Accurate Message Level CAN Simulation

Florian Bogenberger, Carsten Mielenz

In recent years distributed systems using the CAN bus have been increasing in both size and complexity. In order to meet system requirements, the impact of communication on the overall system must be understood early in the design cycle.

Several approaches have been used to address questions like response time at different bus loads, functional partitioning, etc. However most approaches neglect real life conditions like execution time, transmission errors or component damage. This paper presents a modelling strategy to simulate CAN based systems under real life conditions. We present a high performance CAN protocol model working on message level but providing bit level accurate functionality and timing.

The presented methodology allows to analyze the reliability, performance and overall behavior of CAN systems under real life conditions. Results can be used early in the system design cycle to make a tradeoff analysis, repartition functionality, refine algorithms and to optimize the network configuration.

Keywords: Distributed Systems, System Simulation, High Level Modelling, CAN Network, Reliability Analysis, Performance Evaluation, Communication Network

1 Introduction

In the automotive industry a continuous trend of increasing complexity can be observed, especially with respect to electronic systems. With regard to distributed systems this is true not only for ECUs (Electronic Control Unit) but also for the communication between ECUs. The complexity must be handled during system development and be addressed by appropriate simulation capabilities, which are powerful enough to allow the simulation and analysis of a set of ECUs including their interconnection. Moreover turn around times are decreasing, so the traditional sequential development flow will be paralleled even more than today. Concurrent engineering is one of the drivers for „executable specification“. One concrete way to achieve this is to use high level simulation models. The high level approach promises to achieve the required performance even for complex systems. However very often they neglect low level effects, which can have a significant impact on the whole system. Examples are timing or error handling - without a reasonable modelling of errors the reliability aspect of a system cannot be simulated. To avoid expensive pitfalls in later development phases, these effects must be estimated with enough margin to be able to compensate for unexpected problems. If the margin turns out to be too high the system might be more expensive than necessary. In case the margin is too low, expensive reiterations are required to fix the problem, which again increase system costs. As a consequence we think that high level simulation models can be a reasonable approach to simulate complex systems, however they must also take into account those low level effects which have a significant impact on the overall system performance and reliability.

This paper describes a new simulation methodology used to implement a message level CAN model which provides high simulation performance combined with bit level accuracy regarding arbitration, timing and error handling. Section 2 compares characteristics of high level and low level CAN simulations. It is shown why a pure high level approach is not feasible for CAN based distributed systems. In order to overcome this problem we tried to make a mix in order to get the advantages of both worlds, accuracy and high performance, under one umbrella. Section 3 describes the abstractions we choose for the representation of message data, physical time and errors. Section 4 presents a new simulation paradigm followed by some simulation results and an outline of potential application areas. In the last section the key points ar summarized.

2 High Level versus Low Level

The intention of this paper is to describe a high level CAN simulation methodology. In this paper the terms „high level“ and „low level“ are used with respect to the degree of abstraction imposed on models and assumptions. In this context going up the abstraction level means ignoring more and more details of reality. The following two sections try to provide some means to the different levels.

2.1 Characteristics of Low Level

For low level simulations the primary goal is „accuracy“. Usually this means that the models used are relative near to the accurate physical representation, or, in case of software that the simulation behaves very similar to the final target system. Often more or less accurate timing information is taken into account and the number of simulation instances can be very large comparable to the number of bits and pieces in the real system. The computational effort can be tremendously high either because complex mathematical formulas must be solved or because there is a large number of events to process. This is a main reason why a pure low level approach is not feasible for system level.

A second point is that in order to get reasonable simulation results the system needs to be defined quite accurately - which is kind of a chicken-and-egg problem. One important goal of system simulation is to play with parameters, architecture, etc. during early design phases in order to optimize and define further details. So details needed for a low level simulation are not known at this phase.

2.2 Characteristics of High Level

For high level simulations there are three primary goals

- performance must be high enough to have reasonable simulation times
- results must be accurate enough to help making the most critical system decisions
- it must be possible to build a complete system without knowing all details

These goals are met by neglecting all those effects, which are assumed not to affect critical decisions. Timing for example is ignored frequently, instead the focus is on pure functional simulation. Effects that can not be ignored completely are often estimated with statistical formulas based on mathematical theories [KiSch97], measurements, experience and assumptions.

Performance is usually the most critical point, to reach this goal a certain degree of inaccuracy is accepted. However the art is to decide how much inaccuracy is feasible and where assumptions are too coarse.

2.3 Communication Channel Impact

As we are talking about distributed systems one of the most important factors is the communication channel. It significantly affects the system’s behavior regarding performance and reliability. Important factors are

- response time & message latency
- dynamic load of the communication channel
- behaviour under error conditions (reliability)
- handling of concurrent transfer requests (arbitration)
- protocol modes

In a pure high level approach, these aspects can be covered only partly. This is especially true for CAN, where identifiers play a significant role, transfer times depend on many factors, nodes can become error passive or even go in bus-off mode, etc. Things get even worse if errors need to be taken into account, e.g. to analyze the system’s stability under real world conditions. Even though mathematical formulas can be used to estimate minimal, maximal and average response times as well as latency and busload, the actual values depend very much on the concrete application and configuration.

Therefore we believe that when we are talking about CAN it is not sufficient to consider „system level“ to be identical to „high level“ - there are some low level effects that also must be taken into account.

3 Bridging the Gap of Abstraction Levels

As neither a pure „high level“ nor a pure „low level“ approach would fit our needs, we decided to go for a mix of both worlds: We used a network simulator as high level platform and implemented CAN modules, which are able to switch between both levels. The requirements for these modules were

- communication on message level in order to minimize the number of events
- accurate timing
- accurate error handling
- arbitration
- support the complete CAN2.0B protocol according to [ISOStd1] and [ISOStd2]

The following sections show what it means to combine the different abstraction levels addressed in this list.

3.1 Message Representation

„Communication on message level“ means that messages are not transferred bit by bit, but with a single event. Keeping the number of events as low as possible is key for high simulation performance.

Therefore messages are represented by data structures similar to those used in modern programming languages. Table 1 shows this structure for a CAN data frame.

Please note that this structure contains some data which is not part of a real CAN message like receive time, sender ID, etc. Partly the reason is to improve the system transparency by making messages traceable. The field bit duration, however, carries data, which would otherwise be lost (see section 3.2). It is used to emulate the duration of a CAN bit and can be changed to model different bit timings. The current implementation checks all transmitters connected to a single bus to use the same bit timing. Nevertheless different busses can have completely different timing, which might be connected by gateways. Effects like resynchronization and interconnect delays are not modeled, but they can be imitated by the error injection method presented in section 3.3.

Element	Contents
due to transmit	time when transmitter got trigger to transmit the message
Transmission start time	time when message transmission really started
bit duration	Duration of a single CAN bit
receive time	time when complete message was received and accepted
sender ID	string - unique identifier of the sender
Message ID	CAN identifier
data length	DLC field
Message data	Vector with up to 8 bytes containing the data field
is_remote	True if it is a remote message, false otherwise
is_extended	True if the message has an extended identifier, otherwise false

Table 1 – CAN Message Data Structure

On the other hand it lacks some data of its real representation like CRC and stuff bits. In reality this information is needed due to redundancy reasons in order to avoid or recognize transmission errors. Even though the simulator must take into account the delay added by these bits, it not necessary to push around their values. If the values are really needed they can be calculated any time. This might look contradictory to the goal to correctly model error handling, however let us elaborate first on what this representation means with respect to timing and then focus on error modelling.

3.2 Representation of physical Time

A real CAN message has a certain duration depending on the number of bits and the duration of a single bit. However, to keep the number of events low in our model a message is transferred with a single event - which is a point in time rather than a time span. The duration of a message is no more measurable by looking at the continuous change of the bus signal as there is only one change - the transfer event. That is also the reason why the field “bit duration” was added to the message data structure. This little detail has a significant impact on the whole modelling. For example let us look at arbitration.

In reality CAN messages sent concurrently overlap until loss of arbitration. However, an event based simulator can only work on one event a time. Consequently in the model concurrent messages must be sent in sequential events. If a real CAN node looses arbitration it simply stops transmission. In our model all the data is already transferred at the beginning. A node, which received the lower priority message, does not know if the transmitting node saw the higher priority message and therefore stopped transmission. This is why an additional message was invented, which is sent by the node that lost arbitration and marks the time when the node recognized this. Please note that like in reality the transmitter decides about arbitration.

This is an example that going upwards the abstraction level does not only mean to cut away information, but sometimes also means to add data not existing in reality.

Figure 1 shows which events are sent when a sender looses arbitration. There is one event for each message, then the first sender indicates loss of arbitration and acknowledges the receipt of the other message. After the interframe space it restarts transmission.

3.3 Error Modeling

In order to model bit errors another structure was invented containing basically a dynamic error bit array and a strength information. By using a bit array rather than single bits it is still possible to keep the number of events low - even if many consecutive error bits occur, only one event is required. Only very long error bit sequences should be split in several arrays. The strength information is necessary to distinguish between errors that overwrite dominant and recessive bits (like EMC (Electro Magnetic Compatibility) or shortcuts) and those that only overwrite recessive bits (like line driver errors). These error structures are used to overlap the CAN data structure described in 3.1.

This model allows imitating very different kinds of errors. For example the impact of resynchronization can be modeled. This can be achieved by overlapping a message with an error array where some bits of the message are shifted. The injection can be triggered either by statistical or by deterministic processes.

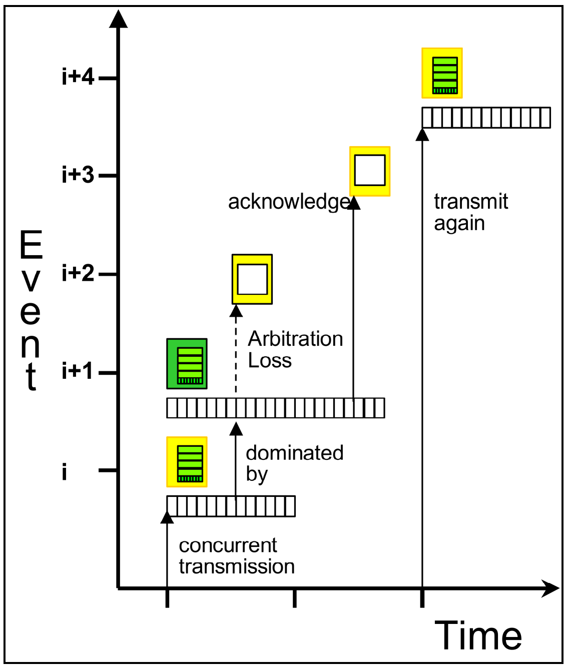

Figure 1 - Arbitration Modelling

4 A New Simulation Paradigm

The data structures described above contain a lot of information. In order to achieve accurate timing and behaviour under error conditions, the CAN simulation module must understand the protocol very well. Obviously it does not make sense to use a high level simulator to implement the CAN protocol.

Instead a specialized simulation engine was developed that is dedicated to the task to link a high level simulator with the low level aspects of the CAN protocol. Similar to a cosimulation this protocol simulator tries to run as far into the future as possible in order to predict the next high level event. However, in contrast to a cosimulation it does so not only based on the current state, but also with consideration of data which is valid only in the future. This is possible due to the message level communication concept. Moreover the protocol predictor is tiny compared to a normal cosimulation kernel, it is not running as a separate process (reduces communication overhead), it is not event based, and it is absolutely dedicated to the CAN protocol not caring about any other overhead.

The task of this module is to predict and schedule events for the high level simulator. Typically these events trigger message transmission, put the node in bus-off mode, trigger message acknowledgement, etc. If it turns out that the prediction was wrong, the event is removed from the schedule list by the protocol simulator and the prediction is restarted.

Figure 2 shows an example of the data flow when an error is injected. First the node starts transmission of a message and schedules an event to check if at least one acknowledgement arrives. Before that it receives errors overwriting partly the transmitted message. The original event is removed, an error frame is sent, and a new event is scheduled for retransmission.

5 Results

The CAN module was tested in different configurations ranging from a small two node system up to complete car networks with more than 30 nodes. Tests were run with and without error injection. We also rebuilt the case study [SiSoRa97] and could reproduce their results. The presented modelling strategy turned out to be very efficient regarding performance, accuracy and usability for high level system descriptions.

5.1 Accuracy

Timing is accurate down to bit level for message delay, arbitration, error behavior, etc. Table 2 shows a protocol excerpt where two nodes „X1“ and „Y“ start transmitting concurrently, then „Y“ looses arbitration, two error bits are injected and node „X1“ sends an error frame.

5.2 Performance

Even in the worst case of a fully loaded two-node system an Ultra Sparc 2 simulates with the performance of a 50kbit CAN network. For systems with more than 30 nodes and a realistic message traffic the simulation performance was measured to be about 10-20 times slower than the real CAN system.

Please note that due to the message level communication the number of events to process is significantly lower than in a bit level simulator. So this approach reduces the number of events by a factor between 44 and 132.

```
CAN Protocol, 250Kbit/s
TNow:     0.020768
Type:     CAN Message

due to transmit    0.02
transmission start 0.020768
sender ID          "Y"
CAN message ID     9
CAN data length    4
is extended        No
is remote          No
data field         Length: 4
Element Index      Element Value
0                  73
...
-----------------------------
TNow:     0.020768
Type:     CAN Message

due to transmit    0.02
transmission start 0.020768
sender ID          "X1"
CAN message ID     5
...
-----------------------------
TNow:     0.020808
Type:     Abort Message

transmission start 0.020808
sender ID          "Y"
-----------------------------
TNow:     0.020852
Type:     bit sequence

transmission start 0.020852
sender ID          "Error Injection"
strength           error injection bits
bit array          Length: 2
Element Index      Element Value
0                  0
1                  0
-----------------------------
TNow: 0.02086
Type:     active error frame

transmission start 0.02086
sender ID          "X1"
```

Table 2 - CAN Protocol

5.3 Application for System Development

The module turned out to be very handy to develop and analyse even large CAN systems in a short time. The CAN module can be stimulated with cyclical or statistical message generators or even by real application code. Different types of errors can be injected easily, modelling statistic or deterministic processes. Probes can be placed almost everywhere in the system, which allows measuring performance, response times, etc. as well as tracking a single node's state transitions.

6 Potential Application Areas & Outlook

6.1 Future Protocols

The presented simulation technology can be applied not only to CAN, but also to any other protocol. It allows analyzing very efficiently the protocol specific impact on a distributed system's behavior. This is also a quick method to compare advantages and disadvantages of different protocols.

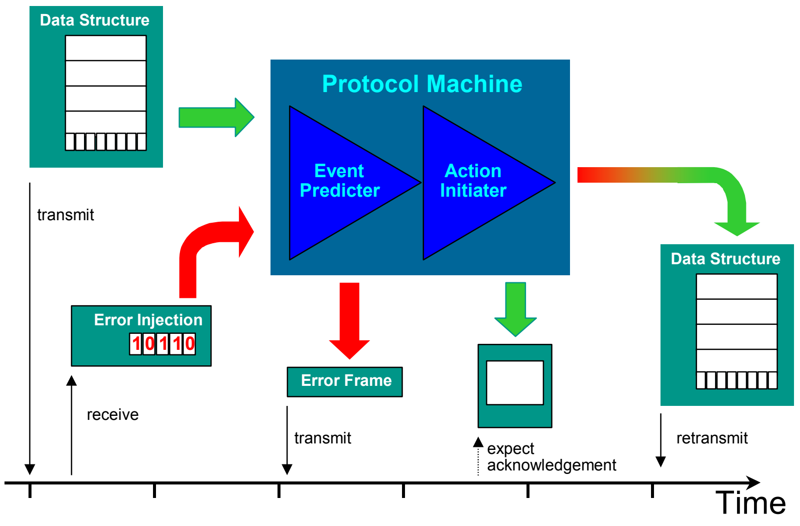

Figure 2 - Event Prediction

6.2 Virtual Hardware Prototypes

Relatively simple hardware blocks like message filters and buffers can be modelled efficiently in the network simulator. This method provides a way to analyze and optimize future CAN controller hardware with low effort. New filtering and buffering concepts can be tested in realistic network environments.

Apart from that the presented simulation technology is not bound to modelling of protocols. Looking at the model as a general CAN controller, any other hardware block, peripheral or even CPU can be modelled the same way. The concept of event prediction combined with higher level data exchange allows to cut down the communication overhead significantly. Peripherals or CPUs can be connected to the CAN unit in the network simulator like any other message source or sink. With those models it will be possible to estimate the CPU load depending on the application, network traffic, filter configuration, buffer scheme, etc.

6.3 Running real Application Software

Due to the fact that the implemented CAN module can be programmed similar to a real CAN controller, only small modifications are necessary (basically the CAN driver) to make real application code work together with that model. Even though the CAN module has accurate timing, this is not true for the application, which would be running in a pure functional mode. However, this might not be feasible for performance critical applications and must be addressed in future developments.

6.4 Comparison to Hardware Prototype Systems

Some of the features described are already fulfilled or even outperformed by hardware prototype systems currently on the market. However, we do not consider this technology as replacement or competition for existing prototyping methods. The presented technology does not have realtime capabilities neither can it be integrated in a real car. Nevertheless, it has other features making it more attractive for the earlier system definition phase, e.g.

- analysis and integration of hardware not yet existing (virtual prototype)
- high flexibility
- low effort to build systems
- highly parameterizable
- error injection
- excellent debugging features

7 Conclusion

The presented simulation methodology allows to run high level simulations of distributed systems with consideration of low level effects of the CAN protocol. Data communication is handled on message level whereas all details of the protocol are handled with bit level accuracy in a separate simulation unit.
Simulations can be used to analyze the system performance and reliability for ideal operating conditions as well as under

worst case conditions with transmission errors. The methodology can be used to make tradeoff analysis early in the design phase. Application software can be linked into the simulation. By building virtual prototypes new hardware architectures can be tested and optimized in a realistic network environment.

8 References

[KiSch97] Uwe Kiencke, Dirk John, Sandra Schneider, „Performance analysis of a distributed automotive real-time system“, ICC’97 Proceedings, page 07-02

[SiSoRa97] F. Simonot-Lion, Y.Q. Song, J. Raymond, „Validating real-time applications distributed over CAN: an interoperability verification“, ICC’97 Proceedings, page 07-09

[ISOStd1] ISO Standard: Low speed controller area network (CAN), ISO/DIS 11519-1

[ISOStd2] ISO Standard: Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication, ISO/DIS 1898

Motorola
{Florian Bogenberger, Carsten Mielenz}
Schatzbogen 7, 81829 Munich, Germany
Tel.: +49-89-92103-{421,311}
Fax: +49-89-92103-820
{ttg560,ttg585}@email.sps.mot.com