Validation of Embedded Systems Using Formal Method Aided Simulation

Daniel Karlsson, Petru Eles, Zebo Peng
Dept. of Computer and Information Science, Linköpings universitet, Sweden
{danka, petel, zebpe}@ida.liu.se

Abstract

This paper proposes a validation approach, based on simulation, which addresses problems related to both state space explosion of formal methods and low coverage of informal methods. Formal methods, in particular model checking, are used to aid the simulation process in certain situations in order to boost coverage. The invocation frequency of the model checker is dynamically controlled by estimating certain parameters, based on statistics collected previously during the same validation session, in order to minimise verification time and at the same time achieve reasonable coverage. The approach has been demonstrated feasible by numerous experimental results.

1. Introduction

Designing IP blocks for embedded systems is a very complex task, and consequently very error-prone. At the same time, other designers using these blocks must be able to rely on their correctness. For this reason, the IP providers must thoroughly validate their blocks. This can be done either using formal methods or informally by simulation.

Both methods, in principle, compare a model of the design with a set of properties (assertions), and answers whether they are satisfied or not. With formal methods, this answer is mathematically proven to be guaranteed. However, using informal methods, this is not the case. The reliability of the result is indicated by a coverage metrics [1]. Unfortunately, formal methods such as, for example, model checking suffer from state space explosion. Although there exist methods to relieve this problem [2], [3], for very big systems simulation-based techniques are needed as a complement in order to obtain any results at all. Simulation techniques, however, are also very time consuming, especially if high degrees of coverage are required.

We propose a validation technique combining both simulation and model checking. The basis of the approach is simulation, but where model checking is added to reach uncovered parts of the state space, thereby enhancing coverage.

Combining formal and informal techniques is however not a new invention. One idea has been to use simulation as a way to generate an abstraction of the simulated model [4]. This abstraction is then model checked. The output of the model checker serves as an input to the simulator to guide the process to uncovered areas of the state space. This will create a new abstraction to model check. If no abstraction can be generated, it is concluded that the specification does not hold. As opposed to this approach, our technique does not iteratively model check a series of abstractions, but tries to maximise simulation coverage given a single model. There is hence a difference in emphasis. They speed up model checking using simulation, whereas we improve simulation coverage using model checking.

Another approach uses simulation to find a "promising" initial state for model checking [5]. In this way, parts of the state space, judged to be critical to the specification, are thoroughly examined, whereas other parts are only skimmed. The approach is characterised by a series of partial model checking runs where the initial states are obtained with simulation, as opposed to our approach which is coverage driven in the sense that model checking is used to enhance the coverage obtained by the simulation.

As opposed to existing simulation-based methods, our approach is able to handle continuous time both in the model under validation and in the assertions. We moreover are able to automatically generate the continuous time "monitors", which are used to survey the validation process, from assertions expressed in temporal logic. In addition to this, we propose a method to dynamically control the invocation frequency of the model checker, with the aim of minimising validation time while achieving reasonable coverage.

This paper continues in section 2 with an overview of the proposed simulation technique. Sections 3, 4, 5 and 6 present issues related to stimulus generation, assertion checking, coverage enhancement and stop criterion respectively. Section 7 provides experimental results and section 8 concludes the paper.

2. Validation strategy overview

In both formal and informal methods, it is common to assume that the model under validation (MUV) can be regarded as a transition system. The basic principle of such methods, in particular simulation based, is to fire one transition of the design at a time and check whether any assertions imposed on the model have been violated. At the same time, the MUV must be fed with inputs (stimuli) consistent with constraints imposed by the model on its environment. We consequently impose the following three assumptions:

- The MUV is modelled as a transition system. In this paper, we assume PRES+, a formalism based on Petri-nets, suitable for describing embedded systems [6].
- Assertions, expressed in temporal logics, stating important properties which the MUV must not violate, are provided. In this paper, CTL [7] is used for this purpose.
- Assumptions, expressed in temporal logics (e.g. CTL), stating the conditions under which the MUV shall function correctly according to its assertions, are provided.

The result of the validation is only valid to the extent expressed by the particular coverage metrics used (defined in section 2.1). Therefore, certain measures are normally taken to improve the quality of the results. This could involve finding corner cases which only rarely occur under normal conditions. Simulation-based techniques consequently consist of the following three parts: assertion checking, stimulus generation and coverage enhancement.

The proposed strategy consists of two phases, as indicated in Figure 1: simulation and coverage enhancement. In the simulation phase, transitions are repeatedly selected and fired at random, while checking that they do not violate any assertions (Line 3 to Line 5). This activity goes on until a certain stop criterion is reached (Line 2). The stop criterion used in this work is, in principle, when a certain number of transitions have fired without any coverage improvement. This will be further elaborated in section 6.

When the simulation phase has reached the stop criterion, the algorithm enters the second phase where it tries to further enhance coverage. An enhancement plan, consisting of a sequence of transitions, is then obtained and executed while at each step checking that no assertions are violated (Line 7 to Line 10).

The two phases, simulation and coverage enhancement, are iteratively executed until coverage is considered unable to be further enhanced (Line 1). This occurs when either 100% coverage has been obtained or when the uncovered aspects, with respect to the coverage metrics in use, have been targeted by the coverage enhancement phase at least once, but failed.

Stimulus generation is not explicitly visible in this algorithm, but is covered by the random selection of enabled transitions (Line 3) or as part of the coverage enhancement plan (Line 7). Subsequent sections will go into more details about the different parts of the overall strategy.

2.1. Coverage Metrics

Coverage is an important issue in simulation-based methods. It provides a measure of how successful a particular validation is. A combination of two coverage metrics is used throughout this paper: assertion coverage and transition coverage. The approach itself does, however, not impose any particular coverage metrics. It can easily be adjusted to any metrics of the designer's choice.

Assertion coverage is obtained by counting the number of assertions which have been *activated* (explained in section 2.2) during the validation process divided by the total number of assertions. Transition coverage is similarly defined as the number of fired distinct transitions divided by the total number of transitions. Combining these two metrics, the total coverage is computed by dividing the sum of activated assertions and fired transitions with the sum of the total number of assertions and transitions.

Assuming, that for a particular validation session 3 out of 5 transitions have been fired and 1 out of 1 assertions have been activated, the total coverage is computed as (3+1)/(5+1) = 67%.

2.2. Assertion Activation Sequence

In the definition of assertion coverage, the concept of assertions being activated was introduced. Intuitively, an assertion is, in principle, activated when all atomic propositions (innermost subformulas) have been satisfied at least once. In order to efficiently determine when an assertion is activated, one or more *activation sequences* are obtained for each assertion. A sequence consists of states (or characteristics of states, for efficiency) which are considered activating the corresponding assertion. In the context of PRES+ and Petri-nets, a state is a marking.

```
1: while coverage can be further enhanced do
2:    while not stop criterion reached do                    } Simulation
3:       select r randomly among the enabled transitions;
4:       fire r;
5:       Check that no assertion was violated;
6:
7:    obtain a coverage enhancement plan P;                  } Coverage Enhancement
8:    for each transition r∈P in order do
9:       fire r;
10:      Check that no assertion was violated;
```

Figure 1. Validation Strategy Overview

3. Stimulus generation

The task of stimulus generation is to provide the MUV with input consistent with the assumptions given by the model on its environment. In our strategy, the stimulus generator consists of another model, expressed in the same design representation as the MUV. The stimulus generator and the MUV are then connected to each other during simulation. For this reason, the stimulus generator is not explicitly visible in Figure 1. An enabled transition selected on Line 3 might belong to the MUV as well as to the stimulus generator. However, in order to promote better assertion coverage already in the simulation phase, transitions, which when fired lead to an increased assertion coverage, are selected with a higher probability.

We have previously developed an algorithm and the corresponding tool to automatically generate PRES+ models from ATCTL formulas [8]. ATCTL is an important subset of TCTL (Timed CTL).

4. Assertion checking

The objective of validation is to ensure that the MUV satisfies certain desired properties, called assertions. The part of the simulation process handling this crucial issue is the assertion checker, also called monitor.

In section 3, an algorithm generating a model from an A(T)CTL formula was mentioned. The same type of models can also be used for assertion checking as monitors.

Figure 2 illustrates the intuition behind assertion checking. Both the input given by the stimulus generator and the output from the MUV are fed into the assertion checker. The assertion checker then compares its observations with the monitor model generated from the assertion. For satisfiability, there must exist a sequence of transitions in the monitor leading to the same output as provided by the MUV, given the same input. This method works based on the fact that the monitor model captures all possible behaviours satisfying the assertion, including the behaviour of the MUV. The essence is to find out whether the MUV behaviour is indeed included in that of the monitor. This comparison is performed efficiently thanks to the regular structure of the monitor models.

5. Coverage enhancement

Experiments have shown that it is well worth the effort to take actions to enhance coverage. After the first simulation phase, total coverage is in average 82% (in all experiments made). After iteratively having applied our coverage enhancement technique, this number increased to 99%.

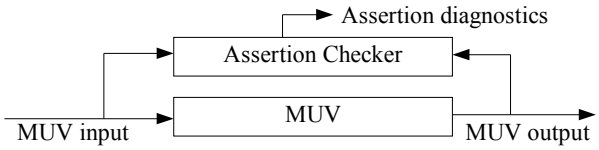

Figure 2. Assertion checking overview

The basic idea behind coverage enhancement is to find a sequence of transitions leading to the enabling of a transition which has not yet been fired, or leading to the activation of an assertion.

In order to effectively find a firing sequence leading to the activation of an unexercised part U, we apply model checking. Checking a property of the form **EF**U (there exists a future where U holds) provides a trace consisting of a sequence of transitions leading to a marking where U is exercised. This trace constitutes the coverage enhancement plan mentioned in section 2.

For enhancing transition coverage, the verified property is **EF** fired(t), where t is a transition which has not been fired previously. In the case of assertion coverage, the property is **EF** e, where e is the next event in the activation sequence of an assertion.

Once the enhancement plan is obtained, it can trivially be executed by firing the transitions in it in order. After each transition fired, the assertion checker must be invoked to check for any assertion violations. Enhancing coverage this way may make so far uncovered parts of the state space more accessible. These parts are then further explored in the subsequent simulation phase.

Sometimes, it could happen that the model checking runs out of resources, for instance time or memory. If this is the case or if an enhancement plan does not exist, coverage cannot be enhanced with respect to that particular transition or assertion. That transition or assertion will not be subject to any coverage enhancement again.

In order to perform model checking on PRES+ models, they have to be translated into the input language of the particular model checker used. We have discussed the problems related to such a translation into timed automata for the UPPAAL model checking environment [9] in [6].

6. Stop criterion

Line 2 in Figure 1 states that the simulation phase ends (and the coverage enhancement phase starts) when a certain stop criterion is satisfied. Section 2 stated that the stop criterion holds when a certain number of transitions are fired without any improvement of the coverage. This number is called *simulation length*. It can however be very difficult to statically determine the simulation length which will result in the shortest possible validation time. In this section, a dynamic heuristic approach, where the simulation length is determined at run-time, is presented.

During each iteration in the simulation phase, coverage keeps increasing until it reaches a certain threshold where it becomes saturated. At this moment, the strategy in Figure 1 concludes that it does not pay off to continue simulating. As a consequence, coverage enhancement is applied to reach uncovered parts of the state space, before continuing in the next simulation phase. The problem is to detect when a saturation level has been reached. Figure 3 presents a diagram illustrating how coverage grows during the course of one simulation phase.

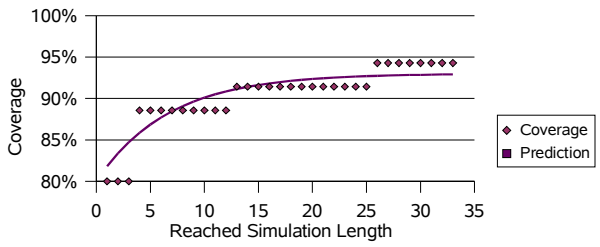

Figure 3. Simulation length vs Coverage

During the simulation phase, gradually increasing simulation lengths will be reached. In the very beginning, transitions are repeatedly fired, as stated in Figure 1. In one moment, one transition has been fired without increasing coverage. This moment is defined as simulation length 1. The coverage obtained at this point (80%) is then marked in Figure 3. When two transitions have been fired in a row without increasing the coverage, simulation length 2 is considered reached and its coverage is also marked in the diagram. The other points were obtained by continuing this process until the final, predetermined, simulation length was reached. The saturation level in this example is 94% and was reached for simulation length 34.

If the coverage for a given simulation length $cov(\sigma)$, average transition firing time t_{fir} and average coverage enhancement time t_{ver} can be predicted, it is possible to estimate the time function and thereby the optimal simulation length.

Unfortunately, these parameters are not known beforehand. Besides, it is too expensive in terms of time to obtain these data prior to actual validation. For these reasons, they have to be obtained during the validation process itself. As a consequence, the predicted simulation length is adjusted during the process as more and more data is collected and better and better estimations can be made. Initially, the parameters are assigned values conformant with an average case. The following paragraphs briefly describe the parameters necessary to estimate the optimal simulation length.

Coverage has been shown by numerous experiments to behave according to the exponential function $cov(\sigma)$ in eq 1, with respect to the simulation length σ. C and D are parameters determining the steepness of the curve, and E is the parameter which denotes the saturation level.

$$\text{(eq 1)} \qquad cov(\sigma) = E - C\,\mathrm{e}^{-D\sigma}$$

By repeatedly applying the linear algebraic *Least Square Method* for different values of D, all parameters can be efficiently obtained. The curve marked "Prediction" in Figure 3 reflects the $cov(\sigma)$ function obtained by applying this method on the coverage points in the same figure.

The transition firing time, t_{fir}, reflects the average time it takes to perform one instance of the following three steps of Figure 1: choosing one enabled transition (Line 3), the actual firing of the transition (Line 4) and assertion checking (Line 5). These steps are performed at every iteration in one simulation phase (Line 2).

The average coverage enhancement time, t_{ver}, reflects the time it takes to perform one instance of the following three steps of Figure 1: obtaining a coverage enhancement plan (Line 7), executing the enhancement plan (Lines 8-9) and asserting the plan (Line 10).

It can be shown in both theory and practice that the total verification time conforms to eq 2.

$$\text{(eq 2)} \qquad (1 - E + C\,\mathrm{e}^{-D\sigma})|T \cup A|\,t_{ver} + t_{fir}\,\sigma$$

Given eq 2, it is straight-forward to analytically compute the optimal simulation length, eq 3.

$$\text{(eq 3)} \qquad \sigma = \frac{\ln \dfrac{t_{fir}}{CD|T \cup A|\,t_{ver}}}{-D}$$

7. Experimental results

This validation approach has been tested on a variety of models and assertions. Part of the results are presented in Table 1. The table compares the time needed and coverage obtained using the static and dynamic stop criteria respectively for several example of different sizes. The values given are the average values of several runs on the same model and assertions. The complexity of the models are given in terms of number of transitions. Although this number is generally not enough to characterise the complexity of a model, it still provides a hint about the size of the model and is used in lack of a more accurate, but still concise, metrics. Examles 1 through 27 consist of, in principle, randomly generated models. Examples 28 through 33, however, model a control unit for a mobile telephone, traffic light controller and a multiplier respectively.

It should be emphasised that the simulation length used for the static stop criterion was obtained by empirically evaluating several different values, finally choosing the one giving the shortest validation time for comparison. The performance given by this method can consequently not be achieved in practice, and only serves as a reference.

As can be seen, in most cases using the dynamic stop criterion results in validation times close to those for the static stop criterion. There exist however cases where there is a big difference. These cases fall into two categories:

- The models differ from the average case on which the estimation of the initial simulation length is based.
- There is a difference in coverage. If one method did not reach as high coverage as the other, that method had more time-outs in its coverage enhancement phase, thus adding to total time.

Table 1. Experimental results

		Time (s)		Time Diff.	Coverage (%)	
Example	Trans.	Dynamic	Static	(%)	Dynamic	Static
1	28	22.67	24.54	-7.62	100	100
2	28	51.75	38.30	35.12	100	100
3	35	42.40	39.74	6.69	100	100
4	35	62.55	60.78	2.91	100	100
5	42	55.38	58.77	-5.77	100	100
6	42	82.67	78.71	5.03	100	100
7	49	70.45	80.42	-12.40	100	100
8	49	101.64	105.16	-3.35	100	100
9	56	93.60	378.28	-75.26	100	99
10	56	143.36	120.73	18.74	100	100
11	63	137.14	289.89	-52.69	100	99
12	63	157.23	161.75	-2.79	100	100
13	70	298.81	151.43	97.33	99.5	100
14	70	345.21	196.22	75.93	99.5	100
15	7	6.29	4.43	41.99	100	100
16	14	261.98	399.91	-34.49	95	95
17	14	270.23	277.01	-2.45	95	95
18	21	627.27	550.78	13.89	95	94
19	21	891.39	821.83	8.46	89	90
20	7	7.57	4.66	62.45	100	100
21	7	16.41	10.49	56.43	100	100
22	14	253.19	240.65	5.21	98	95
23	14	265.08	388.45	-31.76	93	95
24	30	15.27	10.42	46.55	100	100
25	75	119.37	93.06	28.27	100	100
26	150	564.54	504.37	11.93	100	100
27	225	1768.35	1604.84	10.19	100	100
28	31	1043.97	935.68	11.57	98	99
29	31	599.19	417.30	43.59	95	100
30	36	216.41	157.01	37.83	100	100
31	36	279.46	250.10	11.74	100	100
32	8	13.12	10.21	28.50	100	100
33	8	330.47	316.21	4.51	100	100

It can be deduced from the figures that, in average, the dynamic approach is 15% slower than using the static stop criterion. However, in 30% of the cases, the dynamic approach was actually faster. This situation is possible, since choosing the simulation length for the static stop criterion is not a very accurate process. It could happen that the dynamic stop criterion finds a simulation length closer to the actual optimum. The loss in coverage is, on the other hand, very small. In average, coverage is 0.12% lower using the dynamic approach.

Although the dynamic approach performs slightly worse on both aspects, it should be remembered that it in practice is impossible to reach the values listed in the table with the static approach. As mentioned previously, the values were obtained by trying several values for the static simulation length, thus validating the system multiple times. It cannot be known in advance which simulation length results in the shortest validation time.

All models and all assertions have also been validated with pure model checking. In two cases the model checker did not support the type of assertions checked, so no result could be obtained for them. For a few examples with few transitions, the model checker found a solution, but in one order of magnitude longer time than the proposed approach. In all other cases the model checker ran out of memory (2GB) before any result could be delivered.

The models were also verified using pure simulation. The simulation went on for as long time as was used by the dynamic stop criterion. The total coverage obtained by the simulation process after this time, was always less than or, in some cases, equal to our proposed mixed approach.

8. Conclusions

This paper has presented a validation approach, based on simulation, where model checking is applied to increase coverage. The invokation of the model checker is dynamically controlled in order to minimise validation time.

9. References

[1] A. Piziali, *Functional Verification Coverage Measurement and Analysis*, Kluwer Academic Publishers, 2004

[2] F. Wang, A. Mok, E. A. Emerson, "Symbolic model-checking for distributed real-time systems", *Lecture Notes in Computer Science*, Vol. 670, 1993

[3] C. Daws, S. Yovine, "Reducing the number of clock variables of timed automata", in , 1996, pp. 73-81

[4] S. Tasiran, Y. Yu, B. Batson, "Linking Simulation with Formal Verification at a Higher Level", *IEEE Design & Test of Computers*, Vol. 21:6, Nov-Dec 2004

[5] Synopsys whitepaper, "Hybrid RTL Formal Verification Ensures Early Detection of Corner-Case Bugs", 2003

[6] L.A. Cortés, P. Eles, Z. Peng, "Verification of Embedded Systems using a Petri Net based Representation", in *Proc. ISSS*, 2000, pp. 149-155

[7] E.M. Clarke, O. Grumberg, D.A. Peled, *Model Checking*, MIT Press, 1999

[8] D. Karlsson, "Towards Formal Verification in a Component-based Reuse Methodology", Licentiate Thesis No 1058, 2003, http://www.ep.liu.se/lic/science_technology/10/58/

[9] UPPAAL hompage, `http://www.uppaal.com/`