MODELING AND FORMAL VERIFICATION OF COUNTING HEADS FOR RAILWAYS [1]

Sebastian Kinder and Rolf Drechsler
Universität Bremen
Address: Bibliothekstraße 1, D-28359 Bremen, Germany
Phone: +49-421-218-8259, Fax: +49-421-218-7385 ,
E-Mail: {kinder,drechsle}@informatik.uni-bremen.de

Abstract: The demand for safety for electronic systems, especially safety critical systems, is high. Nowadays such systems are tested and simulated with a manually created set of test cases. But testing cannot reach a complete coverage for complex designs. Hence, we present a verification flow for Counting Heads for railways which are used by many electronic railway interlocking systems from SIEMENS. Our approach is based on SystemC, a powerful system level description language. Thereby, efficient modeling and simulation-based validation of railway systems becomes possible. The presented flow allows also for formal verification.

Keywords: railway control systems, domain-specific language, formal methods, bounded model checking, SystemC

1. INTRODUCTION

Today's electronic systems become more and more complex. They are applied in many areas of our personal lives, e.g. cellular phones or entertainment electronics, and it is nearly impossible to imagine a modern life without them. Failures of these devices would usually result in minor problems for the users. But complex electronic systems are also used for in safety critical applications, e.g. like those used in medical equipment, avionics and electronic railway interlocking systems. Especially, in these sectors the demand for safety is exceptionally high because human life may depend on the faultless functionality of such devices.

Nowadays railway systems are designed and tested in a conventional way, i.e. the systems are simulated with a manually created test bench. This has the advantage that the designers have a considerable expertise with this kind of work, but there is still a lot of potential for human failure. Furthermore, testing is very cost-intensive and can never reach complete coverage for large designs. Hence, an integrated design flow for railway systems is needed, which allows for efficient modeling, validation, simulation-based verification as well as formal verification. Therefor, a well accepted approach is the system level description language SystemC, which is known from the hardware design domain. SystemC supports the designer with a suitable methodology for efficient modeling and validation. With an integrated simulation kernel, it facilitates a fast and efficient validation and simulation-based verification. Finally, formal verification can be applied, too.

In this paper we present a design flow for a railway specific application based on

[1] This work was supported in part by the *Rail Automation Graduate School (RA:GS!)* of Siemens Transportation Systems in Braunschweig, Germany.

SystemC. We show the modeling of *Counting Heads* (CHs) for railways (Siemens AG, 2003*a*), which are used to determine whether a specified *Track Vacancy Detection Section* (TVDS) is clear or occupied. Especially for electronic railway interlocking systems as constructed by SIEMENS, which determine automatically whether a TVDS is clear or occupied, the correct function of CHs is crucial: If they fail to work properly, a TVDS would either be falsely indicated as occupied – resulting in a deterioration of availability and reliability – or falsely indicated as clear – possibly introducing a safety hazard. The proof of correctness helps to avoid such situations. Therefore, we present first steps towards a complete formal verification of CHs using *Bounded Model Checking* (BMC).

This paper is structured as follows: In Section 2 we give a brief introduction into the system description language SystemC and into the property specification language PSL. Furthermore we explain the fundamental ideas of BMC. We motivate the usage of CHs in railway systems in Section 3. Afterwards, we explain how they are modelled in SystemC. In Section 4 the simulation-based verification of the CHs is shown. Formal verification results using BMC and inductive reasoning are presented in Section 5. The final section concludes this paper and gives an outlook to future work.

2. Preliminaries

In this section we give some details regarding the preliminaries of modeling and formal verification. First of all we introduce some details of the modeling language SystemC and the property specification language PSL. In the second part we explain bounded model checking..

2.1 SystemC

SystemC (Grötker *et al.*, 2002) is a C++ class library that can be used to create designs from the hardware domain ranging from cycle-accurate architectures to system-level models. Some of the basic concepts of SystemC are given in the following:

- Modules: Modules are the basic building blocks of a system. They encapsulate parts of the design. A module is realized as a C++ class and may contain ports, local data, concurrent processes and submodules. Thus, a hierarchical design description is possible.

- Processes: Processes are used to describe the functionality of modules. They can be sensitive to input signals, clock signals or more complex events. SystemC provides three different process abstractions. Concurrency is modelled by processes.

- Signals: Signals represent physical connections and connect modules through their ports. Signals transport data without information about the direction. The assignment of values to signals takes place according to the principle of deferred assignments known from hardware description languages, e.g. VHDL or Verilog.

- Data types: In addition to all C++ data types SystemC supports data types for multiple design domains, ranging from four valued logic via fixed-point to integer with unlimited length.

The concepts provided by SystemC are sufficient to model hardware designs. But on higher levels of abstraction, such as the transaction level, complex communication protocols are required, e.g. buffered queues. Therefore, the concept of channels and interfaces has been implemented in SystemC.

Since SystemC is a C++ class library the design can be compiled with a standard C++ compiler to produce an executable specification. The current state of the system can be propagated by using C++ routines, e.g. `cout`, or waveforms using the waveform tracing provided by SystemC.

2.2 Property Specification Language

Properties of a system are often described using temporal expressions in hardware design. The system can be checked with these properties using formal techniques. Describing temporal properties for verification can be done in many different ways, since there exist several languages and temporal logics. As property specification language we use a subset of the widely known industrial standard PSL (Accellera, 2004). A PSL-property consists of two parts: a list of assumptions and a list of commitments. Assumptions and commitments have the form:

```
    next[a]    (expression)
or  next_a[a,b](expression)
or  next_e[a,b](expression)
```

where a and b are time points. If all assumptions hold, all commitments in the proof part have to hold as well. All used time points have to be greater or equal to 0. Since a and b are finite, a property argues only over a finite time interval, which is called *observation window*. Temporal dependencies are expressed by using the keywords `next_a` and `next_e`, whereas `next_a` states that the expression has to hold all the time in the interval and with `next_e` the expression has to hold at least once in the specified interval. Also a set of advanced operators and constructs is provided to allow for expressing complex constraints more easily.

In the way we formulate properties, they state that whenever some signals have a given value, some other (or the same) signals assume specified values. Of course, it is also possible to describe symbolic relations of signals. Furthermore the property language allows to argue over time intervals, e.g. that a signal has to be stable in a specified interval.

2.3 Bounded Model Checking

In *model checking* (also called *property checking*) properties for a given system are formulated in a dedicated "verification language". It is then formally proven whether these properties hold under all input and state assignments for the given assumptions. While "classical" CTL-based model checking (Burch *et al.*, 1990) can only be applied to medium sized designs, approaches based on *Bounded Model Checking* (BMC) as discussed in (Biere *et al.*, 1999) give very good results when used for complete blocks. In BMC the properties are only considered over a finite interval. BMC has originally been proposed for circuit verification and in this context considering a finite number of steps is reasonable.

The model has to be unrolled as often as the time interval of the property is long. The unrolled model and the property are translated into a boolean formula. The formula is solved by a SAT-solver and if it is satisfiable, a counter example has been found, disproving the validity of the property. If the SAT instance is unsatisfiable, the property holds. Since there is no restriction to reachable states during the proof of the corresponding SAT instance, a counter-example may start from an unreachable state. Usually, if such a case occurs these states are excluded by additional assumptions. For BMC as used here, it is not necessary to determine the state space diameter of the underlying model.

3. Modeling of the Counting Heads

Counting Heads (CHs) (Siemens AG, 2003*a*) are needed to determine whether a railway track section is vacant or occupied. This is essential for electronic railway interlocking systems in order to position points into the correct direction for the next train. CHs are used in a lot of interlocking systems all over the world. A CH failing to work properly could result in a collision of railways and endanger the life of passengers.

In Figure 1 (Siemens AG, 2003*b*) a small network of railtracks is shown, which could be operated by an electronic railway interlocking system. Every *Track Vacancy Detection Section* (TVDS) is defined by at least two CHs, one at the beginning and one at the end of such a track section. The output of the CH is transmitted to an evaluation computer. The evaluation computer interprets the signals, compares the number of

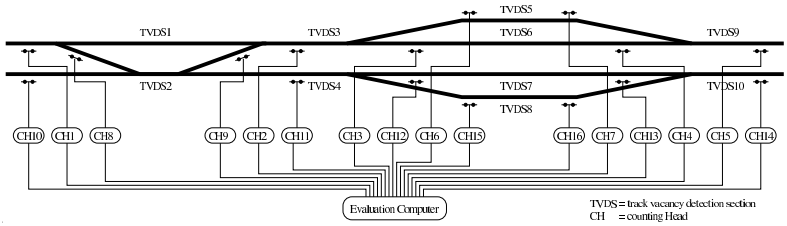
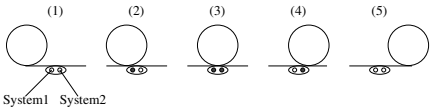
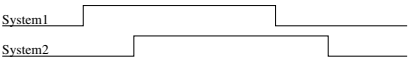


Fig. 1. Track Vacancy Detection Section



(a) Impact on Sensor Systems



(b) Waveform

Fig. 2. Double Wheel Detection

axles which entered and left a TVDS. The computer issues clear or occupied indications and monitors the clear/occupied state of the TVDS. The functionality of a CH is explained in the following section.

3.1 Counting Head

To operate, a CH is connected to a double wheel detector and an evaluation computer. The Figures 2 and 3 are taken from (Siemens AG, 2003*a*).

3.1.1 Counting with Double Wheel Detectors

A double wheel detector is mounted on one of the rails and detects passing wheels of the vehicles. A double wheel detector consists of two sensor systems, which are triggered when an axle crosses. The impacts on both sensor systems occur with a delay, which indicates the direction of the crossing axle as shown in the following example.

Example 1. *The triggering of the sensor system is shown in Figure 2(a). The signal waveform of both sensor systems is given in Figure 2(b). These figures correspond to an axle which crosses the double wheel detector regularly.*

If the space between wheel and sensor system is big enough, both sensor systems are unaffected (state (1)) as can be seen in Figure 2(a). The wheel approaches from the left side and the left sensor is affected first (state (2)). When the wheel is positioned

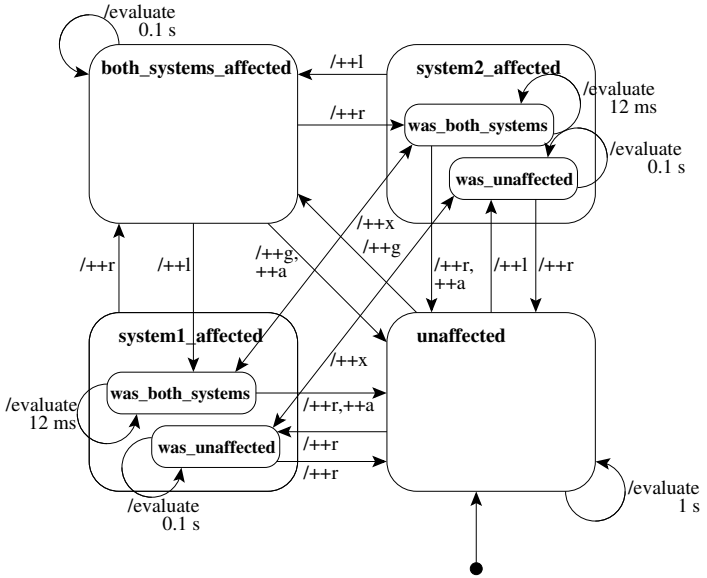


Fig. 3. FSM of the Counting Head

right above the double wheel detector, both sensor systems are affected (state (3)). The axle moves on and the left sensor is not affected anymore (state (4)). Finally, the axle left the double wheel detector and both systems return to their unaffected states (state (5)). If an axle crosses the detector from left to right, the states are always traversed in order (1)-(2)-(3)-(4)-(5) as described in Example 1. If it crosses from right to left, the order is (5)-(4)-(3)-(2)-(1). Different sequences are also possible, e.g. (1)-(2)-(3)-(2)-(1). If this sequence occurs, the evaluation computer recognizes the oscillation of axles.

3.1.2 Axle Counting Procedure

The basic idea for the axle counting procedure is as follows: There are five counters (`l`, `r`, `g`, `x`, `a`). They count a defined type of state transitions for a single axle, while traversing the *Finite State Machine* (FSM) given in Figure 3, e.g. every clockwise transition is counted by the counter `r`. All counters are real integer values. Times are ranging from 12 ms to 1 s. They are not given in form of discrete clock cycles as usually known in hardware design. For a more detailed description see (Siemens AG, 2003*a*).

Every time the FSM enters state `unaffected`, these counters are added to a second set of counters, which are counting the whole group of axles. Afterwards, the counters for a single axle are set to zero. If the FSM was idle in one state long enough, the CH evaluates the counters for the whole group and writes the corresponding values to its attributes. The CH has some attributes which are visible from the outside:

1. *Number of axles* is the signed sum of all axles which crossed the double wheel detector. The sign indicates the direction of the axles.
2. *Number of errors* is incremented every time an error occurred in the CH,

e.g. an undefined counting impulse.

3. *Time the CH was not affected*, states for how long no axle crossed the double wheel detector.

4. *Counter control token* is set, if the CH raises suspicions of malfunctions.

5. *Several failure tokens*, which indicate if a failure occurred, depending on the type of the CH.

The FSM consists of four states. They correspond to the four possible impact combinations on both sensor systems. The states, in which one sensor system is exclusively affected, are divided into the two substates `was_both_systems` and `was_unaffected`. Depending on whether the counting head was in state `unaffected` or in state `both_systems_affected` before entering this state.

The states of the FSM are traversed in a circle either clockwise or counterclockwise, depending on the direction in which the axle crosses the detector. For every state transition clockwise the counter `r` is incremented (see Figure 3) and for every transition counterclockwise `l` is incremented. If a transition from state `unaffected` to state `both_systems_affected` or vice versa occurs, the counter `g` is incremented. For transitions between state `system1_affected` and state `system2_affected` the counter `x` is incremented, analogously. The counters `x` and `g` are necessary to determine if an axle crossed the detector regularly or not. Without interferences either the counter `r` or the counter `l` has got a high value and the `x` and `g` stay equal to zero.

At some of the states in Figure 3 there is a time annotated. This time indicates for how long the counting head has to stay in this state, before the counter values are evaluated. In the evaluation phase the counting head determines how many axles have passed the detector. The direction of the axles is calculated by the values of the counters `l` and `r`. If `l` is greater than `r` the number of axles is decremented by `a`. If `r` is greater than `l` the number of axles is incremented by `a`. If these counters are equal, no axle is counted, because the axles oscillate over the detector. Thus, the overall number of axles is calculated. This number is signed to indicate the direction of the axles. This is only done, if the counters `x` and `g` are equal to zero or at least within the range of tolerance. Accepted values for `x` and `g` are calculated with several parameter, which are defined for every CH type. If these counters are not within the range of tolerance, the CH counts the errors and checks if the counter control token has to be set. If this token is set, the evaluation computer issues a permanent occupied indication. There is a big variety of error patterns, but due to the page limitations they are not enumerated here. A short evaluation of counter values can be seen in Example 2.

Example 2. *A train with 40 axles passed the counting heads. Assuming every axle passed regularly. The train crossed from right to left, i.e. the first sensor system to be affected is system two. After the train passed completely, counter* $a = 40$*, counter* $l = 160$ *and the other counters are equal to zero. After one second in state* `unaffected` *the CH evaluates the counter. The absolute number of axles is* 40*. Since* l *is greater than* r*, the CH transmits* -40 *as the number of axles to the evaluation computer.*

3.2 SystemC Model

SystemC is well suited to model systems like the one at hand. To model this system, it is advisable to partition the design into a module for the finite state machine and a module for the timeout control. A sketch is given in Figure 4.

As can be seen the design has three input signals, the inputs `system1` and `system2` indicate which sensor system of the double wheel detector is affected and

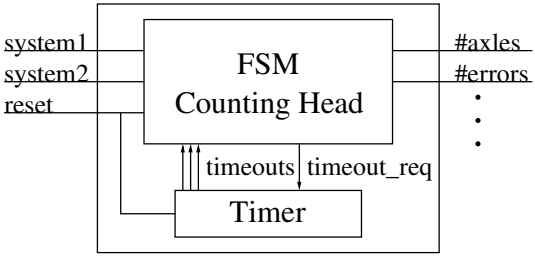


Fig. 4. Abstracted Model of a Counting Head

which is not. If the input `reset` is set, the system is reset into the initial state. The inputs are Boolean values.

As mentioned above, the system consists of two modules, which are interconnected to each other by three timeout signals and one timeout request signal. The timeout request to start the timer is issued every time the module "FSM-Counting Head" reaches another state of the FSM. The timer module activates the timeout signals, when the corresponding timeout is reached. The internal signals are Boolean values, too.

The number of outputs depends on the type of the CH. But all CHs have at least the following outputs:

- Number of axles, integer value
- Number of errors, integer value
- Time the system has been unaffected (idle time), unsigned integer value
- Counter control token, Boolean value

Additionally, there can be several more outputs, called failure tokens, depending on the specific needs for a railroad track.

For a better understanding of the model, we give some implementation details for state `unaffected` and state `system1_affected` of the FSM in Figure 5. The other two states are left out, because they are similar to the presented ones and they are implemented analogously.

State `unaffected` is handled in lines 2 – 23. While traversing the FSM the state transitions are counted for a single axle. In Figure 5 the single axle counters are denoted with the suffix `_s`. When the system enters this state, all counters for a single axle are added to the counters for the whole group of axles (denoted with the suffix `_g` in the figure). This can be seen from line 3 to line 7. Afterwards the counters for a single axle are set to zero. In line 8 the `old_state` is set to the current state, so that the next state can determine in which direction the FSM is traversed. The possible impact combinations on both sensor systems are checked in lines 9 – 21. The corresponding counters are incremented and the new state is set. Additionally, the `timeout_req` is issued for the next state. The last action in this state is the check if the time, the system is allowed to stay in this state, has expired (line 22). If the time is expired, the evaluation phase begins.

State `system1_affected` is entered, when sensor system 1 is affected and sensor system 2 is not affected. The system stays in this state as long as the impacts on the sensor systems do not change. If the impact on the sensor systems change, it is distinguished between the three possible combinations (lines 25 – 41). The counters are incremented and the new state is set accordingly. It is determined if an axle has passed successfully in lines 34 and 35. In this case the system is about to enter

```
1  switch(state){
2  case unaffected:
3    a_g += a_s; a_s = 0;
4    r_g += r_s; r_s = 0;
5    l_g += l_s; l_s = 0;
6    x_g += x_s; x_s = 0;
7    g_g += g_s; g_s = 0;
8    old_state = unaffected;
9    if(system1 == 1 && system2 == 1){
10     x_s++;
11     state = both_systems_affected;
12     timeout_req = 1;
13   }else if(system1 == 1){
14     r_s++;
15     state = system1_affected;
16     timeout_req = 1;
17   }else if(system2 == 1){
18     l_s++;
19     state = system2_affected;
20     timeout_req = 1;
21   }
22   if(timeout(1sec)) evaluate = 1;
23   break;
24 case system1_affected:
25   if(system1 == 1 && system2 == 1){
26     r_s++;
27     state = both_systems_affected;
28     timeout_req = 1;
29   }
30   if(system1 == 0 && system2 == 0){
31     l_s++;
32     state = unaffected;
33     timeout_req = 1;
34     if(old_state == was_both_systems)
35       a_s++;
36   }
37   if(system2 == 1 && system1 == 0){
38     x_s++;
39     state = system2_affected;
40     timeout_req = 1;
41   }
42   switch(old_state)
43   case was_both_systems:
44     if(timeout(0.1sec)) evaluate = 1;
45     break;
46   case was_unaffected:
47     if(timeout(12msec)) evaluate = 1;
48     break;
49   break;
50 case both_affected:
51   ...
52 case system2_affected:
53   ...
54 }
```

Fig. 5. Implementation of the CH-FSM

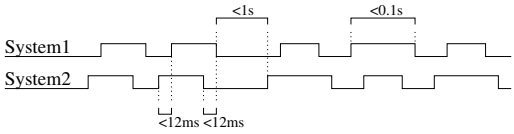


Fig. 6. Stimuli Waveform

state `unaffected` and the `old_state` was state `was_both_systems`. This means that the FSM has been traversed completely from state `unaffected` to state `unaffected`. After that the two substates (as can be seen in Figure 3) have to be distinguished to determine if a timeout for this substate occurred. In substate `was_unaffected` the time until the timeout is triggered is 0.1 s. In substate `was_both_systems` it is 12 ms.

If a timeout is triggered in any state, the system starts the evaluation phase. In this phase the system determines how many axles crossed the double wheel detector, how many errors occurred and if the control token has to be set. During this phase all outputs, except `idle time`, are written. The total `idle time` is written continuously to the corresponding output, if the CH was in state `unaffected` for more than 12 ms. A failure case, which is detected in the evaluation phase, is shown in Example 3.

Example 3. *The counters x and g are equal to zero for a regular crossing of a group of axles. They are only incremented in a case of failure. A CH-error is triggered, if the sum of these counters exceeds a specified value:* $x + g > \lfloor a/gx_{max} \rfloor$. *The variables x, g, and a are counters and* $gx_{max} \geq 4$ *is a parameter, which is defined for every CH type, separately.*

This equation denotes that for $gx_{max} = 4$ *one failure per four axles is tolerated. If* $a < 4$*, no failure at all is accepted.*

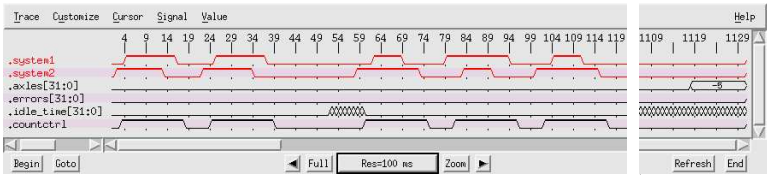
4. Simulation-based Verification

In this section we show the simulation-based verification of the SystemC model of the CH.

Since SystemC is a C++ class library, the modelled system can be compiled to an executable specification. This specification can be simulated with the integrated SystemC simulation kernel. To simulate a design, it is necessary to stimulate its inputs. This is done by an additional module, the stimuli generator. The generator has exactly the same number of outputs as the design's number of inputs. Each input has to be connected to the corresponding output of the stimuli generator. The stimuli generator has three Boolean outputs to stimulate each one of the inputs `system1`, `system2` and `reset`.

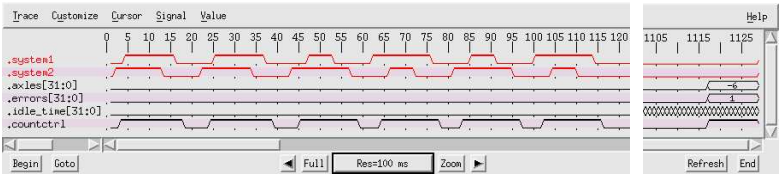
The test cases presented in the following are taken from an official test report for the axle counting procedure. In Figure 6 the stimuli for the inputs are given. The delay between two rising and falling edges on the two sensor systems of the double wheel detector is not allowed to be greater than 12 ms. No high edge may last 0.1 s or longer. Both sensor systems may not be unaffected for 1 s or more at the same time. If any of these times are exceeded, the train isconsidered standing and thus, the evaluation phase of the CH begins.

The stimuli from Figure 6 are produced by a stimuli generator. This waveform can be found again in the resulting waveform of the test run in Figure 7(a). The first two axles crossed the detector in a regular



(a) Part 1 (b) Part 2

Fig. 7. Waveform of a Simulation with a Tolerated Number of Interferences



(a) Part 1 (b) Part 2

Fig. 8. Waveform of a Simulation with an Untolerated Number of Interferences

way. Since sensor system 2 (`system2`) is affected first the FSM of the CH is traversed counterclockwise, i.e. counter `l` would be the only counter to be incremented. But for the last three axles the FSM is traversed from state `unaffected` to state `both_systems_affected` via state `system2 affected` or `system1 affected` and on the same way back. Thus, not only counter `l` is incremented, but counter `r` is incremented, too. After the five axles have passed the detector in the way given in Figure 7(a) the counter values are: $a = 5$, $l = 14$ and $r = 6$. Whether this is within the range of tolerance, is calculated by the equation:

$$1 + a/rl_{max},$$

where $rl_{max} \geq 1$ is a parameter, which can be defined for every CH. In the case at hand rl_{max} equals to one. The maximum number of state transitions contrary to the main direction is 6. Therefore, a counter value $r = 6$ is valid and the five axles are counted correctly as can be seen in Figure 7(b).

The second test run is shown in Figure 8. This run is similar to the first one. But instead of three irregular passing axles there are four of them (Figure 8(a)). Thus, the counter values after these six axles have passed are: $a = 6$, $l = 16$ and $r = 8$. But the maximum number of accepted state transitions contrary to the main direction is 7. Corresponding to the equation $1 + a/rl_{max}$ the system detects an undefined counting impulse and switches into a secure state by setting the counter control token (`countctrl`), as presented in Figure 8(b). If the counter control token is set, the evaluation computer issues a permanent occupied indication for the corresponding TVDS. The TVDS can only be cleared by a reset of the CH.

```
1  property cnt_ctrl_final =
2    always (
3      1
4    ) -> (
5      next[1](
6        ((evaluate == 1) ?
7          (((ss > s_max)
8            || (cc > c_max)) ?
9            cnt_ctrl_token == 1
10           : 1)
11         :1)
12     )
13   );
```

Fig. 9. PSL-property: `cnt_ctrl_token`

These test cases show the robustness of this counting procedure against interferences. But, they also show, that if these interferences exceed a specified threshold the system switches over into a safe state.

Simulation-based verification produces results fast. But as mentioned in the introduction, simulation-based approaches cannot reach complete coverage. Thus, formal proof-techniques are required to verify correctness. In the next section such an approach using BMC and inductive reasoning is presented.

5. Verification

The SystemC model described in Section 3.2 has been implemented in synthesizeable SystemC (Synopsys, 2002). Hence, the design can be synthesized with the front-end ParSyC (Fey *et al.*, 2004). The acquired FSM representation of the SystemC design and a specified PSL property are taken as inputs for a SystemC property checker (Drechsler and Große, 2005).

In this section we present the formal verification of an aspect of CHs, which is very important for the system to avoid safety hazards. There are several mechanisms to manage failures. One of them is the counter control token (`cnt_ctrl_token`). If the `cnt_ctrl_token` is set, the corresponding TVDS is indicated as occupied. There are two different classes of failures, which have to be considered for setting of the `cnt_ctrl_token`:

1. Erroneous impacts on a single sensor system (including a breakdown of a single sensor system).
2. Erroneous impacts on both sensor systems without axle counting.

The necessary evaluation for these failure classes is carried out in two steps. Firstly, the counters for the two failure classes are calculated. Secondly, the counters are compared to predefined constants. For the impact on a single sensor system the constant is called `s_max` and for the impact on both sensor system it is called `c_max`. With the property in Figure 9 the condition to set the `cnt_ctrl_token` is verified. This property states, that if the system is evaluated (line 6) and if the counters for the failures are greater than the corresponding constants (lines 7 – 8), the `cnt_ctrl_token` has to be set. This property is verified in 2.7 seconds on a computer with 1GB main memory and an AMD Athlon 64 3500+ CPU running under Linux. But at this point of the verification we still have to prove that the variables `ss` and `cc` have the correct values, which is performed in the verification steps to follow.

Erroneous Impacts on single Sensor Systems: In this paragraph the correct value

of counter ss is proven. Every time the state unaffected is reached, the counters for the single axle are added to the corresponding counters for the group of axles. Before the counters for a single axle are set to zero, the counter s is calculated and added to the counter ss. To prove the correctness of the latter counter, two steps are needed:

1. The correct implementation of s as a function of the state transitions performed during the counting process has to be verified.
2. The correct implementation of ss as a function of the sequence of s values has to be verified.

The first part is an inductive proof. It takes 7.2 seconds to prove the initial step and the induction step. The second part has to hold under any assumption. This means that no inductive reasoning is necessary to prove that the value of ss always is the sum of the previous value of ss and the currently calculated value of s. Thus, we can conclude, that during the evaluation phase ss is the sum of all s. The proof takes 14.0 seconds.

Erroneous Impacts on both Sensor Systems: The proof of the correct value of cc is analogue to the proof for ss. The counter cc is the sum of all c. And c is calculated at the same time as s with a different equation. Like s, c is proven inductively in 6.6 seconds. That cc is the sum of all previous c, is proven 3.2 seconds.

Altogether, the verification of the counter controller cnt_ctrl_token and its invariants took 33.7 seconds.

6. Conclusions and Future Work

We have shown a design flow oriented at efficient modeling and verification of CHs. This design flow is based on SystemC. We introduced the functionality of a CH and explained its modeling in detail. This model can also be simulated and verified.

The future work consists of the complete formal verification of CHs using bounded model checking and inductive reasoning. We already gave a proof of concept by proving the correct behaviour of the model for the failure class counter control in the preceding section. Another part in future work will be proving completeness of the verification.

REFERENCES

Accellera (2004). *Property Specification Language Version 1.1*.

Biere, A., A. Cimatti, E.M. Clarke, M. Fujita and Y. Zhu (1999). Symbolic model checking using SAT procedures instead of BDDs. In: *Design Automation Conf.*. pp. 317–320.

Burch, J.R., E.M. Clarke, K.L. McMillan and D.L. Dill (1990). Sequential circuit verification using symbolic model checking. In: *Design Automation Conf.*. pp. 46–51.

Drechsler, R. and D. Große (2005). System level validation using formal techniques. *IEE Proceedings Computer & Digital Techniques, Special Issue on Embedded Microelectronic Systems: Status and Trends* **152**(3), 393–406.

Fey, G., D. Große, T. Cassens, C. Genz, T. Warode and R. Drechsler (2004). ParSyC: An Efficient SystemC Parser. In: *Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*. pp. 148–154.

Grötker, T., S. Liao, G. Martin and S. Swan (2002). *System Design with SystemC*. Kluwer Academic Publishers.

Siemens AG (2003*a*). Az S M Multiple-section Axle Counting System. Copyright. Siemens AG.

Siemens AG (2003*b*). Safety for the Rail Services. www.siemenstransportation.co.uk/pdfs/AzSM%20R.pdf.

Synopsys (2002). *Describing Synthesizable RTL in SystemC^{TM}, Vers. 1.1*. Synopsys Inc. Available at http://www.synopsys.com.