Keeping in the loop

How in the loop testing aids embedded system validation By **Goran Begic.**

Embedded systems software can provide competitive advantage. For a vehicle, software can make a comfortable ride even more appealing; it can also reduce cabin noise or fuel consumption. Some of this functionality could, potentially, be implemented in hardware, but that would increase manufacturing cost and product price. Software also enables reuse and can be updated more frequently to satisfy user needs.

As embedded software becomes more complex, its functions become more difficult to test and verify. The verification and validation overhead and cost of fixing of defects and implication of defects that were not detected on time threaten to wipe out all the benefits that software provides.

The embedded software development industry has recognised and embraced graphical models as a way to deal with increased complexity. Coupled with simulation, graphical models of product functionality are an opportunity to improve verification and validation processes.

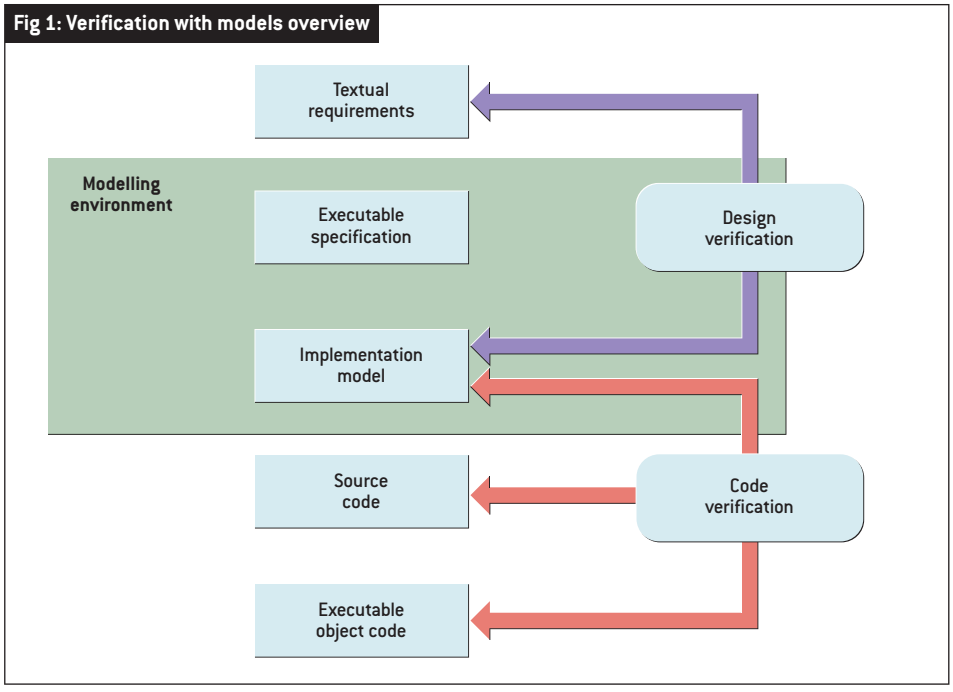
Developing and testing with models

Modelling is the step between the collection of high level requirements and implementation. Models allow testing and verification to be done continuously, in parallel with system design and implementation.

In the early design stages, one can develop behavioural models to clarify and define detailed low level requirements. Such models may have the basic architecture of the solution, but are independent of the target platform. A model used to capture key requirements and to demonstrate correct behaviour in simulation, as well as to demonstrate traceability to high level requirements, is often referred to as executable specification.

Further development of the executable specification and the addition of implementation details leads to the definition of a model that represents a final implementation. Often, such a model is optimised for code generation; it honours the data types, the target architecture and even the required coding style. Changes require a verification process that ensures the change introduced in the model for production code generation doesn't change the model's behaviour.

Fig 1: Verification with models overview



Code verification determines correct behaviour of the model for production code generation and of the generated code. Distribution of the effort in design verification and code verification allows an early start to the process, more focused testing and shorter time to fix the problems. Methods for design verification and code verification include:

- Model testing in simulation
- Software in the loop testing
- Processor in the loop testing
- Hardware in the loop testing

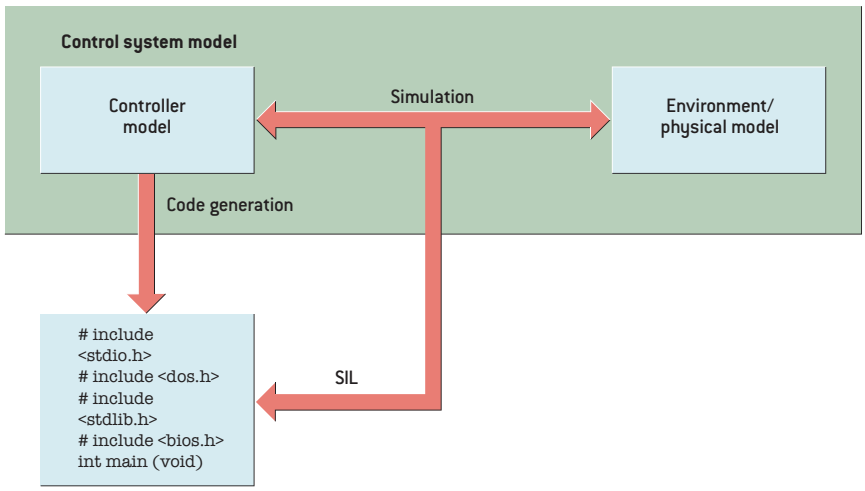
Model testing

Unlike 'static' designs, an executable specification can be evaluated in simulation. Typically, this is done by changing model parameters, or input signals, and by reviewing the outputs, or responses. The behaviour of software functionality captured in simulation must meet the expectation specified in requirements.

Signal inputs and model outputs are represented as time series data; sequences of data points in time. Typically, such test vectors can be derived from requirements, but test data can also come from measurements of existing systems or from a model of the physical system with which the embedded software interacts.

Testing methods where inputs to the software model are created with help from other models are sometimes referred to as 'model in the loop' testing. The term 'in the loop' comes from control systems, where there is feedback

Fig 2: Platform independent testing in simulation



between an embedded controller and the device being controlled.

A model not only includes the software's functional design, but also the controller's environment, links to higher level requirements, test vectors and expectation results. This becomes an 'executable specification'. Results of model testing in simulation verify that software behaviour is correct and that it validates the requirements used as the starting point of the development process. Information collected through simulation becomes the benchmark for code verification.

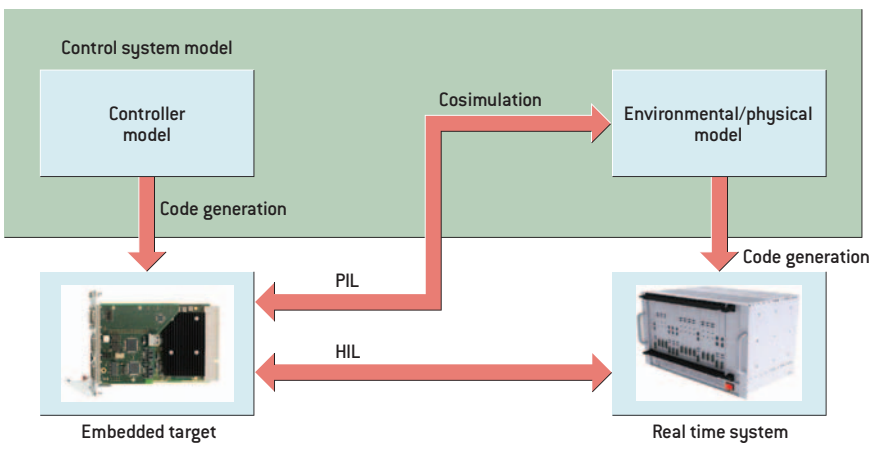
Software in the loop

Software in the loop (SIL) testing evaluates the functions of generated or hand written code in cosimulation on the host machine. As in model testing, input test vectors can come from requirements or other models in the executable specification. SIL tests typically reuse the test data and model infrastructure used for model testing in simulation.

This type of verification is particularly useful when software components consist of a mix of generated code (for example, updates to meet new requirements) and handwritten code (for example, existing drivers and data adapters) that may be necessary for execution on the target platform.

SIL testing is also used for verification when existing algorithms are reused in graphical models. Some legacy code, while correct, may be difficult and expensive to maintain and it makes sense to reimplement and verify it in a graphical environment. In this case, models and simulation are the test framework for comparison of outputs of the new model implementation.

Fig 3: Platform dependent testing



Processor in the loop

A good starting point for the verification of compiled object code on the target platform is to ensure functional equivalence of code running on the target processor relative to the model behaviour captured in simulation.

Conceptually, processor in the loop (PIL) testing is similar to SIL. The key difference is that, during PIL, code executes on the target processor or on an instruction set simulator. The data passed between the model and the deployed object code use real I/O. The SIL model can be reused as a test execution framework for the processor board. With PIL, tests are executed with the embedded algorithm compiled and deployed on the target processor board and with the existing executable specification.

Besides tests for functional equivalence, PIL also lets us step through the assembly level instructions in the debugger and to analyse code compiled, linked and deployed as it will run on the real system. With PIL, we can review the order in which code functions execute, as well as verify calls to OS functions or other libraries required for the execution on the target. The memory footprint can also be reviewed. In some projects, PIL is an opportunity to compare algorithm behaviour on processor boards from different vendors.

Hardware in the loop

The methods mentioned above can't verify the real time aspects of the design because of the simulation and communication overhead with the target board.

To reuse data created via the methods described earlier, and to continue using the results as a guideline and a benchmark for real time testing, we can generate code for the software model's environment and deploy it on a real time target. Such a configuration reduces the risk of testing on actual, and often expensive, devices. This type of verification needs sophisticated signal conditioning and power electronics to stimulate inputs properly and to receive outputs from the target hardware.

Author profile:
Goran Begic is with The MathWorks.