Model-based generation of tests for embedded systems: the MOGENTES EU project

Stefano Marzani, Lorenzo Fantesini
s.marzani@re-lab.it, l.fantesini@re-lab.it

Workshop SPIN, 04/06/2009
Background: RE:Lab MBD approach (1)

Model-based Generation of Tests for Embedded Systems

ALTIA Design

Matlab Simulink

PC Simulation

Code generation for embedded systems

Virtual prototype

RE:Lab CAN BLOCKS

Interfaccia CANbus

On board BUS

On-vehicle prototype
Background: RE:Lab MBD approach (2)
MOGENTES overview

- Model-Based GEneration of TEst-Cases for dependable Embedded Systems

- **Objective:**
  - automatic generation of test cases to enhance *testing* and *verification* of dependable embedded systems
  - target applications: trains, agricultural machines, cars
  - a common model-based approach (*framework*) for test and validation
MOGENTES Consortium

Budget: 4.436.511 €

Partners:
- Austrian Research Centers GmbH - ARC
- Swiss Federal Institute of Technology Zurich / University of Oxford
- Ford GmbH
- Budapest University of Technology and Economics
- Graz University of Technology
- PROLAN
- Prover Technology AB
- SP Technical Research Institute of Sweden
- Thales Rail Signalling Solutions
- RE:Lab
MOGENTES overall framework

- Automatic generation of efficient test cases to verify system safety correctness using formal methods and fault injection
Framework application

- Bucket: off-highway machine
Phase 1: user requirements and specs

- Highlight, analyze and validate system requirements
  - **Classification**: requirements are grouped according to their characteristics
  - **Management**: system specifications are generated and translated into product design guidelines
Phase 1: user requirements and specs

- **Functional**
  - **System Behaviour**
    - The system must read the incoming CAN messages input signals of the control joystick each time a message is received.
    - The ISOBUS joystick must send a status message at least every 100 milliseconds

- **Non functional (e.g. ISOBUS requirements)**
  - **ECU initialization**
    - Initialization of an ECU with address claim and no contention
Phase 2: model development

- Basing on system specifications, the model (prototype) of the final system is developed.
  - Behavioural specifications modelling (Matlab simulink/Stateflow, Visual Studio)
  - Isobus specifications implementation
  - User Interface graphics implementation (Altia, PConvert)
Phase 2: model development (examples)

- System behaviour modelling (State Flow)

- ISOBUS implementation
Phase 2: model development – Overall

- Control Logic and Algorithms
- ISOBUS Protocol and Graphic

Custom Code integration: Simulink S-Functions

Proprietary Graphic Development: Altia Design

Stateflow
Phase 2: model development – CAN bus
Phase 2: model development

- User Interface graphics implementation

  - Proprietary graphics
  - ISOBUS graphics
  - System behaviour model and ISOBUS network

- Not strictly related to CAN messages
- Each object pool related to a CAN message
Phase 2.1: model validation

- At the end of the process, the model becomes the reference for the validation of the final system
- Before, the model should be validated

Model Checking Technique - Validation of the system logic

Signals from CAN and Signal Machine

System behaviour model and ISOBUS network

Model logic output

System specifications (expected output)

Check
Phase 2.1: model validation

Validation of the graphics

System behaviour model and ISOBUS network

Signals from CAN and Signal Machine

System specifications (expected output)

Graphic outputs

Check
Phase 3: code generation

- Code generation: from the validated model to the target final system
Phase 3: code generation

- From CAN blocks, Matlab/Simulink/StateFlow and Altia it is possible to generate code for specific embedded targets.

- Using this target it is possible to generate the C code firmware for a specific microcontroller and to create directly a project file ready for compilation with the proprietary toolchain.
Phase 4: final system validation

- Requirements and specifications
- Model (virtual prototype)
- Code generation for the final system
- Final system (sw and hw)
- Validation comparing Test Cases results
- Signals from CAN and signal machine
- Test case model (UML): signal selected for the test case
Phase 4: final system validation

- Inputs to the model and the final system are:
  - CAN messages, sent using the Vector CANoe tools together with the generable CAN Blocks inside the Matlab/Simulink model environment.
  - Voltage and Current signals, generated by a Signal Machine.

- Outputs
Phase 4: final system validation

- Proprietary graphics validation

Model outputs

Final system outputs
Phase 4: final system validation

- ISOBUS graphics validation
  - Object Pools test is performed by testing the CAN frames through the network bus and verifying if the displayed Object Pool is the correct one with respect to the Standard ISOBUS (ISO 11783).
Final validated system
Conclusions

- MOGENTES: automatic generation of test cases to enhance testing and verification of dependable embedded systems
- Target RE:Lab application: agricultural machine bucket
- Framework for test and validation

![Diagram showing the flow from requirements and specifications through model (virtual prototype) to final system (sw and hw) and test case validation, with signals from CAN and signal machine.]