Codegenerierung für Embedded Systeme leicht gemacht – So geht’s!

Tobias Kuschmider
MathWorks
München, 9.07.2014
Agenda

- Model-Based Design – An Introduction
- Use of Production Code Generation for Implementation
- Proving Functional Equivalence between Model and Code
- Summary
Demonstration – Model-Based Design
Agenda

- Model-Based Design – An Introduction
- Use of Production Code Generation for Implementation
- Proving Functional Equivalence between Model and Code
- Summary
Model-Based Design

- Executable Specifications
- Continuous Test and Verification
- Design with Simulation
- Automatic Code Generation
Model-Based Design
Development Process

**System Design**
- Environment
- Physical Components
- Algorithms

**Component Design**
- Subsystem Design

**Research**
- Data Analysis
- Algorithm Development
- Data Modeling

**Executable Specifications**
- Design with Simulation
- Models
- Continuous Test and Verification

**Automatic Code Generation**
- Code Verification and Validation
- Integration testing
- Complete Integration & Test

**Implementation**
- Subsystem Implementation

**Subsystem Integration & Test**
- User Acceptance Testing

**System-Level Specification**
- Requirements

**Models**
- DSP
- FPGA
- ASIC
- Embedded Software
- Digital Electronics
- C, C++
Model-Based Design
Continuous Verification and Validation

System Design
- Environment
- Physical Components
- Algorithms

Component Design

Research
- Data Analysis
- Algorithm Development
- Data Modeling

Implementation
- Embedded Software
  - C, C++
- Digital Electronics
  - VHDL, Verilog
- Hardware
  - MCU, DSP, FPGA, ASIC

Verification and Validation

Requirements

Subsystem Design

Subsystem Implementation

User Acceptance Testing

Integration testing

System-Level Specification

Complete Integration & Test

System-Level Test

Subsystem Integration & Test
Model-Based Design
Development Process

- Requirements
- Requirements capturing in Word, DOORS, etc.

- System Design
  - Environment
  - Physical Components
  - Algorithms

- Component Design
- Subsystem Design
- Research
  - Data Analysis
  - Algorithm Development
  - Data Modeling

- Implementation
  - Embedded Software
    - C, C++
  - Digital Electronics
    - VHDL, Verilog
  - Hardware
    - MCU, DSP, FPGA, ASIC

- Code Verification and Validation
- Integration testing
- User Acceptance Testing
- System-Level Integration & Test
- Complete Integration & Test

- Subsystem Implementation
Model-Based Design
Multi-Domain Modeling and Algorithm Development

Methods for modeling systems in different domains

- Requirements
- System-Level Specification
- Environment
- Physical Components
- Algorithms

- Data Flow (Block diagram)
- Physical Modeling (Schematic)
- Event-Driven Systems
- Programing Language (Textual)

- Research
  - Data Analysis
  - Algorithm Development
- Data Modeling

- User Acceptance Testing
- Complete Integration & Test
- System-Level Integration & Test
Model-Based Design

Early Concept Verification

- Executable specifications
- Predict dynamic system behaviour by simulation
  - System & environment models
  - Less physical prototypes
- Use of simulation results for system design
  - Fast What-If studies
  - Short iteration cycles

Requirements

System-Level Specification

Research

Data Analysis

Algorithm Development

Data Modeling

System

Environment

Physical Components

Algorithms

Idea

Simple Model

Detailed Model

User Acceptance Testing

Complete Integration & Test

User Acceptance Testing
Demonstration – Motor Control, Design
Agenda

- Model-Based Design – An Introduction
- Use of Production Code Generation for Implementation
- Proving Functional Equivalence between Model and Code
- Summary
Model-Based Design
Development Process

- **System-Level Specification**
  - Environment
  - Physical Components
  - Algorithms

- **Component Design**
- **Subsystem Design**
- **Implementation**
  - Embedded Software
  - Digital Electronics
  - MCU
  - DSP
  - FPGA
  - ASIC

- **Integration & Test**
- **Code Verification and Validation**
- **User Acceptance Testing**
- **Complete Integration & Test**
Model-Based Design
Automatic Code Generation

- C/C++, VHDL and PLC-Code Generation from **one model**
- Support for Fixed Point Data Format
  - Automatic scaling
  - Supported in Simulation and Code-Generation
- Easy integration of legacy C/C++-Code
- System development independent of the target
Demonstration – Motor Control, Code Generation
Codegeneration Report

- Automatically generated
- HTML links between model and code
- Contains all relevant information
Plattform Independent ANSI ISO C-Code

- Plattform independent C/C++ Code
- Navigation between model and code ensuring traceability

```c
/* Gain: '<S14>/number_of_pole_pairs'

sin_coefficient = ctrlParams.PmsmPolePairs * B.Switch_fr;

/* Trigonometry: '<S14>/Trigonometric Function1' */

cos_coefficient = (real32_T)cos(sin_coefficient);
sin_coefficient = (real32_T)sin(sin_coefficient);
```
Optimized Code for ARM Cortex-M

- Code Replacement Tables
- Use of CMSIS library for code optimization
Agenda

- Model-Based Design – An Introduction
- Use of Production Code Generation for Implementation
- Proving Functional Equivalence between Model and Code
- Summary
Model-Based Design
Continuous Verification and Validation
Model-Based Design
Continuous Verification and Validation

Requirements

System Design
- Environment
- Physical Components
- Algorithms

System-Level Specification

System Tests
- Test vectors and expected responses
- System tests in simulation

Component Tests
- Test vectors and expected outputs
- Run component tests on target
- Run system tests on integrated controller

Component Design

Implementation
- Embedded Software
  - C, C++
- Digital Electronics
  - VHDL, Verilog
- MCU
- DSP
- FPGA
- ASIC

Research
- Data Analysis
- Algorithm Development
- Data Modeling

Subsystem Design

Complete Integration & Test

Integration testing

User Acceptance Testing

System-Level Integration & Test

Subsystem Integration & Test

Subsystem Implementation

System validation/Acceptance Test

Run system tests on integrated controller

Component tests in simulation

System tests in simulation
Model-Based Design
Subsystem-Level Integration & Testing

Processor-In-The-Loop Simulation
- Co-Simulation of real hardware and simulated environment
- Testing functional Equivalence

Integration testing
System-Level Integration & Test
User Acceptance Testing
Complete Integration & Test
Verification and Validation
Subsystem Integration & Test
System Analysis

Requirements

Requirements
Demonstration – STM32F4 – Discovery, Equivalence Testing
Proving Functional Equivalence

- Pure simulation vs. Processor-In-The-Loop simulation
- Proof of functional equivalence successful
ARM Cortex-M4: Add Peripheral Blocks, Generate Code, and Deploy!

STM32F4xx Blockset
Agenda

- Model-Based Design – An Introduction
- Use of Production Code Generation for Implementation
- Proving Functional Equivalence between Model and Code
- Summary
Benefits of Model-Based Design

- Models: Core of the Development Process
- Unambiguous Description of Requirements (Executable Specification)
- Fast Evaluation of Design Variants
- Frontloading - Early Test and Verification
- Automatic Code Generation

⇒ Better Cooperation, Communication and Collaboration
⇒ Higher Product Quality