- Power Seat
- Back-up Camera
- Long-Range Radar
- All-Wheel Drive
- Tire Pressure Monitor
- Ultrasonic Sensor
- Stability Control
- Adaptive Cruise Control
- Automatic Parking
- Smart Junction Box
- Battery Management
- Instrument Panel
- Airbag
- Electric Power Steering
- Engine Management
- Transmission Control
- Forward Camera
- Adaptive Front Lighting
- HVAC Control
- Vehicle-to-Vehicle
- E-Call
- Keyless Entry
- Short-Range Radar
- DC/DC Converter
- Propulsion Motor Control
- Navigation

Growing Complexity of Automotive Controls

McKendrick, J. “Cars become ‘datacenters on wheels’, carmakers become software companies,” ZDJNet, 2013
Growing Complexity of Automotive Controls

Growing Complexity of Automotive Controls

- Emergency Braking
- Adaptive Cruise Control
- Stability Control
- Automatic Parking
- Smart Junction Box
- Battery Management
- Instrument Panel
- Airbag
- Electric Power Steering
- Engine Management
- Propulsion Motor Control
- DC/DC Converter
- Transmission Control
- Infotainment
- Navigation
- Forward Camera
- Adaptive Front Lighting
- HVAC Control
- Vehicle-to-Vehicle
- Vehicle-to-Infrastructure
- E-Call
- Keyless Entry
- Short-Range Radar
- Long-Range Radar
- Ultrasonic Sensor
- Body Control Module
- Voice Recognition
- Power Window
- Power Liftgate
- Power Seat
- Back-up Camera
- All-Wheel Drive
- 4-Wheel Steer
- Active Damping
- Smart Junction Box
- Stability Control
- Tire Pressure Monitor

Development Challenges

- Representing complex systems
- Coordinating work across teams
- Working efficiently
- Ensuring quality
Traditional Development Process

Textual Requirements → Design Specification

C/C++ Hand code → Object code

Manual Coding → Compilation and Linking
Models for Specification
Model Abstraction – Work at an appropriate level of detail
Complete Model Based Design Workflow, Concept to Code
Complete Model Based Design Workflow, Concept to Code

How do you ensure correctness?
Model-Based Design Maturity, Automotive Industry

- **Modeling**
- **Simulation and Analysis**
- **Implementation**
- **Verification and Validation**
- **Process, Tools and Infrastructure**
- **Enterprise Management**
Model-Based Design Maturity, Automotive and Aerospace

- Enterprise Management
- Process, Tools and Infrastructure
- Verification and Validation
- Modeling
- Simulation and Analysis
- Implementation

Automotive
Aerospace
Model Based Design Verification Workflow

Textual Requirements → Executable Specification → Model used for production code generation → Generated C/C++ code → Object code

Component and system testing

Review and static analysis

Equivalence testing

Equivalence checking

Modelling

Code Generation

Compilation and Linking
Model Based Design Verification Workflow

- Perform simulation
  - Link and review requirements
  - Isolate and test components
  - Measure model coverage
  - Address missing coverage
  - Property proving

Component and system testing

Textual Requirements → Executable Specification → Model used for production code generation

Modelling → Code Generation → Generated C/C++ code → Object code

Compilation and Linking
Ad-Hoc Simulation: Explore Behavior Virtually
Model Based Design Verification Workflow

- Perform simulation
- Link and review requirements
- Isolate and test components
- Measure model coverage
- Generate tests for missing coverage
- Manage and organize tests
- Property proving

Component and system testing

Textual Requirements → Executable Specification → Model used for production code generation

- Generated C/C++ code
- Object code

Modelling → Code Generation → Compilation and Linking
Test Harnesses

From any subsystem ...
Test Harnesses

From any subsystem …

Isolate it with content it to drive inputs and analyze outputs

Simulate independently

Can be embedded in design model file.
A test sequence block can drive inputs
A test sequence block can drive inputs and assess outputs.
Test Sequence Block Syntax

Start
speed = ramp(10);
throttle = 25;
verify(gear == 1);
Test Sequence Block Syntax

Define Inputs

Start
speed = ramp(10);
throttle = 25;
verify(gear == 1);
Defining Pass/Fail Criteria

Start
speed = ramp(10);
throttle = 25;

verify(gear == 1);

Test Sequence

vehicle speed

gear

shift_logic

Fail
Pass
Untested

0 1
Model Coverage

Identify testing gaps:
• Untested switch positions
• Subsystems not executed
• Transitions not taken
• Many more …
Addressing Missing Coverage

Test Cases

Partial Coverage
Addressing Missing Coverage

Test Cases

Simulink Design Verifier

Partial Coverage

Test Generator
Addressing Missing Coverage

Test Cases

New Test Cases

Test Generator

Simulink Design Verifier

Partial Coverage
Addressing Missing Coverage

New Test Cases

Test Cases

Full Coverage
Model Based Design Verification Workflow

- Perform simulation
- Link and review requirements
- **Isolate and test components**
- Measure model coverage
- Generate tests for missing coverage
- Manage and organize tests
- Property proving

Component and system testing
Model Based Design Verification Workflow

- Manual review
- Standards compliance checking
- Design error detection
- Complexity analysis
Detecting Hidden Run-Time Design Errors

- Integer overflow
- Division by zero
- Array out-of-bounds
- Range violations
- Dead Logic
Detecting Hidden Run-Time Design Errors
Detecting Hidden Run-Time Design Errors
Model Based Design Verification Workflow

- Perform SIL Testing
- **Measure code coverage**
- Verify code with Polyspace
- Verify consistency with Simulink

Component and system testing

Review and static analysis

Equivalence checking

Textual Requirements → Executable Specification → Modelling → Model used for production code generation → Generated C/C++ code → Code Generation → Object code

Modelling

Compilation and Linking
if (rtb_ActiveControl) {
    /* Sum: 'c62>/Sum' incorporates:
       * DiscreteIntegrator: 'c62>/Discrete-Time Integrator'
    * Gain: 'c62>/Kp'
    * Gain: 'c62>/Kp1'
    */
    rty_throt = 0.02 * rtb_Switch2 + 0.01 *
    localDW->DiscreteTimeIntegrator_DSTATE;
    /* Update for DiscreteIntegrator: 'c62>/Discrete-Time Integrator'
    localDW->DiscreteTimeIntegrator_DSTATE += 0.01 * rtb_Switch2;
    */
    if (localDW->DiscreteTimeIntegrator_DSTATE >= 5.0) {
        localDW->DiscreteTimeIntegrator_DSTATE = 5.0;
        else {
            if (localDW->DiscreteTimeIntegrator_DSTATE <= -5.0) {
                localDW->DiscreteTimeIntegrator_DSTATE = -5.0;
            }
        }
    }
Coverage for Generated Code (R2016a)

Can also be highlighted on model
Model Based Design Verification Workflow

- Perform PIL Testing
- Perform HIL Testing

Component and system testing

Textual Requirements $\rightarrow$ Executable Specification $\rightarrow$ Modelling

Review and static analysis $\rightarrow$ Equivalence testing

Equivalence checking $\rightarrow$ Generated C/C++ code $\rightarrow$ Object code

Model used for production code generation

Code Generation $\rightarrow$ Compilation and Linking
Model Based Design Verification Workflow

Model used for production code generation

Generated C/C++ code

Object code

Code Generation

Compilation and Linking

Component and system testing

Review and static analysis

Equivalence testing

Equivalence checking

Textual Requirements

Executable Specification

Modelling
Systematic Verification

- Ensure that verification is systematically performed across:
  - All requirements
  - Complete model structure
  - Complete code structure
  - All design behaviors
Test and Verification

- Essential
- Expensive
- Complex

Pain Points
Test and Verification

- Essential → More Complete
- Expensive → Faster
- Complex → Simpler

Frequent, Incremental T&V
Thank You!