Towards Provable Protocol Conformance of Serial Automotive Communication IP

Jens E. Becker*, Oliver Sander*, Alexander Klimm*, Jürgen Becker*, Katharina Weinberger†, Slava Bulach†
*Karlsruhe Institute of Technology (KIT), Germany - {jens.becker,sander,klimm,becker}@kit.edu
†Robert Bosch GmbH, Germany - {Kat.Weinberger,Slava.Bulach}@de.bosch.com

***Abstract*—Serial communication protocols are the backbone in today's automotive electric/electronic-architectures. Protocol conformance is of paramount importance to ensure interoperability, error free and reliable communication of electronic control units. Practice shows that, despite extensive simulation and conformance testing, there is no guarantee for the absence of errors in any given state of the communication cycle in any given combination of communication partners. Therefore sporadic functional errors can occur in the field. By nature this class of errors is almost impossible to reconstruct.**

Formal verification is considered an approach to cover such rare corner cases and to prove the absence of functional errors. Up to now the full verification of deep sequential protocols has been beyond the reach of formal verification. Also the question whether the developed assertions cover the full functionality has been left unanswered. Within the project HERKULES funded by the german government research teams of six companies and six research institutions collaborated on these topics.

A formal verification approach was developed that allows to ensure the absence of functional errors in critical IP in the design phase already. This verification technology and methodology as well as formal coverage analysis have been evaluated on the automotive protocols LIN and FlexRay, thereby assessing the effectiveness and performance of the new approach.

This paper discusses the results and shows with an exemplary application on a previously extensively verified and assumably error free LIN IP, how previously unknown errors were found. Additionally it shows that the verification effort using the formal verification approach has been less than that of the previous simulation-based verification techniques. The results demonstrate that the developed approach is a powerful, scalable alternative to extensive testing also under industrial requirements. For the first time, highest quality of deep serial communication IP can be achieved.

I. Introduction

One of the most important basis for electric/electronic-architectures (E/E-architectures) in modern vehicles is a reliable and error free collaboration of electronic control units (ECU) in a network. Integration of ECUs of different suppliers is one of the major challenges for the original equipment manufacturer (OEM). A variety of standardization measures allows for principle compatibility of different ECUs.

Communication between ECUs is mostly standardized by the use of bus standards such as LIN [3], CAN [1] and FlexRay [2]. As automotive applications demand highest quality it is required that a bus controller is working correctly within the complete parameter space in order to ensure interoperability within the system. Today bus controllers are checked using conformance tests. They consist of a multitude of scenarios where the controller has to show correct behavior. Within this approach there is the chance that error cases remain undiscovered as they have not been tested in the conformance test. Even if all modules of the system passed the conformance test, it is possible that, when using a certain set of parameters, there might be sporadic errors or even no communication is possible at all. Thus elusive bugs show up only later in the field. Even with the knowledge of a bug and potential problems it causes for the whole system, suppliers have to carefully ponder whether they will fix it or not. The reason is that they can never be sure, if the bug fix will not introduce new errors that can not be discovered with the existing test and might cause even more problems. Therefore these bugs tend to stay unfixed if a work-around is possible.

To overcome this problem, the goal is to eliminate bugs directly in the development process. An approach for this is the usage of formal verification methods. The idea behind this is to formulate a set of properties that is expected to completely cover the behavior described in the specification. Then it is formally proven that a given implementation complies with these properties. This is denoted as property checking or model checking.

Nowadays formal verification is an approved method to prove the conformance of an actual implementation with the given specification. It is well suitable for modular designs with blocksizes of 50.000 to 100.000 gates composing to more complex System-on-Chip (SoC) with up to 100 mio. gates. Nevertheless existing tools and toolflows are limited to processor-like designs that consider only time spans of a few clock cycles per property to be verified as an increasing time span to be considered also drastically increases the run time of the tools. Serial protocols imply observation times that can range from 100 to several 1000 clock cycles. This leads to unacceptable verification run times. In addition, communication history also has to be taken into account thus increasing the complexity of the problem even more. All these constraints make it practically unfeasible to verify serial protocols in a reasonable amount of time with existing tools and methodologies.

In order to overcome this problem six companies and six research institutions collaborated in the German government funded project HERKULES to investigate new approaches for formal verification of serial communication systems. Within this project, existing tools have been enhanced and new approaches have been developed to allow for efficient veri-

fication of serial protocols. Special attention has been spend on coverage analysis in order to be able to conclude whether an IP block completely fulfills a given specification. To show the capability of these new approaches they have been applied to CAN, LIN and FlexRay IPs.

The case study presented here shows the advantages of the formal verification approach for automotive communication protocols by the example of the verification of the *break behavior* of an industrial LIN controller. We show that the effort to prove specification conformance using formal verification can be much lower then with simulation and that it allows for stronger declarations on the quality of an IP block as it can prove correct behavior.

The paper is structured as follows: Section II introduces the problem tackled in this study using an abstract communication model. In section III we give some basic information to the LIN protocol and our implemented methodology that is needed to understand the performed case study which is then described in section IV. The results of the study are presented and discussed in section V. Section VI evaluates the methodology used in this work while section VII concludes the paper.

II. Problem Description

We assume a communication system with N entities. At every point in time we have exactly one entity M that is the master of the system and $N-1$ entities S that are slaves. Each transmission is based on frames consisting of transmission initialization by the master and a subsequent field which is sent by at least one slave.

Only the master M is allowed to start a new transmission issuing a dedicated trigger signal or signal pattern T. The trigger T interrupts all ongoing communication, resets all communication nodes to the corresponding internal state for new frame processing and must be detected by every slave in the system. To enable for a robust reset of all nodes to a new frame, T has to be recognized in any phase of the transmission.

The characteristics of the serial communication model comprises the following:

1) *Variation of the length of the message.* The length t_{frame} of any message may contain an arbitrary number of bits $0 < n_{bit} < n_{maxBit}$.
2) *Variation of the occurance of the trigger signal.* The trigger signal may occur at any point in time during a transmission. We refer to this point in time as $t_{trigger}$.
3) *Variation of the trigger symbol.* The trigger may consist of more then just one single bit, thus forming a fixed trigger bit pattern.
4) *External signals and events that are not part of the protocol.* External input signals to the entities may influence their communication behavior, even if the external signals are not directly related to the communication. An example for such a signal is the `reset` signal that sets the entity to its inertial state not taking into account the bus signals.
5) *History of the entity.* Any communication or signal variation (as defined above) over time prior to $t_{trigger}$ may have an influence on the communication behavior of the entity. For example the internal configuration could be changed in such a way that a correct communication is not possible any more.

To verify the behavior described above by means of simulation, a set of testbenches tests the reaction of the system to a trigger T for each transmission step. This leads to one testbench for every point in time where T could be issued. In other words the number of testbenches is basically given by the length of the frame measured in clock cycles. In addition several parameters influence the number of simulation runs needed to get an exhaustive result.

Because of the variety of possible combinations of the characteristics the state space to be covered by the implemented tests becomes extremely large. When adhering to bit wise simulation this means that every combination of internal signals and states of the design under test (DUT) has to be considered. It is necessary to provide a set of simulation stimuli to send the DUT into all possible internal states. It has to be assured that all internal signal and state combinations that can result from the communication history of the DUT (that can be of arbitrary length), messages sent or received, and all external signal variants over time are covered. The more complex the core gets, the larger the number of test cases gets in order to reach full coverage. This makes it impossible to exhaustively test complex designs in a reasonable amount of time.

Therefore the verification is most commonly restricted to a selected number of test cases that are found to be meaningful by some metrics. However the generation of the test cases itself is error-prone as there are few means to detect erroneous simulation scenarios. In addition the problem here is to know whether all relevant test cases are covered in the test bench since a full test coverage is usually not achievable.

III. Background

In this chapter we provide some basic information on the LIN protocol that should help to understand the presented case study. We will also shortly introduce the applied verification methodology.

A. LIN Protocol

The Local Interconnect Network (LIN) is a serial protocol with deterministic data cycles. It is mostly used for robust, low frequency communication links in simple actuator sensor networks for non safety critical applications.

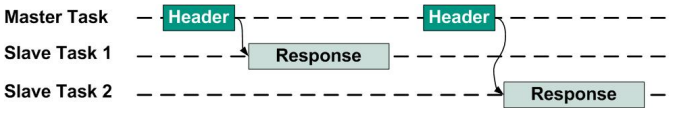

Fig. 1. LIN Communication

LIN is as a single master, multi slave system thus not requiring complex arbitration schemes. All communication is controlled by the central master that is polling the slaves

regularly based on a predefined schedule table. Every node in the LIN network implements a slave task to handle the communication protocol. The single master node implements an additional master task that is responsible for controlling the communication as roughly depicted in figure 1. A header is issued by the master task, followed by a response of one single slave task.

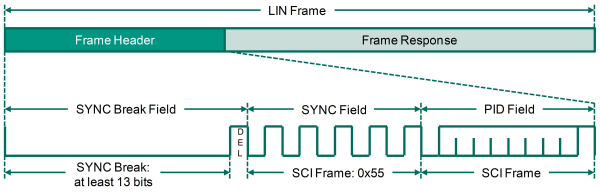

Fig. 2. LIN Protocol Frame

Figure 2 gives an overview of the header's content. The first field of the header is the SYNC Break Field (from here on denoted as BREAK). It marks the begin of a new transmission by sending at least 13 dominant bits followed by one recessive delimiter bit. As the LIN uses open-collector bus drivers, the dominant bits override all other transmission. The LIN specification states that this break condition has to be detected after 11 bits at any time during operation.

Following the BREAK is the SYNC field, consisting of ten alternating bits that are used to synchronize the slave to the master clock. For the clock frequency, the LIN specification allows for deviations of up to $\pm 15\%$ to the nominal clock frequency.

Finally the header frame is closed by the Protected Identifier (PID) field that addresses the slaves and determines their desired behavior.

The slave response can then consist of up to eight data bytes and an additional checksum byte. Every data byte of a transmission is extended by a start bit as well as a stop bit. The only exception to this rule is the BREAK symbol.

B. Methodology

In this chapter we give a short overview of the formal verification methodology used in our case study. We are using the GapFreeVerification^{TM} [4] technology developed by OneSpin solutions, one of the partners in the HERKULES project. This technology ensures that the behavior of the Design Under Test (DUT) is fully covered by the property set that is used for verification. One element of the technology is the so called property graph that links together all properties of a property set. Thus all properties are linked together having one or more predecessors and successors. This means that each individual property describes only part of the behavior of the DUT usually making them shorter and more comprehensible. Therefore the verification engineer has to find a suitable way to split the desired behavior of the DUT into smaller parts that can then be transformed into properties.

For our study the division into individual properties is oriented on the parts of a LIN frame as can be seen in figure 3. For example, for each field of the header (BREAK, SYNC

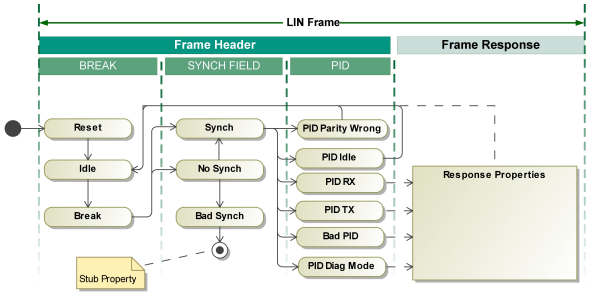

Fig. 3. LIN Slave - Property Graph

and PID field) there are one or more properties assigned to the individual field that describe the desired behavior for this part of the protocol. The properties are the linked together to form the property graph. As we only want to give the idea of the methodology, all properties that belong to the slave response are summarized as "response properties".

IV. Case Study

To show the benefits of this verification methodology we here present a case study of a selected property that has been carried out under industrial constraints. It demonstrates the power of our approach with its advantages and the good usability especially under industrial constraints.

As communication entity we used a hardware LIN slave IP block from Robert Bosch GmbH [5] that can be parametrized during design-time to be compliant to the LIN specification version 1.3, 2.0 or 2.1. For our study we have configured it to version 2.0. The core is a hardware module that implements the LIN protocol without using additional software running on a CPU. Internally it uses parallel finite state machines (FSM) to implement the protocol behavior as well as checking for possible error conditions. The bus signal is acquired using 16-times oversampling and majority voting. Besides the LIN bus signals it provides additional API signals that are used to exchange data to be send or received with externally connected devices. In addition there are several other input signals that are used to configure the core and to control its behavior during run-time. Some configuration can be done via the LIN bus using diagnostic frames issued by the master.

This core has already been extensively tested during design-time and has passed all compliance tests required for LIN approval. It has been used for years in current applications in the automotive domain. Up to now, no severe errors have been reported. Thus the actual version has been seen as presumably error free.

A. Verification of BREAK behavior

One goal of the verification is to prove that the IP core is able to detect and react on a break symbol (our instance of T, see chapter II) at any time and under any condition. Only the assertion of a reset signal may override this behavior.

The property to verify the break behavior of the core serves as basis for all other properties that prove correct protocol behavior of the IP core. A simplified version of the property is given in listing 1. The property is written in InTerval Language

```
1  PROPERTY break IS
2  DEPENDENCIES:
3    no_reset,
4    configuration_stable;
5
6  ASSERTIONS:
7    all_states_vaild,
8    all_registers_valid;
9
10 FOR TIMEPOINTS:
11   t_break_start = t+4,
12   t_rx_up = t_break_start+208;
13
14 ASSUME:
15   DURING[t_break_start-4, t_break_start-1]: lin_rx = '
       1';
16   DURING[t_break_start, t_rx_up-1]: lin_rx = '0';
17   DURING[t_rx_up, t_rx_up+4]: lin_rx = '1';
18
19 PROVE:
20   AT t_rx_up+4:
21     fsm_eval_syncfield,
22     registers_eval_syncfield;
23
24   DURING[t_rx_up, t_rx_up+4]: lin_tx = '1';
25
26 END PROPERTY;
```

Listing 1. Break Property in InTerval Language

```
1  assume property (@(posedge clk) disable iff (reset)
       configuration_stable);
2  assume property (@(posedge clk) disable iff (reset)
       all_states_valid);
3  assume property (@(posedge clk) disable iff (reset)
       all_registers_valid);
4
5  sequence t_break_start; nxt(t, 4); endsequence
6  sequence t_rx_up; nxt(t_break_start, 208); endsequence
7
8  property break_p;
9    during_excl(t, t_break_start,       lin_rx == 1'b1)
       and
10   during_excl(t_break_start, t_rx_up, lin_rx == 1'b0)
       and
11   during(t_rx_up, nxt(t_rx_up, 4),    lin_rx == 1'b1)
12 implies
13   t_rx_up##4 fsm_eval_syncfield() &&
       registers_eval_syncfield() and
14   during(t_rx_up, nxt(t_rx_up, 4), lin_tx == 1'b1);
15 endproperty
16
17 break: assert property(@(posedge clk) disable iff (
       reset) break_p);
```

Listing 2. Break Property in SystemVerilog

(ITL) developed by OneSpin Solutions. All explanations refer to this listing. As additional information listing 2 gives the same property described in SystemVerilog.

The first lines of the property state some environment constraints considered for the verification presented in this case study. One is the exclusion of the reset case (line 3) and the other is the assumption that the configuration of the core is not changed via external API signals during normal operation (line 4), meaning that these configuration signals remain stable. Nevertheless no assumptions on the actual values of these signals have been made. Arbitrary configuration of the core using diagnostic messages send over the bus is still possible.

Besides these externally induced constraints we also added some assertions to the property in order to make sure that only valid combinations of states for the FSMs as well as valid assignments for the internal registers are assumed (lines 7 and 8). As they are realized using assertions, they are proven to be always correct by the verification tool as well.

In lines 11 and 12 some variables are defined that can be used to refer to certain timepoints more easily. We then assume a correct break symbol to appear on the bus, meaning that at t_break_start the bus signal changes from recessive level to dominant level and remains there for 208 clock cycles. After that the bus becomes recessive again (lines 15 to 17). The 208 clock cycles result from the core using 16 times oversampling to determine the values of the bus signal. Therefore 13 dominant bits match the 208 clock cycles used in this property. This concludes all of the property's assumptions and constraints.

As required by the LIN protocol specification, the prove part of the property states that after the end of the break symbol (at $t\_rx\_up + 4$) all internal states and registers of the core are set to the appropriate values that correspond to a successful detection of a break symbol (lines 21 and 22). Line 21 defines the according combination of states in all state machines. The register values are defined by the macro in line 22. They may have certain values or have to be in a certain range. Of course this description requires some internal knowledge of the IP implementation. However this would be necessary for simulation as well. Furthermore we prove for the outgoing bus line that there is no transmission going on (line 24) as required by the LIN protocol specification.

V. Verification Results

When running the property described above, a counter example was generated (see figure 4). It turned out that there is a situation where a break signal sent on the bus was not detected properly by the core. As illustrated in figure 4 a break symbol has been detected by the core in clock cycle 810. It is signaled by setting `breaksymbol_reg` to '1'. This should result in the core's FSMs to change to the `evalsyncfield` state, but the corresponding state machine remains in the state `prtfsm_idle`. In order to understand the reason for this counter example we have to explain in more detail the timing behavior of the LIN bus.

A. Counterexample Analysis

The LIN specification allows for a deviation of up to $\pm 15\%$ of the actual bit time compared to nominal bit time. The slave has to sync to the actual bit time in order to be able to correctly interpret the data. Therefore the master task sends a sync field after the break symbol consisting of alternating ones and zeros (0x55) (see fig. 2). This sequence is used by the slave tasks to determine the actual bit time and adapt to it. The IP core evaluated in this work uses an internal register called `timebase_reg` to determine the correct sampling points. In contrast, to detect the break and sync field fixed sampling points are used that are independent of `timebase_reg` and refer to the nominal bit time.

For the counterexample the scenario is as follows. The slave is receiving a normal transmission with a slow sync, meaning

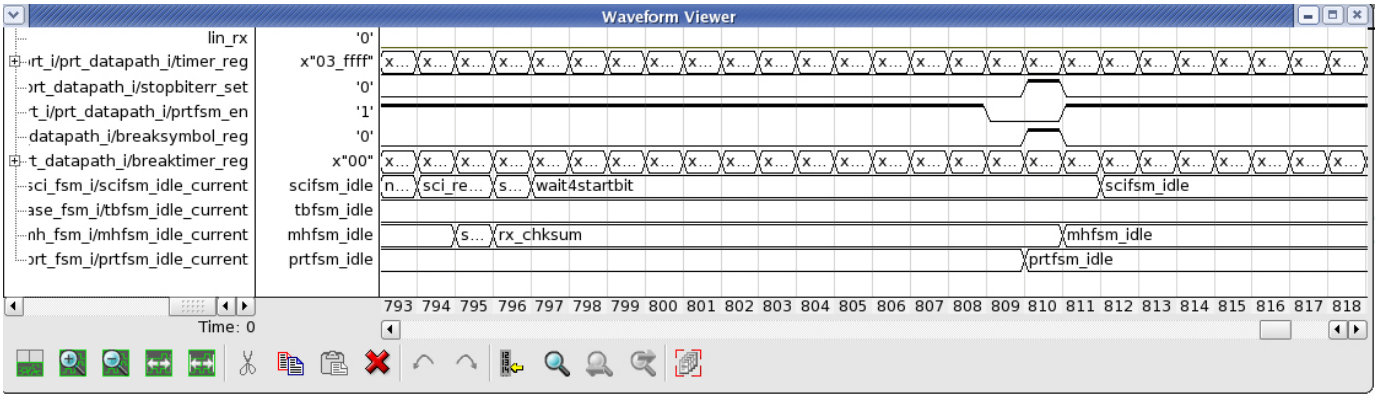

Fig. 4. Counterexample

that the actual bit time is longer than the nominal bit time. Following the stop bit, the slave constantly receives a dominant signal on the bus. One of the core's internal state machines is interpreting this signal as normal data transmission, using the `timebase_reg` to separate the signal into individual bits. An other state machine is constantly monitoring the bus for the occurrence of a break signal. This FSM is not using `timebase_reg` as the receiving FSM but the nominal bit time to separate the individual bits. This means that both FSMs count for a different number of bits in the same amount of time.

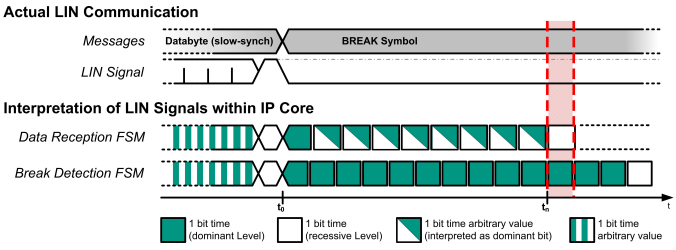

Fig. 5. Ambiguous Error Detection

The receiver FSM will interpret the dominant level on the bus as a sequence of a start bit followed by 8 dominant data bits. It would then expect the next (tenth) bit to be a recessive stop bit. This condition is violated as the bus is still on dominant level, causing the FSM to detect a stop bit error and assert the internal flag `stopbiterr_set` that is only valid for one clock cycle. At the same time, the other state machine responsible for break detection is interpreting the actual bus signal as 11 dominant bits in a row, forming a break symbol. Therefore it asserts the flag `breaksymbol_reg` for one single clock cycle (see figure 4).

Due to the actual IP implementation the stop bit error is evaluated in the following clock cycle and is taking priority over the break detection. This means that although a break has been detected the actual transmission is discarded and the core returns to idle state, waiting for a new break signal to start another transmission. As can be seen in figure 4 at timepoint 811 all internal state machines (`tbfsm_idle_current`, `mhfsm_idle_current`, and `prtfsm_idle_current`) are set back to their idle state. One timepoint later this causes the serial communication interface FSM (`scifsm_idle_current`) to fall into the idle state as well. The flag that indicated the detection of a break symbol is deasserted after one clock cycle and can not be evaluated after timepoint 810. This means that the core will not recognize the transmission that has been started by the break signal. Therefore the core does not handle the break symbol correctly according to [3] and data may be lost..

This error condition is not limited to one single event but can occur for divers combinations of assumed clock frequency for the ongoing transmission and clock frequency used to send the break signal. It is very unlikely that such an error will be detected by simulation as it requires a very specific scenario of matching parameters and internal signals in order to occur. However these parameters may be reached in reality and lead to elusive errors which may result in indeterministic failures of the superior system.

B. Design Improvements and Detection of new Counter examples

After the bug was detected and verified, the design department has been contacted in order to fix the bug. As described above the bug related most likely to a wrong prioritization of the different error flags that resulted in the loss of the second flag for the break detection. A fix was made to cope with this problem and the fixed code of the core was rechecked with the same break property again.

The result was that the property still did not hold, as a new counter example was generated. Analysis of this new counterexample revealed a new error that belongs to the same class as the error detected before. It results from another type of protocol error that is detected in parallel to the break detection. As this error is also prioritized to the break error, the break is still lost and the subsequent LIN frame is discarded by the design under test (DUT).

The detection of the second error makes clear that formal verification never shows all errors of a design at once. It provides only a single counterexample out of many that all would have led to a failing property. On the other hand one can see that formal verification is much more powerful than

simulation as the same property can be used over and over iteratively, until all errors have be fixed in a given design. For simulation, individual test scenarios have to be implemented for each of the two error cases discussed here.

VI. Methodology Evaluation

When comparing simulation and formal verification, one might think that they are very similar at first sight. Both show that the desired behavior of a DUT, based on assertion of signals to the system, is identical to the behavior demanded by a given specification. But this description of the signal assertions is where both methods differ greatly.

For simulation a two step approach is required. At first the correct behavior of the design is tested with a test bench that generates a correct stimulus as described in the specification. One requirement for simulation is that all input signals are determined during runtime of the test bench. That is why, signals that are supposed to have no influence on the tested system behavior are bound to fixed values.

Doing so, the correct reaction of the design to the correct stimulus can be shown. In a next step, the designs reaction to possible error conditions is checked. This means that stimuli are created that are supposed to lead to incorrect behavior. As before, all signals have to be described in the test benches. The problem here is that the verification engineer has to imagine all possible error scenarios and has to create a test bench for every case. All error conditions that the verification engineer can not imagine are not checked.

While for smaller designs with few internal states and input signals, it can be feasible to check all input combinations, this is unfeasible for larger designs. Therefore, simulation normally only covers parts of the complete event space, taking into account only those scenarios that the verification engineer considers to be important. Correct behavior of a design under all conditions can never be guaranteed for these large designs.

With our communication model presented in chapter II A in mind we can roughly estimate the effort required to verify the correct reaction to the break symbol using a simulation environment. Assuming an oversampling factor of 16, 8 data bytes, and LIN as protocol we already need a set of 6,720 different simulations that differ in the break pattern position.

$$sim_1 = length(header) + length(response) \quad (1)$$
$$= (33 + 8 * 10 + 10) * 16 = 6,720 \quad (2)$$

By further adding variations as mentioned in chapter II we get even more simulation scenarios. For example the number of data bytes gives factor 8, synchronization variation factor 5, resulting in $sim_2 = 8 * 5 * sim_1 = 268,800$ different simulation runs that are easily covered by one single formal property. Moreover we have to be aware that this estimation is extremely conservative as we left out data values, additional external signals and history based core behavior.

Completely different from that is our formal verification approach. Here we also describe the actual state of the design and the input stimulus of the system. But in contrast to simulation all signals that are supposed to have no influence to the actual behavior are left undefined. The verification tool will then use this degree of freedom in order to test whether the design behaves as described in the prove part for every possible combination of the input signals and internal states that have been left open by the verification engineer.

If the property holds, this proves that the design will allways behave as described and that there are no other undescribed corner cases left. Using this methodology enables the verification engineer to concentrate on the description of the intended design behavior and is unburdened from the need to imagine all possible error scenarios that the design might fall into. This greatly reduces verification effort compared to simulation.

VII. Conclusion

This paper shows the advantages of formal verification methods to realize proven protocol conformance for automotive communication systems. As an example we have presented the verification of the break behavior of a LIN controller. We were able to discover an error that never showed up in conventional conformance tests based on simulations, although the LIN IP has been extensively tested during the last couple of years and no erroneous behavior of the LIN IP has been reported yet.

Using formal verification, it is possible to completely cover large event spaces for design verification that take into account all possible history states and input combinations that might influence the systems behavior. In contrast to simulation approaches it is even possible to prove that a design is error free while this is nearly impossible for simulations.

The verification of the break behavior is embedded in a broader approach to completely verify the correct behavior of a LIN core as described in [6]. With the methodology developed in the HERKULES project it is now possible to guarantee correct functionality of serial protocols. As we have shown, the effect of applying formal verification under industrial constraints is not only possible but also achieves much higher quality of verification, covering even seldom reached corner cases that are not taken into account in simulation. Correct protocol behavior of a device compared to a given specification can now be proven.

Acknowledgment

This work has been founded by the German Federal Ministry of Education and Research (BMBF) in the project HERKULES (Förderkennzeichen: 01 M 3082).

References

[1] Bosch. CAN Specification Version 2.0, March 1997.
[2] FlexRay Consortium. FlexRay Communications System Protocol Specification, Ver 2.1, Revision A. Available electronically at http://www.flexray.com/, 2006-2009.
[3] LIN Consortium. Lin specification package, revision 2.0, 2003.
[4] OneSpin. www.onespin-solutions.com/news_GapFree.php, 2008.
[5] Robert Bosch GmbH. C_LIN, LIN Module Designers Guide, Revision 1.5.1, 2007.
[6] O. Sander, A. Klimm, A. Hogh-Binder, S. Bulach, and K. Weinberger. A top-down formal verification approach of lin hardware ip based on the gapfreeverification(tm) process. *EDA Workshop, Dresden, Germany*, 2009.