Proceedings,
The 2013 ASEE North Midwest Section Conference,
October 17-18, 2013, Fargo, North Dakota.

ASEE-NMWSC2013-0049

Incorporating On-going Verification & Validation Research to a Reliable Real-Time Embedded Systems Course

Nannan He
Department of Electrical, Computer Engineering and Technology
Minnesota State University, Mankato, MN 56001

Abstract

This paper presents the enhancements to a senior-level and graduate-level course, Reliable Real-time Embedded Systems, in terms of introducing advanced verification and validation (V&V) approaches. Traditionally, this course covers the topics of fundamental principles in real-time operation systems like scheduling with little emphasis on the design V&V. In order to equip students with the advanced knowledge of developing reliable embedded systems, this course is enhanced from two aspects. First, an on-going research project results on model-based testing and formal methods are incorporated into this course. Model-based testing is an important feature of Model-based Design (MBD) methodology, which can be used to check whether the design model satisfies the functional or nonfunctional requirements like timing. An automated formal method Model Checking, which is one of the most commonly used formal verification techniques, is exposed to students. It has been applied to deriving test cases from real-time design models. Second, students are introduced existing model checking tools like Uppaal and CBMC. Such new enhancements could greatly help students grasp the comprehensive knowledge of designing reliable embedded systems.

Introduction

Nowadays, embedded real-time computer system is playing an important role in many mission-critical ('mission' refers to the safety, reliability and security here) applications. Examples can be found in not only the controls of automotive, railways, aircraft and aerospace areas, but also the medical devices, "intelligent" home, factories and other sectors in our daily lives. Recent new processors and methods of processing, sensors, actuators, and communication infrastructure enable a truly pervasive computing environment. With the increasing popularity of real-time computer systems, their reliability is of paramount importance.

The normal execution of real-time systems must satisfy certain deadline constraints. For example, the response time of a system, which is the time between the presentation of a set of inputs and the realization of the required behavior, should be as short as 5 milliseconds. Thus, the logical correctness of such systems is based on the functional correctness (i.e., the correctness of outputs) as well as their timeliness. In mission-critical embedded real-time systems, an error could potentially cause disastrous results or severe economic consequences. Therefore, the correctness of embedded real-time systems must be rigorously validated and verified before they are put into operation.

Model-based design is emerging development methodology applied in designing modern embedded systems. It has an enormous potential for improving verification, testing and synthesis of embedded systems. Continuous model-based verification and validation (V&V) is an

important feature of this methodology. Technically speaking, validation checks that the system design specification meets the intended user requirements; verification checks that the system built after each development step from system design, implementation and system integration, satisfies the design specification. Testing and formal methods are two popular approaches in verification. In the context of model-based testing, traditional code-based metrics are no longer sufficient and accurate to estimate the efficiency of software testing. On the other hand, model checking is a kind of formal methods that is particularly well suited to integration in MBD paradigm. With little to no user interaction, a model checker exhaustively examines every possible combination of system input and state, and proves or disprove that a set of properties is true. A research project has been conducted in applying model checking to deriving test suite from real-time system design models, so as to improve the quality of model-based testing. This project is supported by the Faculty Research Fund in Minnesota State University at Mankato.

Technically speaking, the increasingly sophisticated functional and non-function demands, like timeliness of an embedded real-time system, require engineers to be equipped with the advanced V&V techniques. Model-based design methodology and model checking can be applied for the rigorous and cost-effective construction and verification of embedded systems. This paper presents our experience of introducing these techniques in the course “Real-time embedded systems”. First, an overview of this new course is described. Next, two advanced verification techniques - model-based testing and automated formal method model checking are presented. The work of enriching the course contents with the assist of an on-going research project, which has explored the synergistic results of combining both techniques, is then discussed. Two verification tools supporting formal checking of embedded real-time software systems are also exposed to students.

Overall Course Description

This Real-time Embedded Systems (RTES) course targets learning real-time systems design and applications from the practitioner’s point of view. It has three objectives: i) improve students’ awareness of real-time specifications in critical embedded systems; ii) allow engineering student to apply modern development tools to correctly designing and small-scale real-time systems including both software system and hardware; iii) enable student to develop reliable real-time applications to solve problems with specific timing requirements.

The topics covered in this course are grouped into seven top-level topics: 1) Fundamental concepts in embedded real-time systems; 2) Hardware for real-time systems; 3) Real-time operating systems; 4) Requirement engineering (formal or semi-formal methods); 5) Performance analysis; 6) Application issues (i.e., design for fault tolerance, verification and validation techniques); 7) Case studies.

The overall aim of this course is to equip students the knowledge of correctly designing real-time systems and developing real-time applications to solve engineering problems in practice. Twelve course learning outcomes are stem from this aim, classified into three core components:

1. To demonstrate the ability of correctly defining real-time systems
2. To demonstrate the ability of correctly designing real-time systems
3. To demonstrate the ability of basic real-time application development

Model-based Verification and Validation

Common development challenges exist in various embedded and real-time applications, ranging from the automotive industry, avionics, control systems and consumer electronics. They include the ever-increasing size and complexity of system, demands for reduced time to market, and rapid changes in technology. At the same time, it is not accepted that these mission-critical systems will contain a certain number of errors. Conventional quality assurance and testing techniques are too expensive and ineffective.

MBD is a promising approach that can contribute significantly to solving these challenges. It promotes the use of domain-specific models to graphically represent specifications and designs, with the goal of early identifying design flaws (via model simulator and model checking) to avoid costly late-stage design fixes. The Matlab/Simulink and LabVIEW are two popular system design environments supporting MBD. Furthermore, to encounter the increasing complexity of distributed embedded systems, some industrial research projects have investigated basic domain-independent technology for moving from stand-alone to distributed architectures. A recent European project DECOS [1] has developed the model-based tool-chain to accompany the dependable and cost-effective system development process from design to system integration deployment.

Model-based testing

Model-based testing is an important application of MBD for deriving test suite from design models, which promises better scalability and is applicable before the implementation phase. The test suite can be used to check whether the models meet user requirements or an implementation is correct with respect to the design models. Model-based testing allows test engineers to concentrate on the intellectual challenge of specifying and modeling the system behavior at a high level of abstraction, instead of focusing on manual test-case generation and manual test execution.

In the context of model-based testing, the question of suitable coverage metrics has to be reconsidered. Coverage is an indicator of the testing quality, and determines when testing is sufficient enough to stop. Traditional testing metrics, such as statement or branch coverage of the code, are no longer sufficient and accurate to estimate the efficiency of software testing. This is because the test cases are derived from the models instead of implementation source code. Mutation coverage, which is a kind of fault-based coverage, can be applied in model-based testing. With this coverage metric, the quality of the test suite is decided by two steps: 1) inject a set of mutations (faults or small modifications) into the model (the mutated model is called mutant), 2) measure which percentage of these mutations can be detected by exercising the test cases. Here, a mutant is *detected* if by executing the same inputs, the mutant and the original model can be observed to produce *different* outputs values or execution sequences. Generally speaking, mutation-based test coverage can be more accurate than code-based coverage metrics in the model-based testing setting. However, the generation of test-suite that achieves high mutation coverage is quite challenging.

Several approaches have been proposed to the mutation-based test case generation for Simulink models. For example, a formal technique - bounded model checking has been used to automatically compute test suites for given fault models [2]. Several optimizations, such as formal concept analysis [3] has been applied to making the approach practical for realistic Simulink programs and fault models, and to obtain accurate coverage measures. The problem of equivalent mutation detection in mutation-based testing has been investigated to avoid redundant and expensive search for the test cases which actually do not exist. [4]

Formal verification and model checking

Formal methods and testing are two fundamental verification approaches. Formal methods aim at proving the absence of errors with respect to specified properties; while testing attempts to show the presence of errors in the system. Formal verification conducts an exploration of all the possible behaviors based on formal models of the system and the formal specification of the intended requirements. The main advantage of this verification approach is the completeness it offers, which can eliminate the notion of inadequate coverage that conventional testing faces. This feature is in great favor in safety and reliability critical embedded applications. More importantly, some automated formal methods like model checking can be used to detect hard corner-case errors, which are very difficult to be detected by testing alone. With the significant advances in automated reasoning and computing capability of modern computers, formal verification is no longer of academic interest only. But, the limited scalability is still the major problem of most formal techniques in dealing with practical applications.

Introduced in 1981, model checking is one of the most commonly used formal verification techniques in industry. The inventors of model checking have been recognized by the 2007 ACM Turing award. This technique has been used to verify the specified property of the finite state model defined by the system, through an explicit or implicit enumeration of all the reachable states and behaviours. Model checking can be fully automatic without much expertise in formal methods. It differs from testing as it aims at an exhaustive exploration of the state space of the model, thereby providing a correctness guarantee that is rarely achieved by means of testing. More importantly, when the models under verification fail to satisfy a given specification, counterexamples can be generated, which illustrate the erroneous behaviours of the system design. This information can be very valuable for debugging. Model checking has been successfully applied to formally verifying the real-world hardware designs in industry. Many researchers have explored its applications in software and system verification.

Bounded model checking (BMC) is a variation of model checking which restricts the space exploration to execution traces up to a certain length *k*. It can provide a guarantee that the first *k* execution steps of the system are correct with respect to the specified properties. If the properties are not satisfied, BMC can automatically return a counterexample of the length at most *k*. The ability to report counterexamples is the essential feature that has been used to generate test cases, which will be discussed in the following subsection. With the recent dramatic advances in SAT-solvers, BMC is becoming increasingly popular. SAT-solvers can decide the satisfiability of a logic formula. A formula is satisfiable if an assignment exists under which the formula evaluates to TRUE. If no such assignments exist, the formula is unsatisfiable. BMC has two main advantages. First, the counterexamples with the shortest path can be found much faster compared

to symbolic model checking. Second, it needs much less space than model checking with other symbolic methods.

Research on model-based testing of embedded real-time software

The primary goal of our research is to significantly enhance the performance of V&V of embedded real-time software by means of the novel hybrid of static analysis and dynamic analysis. Another goal is to leverage the research outcomes and modern tools to enrich the contents of the senior and graduate courses with advanced V&V techniques.

Static analysis analyzes all possible program executions. This research explores model checking, which is one of static analysis techniques for formal verification. As discussed earlier, it exhaustively searches the entire state-space of a program for faults, and is therefore suitable for searching corner-behavior, and complex concurrency errors. The main advantage of this technique is that it produces a diagnostic counter-example in case the property is refuted. This counter-example can be very helpful to diagnose and correct the error. On the other hand, dynamic analysis runs/simulates a program and analyzes the properties of this running program. Dynamic analysis techniques have been used since the early seventies, initially mainly for performance analysis purpose. This research focuses on the techniques that analyze program executions to detect derivations from specific requirements, like testing. The basic idea is to receive events (i.e., test suite) from probes, run the system under analysis with these stimuli events and compare the observed actual events on-the-fly with the expected outcomes derived from the specification.

The combination of static and dynamic analysis techniques is an active research field which has not been fully explored yet [6]. An influential recent work combines test case generation and model checking to systematically execute all feasible program paths [7]. This approach – Directed Automated Random Testing (DART) – was proposed by Microsoft Research. In practice, directed search cannot explore all feasible program paths, but it can achieve much better coverage than random testing, so it can find more bugs. In this research project, we further explore this approach, and investigate the hybrid of static and dynamic analysis in the model-based testing of embedded real-time software. One way is to use the counter-example generated by bounded model checker for test suite generation. Formal methods are applied for two main purposes: 1) Generate and utilize formal models of embedded real-time software to accurately capture valid operation sequence of the software with respect to specifications. 2) Compute input test sequences based on the counter-examples derived from the formal models to achieve expected test coverage. These test sequences could be used as inputs stimuli for verifying the software implementation.

Two formal verification tools

In this course, students are exposed the basic concepts of the advanced V&V techniques, which have been investigated in this research project and have inspired our primary research direction. Moreover, in order to encourage students to participate in this research project and ease the application of these techniques to solve practical problems, two open source model checking tools whose status are active are also introduced to students.

UPPAAL

UPPAAL is a well-known and widely used model checking tool for real-time systems [8]. It is jointly developed by Aalborg University, Denmark, and Uppsala University, Sweden. With UPPAAL, the behavior of timed systems can be graphically modeled using the timed automata formalism extended with various modeling features. For example, concurrency and C-like functions and data structures are added to make it practically expressive and user-friendly. This tool consists of a graphical editor and simulator, and a model-checker. This checker performs an exhaustive symbolic analysis of the model and provides either a proof that the model satisfies a property, or a counter-example including a trace of actions and delays exemplifying how the property is violated. It has been applied successfully to a variety of industrial cases.

Recently, this tool was extended with new functions for test generation and controller synthesis. The ultimate goal of these updates is to enhance UPPAAL as an integrated tool suite for the MBD development lifecycle of embedded real-time systems

CBMC

CBMC is a bounded model checker for software verification. It can take as input a low-level ANSI-C program and, formally check safety properties like the correct usage of pointer constructs, array bounds and user-provided C assertions. Given a program *C*, a property *P* and a bound *k*, the verification includes three steps: i) unrolling k times all loops structures in C; then ii) translating the resulting program without loops and property into a Boolean formula in Conjunctive Normal Form (CNF); and finally (iii) giving the result to a SAT solver like MiniSat. If the SAT solver returns false, the property holds, otherwise the property does not hold within the bound *k*. This tool is developed and maintained by the Formal Verification Group from Oxford University, UK. This tool can be used to directly verify safety-critical properties in the implementation source code and indirectly verify high-level language models like Simulink after being transformed into C code.

Conclusions

The paper describes the introduction of advanced verification and validation techniques to a real-time embedded systems course. Since embedded real-time computer systems are becoming more popular their reliability is of paramount importance. The paper presents the Model-based design (MBD) methodology, especially model-based testing and model checking techniques that have great potential for the rigorous verification of embedded real-time systems. Two verification tools are also exposed to students to help them utilize these tools for conducting research and for solving practical problems at work.

References

1. The DECOS European project, http://www.decos.at
2. Brillout, A., He, N., Mazzucchi, M., Kroening, D., Purandare, M., Rummer, P., Weissenbacher, G. “Mutation-based test case generation for Simulink models”. In proceedings of *Formal Methods for Components and Objects* (FMCO). LNCS, vol. 6286, pp. 208-227. Springer, 2009.
3. He, N., Rummer, P., Kroening, D. “Test-case generation for embedded Simulink via formal concept analysis”. In the proceedings of *Design Automation Conference*, 2011.
4. A. F. Donaldson, N. He, D. Kroning, P. Rummer. “Tightening test coverage metrics: a case study in equivalence checking using k-induction”. In proceedings of *Formal Methods for Components and Objects* (FMCO). Springer, 2012.
5. E. M. Clarke, Jr., O. Grumberg and D. A. Peled, “Model Checking”, MIT Press, 1999, ISBN 0-262-03270-8.
6. Michael D. Ernst. Static and dynamic analysis: Synergy and duality. In WODA: ICSE Workshop on Dynamic Analysis, pp 24-27, 2003.
7. 8. Patrice Godefroid, Nils Klarlund, and Koushik Sen. *DART: directed automated random testing.* In Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005.
8. UPPAAL tool, http://www.uppaal.com/
9. E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In TACAS, pages 168-176. Springer, 2004.