

On-Target Testing in the Simulink Model-Based Design Environment

Richard Anderson
Senior Developer MathWorks UK Ltd.

Agenda

- Introduction to Model-Based Design and automatic code generation
- Use of processor-in-the-loop (PIL) for on-target testing

Airbus Develops Fuel Management System for the A380 Using Model-Based Design



Airbus A380, the world's largest commercial aircraft.

Challenge

Develop a controller for the Airbus A380 fuel management system

Solution

Use MATLAB, Simulink, and Stateflow for Model-Based Design to model and simulate the control logic, communicate the functional specification, and accelerate the development of simulators

Results

- Months of development time eliminated
- Models reused throughout development
- Additional complexity handled without staff increases

“Model-Based Design gave us advanced visibility into the functional design of the system. We also completed requirements validation earlier than was previously possible and simulated multiple simultaneous component failures, so we know what will happen and have confidence that the control logic will manage it.”

Christopher Slack
Airbus

Eurocopter Uses Model-Based Design to Accelerate Development of DO-178B Certified Systems

Challenge

Speed up DO-178 development cycle while stabilizing system and software definitions by using models for validation and reusing the data for verification

Solution

- Develop Plan for Software Aspects of Certification (PSAC) consistent with latest recommendations from European Aviation Safety Agency (EASA) for DO-178B, taking into account DO-178C concepts for Model-Based Design
- Create models in Simulink for software architecture, high-level requirements, and low-level requirements
- Generate flight source code using Embedded Coder

Results

- Early requirements validation and execution of simulation test cases with Simulink
- Seamless object code verification by reusing simulation test cases
- EASA approval for the software certification with use of code generated by Embedded Coder



“Using Simulink for systems and software development has provided efficient means to validate the requirements and design the system and saves time on verification and validation.”

Ronald Blanrue
Eurocopter Group – Avionic System
Avionic Certification/EADS Expert

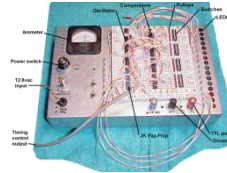
Traditional development processes prevent errors from being caught early in the program

Requirements and Specs



Text documents
prevents rapid iteration

Design



Physical prototypes
incomplete, expensive

Implementation



Manual implementation
separate tools & human error

Test and Verification



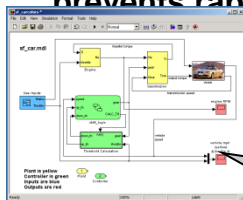
Traditional testing
errors found late in process

Can the work flow be improved?

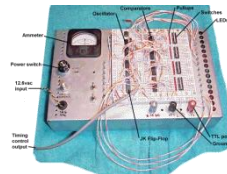
Requirements and Specs



Text documents prevents rapid



Design



Physical prototypes incomplete, expensive

Implementation



Manual implementation separate tools & human error

Test and Verification



Traditional testing errors found late in process

Executable Specification

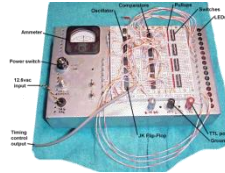
- Unambiguous spec, supplemented by text
- One set of models for all teams
- Model whole system including environment
- Block diagram description
- Early validation and test development

Requirements and Specs

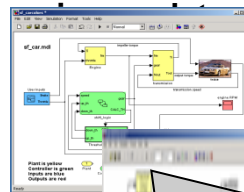


Text documents
prevents rapid iteration

Design



Physical prototypes



Implementation



Manual implementation
separate tools & human error

Test and Verification



Traditional testing
errors found late in process

Design with Simulation

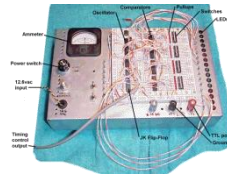
- Systematic design exploration and optimisation
- Find flaws before implementation
- Bit/cycle-accurate simulation of hardware-specific components
- Incremental design from system level to implementation

Requirements and Specs



Text documents
prevents rapid iteration

Design

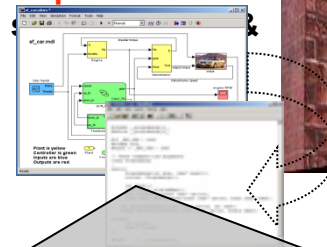


Physical prototypes
incomplete, expensive

Implementation



Manual implementation



Test and Verification



Traditional testing
errors found late in process

Automatic Code Generation

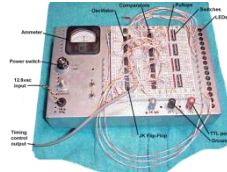
- No manual coding errors
- Hardware target portability
- Improved testability due to repeatability
- Bridge between domain, software and hardware knowledge

Requirements and Specs



Text documents
prevents rapid iteration

Design



Physical prototypes
incomplete, expensive

Implementation

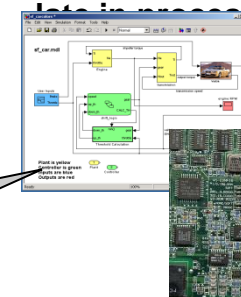


Manual implementation
separate tools & human error

Test and Verification



Traditional testing
errors found late in process



Continuous Test and Verification

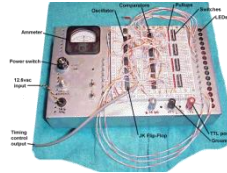
- Detect errors early in development
- Reduce dependency on physical prototypes
- Implementations that work the first time
- Reuse test suites across development stages

Model-Based Design

Requirements and Specs



Design



Implementation

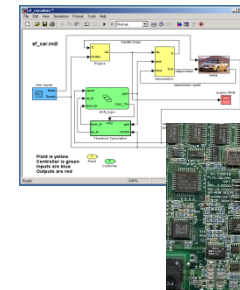
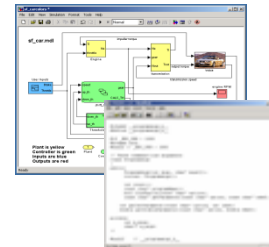
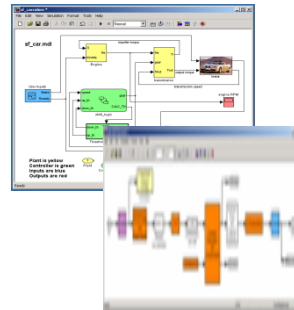
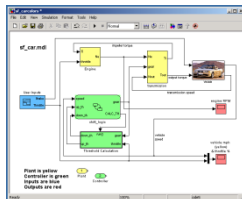


Test and Verification



Model elaboration

Continuous verification



Benefits of the tools

- MATLAB and Simulink provide a **flexible software environment for designing multi-domain systems**, simulating high-fidelity behavioural dynamics, testing and analysis, and generating safety-critical computer code
- MATLAB and Simulink **promote agility and communication along the supply chain**, by providing a common software environment for sharing data, designs, and specifications across organisations
- This approach **minimizes program risk** and enables teams to **develop mission-critical systems faster**

Model-Based Design

Engineers and scientists worldwide rely on our products to accelerate the pace of discovery, innovation, and development



Process	Previous time taken	Model-Based Design time taken
Overall development effort	≈ 100 man years	≈ 10 man years
Original design to code (1 st time model elaboration)	> 2 years	3 months
Subsequent design iterations	> 2 months	< 1 week
Testing	> 2 weeks	8 hours
Documentation update	> 2 weeks	10 minutes

Continuous Verification

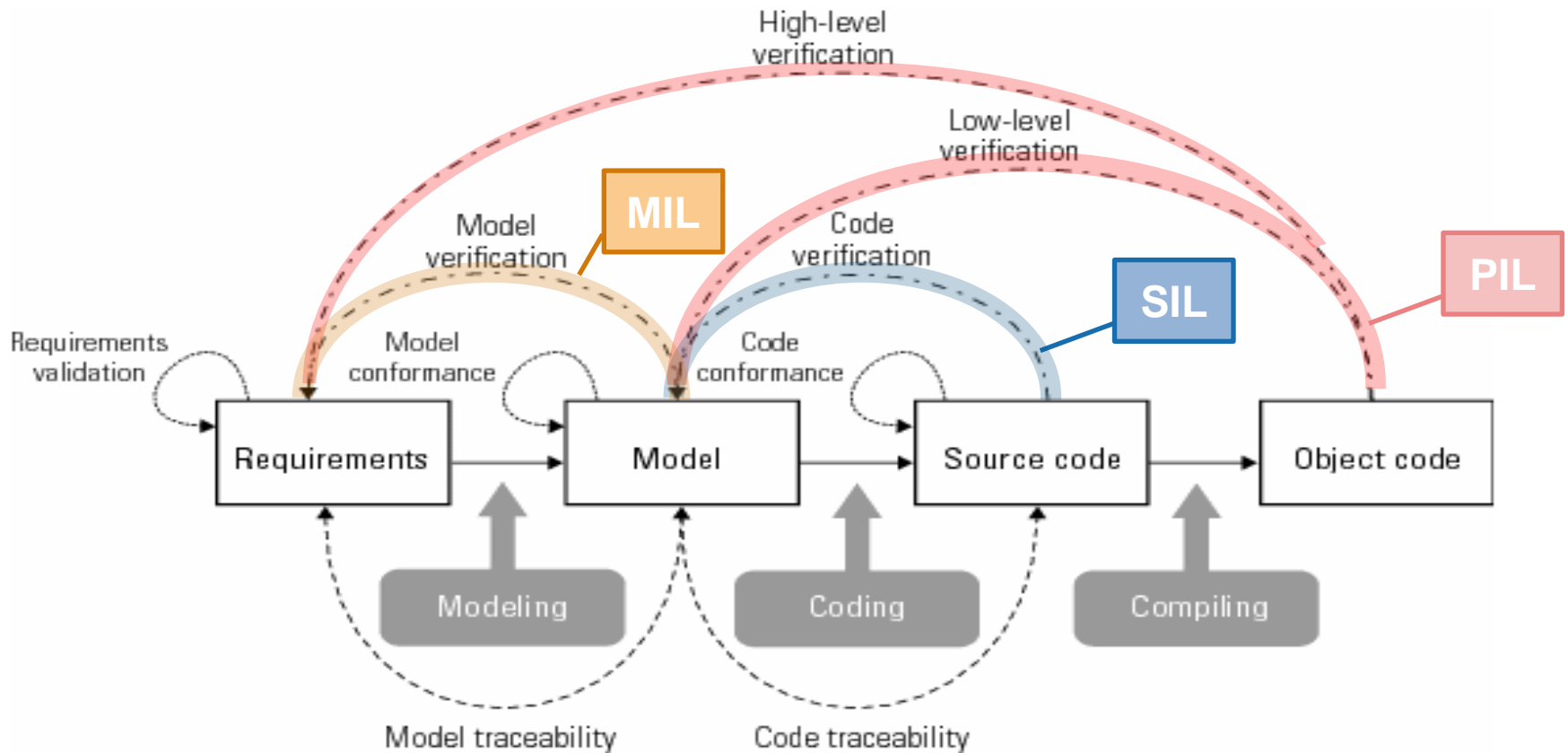
- Executable specifications allow system behaviour/performance to be continually evaluated
- Our Verification tools include
 - Static Analysis
 - Standards conformance
 - Dynamic Verification

Dynamic Verification within Simulink

- Examine behaviour of system using
 - Closed loop plant models
 - Captured input data
 - Specific test cases
 - Auto-generated test cases
 - To demonstrate particular system properties
 - Round out test coverage

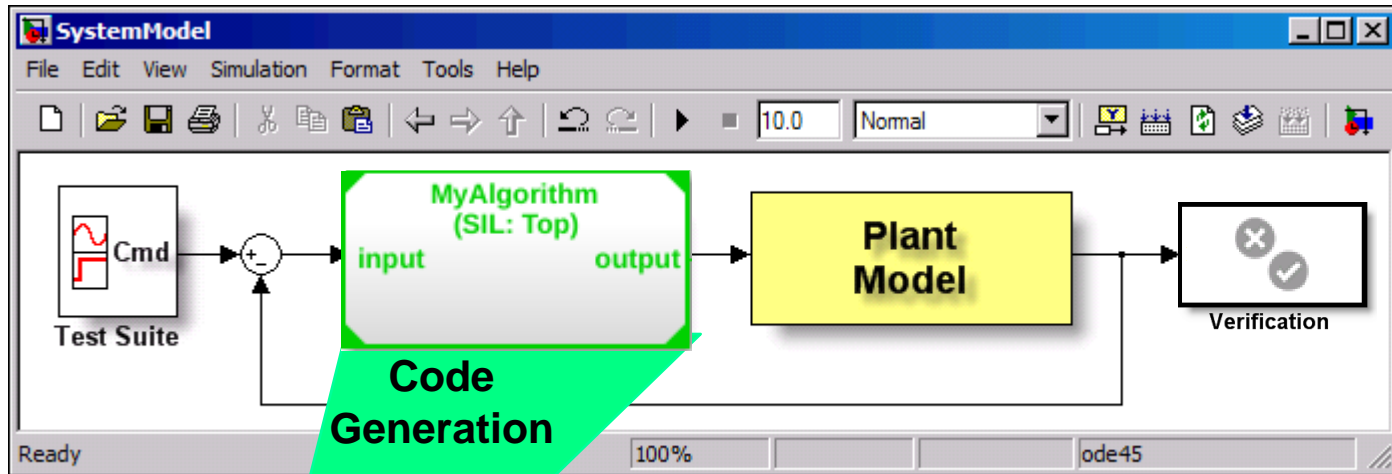
- In-the-loop testing extends these to the generated code
 - Software-in-the-Loop Testing
 - Processor-in-the-Loop Testing

Simulation (MIL), SIL and PIL within the High Integrity Workflow

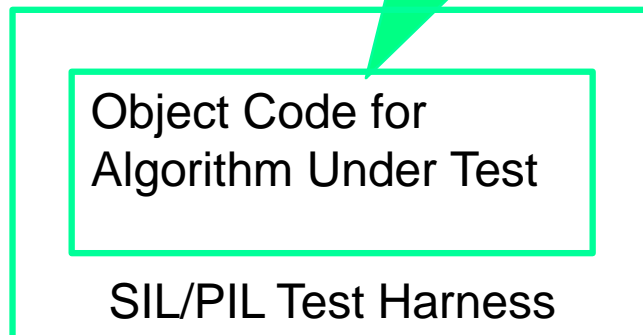


How SIL and PIL work

The Test Harness



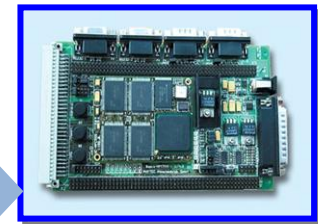
Code Generation



MATLAB Host computer (SIL)

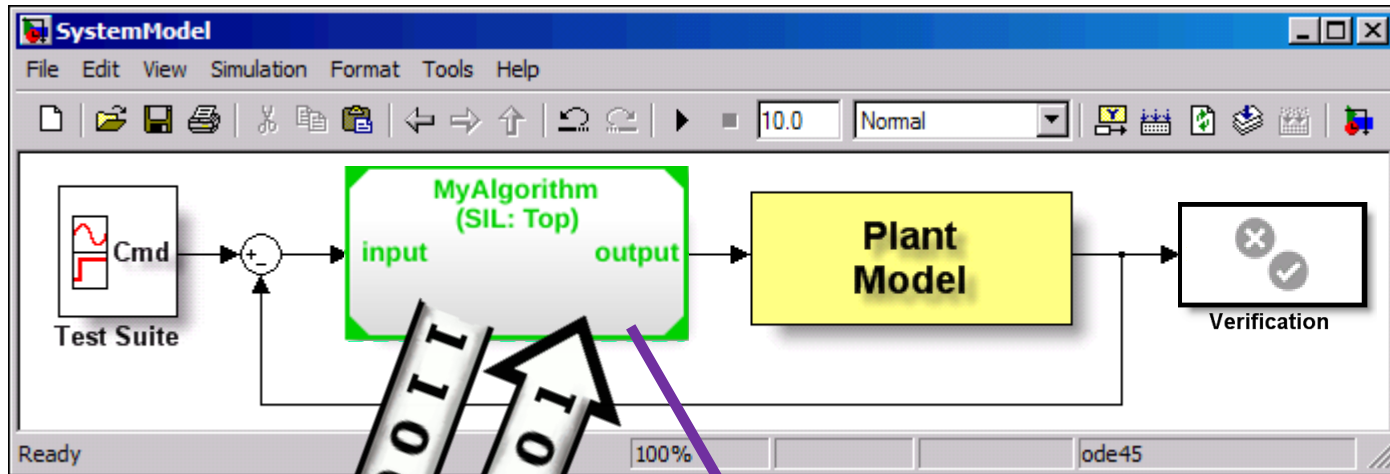


Production Processor (PIL)

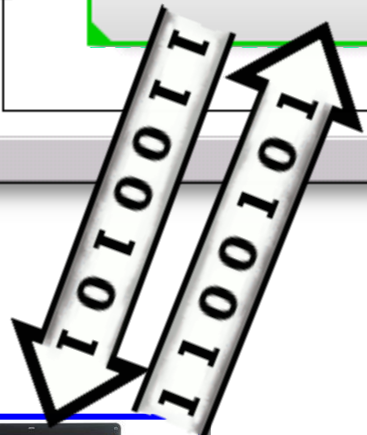


Software-in-the-Loop (SIL)

Verify compiled object code matches simulation



Non-real-time execution:
synchronized with simulation

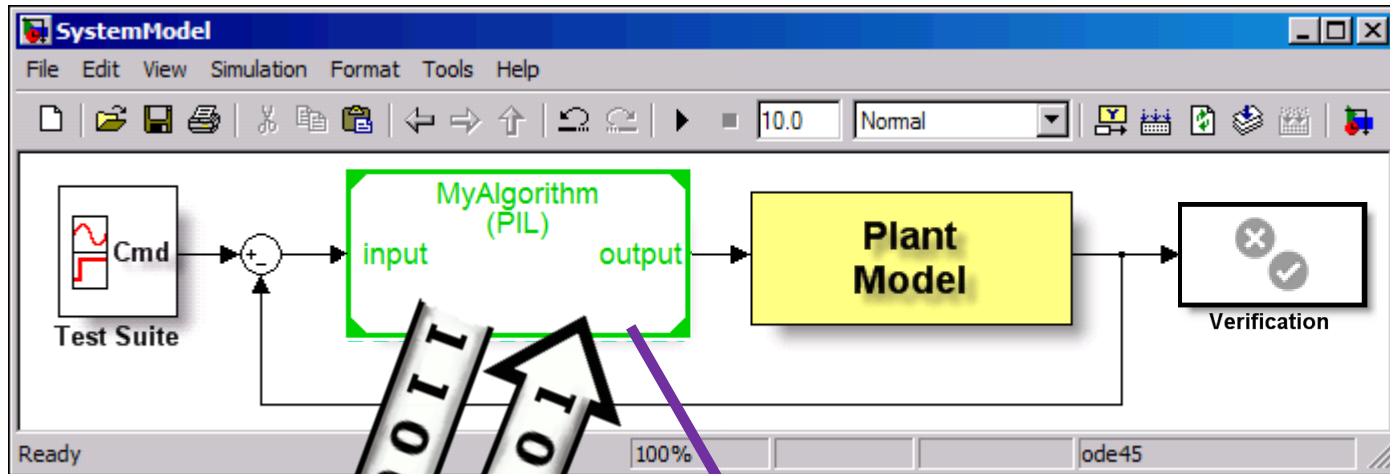


- Verify numerical equivalence
- Assess execution time
- Collect code coverage
- Create certification artifacts

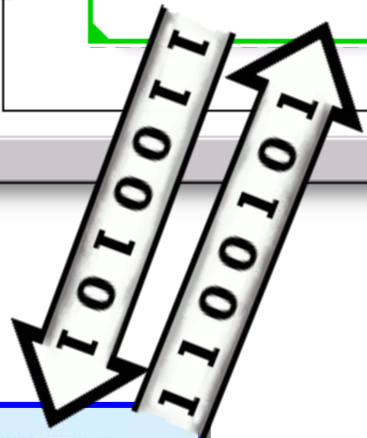
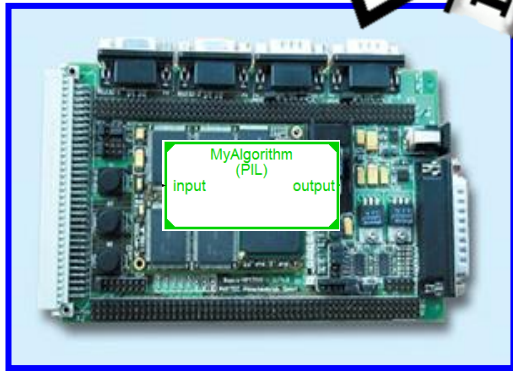
- **Software-in-the-Loop (SIL)** No additional tools / hardware required

Processor-in-the-Loop (PIL)

Verify compiled object code matches simulation



Non-real-time execution:
synchronized with simulation



- Verify numerical equivalence
- Assess execution time
- Collect code coverage
- Create certification artifacts

- **Processor-in-the-Loop (SIL)** for testing on production hardware

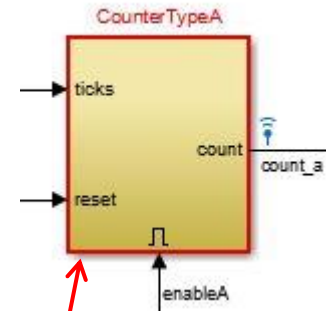
Key Benefits of SIL and PIL

- Controlled and easy to debug environment owing to non-real-time execution in the context of a Simulink simulation
- Verify correct execution behaviour of compiled code
- Collect metrics for the generated code
 - Code coverage (Bullseye, LDRA)
 - Execution profiling
- Evaluate hardware specific optimisations
- Generate artefacts for IEC 61508, ISO 26262, EN 50128, and DO-178 certification
- Early verification and defect detection reduces costs

Function Execution Times for Code Running on Embedded Hardware

Comprehensive measurement of function call execution times

- Call-site instrumentation to measure execution time of functions in the generated code
- Includes initialization, shared utility and math library functions
- Configurable units for reporting of measured execution times



Default units are nanoseconds

Model	Code Section Number	Maximum Execution Time
[-] rtwdemo_sil_topmodel_initialize	<u>1</u>	202
[+] CounterTypeA	<u>2</u>	105
[+] CounterTypeB	<u>4</u>	43
[-] rtwdemo_sil_topmodel_step[0,10]	<u>6</u>	1074
CounterTypeA	<u>7</u>	399
CounterTypeB	<u>8</u>	44

```
/* Start for Enabled SubSystem: '<Root>/CounterTypeA'
PROFILE_START_809d18550fe660e8 (2147483645U);
CounterTypeA_Start();
PROFILE_END_rt_043493862f001c94 (2147483645U);
```

```
PROFILE_START_TASK_SECTION(2147483640U);
rtwdemo_sil_topmodel_step();
PROFILE_END_TASK_SECTION(2147483640U);
```


Target Support Packages available for Download

Support Package Installer

Show: All (72)

Support for:

- ARM Cortex-A
- ARM Cortex-M
- ARM Cortex-based VEX Microcontroller
- AUTOSAR Standard
- Altera FPGA Boards
- Altera SoC
- Analog Devices DSPs
- Android Sensors
- Apple iOS
- Arduino
- BEEcube miniBEE Platform
- BeagleBoard
- BeagleBone Black
- DCAM Hardware
- Data Translation Frame Grabbers
- Digilent Analog Discovery
- DirectSound Audio
- Freescale Kinetis Microcontrollers
- GenICam Interface
- GigE Vision Hardware
- Hamamatsu Hardware
- IP Cameras
- Kinect for Windows Sensor
- Kvaser CAN Devices
- LEGO MINDSTORMS EV3
- LEGO MINDSTORMS NXT
- Matrox Hardware
- NI Frame Grabbers
- NI-845x I2C/SPI Interface
- NI-CAN
- NI-DAQmx

Installation folder:

Action	Installed Version	Latest Version	Description	Required Base Product	Supported Host Platforms
1 <input checked="" type="checkbox"/> Install					
2 <input checked="" type="checkbox"/> Install					

http://www-integ2.mathworks.com/hardw

Hardware Support

[Contact support](#)
[Contact sales](#)
[Trial software](#)

[Overview](#)
[Search Hardware Support](#)
[Request Hardware Support](#)

- ▶ ARM Cortex A Support from Embedded Coder
- ▶ ARM Cortex-M Support from Embedded Coder
- ▶ ARM Cortex-M CMSIS Library Support from DSP System Toolbox
- ▶ ARM Cortex A Ne10 Library Support from DSP System Toolbox

Expand all

▼ Getting Started Resources

▼ Videos

- ARM Cortex-A, -R, -M Optimized Code Generation using MATLAB and Simulink 48:36

▶ Examples

▶ Documentation

▶ Community


▶ Other

▶ MathWorks Requirements

▶ Third-Party Requirements

[Request Hardware Support](#)

MATLAB Coder™, Simulink Coder™, and Embedded Coder® generate ANSI/ISO C/C++ code that can be compiled and executed on ARM® Cortex® A processors. Embedded Coder lets you easily configure the code generated from MATLAB® and Simulink® algorithms to control software interfaces, optimize execution performance, and minimize memory consumption.

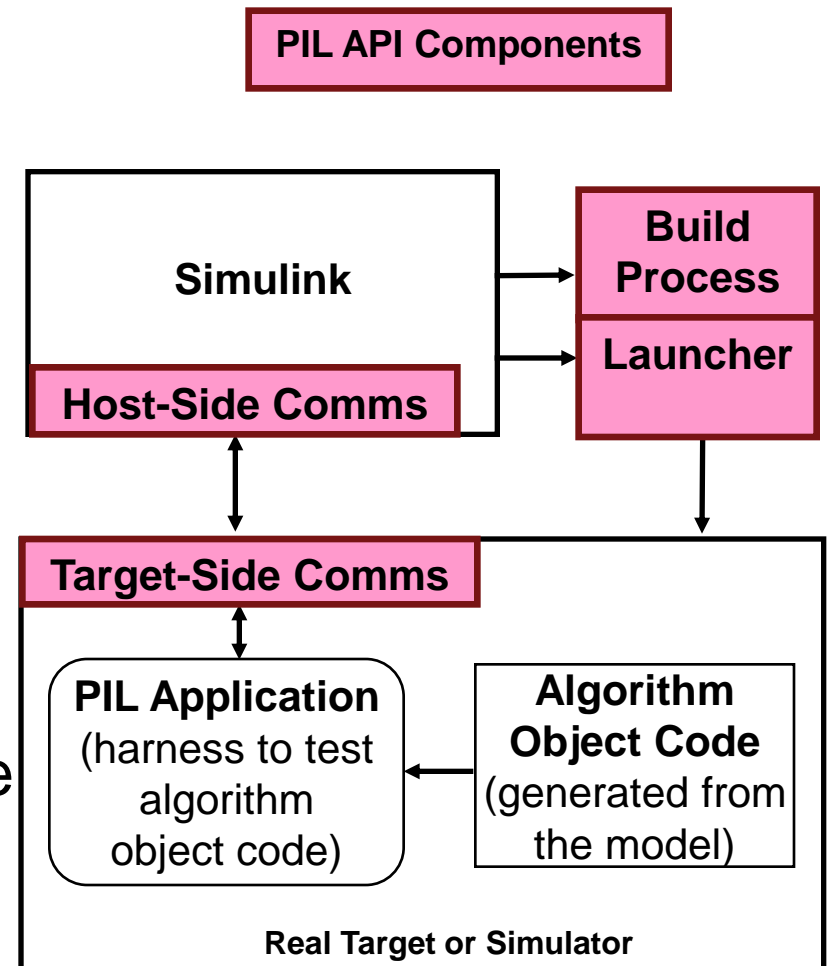


ARM Cortex-A processors compatible with Embedded Coder generated code include:

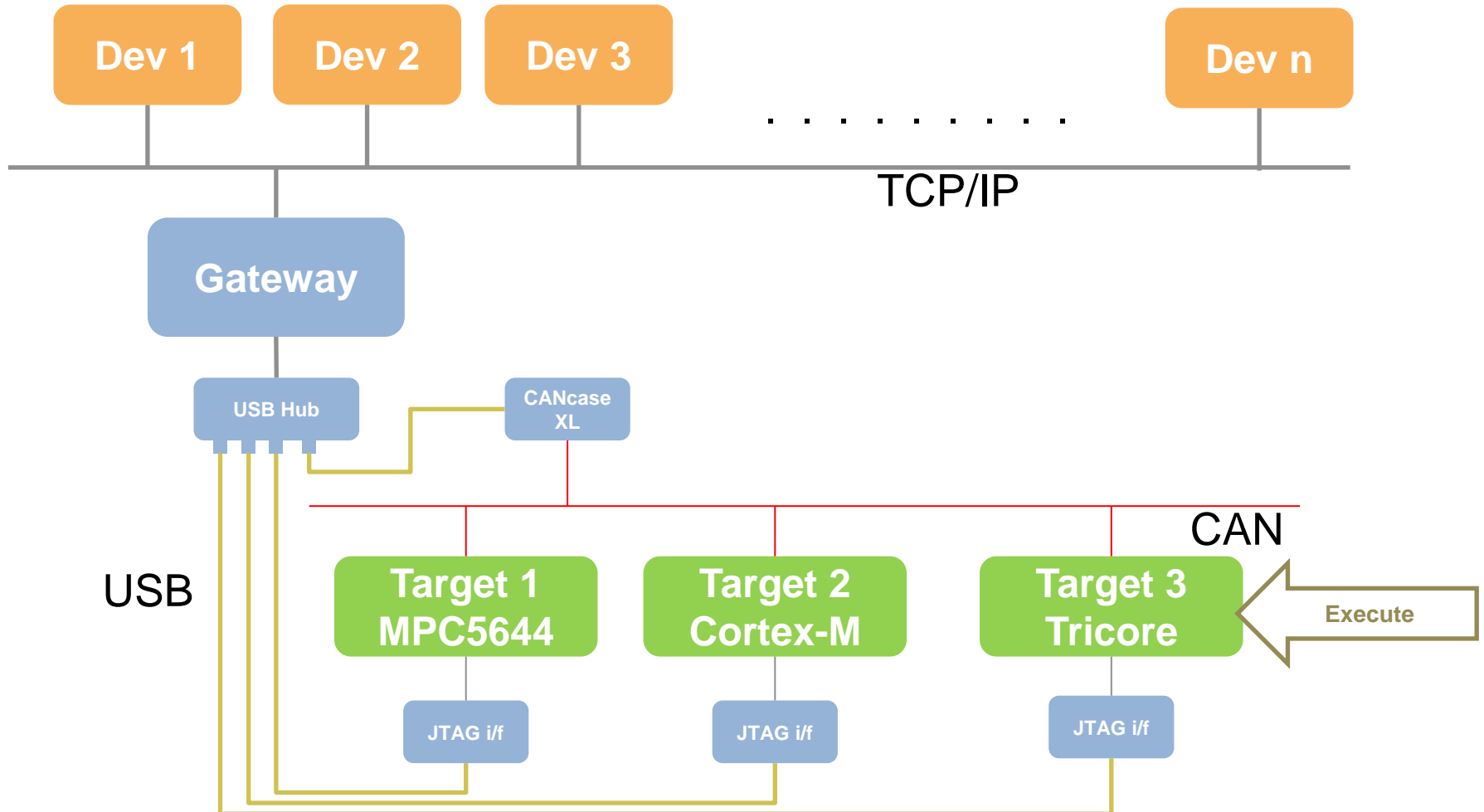
Cortex – A72
Cortex – A57
Cortex – A53
Cortex – A17
Cortex – A15
Cortex – A9
Cortex – A8
Cortex – A7
Cortex – A5

Integration API to extend to your Hardware

- There is a growing list of Support Packages
- Support Packages cannot support PIL for an arbitrary combination of
 - Processor
 - Compiler
 - Debugger or download utility
 - Communications channel
- A fully documented API stable across MathWorks releases



Multiple PIL Targets on the Network



Takeaways

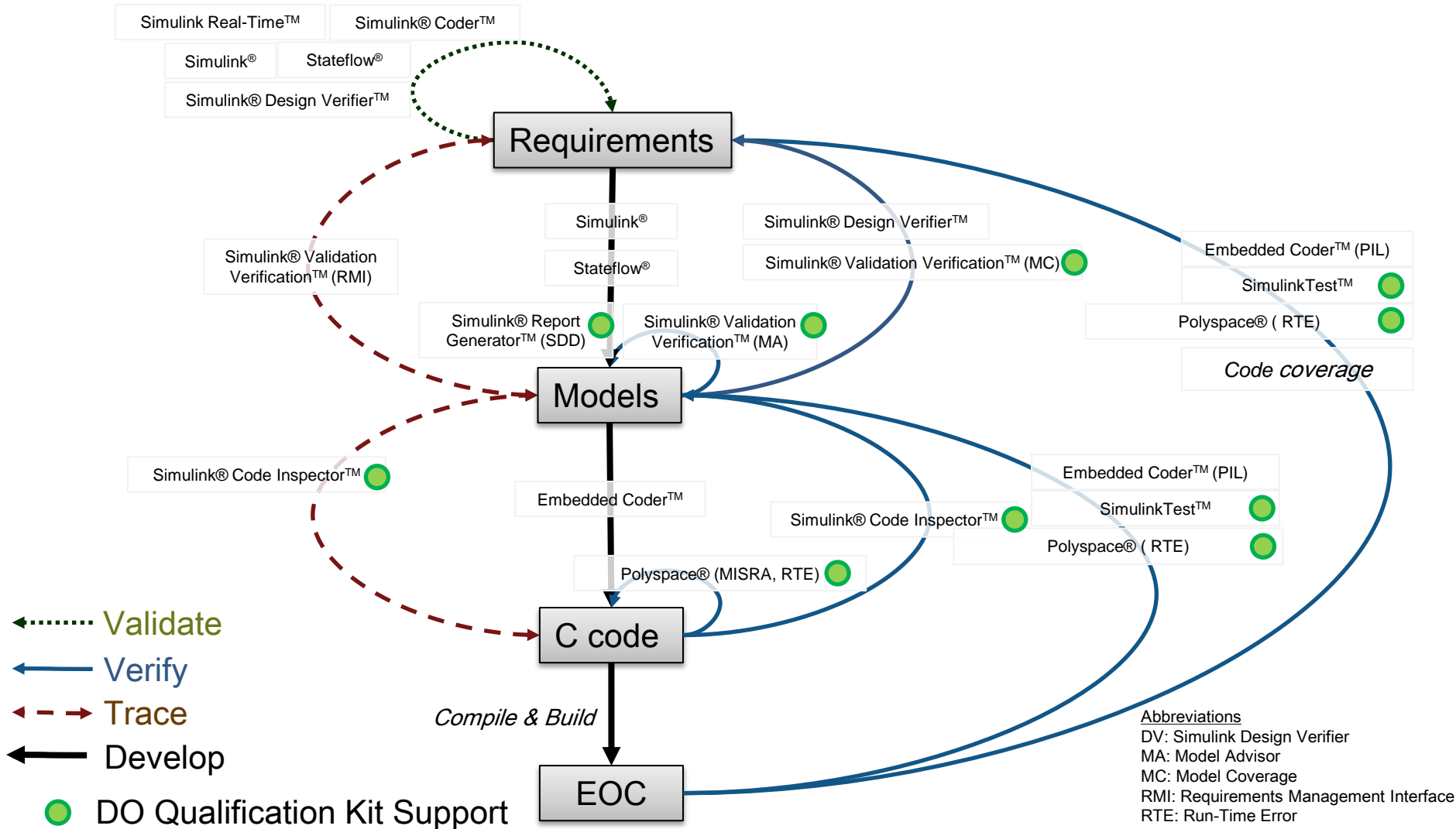
- Model Based Design provides
 - Easier collaboration, bringing
 - cost savings, and
 - innovation
 - Earlier problem detection
 - An Executable specification supporting system optimisation and exploration throughout the design lifecycle
- Processor-in-the-Loop promotes
 - Early and accessible on-target testing
 - Gathering metrics
 - Easy reuse of tests with real production hardware

Questions?

- Thank you
 - Richard.Anderson@Mathworks.co.uk

Overview of MathWorks Tool Chain

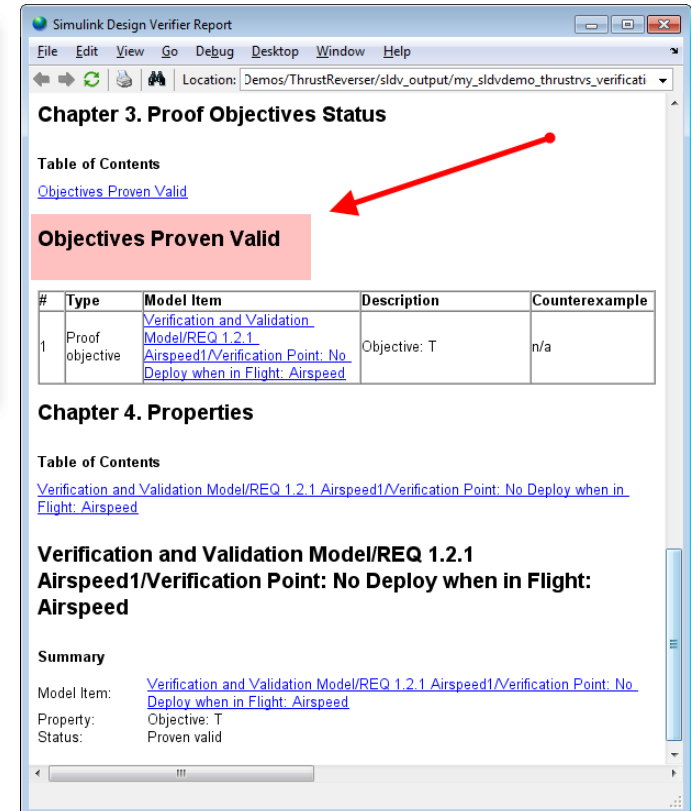
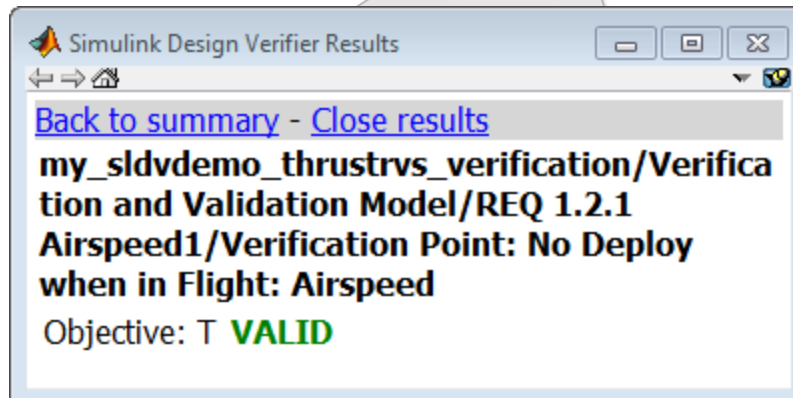
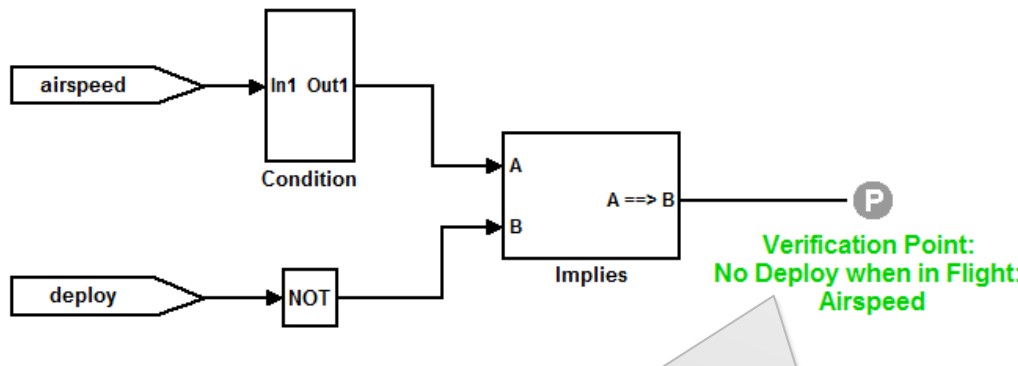
Example of DO-178 Software life-cycle



Simulink Design Verifier

Formal methods on model to:

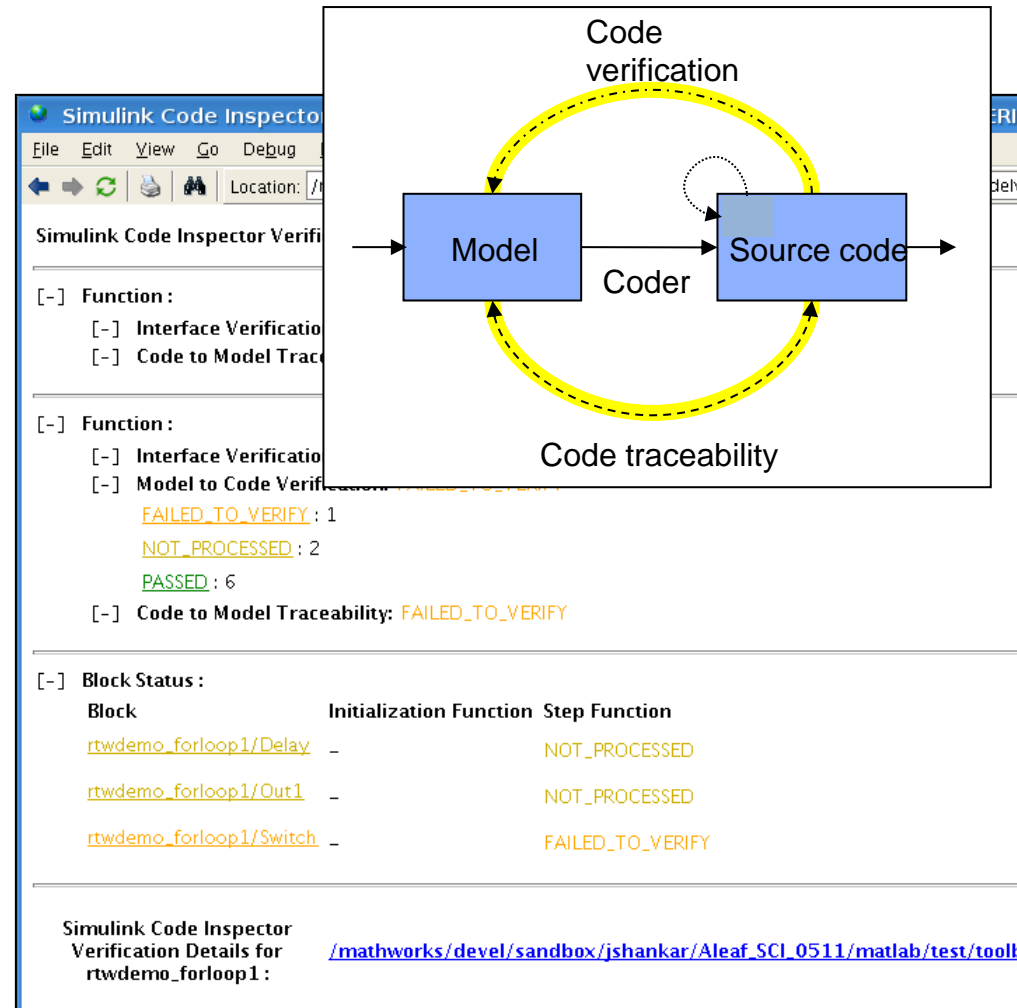
- Generate Tests (or find untestable algorithm)
- Detect Design Errors
- Prove Properties: e.g. thrust reverser shall not deploy when *[Condition]*



Automatic Code Reviews

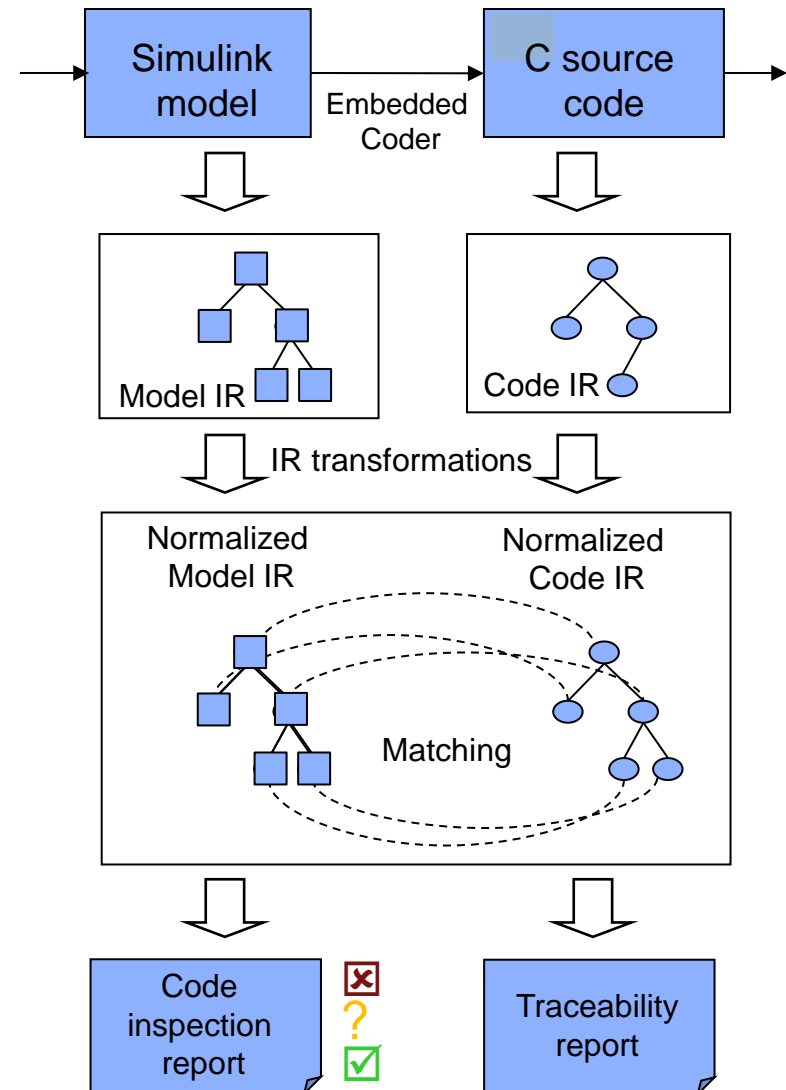
- Simulink Code Inspector

- Demonstrate that model and source code match structurally
- Provide model \leftrightarrow code traceability data
- Eliminate / reduce manual code reviews for DO-178 software
- Independence from Coder
- Same credits as qualified coder



Simulink Code Inspector

- **Independently verify that Embedded Coder generated code traces to and complies with low-level requirements**
- Demonstrate that model and source code match structurally and functionally
- Provide model \leftrightarrow code traceability data
- Eliminate/reduce manual code reviews for DO-178 software



How does *Polyspace* help you?



- Finds bugs
- Checks coding rule conformance (MISRA/JSF/Custom)
- Provides metrics (Cyclomatic complexity etc.)
- Proves the existence and absence of errors
- Indicates when you've reached the desired quality level
- Certification help for DO-178 C, ISO 26262, ...

Code Coverage via On-Target (PIL) Simulation

- Code coverage via SIL is fully automated
 - Using LDRA Testbed or Bullseye
- Use of PIL for code coverage is an alternative to code coverage via SIL
- Code coverage via PIL
 - Fully automated if target (e.g. instruction set simulator) can write directly to the host file system
 - Possible for any target using custom approach for data collection
- Code coverage via PIL is as simple as code coverage via SIL