Functional verification on PIL mode with IAR Embedded Workbench

The increase of complexity of embedded system components combined with time-to-market constraints push toward the adoption of more efficient techniques for computer-aided modeling, simulation and verification of industrial Embedded Software (ESW) applications. In particular, ESW verification is becoming a more critical activity in the production process in terms of required costs and quality.

Nowadays, the most common tools and solutions available on the market for verifying ESW are based on structural code analysis and concentrate on checking that all structural items, e.g., executable statements, branches, and paths, within the ESW application have been exercised. A much more important and expensive task consists of checking if the software respects what is specified by its requirements. Functional verification is the process of proving that the software behaves exactly as defined by specifications and is inherently concerned with behavioral or functional aspects that are orthogonal to the structural code analysis. Functional verification is currently the most expensive task of the testing phase; while coverage-based testing approaches are fully automated, functional verification is still a human-based process.

This article presents an overview of a new solution for performing functional verification of ESW in an industrial domain. State-of-art techniques for functional verification are implemented in a tool chain that automatizes their application to a real design. The verification sessions are performed in a well-known debugging environment provided by IAR Systems’ C-SPY Debugger that allows performing verification with Processor-in-the-Loop (PIL) mode. In order to guarantee the maximum portability of the proposed solution, we have also defined a well-structured architecture, which allows automatically integrating with different IAR Systems debugging suites and to immediately run PIL verification on different HW target. This solution allows verifying if the implemented software behaves correctly on its destination hardware and if the requirements provided are satisfied.

Functional verification of Embedded SW

Nowadays, the de-facto approach to guarantee the correct behavior of ESW is monitoring the system simulation: company verification teams are responsible for putting the system into appropriate states by manually generating the required input stimuli, judging when stimuli should be executed, manually simulating environment and user interactions, and analyzing the results to identify unexpected behaviors. In this context, dynamic Assertion-based Verification (ABV) provides verification engineers with a way for formally capturing the intended specifications and runtime check their compliance with the actual design.

In ABV, specifications are expressed by means of temporal assertions. To overcome the ambiguity of natural languages, they are defined according to formal assertion languages, like the Property Specification Language (PSL). PSL assertions can be verified by model checking (static ABV) or by simulation (dynamic ABV). However, dynamic ABV is preferred in case of large designs due to its scalability. In particular, in the hardware domain, dynamic ABV is affirming as a leading strategy in industry to guarantee fast and high-quality verification of hardware components. In dynamic ABV, assertions defined in formal languages are compiled into modules that capture assertion meaning and monitor if they hold during application simulation.

radCHECK is comprehensive suite that integrates dynamic ABV earlier in the ESW design flow also before the final source-code and target platform are available. radCHECK guides the requirements formalization for automatic generations of assertions and stimuli. Assertions turn design specifications into verification objects to automatically control that the right things do happen during simulation. The user needs to verify if the whole design meets the defined assertions. Then, radCHECK performs the automatic conversion of the assertions into C++ modules (i.e., checkers) that can be simulated or executed into a real-time environment. This allows moving from the static verification of formal properties to the dynamic verification of the corresponding checkers, making the verification process
feasible and faster also for complex systems. Another important feature is the automatic generation of test cases to simulate the design and automatize its execution. Figure 1 presents an overview of the workflow performed by radCHECK.

Figure 1: Functional verification workflow

However, simulation techniques do not consider requirements and restrictions imposed by the hardware, e.g. timing constrains, drivers and API impact on ESW, etc. Very often the release of the final software version running on target requires adjustments that can involve significant difference from the original code.

Dynamic ABV implemented in radCHECK can be particularly powerful when moving from PC-based simulation to run verification on the application running on the actual hardware. Processor-in-the-loop (PIL) is a technique useful for testing and validating the controller algorithm on the processor. It is also a very effective but cheap solution with respect to Hardware-In-the-Loop as it provides an intermediate stage between simulation and deployment and does not require any particular hardware stubs further to the final target board.

The guided formalization of functional specifications, the model-driven design of the ESW, the automatic stimuli generation, and the accurate simulation in a PIL environment permits to early identify design flaws before moving to real-world test of the physical plant. In order to provide a solution for application fields that traditionally do not adopt functional verification due to its complexity, we analyzed IAR Embedded Workbench and defined a feasible and profitable solution for integrating PIL with ABV verification performed in radCHECK.
Solution for PIL ABV with the C-SPY Debugger

We defined a combination of Model-driven Design (MDD) and dynamic ABV as an effective solution for ESW verification, that combined with the IAR Embedded Workbench platform allows dynamically verifying the complete system before moving to real-world tests. This solution has been successfully implemented integrating 3 off-the-shelf tools: MDD with radCASE, a rapid-application-development environment that covers the complete modeling and synthesis process of ESW; dynamic ABV, performed with radCHECK that guides to requirements formalization for automatic checker and effective stimuli generation; IAR Embedded Workbench, a powerful and complete framework for the development and debugging of ESW, which with the C-SPY Debugger provides a platform for running the test stimuli and checkers to accomplish PIL verification.

An important characteristic of the implemented solution is a well-defined abstraction layer and architecture that guarantee the portability to any ARM Cortex-based microcontroller supported by IAR Embedded Workbench. As a case of study we have used IAR KickStart Kit for LPC1788 with an ARM Cortex-M3 microcontroller from NXP, which is shown in Figure 2.

![Figure 2: IAR KickStart Kit for LPC1788](image)

Starting from the informal specifications and requirements, the designer, with the model editor of radCASE, defines the ESW model using an UML-based approach. radCASE generates the application C code that is compiled and downloaded on the target using IAR Embedded Workbench.

To guarantee the portability (of the radCASE application) on all IAR Embedded Workbench families, we have created a well-defined abstraction layer. To perform this task we have exploited the Hardware-Abstraction Layer provided by radCASE working up to the data elements and implemented the architecture depicted in Figure 3. This architecture provides an interface mechanism and isolates the target-specific instructions in order to reduce modifications when using different version of IAR Embedded Workbench.

We used mainly two design patters to implements the integration between radCASE and IAR Embedded Workbench: a bridge that provides an interface that must be implemented for each target (all target-dependent files are isolated in a separated directory), and an adapter that wraps high-level functions (called by the radCASE generated code) and invokes the functions defined in the bridge interface.
The Api package is the most important module, as it separates the application logic of radCASE components and the target components. By using the bridge design pattern we have completely segregated these two parts: only the functions implementation in the board directory have to be modified/adapted if the hardware changes. Therefore, a radCASE project can be compiled on IAR Embedded Workbench for different targets with only few simple modifications to a call interface (a C file).

This approach provides an effective solution for monitoring and debugging of ESW. While the application is executed on the hardware target, the user can exploit the debugging suite of IAR Embedded Workbench and the monitoring environment of radCASE to check his application. Figure 4 shows an application example compiled and debugged with IAR Embedded Workbench (on the left side) and monitored with radCASE environment (on the right).

Concurrently, with the property editor of radCHECK the user defines a set of PSL assertions that the application must fulfill. As introduced before, radCHECK automatically generate executable checkers from the defined PSL assertions. Checkers are executable components that monitor the evolution of the ESW during dynamic ABV. The dynamic ABV is guided by stimuli automatically generated by the Test Pattern Generator. The checkers are executed in parallel with the ESW and monitor if the state of the application causes an assertion to falsify.
Figure 5 presents a snapshot of a verification phase executed in radCHECK with the ESW simulator provided by radCASE. The properties that are not satisfied are highlighted in red and reported with the corresponding processed sequence (a test case is composed by test sequences and each sequence contains a list of test vectors) whose simulation lead to the failure. The designer can use the resulting information, i.e., failed requirements, for refining the UML specifications incrementally and in an iterative fashion.

Thanks to the portability of this approach, the verification phase can be performed on top of an ESW simulator or on the application running on the target hardware. The same test cases and checkers can be exploited for performing PIL verification with C-SPY, to verify if the EWS behaves correctly also when it is executed on the hardware. C-SPY provides a wide set of functionalities for code inspections that allows accessing all system variables when the application is executed on the target hardware; this feature combined with the HAL structure provided by radCASE-generated code allows accessing all elements that should be monitored by the checkers or stimulated by the test case generator.

Figure 6 shows an overview of the integration workflow between C-SPY and radCHECK for performing PIL verification. The plug-in is the core mechanism that is implemented with the C-SPY SDK.

The radCHECK plug-in is executed when the user starts the debug of the application in C-SPY. It loads all debugging stubs automatically provided by radCHECK: the data map of radCASE elements which is used in C-SPY for read (in checker evaluation) or write (for providing the input stimuli) operations, the test cases database and the checkers C functions, which are compiled in a dll and dynamically linked to the radCHECK plug-in.
An overview of the implemented mechanism is presented in Figure 7. A radCASE application has a main function that is invoked by `main()`. This function (named `do_perm()`) contains an initialization phase and a loop to cyclically execute the functionality. At startup, the application is initialized and the plugin resets all checkers and reads all the application elements with their default value (physical inputs, outputs, parameters, internal variables).

The verification step is composed by two components that work on the main loop function of the application: one to inject the input values and one to evaluate checkers. The radCHECK plug-in exploits the breakpoints mechanism provided by the SDK to manage the target and the verification operations in a synchronized way. Inside the application main function (`do_perm()`) we call two empty functions, respectively `iarHookInit()` for synchronizing on the initialization phase and `iarHookVerify()` for managing the breakpoint on verification steps.

After initialization of the environment, the plug-in registers two breakpoints used to keep the target and the verification operations synchronized. When the execution starts, these breakpoints are hit and an event triggered by the C-SPY Debugger wakes up the breakpoint handler that executes the corresponding functions for initializing the verification environment (`init_environment()` and run verification steps (`valuate_checkers() and `run_test_vector()`). Then the breakpoints handler returns the execution to the target.

These functions exploit the API provided by the C-SPY Kernel API for accessing addressable memory in the target system (read and write functions of `DKLocation` class). In particular, the function `valuate_checkers()` reads data value from target memory and call the corresponding functions of checker.dll for checker evaluation and `run_test_vector()` injects test vectors by writing the data value on the target.

The plug-in also provides a GUI that shows the verification results: for each property that fails, the plugin GUI shows the corresponding test sequence and test vector that caused the violation. It shows the verification results in the violations list that is automatically updated each time that an input sequence is processed. The verification results can also be loaded in radCHECK for checking the corresponding properties. Figure 8 presents the radCHECK plugin after the execution in the C-SPY environment of a PIL verification session with the achieved violations list.

```c
main()
{
    do_stuff();
    do_perm();
}

do_perm()
{
    init_PERM();
    iarHookInit();
    for(;;)
    {
        execute_PERM();
        iarHookVerify();
    }
}
```
Benefits

This paper presents a solution for drastically reducing time-to-market and cost of industrial applications with product quality enhancement. The combined application of PIL with ABV allows performing rapid prototyping and verification of designs according to specifications. The proposed methodology allows verifying the actual ESW running on its final hardware and not only its model or debugging version.

The presented approach shows how these methodologies are no more limited to the academic world, but represent a powerful solution for industrial applications: we show how to quickly perform a thorough functional verification by using an affordable PIL methodology with off-the-shelf instruments. The automatic generation of test cases and checkers allows everyone developing embedded controllers to automatically perform functional verification. In particular, application fields that traditionally do not adopt specification conformance due to its complexity can achieve great benefits. Indeed, the guided formalization of functional specifications permits to early identify design issues, measure and certify results.

Contact

Cristina Marconcini
Product Manager
STM Products S.r.l.
Via Morgagni, 14 - 37135 - Verona – Italy
Phone: +39 045 585700 Fax: +39 045 585730
Web: http://www.radcase-ux.com
http://www.verificationsuite.com
E-mail: cristina.marconcini@stm-products.com