MATLAB EXPO 2015
KOREA
2015년 5월 21일 목요일
인터컨티넨탈 코엑스, 서울
신호처리 응용을 위한 Model Based Design Workflow

이웅재 부장
CASE:
Software in Signal Processing Application (Medical)

Medical devices are increasingly driven by complex software

- Pacemaker → 80,000 LOC*
- Infusion Pump → 170,000 LOC
- MRI Scanner → 7 Million LOC

* LOC – Lines of Code
**Problem:**
Increasing Device MDRs* and Recalls

- “1 in 3 of all software-based medical devices sold in America between 1999 and 2005 had been recalled for software failures”**

- Devices Commonly Recalled
  - Infusion pumps
    - largest category of recalls and MDRs
  - Implantable medical devices
  - Automated External Defibrillators

---

* Medical Device Report – usually submitted by users or manufactures of Medical Device for FDA when adverse experiences with the device is reported
** Economist June 2012 article “When code can kill or cure”, referring to study done by University of Patras in Greece on Medical Device recalls
Example:
Infusion Pumps

- Between 2005 - 2009, FDA received 56,000+ reports of adverse events relating to infusion pumps

- Numerous device recalls annually
  - 87 recalls, with 14 Class I
  - Causes: software, user interface, mechanical/electrical

- “Infusion Pump Improvement Initiative” in April 2010
  - Assurance cases
  - Application-specific software safety requirements
  - Reference Specifications/Requirements
Agenda

Introduction to Model-Based Design
*Case Study – ECG Beat Detection Demo*

Algorithm Modeling and Simulation
*Build, simulate and refine the model of the ECG beat detection algorithm*

Requirements Traceability
*Connect model elements with Requirements and generate traceability reports*

Verification, Validation, and Testing
*Functionally test the model, perform model coverage analysis, generate test cases*

Implementation
*Generate C code and verify them on your processor*

Code Verification
*Perform static code analysis with Polyspace to formally prove software safety*
Traditional Development Process

- **RESEARCH**
- **REQUIREMENTS**
- **SPECIFICATIONS**
- **DESIGN**
  - EDA: Electrical Components
  - Algorithm Design: Embeddable Algorithms
  - MCAD/MCAE: Mechanical Components
- **IMPLEMENTATION**
  - HDL: FPGA, ASIC
  - C/C++: MCU, DSP
  - ST: PLC, PAC

**INTEGRATION AND TEST**

MATLAB EXPO 2015
Traditional Development Process

**RESEARCH**

**SPECIFICATIONS**
- EDA
  - Electrical Components
- Algorithm Design
  - Embeddable Algorithms
- MCAD/MCAE
  - Mechanical Components

**DESIGN**

**IMPLEMENTATION**
- HDL
  - FPGA
  - ASIC
- C/C++
  - MCU
  - DSP
- ST
  - PLC
  - PAC

**INTEGRATION AND TEST**

- Requirement Documents
  - Difficult to analyze
  - Difficult to manage as they change

- Paper Specifications
  - Easy to misinterpret
  - Difficult to integrate with design

- Physical Prototypes
  - Incomplete and expensive
  - Prevents rapid iteration
  - No system-level testing

- Manual Coding
  - Time consuming
  - Introduces defects and variance
  - Difficult to reuse

- Traditional Testing
  - Design and integration issues found late
  - Difficult to feed insights back into design process
  - Traceability
Model-Based Design

- RESEARCH
- REQUIREMENTS
- DESIGN
  - Environment Models
  - Physical Components
  - Algorithms
- IMPLEMENTATION
  - C, C++
  - VHDL, Verilog
  - Structured Text
  - MCU
  - DSP
  - FPGA
  - ASIC
  - PLC
- INTEGRATION
- TEST AND VERIFICATION
  - Design as Executable Specification
  - Requirements Traceability
  - Continuous and early Verification
  - Document and Report Generation

MATLAB EXPO 2015
Model-Based Design Fits within Your Development Process
The Challenge: ECG Beat Detection

- Algorithm created by the research team and published
- Next step is to take it to production complying with all the requirements of a standard such as IEC 62304
From Idea to Requirements
Detecting beats in ECG with high accuracy
Workflow Summary: ECG Beat Detection

- **Verify** Requirements
- **Model** Source Code
- **Object Code**
- **Quality Standards**
- **Validate**
ECG Beat Detection – A Quick Look
Requirements Management Interface
Traceability between requirements and model elements

- **Associating models and requirements**
  - Establishing a link from a model block or test case to requirement

- **Managing changes** in models and requirements
  - Detecting changes in requirements associated to a model/test element
  - Detecting changes in model/test associated with a requirement

- **Reporting** requirement coverage for model blocks and test cases
  - What algorithmic blocks are covered by requirements and vice-versa
  - What test cases are covered by requirements and vice-versa

- **Identify unmapped elements**
  - Additional elements in your design not mandated by requirements?
  - Missing requirements?
Model Advisor Report
Process documentation and audit tool

Useful as a process audit document:

- Detailed summary
- Report follows exact order of the Model Advisor tree
- Model Check results: Pass, Fail, Warning, and Not Run
Model Coverage
Measure of test completeness

- Execution analysis
  - Dynamic data collection during simulation

- Coverage results
  - Displayed directly in the model
  - Available in a separate html report linked with the model objects

Decision coverage
Condition coverage
MC/DC
Lookup table coverage
Signal range coverage

Supported coverage types
Test Generation
Increase Design Coverage

- Automatically generates test vectors to achieve 100% coverage
  - Detects unreachable states
  - Find test vectors for coverage not achieved by functional tests

- Save test vectors
  - Automatically generate a separate model with test harness
  - Export test vectors to .CSV file

- Automatically generates test vector report
  - Two-way mapping of objectives and generated vectors

MATLAB EXPO 2015
Property Proving
Formal Proof of Validity or Invalidity of Model Properties

- Generates a proof for a requirement
  - Example: Heart rate of patient model + pacemaker must lie between 50 and 70 bpm
- Proves model properties and generates reports and examples of violations

Uses **formal methods**, not simulation
Use the Traceability Report section of the Real-Time Workshop Embedded Coder code generation report to review mapping between model elements and generated code.
Use Simulink Verification and Validation to trace between model elements and requirements. Use Embedded Coder to trace between generated code and its source model.

Model used for production code generation

C source code

/* Exported block signals */
real_T INPUT;
real_T OUTPUT;

/* Exported block parameters */
real_T k = 5.0;

. OUTPUT = INPUT * k;
Processor-in-the-Loop Testing
Functional Verification of Object Code (after cross-compilation)

- Model in simulation and code on the processor running in parallel

PIL also provides execution profiling, code coverage reports, and interactive debugging.
Polyspace
Prove presence or absence of run-time errors

Formal Methods based Static Code Analysis

```c
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;
    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }
    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5;
        } else {
            i++;
        }
    }
    i = get_bus_status();
    if (i >= 0) {
        *(p - i) = 10;
    }
}
```

**Green:** reliable
- safe pointer access

**Red:** faulty
- out of bounds error

**Gray:** dead
- unreachable code

**Orange:** unproven
- may be unsafe for some conditions
Summary

Model Based Design

RESEARCH

REQUIREMENTS

DESIGN

Environment Models
Physical Components
 Algorithms

IMPLEMENTATION

C, C++
VHDL, Verilog
Structured Text
MCU DSP FPGA ASIC PLC

INTEGRATION

TEST AND VERIFICATION

Design as Executable Specification
Requirements Traceability
Continuous and early Verification
Document and Report Generation