Complete Formal Hardware Verification of Interfaces for a FlexRay-Like Bus

Christian Müller* and Wolfgang Paul

Saarland University, Computer Science Department,
66123 Saarbrücken, Germany
{cm,wjp}@cs.uni-saarland.de

Abstract. We report the first complete formal verification of a time-triggered bus interface at the gate and register level. We discuss hardware models for multiple clock domains and we review known results and proof techniques about the essential components of such bus interfaces: among others serial interfaces, clock synchronization and bus control. Combining such results into a single proof leads to an amazingly subtle theory about the realization of direct connections between units (as assumed in existing correctness proofs for components of interfaces) by properly controlled time-triggered buses. It also requires an induction arguing simultaneously about bit transmission across clock domains, clock synchronization and bus control.

1 Introduction

Clean formal definitions of time-triggered systems have been given in [11] and [9]. The simple time-triggered systems whose hardware realization is studied in this paper are inspired by the FlexRay standard [5]. They are constructed by coupling several processors by means of bus interfaces to a single real time bus as shown in Figure 1.

A processor together with its bus interface is called an electronic control unit (ECU). We denote by ECU_i the i'th ECU. The hardware of each ECU is clocked by its own oscillator. Oscillators of different ECU's have almost but not exactly identical clock periods τ_i (corresponds to ECU_i). As a consequence of this, timers on different ECU's tend to drift apart and need to be synchronized periodically.

On the bus interface of each ECU_i pairs of send and receive buffers $ECU_i.sb(j)$ and $ECU_i.rb(j)$ with $j \in \{0,1\}$ serve both for the local communication between processors and their local bus interface and for communication between bus interfaces over the bus.

Time-triggered systems work in rounds r each consisting of a fixed even number ns of slots $s \in \{0,\ldots,ns-1\}$ according to a fixed schedule which is identical for each round. The work consists of:

* The author was supported by the German Federal Ministry of Education and Research (BMBF) in the Verisoft project under grant 01 IS C38.

G. Gopalakrishnan and S. Qadeer (Eds.): CAV 2011, LNCS 6806, pp. 633–648, 2011.

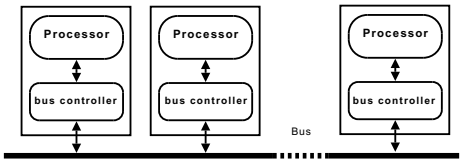


Fig. 1. ECU's Interconnected by a Communication Bus

- Local computation. During slot s each processor can access buffers $sb((s+1) \bmod 2)$ and $rb((s+1) \bmod 2)$ of its bus interface via memory mapped I/O for local computation. We will not consider local computation in this paper.
- Message broadcast. A fixed scheduling function *send* specifies for each slot s the ECU $ECU_{send(s)}$ whose bus facing send buffer $sb(s \bmod 2)$ will broadcast to the bus facing receive buffers $rb(s \bmod 2)$. Observe that in the following slot $s+1 \bmod ns$ these receive buffers face the processors. This allows to overlap local computation with message broadcast. It is a special case of pipelining in the sense of [10].

We refer to the slot s of round r by (r,s). We denote the first local cycle of slot (r,s) on ECU_u by $\alpha_u(r,s)$ and the last cycle by $\omega_u(r,s)$. The values $\alpha_u(r,0)$ and $\omega_u(r,ns-1)$ at boundaries of rounds are determined in a non-trivial way by clock synchronization. The standard local length of a slot is cs cycles. How many cycles of a slot have locally passed is tracked by local counters. An 'interior' slot boundary $\alpha_u(r,s) = \omega_u(r,s-1)$ occurs, if the local counter is 0 mod cs.

We denote by ECU_u^j the state of ECU_u at hardware cycle j. The essential correctness statement of the bus interfaces whose formal proof can be found at [1] is then a very simple and clean statement about message transfer:

Theorem 1. $\forall\, u,r,s:\ ECU_{send(s)}^{\alpha_{send(s)}(r,s)}.sb(s \bmod 2) = ECU_u^{\omega_u(r,s)}.rb(s \bmod 2)$

A hardware realization of such a bus interface has obviously to deal with the following 5 problems:

1. Definition of a hardware model with multiple clock domains.
2. Bit transfer across clock domains. As setup and hold times for registers cannot be guaranteed across different clock domains, i.e., here over the bus, the sender puts each message bit for $n > 1$ cycles on the bus. The receiver will try to sample $m \leq n$ of these 'hardware' bits roughly in the middle of the n bits.[1] Situations where hardware bits are incorrectly sampled from the bus due to missed set up or hold times have to be dealt with. Note that in such situations the receiving registers do not necessarily sample bits in an unpredictable way. They can also become metastable.[2] Registers that are not clocked can stay metastable for very many cycles.

[1] The FlexRay standard requires $n = 8$, $m = 5$ and a majority vote on the sampled bits. This allows the correction of certain bit errors on the bus.

[2] They hang at the voltage between the thresholds recognized as 0 and 1.

3. Message transfer across clock domains. If a message $m[0 : \ell - 1]$ consisting of ℓ bytes is transmitted, then the sender inserts at the start of each byte $m[i]$ so called *sync edges* SE into the message. It also inserts a *message start sequence* MS and a *message end sequence* ME.[3] Thus, message m is transformed by the sender into:

$$f(m) = MS \circ SE \circ m[0] \ldots SE \circ m[\ell - 1] \circ ME$$

and then each bit of $f(m)$ is put n times on the bus. The purpose of the sync edges is to permit the receiver a *low-level clock synchronisation*. Because sync edges occur at regular intervals, the receiver knows when to expect them in the absence of clock drift. If a sync edge before byte $m[i]$ occurs 1 receiver cycle earlier/later than without clock drift, then the receiver knows that his clock has slipped/advanced against the sender clock and adjusts the cycles when it samples the m hardware bits belonging to the same n copies of a message bit accordingly.

The final Theorem 1 we aim at is clearly a theorem about message transfer using such a mechanism. Hardware devices performing such a transfer are called *serial interfaces*. Correctness theorems about serial interfaces assume a single sender in one clock domain directly connected by a wire to a single receiver in a second clock domain.

4. Control of bus contention. During each slot s there must be a *transmission window* where for all ECU's the local timers indicate that they are in slot s. During this window $ECU_{send(s)}$ broadcasts a send buffer content and all other ECU's stay off the bus. This is achieved in the following way: (i) Let $D = Q + d + 1$ where Q is the maximum difference of local cycle counts between clock synchronizations, d is the pipeline depth of the transmission pipeline and the 1 accounts for effects of cross domain clocking (see Schedule Constraint 2 of [10]). Then the sender $ECU_{send(s)}$ begins transmission only D cycles after the local start of slot s and finishes transmission D cycles before the start of the next slot. (ii) All other ECU's stay off the bus during the complete slot s.

   Note that during the transmission window of slot s the bus acts for each ECU_i like a direct wire between $ECU_{send(s)}$ and ECU_i. Thus, correctness of serial interfaces can be applied during this window. Note however that bus contention control hinges on clock synchronization.
5. Clock synchronization. At the start of each round ECU's exchange synchronization messages in order to synchronize clocks according to some protocol. Non-trivial protocols based on Byzantine agreement are used if one wants to provide fault tolerance against the failure of some ECU's. Without fault tolerance a single sync message broadcast from a master ECU suffices at the start of each round. Note however that sync messages need to be transferred. The natural vehicle for this transfer is the bus. Thus, clock synchronization hinges on message transfer (at least for the synchronization messages),

[3] Along the lines of the FlexRay protocol for instance, we use falling sync edges 10 before each byte, 01 as start sequence and end sequence.

message transfer on bus contention control and bus contention control on clock synchronization. Any theorem stating in isolated form the correctness of clock synchronization, bus contention control or message transfer alone must use hypotheses which break this cycle in one way or the other. If the theorem is to be used as part of an overall correctness proof, then one must be able to discharge these hypotheses in the induction step of a proof arguing simultaneously about clock synchronization, bus contention and message transfer. A paper and pencil proof of this nature can be found in [7].

The remainder of this paper is organized as follows. In Section 2 we discuss hardware models with multiple clock domains and develop some basic theory about the operation of buses spanning multiple clock domains. In Section 2.3 and 3 we review related work about the verification of serial interfaces and clock synchronization algorithms. In our presentation we highlight situations, where existing proofs assume direct connections between units that need to be realized by properly controlled buses in the overall proof. Section 4 outlines the overall correctness proof. It contains a quite involved induction hypothesis (Theorem 3) and a quite subtle argument that 'nothing happens' between the end of the message window of the last slot of round $r-1$ and the transmission of the synchronization message of round r. We also present some details about the mechanization of the proof. In Section 5 we conclude and discuss future work.

2 Models for Hardware with Multiple Clock Domains

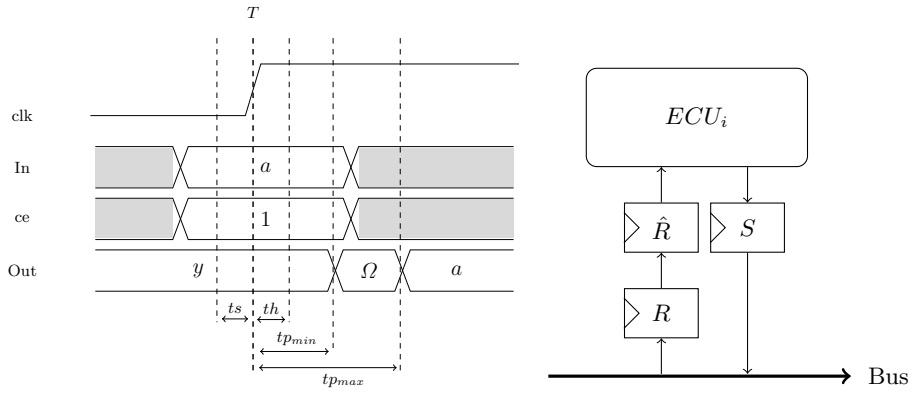
2.1 The Detailed Model

The obvious *detailed* model for this purpose is obtained by formalization of data sheets for hardware components as in [12]. The logic has three values 0, 1 and Ω. The latter value models any voltage between the thresholds recognized as 0 and 1. Signals are mappings from real time $\mathbb{R}$ to $\{0, 1, \Omega\}$; thus, $I(t)$ denotes the value of signal I at real valued time t. This allows to give algebraic definitions for detailed timing diagrams of circuits consisting of gates, registers and memories. The part of this model relevant for 1-bit registers with data input signal In, data output signal Out and clock enable signal ce has as parameters setup time ts, hold time th, as well as minimum and maximum propagation delays tp_{min} and tp_{max}. It is easy to specify that setup and hold times, e.g., for the data input are met for a clock edge at time T:

$$\exists\, a \in \{0,1\} : \forall t \in [T - ts, T + th] : In(t) = a$$

If we also assume that setup and hold times are met for the clock enable signal ce at time T and $\forall t \in [T - ts, T + th] : ce(t) = 1$, then the output of the modeled register i) does not change before $T + tp_{min}$, ii) becomes undefined in $(T + tp_{min}, T + tp_{max})$ and iii) assumes the new value $In(T)$ from $T + tp_{max}$ until the next clock edge say at time $T + \tau_i$:

$$Out(t) = \begin{cases} Out(T) & : t \in (T, T + tp_{min}] \\ \Omega & : t \in (T + tp_{min}, T + tp_{max}) \\ In(T) & : t \in [T + tp_{max}, T + \tau_i] \end{cases}$$



(a) Update of an Analog Register (b) Connection of an ECU to Bus

Fig. 2.

This behaviour is also illustrated in Figure 2(a).

This simple part of the definition which models completely regular clocking has an important consequence. Imagine we have $In(T) = Out(T)$, i.e., we clock the old value again into the register. Then in the digital abstraction the output is constant in two consecutive cycles. In the detailed hardware model however (and in reality) we have a possible spike $Out(t) = \Omega$ for $t \in (T + tp_{min}, T + tp_{max})$. If we want to guarantee, that the output really stays stable ($Out(t) = Out(T)$ for $t \in (T, T + \tau_i)$) , we need to disable clocking at edge T: $ce(t) = 0$ for $t \in [T - ts, T + ts]$. If clocking is properly enabled but setup or hold time is violated or the input is undefined, then after the maximum propagation delay the output stays 0, 1 or undefined (the latter case models metastabilty):

$$\exists a \in \{0, 1, \Omega\} : \forall t \in [T + tp_{max}, T + \tau_i] : Out(t) = a$$

Metastabilty is highly unlikely to occur. That a metastable value in one register R which is clocked into a second register $\hat{R}$ results in the metastability of $\hat{R}$ too is so unlikely that one models it as impossible.[4] The result is an unpredictable digital value:

$$\exists a \in \{0, 1\} : \forall t \in [T + tp_{max}, T + \tau_i] : \hat{R}(t) = a$$

Using minimal and maximal propagation delays of gates and access times of memories one can complete this model in a straightforward way. Correctness theorems about any hardware - with multiple clock domains or not - should in the end hold in this one detailed model.

We will denote by single capital letters, e.g., R_u digital registers of clock domain u; we denote by $In_{R,u}$ their analog input signals and by $Out_{R,u}$ their analog output signals in the detailed model.

[4] Using two subsequent registers to avoid metastability is a common technique in hardware design.

2.2 The Hybrid Model and the Bus

Fortunately, in our case we can restrict the use of the detailed model to the part of the hardware, where clock domain crossing occurs. This portion consists of the bus, the send register S and the receiver register R as depicted in Figure 2(b). The remaining hardware is partitioned into clock domains, one for each ECU. In each clock domain u (clock domain of ECU_u) we can abstract from the detailed model local digital hardware configurations h_u with, e.g., register components $h_u.R \in \{0,1\}$, hardware cycles $i \in \mathbb{N}$, configuration h_u^i during cycle i in the following way. We couple local cycle numbers i on ECU_u (hence, in clock domain u) with the real time $e_u(i)$ of the i'th local clock edge on ECU_u by setting for some constant c_u: $e_u(i) = c_u + i \cdot \tau_u$. This edge starts local cycle i on ECU_u. For $t \in [e_u(i) + tp_{max}, e_u(i) + \tau_u]$, i.e., after the propagation delay the output $Out_{R,u}(t)$ of register R on ECU_u is stable for the rest of local cycle i of ECU_u. We abstract this value to the digital register value:

$$h_u^i.R = dig(Out_{R,u}, e_u(i+1))$$

The function $dig(s,t)$ digitizes the output of signal s at real valued time t: $dig(s,t) :=$ **if** $s(t) \neq \Omega$ **then** $s(t)$ **else** x, for some $x \in \{0,1\}$.

If we now define a hypothesis 'Correct detailed timing analysis' stating that for all local cycles setup and hold times for register inputs are met[5], then one can show that the hardware configurations h_u^i we just abstracted are exactly the configurations one would get by applying the transition function δ_H of ordinary digital hardware models

Theorem 2. *Assume correct detailed timing analysis. Then:* $\forall i : h_u^{i+1} = \delta_H(h_u^i)$.

This justifies the use of ordinary digital logic within clock domains and restricts the use of the detailed model to the boundaries between the domains, in our case the bus.

2.3 Transmitting Bits across the Bus

We use two lemmas from Schmaltz [12] dealing with bit transmission across clock domains in the detailed register model. Schmaltz has formally verified the bit transmission correctness between two directly linked digital registers with different clocks.

To introduce these lemmas formally we need one more definition. Let R and S be two registers from two different clock domains u and v, respectively; they are interpreted using the detailed register model. Assume the clock enable signal of the send register S is active in cycle c. By the definition of the detailed register model, the output of S will change right after $e_v(c) + tp_{min}$. We want to know at which minimal cycle the receive register R will 'notice' the change of its input signal changing its own content. We call such cycle the *next affected cycle* and define it as: $cy_{u,v}(c) = min\{\ x \mid e_v(c) + tp_{min} < e_u(x) - ts\}$. Thus, the receiver's

[5] Summing propagation delays along appropriate paths.

next affected cycle of sender's cycle c is the first cycle, whose set up time lies right after the output change of S.

Lemma 1 (Low-Level Bit Transmission Correctness). *Assume that the clock enable signal of register R is always active, and the analog input of R is the analog output of S during n sender cycles starting from some cycle c:*

$$\forall i \in [e_v(c) + tp_{min}, e_v(c+n)] : In_{R,u}(t) = S_{Out,v}(t)$$

If S samples a stable input value $\mathcal{V}$ at cycle c and then clocking of S is disabled for n subsequent cycles, then R will sample the new output of S for at least $m+\sigma$ cycles starting from the next affected cycle $\xi = cy_{u,v}(c)$:

$$\forall i \in [0, m-1] : R^{\xi+i+\sigma} = \mathcal{V}, \text{ for } \sigma \in \{0, 1\}$$

Note that $m \leq n$ due to possible clock drifts of the register clocks. The additional cycle σ (delay) may arise if set up or hold times are violated.[6] The direct linking of registers should be considered as an abstraction of the bus, when there is no bus contention. Moreover, Schmaltz extended Lemma 1 to a high-level message transmission correctness for a concrete send and receive units obeying the message protocol mentioned in Section 1.

Lemma 2 (High-Level Message Transmission Correctness). *Let su be a send unit of ECU_v; let ru be a receive unit of ECU_u. Let R be the input register of ru and S the output register of su. Let L be the length of a transmitted message in bytes. Let ts be the length of a message transmission in local cycles. Let $bc_v(i)$ be the cycle, when the send unit su starts the broadcast of the i'th byte. Let $cy_{u,v}(c) = \xi$ for a receiver cycle ξ. Assume that:*

1. *su starts a message broadcast in cycle c;*
2. *ru is idle in cycle ξ.*
3. *send and receive registers are directly connected for ts cycles:*
   $\forall t \in [e_v(c) + tp_{min}, e_v(c + ts - 1)] : In_{R,u}(t) = S_{Out,v}(t)$

Then, every byte of the transmitted message is transmitted correctly from the send buffer $su.sb$ to an intermediate byte buffer $ru.rByte$:

$$\forall\ i \leq L\ :\ su^{bc_v(i)}.sb[i] = ru^{cy_{u,v}(bc_v(i))+80+\gamma}.rByte,\ \text{with } \gamma \in \{-1, 0, 1\}$$

On the left side, $su^{bc_v(i)}.sb[i]$ denotes the content of the i'th byte in send buffer $su.sb$ (slot is not fixed, so sb can be instantiated by any of two buffers) of send unit su at cycle $bc_v(i)$. On the right side, we have the content of a buffer $ru.rByte$ of the receive unit ru at the next cycle $cy_{u,v}(bc_v(i))$ affected by the transmission of the i'th byte, plus $80 + \gamma$. That is, due to clock drift, the entire transmitted byte will be successfully sampled either in 79 cycles after the next affected cycle, or in 80/81 cycles. This Lemma was proven by model checking of control automata of su and ru, and by interactive combining of these results using Lemma 1.

[6] This happens, if $e_v(c) + tp_{min} < e_u(\xi) - ts < e_v(c) + tp_{max}$.

2.4 Modeling the Bus

We model the outputs of send registers S_v facing the bus as open collector output. In this case the bus computes the logical 'and' of the signals put on the bus:

$$\forall t : bus(t) = \bigwedge_v Out_{S,v}(t)$$

The intended use of the bus is to simulate for all slots s the direct connection of a sender register $S_{send(s)}$ to the receiver registers R_u on the bus during the transmission window $W(s) \subset [e_u(\alpha(r,s)), e_u(\omega(r,s))]$[7] of slot s, because this permits to apply results about clock domain crossing bit transmission for pairs of directly connected senders and receivers. Obviously, this is achieved by keeping $Out_{S,u}(t) = 1$ for all t in the window and all $u \neq send(s)$.

Lemma 3 (Absence of Bus Contention). *If for all $u \neq send(s)$ and for all $t \in W(s)$ we have $Out_{S,u}(t) = 1$, then $\forall t \in W(s) : bus(t) = Out_{S,send(s)}$*

While this lemma is trivial, showing the hypothesis requires not only to show that the S_u have constant value 1 in the digital model for $u \neq send(s)$. One has also to establish the absence of spikes by showing (in the digital model) that clocking of these registers is disabled in the transmission window. This depends on correct schedule execution, which, in turn, depends on the absence of bus contention at the previous synchronizations, etc.

2.5 An Alternative Model

A model more suited for model checkers than the hybrid model above has been proposed and used in [4]. This clock model is based on so-called *timeout automata*. The progress of global time is enforced cooperatively by sender and receiver clocks. This model deals with metastability but does not model the set up and hold times explicitly. The clock modeling is partially protocol-dependent. Since the sender's clock progress depends on the receiver's clock progress, it is not fully clear how to extend this model to several receiver clocks. Unfortunately, no discussion was provided about the gap between the given stack of abstractions and modeling of actual hardware (counterpart to our Theorem 2).

3 Related Work and Results Used

In [9], Pike has presented several results. One part of his work related to this paper was a corrected and significantly extended version of Rushby's formalism [11], which allows to verify time-triggered systems by abstracting them to synchronous protocols. He specifies timing constraints, which a schedule of a time-triggered protocol has to fulfill, to form in its system run so-called 'cuts'. These cuts are

[7] $W(s)$ is the time segment in global time where all ECU's are locally in slot s; it should be long enough to transmit a message.

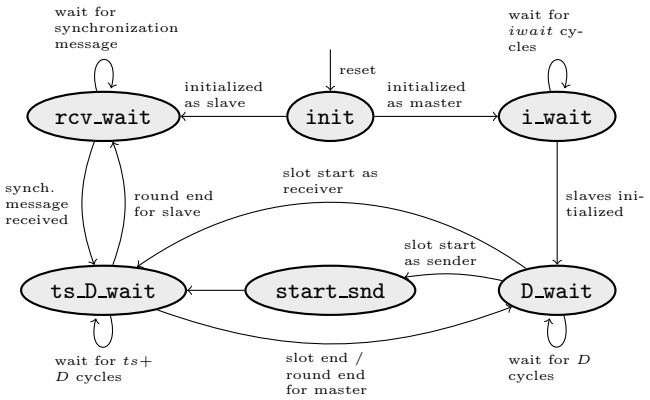


Fig. 3. Scheduler Automata

time segments, where the state of the time-triggered system can be related to a state of its synchronous counterpart. The correctness of the time-triggered schedule would then follow from the correctness of the synchronous protocol. He also applies this approach to verify the schedule timings of two protocols of the SPIDER bus architecture: Clock Synchronization and Distributed Diagnosis.[8] However, it remains unclear how exactly the timing properties of hardware implementations of these protocols were derived and mapped to the formal model. Moreover, the proposed technique assumes even in the proof of timing properties of a Clock Synchronization protocol already synchronized clocks initially. That is, to extend the proof to *all round correctness* one needs to use the proposed proof as an induction step. However, to show the initial clock synchronization the startup correctness might become necessary.

In [8], Pfeifer formally analyzes two fault-tolerant algorithms implemented in the Time-Triggered Protocol TTP/C: Group Membership and Clock Synchronization. Both algorithms were analyzed, using a hand-derived mathematical specification of the TTP/C protocol. Although, the algorithms have a circular dependency on each other, Pfeifer has analyzed them in isolated form and on different levels of abstraction. He also introduces an abstract principle how to combine both proofs resolving the dependency.

Böhm has used Lemma 1 to verify the correct schedule execution for directly wired bus controllers [3]: one master and n slaves. The master ECU always sends a message in slot 0. The first bit of this message serves as a synchronization signal. The scheduler used in the connected ECU's is the main control block of the bus controller. It computes the internal state of the controller and counts passed slots. The schematic representation of the scheduler state automaton is depicted in Figure 3. After the initial reset signal, all ECU's are in state `init`. After all configuration registers are written, the operating system running on top of the processor initiates a special signal, which forces all ECU's to change

[8] Note that no correctness of the synchronous version of these protocols was provided.

either to state `i_wait` if the ECU was configured as a master ECU, or to state `rcv_wait` otherwise. The master waits in state `i_wait` for a certain amount of cycles *iwait*. Assuming that all ECU's are started roughly at the same time, it should be guaranteed that after the master's initialization and additional *iwait* cycles all slaves are initialized and are in state `rcv_wait`. The size of *iwait* can be estimated by industrial worst case execution time analyzers [7]. In state `rcv_wait` slaves are waiting for a synchronization message after the initialization and after the end of each round. In state `ts_D_wait` each ECU is waiting for $ts + D$ cycles, which is a sum of transmission length (ts) and offset D before the end of a slot. In this state an ECU is sending or receiving, depending on its role in the current slot. Then a slave ECU either switches back to `rcv_wait` if the maximal number of slots in a round is reached, or to state `D_wait`. In `D_wait` all ECU's are waiting for D hardware cycles. This is a waiting time (cf. Section 1) before each new slot, which is necessary due to clock drift to guarantee that all ECU's have finished the previous slot. Afterwards, each ECU enters a new slot. If it enters a slot as a sender, the scheduler sends a broadcast signal to the send unit going through state `start_snd`. Otherwise, it switches directly to state `ts_D_wait`. The master ECU acts as a sender in the first slot of each round. It goes through state `start_snd` sending a new synchronization message to the bus. Afterwards, it switches as a receiver between states `D_wait` and `ts_D_wait` during $ns - 1$ slots.

We say synchronization happens i) on the master at the state transition from `D_wait` to `start_send`; at this point the local timer of master has value D. ii) on a slave at the state transition from `rcv_wait` to `ts_D_wait` which is triggered by the synchronization message sent by the master. On this transition the slave sets its local timer to D.

W.l.o.g., we assign to the master ECU the index 0: ECU_0. We can now define the missing slot boundaries. On the master all slots have length cs: $\alpha_0(r, s) = \omega_0(r, s-1) = \alpha_0(r, s-1) + cs$. The master executes its schedule statically. On slaves u slot $(r-1, ns-1)$ ends and slot $(r, 0)$ starts when the synchronization message is received: $\alpha_u(r, 0) = \omega_u(r-1, ns-1) = cy_{u,0}(\alpha_0(r, 0) + D)$.

We define that the system is ready for synchronization at cycle c of the master, when:

- in local cycle c the master is in state `D_wait`, its local timer is $D - 1$ and its serial interface is idle and, hence, ready for data transmission;
- for $u \neq 0$ in local cycle $cy_{u,0}(c)$, the receiving ECU_u is in state `rcv_wait`, its serial interface is idle, and, thus, ready for data sampling and the internal pipeline of the receive units has sampled Ones from the bus in the last e local cycles for some fixed constant $e < d$ (thus no spurious sync messages are already in the pipe).

By assuming that synchronization message is transmitted via *dedicated* direct connections between the master and the slaves, Böhm does not need to assume bus contention control for the transmission of the synchronization signals and, thus, he is in a position to do an induction over all slots s of all rounds r. However in his main result - that we use - he assumes that at the start of some round r ECU's are ready for synchronization and then he argues only about the slots of

this round r. He does not argue that at the end of the round the system is ready for synchronization again. We use the following result from [3].

Lemma 4. *Assume: 1. the system is ready for synchronization and the master is in cycle c; 2. the output of master's register S is the input of all R_u registers during 8 master cycles: $\forall\ t \in [e_v(c), e_v(c+7)] : In_{R,u}(t) = Out_{S,v}(t)$.*

Then, the master initiates the synchronization message, and all directly connected slaves and the master itself will start the execution of the fixed schedule consisting of ns slots. Each slot on each ECU starts before and ends after the message transmission window defined as transmission time of the sender. After cs local cycles of slot $ns-1$ the master is in state `D_wait` *and the slaves are in state* `rcv_wait`*. Formally:*

$\forall$ *slot s of some round r : u is receiver in slot s* $\rightarrow$

1. *synchronization happens at round r* –
   slave u receives the synchronization message at cycle $cy_{u,0}(c) + \sigma + d$
2. $W(s) \subset [e_u(\alpha_u(r,s)), e_u(\omega_u(r,s))]$
   where $W(s) = [e_{send(s)}(\alpha_{send(s)}(r,s) + D), e_{send(s)}(\omega_{send(s)}(r,s) - D)]$
3. *ECU_u with $u \neq send(s)$ is in state* `rcv_wait` *in cycle $\alpha_u(r, ns-1) + cs$, and $ECU_{send(s)}$ is in state* `D_wait` *in cycle $\alpha_{send(s)}(r, ns-1) + cs$.*

Remember that d is the depth of transmission pipeline, i.e., the delay of the path of the signal from the scheduler of the sender to the scheduler of the receiver. That the synchronization message is received by the slaves was proven by applying Lemma 1. After synchronization is achieved the proof of the lemma boils down to a statement about the values of the local counters and can be derived with a very high degree of automation. The timing properties of the schedule hardware were proven interactively by multiplying local times with the corresponding τ.

However, we can only apply Lemmas 1, 2 and 4 in our generalized semantic, consisting of n ECU's interconnected by a bus, if we argue about the absence of bus contention during all signal transmissions on the bus. We have to justify the bus abstraction, which will be assumed using the directly wired send and receive registers. This abstraction will be discharged in the next section.

4 Overall Proof

The correctness of message transmission between all ECU's of our bus system can be roughly split in two parts: (1) the bus value correctness during a message transmission (absence of bus contention), and (2) the high-level message transmission. The first part depends only on the correctness of the scheduler unit and low-level bit transmission between ECU's over the bus. The high-level message transmission relies on the bus value correctness, low-level bit transmission correctness and the correctness of send and receive units executing the message protocol. Thus, formally verifying the first property, we provide a bus architecture, which can be instantiated with an arbitrary message protocol. In

Section 4.2 we present a proof sketch of the first property. In Section 4.3 we outline the correctness proof sketch of the high-level message transmission, consolidating the results of Section 4.2 with results achieved in previous verification efforts explained in Subsection 2.3 and Section 3.

4.1 Global Assumptions

To verify the automotive bus controller, the following axioms were assumed and are implicitly used in every theorem below:

1. Clock periods deviate by at most 0.38%;[9]
2. Slot 0 starts on the master immediately after leaving the `i_wait` state;
3. When master enters slot 0, all slaves are initialized and are waiting for the synchronization;
4. ECU's are configured in such way, that in every slot exactly one ECU is a sender.

Note that we omit assumptions of technical nature here, which are not relevant for the communication protocol itself, like assumption that the processor doesn't change it's configuration registers during a system run, etc.

4.2 Proof Sketch of the Bus Value Correctness

While previous lemmas have assumed in slot s a direct connection between a slave u and the master ($In_{R,u}(t) = Out_{S,send(s)}$), the generalized model extended by the bus, models the analog input of every R_u register as output of the bus:

$$\forall \text{ ECU } u : \forall t : In_{R,u}(t) = bus(t) \tag{1}$$

To use previous results for an overall message transmission correctness, we have to show that $bus(t)$ can be substituted by $Out_{S,send(s)}$ during the transmission time $W(s)$ of every slot s.

Theorem 3 (Bus Value Correctness). *For all rounds r holds:*

1. *the system is ready for synchronization at cycle $\alpha_0(r, 0) + D$ of the master;*
2. *during the entire message transmission time in every slot s of round r the bus value is equal to the analog output of the send register of the sender:*

$$\forall t \in W(s) : bus(t) = Out_{S,send(s)}(t)$$

Before we sketch the proof of Theorem 3, we list three helper lemmas, which do not depend on the cyclic argumentation necessary for the proof of Theorem 3.

Lemma 5. *After the initialization the system is ready for synchronization and the master begins a message broadcast in cycle $\alpha_u(0, 0) + D$.*

[9] This constant is derived from the concrete parameters of our implementation. Note that FlexRay standard assumes maximal deviation of size 0.15%.

This lemma is proven by construction of the scheduler and the control logic of configuration registers. Using mostly a model checker, we have formally verified that after a reset each ECU reaches state `rcv_wait` or `i_wait` according to its status (master/slave). Then we *assume* (cf. Global Assumption 3) that if the master leaves the `i_wait` state, all slaves are in the state `rcv_wait`.

Lemma 6. *Let the send unit of the serial interface of ECU_u be idle during time interval I. Then during I the analog output of the send register of ECU_u is 1:*

$$\forall t \in I : Out_{S,u}(t) = 1$$

This lemma was proven by showing that the send register S is only clocked if the send unit is not idle. We define the time interval of local slot (r, s) of ECU_u as: $slot_u(r, s) = [e_u(\alpha_u(r, s), \omega_u(r, s)]$ and show the next lemma.

Lemma 7. *Let $u \neq send(s)$ and let r be any round. Then the send unit of ECU_u is idle during $slot_u(r, s)$.*

A simple induction about the scheduler automaton shows, that at the start of a slot the send unit is idle. The send unit is only started in state `start_send`. But if ECU_u is not the sender, then the automaton is during the slot only in states $\neq$ `start_snd`. We are finally ready to argue by induction simultaneously about synchronization and bus contention control.

Proof (of Theorem 3). By induction on rounds r. *Base case,* $r = 0$. We first show Claim 1 for $r = 0$. After startup the send units of all slaves are idle and the system is ready for synchronization by Lemma 5.

Next we show Claim 2 for round $r = 0$. We first discharge hypothesis 2 of Lemma 4 for $c = \alpha_0(r, 0) + D$ stating that the bus behaves during the transmission of the sync message like a direct connection between the master and the slaves. A slave ready for synchronization is in state `rcv_wait` and the send unit of its serial interface is idle. In 9 cycles it cannot reach state `start_send`. Using Lemma 6 with $I = [e_0(c), e_0(c + 7)]$ we conclude that ECU_u transmits Ones on the bus during I. By Lemma 3 we get $In_{R,u}(t) = Out_{S,0}(t)$ for all $t \in I$. We are ready to apply Lemma 4. By Claim 1 of Lemma 4 we can conclude, that synchronization takes place and thus slot 0 of round 0 starts on all ECUs. Thus from now on we can use lemmas that argue about slots in round 0.

Consider any slot s of round 0 and any $u \neq send(s)$. By Lemma 7 ECU_u stays off the bus during $slot_u(0, s)$. By Claim 2 of Lemma 4 we have $W(s) \subset slot_u(0, s)$. By Lemma 3 the bus acts like a direct connection between $Out_{S,send(s)}$ and $In_{R,u}$ during $W(s)$ and we have shown Claim 2 of Theorem 3 for round $r = 0$.

Induction step, $r - 1 \rightarrow r$. We assume the theorem holds for round $r - 1$ and show that it holds for round r. We start with Claim 1 for round r. By Claim 1 of the induction hypothesis the system is ready for synchronization at cycle $\alpha_0(r - 1, 0) + D$ of the master. By Claims 1 and 2 of the induction hypothesis for slot 0 of round $r - 1$ Hypothesis 1 and 2 of Lemma 4 hold. By Claim 3 of Lemma 4 each slave u is in state `rcv_wait` in local cycle $\alpha_u(r - 1, ns - 1) + cs$ and the master is in state `D_wait` in local cycle $\alpha_0(r - 1, ns - 1) + cs$.

We show a subtle technical lemma about the time when slaves are waiting for the synchronization signal of the master at the end of rounds. Note that for slaves u the start cycles $\alpha_u(r,0)$ are only defined after we have shown that synchronization has occurred, thus we cannot use them yet. Fortunately for the master $\alpha_0(r,s)$ is always defined.

Lemma 8. *Let $u \neq 0$ be the index of any slave. Let*

$$t \in [e_u(\alpha_u(r-1, ns-1) + cs, e_u(cy_{u,0}(\alpha_0(r,0) + D))$$

be a time when slave u waits for the sync signal from the master. Then the slave observes an idle bus at time t: $bus(t) = 1$.

This lemma is proven by contradiction. We fix the first time point t between the last slot of a round $e_u(\alpha_u(r-1, ns-1) + cs)$ and the start of the next round $e_u(cy_{u,0}(\alpha_0(r,0)+D))$, where the bus value is not equal '1'. By bus construction there is at least one ECU producing bus activity at t. Since t lies *before* the start of a new round, the disturbing ECU cannot be master, since there was no synchronization message yet. If the disturbing ECU is a slave, we show that it should be in state `rcv_wait`, because there was no bus activity before t, hence no synchronization message was received by any slave. Since no one is sending, we use Lemma 6 and the bus construction to show that its value is the idle value '1' which contradicts to our assumption.

Lemma 8 implies that all slaves stay in state `rcv_wait` until local cycle $cy_{u,0}(\alpha_0(r,0)+D))$. By construction of the schedule automaton the master stays in state `D_wait` until local cycle $\alpha_0(r,0)+D$. By construction of the serial interfaces the interface of an ECU in state `rcv_wait` is idle. This applies here for the slaves. By the construction of the hardware, non-idle send units of serial interfaces occur only on ECUs in state `ts_D_wait`. Thus, at cycle $\alpha_0(r,0)+D$ of the master the system is ready for synchronization, Claim 1 of Theorem 3 is shown for round r and we can argue about the slots in round r also for the slaves. The proof of Claim 2 for round r now proceeds along the lines of the proof for round 0. It is somewhat simpler because Hypothesis 2 of Lemma 4 follows directly from Lemma 7. □

4.3 Message Transmission Correctness

After we have proven the correctness of Theorem 3, we can use it to ensure the absence of bus contention during all transmissions of all rounds. We discharge Hypothesis 1 and 2 of Lemma 2 for cycle $c = \alpha_{send(s)}(r,s) + D$ (similarly to the argumentation in Theorem 3) by showing that if a sender ECU_i starts a message transmission at a cycle (r,s), then the receive unit of every receiver will be idle at the corresponding next affected cycle. Hypothesis 3 of Lemma 2 follows from Theorem 3. We also show that all bytes written to the intermediate byte buffer $rByte$ of the receive unit will be correctly written to the receive buffer.

Lemma 9. *All bytes written to the intermediate buffer $ru_u.rByte$ in slot s will eventually be written to the receive buffer $ru_u.rb$, and will stay there unchanged until the end of a slot:*

$$\forall\ i \leq L\ : ru_u^{cy_{u,v}(bc_v(i))+80+\gamma}.rByte = ru_u^{\omega_u(r,s)}.rb(s \bmod 2)[i]$$

We also show stability of the active send buffer content. This is possible due to double buffer construction and restricting of the processor access to the buffer $sb(s \bmod 2)$ during a message transmission.

Lemma 10. *The content of the send buffer remains stable during slot s:*

$$\forall\ i \leq L\ : su_v^{\alpha(r,s)}.sb(s \bmod 2)[i] = su_v^{bc_v(i)}.sb(s \bmod 2)[i]$$

Finally, using Lemmas 2, 9 and 10 we can show the main message transmission correctness during all slots of all rounds stated in Theorem 1.

5 Conclusion and Future Work

We presented recent results on verification of an automotive time-triggered real-time bus system, which was inspired by the FlexRay standard. The network consists of electronic control units interconnected by a bus. We have verified the startup routine, the clock synchronization, the correct TDMA scheduling and a low-level bit transmission. Finally, consolidating these results, we have shown the correctness of recurrent message broadcasts during a system run. In contrast to most of previous research, we not only show the algorithmic correctness of protocols, but we also provide justified models to link these protocols to a concrete gate-level hardware.

The verification was carried out on several abstraction layers using a combination of an interactive theorem prover Isabelle/HOL supported by the model checking technique [14]. The ECU implementation has been automatically translated to Verilog [13] directly from formal Isabelle/HOL hardware descriptions and has been synthesized with the Xilinx ISE software. The implementation which consists of several FPGA boards interconnected into a prototype of a distributed hardware network was reported in [6].

During the presented verification work, several bug were discovered in the hardware implementation, as well as in previous verifications. For example, Lemma 1 has originally assumed a permanent connection between the send and receive registers. This abstraction cannot be justified in a bus architecture with multiple senders. Moreover, Lemmas 4 and 2 have assumed that the second receive register $\hat{R}$ contains value '1' in cycle $cy_{u,v}(c) + 1$. This cannot be shown, in the case when the timing constraints of the first receive register R are not met and the metastable value sampled in cycle $cy_{u,v}(c)$ flips to 0 in the next cycle. Furthermore, in the original proof of Lemma 2 used an assumption like: $cy_{u,v}(k + 80 \cdot i) = cy_{u,v}(k) + 80 \cdot i$ which is in general not true.[10]

As part of the future work we see an extension of the presented controller by fault-tolerance features [2]. For example, supplying each ECU with a bus guardians should be easy, by taking the same scheduler with independent clock and slightly modified timing parameters. Moreover, due to redundancy in the

[10] The proof was fixed by Schmaltz on our demand.

message protocol, the bus controller is already fault-tolerant against signal jitter. The computing and verifying of its maximal fault assumptions remains as future work.

References

1. Formal proof of a FlexRay-like bus interface, http://www-wjp.cs.uni-sb.de/~cm/abc.html
2. Alkassar, E., Boehm, P., Knapp, S.: Correctness of a fault-tolerant real-time scheduler algorithm and its hardware implementation. In: MEMOCODE 2008, pp. 175–186. IEEE Computer Society Press, Los Alamitos (2008)
3. Böhm, P.: Formal Verification of a Clock Synchronization Method in a Distributed Automotive System. Master's thesis, Saarland University (2007)
4. Brown, G., Pike, L.: Automated verification and refinement for physical-layer protocols. Formal Aspects of Computing, 1–24 (2010)
5. FlexRay Consortium. FlexRay – the communication system for advanced automotive control applications (2006),http://www.flexray.com/
6. Endres, E.: FlexRay ähnliche Kommunikation zwischen FPGA-Boards. Master's thesis, Wissenschaftliche Arbeit, Saarland University, Saarbrücken (2009)
7. Knapp, S., Paul, W.: Realistic worst-case execution time analysis in the context of pervasive system verification. In: Reps, T., Sagiv, M., Bauer, J. (eds.) Wilhelm Festschrift. LNCS, vol. 4444, pp. 53–81. Springer, Heidelberg (2007)
8. Pfeifer, H.: Formal Analysis of Fault-Tolerant Algorithms in the Time-Triggered Architecture. PhD thesis, Universität Ulm, Germany (2003)
9. Pike, L.: Formal Verification of Time-Triggered Systems. PhD thesis, Indiana University (2006), http://www.cs.indiana.edu/~lepike/phd.html
10. Pike, L.: Modeling time-triggered protocols and verifying their real-time schedules. In: FMCAD 2007, pp. 231–238. IEEE, Los Alamitos (2007)
11. Rushby, J.: Systematic formal verification for fault-tolerant time-triggered algorithms. IEEE Transactions on Software Engineering 25(5), 651–660 (1999)
12. Schmaltz, J.: A Formal Model of Clock Domain Crossing and Automated Verification of Time-Triggered Hardware. In: FMCAD 2007 (2007)
13. Tverdyshev, S., Shadrin, A.: Formal verification of gate-level computer systems. In: LFM 2008, NASA (2008)
14. Tverdyshev, S., Alkassar, E.: Efficient bit-level model reductions for automated hardware verification. In: TIME 2008, pp. 164–172. IEEE, Los Alamitos (2008)