MathWorks® | Training Services
Accelerating the pace of engineering and science

Embedded Coder for Production Code Generation

Training Objectives

This hands-on, three-day course focuses on developing models in the Simulink® environment to deploy on embedded systems. The course is designed for Simulink users who intend to generate, validate, and deploy embedded code using Embedded Coder®.

Topics include:

- Generated code structure and execution
- Code generation options and optimizations
- Integrating generated code with external code
- Generating code for multirate and nonperiodic systems
- Customizing generated code
- Customizing data
- Testing generated code on target hardware
- Deploying code

Prerequisites

Simulink Fundamentals (or *Simulink Fundamentals for Automotive System Design* or *Simulink Fundamentals for Aerospace System Design*). Knowledge of C programming language.

Products

- Embedded Coder®

Course Outline

Day 1 of 3

Generating Embedded Code (2.0 hrs)

Objective: Configure Simulink models for embedded code generation and effectively interpret the generated code.

- Architecture of an embedded application
- System specification
- Generating code
- Code modules
- Logging intermediate signals
- Data structures in generated code
- Verifying generated code
- Embedded Coder® build process

SLEC424V1

Optimizing Generated Code (2.0 hrs)

Objective: Identify the requirements of the application at hand and configure optimization settings to satisfy these requirements.

- Optimization considerations
- Removing unnecessary code
- Removing unnecessary data support
- Optimizing data storage
- Profiling generated code
- Code generation objectives

Integrating Generated Code with External Code (2.0 hrs)

Objective: Modify models and files to run generated code and external code together.

- External code integration overview
- Model entry points
- Integrating generated code into an external project
- Controlling code destination
- Packaging generated code

Controlling Function Prototypes (1.0 hrs)

Objective: Customize function prototypes of model entry points in the generated code.

- Default model function prototype
- Modifying function prototypes
- Generated code with modified function prototypes
- Model function prototype considerations
- Reusable function interface
- Function defaults

Day 2 of 3

Customizing Data Characteristics in Simulink® (1.5 hrs)

Objective: Control the data types and storage classes of data in Simulink.

- Data characteristics
- Data type classification
- Simulink data type configuration
- Setting signal storage classes
- Setting state storage classes
- Impact of storage classes on symbols

Customizing Data Characteristics Using Data Objects (1.75 hrs)

Objective: Control the data types and storage classes of data using data objects.

- Simulink® data objects overview

- Controlling data types with data objects
- Creating reconfigurable data types
- Controlling storage classes with data objects
- Controlling data type and variable names
- Data dictionaries

Creating Storage Classes (1.25 hrs)

Objective: Design storage classes and use them for code generation.

- User-defined storage classes
- Creating storage classes
- Using user-defined storage classes
- Sharing user code definitions

Customizing Generated Code Architecture (1.25 hrs)

Objective: Control the architecture of the generated code according to application requirements.

- Simulink model architecture
- Controlling code partitioning
- Generating reusable subsystem code
- Generating variant components
- Code placement options

Model Referencing and Bus Objects (1.25 hrs)

Objective: Control the data type and storage class of bus objects and use them for generating code from models that reference other models.

- Creating reusable model references
- Controlling data type of bus signals
- Controlling storage class of bus signals
- Model Reference software testing

Day 3 of 3

Scheduling Generated Code Execution (1.75 hrs)

Objective: Generate code for multirate systems in single-tasking, multitasking, and function call-driven configurations.

- Execution schemes for single-rate and multirate systems
- Generated code for single-rate models
- Multirate single-tasking code
- Multirate multitasking code
- Generating exported functions

Testing Generated Code on Target Hardware (2.0 hrs)

Objective: Use processor-in-the-loop (PIL) simulation to validate, profile, and optimize the generated code on target hardware.

- Hardware support overview
- Arduino setup
- Validating generated code on target
- Target optimization overview
- Profiling generated code on target
- Using code replacement libraries
- Creating code replacement tables

Deploying Generated Code (1.25 hrs)

Objective: Create a working real-time application on an Arduino® board using provided hardware support.

- Embedded application architecture
- Creating a deployment harness
- Using device driver blocks
- Running a real-time application
- External mode

Integrating Device Drivers (1.5 hrs)

Objective: Generate custom blocks to integrate device drivers with Simulink and generated code.

- Device drivers overview
- Using Legacy Code Tool
- Customizing device driver components
- Developing a device driver block for Arduino

Improving Code Efficiency and Compliance (0.5 hrs)

Objective: Inspect the efficiency of generated code and verify compliance with standards and guidelines.

- Model Advisor
- Hardware implementation parameters
- Compliance with standards and guidelines

Appendix A: Embedded System Terminology (0.5 hrs)

Summary:

- Real-time systems
- Scheduling methods
- Glossary

Appendix B: TLC Overview (0.5 hrs)

Summary:

- TLC overview
- A first program with TLC
- TLC directives

Appendix C: Fixed-Point Design (1.5 hrs)

Summary: Use Fixed-Point Tool to convert your Simulink model to fixed point.

- Fixed-point scaling and inheritance
- Fixed-Point Designer workflow
- Fixed-Point Tool
- Command-line interface

Appendix D: Stateflow® Code Generation (1.0 hrs)

Summary:

- Code generation with Stateflow®
- Stateflow chart data
- Stateflow storage classes
- Stateflow machine architecture
- Controlling code partitioning in Stateflow charts

Appendix E: Advanced Customization Techniques (1.0 hrs)

Summary:

- Code generation process review
- Custom file processing
- Modifying code generation template (CGT) files
- Modifying custom file processing (CFP) templates

Appendix F: Creating Custom Hardware Targets (2.5 hrs)

Summary:

- Motivation for custom targets
- Custom target development process
- Overview of toolchain integration
- Creating a custom Arduino target
- Deploying code to an Arduino board