Embedded Coder for Production Code Generation

Prerequisites

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

Day 1 of 3	
Generating Embedded Code	**Objective:** Configure Simulink models for embedded code generation and effectively interpret the generated code. System specification Generating code Code modules Data structures in generated code Embedded Coder build process
Integrating Generated Code with External Code	**Objective:** Modify models and files to run generated code and external code together. Overview of external code integration Overview of model entry points Using an execution harness Including custom routines Configuring data exchange with external code
Real-Time Execution	**Objective:** Generate code for multirate systems in single-tasking and multitasking configurations. Real-time harness Execution schemes for single-rate and multirate systems Generated code for single-rate models Multirate single-tasking code Multirate multitasking code
Controlling Function Prototypes	**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 Calling generated code with customized entry points Model function prototype considerations

Day 2 of 3

Day 2 of 3	
Optimizing Generated Code	**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 Code generation objectives
Customizing Data Characteristics in Simulink	**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 Setting parameter storage classes Impact of storage classes on symbols
Customizing Data Characteristics Using Data Objects	**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 Custom storage classes Controlling storage classes with data objects Controlling data type and variable names Data dictionaries
Creating Custom Storage Classes	**Objective:** Design custom storage classes and use them for code generation. User-defined custom storage classes Creating a Simulink data class package Creating a custom storage class Using custom storage classes
Bus Object and Model Referencing	**Objective:** Control the data type and storage class of bus objects and use them for generating code from models that reference other models. Bus signals and model referencing Controlling the data type of bus signals Controlling the storage class of bus signals

Day 3 of 3

Day 3 of 3	
Customizing Generated Code Architecture	**Objective:** Control the architecture of the generated code according to application requirements. Simulink model architecture Controlling Simulink code partitioning Generating reusable code Data placement options Priority of data placement controls
Advanced Customization Techniques	**Objective:** Use code generation templates to control the generated files. Review of the code generation process Overview of code generation templates Custom file processing Defining code generation templates Using code generation templates
Deploying Generated Code	**Objective:** Create a custom target for an Arduino® board and deploy code using the target. Custom target development process Overview of toolchain method Creating a custom Arduino target Deploying code to an Arduino board
Integrating Device Drivers	**Objective:** Identify the workflow for integrating device drivers with Simulink and generated code. Device drivers overview Using the Legacy Code Tool Customizing device driver components Developing device driver blocks for Arduino
Improving Code Efficiency and Compliance	**Objective:** Inspect the efficiency of generated code and verify compliance with standards and guidelines. The Model Advisor Hardware implementation parameters Compliance with standards and guidelines