Rapid-Prototyping of a CAN-Bus Controller: A Case Study

Andreas Kirschbaum Frank M. Renner Alexander Wilmes Manfred Glesner
Darmstadt University of Technology
Institute of Microelectronic Systems
Karlstr. 15, D-64283 Darmstadt, Germany
andreask@microelectronic.e-technik.th-darmstadt.de

Abstract

In this paper we investigate the prototype generation of a Basic CAN-Bus controller. The controller is modeled as a hierarchical finite-state-machine from which a behavioral VHDL description is generated automatically. After synthesis the device is prototyped using a FPGA-based emulating system. An experimental CAN-Bus system is presented for design validation. We show that this rapid-prototyping approach, which exclusively uses commercial hard- and software, is feasible and has been successful in two respects: The turnaround time for prototyping has been reduced significantly from 2 months for ASIC fabrication to 10 hours while achieving an operation frequency for the very first prototype totalling one fourth of that of the anticipated final system.

1. Introduction

Apart from functional correctness and performance mainly economical aspects determine the success of a new product under development. With the growing number of competitors on the market *time-to-market* increasingly becomes one of the key factors determining the commercial success of a product. Rapid-prototyping, being defined as almost automatically generating a functionally equivalent prototype from an behavioral system description, can shorten the product development significantly. Working on a higher level of abstraction, implementation details are not of primary interest any more and the engineer is able to concentrate on more global decisions, i.e. choice of algorithms, system architecture etc.. This will result in a decreasing number of design iterations along with a reduction of the turnaround time by automatic prototype generation.

Nevertheless the acceptance of this design approach is still low due to high investment costs in tools and hardware and the lack of experience. Therefore the goal of this case study is to demonstrate the feasibility and the advantages of rapid-prototyping using a real-world design example together with exclusively commercial soft- and hardware.

The selected CAN-Bus controller (Controller Area Network) is a medium-complex interface circuit for a general-purpose-microprocessor/microcontroller (Section 2). Starting from a specification of the controlflow-oriented design as an extended finite-state-machine using Statecharts a VHDL-description of the design is generated automatically. After synthesis the register-transfer-level netlist is mapped onto a FPGA-based (Field Programmable Gate Array) hardware emulator (Section 3). For design validation an experimental CAN-Bus system consisting of the emulated prototype and a commercial controller is presented. In section 4 the main results of the prototype generation are summerized.

2. The CAN-Bus Controller

The Controller Area Network (CAN) [3, 5] is a serial communication bus designed for broadcasting short real-time control messages. Originally constructed for automotive applications by Bosch in 1983 CAN recently has also become popular in industrial automation (e.g. in textile industry) and medicine engineering for connecting distributed intelligent systems. Remarkable features are its minimal latency, fail-safe behavior due to error correction precautions which have been implemented and low connection costs for subscriber circuits.

The CAN-Bus is based on a line-topology. The maximal transfer rate correlates with the maximal extension of the bus system. Assuming a maximal transfer rate of 1 MBit/s the network extension is restricted to 40 meters, whereas maximally 50 kBit/s are achieved for the maximum extension of 1000 meters. 32 CAN-Bus nodes are allowed for maximal and up to 128 for lower transfer rates. Bus access is controlled via CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). CSMA/CA allows multiple, simultaneous bus accesses from subscriber nodes. In case of bus collisions the node with the highest priority gains bus

0-8186-7603-5/96 $5.00 © 1996 IEEE

access (bitwise arbitration) without a loss of data which has already been sent. This allows predicting a maximum latency of one message in spite of random bus access; a main prerequisite of real-time applications.

The data link layer (level 2) of the OSI-reference model (Open System Interconnection) of data communication is realized via a CAN-Bus controller. Its main tasks are: bit synchronization, bus arbitration, sending/receiving of messages and error correction. Beyond message filtering, message administration and autonomous message replies can also be handled. CAN-Bus controllers are available as stand-alone versions or integrated add-ons for standard microcontrollers/microprocessors or I/O-devices.

One has to distinguish between two different implementation concepts of message administration, namely Basic-CAN and Full-CAN [3, 5]. The Basic-CAN controller only provides one send buffer and a two-stage FIFO as receive buffer. Therefore with this concept the message transfer rate mainly determines the load of the controller's host processor which is responsible for the subsequent message processing. Consisting of an array of buffers to send and receive messages the Full-CAN controller relieves the host processor from a number of message administration tasks. Even an automatic message reply without the host interference is possible.

This paper focuses on the development of the prototype of a Basic-CAN controller (DBCAN = Darmstadt University of Technology BCAN). Generally it can be used as an integrated add-on for any custom specific processor. Within the Special Research Program 241 *Innovative Mechatronic Systems (IMES)* [1] at Darmstadt University of Technology DBCAN is needed in a single chip realization as a communication controller for an Application Specific Integrated Processor (ASIP). In that application the friction between the tyre of an automobile and the road surface is calculated in real-time [9]. The 32 bit ASIP evaluates the sensor signals of a specially prepared tyre and exchanges the results via DBCAN and the CAN-Bus with other superior data processing systems within the car (e.g. motormanagement or ABS/ASR-management).

Our device is almost compatible with the Philips Basic-CAN controller PCA82C200 [7]. Following the CAN specification 2.0 [6] 11-bit identifiers for data frames are used but also extended identifiers (29 bit) are tolerated. The host interface was modified to support full 32 bit access to the message buffers and control-/status-registers of the CAN controller. DBCAN consists of six basic building blocks (Fig. 1).

The *interface management processor* (IMP) executes the host processor commands, controls all DBCAN building blocks and interrupts the host if necessary. DBCAN is synchronized with the incoming bit stream via the *bit timing logic* (BTL). BTL also introduces bit stuffing in order to in-

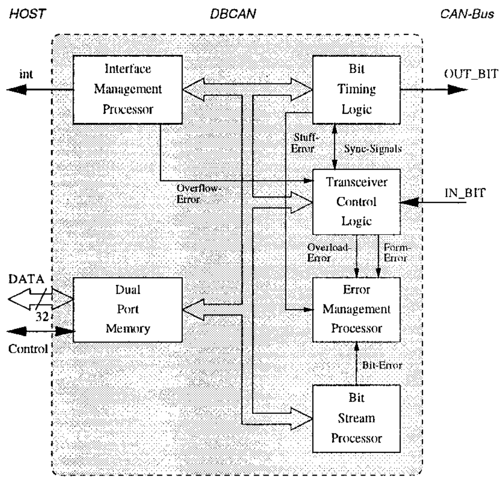

Figure 1. Block diagram of DBCAN.

crease communication reliability. The *transceiver control logic* (TCL) manages bus arbitration and sending respectively receiving of data frames. If any error occurs during data transmission TCL automatically sends an error frame to the other CAN-Bus nodes. Special counters in the *error management logic* (EML) keep track of all occurring errors and dynamically control bus access depending on the node's error frequency. The *bit stream processor* (BSP) serializes/parallelizes the data bitstream and detects arbitration losses caused by permanently comparing incoming and outgoing bits. All status-, control- and configuration registers of DBCAN are located in a *dual port memory* (DPM) which is accessible by the host processor.

3. Prototype Generation

The basic designflow followed during the generation of the DBCAN prototype is shown in Fig. 2.

For modeling the behavior of DBCAN an extended finite-state-machine description of the controller was created based on Statecharts using the tool *ExpressV-HDL*[1] (Section 3.1). *ExpressV-HDL* provides graphical design entry and animated, interactive simulation of the finite-state-machine. After system validation on the behavioral level a synthesizable VHDL description of DBCAN was generated without user interference. Afterwards the *Design-Compiler*[2] was used to synthesize the design to gate-level and to map it onto the ASIC target library (Section 3.2). In order to emulate the controller with the FPGA/FPIC-network (Field Programmable Interconnection Circuit) of the *Logic Animator*[3] the generated netlist had to be prepro-

[1] *ExpressV-HDL* is trademark of i-Logix Inc.
[2] *Design Compiler* is trademark of Synopsys Inc.
[3] *Logic Animator* is trademark of Quickturn Design Systems Inc.

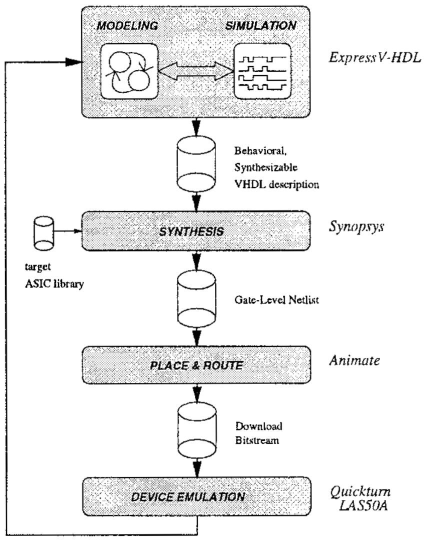

Figure 2. Rapid-Prototyping Designflow.

cessed, partitioned and mapped onto a number of FPGAs. This was done fully automatic with the *Animator*[4] software including placement, routing and bitstream generation for the different FPGAs and FPICs (Section 3.3). In an experimental network the emulated netlist of DBCAN was connected with the of-the-shelf BCAN controller 82C200 of Philips (Section 3.4). Tests for functional validation of DBCAN and performance measurements have also been made.

3.1. Modeling and Simulation

For easier design entry DBCAN was modeled with Statecharts [2]. The language was designed primarily for specifying reactive systems, which are essentially event-driven and control-dominated, for example controllers in communication networks. Statecharts are an extension of the traditional FSM description including three additional features: hierarchy, concurrency and communication.

A Statechart comprises different states and transitions, which are activated depending on a combination of conditions and events. Fig. 3 shows the Statechart of the *error management logic* of DBCAN. The EML is described by four states namely *recessive_count*, *waitforimp*, *bus_off* and *bus_on*. Hierarchy is introduced with state *bus_on* which itself consists of several concurrent sub-states (*tec*, *rec*, *error_state*). This strategy of design decomposition significantly reduces the exponential increase in states and transitions. Normally this occurs in complex FSMs which are inherently flat and sequential. By broadcasting events and data changes to all Statecharts in the design synchronization is realized. For example, state *error_state* is left and reentered if event *sw_reset* occurs which is generated by another Statechart. Also, status (condition) detection can be used for inter-chart synchronization. In Table 1 the characteristic parameters of the Statechart model of DBCAN are summerized.

[4]*Animator* is trademark of Quickturn Design Systems Inc.

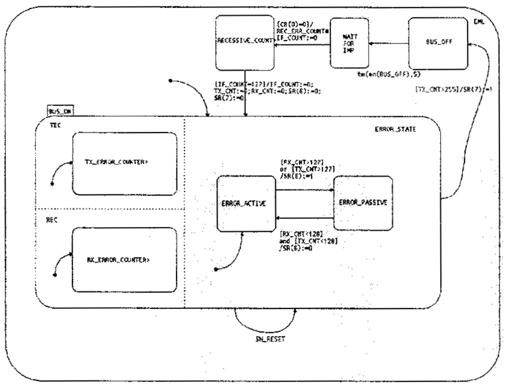
EML
RECESSIVE_COUNT
[CR(0)=0]/ REC_ERR_COUNT# IF_COUNT:=0
WAIT FOR IMP
BUS_OFF
tm(en(BUS_OFF),5)
[IF_COUNT=127]/IF_COUNT:=0; TX_CNT:=0;RX_CNT:=0;SR(6):=0; SR(7):=0
[TX_CNT>255]/SR(7):=1
BUS_ON
TEC
ERROR_STATE
TX_ERROR_COUNTER>
[RX_CNT>127] or [TX_CNT>127] /SR(6):=1
ERROR_ACTIVE
ERROR_PASSIVE
REC
RX_ERROR_COUNTER>
[RX_CNT<128] and [TX_CNT<128] /SR(6):=0
SW_RESET

Figure 3. Statechart of the Error Management Logic.

8	Levels of Hierarchy	66	Data-Items
19	Statecharts	51	Events
170	States	18	Conditions

Table 1. Parameters of the DBCAN Statechart Model.

Powerful data manipulation and timing functions are available to simplify design entry. Nevertheless the impact on the generated hardware has to be considered when using such functions. The timeout instruction *tm(en(bus_off),5)*, for example, forces the FSM to leave state *bus_off* six clock periods after entering it. However for each timeout instruction introduced a separate binary counter (3 bits in this case) is instantiated which can blow up the design when used extensively.

Besides the graphical entry of FSMs *ExpressV-HDL*[4] also offers static and dynamic tests of the specified states, actions and transitions. The model can be checked for e.g. deadlock situations, state reachability and nondeterminism at the earliest, possible design stage. In the DBCAN design these analysis methods detected systematic as well as design entry faults and helped to reduce the number of time- and cost-consuming redesign cycles.

For behavioral simulation of DBCAN the interactive simulation environment of *ExpressV-HDL* was used (Fig. 4). A graphical control panel was created in which internal data items (e.g. *transmit/receive buffers, registers, error-*

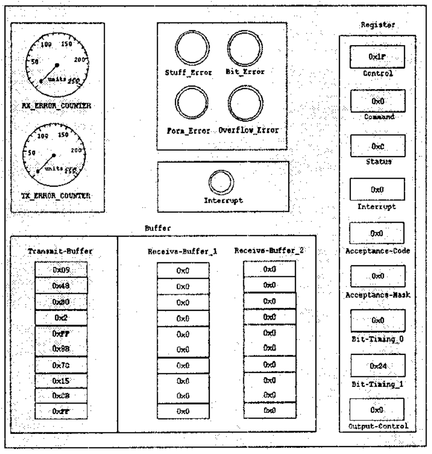

Figure 4. Interactive Simulation Environment.

counters) as well as events and conditions (e.g. *stuff_error, bit_error, interrupt*) were displayed easily. A waveform display is also available for the display of simulation results. During simulation all active states and transitions were highlighted and input data was changed by the user either via the panel or via a batch file. Running several simulations in single step and batch mode DBCAN was intensively tested. A second Basic-CAN controller, which introduced e.g. errors while sending or receiving data frames to or from DBCAN, was simulated by a batch file. Nevertheless only a small subset of all possible situations were reasonable to simulate due to a rather slow simulation execution (approx. 10 minutes for a medium sized data transfer on a Sparc 10 Workstation). An FPGA-based prototype of DBCAN will speed up the tests to real-time, allowing huge test sequences to be executed.

3.2. Synthesis

Starting with the tested DBCAN model *ExpressV-HDL* generated VHDL code suitable for the following synthesis step. No user interaction was necessary for the code generation except the definition of some compiler options such as reset logic type, clocking scheme, state encoding style etc.. Fig. 5 shows a generated VHDL code segment of the previously described *error management logic*. As VHDL is only used as a design entry language for synthesis there is no need to have a deeper knowledge of this hardware description language. For the complete DBCAN description 4256 lines of VHDL code were generated automatically.

Although design synthesis via VHDL allows easy retargeting of the designed system, we used the $0.7\mu m$ CMOS standard-cell process of the final device for prototype synthesis. The resulting prototype was close to the final im-

```
procedure exec_st2EML is
begin
  case st2EML_isin is
    when BUS_ON =>
      if TX_CNT > 255 then
        next_SR(7) <= '1';
        enter_BUS_OFF;
      elsif (SW_RESET /= prev_SW_RESET) then
        next_st2EML_isin <= BUS_ON;
        next_ERROR_STATE_isin <= ERROR_ACTIVE;
      else
        exec_BUS_ON;
      end if ;
    when BUS_OFF =>
      if ((tmENBUS_OFF /= prev_tmENBUS_OFF) and
          not ((ENBUS_OFF /= prev_ENBUS_OFF))) then
        next_st2EML_isin <= WAITFORIMP;
      end if ;
    when RECESSIVE_COUNT =>
      if IF_COUNT = 127 then
        next_IF_COUNT <= 0;
        next_TX_CNT <= 0;
        next_RX_CNT <= 0;
        next_SR(6) <= '0';
        next_SR(7) <= '0';
        next_st2EML_isin <= BUS_ON;
        next_ERROR_STATE_isin <= ERROR_ACTIVE;
      else
        exec_RECESSIVE_COUNT;
      end if ;
    when WAITFORIMP =>
      if CR(0) = '0' then
        next_REC_ERR_COUNT <= 0;
        next_IF_COUNT <= 0;
        next_st2EML_isin <= RECESSIVE_COUNT;
      end if ;
  end case;
end exec_st2EML;
```

Figure 5. VHDL code segment of EML.

plementation because the same synthesis run could be used. Nevertheless the targeted standard-cells had to be mapped onto the basic building blocks of the emulator's FPGAs. For that purpose a conversion library, which was needed for the emulator specific design compilation later on (Fig. 6), was built up and validated by Verilog simulations. As a first approach the design was synthesized straight forward with the *Design Compiler* [10] and a gate-level netlist of DBCAN in EDIF-format (63859 lines, 6390 instances) was extracted.

3.3. Emulation

Emulation is an efficient way to test a prototype of the system under real-time conditions. It is especially useful for reactive real-time systems as DBCAN. Because of DBCANs large amount of interaction with its environment (e.g. bus accesses) simulation is too slow for complete design validation. Due to the speed-up factor of emulation (up to 10^6 compared to simulation) real-time tests in the target system become feasible. Nevertheless, since a configurable network of FPGAs and FPICs is used for logical emulation of the gate-level netlist, the resulting prototype is still

10 times slower than the final hardware system. The signal delays of the used SRAM based FPGAs from *XILINX* heavily depend on the final placement and routing of the devices and therefore are not predictable. So only a prototype which exhibits identical behavior on clock cycle level rather than in terms of absolute timing can be expected.

In contrast to conventional breadboarding techniques emulation reduces the time for generating a prototype significantly. Supported by powerful software tools for partitioning and placement a first prototype of a gate-level netlist can be generated within hours (Table 3) compared to approximate 2 months needed for ASIC fabrication. Moreover concurrent engineering is encouraged by simply spreading the bitstreams to different hard- and software designers, who can download them to their emulators easily.

The *Logic Animator LAS50A* (Quickturn) [8] was used to emulate DBCANs netlist. With the capability of emulating designs up to 50 K gates with an average frequency of 8 to 16 MHz it was sufficient for this application. The basic design steps during emulation are shown in Fig. 6.

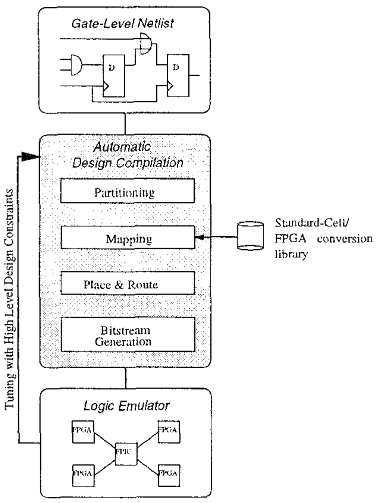
Gate-Level Netlist
Automatic Design Compilation
Partitioning
Mapping
Place & Route
Bitstream Generation
Standard-Cell/ FPGA conversion library
Logic Emulator
FPGA
FPIC
Tuning with High Level Design Constraints

Figure 6. Emulation Design Steps.

At first, the netlist had to be partitioned because it did not fit into one single FPGA. After mapping the design to the available basic building blocks, placement and routing of each used FPGA and FPIC was started. All nodes of the emulated design, i.e. interface pins as well as internal signals and registers are observable through *external interface modules* (EIM). Up to 448 bidirectional signal-pins can be assigned to external connectors of the emulator during the compilation process. As for all control-flow driven designs the state registers were of major interest for the behavior of DBCAN. Therefore they have been made accessible through external connectors for monitoring along with several control- and status registers of the controller. After the bitstream generation for the multiple FPGAs and FPICs the design was ready for downloading.

The design compilation process was fully supported by software. In order to get a first and quick prototype little effort was spent to increase the emulation speed. Consisting of 11824 gates and 736 registers the netlist could be emulated in a first approach with a frequency of 4 MHz, which is one fourth of the frequency of the final system.

The question of how to increase emulation frequency is currently under investigation. For tuning the compilation process tightly coupled logic has to be grouped into a single FPGA and the placement of nets, instances, and devices into high speed I/O logic has to be controlled by introducing priorities during placement and routing.

Totally 14 out of 24 available FPGAs used.			
Gates	Pins	FFs	Tristates
21,3%	89,3%	3,4%	0,3%

Table 2. Average Utilization of the used FPGA Resources.

The analysis of the average resource utilization of the used FPGAs (Table 2) obviously demonstrates that the prototype design is I/O-driven, since almost all pins of the FPGAs are occupied. The remaining 10 percent were intentionally left free to allow incremental changes without re-compilation of the whole design. Currently further improvement in emulation speed is analyzed by adjusting the parameters for resource utilization.

3.4. System Validation

For real-time testing of the DBCAN prototype an experimental CAN-Bus system consisting of two BCAN nodes has been built up (Fig. 7).

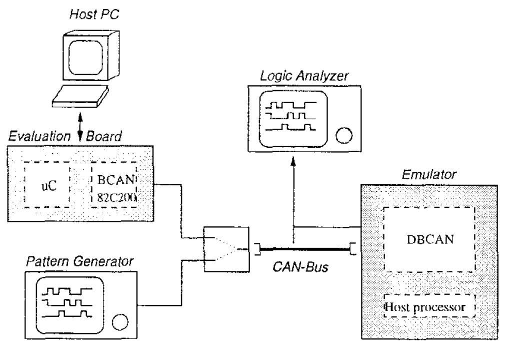

Figure 7. Testsystem with emulated DBCAN.

The emulated version of DBCAN was connected via a CAN-Bus with an of-the-shelf BCAN controller. Communication was established with a data transfer rate of 250 KBit/s according to the controller's limited clock frequency of 4 MHz. The host processor for DBCAN was modeled as an FSM and was emulated together with the CAN controller. The status of the commercial BCAN controller (send-/receive-buffers, status-/control-registers etc.) could be visualized on a host PC. In conjunction with a logic analyzer, which traced bus data as well as internal signal nodes of DBCAN, a functional validation of the prototype was possible. If errors (bit error, stuff error etc.) had to be introduced into the transmission for test purposes, a pattern generator was used instead of the commercial controller.

High emulation speed (real-time) allowed the application of a large amount of input sequences compared to that already covered by simulation and therefore more exhausive design validation became possible. Also, the time-consuming simulation of other system components (e.g. the second BCAN controller) was superfluous because the device under test operated in its real target environment.

4. Results

Considering the time schedule for the prototype generation presented in Table 3, most of the design effort was spent on system modeling and on a first behavioral simulation.

Modeling with Statecharts	2 months
Simulation (behavioral)	3 weeks
VHDL-Code Generation	5 minutes
Synthesis with Synopsys	5 hours
Place & Route for Emulation	5 hours

Table 3. Time schedule for the Prototype Generation.

The executable model we used allowed us to check the design against its specification prior to prototype generation. This revealed several errors and therefore reduced the number of redesign cycles. Once modeled, a first prototype was almost automatically generated by the software tools without detailed knowledge of VHDL and the underlying hardware (FPGA/FPIC) within several hours. Even this first prototype met its specification and operated with an emulation frequency of 4 MHz and a maximal data transfer rate of 250 KBit/s; one fourth of the rate of the anticipated final system. If the achieved performance of the prototype is not satisfying, more detailed knowledge of modeling, synthesizing and emulation will be necessary for emulation speed improvement.

5. Conclusion and Future Work

In this paper the prototype generation of a Basic CAN-Bus controller (DBCAN) has been presented. Starting with modeling DBCAN as a hierarchical finite-state-machine the complete designflow down to the FPGA-based hardware prototype has been investigated.

The feasibility of this rapid-prototyping approach has been proved by demonstrating that a functional hardware prototype can be build up within a few weeks. Redesign iteration loops can be executed even within hours. So the turnaround time of several months for traditional prototyping using breadboarding techniques along with first ASIC prototypes has been reduced significantly.

It was pointed out that the designer can almost be relieved from implementation details and therefore (s)he is able to concentrate on system modeling aspects. This resulted in a qualitatively better prototype and reduced the number of redesign cycles.

Further investigations will concentrate on the optimization of the maximum emulation frequency. More extensive functional tests using the emulated design are planned prior to device fabrication. The prototype of a Full CAN-Bus controller, which has already been modeled and simulated is being developed using the same technique.

References

[1] *Fortschrittberichte VDI: Fachtagung Integrierte Mechanisch-Elektronische Systeme.* VDI Verlag, March 1993.

[2] D. Drusinsky and D. Harel. Using Statecharts for hardware description and synthesis. *IEEE Transactions on Computer-Aided Design*, 1989.

[3] K. Etschberger. *CAN Controller-Area-Network, Grundlagen, Protokolle, Bausteine, Anwendungen.* Hanser Verlag, Muenchen, Wien, 1994.

[4] i-Logix, Inc., Andover. *ExpressV-HDL User Reference Manual V3.1*, 1994.

[5] W. Lawrenz, editor. *CAN Controller-Area-Network, Grundlagen und Praxis.* Huethig Verlag, Heidelberg, 1994.

[6] Philips Semiconductors, Hamburg. *CAN Specification V2.0*, 1991.

[7] Philips Semiconductors, Hamburg. *PCA82C200 Standalone CAN-controller, Data Sheet*, 1992.

[8] Quickturn Design Systems, Mountain View, California. *Logic Animator Design Compilation User's Guide V1.0*, 1994.

[9] J. Stoecker, A. Kirschbaum, I. Aller, B. Breuer, M. Glesner, and H.-L. Hartnagel. Der intelligente Reifen – Zwischenergebnisse einer interdisziplinaeren Forschungskooperation. *Automobiltechnische Zeitschrift (ATZ)*, 97(12):824–832, December 1995.

[10] Synopsys Inc. *Design Compiler Family Reference V3.3a*, 1994.