

# MATLAB EXPO 2016

## KOREA

4월 28일 (목)

등록 하기 [matlabexpo.co.kr](http://matlabexpo.co.kr)



# Product Code Generation and Real-Time Testing

김종헌 차장

**Senior Application Engineer**

**MathWorks Korea**

# Agenda

- Production Code Generation
  - MathWorks' Code Generation Products
  - Embedded Coder
  - Equivalence Test with SIL and PIL
  
- Integration Test
  - What's Simulink Real-Time
  - Automation of Real-Time Testing

# Code Generation Products

## **MATLAB Coder**

Generate C and C++ code from MATLAB code

## **Simulink Coder**

Generate C and C++ code from Simulink and Stateflow models

## **Embedded Coder**

Generate C and C++ code optimized for embedded systems



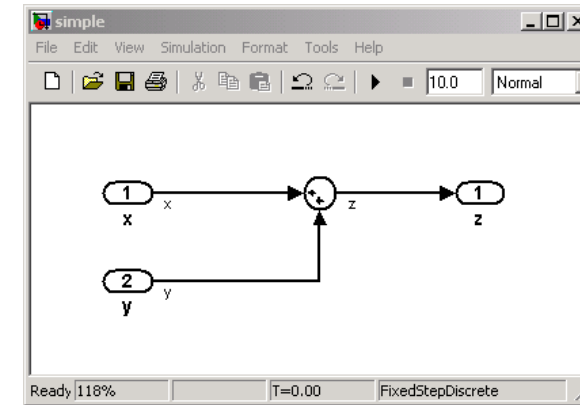
# Code Generation Products: Simulink Coder and Embedded Coder

## Simulink Coder

- Generates code for use in simulation and prototyping applications
- Comes with Generic Real-Time (GRT) based targets

## Embedded Coder

- Generates efficient code that can be customized to look like hand code for production
- Comes with Embedded Real-Time (ERT) based targets

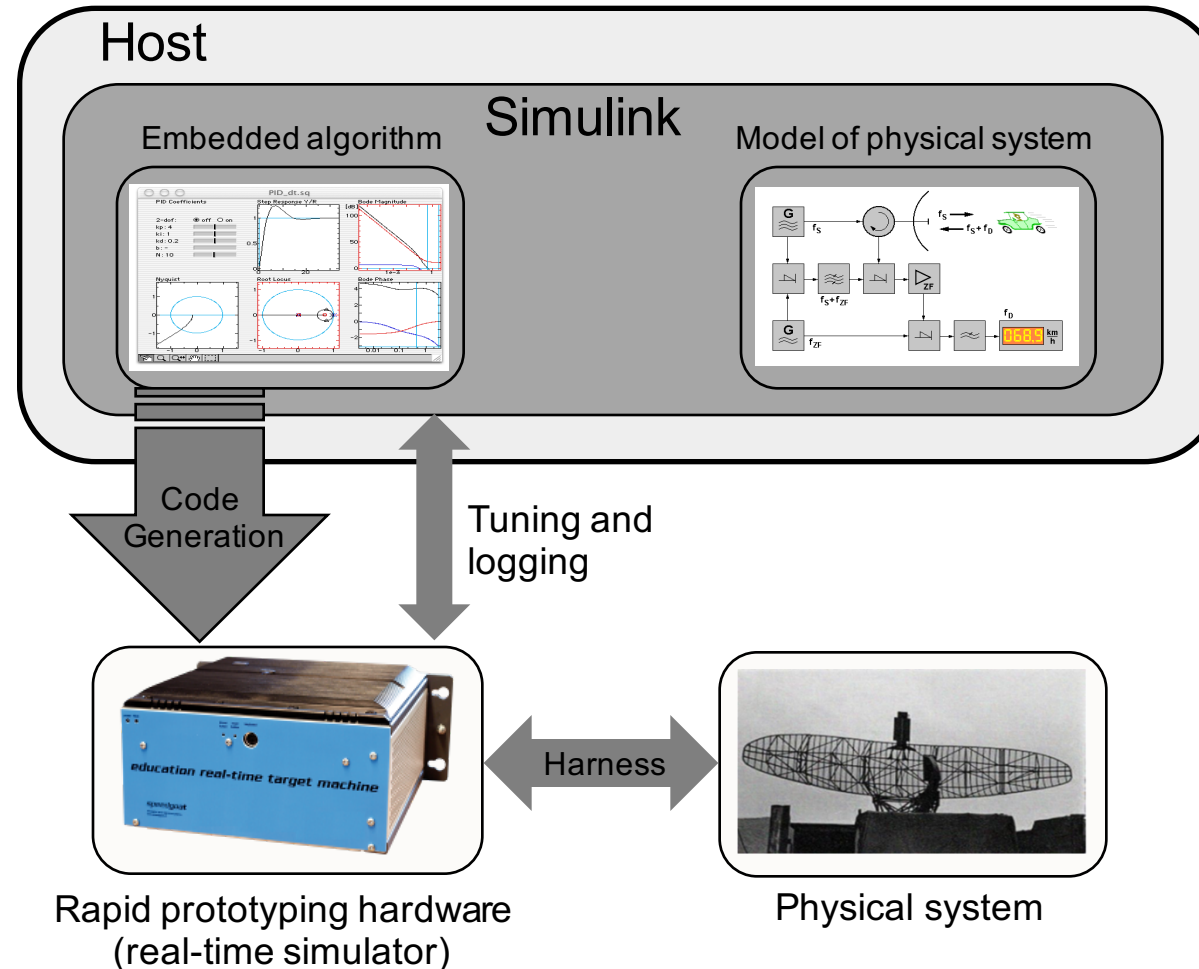
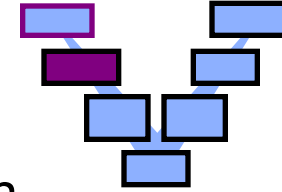


```
File: d:\matlab7\work\simple_ert_rt\simple.c  
1 #include "simple.h"  
2 #include "simple_private.h"  
3  
4 real_T z;  
5  
6 void simple_step(void)  
7 {  
8  
9     z = x + y;  
10  
11 }  
12  
13 void simple_initialize(boolean_T firstTime)  
14 {  
15 }  
16
```

# Rapid Prototyping

## Simulink Coder with Simulink Real-Time

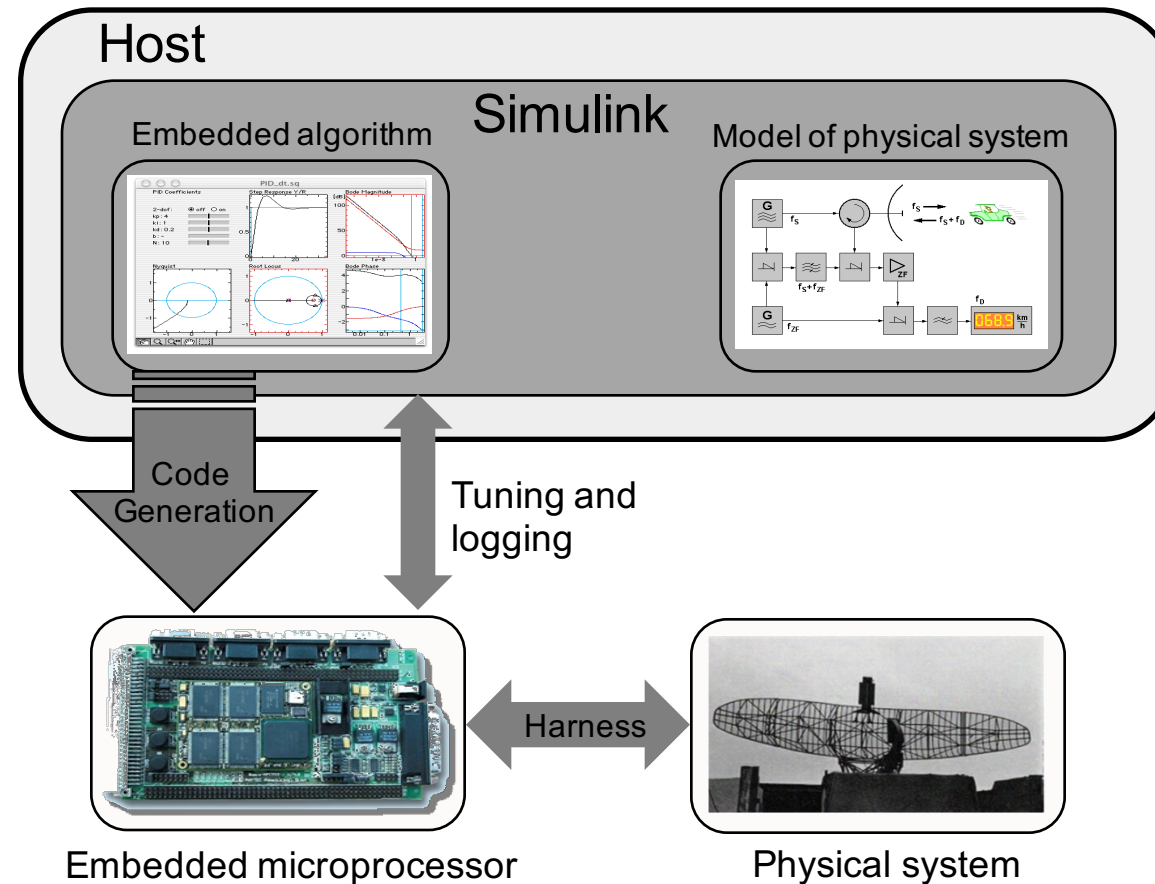
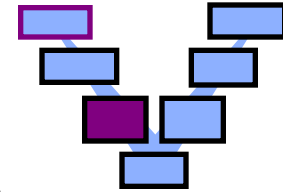
Generate, deploy, and tune code for a component (algorithm or controller) on a real-time simulator connected to system hardware



# Rapid Prototyping on Embedded Processors

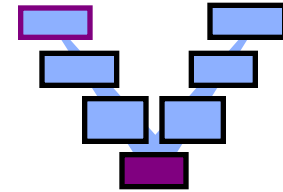
## Embedded Coder

Run the generated code in real time, tune parameters, and monitor real-time data on the same processor you plan to use in mass production, or a close equivalent to it.

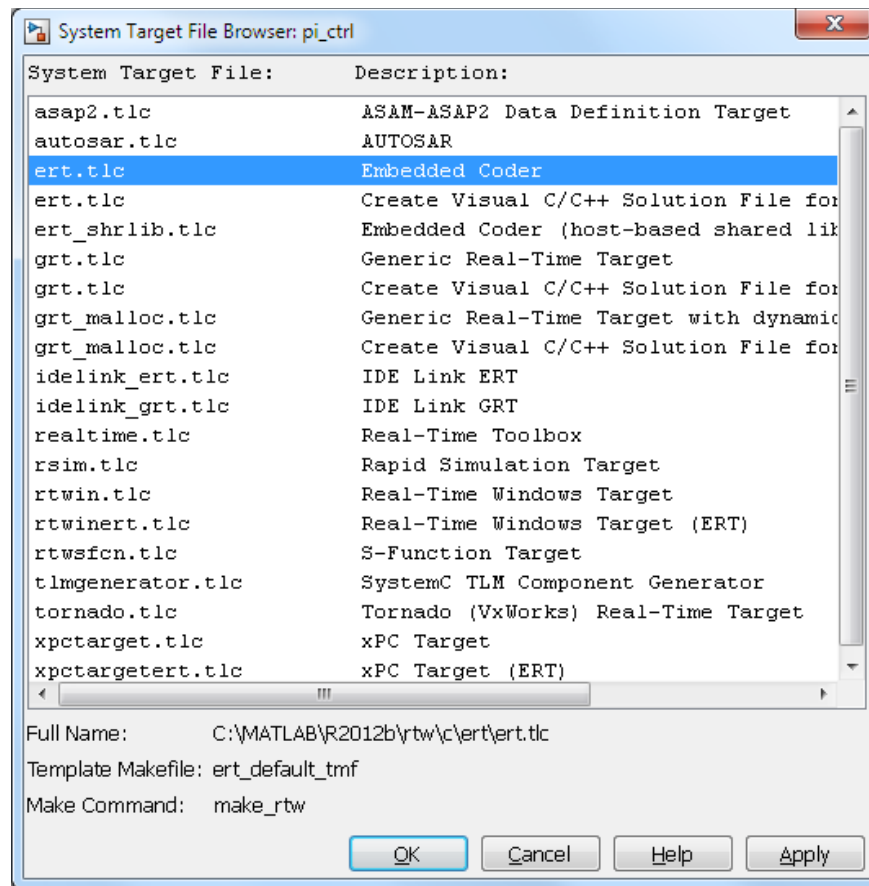


# Production Code Generation

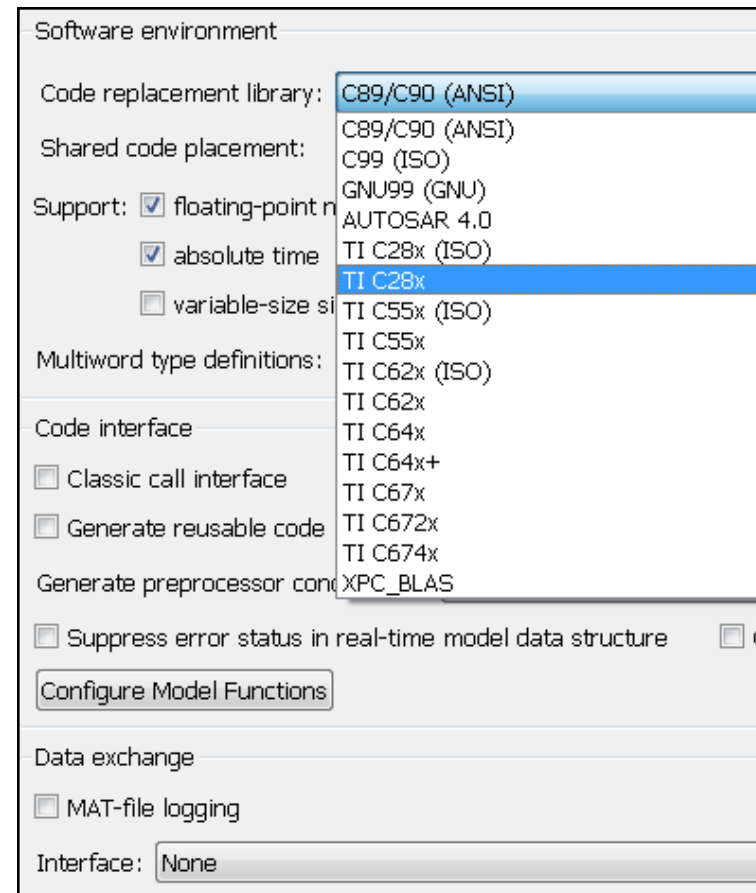
## Embedded Coder



### Select Target

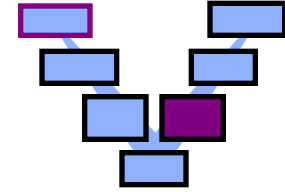


### Choose Optimizations and File Packaging

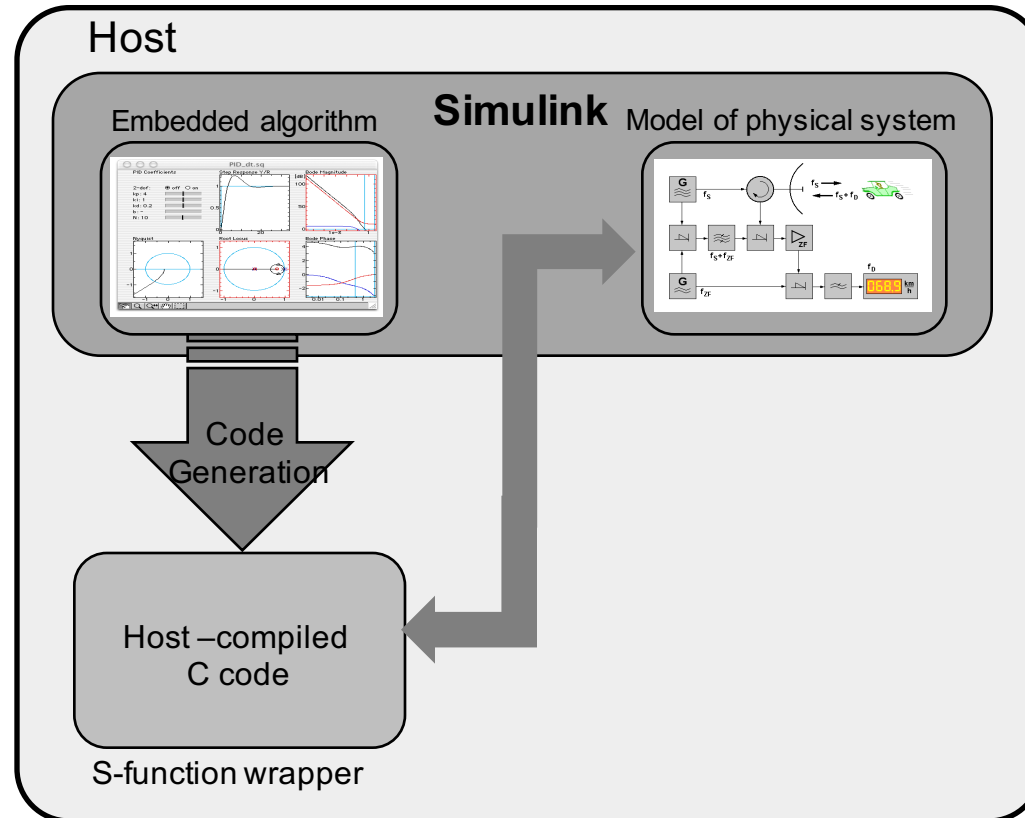


# Software-in-the-Loop Testing

## Embedded Coder

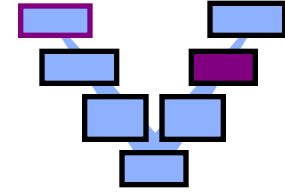


Test generation production code with your environment or plant model to verify a successful conversion of the model to code.

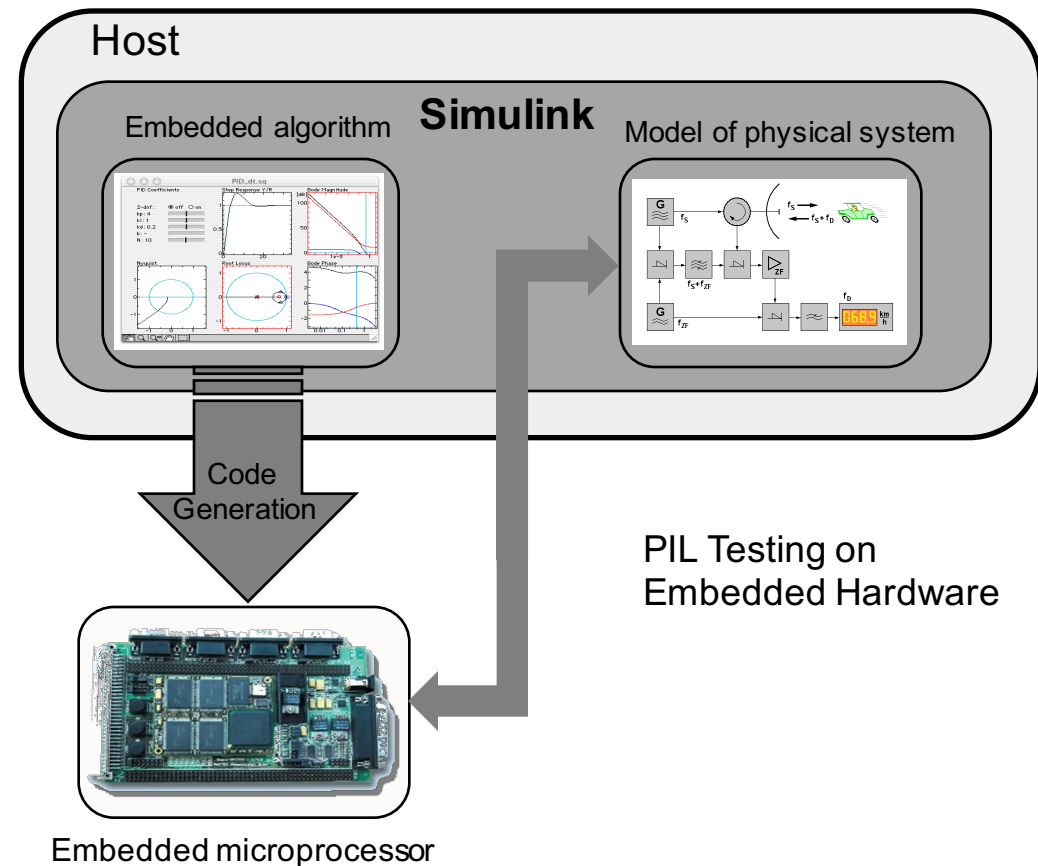


# Processor-in-the-Loop Testing

## Embedded Coder



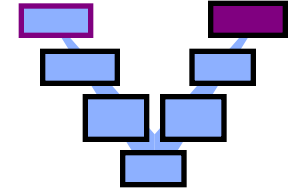
Use processor-in-the-loop PIL to evaluate the behavior of a candidate algorithm on the target processor.



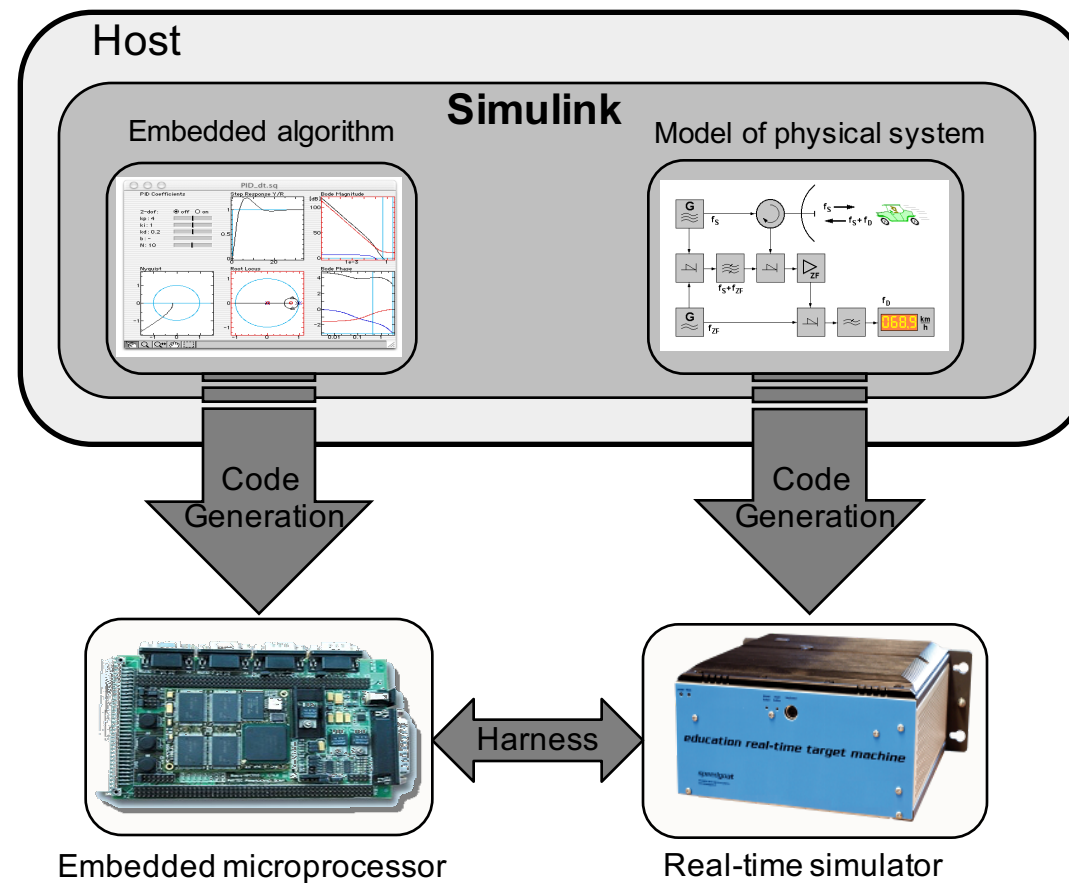


# Hardware-in-the-Loop Testing

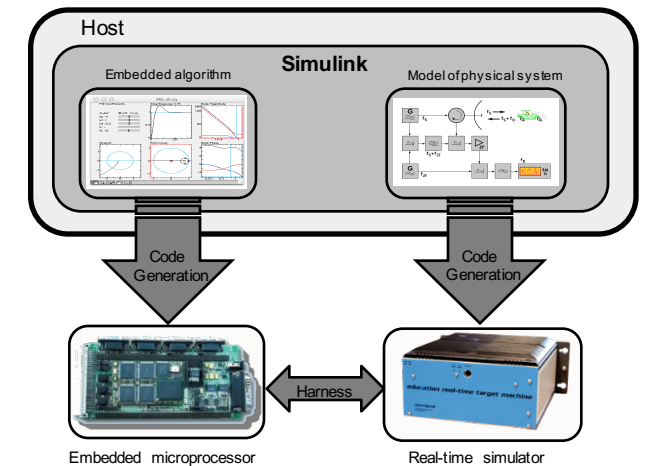
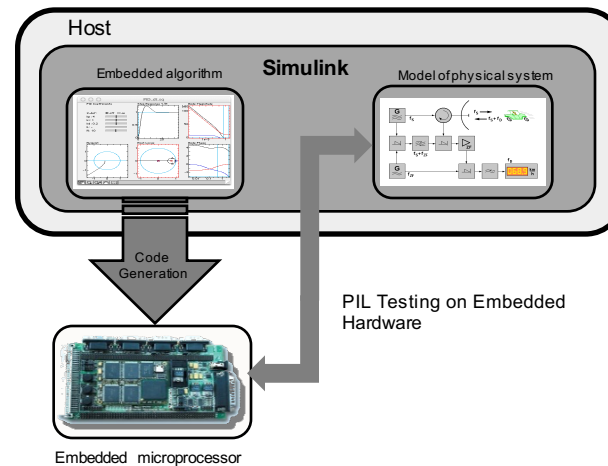
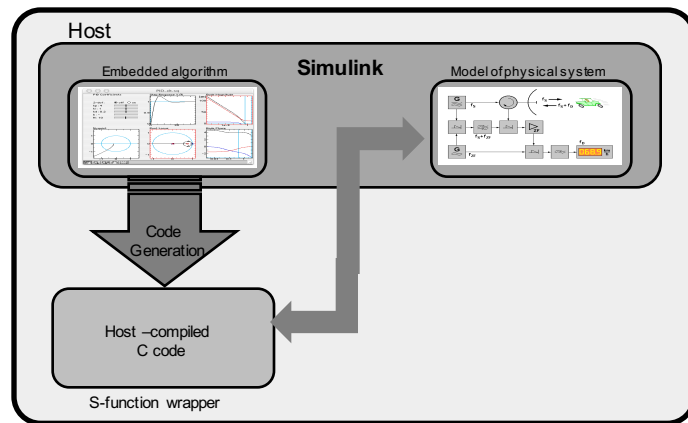
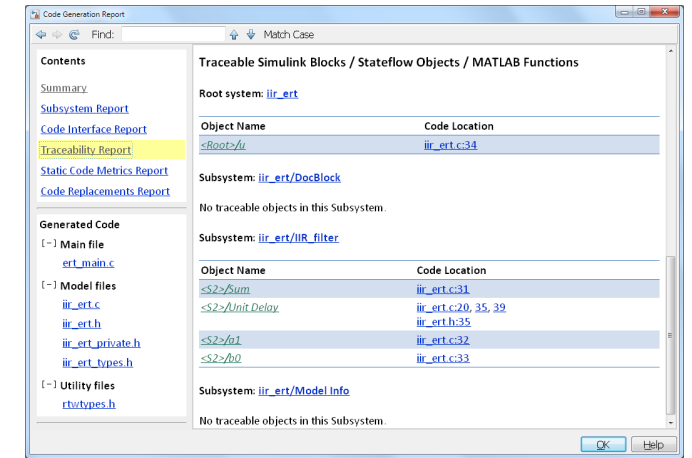
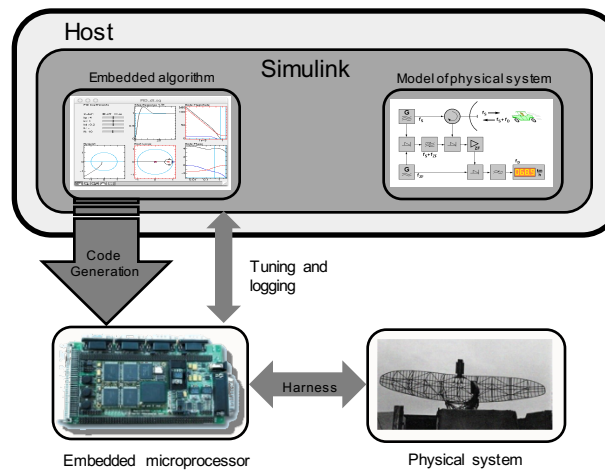
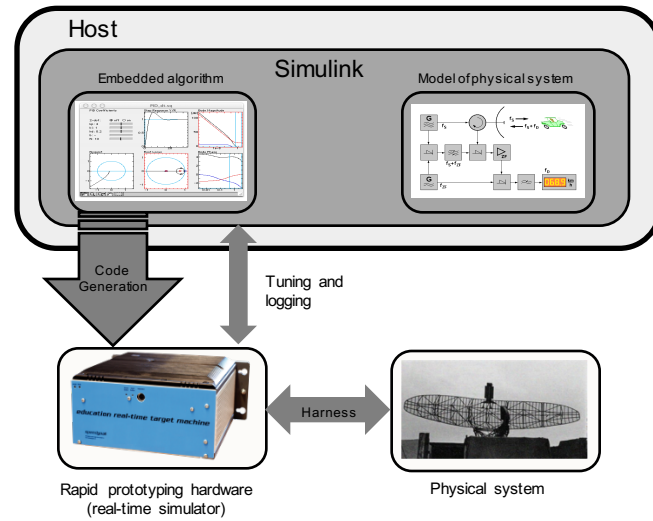
## Embedded Coder and Simulink Coder with Simulink Real-time



Final test before integration using simulated plant executing in real time.



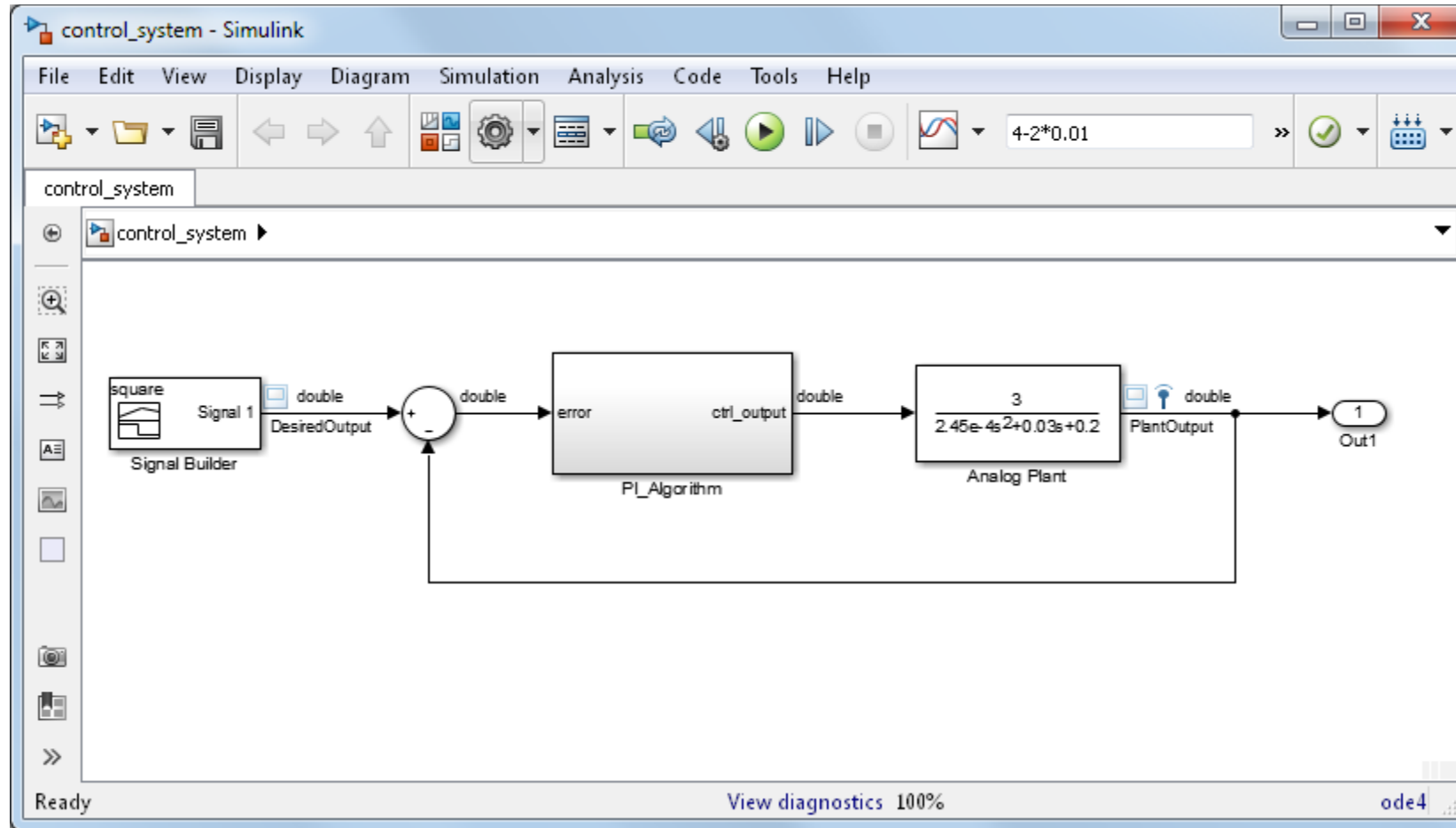
# Usage of Code Generation Products



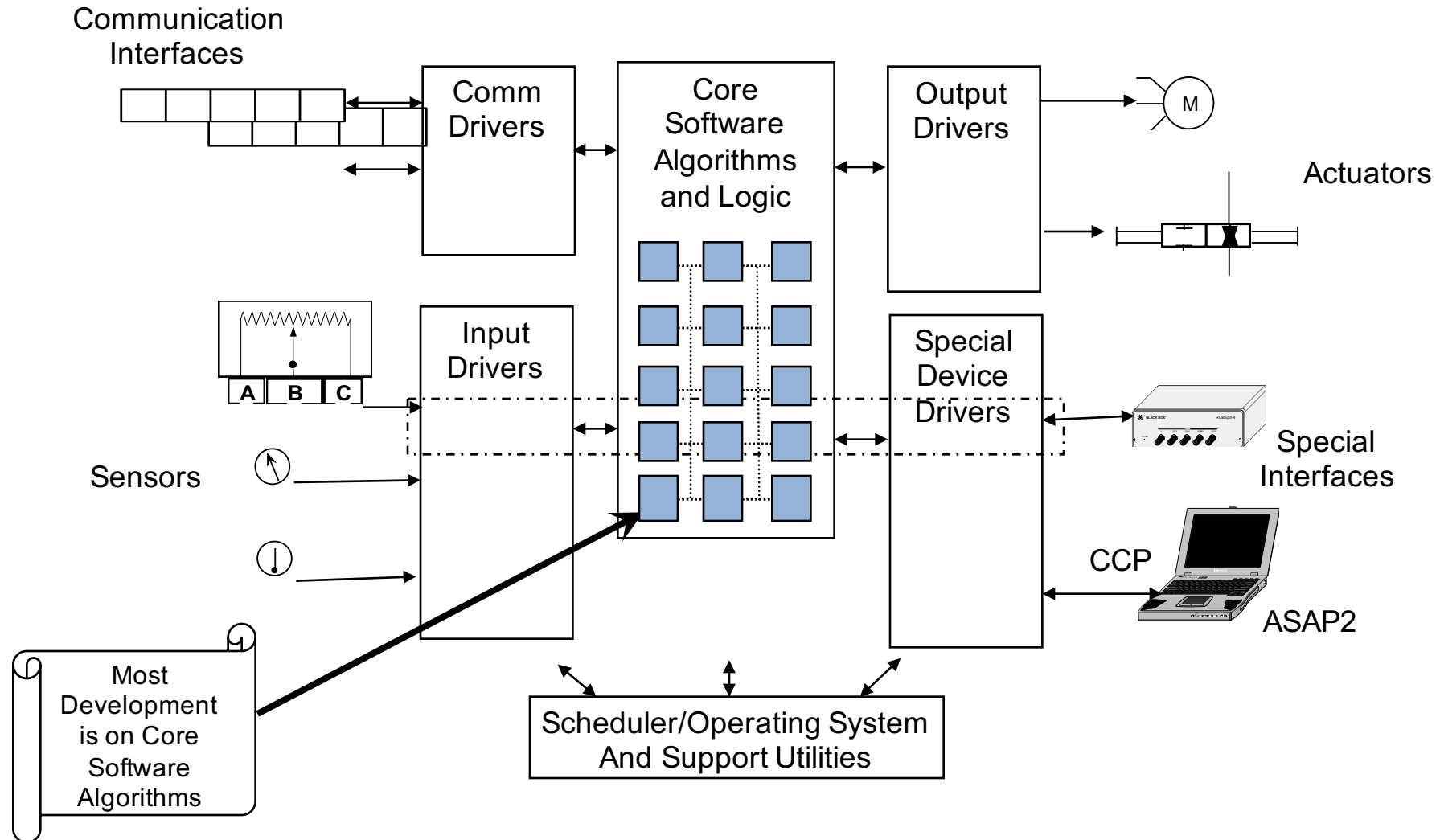
# Agenda

- Production Code Generation
  - MathWorks' Code Generation Products
  - Embedded Coder
  - Equivalence Test with SIL and PIL
  
- Integration Test
  - What's Simulink Real-Time
  - Automation of Real-Time Testing

# Example) Controller of Landing Gear

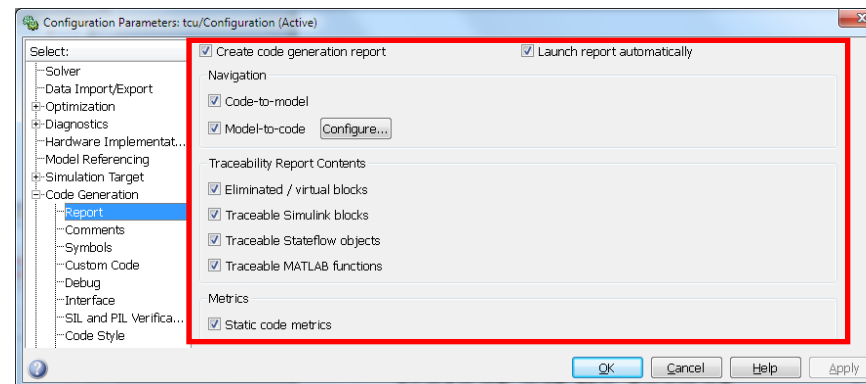
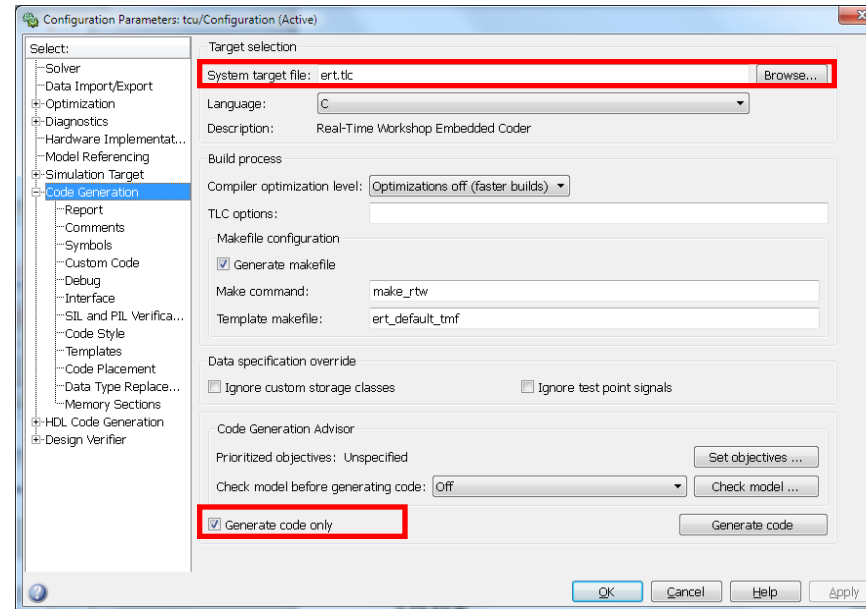
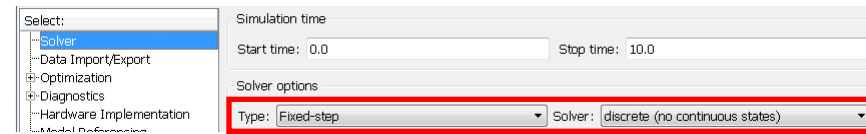


# Simple Software Architecture



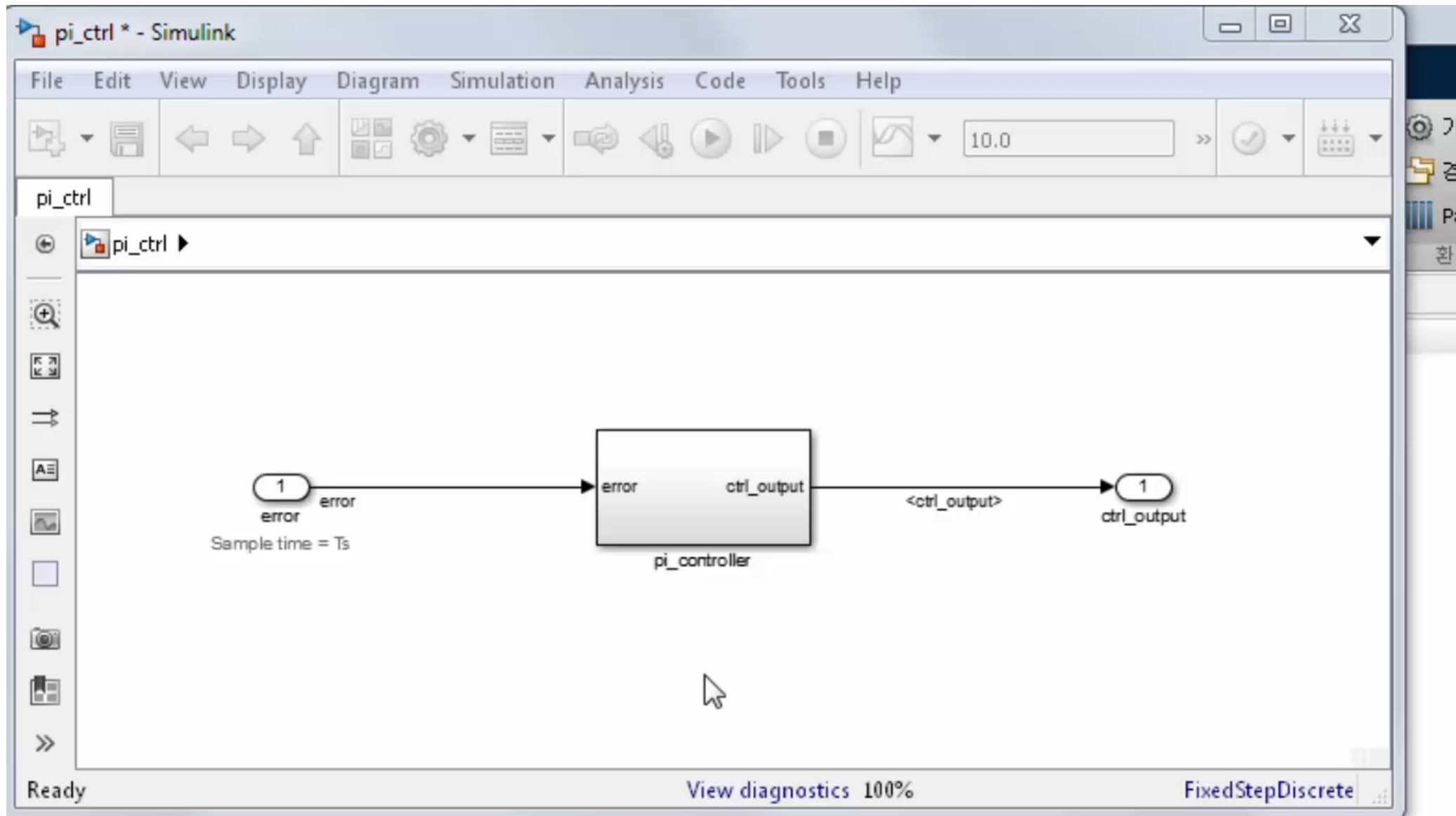
# Basic Code Generation Workflow for Embedded Target

- Select solver to be fixed step with discrete
- Select ert.tlc target
- Select generate code only
- Select HTML Report for easy review
- Generate code for model
- Review code in browser





# Demo: Code Generation Workflow



# Code Generation Reports

- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report

Code Generation Report

Find:  Match Case

**Contents**

- Summary
- [Subsystem Report](#)
- [Code Interface Report](#)
- [Traceability Report](#)
- [Static Code Metrics Report](#)
- [Code Replacements Report](#)

**Generated Code**

- [ - ] Main file
  - [ert\\_main.c](#)
- [ - ] Model files
  - [iir\\_ert.c](#)
  - [iir\\_ert.h](#)
  - [iir\\_ert\\_private.h](#)
  - [iir\\_ert\\_types.h](#)
- [ - ] Utility files
  - [rtwtypes.h](#)

**Traceable Simulink Blocks / Stateflow Objects / MATLAB Functions**

Root system: [iir\\_ert](#)

Object Name	Code Location
<a href="#">&lt;Root&gt;/u</a>	<a href="#">iir_ert.c:34</a>

Subsystem: [iir\\_ert/DocBlock](#)

No traceable objects in this Subsystem.

Subsystem: [iir\\_ert/IIR\\_filter](#)

Object Name	Code Location
<a href="#">&lt;S2&gt;/Sum</a>	<a href="#">iir_ert.c:31</a>
<a href="#">&lt;S2&gt;/Unit Delay</a>	<a href="#">iir_ert.c:20, 35, 39</a> <a href="#">iir_ert.h:35</a>
<a href="#">&lt;S2&gt;/a1</a>	<a href="#">iir_ert.c:32</a>
<a href="#">&lt;S2&gt;/b0</a>	<a href="#">iir_ert.c:33</a>

Subsystem: [iir\\_ert/Model Info](#)

No traceable objects in this Subsystem.

OK Help

# Demo: Code Generation Reports

The image shows a Simulink model of a PI controller and its corresponding Code Generation Report. The Simulink diagram on the left includes an input block '1' labeled 'error' with 'Sample time = Ts'. The signal passes through a summing junction. One path goes through a gain block 'Kp' labeled 'proportional\_gain' to produce output 'yP'. The other path goes through a delay block 'Ts' labeled 'sample\_time', then a gain block 'Ki' labeled 'integral\_gain', and finally to the same summing junction. The output of the summing junction is the controller's output.

The Code Generation Report window on the right is titled 'Code Generation Report for 'pi\_ctrl''. It contains the following sections:

- Contents:** Summary (highlighted), Subsystem Report, Code Interface Report, Traceability Report, Static Code Metrics Report, Code Replacements Report.
- Generated Code:**
  - Main file:** [ert\\_main.c](#)
  - Model files:** [pi\\_ctrl.c](#), [pi\\_ctrl.h](#), [pi\\_ctrl\\_private.h](#), [pi\\_ctrl\\_types.h](#)
  - Utility files:** [rtwtypes.h](#)
- Summary:** Code generation for model "pi\_ctrl".

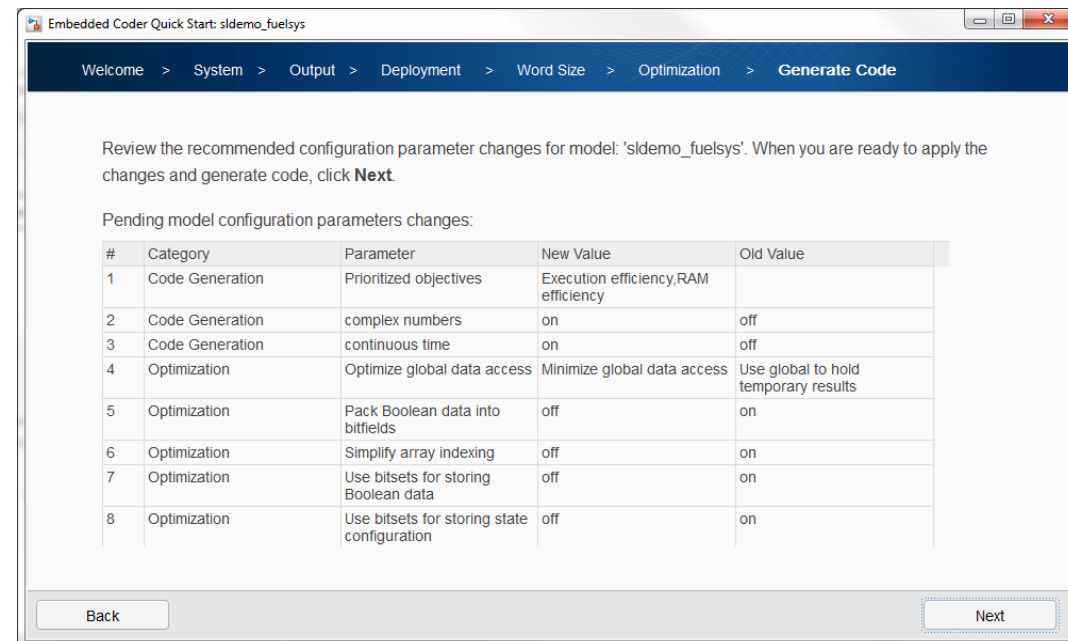
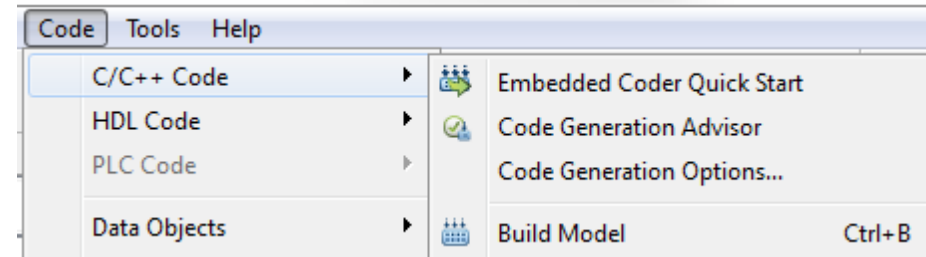
Model version	1.10
Simulink Coder version	8.9 (R2015b) 13-Aug-2015
C source code generated on	Mon Feb 01 13:36:49 2016
C source code generated at	C:\Applications\Seminars\PCG_Seminar\work\pi_ctrl_ert_rtw\

Configuration settings at the time of code generation: [click to open](#)  
Code generation objectives: **Unspecified**  
Validation result: Not run

# Embedded Coder Quick Start

## Easily configure Simulink Model to generate production code

- Ask questions about code generation goals
- Auto configure and validate model against the selections
- Show recommended configuration changes
- Apply configuration changes and generate code



# Demo: Embedded Coder Quick Start

The screenshot displays the Simulink environment for a model named 'pi\_ctrl'. The main workspace shows a block diagram with the following components and connections:

- An input port on the left labeled 'error' with a value of '1' and a sample time of  $T_s$ .
- A central block labeled 'pi\_controller' with an input port 'error' and an output port 'ctrl\_output'.
- An output port on the right labeled 'ctrl\_output' with a value of '1'.
- A signal line connects the 'error' input to the 'pi\_controller' block.
- A signal line connects the 'ctrl\_output' output of the 'pi\_controller' block to the 'ctrl\_output' output port, labeled with the signal name '<ctrl\_output>'.

The Simulink interface includes a menu bar (File, Edit, View, Display, Diagram, Simulation, Analysis, Code, Tools, Help), a toolbar with various simulation and editing tools, and a status bar at the bottom showing 'Ready', '100%', and 'FixedStepDiscrete'.

# Optimization Considerations

Configuration Parameters: pj\_ctrl/Configuration (Active)

Select:

- Solver
- Data Import/Export
- Optimization
  - Signals and Parameters
  - Stateflow
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface
  - Verification
  - Code Style
  - Templates
  - Code Placement
  - Data Type Replacement
  - Memory Sections
  - HDL Code Generation

Simulation and code generation

- Block reduction
- Implement logic signals as Boolean data
- Conditional input branch execution

Use division for fixed-point net slope corrections:  On  Off

Use floating-point multiplication to handle net slope corrections:  On  Off

Default for underspecified data type:  GRT  ERT

Code generation

- Optimize using the specified minimum number of iterations

Data initialization

- Remove root level I/O zero initialization
- Remove internal data zero initialization

Integer and fixed-point

- Remove code from floating-point to integer conversions that wraps out-of-range values
- Remove code from floating-point to integer conversions with saturation that maps NaN to zero
- Remove code that protects against overflow

Accelerating simulations

Compiler optimization level:

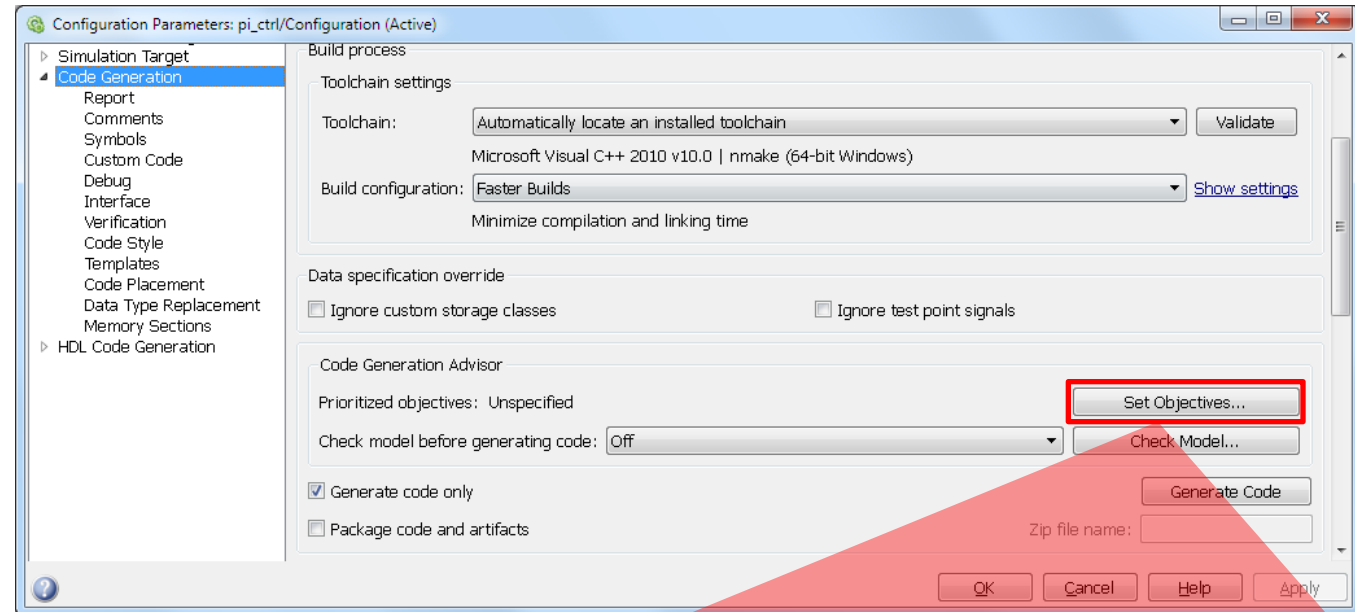
- Verbose accelerator builds

**Mapping Application Requirements to the Optimization Pane: General Tab**

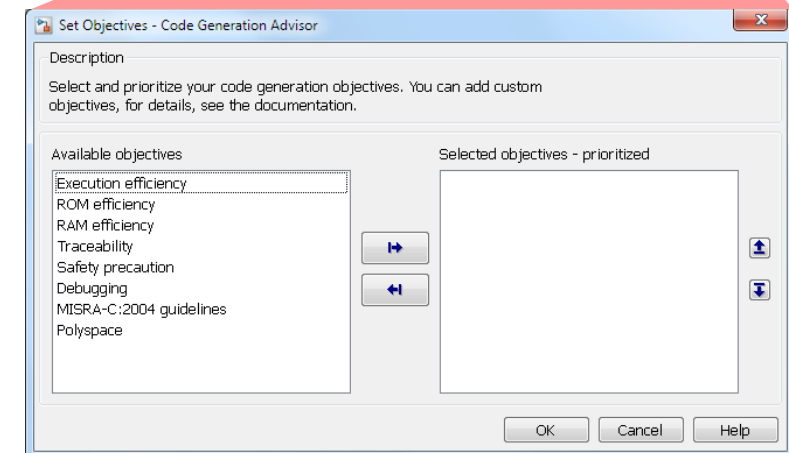
Configuration Parameter	Settings for Building Code				Factory Default
	Debugging	Traceability	Efficiency	Safety precaution	
<a href="#">Block reduction</a>	Off (GRT) No impact (ERT)	Off	On	Off	On
<a href="#">Implement logic signals as Boolean data (vs. double)</a>	No impact	No impact	On	On	On
<a href="#">Conditional input branch execution</a>	No impact	On	On (execution) No impact (ROM, RAM)	No impact	On
<a href="#">Application lifespan (days)</a>	No impact	No impact	Finite value	inf	inf
<a href="#">Use memset to initialize floats and doubles to 0.0</a>	No impact	No impact	On* (execution, ROM) No impact (RAM)	No impact	On
<a href="#">Use floating-point multiplication to handle net slope corrections</a>	No impact	No impact	On (when target hardware supports efficient multiplication) Off (otherwise)	Off	Off
<a href="#">Remove code from floating-point to integer conversions that wraps out-of-range values</a>	Off	Off	On (execution, ROM) No impact (RAM)	Off (GRT) On (ERT)	Off
<a href="#">Remove code from floating-point to integer conversions with saturation that maps NaN</a>	Off	Off	On		



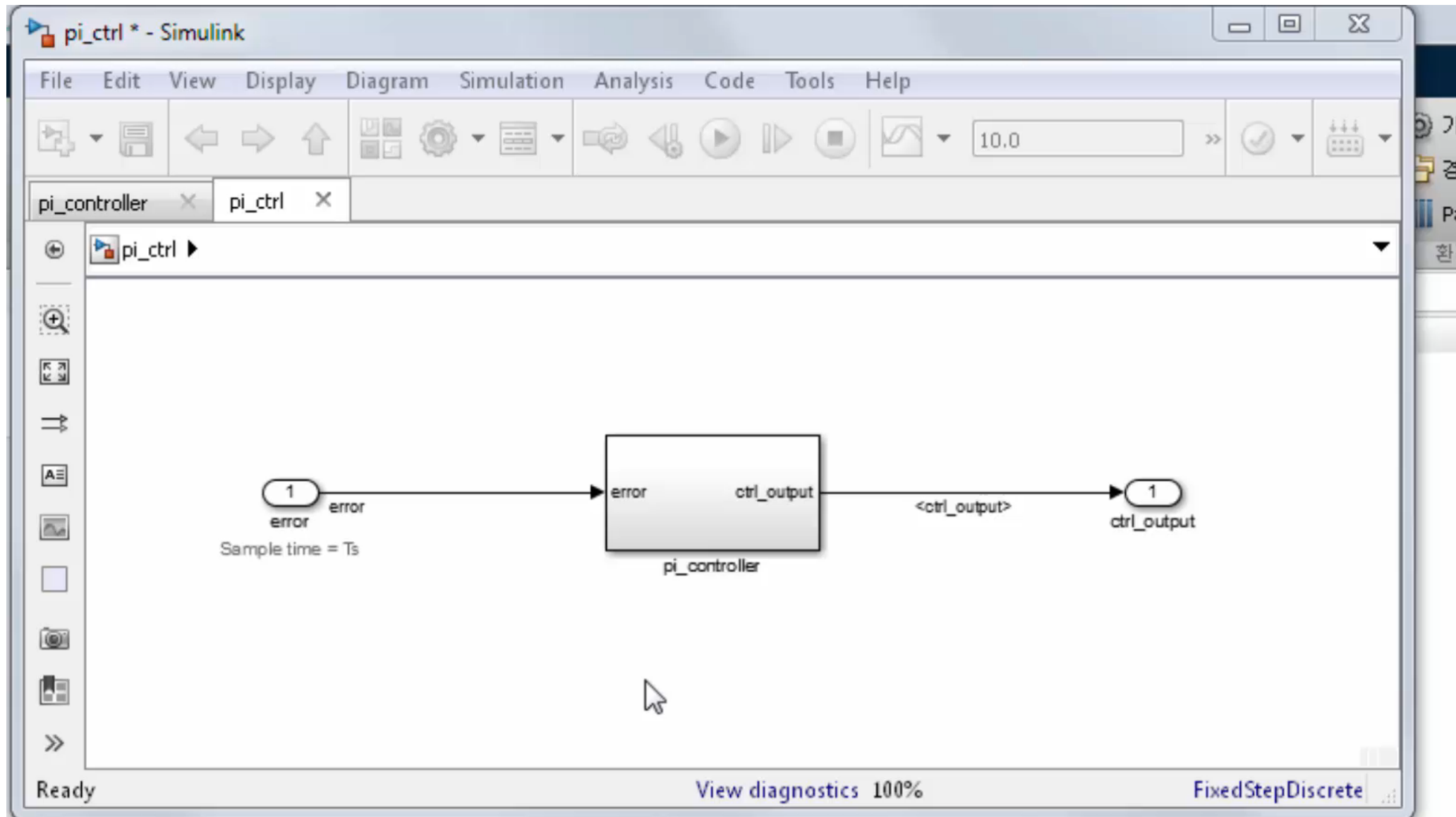
# Code Generation Objectives



- Models have a lot of possible settings
- Code Generation Objectives gives a starting point

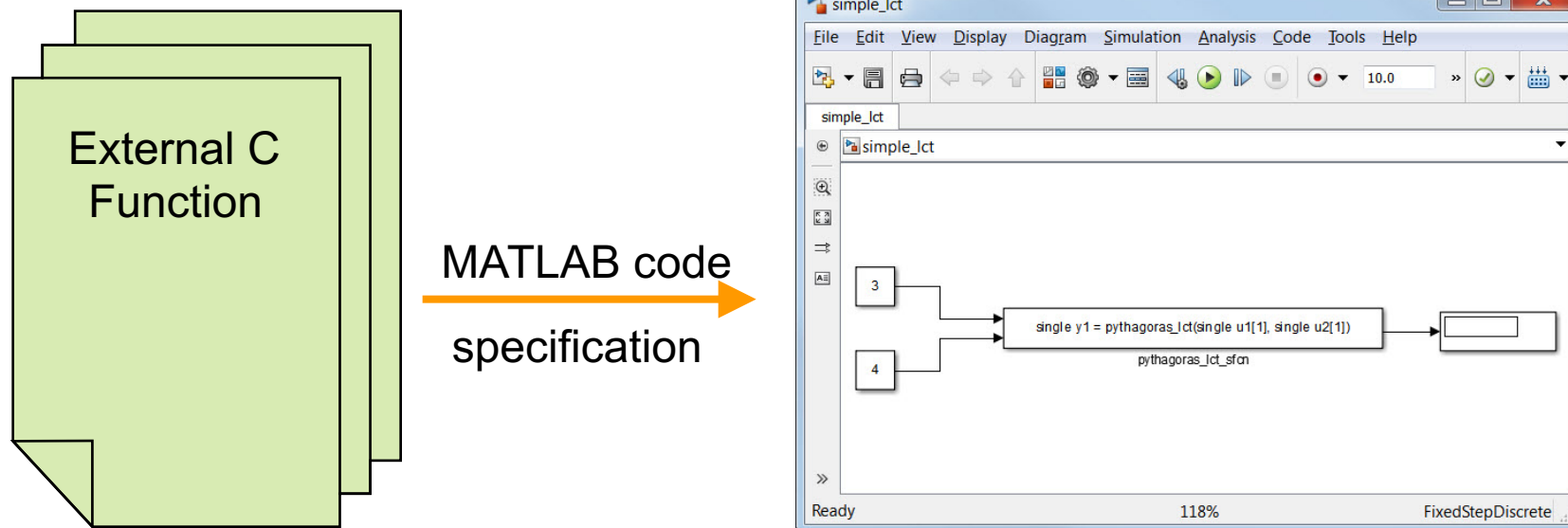


# Demo: Optimizing Generated Code

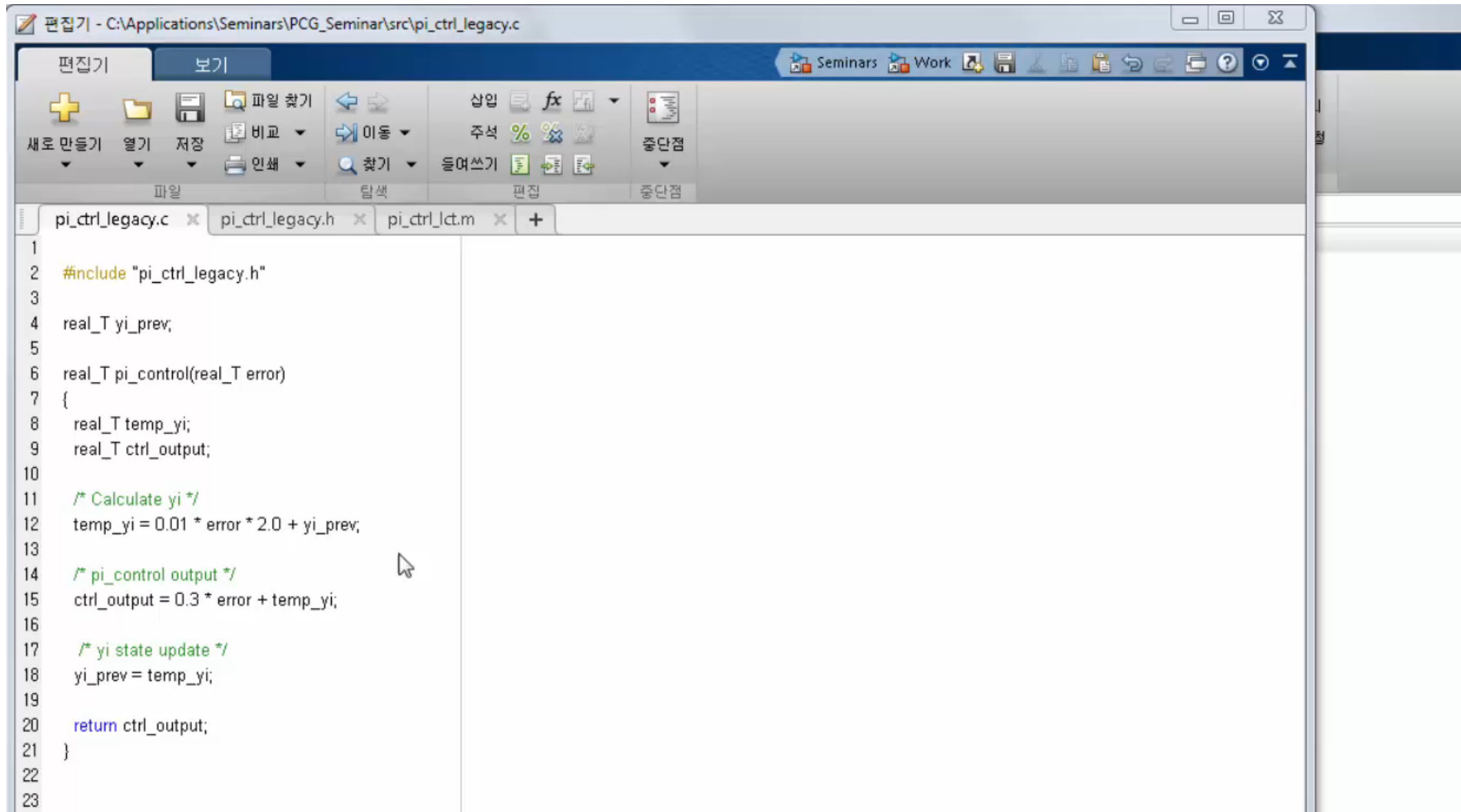


# Legacy Code Tool

- Legacy Code Tool is a utility that generates an S-function automatically from existing C code
- It can also insert an appropriate call to generated code



# Demo: Code Generation with Legacy C Code



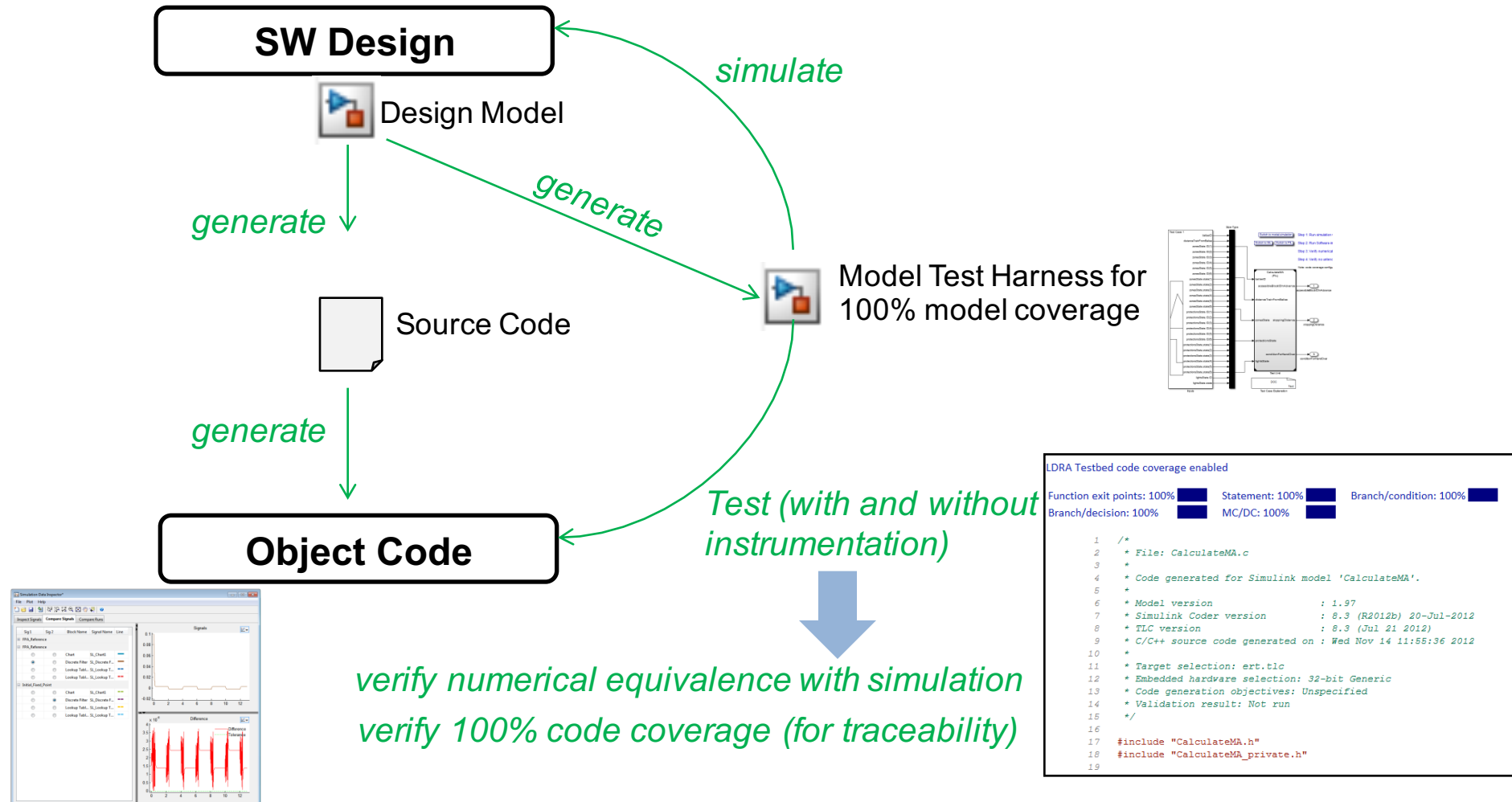
```
편집기 - C:\Applications\Seminars\PCG_Seminar\src\pi_ctrl_legacy.c
편집기 보기 Seminars Work
+ 파일 찾기 상입 fx
새로 만들기 열기 저장 비교 이동 주석 %
인쇄 찾기 들여쓰기 중단점
파일 탐색 편집 중단점
pi_ctrl_legacy.c x pi_ctrl_legacy.h x pi_ctrl_ict.m x +
1
2 #include "pi_ctrl_legacy.h"
3
4 real_T yi_prev;
5
6 real_T pi_control(real_T error)
7 {
8     real_T temp_yi;
9     real_T ctrl_output;
10
11     /* Calculate yi */
12     temp_yi = 0.01 * error * 2.0 + yi_prev;
13
14     /* pi_control output */
15     ctrl_output = 0.3 * error + temp_yi;
16
17     /* yi state update */
18     yi_prev = temp_yi;
19
20     return ctrl_output;
21 }
22
23
```

# Agenda

- Production Code Generation
  - MathWorks' Code Generation Products
  - Embedded Coder
  - Equivalence Test with SIL and PIL
  
- Integration Test
  - What's Simulink Real-Time
  - Automation of Real-Time Testing

# Equivalence Test

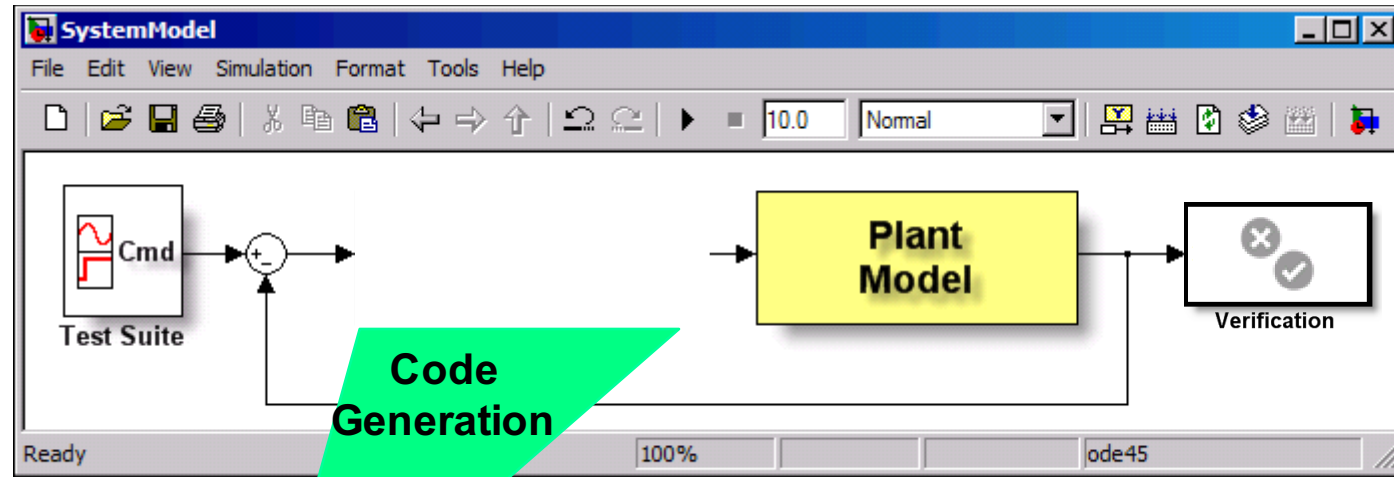
## SIL (Software-In-the-Loop) and PIL (Processor-In-the-Loop)





# Software-in-the-Loop (SIL) Testing:

*Verify Production Controller with Software-in-the-loop*



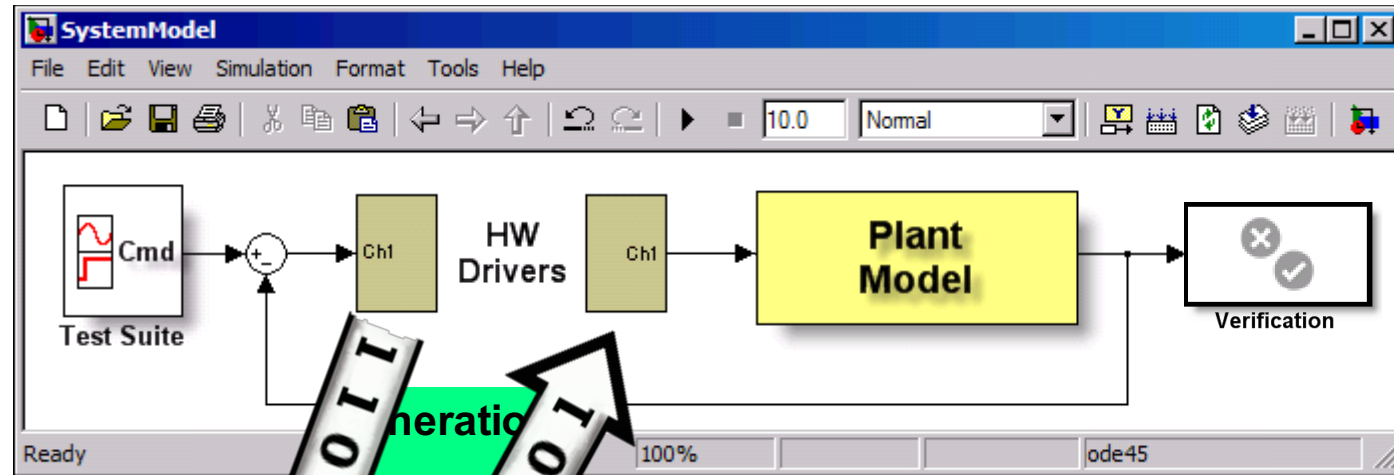
## Execution

- Host/Host
- Nonreal-time

**Compiled C Code  
S-Function  
(Windows DLL)**

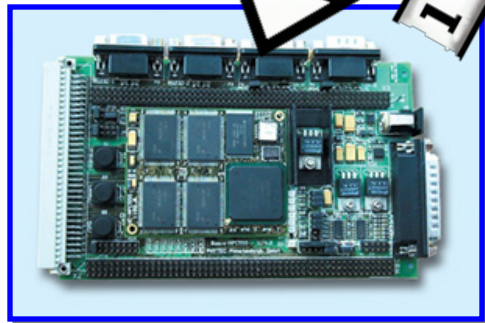
# Processor-in-the-Loop Testing:

*Verify Production Controller with Processor-in-the-loop*



## Execution

- Host/Target
- Nonreal-time



*Production Processor*

# Processor-in-the-Loop (PIL) API

## Problem

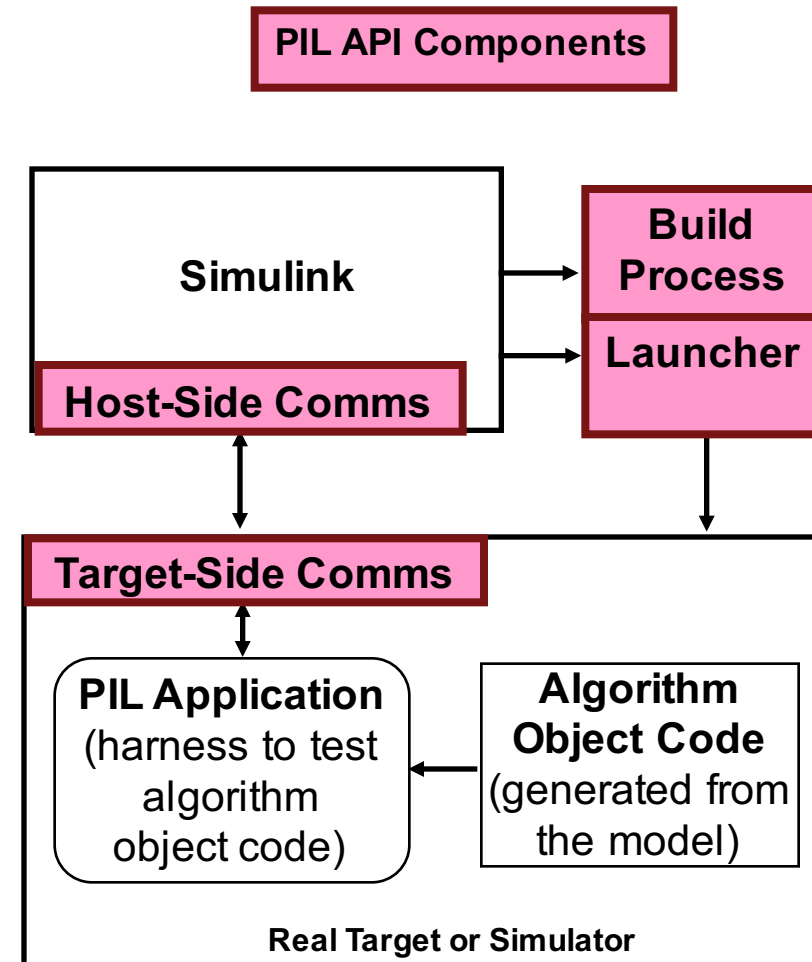
- Embedded IDE Link does not support PIL for an arbitrary combination of
  - Processor
  - Compiler
  - Debugger or download utility
  - Communications channel

## Solution

- Provide an API that allows integration of third-party or customer tools for
  - Building the PIL application
  - Downloading and running the application
  - Communicating with the application

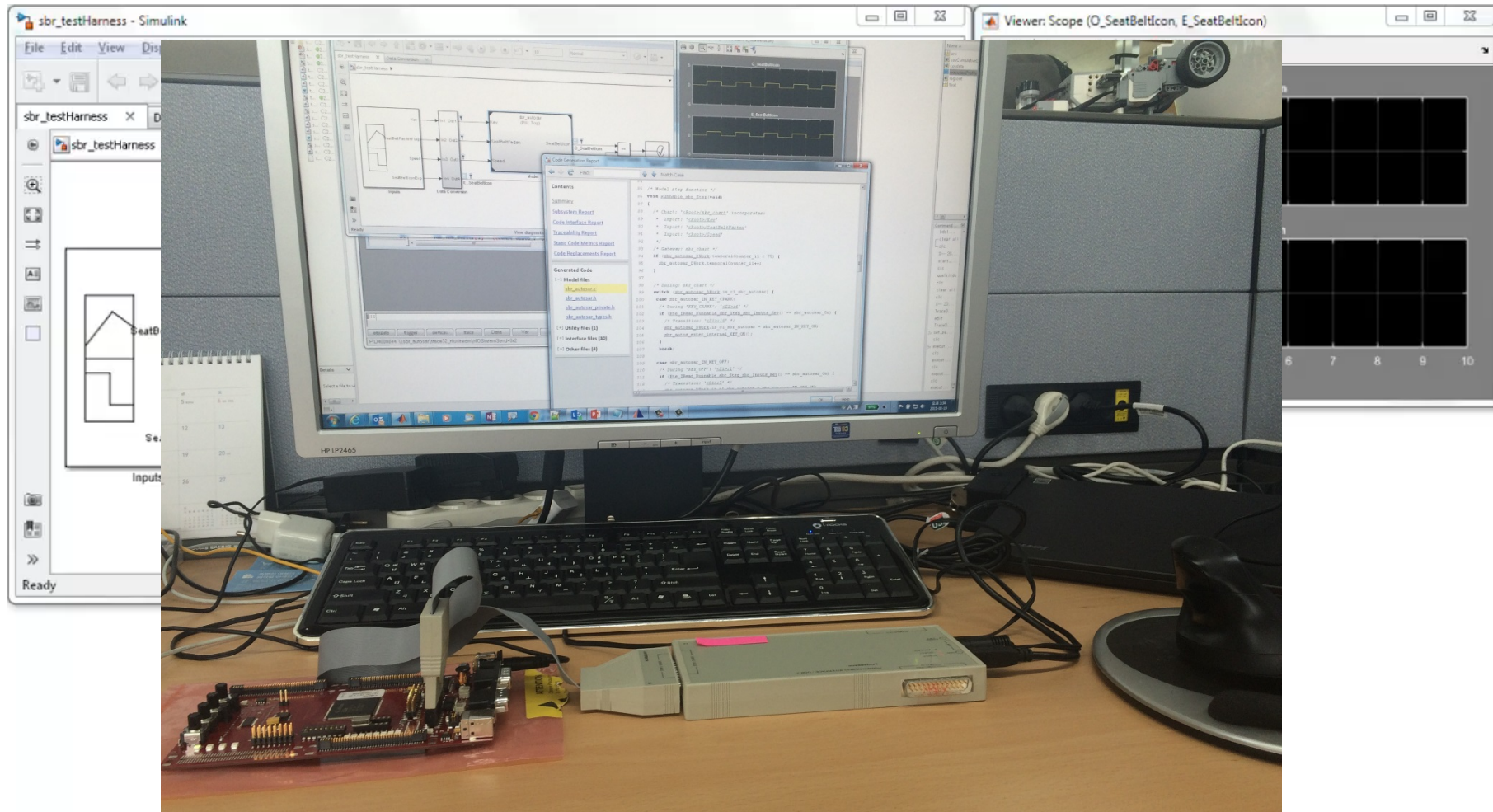
## Benefit

- The power of PIL verification is easily adaptable for any target environment
- A fully documented API is stable across MathWorks releases



# PIL Testing Example

## *Infineon Tricore with Trace32 Debugger*



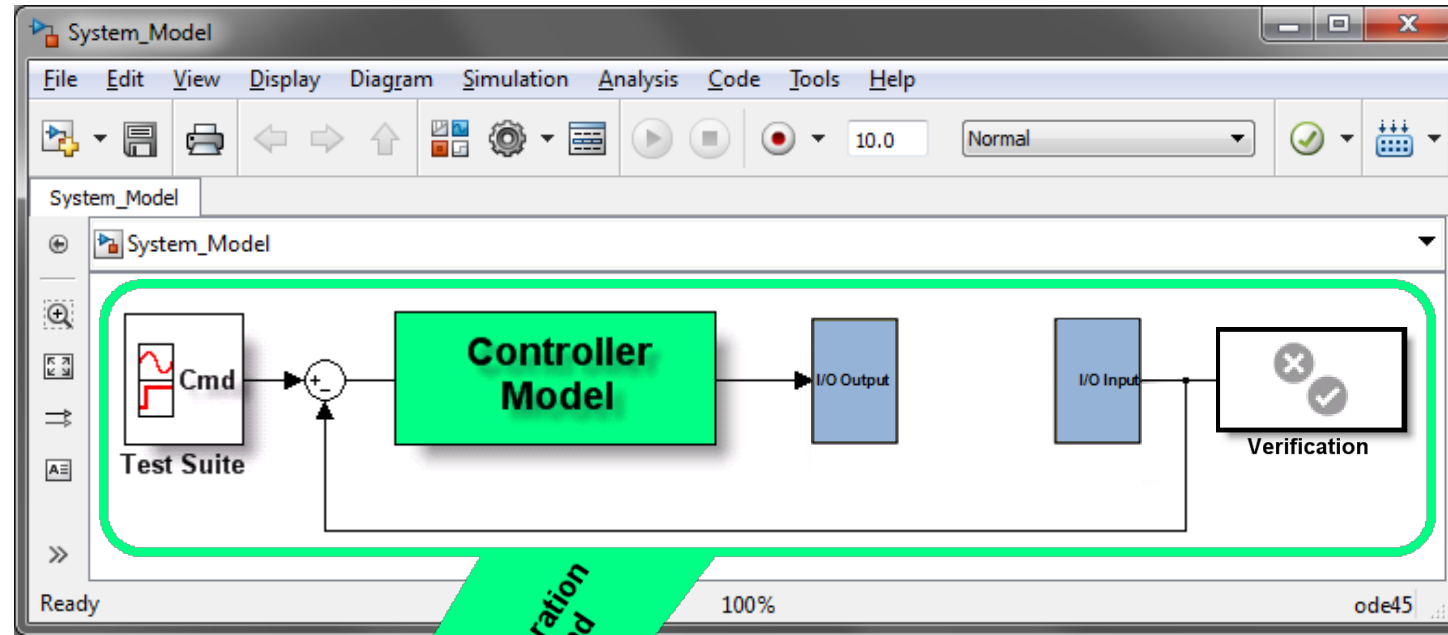
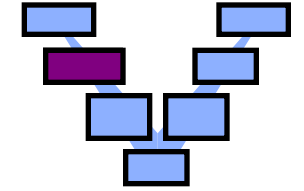
# Key Benefits of SIL and PIL

- Reuse test vectors for simulation, SIL and PIL
  - Verify correct execution behaviour of compiled code (including on production hardware)
  - Collect metrics for the generated code
    - Code coverage
    - Execution profiling
    - Stack profiling
- Evaluate hardware specific optimizations
- Generate artifacts for IEC-61508, IEC-62304, ISO-26262, EN-50128, and DO-178 certification
- Early verification and fixing of defects reduces cost

# Agenda

- Production Code Generation
  - MathWorks' Code Generation Products
  - Embedded Coder
  - Equivalence Test with SIL and PIL
  
- Integration Test
  - What's Simulink Real-Time
  - Automation of Real-Time Testing

# Real-Time Simulation and Testing Tasks: Rapid Controls Prototyping



**1 - Click  
Code Generation  
and Download**



**Target Computer Hardware**



**Wiring and  
Signal Conditioning**

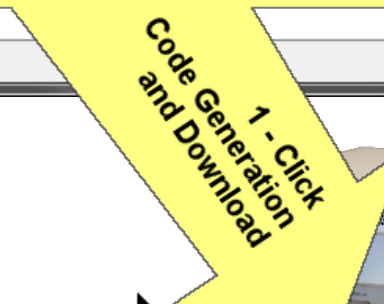
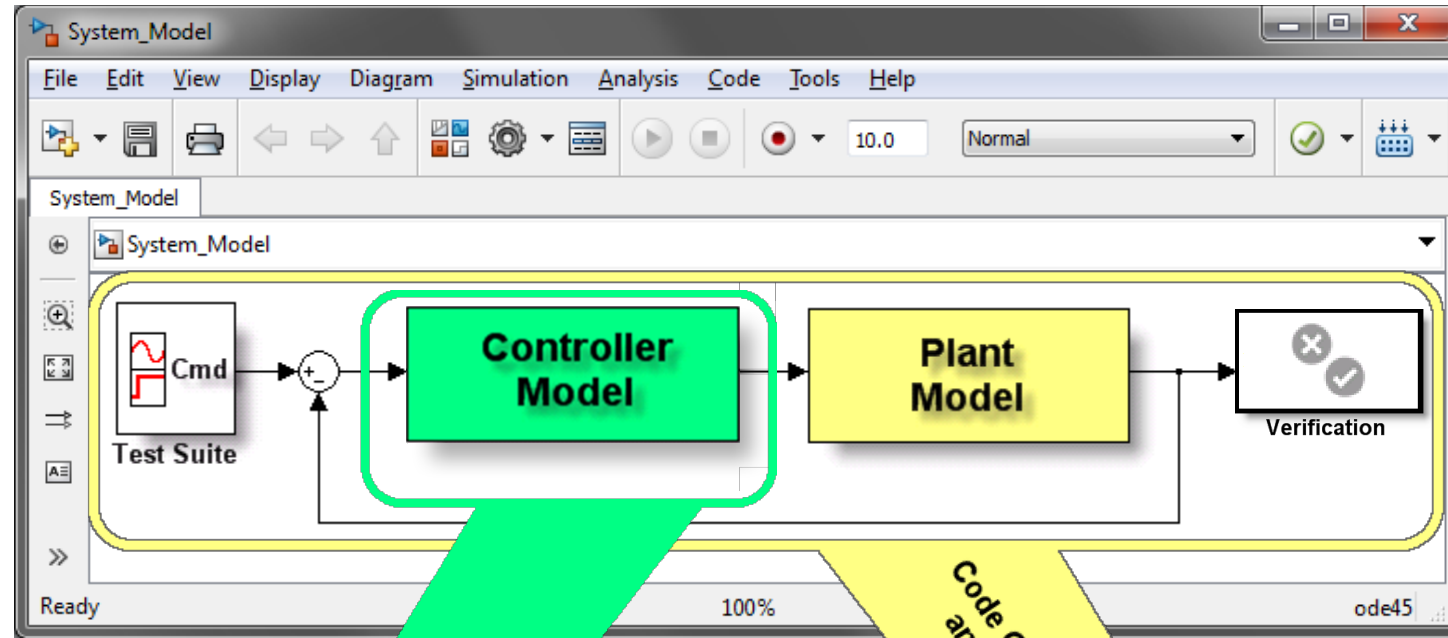
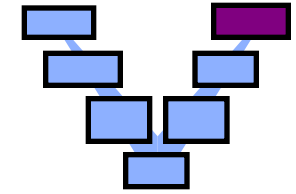


**Physical Plant Hardware**



# Real-Time Simulation and Testing Tasks:

## Hardware-in-the-loop (HIL) Simulation



Embedded Controller Hardware



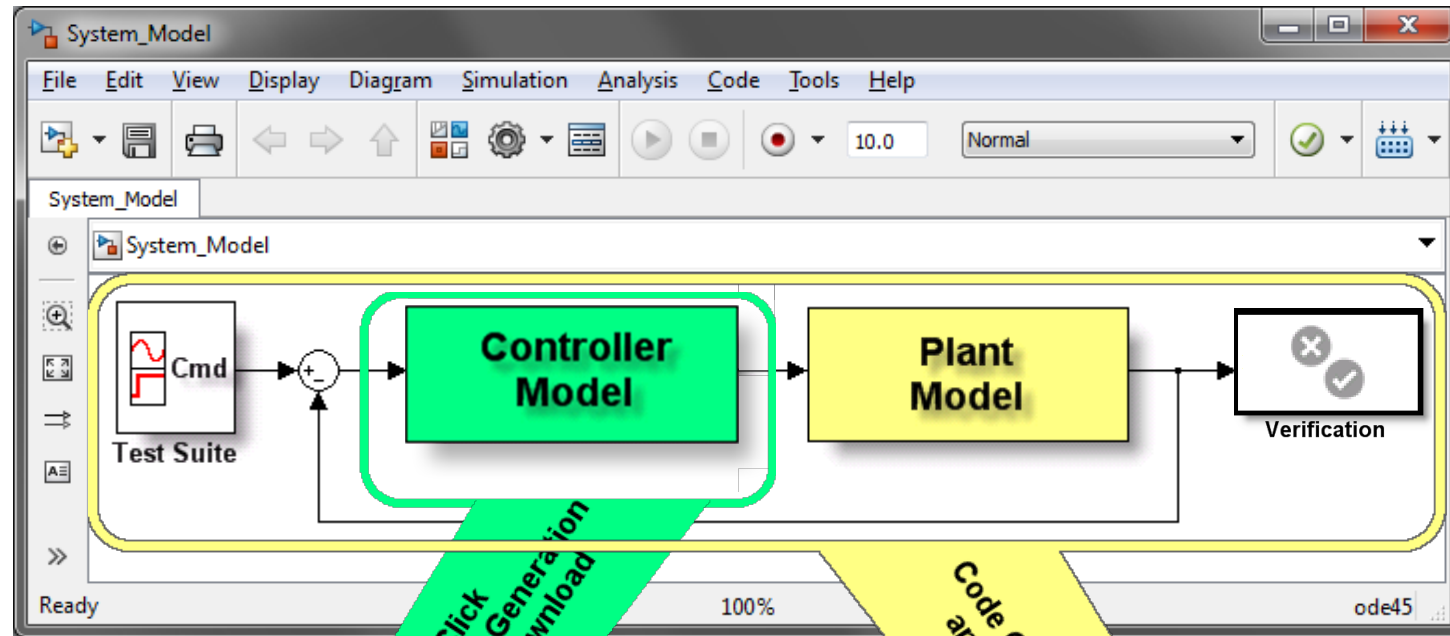
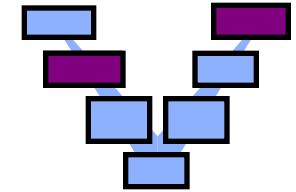
Wiring and Signal Conditioning



Target Computer Hardware



# Today's Configuration



1 - Click  
Code Generation  
and Download

1 - Click  
Code Generation  
and Download



Target Computer Hardware



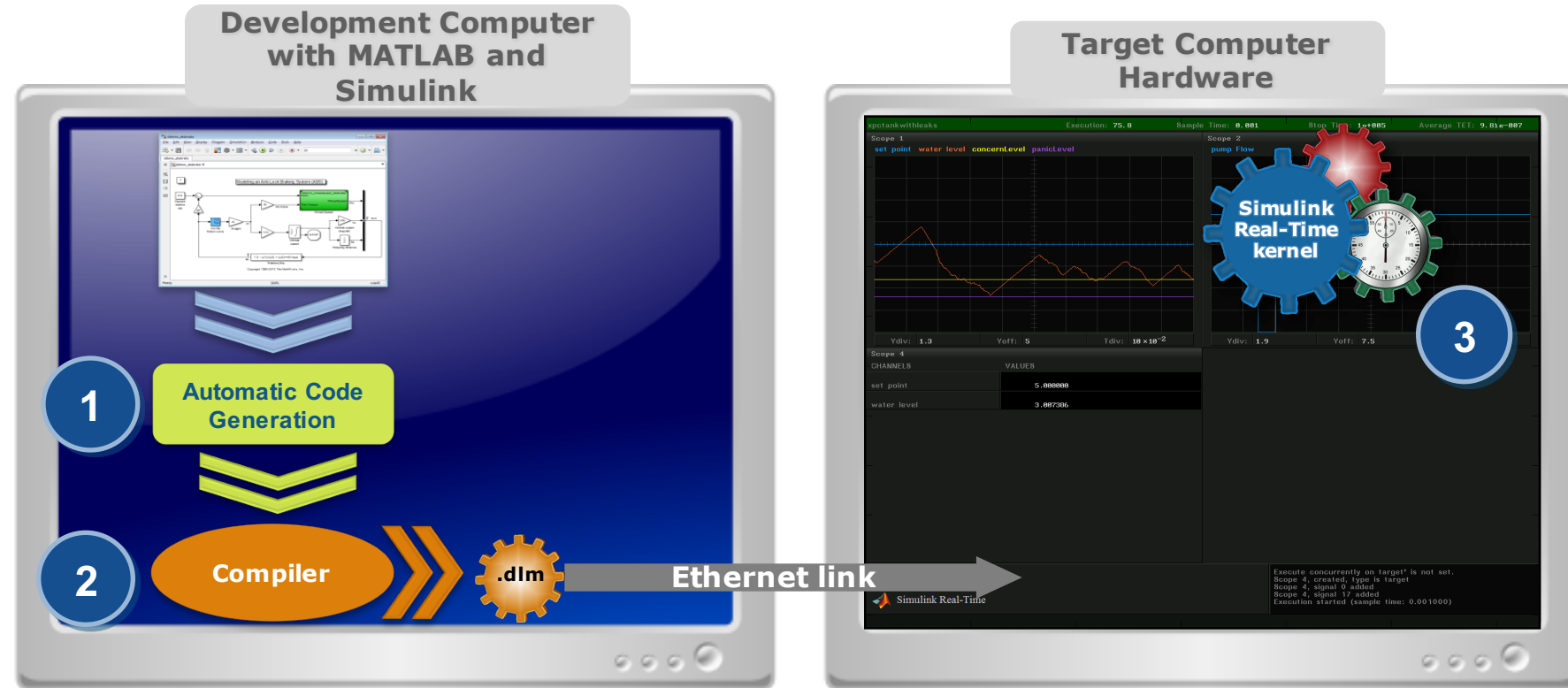
CAN



Target Computer Hardware

# What is Simulink Real-Time?

*From desktop simulation to real time*

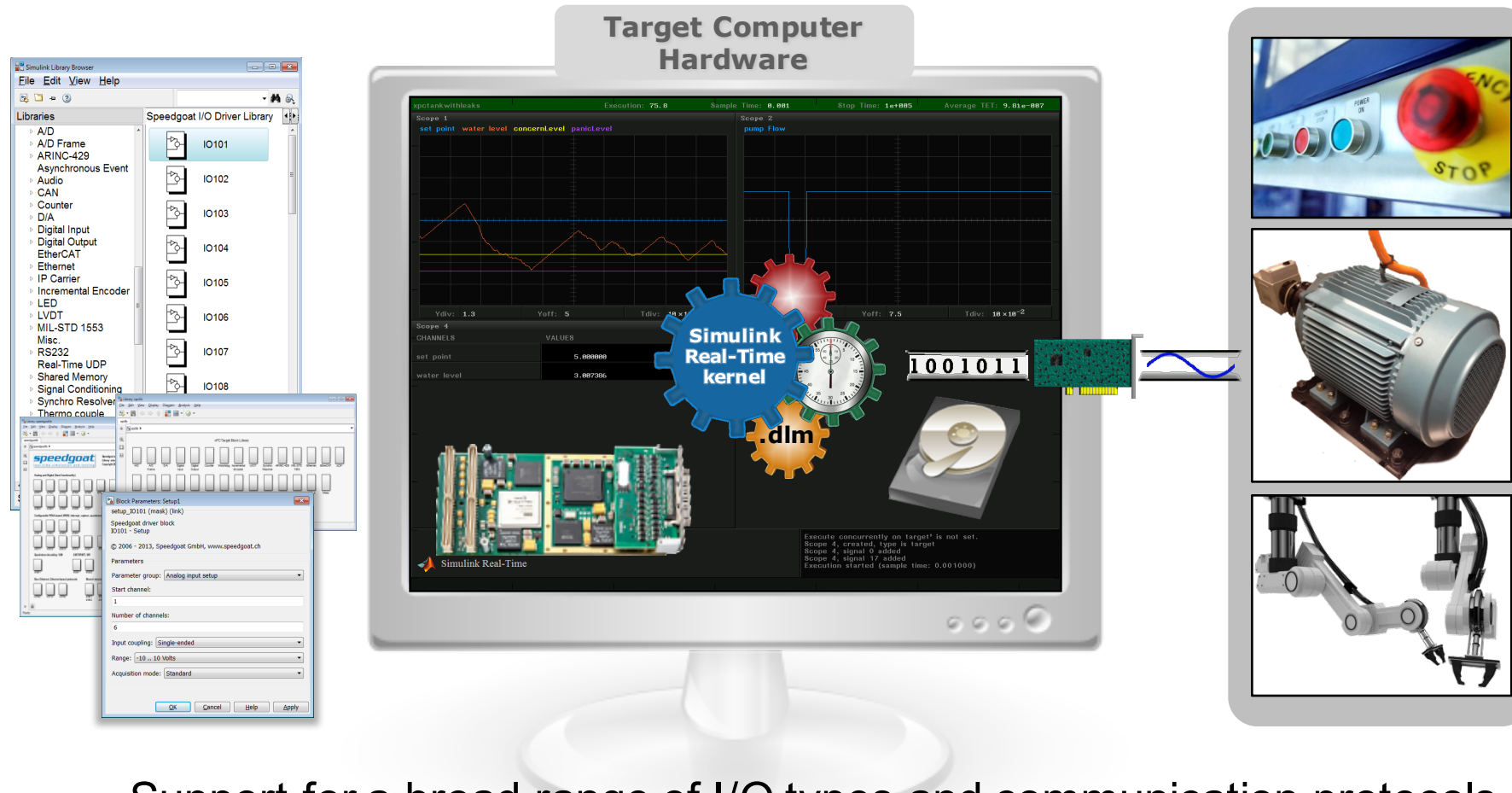


Creation of real-time applications from Simulink models and loading them onto dedicated target computer hardware in 3 automated steps:

- 1 Code Generation
- 2 Compile and Link
- 3 Download and Ready to Run

# What is Simulink Real-Time?

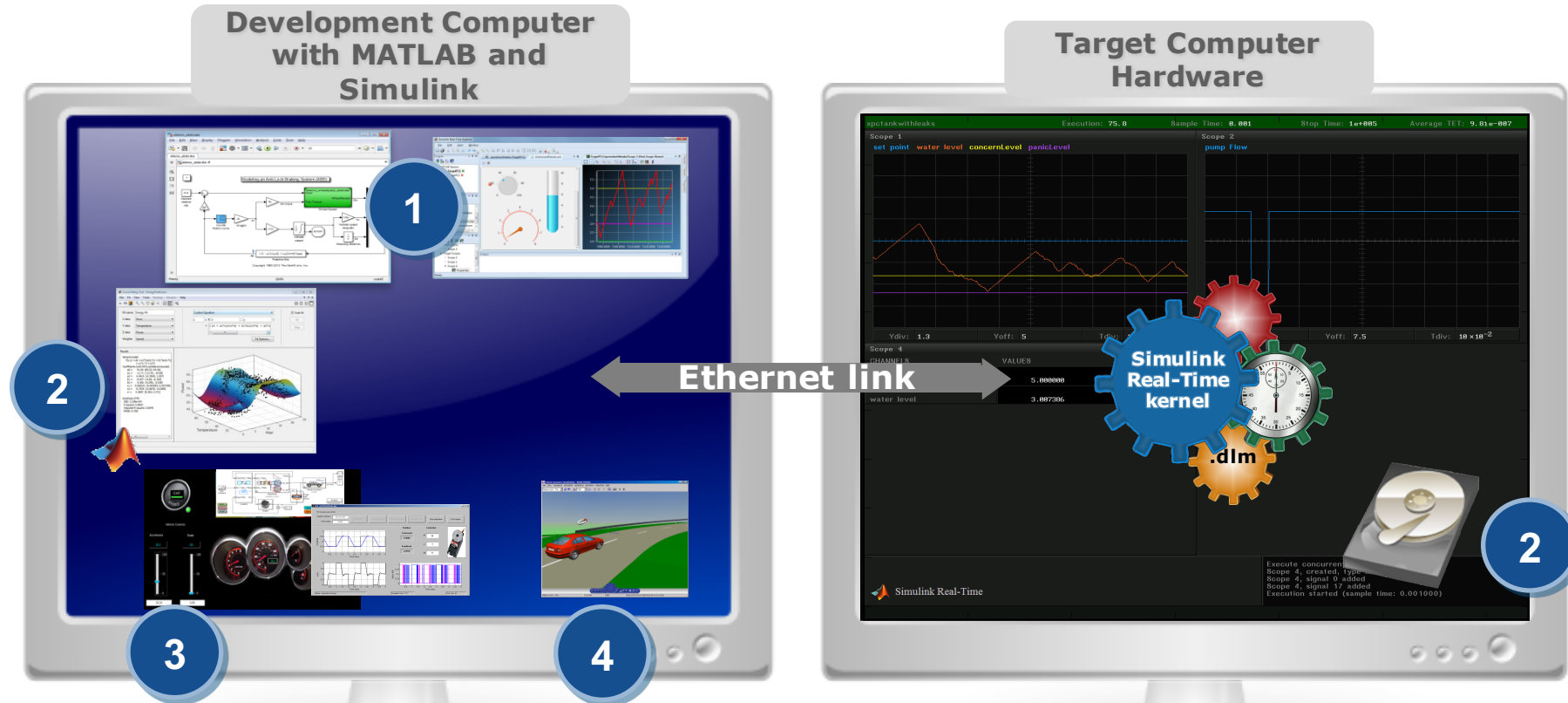
*Connect to your physical system*



- Support for a broad range of I/O types and communication protocols
- Easy drag and drop and configuration within a Simulink model

# What is Simulink Real-Time?

*Extendable, integrated, and interactive*



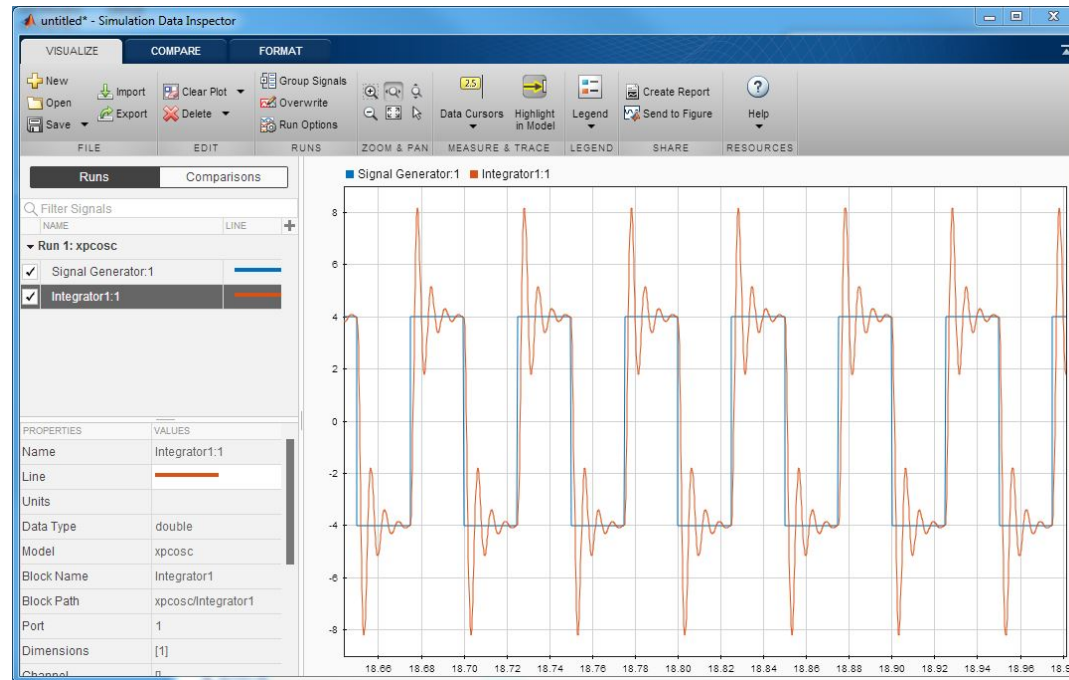
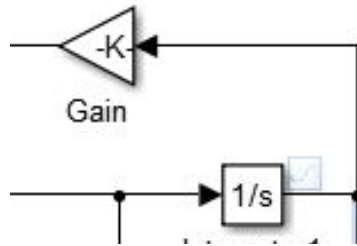
**1** Live parameter tuning, signal monitoring, and execution control

**2** Data logging for offline analysis in MATLAB

**3** UI/HMI connectivity

**4** Extensibility with other software tools (e.g. virtual reality)

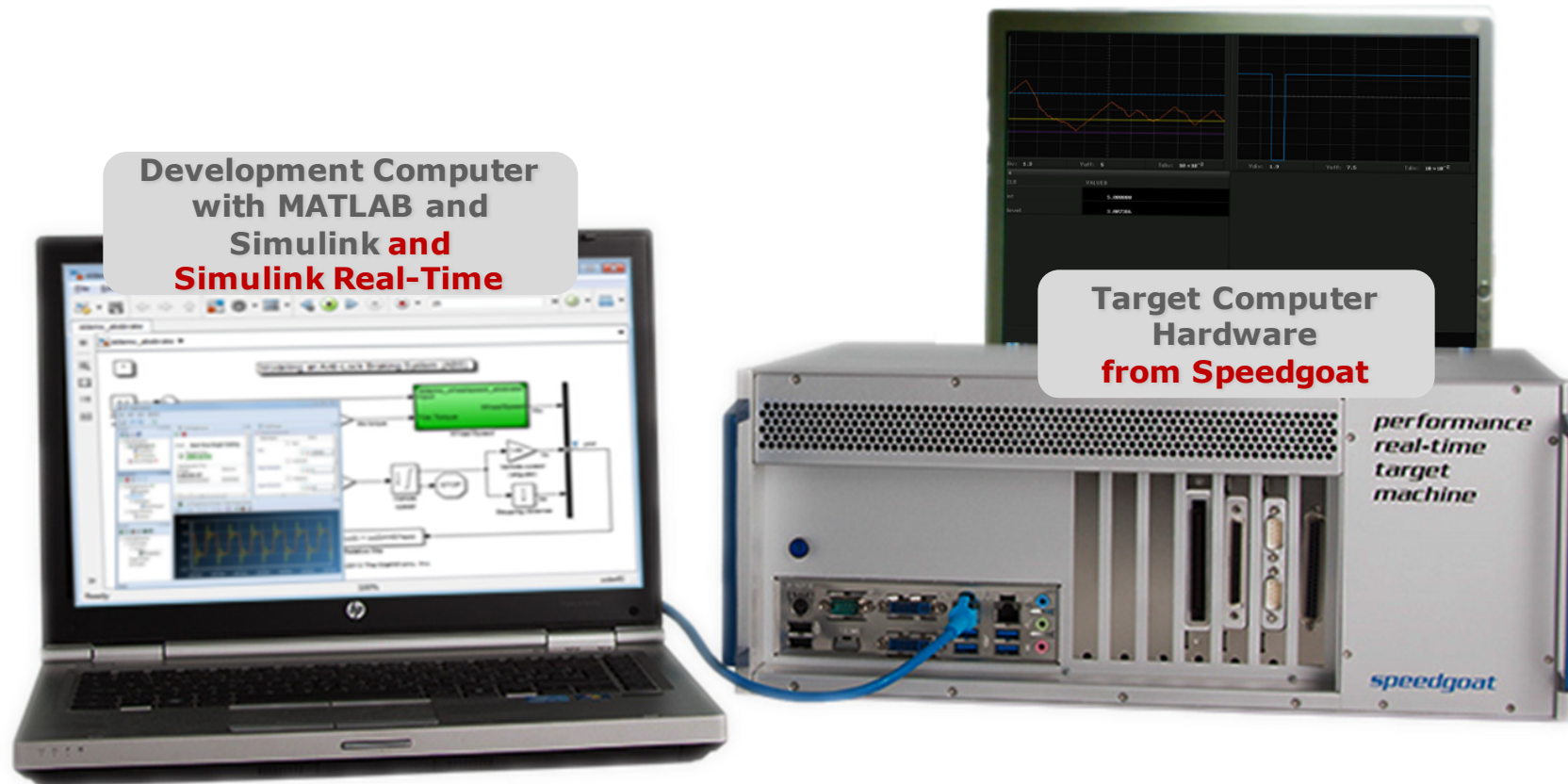
# Streaming to the Simulation Data Inspector



1. Select the signals to stream
2. Connect to the running target computer
3. Visualize in the Simulation Data Inspector

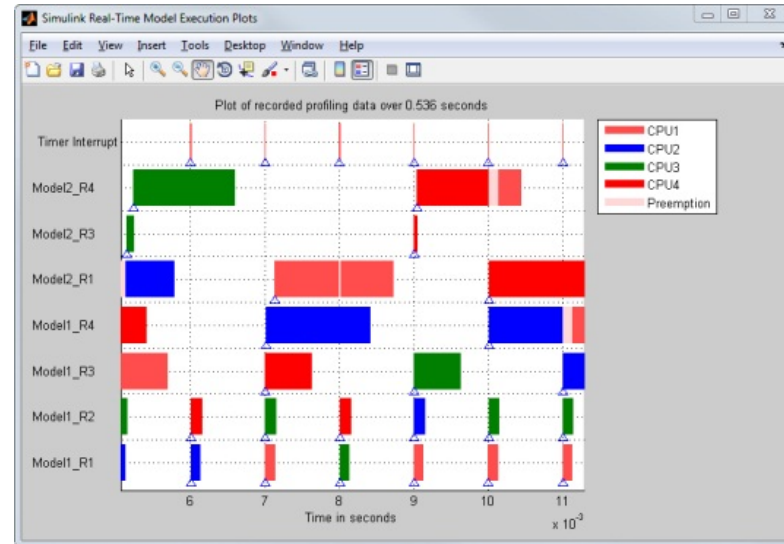
# What Hardware is used with Simulink Real-Time?

*Development computer + target computer*





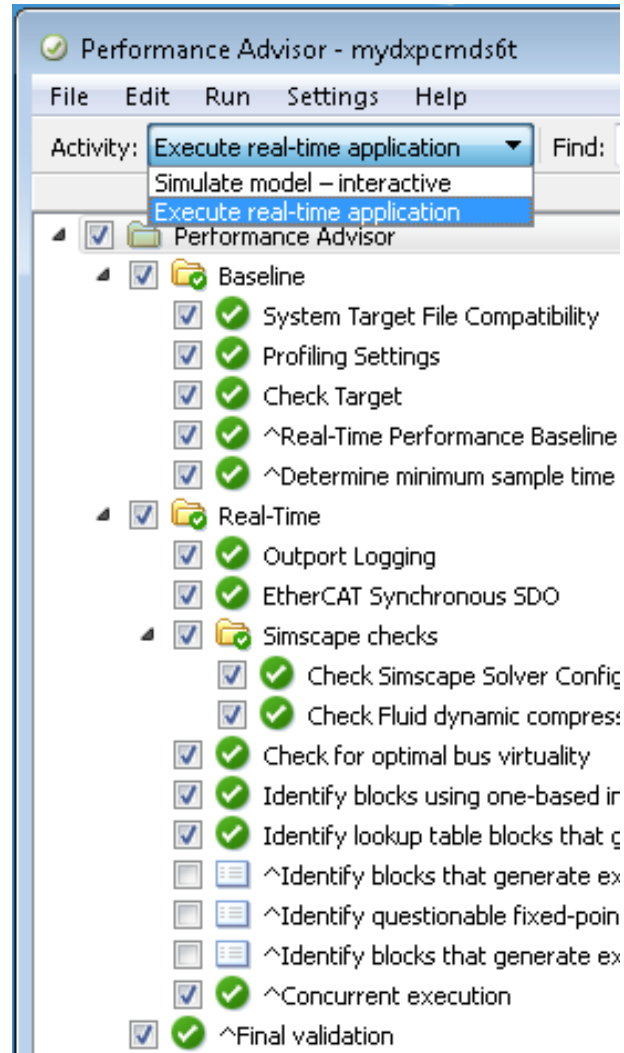
# Current Performance Level



real-time  
multi-core  
scheduler

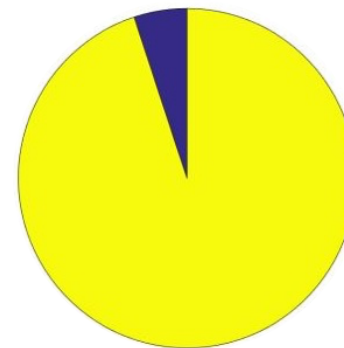
- 25 microsecond minimum sample time
- < 1 microsecond sample time with FPGA's
- High performance quad core Intel processors
- Expandable, low latency I/O

# Performance Advisor for Real-Time Execution

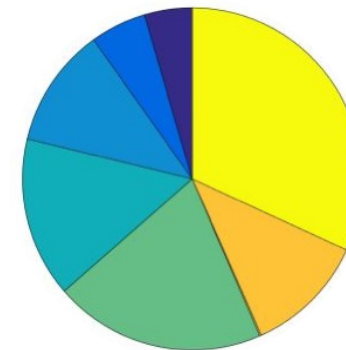


- Encodes best practices for transitioning to real-time
- Adds testing on the target computer.

Before:  
4000 ⚡ Sec



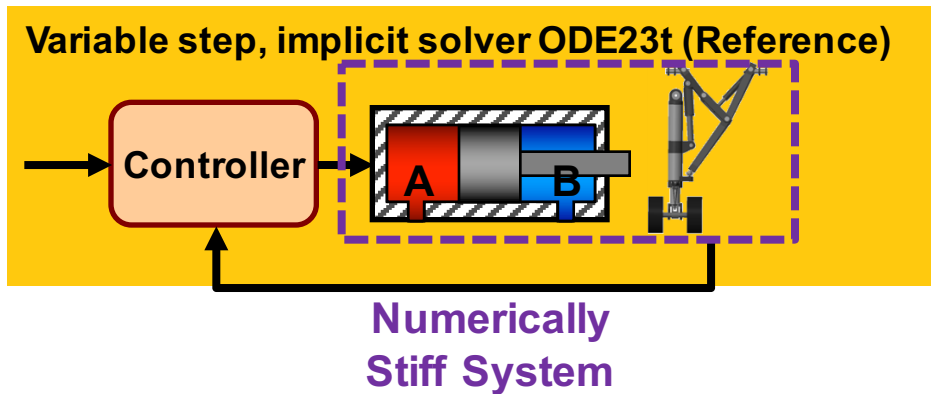
After:  
300 ⚡ Sec





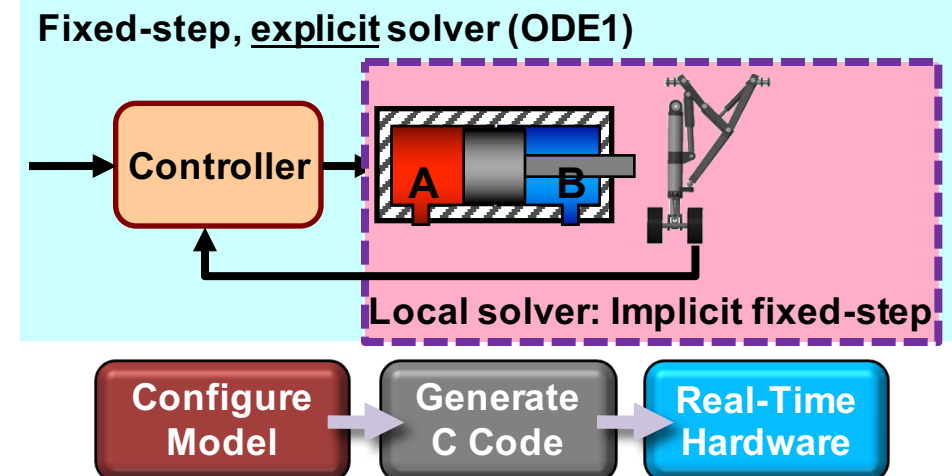
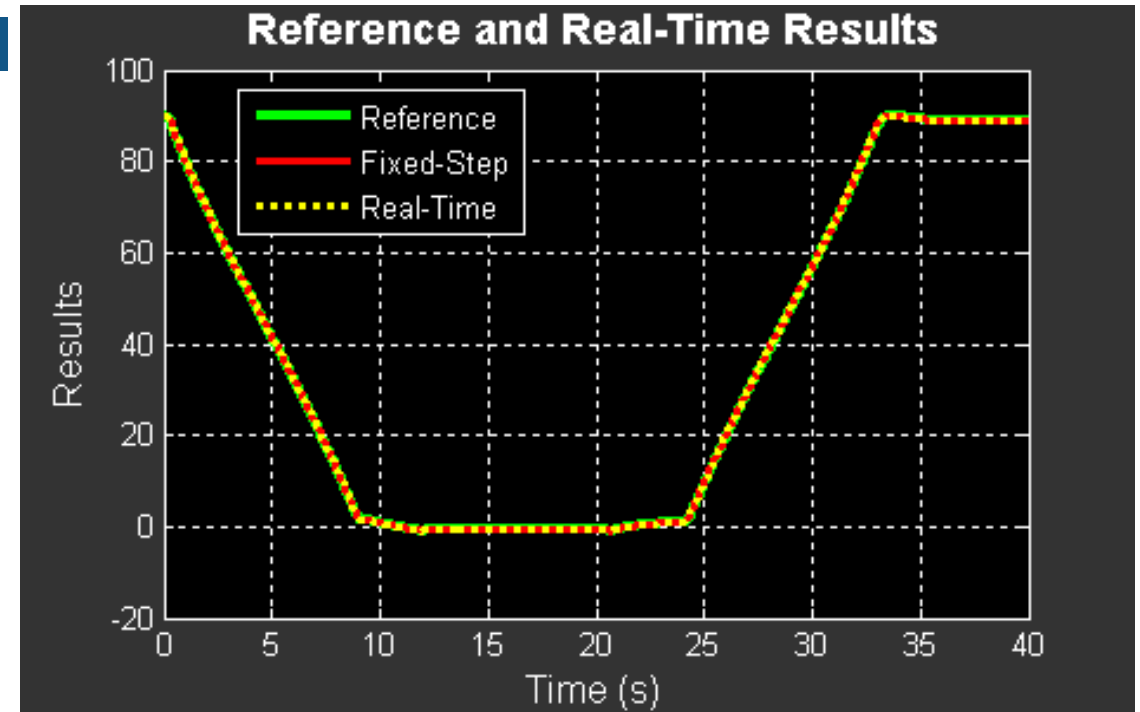
# Configuring Landing Gear Model for HIL Testing

## Model:



**Problem:** Configure solvers to minimize computations so the model can simulate in real time

**Solution:** Use **local solvers** on stiff physical networks and explicit solvers elsewhere



# Agenda

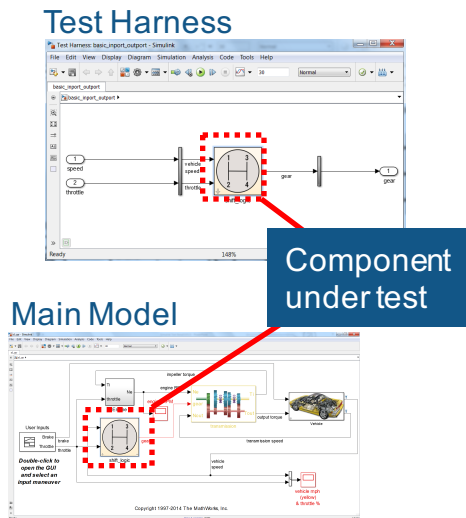
- Production Code Generation
  - MathWorks' Code Generation Products
  - Embedded Coder
  - Equivalence Test with SIL and PIL
  
- Integration Test
  - What's Simulink Real-Time
  - Automation of Real-Time Testing

# Let's remind Simulink Test....

*Tool for authoring, managing, and executing simulation-based tests*

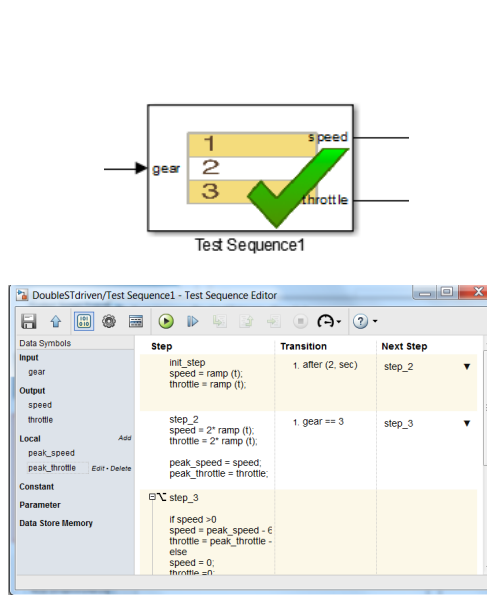
## Test Harness

*Embedded canvas for isolation testing of components*



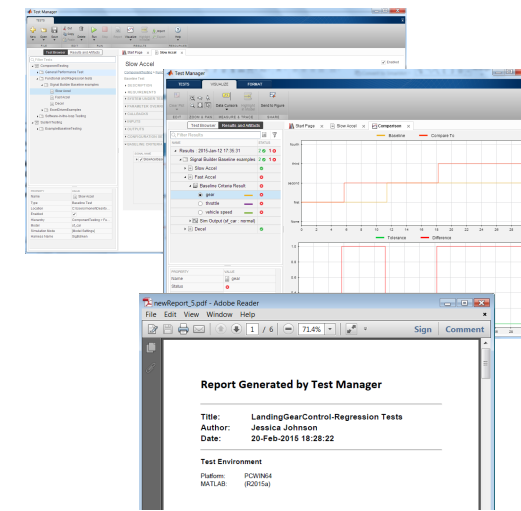
## Test Sequence

*Easily express logic-based tests*



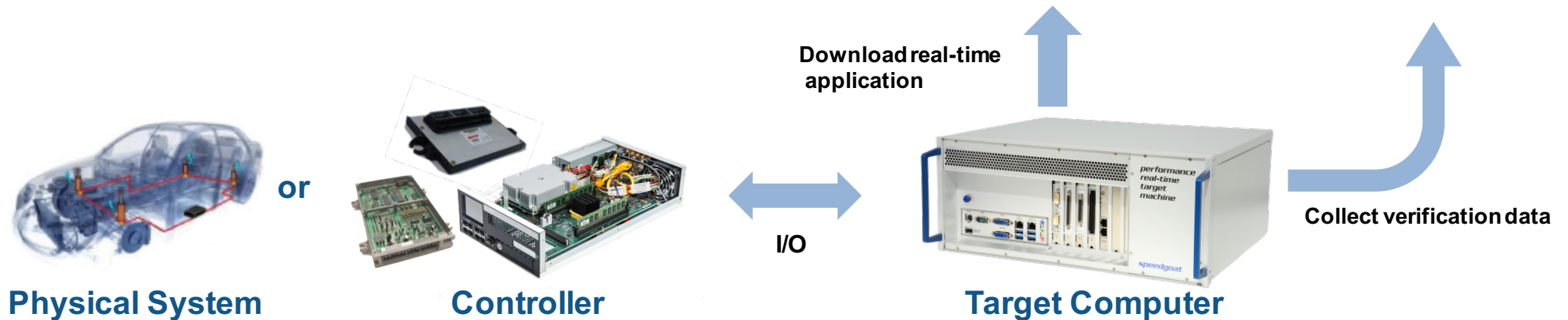
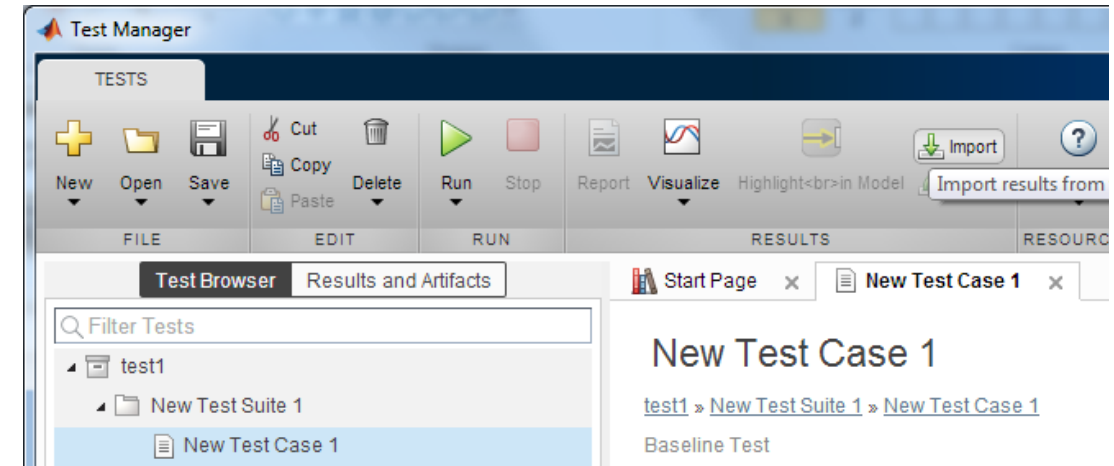
## Test Manager

*Systematic authoring and management of test cases*



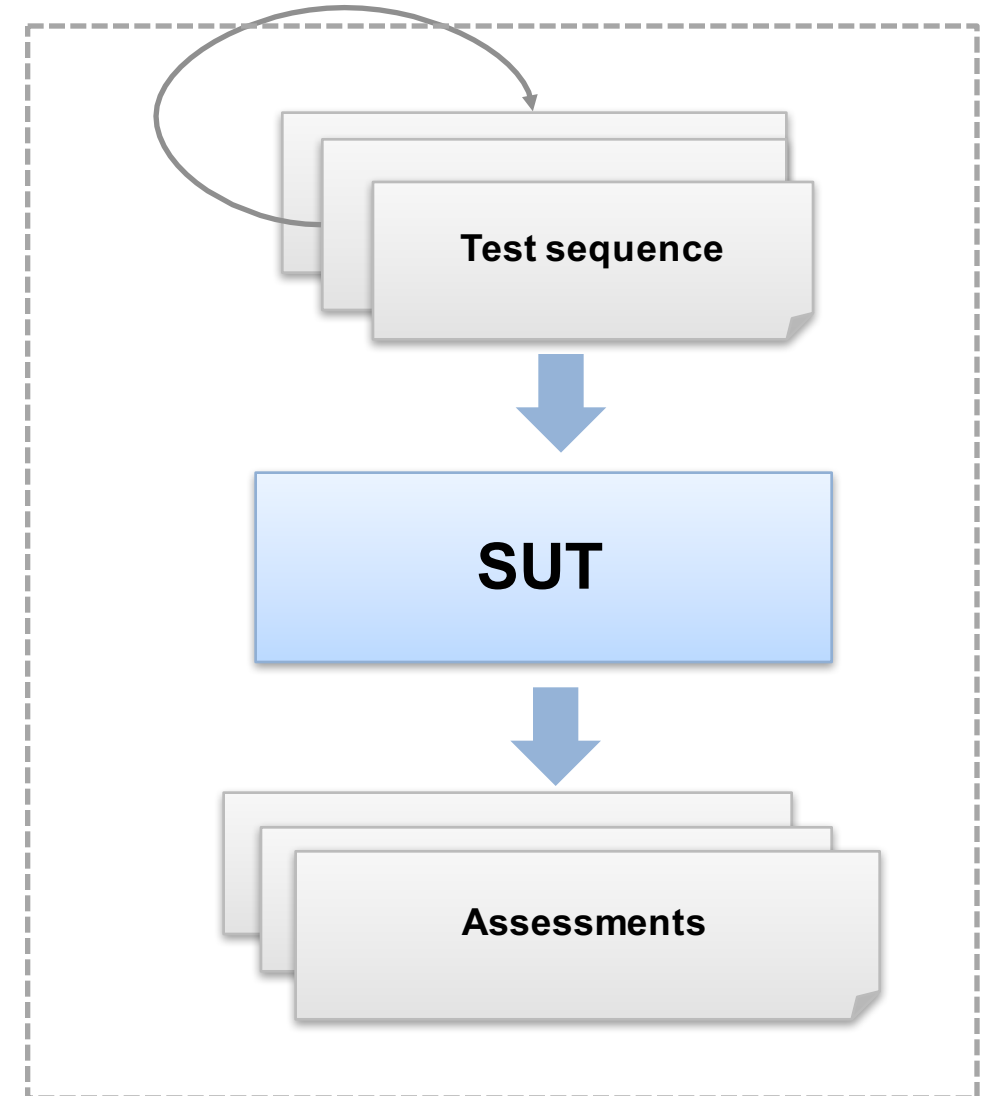
# Test Automation with Simulink Test

- Available to verify algorithm in real-time
  - Reusing Test Harness and Test Sequence in Simulink Test
  - Avoiding multiple build/download to target
  - Avoiding additional programming to access test results



# Key Design Principles

- Use Test Sequence Block for Assessments
  - Evaluated in real-time on target
  - Non-fatal verification language
    - Failure does not stop execution
  - Language constructs for fault recovery
    - Prevent hardware damage
- Runtime variants
  - Avoid multiple build/download to target
- Rapid Iterations
  - Over runtime variants on target hardware



# Summary

- Production Code Generation
  - You can get a code for production without human error.
  - Embedded Coder provides “Quick Start” for beginner to try code generation easily.
  
- Real-Time Testing
  - You can get many benefits with real-time testing
    - Reduce hardware testing
    - Avoid breaking expensive equipment
    - Improve product quality
  - You can do real-time testing in one environment with Simulink.
  - For real-time testing, you can reuse all test cases developed to verify models.