SOFTWARE DEVELOPMENT DEMANDS IN AUTOMOTIVE

Challenges:
Quality and cost
Debugging million lines of code
Efficient Calibration
Selecting across variants

Solutions:
Standard Design & Architecture
Model Based Development
Parallel Software Development
Multi-Core ECU

MODEL BASED DEVELOPMENT

Modelling

- Virtual representation of real world problem
- Codes are generated based on models

Simulation

- Testing the behavior of models in conditions difficult to reproduce the physical models

Validation

- Validation at every iteration between modeling and simulation
- Improves quality of the final software product

SDLC – V CYCLE: Software Requirement Analysis (done by Program Manager), Software High level Design (done by Development team), Software Low Level Design, Model Dev, Model In Loop testing, Code Generation, Software-In-Loop Testing, Hardware-In-Loop Testing (Testing team), Software Acceptance Test.

TRADITIONAL SYSTEM DEVELOPMENT CYCLE

- Requirements and Specifications (Requirement in form of text: monotonous and difficult to interpret)
- Design (Design to Hand Written Codes – Time consuming, error prone and expensive)
- Implementation, Test and Verification: Direct test to Hardware – Too late to fix

BENEFITS – MODEL BASED DEVELOPMENT

- Unambiguous description of requirements (Executable specification)
- Fast evaluation of design variants
- Test and validation at each stage
- Reduced development cost
- Quicker code generation
- Auto code model templates, once developed can be re-used
- Easy to maintain model versions and keep updating in future

REQUIREMENT SPECIFICATION: SOFTWARE MODEL

- Requirement model emphasizes states, interfaces and logical rules – most of the logics, tables and state machines can be translated to a requirement model. Logic at simulation works exactly the same as logic in vehicle.
- Requirement model emphasizes readability over efficiency- avoid unreadable and cryptic modelling techniques; clarity in specifications prevent erroneous implementation.
- Requirement model should be executable- using minimum number of tools lead to good modelling efficiency.

VERIFICATION AND VALIDATION

- Verification: The process of evaluating a system or a component to determine whether the product of the given development phase satisfies the conditions imposed at the start of the phase.
- Does the software comply with specifications..?

- Validation: The process of evaluation of a system or component during or at the end of the development process to determine whether it satisfies specified requirements.
- Does the software do what the customer really wants.. ?

- Software criticality level helps in determining efforts required for verification and validation
- Malfunctions at the Embedded Software System level may result in (Safety critical) Injury, illness or loss of life and (Business critical) serious environmental damage, significant loss or damage of property and major economic loss.

X. DYNAMIC VERIFICATION AND VALIDATION

Dynamic verification and validation strategies involving simulation code and observing the behavior to identify the errors. Types of techniques are: Model-in-loop, Software-in-loop, and Hardware-in-loop.

X.1 MODEL-IN-LOOP TESTING

- Model-in-loop testing (MIL) and simulation is a technique used to abstract the behavior of a system or sub-system in a way that this model can be used to test, simulate and verify that model.
- By using an industry standard toolchain such as Simulink for model definition, you can test and refine that model within a desktop environment, allowing a complex system to be managed efficiently.
- Once you have modelled your control system or environment (plant model), you can use this plant model to test your controller strategies. Control strategies are usually test cases which are developed in tune with the functional requirement of the system.
- MIL testing is done at a stage when we are ready with the base design of the requirement. All it needs is to have a simulatable model with all inputs defined in workspace.
- Apart from MATLAB, even 3rd party tools can aid in Model-in-loop testing.

X.2 HARDWARE-IN-LOOP TESTING

- Hardware-in-loop (HIL) simulation is a type of real-time simulation. HIL simulation is used to test the controller design.
- HIL simulation shows how the controller responds, in real time, to realistic virtual stimuli. HIL is also used to determine if the physical system (plant) model is valid.
- In HIL simulation, a real-time computer is used as virtual representation of the plant model and a real version of the controller.
- The development hardware contains the real-time capable model of the controller and plant. The hardware contains an interface with which to control the virtual input to the plant.
- The controller hardware contains the controller software that is generated from the controller model.
- The real-time processor (target hardware) contains code for the physical system that is generated from the plant model.
- Use HIL simulation to test the design of your controller when you are performing Model-Based Design (MBD). The figure shows where HIL simulation fits into the MBD design-to-realization workflow.
- One does not have to rely on the on a naturalistic or environmental test setup. By utilizing the model to represent the plant, HIL simulation offers benefits in cost and practicality.
- There are several areas in which HIL simulation offers cost savings over validation testing.
- HIL simulation tends to be less expensive for design changes. HIL simulation when performed earlier than validation in the MBD workflow so you can identify and redesign for problems relatively early in the project.

X.3 SOFTWARE-IN-LOOP TESTING

- Software-in-loop testing (SIL) enables the user to verify the production – ready source code and complied object code.
- In SIL, the model can be tested and refined within a desktop environment, allowing a complex system to be managed efficiently.

Y. TESTING THEORY

- Testing is a systematic way to check the correctness of a system by means of experimenting with it.
- Tests are applied to a system in a controlled environment, and a verdict about the correctness of the system is given, based on the observation during the execution of the test.
- Thorough testing is necessary to be confident that the system works as it was intended to in its intended environment.
- When testing software, there are often a massive amount of possible test cases even in quite simple systems.
- Tests are normally performed in several different stages of the V-cycle; the V-cycle shows breaking down of requirements in more detailed manner and integrating it into a ready product.
- Common types of testing, in concurrence with different stages of V-cycle are: Unit Testing, Integration Testing, System Testing and Acceptance Testing.

Y.1 TYPES OF TESTS

- **Unit testing:** A unit test is performed on a single component of a system, or in the case of a software, it would be single program or a function. In the unit tests, it is interesting to look at different coverage criteria to verify that most of the system has been covered by the test.
- **Integration testing:** In the integration test, several units that are supposed to interact in the system are integrated and tested to debug and verify the correctness of the system.
- **System testing:** System testing is sometimes divided into functional system testing and non-functional system testing. In the functional system tests, it is possible to test the whole system against the specification. The requirements and the functionality of the system can be evaluated. Non-functional system testing is a way to test properties such as performance, flexibility etc.
- **Acceptance testing:** Acceptance testing is the last level of the validation process. It should be performed in as realistic environment as possible. The acceptance test is often done by or with the customer. In this test, the focus is on getting requirements met for the system, not to try and find the defects.

Z. TYPES OF COVERAGE METRICS

As per the test requirement, one can choose one among the following coverage analysis:

- Decision coverage
- Condition coverage
- Modified condition/ Decision coverage (MCDC)

- **Decision coverage:** analyzes elements that represent decision points in a model, such as a switch block or State Flow states. For each item, decision coverage determines the percentage of the total number of simulation paths through the item that the simulation traversed.

- **Condition coverage:** analyzes blocks that output the logical combination of their inputs (for example, the logical operator block) and State Flow transitions. A test case achieves full coverage when it causes each input to each instance of a logic block in the model and each condition on a transition to be true at least once during the simulation. Condition coverage analysis reports whether the test case fully covered the block for block in the model.

- **Modified condition/decision coverage:** analyzes blocks that output the logical combination of their inputs and State Flow transitions to determine the extent to which the test case tests the independence of logical block inputs and transition conditions.
  - A test case achieves full coverage for a block when a change in one input, independent of any other inputs, causes a change in the block output.
  - A test case achieves full coverage for a State Flow transition when there is at least one time when a change in the condition triggers that transition for each condition.
  - If the model contains blocks that define expressions that have different types of logical operators and more than 12 conditions, the software cannot record MCDC coverage. You can achieve 100% MCDC coverage without achieving 100% decision coverage.