MathWorks Vision for Systematic Verification and Validation

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

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

Development Challenges
Development Challenges

- Representing complex systems
Development Challenges

- Representing complex systems
- Coordinating work across teams
Development Challenges

- Representing complex systems
- Coordinating work across teams
- Working efficiently
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

- **Textual Requirements** → **Executable Specification** → **C/C++ Hand code** → **Object code**

- **Compilation and Linking**

- **Manual Coding**
Model Abstraction – Work at an appropriate level of detail

Simscape Fluids

Simscape Multibody

Simulink

Stateflow

MATLAB

```
% Predicted state and covariance
x_pred = A * x_est;
P_pred = A * P_est * A' + Q;

% Estimation
x = B * p_pred';
R = B * p_pred';

% Compute the estimated measurements
y = B * x_est;
```
Complete Model Based Design Workflow, Concept to Code

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

Modeling

Enterprise Management

Simulation and Analysis

Process, Tools and Infrastructure

Implementation

Verification and Validation
Model-Based Design Maturity, Automotive and Aerospace

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

Automotive
Aerospace
Model Based Design Verification Workflow

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

Modelling

Component and system testing

Review and static analysis

Equivalence checking

Equivalence testing

Simulink Models

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
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

Simulink Models

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

Start
speed = ramp(10);
throttle = 25;
verify(gear == 1);

Define Inputs
Defining Pass/Fail Criteria

Start
speed = ramp(10);
throttle = 25;
verify(gear == 1);

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
Addressing Missing Coverage

New Test Cases

Test Cases

Test Generator

Simulink Design Verifier

Partial Coverage

speed > down_th was never true.
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

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

Simulink Models

Modelling

Code Generation

Compilation and Linking
Model Based Design Verification Workflow

- **Modelling**
- **Textual Requirements**
- **Review and static analysis**
  - Manual review
  - Standards compliance checking
  - **Design error detection**
  - Complexity analysis

---

**Simulink Models**

- **Executable Specification**
- **Model used for production code generation**
- **Generated C/C++ code**
- **Object code**
- **Code Generation**
- **Compilation and Linking**
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 Code Inspector

---

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

Equivalence checking

Modelling  →  Code Generation  →  Compilation and Linking
Coverage for Generated Code (R2016a)

```
cruise_control
  (SIL)
```
Coverage for Generated Code (R2016a)

Press Play

cruise_control (SIL)
if (rtb_ActiveControl) {
    /* Sum: '<S2>/Sum' incorporates:
    * DiscreteIntegrator: '<S2>/Discrete-Time Integrator'
    * Gain: '<S2>/Kp'
    * Gain: '<S2>/Kp1'
    */
    rty_throt = 0.02 * rtb_Switch2 + 0.01 *
    localDW->DiscreteTimeIntegrator_DSTATE;

    /* Update for DiscreteIntegrator: '<S2>/Discrete-Time Integrator'
    */
    localDW->DiscreteTimeIntegrator_DSTATE += 0.01 * rtb_Switch2;

    if (localDW->DiscreteTimeIntegrator_DSTATE >= 5.0) {
        localDW->DiscreteTimeIntegrator_DSTATE = 5.0;
        clsc;
    }
    if (localDW->DiscreteTimeIntegrator_DSTATE <= -5.0) {
        localDW->DiscreteTimeIntegrator_DSTATE = -5.0;
    }
}
Coverage for Generated Code (R2016a)

```matlab
77 if (rtb_ActiveControl) {
78    /* Sum: '<S2>/Sum' incorporates:
79       * DiscreteIntegrator: '<S2>/Discrete-Time Integrator'
80       * Gain: '<S2>/Kp'
81       * Gain: '<S2>/Kp1'
82     */
83     rty_throt = 0.02 * rtb_Switch2 + 0.01 *
84       localIX->DiscreteTimeIntegrator_BSTATE;
85     /* Update for DiscreteIntegrator: '<S2>/Discrete-Time Integrator' */
86     localIX->DiscreteTimeIntegrator_BSTATE += 0.01 * rtb_Switch2;
87     if (localIX->DiscreteTimeIntegrator_BSTATE > 5.0) {
88        error = -5.0;
89     } else if (localIX->DiscreteTimeIntegrator_BSTATE < -5.0) {
90        error = 5.0;
91        % return error
92        return;
93     } else {
94        error = rtb_Switch1;
95     } % return error
96     % return error
97     return;
98     }
99     }
```

Can also be highlighted on model
Model Based Design Verification Workflow

- Perform PIL Testing
- Perform HIL Testing

Equivalence testing
Model Based Design Verification Workflow

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

**Modelling**

- **Component and system testing**
- **Review and static analysis**
- **Equivalence testing**
- **Equivalence checking**

**Code Generation**

**Compilation and Linking**
Model Based Design Verification Workflow
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
Thank You!