MATLAB TOUR 2017
Verificación de código generado automáticamente

Luis López
Development Lifecycle

Simulink & Stateflow

Embedded Coder

Compiler/IDE

Requirements

Models

Source Code

Object Code

When should I start verification?

And which tools should I use?
Verification with MATLAB and Simulink

```
simOut = sim('myModel');
```

Simulink & Stateflow

Requirements

Models

Source Code

Object Code

Refine Requirements

Iterate Design

Simulation Data Inspector (SDI)

Embedded Coder

Compiler/IDE

Model Advisor

MATLAB TOUR 2017
Verification with Embedded Coder

- **Requirements**
- **Models**
- **Source Code**
- **Object Code**

**Software-in-the-Loop (SIL)**

**Simulink & Stateflow**

**Processor-in-the-Loop (PIL)**

**Requirements Based Testing With PIL**

**Embedded Coder**

**Compiler/IDE**
Automated Dynamic Testing
Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL)

- Verify numerical equivalence
- Assess execution time
- Assess code coverage
- Create certification artifacts
Demo – SIL/PIL with Emulator (QEMU)
Extend Model Coverage to Code Coverage

Collect Code Coverage during SIL/PIL Simulations

- Using LDRA Testbench
- Using Simulink Verification and Validation (R2016b)
Dynamic Verification Workflow

- Use Simulink simulation to verify your models and your code
  - Requirements based tests
  - Functional tests
  - Coverage Tests

- Use Processor-in-the-Loop to
  - Assess numerical behaviour
    - Using full target toolchain and libraries
  - Gather performance metrics
  - Demonstrate testing coverage
But it’s not just Simulink based
Have I missed anything?

Does the code meet my company coding standard?

- MISRA C Checker { 2012, 2004 }
- MISRA AC AGC Subset
  - application of MISRA-C for generated code
- MISRA C++ Checker
- JSF++ Checker
Does the code match my design?

- Demonstrate that model and source code match structurally and functionally
- Provide model→code traceability data
- Reduce manual code reviews for DO-178 software
Are there any runtime errors in the system?
Polyspace in action
Polyspace product family for C/C++

- Polyspace Bug Finder
  - Quickly find bugs in embedded software
  - Check code compliance for MISRA and JSF
  - Intended for every day use by software engineers

- Polyspace Code Prover
  - Proves code to be safe and dependable
  - Deep verification of software components
  - Perform QA signoff for production ready code

Ada language also supported for proving code
Upgrading to a New Release

Multiple benefits:

✓ New features or products
✓ Latest advances in code generation

But, you have already verified code from previous release(s)

• Re-generate and re-verify the code
• Reuse and manually integrate the existing code with newly generated code
Code Reuse Across Releases (R2016b)

- Avoid re-verifying code spanning MATLAB releases
- Support simulation workflows via SIL/PIL
- Automate integration with newly generated code as part of Build action
What have I learned …

- Start verification early, using the power of MATLAB and Simulink

- Reuse your simulation tests to verify the code on real hardware with PIL
  - Gather code coverage metrics
  - Capture execution time
  - Demonstrate numerical equivalence to design

- Use static analysis to
  - Ensure code standards conformance
  - Spot weaknesses in your design
  - Prove the absence of runtime errors
Questions?