Formal verification of the TTCAN protocol

G. Leen¹ and D. Heffernan²

The following article is intended as a semi-formal description of a portion of work relating to the formal verification of the Time-Triggered Controller Area Network (TTCAN) protocol.

The past two decades have seen an increasing dependence on distributed electronic systems and associated software in vehicles. Today the electrical and software content of vehicles can account for up to 23% of the vehicle manufacturing cost [i]. Vehicle control systems vary widely in functionality from seat heating to engine management systems [ii]. Despite their diversity, all such systems may be divided into two definitive classifications, namely real-time systems and non-real-time systems. Such classification is based upon the consequences for system failure. For example the failure of the air conditioning unit is generally not a life-threatening event, however the failure of the anti-lock brakes (ABS) system may be. In view of the consequence of a critical system failure, much research effort is being expended to ensure that such failures do not occur. One primary focus of concern is the underlying in-vehicle control communications system as this component is common to all distributed system elements and represents the most vulnerable single point of failure for the entire system [iii]. Currently the Controller Area Network (CAN) is the most popular in-vehicle control network, with current annual transceivers sales running at over 200 million [iv].

Initiatives to increase the reliability, and to some degree reduce the failure modes, of the communications system have resulted in the design of several new robust communication protocols based on a time triggered model. The Time Triggered Protocol (TTP) protocol defines an entire fault tolerant system architecture, incorporating redundancy [v]. Other emerging time-triggered solutions such as Flexray [vi] and Byteflight [vii] look promising. All these protocols implement some form of the Time Division Multiple Access (TDMA) data transfer model. The culmination of practical experience and research in recent years has indicated that TDMA-based or time-triggered protocols have several strong advantages over event-triggered communication schemes for control networks in the context of safety critical applications [viii] [ix].

The vehicle system designer’s road map indicates an increased dependency on more sophisticated electronic systems within vehicles. The imminent deployment of drive-by-wire systems illustrates this trend [x] [xi]. For example, in the case of steer-by-wire, there exists no mechanical connection between the road wheels and the steering wheels, but rather a control level communication network is used to link sensors on the steering wheel and electric motors on the road wheels. Such initiatives allow software algorithms to dynamically optimise the vehicle performance characteristics, unlike the mechanical alternative, which is often, by necessity, a rigid compromise across the entire operating range. Additional innovations such as integrated chassis management [xii], advanced RADAR based systems [xiii], telematics and infotainment systems [xiv] are dramatically increasing the software content in our vehicles [xv]. Experts have estimated that more than 80% of new innovations within vehicles are now based on this combined software and electronic content.

[¹] PEI – Circuits & Systems Research Centre, Foundation Building, University of Limerick, Ireland. Gabriel.leen@ul.ie
[²] Electronic & Computer Engineering Department, University of Limerick, Ireland. Donal.Heffernan@ul.ie

© 2002 G. Leen & D. Heffernan
Given the inevitable continued growth in reliance on vehicle electronics and software and the need to physically distribute and schedule interacting control messages, the ISO (International Standards Organisation) have established a task force of experts to specify a new enhanced version of CAN designed to meet future application requirements. The result of their efforts has been the development of a session layer protocol specification based on the time-triggered paradigm: Time-Triggered CAN (TTCAN) [xvi]. Currently the protocol has already been prototyped both in software by NEC [xvii] and in hardware by Bosch [xviii] and Hitachi [xix].

In parallel with the development of the TTCAN protocol specification, the modeling and formal verification of the protocol specification has been undertaken by the authors. This is, as far as the authors are aware, the first time that the formal verification of an ISO defined control network protocol was conducted in parallel with the specification development. The CAN protocol itself has previously been formally verified using the Z specification language [xx] and using Petri Net models [xxi]. Some CAN control systems have been validated using the bCANDLE modeling language [xxii].

Formal verification is not to be confused with testing or simulation. Such verification provides a rigorous proof of correctness, as described by David Dill [xviii]. Much like one would prove Pythagoras’s theorem, not by testing the proposition, nor by simulating squares and triangles, but by mathematically defining the problem and thereafter using a logical progression to prove (or otherwise) the property that is Pythagoras’s theorem. As E. Dijkstra has pointed out, testing and simulating can only show the presence of errors, it can seldom, if ever, prove their absence. Consider for a moment the case of Euler’s conjecture, where he claimed that there was no integer solution, for x, y, z and w to the following equation:

\[ x^4 + y^4 + z^4 = w^4 \]

Equation 1  Euler’s conjecture

For over two hundred years this statement was neither proven nor disproved, despite years of testing on computers. In 1988 Naom Elkies of Harvard University discovered a solution, Equation 2, and also proved that there are an infinite number of solutions [xxiv].

\[ 2,682,440^4 + 15,365,639^4 + 18,796,760^4 = 20,615,673^4 \]

Equation 2  A solution to Euler’s conjecture

The Euler conjecture example illustrates that one cannot use evidence from the first million numbers, or tests for that matter, to prove a conjecture about all numbers or, similarly about all possible states of a system.

Formal verification reduces the process of system verification to a calculation [xxv], while providing a result that is as rigorous as any mathematical proof. In theory such methods are capable of producing flawless designs without the need to test the system, provided that the mathematical description of the system is accurate and sufficiently detailed. Nonetheless, in practice formal verification will not completely substitute for laboratory tests and field trials. As John Rushby [xxvi] points out:

---

3 This equation is not dissimilar to Fermat’s last theorem (proposed 1637, proven 1993), which has only recently been proved despite several hundred years of attempted proof and ‘testing’.
‘Formal methods can provide important evidence for consideration in certification but they can no more “prove” that an artifact of significant logical complexity is fit for its purpose than a finite-element calculation can “prove” that a wing span will do its job’.

This limitation is a reality, and as the proof is based on an abstraction it is not ipso facto. Yet no modern commercial airplane is designed without using mathematical modeling in the form of finite element analysis. Formal methods provide a significant increase in the level of confidence or otherwise, which we have in a system, which is in itself of significant importance. Barendregt states [xvii]:

“It is fair to state, that in this digital era correct systems for information processing are more valuable than gold”

Software and hardware are the coal and steel of the computer revolution, yet as Gibbs puts it:

‘despite 50 years of progress, the software industry remains years – perhaps decades – short of the mature engineering discipline needed to meet the demands of an information-age society [xxviii].’

Notwithstanding every effort, for complex systems it is often ‘impossible’ to test all execution threads through a system [xix] using conventional techniques. Similarly, digital hardware which may be viewed as a hard-coded software implementation (e.g. VHDL, and other hardware description languages) suffers from similar difficulties. When it comes to complex real-time systems, testing is at best a potential fault discovery process, while formal verification is a fault avoidance approach. To statistically guarantee that a system has a $10^{-9}$ probability of failure in a 1 hour mission one must test the system for more than 114,000 years [xxx].

A hardware system may physically fail due to electrical or physical stress, alpha particle bombardment, electromagnetic fields, or other such physical phenomena. However, software system faults are present at the beginning and up until the end of a system’s lifetime. It is interesting to note Butler and Finelli’s comment on this [xxiv]:

“...software reliability is meaningless – software is either correct or incorrect with respect to its specification.”

In this paper we present a formal specification and formal verification of the TTCAN protocol, which may help to ensure that the specification is also without flaw.

The paper is organized as follows: The next section presents an overview of the TTCAN protocol. Following this the framework for the formal verification of the TTCAN protocol is described, including a brief description of the UPPAAL formal verification language and tool set. Thereafter the TTCAN system model is described. Results of the formal verification process are then provided. The paper concludes with a summary of the work presented and comments on the value of formal methods in this project. The process variables and constants are provided in the appendix to this paper.

**TTCAN protocol description**

This section briefly describes the TTCAN protocol. For a in-depth description of the protocol refer to [xvi] [xxxi] [xxvii] [xxviii] [xxix]. As previously stated the TTCAN protocol is realised by adding a time-triggered session layer protocol to the already existing data link and physical layers of the CAN protocol stack. Here
in the context of describing TTCAN only the salient features of the CAN specification are given. For a
detailed description of the CAN protocol refer to [xxxvi] [xxxvii] [xxxviii] [xxxix] [xl].

The following description of the TTCAN and CAN protocols is somewhat simplified for clarity. Too much
detail is avoided so as not to distract the reader from the relevant essential features of the protocol.

**The CAN protocol**

A CAN network consists of a number of electronic nodes connected in a simple bus topology, using a
twisted pair wire transmission medium, see Figure 1. CAN is a multi-master protocol where all nodes may
compete for access to the medium provided the medium is idle when they initiate their frame transmission.
In CAN when two or more nodes simultaneously attempt to transmit a frame the resulting bus collision is
resolved in a non-destructive fashion. This is achieved using non-destructive bit-wise arbitration (NDBA)
during the identifier portion of the CAN frame, see Figure 2. The frame with the lowest numerical
identifier value is given priority and allowed to transmit while the unsuccessful frames back-off and
attempt to re-transmit when the medium again becomes idle. In the normal CAN protocol this re-transmit
mechanism is always active, however, in TTCAN it is normally disabled and only enabled in defined
special cases. All nodes connected to the network receive all messages which are broadcast on the network.
In normal CAN any node may attempt to transmit a message at any time provided the network is idle. If
transmission is postponed due to the re-transmission mechanism the waiting message effectively joins a
distributed network wide queue for re-transmission. This distributed queue is priority ordered based on the
frame identifier, by virtue of CAN’s arbitration mechanism, with the highest priority frame always at the
head of the queue ⁴. CAN may be described as an event-triggered network, in that a message transaction
may be triggered arbitrarily at any time, although it must wait for the bus to become idle before attempting
to release a message.

---

⁴ This network wide arbitration scheme forming a distributed queue is not to be confused with the queuing of message frames
internally in the CAN protocol engine.
Formal Verification of the TTCAN protocol

Figure 1 CAN network

Figure 2 A CAN version 2.0A data frame

IFS: Inter Frame Space  DLC: Data Length Code
SOF: Start Of Frame    CRC: Cyclic Redundancy Check
RTR: Remote Transmit Request  ACK: ACKnowledge
IDE: IDentifier Extension   EOF: End Of Frame
r0: Reserved bit
The TTCAN protocol

The TTCAN protocol realises a global and static schedule for message transactions based on a TDMA structure. The TTCAN protocol itself is essentially based on the addition of a session layer (OSI layer 5) to the existing CAN protocol stack (OSI layers 1 and 2), see Figure 3. Time is divided into time windows and messages are scheduled for transfer within the bounds of these time windows. The schedule itself, known as the Matrix Cycle (MC), defines a finite number of transactions, over a finite time interval. Once the schedule has completed it repeats indefinitely (much like a weekly bus time table repeats each week). Figure 4 illustrates an example TTCAN MC5. The MC schedule is divided into Basic Cycles (BC), where

---

5 Note: ‘?’ in the context of this illustration represents ‘or’ and indicates that two or more messages are competing for medium access at this time.
each BC commences with a message known as the Reference Message and terminates with the occurrence of the next Reference Message. A BC is subdivided into time windows. The duration sequence of time windows in all BCs is equal. This regular structure correlates to a 2-dimensional matrix, where a row represents a BC, and a column, known as a Transmission Column (TC), corresponds in duration to a time window interval. Thus time windows of a TC are of equal value in the time domain, however they may differ in the data domain i.e. they may contain different messages.

Figure 4  TTCAN matrix cycle

TTCAN defines a sub time window interval at the start of each TC windows (except for TC zero), where messages are scheduled for transfer. The interval, known as the Transmission Enable Window (TEW), defines the ‘launch window’ for the message. Messages may commence transmission only during this interval, provided the medium is idle. Such a requirement ensures that late messages released in their respective time window do not over-run into the next time window and corrupt the temporal integrity of the MC.

There are three basic types of time window: free windows, exclusive windows and arbitrating windows. Free windows are scheduled bus idle periods; they allow for later system expansion. Exclusive windows are intervals where a single specific message is scheduled to have exclusive transmission rights on the medium, without competition from other nodes on the network. During arbitrating time windows two or
more nodes may arbitrate for medium access. CAN’s native NDBA mechanism resolves conflicts in this situation. When two or more arbitrating windows are sequential they may be optionally appended forming a larger merged arbitrating time window. In this case the TEW for the merged arbitrating windows are joined as illustrated in Figure 4, for BC 1 with the windows TC2 and TC3 merged.

The TTCAN protocol defines two register sets to control the transmission and reception of messages, these are Tx_Triggers and Rx_Triggers respectively. Associated with a Tx_Trigger register set is a pointer to the single specific message, an index for the TC and BC to define when the message is to be released and a repeat factor which sets the period within a TC when the message is again released, provided the message is periodic within the scope of a TC. Rx_Triggers are similar to Tx_Triggers, however they evaluate whether or not a given message has been received since the start of a given BC.

Associated with each message appearing in an exclusive window is a Message Status Counter (MSC). An MSC has a bounded range of 0-7 and records successful and failed message transactions by incrementing and decrementing the MSC appropriately. Figure 5 illustrates the error state transition behaviour for a TTCAN node.

There are two levels of synchronisation quality in TTCAN: Level 1 and Level 2. Level 2 is an extended version of Level 1. In both implementations system time is measured in units known as Network Time Units (NTU). In Level 1 the NTU is equal in duration to a nominal CAN bit time. In Level 2 the NTU is a fraction of the physical second. Additionally, Level 2 provides mechanisms to improve the synchronisation quality within the network (Level 2 will not be discussed further here, see reference for more detail [xli] [xlii]).

Synchronisation of the member nodes in a TTCAN network is realised through the creation of a global event in time to which all other events are referenced. This event is the frame synchronisation pulse generated at the sampling point of the Start of Frame (SOF) bit of a Reference Message. Conceptually on the occurrence of this event all member nodes reset a counter which records the passage of time during a BC, this concept of time is referred to as Cycle Time. System events such as the activation of a Tx_trigger or a Rx_trigger are referenced using Cycle Time.

As the occurrence of a Reference Message ensures the progression of the MC, measures have been taken to ensure the presence of the Reference Message. Nodes capable of releasing a Reference Message are known as ‘potential time masters’ and the node currently responsible within the system for the release of Reference Messages is known as the ‘active time master’. If the active time master fails for any reason then a potential time master will release a Reference Message thereby synchronising the TTCAN node group and ensuring the proper execution of the time-triggered communications cycle.
To draw a simple TTCAN analogy, consider for a moment an airport runway to be a limited resource, much like the physical medium in the case of a CAN network. Now consider airplanes to be equivalent to message frames on the CAN network. Then, as in the case of airports, in order to realise the maximum potential of the limited resource a schedule or timetable is enforced for all flight arrivals and departures. Similarly in the case of a TTCAN network a schedule is enforced for message transactions.
Framework for the TTCAN formal verification

A surprisingly wide variety of mathematical construct languages and logical proof systems are available which are capable of the formal specification and rigorous analysis of communication system protocols. These include languages and tools such as Communicating Sequential Processes (CSP)[xliii], SPIN[xliv][xlv], Practical Verification System (PVS)[xlvi], HyTech[xlvii][xlviii], Kronos[xlix][l] and several others. However, the authors chose the UPPAAL tool for this project. UPPAAL is an automated verification tool suite, jointly developed by BRICS at Aalborg University, Denmark and by the Department of Computer Systems at Uppsala University, Sweden. UPPAAL has been used in the analysis of many case study systems and real-world systems. For example UPPAAL has been used in the verification of an audio control protocol developed by Philips[li][lii], a gear box controller[liii], the TDMA start-up mechanism of the DACAPO[liv] protocol[lv], and several other systems[lvi].

The initial step in the formal verification in the TTCAN protocol was to take the textural TTCAN protocol specification and translate it to a series of behavioral diagrams which encapsulated the protocol process in a pseudo flow chart manner. From these behavioral diagrams the mathematical description of the protocol was realised using the UPPAAL modeling and verification tool suites[lvii][lviii].

The following section presents a brief overview of timed automata and associated networks as applied in the UPPAAL verification tool. For a comprehensive and definitive description, refer to: [lix][lx][lxi][lxii][lxiii][lxiv], upon which the following is based. The timed automata used here to model finite-state, real-time systems are finite state automata decorated with a finite set of real-value clock variables.

Time transition systems

The timed transition system used to semantically describe the TTCAN model is a labeled transition system with two associated label types: atomic actions and delay actions. The following definitions are used:

Define:

\[ \text{Act} \quad \text{a finite set of actions ranged over by a, b,\ldots} \]
\[ \text{P} \quad \text{a set of atomic propositions ranged over by p, q,\ldots} \]
\[ \mathbb{R} \quad \text{the set of non-negative real numbers} \]
\[ \Delta \quad \text{the set of delay actions: } \{\varepsilon (d) \mid d \in \mathbb{R}_{\geq 0}\} \]
\[ L \quad \text{the set formed by the union: } \{\text{Act} \cup \Delta\} \]
\[ \text{Act}_{\tau} \quad \text{the set formed by the union } \{\text{Act} \cup \tau\} \]

The symbol \(\tau\) does not occur in \text{Act}, and stands for an internal action of the system[lxiv]. This final definition is added for completeness and to unify the notation used in several texts. In the following the abbreviation \(L\) will be used and its meaning will be understood taken in conjunction with definition 1 (given below), and the following alternating action and delay progression, where \(S\) signifies an automaton state:

---

6 BRICS – Basic Research in Computer Science, is a fundamental research center funded by the Danish government at Aarhus University and Alborg University in Denmark.
Formal Verification of the TTCAN protocol

\[ s \xrightarrow{a} s' \iff \exists s'' \text{ st. } s \xrightarrow{r} s'' \xrightarrow{a} s' \]

\[ s \xrightarrow{e(d)} s' \iff \text{there exists a computation:} \]

\[ s = s_0 \xrightarrow{\alpha_1} s_0' \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n = s' \quad (n \geq 0) \]

(1) \( \forall i \in \{1, \ldots, n\}, \alpha_i = \tau \text{ or } \alpha_i \in \Delta \)

(2) \( d = \Sigma \{d_i \mid \alpha_i \in (d_i)\} \)

If the set \( \{d_i \mid \alpha_i \in (d_i)\} \) is a null set, then \( \Sigma \{d_i \mid \alpha_i \in (d_i)\} = 0 \). By convention the relation \( s \xrightarrow{e(0)} \) is equivalent to \( s \xrightarrow{\tau} \), that is the reflexive closure of \( s \xrightarrow{\tau} \). In the notation which follows the transition relation \( s \xrightarrow{e(d)} \) with \( d = 0 \) will generally be used and its relationship to the equivalent \( \tau \)-transition understood.

**Definition 1 (timed transition system)**

The timed transition system (or Timed Labeled Transition System (TLTS)) over \( \text{Act and } \mathcal{T} \) is a tuple of the form \( S = \langle S, s_0, \rightarrow, V \rangle \)

Where:
- \( S \) is a set of states
- \( s_0 \) is the initial state
- \( \rightarrow \) is a transition relation \( \subseteq S \times L \times S \)
- \( V \) is a proposition assignment function, \( V: S \mapsto 2^b \)

The transition relation \( \rightarrow \) defines how to evolve from one state to another and satisfies the following properties:

**Time determinism:** for every \( s, s_1, s_2 \in S \), and \( d \in \mathbb{R}_{\geq 0} \), then:

\[ \text{if } s \xrightarrow{e(d)} s_1 \text{ and } s \xrightarrow{e(d)} s_2 \text{ then } \Rightarrow s_1 = s_2 \]

**Time continuity**\(^7\): for every \( s, s', \in S \), and \( d, e \in \mathbb{R}_{\geq 0} \), then:

\[ s \xrightarrow{e(d+e)} s' \iff \exists s'' \text{ st. } s \xrightarrow{e(d)} s'' \xrightarrow{e(e)} s' \text{ for some } s'' \in S \]

**Zero-delay:** for every \( s, s', \in S \)

\[ s \xrightarrow{e(0)} s' \iff s = s' \]

**Parallel composite systems of timed automata**

To compose an interleaved system consisting of two or more timed transition systems, the parallel composition function is defined in the form of a synchronisation function. This function enables a compound timed transition system to synchronise on actions.

\(^7\) Also referred to as time additivity.
Define:

\( f \) is a synchronisation function

\( f \) is a partial function\(^8\): \((\text{Act} \cup \{0\}) \times (\text{Act} \cup \{0\}) \twoheadrightarrow \text{Act} \), where 0 is a distinctive non-action symbol\(^9\).

Using the \( f \) function we can define the parallel composition of a number of timed transition systems.

Let \( S_i = \langle S_i, s_{i0}, \rightarrow_i, V_i \rangle \), \( i = 1, 2 \) be two timed transition systems, then the parallel composition \( S_1 \mid f S_2 \) is the timed transition system \( \langle S, s_0, \rightarrow, V \rangle \),

where \( s_1 \mid f s_2 \in S \), whenever:

\( s_1 \in S_1 \) and \( s_2 \in S_2 \)

and \( s_0 = s_{10} \mid f s_{20} \)

Then the transition relation \( \rightarrow \) can be inductively defined as follows:

\[
s_1 \mid f s_2 \xleftarrow{c} s_1' \mid f s_2' \quad \text{if} \quad s_1 \xrightarrow{a} s_1', \quad s_2 \xrightarrow{b} s_2' \quad \text{and} \quad f(a, b) = c
\]

This illustrates the decomposition of the compound transition atomic actions of the parallel system into the ‘sub’ transition atomic actions of the component timed transition systems: \( S_1 \) and \( S_2 \).

The following defines similarly the composition of the atomic delay actions of the parallel system.

\[
s_1 \mid f s_2 \xleftarrow{d} s_1' \mid f s_2' \quad \text{if} \quad s_1 \xrightarrow{d} s_1', \quad s_2 \xrightarrow{d} s_2' \quad \text{and} \quad f(a, b) = c
\]

The proposition assignment function \( V \) which maps propositions to the component system automata, is defined as:

\[
V(s_1 \mid f s_2) = V_1(s_1) \cup V_2(s_2)
\]

A trace \( \sigma \) is defined as a finite alternating sequence\(^10\) (of actions and delays) of the form:

\[
\sigma = s_0 \xleftarrow{d_0} s_0' \xrightarrow{a_1} s_1 \xleftarrow{d_1} s_1' \xrightarrow{a_2} s_2 \xleftarrow{d_2} s_2' \ldots \xrightarrow{a_n} s_n \xleftarrow{d_n} s_n'
\]

where: \( d_i \in \mathbb{R} \)

Define:

\( \pi \) a position of the trace,

---

\(^8\) A partial function is one which is not defined for some of its domain. For instance, division is usually a partial function since a number divided by 0 is usually undefined in some number systems.

\(^9\) 0 simply denotes a distinguishable non-action symbol st. \( s \xrightarrow{0} s_1 \iff s = s_1 \)

\(^10\) Transformation to an alternating trace sequence may require application of the time continuity property to avoid two consecutive delay transitions. Some 0-delays may have to be inserted in order to avoid consecutive action transitions.
where: \( \pi = (i, d) \), \( i \in 0..n \), i.e. the position consists of a state \( i \) and an associated delay \( d \) at this position where, \( 0 \leq d \leq d_i \)

Define:
\[
\Delta(\sigma, \pi) = \sum_{j < i} d_j + d \quad \text{the accumulated delay of the trace } \sigma \text{ up to position } \pi,
\]

and \( \sigma(\pi) \) for the remainder of the trace \( \sigma \) starting from \( \pi \), i.e.

\[
\sigma(\pi) = s_i^d \xrightarrow{\epsilon(d_i - d)} s_1 \xrightarrow{a_1} s_{i+1} \xrightarrow{\epsilon(d_{i+1})} \ldots \xrightarrow{a_n} s_n \xrightarrow{\epsilon(d_n)} s_n'
\]

Path \( \sigma \) is called time-divergent if \( \lim_{\pi \to \infty} \Delta(\sigma, \pi) = \infty \). An example of a non-time-divergent path is a path which contains an infinite number of states while \( \Delta(\sigma, \pi) \) is finite.

\[
\sigma = s_0 \xrightarrow{\epsilon(2^0)} s_1 \xrightarrow{\epsilon(2^1)} s_2 \xrightarrow{\epsilon(2^2)} s_3 \xrightarrow{\epsilon(2^3)} \ldots
\]

This trace path is not time-divergent as an infinite number of states are visited in the bounded interval \([\frac{1}{2}, 1]\). A timed transition system \( S \) is referred to as non-Zeno if from any of its states some time-divergent path can start. Generally, for practical reasons, only non-Zeno trace paths are considered, as Zeno paths are unrealistic.

\section*{Networks of timed automata}

A TLTS \([lxv][lxvi]\) is essentially a finite-state automaton with data variables ranging over finite data domains extended with a finite collection of clocks\(^{12}\).

The clocks may be conceptually regarded as system clocks, which proceed at the same rate and quantify the time elapsed since they were last reset. Clock values may be tested (compared with real numbers) and reset (assigned to zero).

\section*{Definition 2 (clock constraints and assignment)}

Define \( C \) to be a set of real value clocks.

Define \( \Phi(C) \) the set of boolean expressions ranging over \( C \) by \( g \), i.e. the clock constraint systems over \( C \).

The syntax for these formulas is generated by the formula: \( g ::= c \mid g \land g \)

Where:

\( c \) is an atomic constraint of the form: \( x \sim n \) or \( x - y \sim n \),

for \( x, y \in C \),

the relation \( \sim \) is such that: \( \sim \in \{\leq, \geq, <, >, =\} \)

and \( n \) is a natural number\(^{13}\).

\(^{11}\) Zeno of Citium was a Greek philosopher who formulated paradoxes that defended the belief that motion and change are illusory (ca 335-263 BC), e.g. Achilles and the Tortoise.

\(^{12}\) Timed transition systems may alternatively be described using timed process calculi.

\(^{13}\) Natural numbers, also known as counting numbers, are contained by the set beginning with 1, with each successive number greater than its predecessor by 1. If the set of natural numbers is denoted by \( N \), then \( N = \{1, 2, 3, \ldots, n, (n+1), \ldots\} \)
**Definition 3 (timed automaton)**

A timed automaton $A$ over:

- Actions $\text{Act}$
- Atomic propositions $\mathcal{P}$
- Clocks $\mathcal{C}$

is defined as a tuple of the form $A = \langle N, \ell_0, E, I, V \rangle$,

Where

- $N$ is a finite set of nodes (control-states of the transition system)
- $\ell_0$ is the initial node
- $E$ is the set of edges of the form $E \subseteq N \times \beta(C) \times \text{Act} \times 2^C \times N$
- $I$ is a clock constraint function, $I : N \rightarrow \beta(C)$, which assigns the invariant conditions of a node
- $V$ is a proposition assignment function, $V : N \rightarrow 2^\mathcal{P}$ which assigns a set of propositions true while control rests in the nodes

An example of a tuple of the form $\langle N, \ell_0, E, I, V \rangle$ is: $\langle \ell, g, a, r, \ell' \rangle \in E$, written $\ell \xrightarrow{g,a,r} \ell'$, this represents a edge from the node $\ell$ to the node $\ell'$ with clock constraint $g$ (enabling condition or guard), action $a$, and a set of clocks $r$ to be reset on the transition.

Every control node $\ell$ of the automaton has an associated invariant condition $I(\ell)$ which is a clocks constraint. Thus the individual states of an automaton $A$ may be represented as the pair $(\ell, u)$ where $\ell$ is the particular node in question and $u$ a clock assignment for $C$, which maps each clock in the set $C$ to a value in $\mathbb{R}_{\geq 0}$.

Define:

$u$ is a *time assignment* or *valuation* function which maps from $C$ to $\mathbb{R}_{\geq 0}$.

This constraint must be satisfied by the system clocks while the process is operating in that particular control node. The system starts at $\ell_0$ with all clocks initialised to zero, $(\ell_0, u_0)$. The clocks increment synchronously (provided there rates are defined to be equal), and control may remain in a given node provided the invariant condition for that node is satisfied $I(\ell)$. At any point in time, the automaton may change control location by following the an edge $\ell \xrightarrow{g,a,r} \ell'$ provided the value of the clock satisfies the enabling guard condition $g$.

The semantics of $A$ are given by the timed transition system $S_A = \langle S, s_0, \rightarrow, V \rangle$ where:

- $S$ is the set of states of $A$
- $s_0$ is the initial state $(\ell_0, u_0)$
- $V$ is the proposition assignment function is extended to $S$ by $V(\ell, u) = V(\ell)$
- $\rightarrow$ is the transition relation defined below:

$(l, u) \xrightarrow{a} (l', u')$ if there exists $r$ and $g$ such that $l \xrightarrow{g,a,r} l'$, $g$ is satisfied by $u$ and $u' = r(u)$.

---

$^14 \ell \equiv l$

© 2002 G. Leen & D. Heffernan
Formal Verification of the TTCAN protocol

\[(l, u) \xrightarrow{\ell_0, u_0} (l', u')\] if \(\ell = \ell', u' = u + d\) and \(I(\ell)\) is satisfied by \(u'\) \(16\)

We define Tr(A) as the set of all traces of \(S_A\) starting with the initial state \((\ell_0, u_0)\). The notion of parallel composition as defined in definition 1 may now be extended to timed automata as follows:

For some two timed automata \(J\) and \(K\) and a synchronisation function \(f\), the parallel composition \(J_1 \mid f K_2\) denotes the timed transition system \(S_J \mid f S_K\).

The following introduces the dense-time logic \(L_v\). This branch of logic can be regarded as a subset of the \(\mu\)-calculus described by Henzing et. al. in \([lxvii]\) \([lxviii]\). This timed logic is now applied to the proceeding system definitions as follows:

**Definition 4**

Define:

- \(K\) to be a finite set of clocks
- \(Id\) to be a set of identifiers
- \(L_v\) the set of formula over \(K, Id, Act, \) and \(\Phi\)

The formulas \(\varphi\) and \(\psi\) ranging over \(L_v\) can be generated using the abstract syntax:

\[\varphi ::= c \mid p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \forall \varphi \mid \exists \varphi \mid \langle \rangle \varphi \mid \lceil \varphi \rceil \mid |x + n - y + m| \mid Z\]

where:

- \(c\) is an atomic clock constraint of the form \(x \sim n\) or \(x - y \sim n\)
- \(x, y\) are elements of \(K\), i.e. clocks
- \(n\) is a natural number
- \(p\) is an atomic predicate\(^{17}\), i.e. an element of \(\Phi\)
- \(\triangleright\) is an element of \(Act\), i.e. an atomic action
- \(z\) is an element of \(K\)
- \(Z\) is an element of \(Id\), i.e. an identifier

Model checking algorithms of UPPAAL test reachability propositions of the form \(\exists \Diamond \beta\) and invariance properties \(\forall \Box \beta\), where \(\beta\) is a location property. For a detailed description of the algorithms used see \([lxix]\) \([lx]\).

**UPPAAL system specification, simulation and verification**

The UPPAAL environment consists of a system editor, a simulator and a verifier. In the system editor component one may construct the formal system definition in a graphical format. Figure 7 (which will be discussed later in the text) and subsequent illustrations provide examples of this format. Circles, i.e. vertices in the graph, represent control nodes or states. Directed lines linking states in the graph structure

\(^{15}\) \(r(u)\) is the assignment such that \(r(u)(x) = 0\) if \(x \in r\), i.e. a clock assignment on the clock \(x\) st. \(x\) is reset to zero if it is an element of the set \(r\), i.e. the set of clocks which are to be reset. Otherwise \(r(u)(x) = u(x)\), which means that the clock may be allowed to progress as normal. For every given subset of clocks \(C\), then \([C' \rightarrow 0]u\) or \(r(C')\) maps every clock in \(C'\) to zero and agrees with \(u\) over \(C'\).

\(^{16}\) \((u+d)\) is the assignment such that \((u+d)(x)=u(x)+d\), which states that control may remain in a node provided the applicable clock constraints are satisfied.

\(^{17}\) i.e. the assignment of an attribute
represent transitions or edges. State invariants are in blue text, while transition guards and assignments are in red text. States containing the letter ‘c’ are committed states, meaning that no delay is allowed in the state and valid exiting action transitions are committed to occur without the passage of system time. States containing a circle represent the initial state of an automaton. Synchronisation channels are syntactically represented as pairs: channel_name! and channel_name?, where the exclamation mark signifies the instigator of the synchronisation function request. The synchronisation action may be used to force parallel transitions, however additional shared variables are necessary for data transfer.

Once the system has been correctly entered it may be simulated in the simulator component of the UPPAAL tool suite. This feature provides valuable feedback during the system design phase. Diagnostic traces generated in the simulator tool and later by the verification engine may be examined step-by-step in the simulator. Once the system specification is complete the verification engine may be activated and system properties verified. The following BNF based representation gives the grammar for the requirement specification. Names with capital initial letters are used to denote non-terminals, NAT denotes a natural number (including 0), and ID denotes valid identifier names.

Property ::= A[] SP | E<> SP | E[] SP | A<> SP | SP → SP
SP ::= AP | not SP | ( SP ) | SP or SP | SP and SP | SP imply SP | deadlock

AP ::= ID.ID | CGuard | IGuard
CGuard ::= ID REL NAT | ID REL ID | ID REL ID + NAT | ID REL ID - NAT
IGuard ::= IExpr REL IExpr | IExpr != IExpr
IExpr ::= ID | ID[IExpr] | NAT | -IExpr | (IExpr) | IExpr OP IExpr

REL ::= < | <= | >= | > | ==
OP ::= + | - | * | /

**TTCAN system model specification**

The TTCAN protocol behaviour was abstracted from the text-based specification and manually translated into a mathematical model representing the essence of the protocol. The model defines a system consisting of two potential time master nodes, a time receiving node and a CAN physical layer, realising a Level 1 implementation. Using the UPPAAL language 10 timed automata were designed. The individual system automata for each node were: a combined error state machine and error handler automaton, a protocol scheduler automaton and transceiver automaton. Each of the three network nodes contains these elements and communicated via the physical medium automaton. Figure 6 illustrates a top-level view of the automata.
Each network node group contains a scheduler. Encoded within the scheduler is the MC information relevant for the correct operation of the respective node. The scheduler also contains a local clock, which records Cycle Time. Thus the scheduler determines at what point in local time the Rx_Triggers and Tx_Triggers become active and for which messages. The transceiver automata interacts with the physical medium automaton by transferring and receiving messages to and from the physical medium and by observing the state of the physical medium, i.e. idle and not-idle. The physical medium automation performs the identifier arbitration function of the underlying CAN protocol. The error handler monitors the progress of the message transaction sequence as defined by the portion of the MC held within the relevant node scheduler. This function is performed predominantly through the manipulation of MSCs along with additional information derived from the node’s scheduler and transceiver. The error state machine section evaluates the calculations of the error handler portion of the automata and determines the error level status of the node.

Each node, once synchronised to the active system time master, transmits messages and registers the reception of messages as per the relevant portion of the MC. For instance, Node 1 may attempt to transmit a message with CAN frame identifier 4 to the physical medium automaton in the time window defined by BC 0, TC 1. Provided the bus is idle at the time of transmission and media arbitration is successful, then Node 2 and Node 3 will receive the CAN frame via their respective transceivers. If an Rx_Trigger has been configured for this message then a node will observe that the message was correctly received and the MSC corresponding to message identifier 4 will be updated appropriately. Should the updated value of this MSC result in a change of error level then the error state handler will observe and realise this change.
Criteria and restriction used in the verification of the TTCAN protocol

The TTCAN protocol model presented here makes a number of restrictions on the behaviour of a TTCAN network. Only the functional behaviour and performance of the TTCAN protocol is of interest here, hence other un-necessary detail is avoided.

The restrictions listed below help to remove irrelevant complexity from the model and prevent a state-space explosion during the verification process. Further iterations of the verification process will focus on other specific properties of the protocol, such as variable message length, clock drift between nodes etc. and a number of these restrictions will be relaxed. The primary restrictions enforced in the design of the TTCAN model described here are:

1. The medium does not introduce errors
2. There are no error frames, once a message successfully arbitrates it is transferred without error (by virtue of restriction 1)
3. Reference Messages are of fixed length (in bits) regardless of which potential time master sends them
4. Normal messages of the BC are fixed in length
5. The data field and CRC field are assumed to be consistent and correct, and are therefore ignored
6. The correct acknowledgement of all message is assumed
7. For the purposes of this model all clocks are assumed to proceed at the same rate and thus the NTU is a constant within the system
8. Data processing of a logical context takes zero time
9. Certain timeout values have been reduced in order to help minimise the verification state space

The standard error containment mechanisms of the CAN protocol are redundant based on the above assumptions, e.g. frame form checking, etc. This restriction helps to significantly reduce the complexity of the system.
Description of a potential time master scheduler automation

In a temporal sense there are two classes of network nodes in a TTCAN system, a potential time master node and a time receiving node, as illustrated in Figure 6. Essentially the main difference between these two nodes is realised in the scheduler component of the node. The following is a description of the scheduler automaton for a potential time master. Figure 7 corresponds to section A, while Figure 8 corresponds to the lower section B of the scheduler. All parameters used in the model are listed in the appendix to this paper. As the overall system is extremely large only the automata for a single node and the physical medium are presented here. The automata for the remaining two nodes are similar and differ mainly in the configuration of the scheduler automata.

With reference to Figure 7 and Figure 8, the potential time master automaton initialises in the location labeled RESET, in Figure 7. As this location is a committed location it is immediately forced to take the transition to location S74 without any time delay. On making this transition a number of local variables are initialised, as follows:

\[
\begin{align*}
\text{State}_1 & := \text{Initialisation}, \\
\text{Cycle}_{\text{Time}}_1 & := 0, \\
\text{Ref}_{\text{Trig}}_{\text{Offset}}_1_{\text{P}} & := \text{Init}_{\text{Ref}}_{\text{Trig}}_{\text{Offset}}_1, \\
\text{Ref}_{\text{Trig}}_{\text{Offset}}_1_{\text{N}} & := 0, \\
\text{Ref}_{\text{Trigger}}_1 & := \text{Gap}_{\text{T}}_{\text{O}}_1, \\
\text{Tx}_{\text{Count}}_1 & := 0, \\
\text{Cycle}_{\text{Count}}_1 & := 0, \\
\text{Sync}_1 & := 0
\end{align*}
\]

Control waits in location S74 until either it receives a Reference Mark (generated by the transceiver automaton) or until the location invariant is reached forcing the system to take the transition to S87, when the guard: \( \text{Cycle}_{\text{Time}}_1 == \text{Gap}_{\text{T}}_{\text{O}}_1 \), allows this transition. The constant \( \text{Gap}_{\text{T}}_{\text{O}}_1 \) is the value of the maximum Gap time expected within the system plus a Reference Trigger Offset for this specific potential time master.

Assuming that this node is the primary potential time master and the node reaches this time mark \( \text{Gap}_{\text{T}}_{\text{O}}_1 \) prior to the release of a Reference Message from another potential time master, then the transition to location S87 will be taken. In location S87 the node signals to its transceiver automaton, via the urgent channel TX_TRIGGER_1, to commence a transmission. The identifier of the requested message for transmission has been transferred to the transceiver via the shared variable Pt_Tx_Message_1. If the transmission is unsuccessful then control will be forced from this location as follows: S39 → S31 → S21 → S26 → S34 and a Watch Dog time out error will be flagged.

Assuming that either the transmission of the current node’s Reference Message is successful or the Reference Message of another potential time master is received, then the urgent synchronisation channel REF_MARK will force control from location S87 to S80. Here the priority of the received message is compared with the priority of the current node and based on this evaluation control will either pass to location: S78, S79 or S81.

Assuming that the Reference Mark signal was generated as a result of the release of its own Reference Message then control will be forced to location S81. Here control waits for the remaining duration of a BC. During this interval, a Reference Message may or may not be received from another potential time master. In the event of a Reference Message not being detected during this period, control will pass from location S81 to location S89. In location S89 the automaton will once again attempt to transmit a Reference
Assuming that the node successfully transmits its own Reference Message, control will proceed as follows: $S89 \rightarrow S83 \rightarrow S86 \rightarrow S95 \rightarrow S33 \rightarrow S27$, without the loss of time as these locations are committed locations. In location $S27$ the automaton selects a transition based on the value of $\text{Cycle}\_\text{Count}$. Assuming that the node has become the active time master for the system and the MC is just commencing, then control will proceed to location $S19$. Control rests here until an RX_TRIGGER for message identifier $5$ has been reached for TC $1^{18}$.

Assume that control proceeds in a similar fashion to location $S40$ (see Figure 8). Here the schedule has been configured to transmit a message with identifier $3$. During the interval bounded by a Cycle_Time greater than or equal to the time mark $\text{Start}\_\text{Tx}\_\text{Col}\_3$ and less than or equal to the time mark $\text{End}\_\text{Tx}\_\text{Enable}\_\text{Col}\_3$, the node will attempt to transmit this message. Failing to do so will result in the urgent channel $\text{MSC}\_\text{TX}\_\text{NOK}$ signaling this event to the relevant error handler automaton.

Assume the automaton proceeds until the time mark $\text{Rx}\_\text{Trigger}\_\text{Col}\_4$ is reached. The Rx_Trigger is set for message identifier $6$ and control proceeds as follows $S0 \rightarrow S32 \rightarrow S35 \rightarrow S13 \rightarrow S6 \rightarrow S68 \rightarrow S69$. At this point in time an artificial clock, $C\_\text{Tick}\_1$, is synchronised with the value of Cycle_Time$^{19}$. The artificial clock $C\_\text{Tick}\_1$ is compared to the time mark for the reference trigger. When the Ref_Trigger time mark is reached control proceeds at location $S59$ and the node will attempt to transmit a Reference Message. Assuming that this transmission is successful the automaton will take the path $S2 \rightarrow S57 \rightarrow S61 \rightarrow S43 \rightarrow S18 \rightarrow S8$ and the second BC will commence.

---

$^{18}$ In the currently described system RX_TRIGGERS have been configured to occur after the expected termination of the message reception interval, however within the time window where the message was expected. Within the specification such RX_TRIGGERS may occur much later than this, at the end of the BC as is the case in TC 3.

$^{19}$ This is necessary to implement the variable Reference Message transmission time out mechanism of the potential Time Master, as the UPPAAL syntax does not allow the comparison of clock variables and integer variables.
Figure 7  Potential time master scheduler, section A
Figure 8  Potential time master scheduler, section B
**Description of error handler and error state machine automata**

Figure 9 presents the error containment mechanism for node 1. This consists of a combined error handler and error state machine. The function of the automaton is to evaluate the values of the received and transmitted message status counters, the transmission count variables and the watchdog timer in order to determine the appropriate error level for the TTCAN node.

Control initialises in the committed location labeled Quiescent_State. Depending on which urgent synchronisation channel is activated control leaves this location along the appropriate transition. The possible edges, which trigger an error status evaluation, are given in Table 1.

<table>
<thead>
<tr>
<th>Urgent Channel</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX_TRIGGER_1</td>
<td>Check for correct message update</td>
</tr>
<tr>
<td>TX_TRIGGER_E_1</td>
<td>Transmission trigger active for an exclusive time window</td>
</tr>
<tr>
<td>MSC_TX_OK</td>
<td>Message transmission successful</td>
</tr>
<tr>
<td>MSC_TX_NOK</td>
<td>Message transmission unsuccessful</td>
</tr>
<tr>
<td>CHECK_MSC</td>
<td>Evaluate the MSC difference, Tx underflow and Tx overflow</td>
</tr>
</tbody>
</table>

**Table 1  Error containment stimuli**

If after taking one of these transitions an error variable is altered such that it may result in a change in error level, then the error level variable: E_State is re-evaluated and the appropriate error containment procedure is activated.

As an example, synchronisation channel TX_TRIGGER_E_1 is activated every time the transmission of a message, in an exclusive time window, takes place. Thus this signal may be used to evaluate if there is a transmission overflow or indeed underflow for a given MC. As illustrated in Figure 9, on the occurrence of TX_TRIGGER_E_1, control moves from the location Quiescent_State, to S4. Based on the value of the current transmission count variable Tx_Count relative to the expected number of transmissions per MC, control will progress either to location S18, if an overflow has not occurred, otherwise to location TX_OVERFLOW.

Assuming an overflow is detected, then this error immediately forces a transition for node 1 to TTCAN error level S2, where further message transmissions are disabled. This process is modeled by the control transition from location TX_OVERFLOW to S27, which sets the boolean variable TX_OVER_1. Control then progresses from S27 via the intermediate locations to S32, where TX_OVER_1 is tested as true and the variable S_Two_1 is set. Control then follows S33 → S34 → S35 → S36, here the variable S_Two_1 tests positive the node error state E_State_1 is set, transmissions are disabled and the Reference Trigger offset value is maximised.

The remaining error containment stimuli behaviour may be understood from Figure 5, taken in conjunction with Figure 9.
Figure 9  Error handler and state machine automaton
Description of the physical medium automaton

Figure 10 illustrates the physical medium automaton. This automaton receives information from the 3 transmitter automata and determines which message successfully arbitrates for use of the medium. The TX_FRAME urgent channel signals that a transmitter is attempting to transmit a frame on the medium. Once the time duration where two or more transmitter automata may simultaneously attempt to transmit, (i.e. during the SOF field) has elapsed the automata signals that the bus is no longer idle, via the shared variable Bus_Frame_Id, and signals to idle transmitters that a transmission has commenced via the SOF urgent channel. The remaining portion of the automata evaluates the result of arbitration and waits for the end of the successful message’s transmission. When the message has been transferred the active transmitter signals via the urgent channel ENDFRAME that the message has finished and signals this via the EOF urgent channel. The automaton then waits two bit times before returning the medium status to idle.

As Figure 10 illustrates control initialises in the committed location S0 and proceeds directly to the location labeled idle. This transition initialises the relevant shared variables to the state Bus_Idle. As can be seen the automation remains in idle until a transmission is initiated via the TXFRAME channel. The remainder of the automaton can be understood from Figure 10.

Figure 10   Physical Media Automaton

\[\text{Figure 10 Physical Media Automaton}\]

---

20 The two bit times are taken to be part of the EOF field of a CAN frame.
Description of a transceiver automata

Figure 11 illustrates a transceiver automaton. The system initialises and remains in location S1 until there is a transmission request or a transmission is detected. If a transmission is detected control is forced from the S1 location to the S3 location. When the arbitration phase of transmission is complete, the STABLE_ID urgent channel indicates that the successful message is now on the medium. The message identifier value is evaluated to determine whether the frame is a Reference Message (S8 → S21 → S12 → I_FS) or a normal message (S8 → S20 → S13 → I_FS). For standard messages a bit is set in the Rx_Bit_Array to indicate that this message has been received. If the message is a Reference Message then a Reference Mark is generated, the value of Cycle_Count is updated and the Cycle Time is reset to reflect the time mark when the SOF field of the Reference Message has occurred.

If a transmission is requested by the node’s scheduler automaton and automaton control rests in location S1, then either one of three transitions may occur depending on the context of this request.

1. If the request is for the transmission of a Reference Message then the transition to S0 is taken
2. If the request is for the transmission of a normal message and the transceiver is in error level 2 then the transition to S11 is taken
3. If the request is for the transmission of a normal message and the transceiver is in error level 0 or 1 then the transition taken is to location S2

Assuming that the request is for a normal message then the shared variable Pt_Tx_Message will contain the value of the identifier of this message. Provided the medium is idle and the transmitter is enabled then control moves to location S2. Control immediately leaves this location and signals via the urgent channel TX_Frame, to the medium, that it is starting to transfer a message on the network. The Bit_Clock clock is reset, this clock will ensure progression in the transmitter automaton. Depending on the identifier value of the frame being transmitted the automaton will select the appropriate transition on leaving location S18. The automaton then waits in the next location either until the message has successfully transferred or until the message fails the arbitration process. If the message fails the arbitration process, control will be forced to location S3 with activation of the FAILED_ARB urgent channel. If the message successfully arbitrates then it will remain in this location for the duration of the message transfer and then move to location S10 and signal the message termination via the ENDFRAME urgent channel. The MSC for this message will then be updated via the MSC_TX_OK urgent channel.
Description of the Matrix Cycle

Table 2 provides the MC schedule used in the model system\(^\text{21}\). All times are in NTUs. The TEW (Transmission Enable Window) is taken to be a constant interval of 8 NTUs for the system. Message lengths are inclusive of worst-case bit stuffing. The Gap time interval is taken to be zero. With reference to Table 2, as an example, BC0, TC 1 start at time 130 NTU and ends at time 290 NTU. The TEW starts at time 130 NTU and terminates 8 NTU later. The transmission trigger for the message in this column is activated at time zero. The receive trigger is activated at time 286 NTU, which is after the expected transmission completion of message 2. The message appearing in this column is 120 NTU in duration. As another example, a merged arbitrating message window appears in BC 1, formed by TCs 2 and 3. Messages from nodes one, two and three compete during this time interval to release messages 4, 7 and 11 respectively.

\(^{21}\) Note: ‘?’ in the context of this table represents ‘or’ and indicates that two or more messages are competing for medium access at this time.
Verification of the TTCAN protocol model

In this section we present the results of the analysis of the TTCAN protocol, which has been described in the previous sections. The TTCAN protocol system was created in the editor component of the UPPAAL tool suite and loaded into the verification component of the tool suite and various properties were examined. Conforming to current nomenclature, an implicit proposition at(A,l) will be denoted A.l. In addition invariance properties are of the form $\forall \beta$, where $\beta$ is a local property.

Table 3 provides the mapping of process assignment parameters to process definition parameters. In this case it is simply a renaming procedure. For example the automation Error_Containment_Node_1 is renamed for simulation and verification purposes as Er_H_S_1S. Thus the initialisation location for the error handler and error state machine automaton for node 1 is referred to as ‘Er_H_S_1S.Quiescent_State’ as can be seen in Figure 9.

<table>
<thead>
<tr>
<th>Process assignment</th>
<th>Process definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx_1S</td>
<td>Transceiver_Node_1( )</td>
</tr>
<tr>
<td>Sch_1S</td>
<td>Scheduler_Node_1( )</td>
</tr>
<tr>
<td>Er_H_S_1S</td>
<td>Transceiver_Node_1( )</td>
</tr>
<tr>
<td>Tx_2S</td>
<td>Transceiver_Node_2( )</td>
</tr>
<tr>
<td>Sch_2S</td>
<td>Scheduler_Node_2( )</td>
</tr>
<tr>
<td>Er_H_S_2S</td>
<td>Error_Containment_Node_2( )</td>
</tr>
<tr>
<td>Tx_3S</td>
<td>Transceiver_Node_3( )</td>
</tr>
<tr>
<td>Sch_3S</td>
<td>Scheduler_Node_3( )</td>
</tr>
<tr>
<td>Er_H_S_3S</td>
<td>Error_Containment_Node_3( )</td>
</tr>
<tr>
<td>Bus_S</td>
<td>Physical_Medium( )</td>
</tr>
</tbody>
</table>

Table 3 Process assignment and definition mapping
Verification of the error state machine and error handler mechanism in the context of a correctly configured MC was verified using the queries listed below. Location names as presented in property one below may be found in Figure 9 showing the B section of the combined error handler and state machine for potential time master 1. Location Er_H_S_1S.S39 represents a location only entered if an S1 error is active. Location Er_H_S_1S.S40 is only entered if an S2 error is active and so on. Properties two and three represent similar locations in the error state machine components of the remaining two node groups.

1. $\forall \Box \neg (Er_{\_H\_S\_1S}.S39 \lor Er_{\_H\_S\_1S}.S40 \lor Er_{\_H\_S\_1S}.S41)$
2. $\forall \Box \neg (Er_{\_H\_S\_2S}.S39 \lor Er_{\_H\_S\_2S}.S40 \lor Er_{\_H\_S\_2S}.S41)$
3. $\forall \Box \neg (Er_{\_H\_S\_3S}.S39 \lor Er_{\_H\_S\_3S}.S40 \lor Er_{\_H\_S\_3S}.S41)$

As these query locations represent the cumulative effect of errors within the system a large number of possible failures can be examined by observing a node’s reported error level. For a correctly configured MC the verification shows that all these properties were satisfied, as represented by the combination of queries 1, 2 and 3. These properties also infer the correctness of properties 4 to 10 below. Nevertheless these properties were also verified for completeness. A description of the variables appearing in these properties is available in the appendix to this paper. This result formally verifies that a system designed in accordance with the chosen interpretation\(^{22}\) of specification ISO11898-4 (draft) will never inadvertently enter an undesired error state. This means that the specification itself defines a network node implementation which has no ‘hidden’ execution traces into the aforementioned states for a correctly configured MC, in the previously stated context of this model.

4. $\forall \Box \neg (MSC\_DIFF\_1==1 \lor MSC\_DIFF\_2==1 \lor MSC\_DIFF\_3==1)$
5. $\forall \Box \neg (RX\_MSC7\_1==1 \lor RX\_MSC7\_2==1 \lor RX\_MSC7\_1==1)$
6. $\forall \Box \neg (TX\_MSC7\_1==1 \lor TX\_MSC7\_2==1 \lor TX\_MSC7\_1==1)$
7. $\forall \Box \neg (TX\_UNDER\_1==1 \lor TX\_UNDER\_2==1 \lor TX\_UNDER\_3==1)$
8. $\forall \Box \neg (TX\_OVER\_1==1 \lor TX\_OVER\_2==1 \lor TX\_OVER\_3==1)$
9. $\forall \Box \neg (WATCH\_1==1 \lor WATCH2==1 \lor WATCH3==1)$
10. $\forall \Box (E\_State\_1==0 \land E\_State\_2==0 \land E\_State\_3==0)$

The ‘deadlock free’ operation of the system was also verified using property 11 below\(^{23}\).

11. $\forall \Box \neg \text{deadlock}$

Verification of the error state machine and error handler mechanism in the context of an incorrectly configured MC was verified using the following queries:

12. $\forall \Box (Er_{\_H\_S\_1S}.S39 \land Er_{\_H\_S\_2S}.S39)$, with Node 3 absent
13. $\forall \Box (Er_{\_H\_S\_1S}.S39 \land Er_{\_H\_S\_3S}.S39)$, with Node 2 absent
14. $\forall \Box (Er_{\_H\_S\_2S}.S39 \land Er_{\_H\_S\_3S}.S39)$, with Node 1 absent
15. $\forall \Box \neg \text{(deadlock)}$, with two masters, and with one potential master and one slave
16. $\forall \Box (State\_1 == 0 \lor State\_1 == 1)$ phased startup of time masters, with node 2 starting early.

\(^{22}\) As the specification document is written in natural language, ambiguity, unintentional or otherwise, while allowing scope for implementers’ creativity, also provides scope for correct interpretation or otherwise.

\(^{23}\) Note that this semantic for deadlocks uses syntactic transitions, thus invariants on the target state have no influence on whether the source state is deadlocked or not.
The propositions 1 to 11 inclusive, listed above, were satisfied to be correct. Properties 12 – 14 failed to be satisfied as expected, due to the absent MC messages when the respective nodes were removed from the virtual network. Properties 15 and 16 were also satisfied. In addition to the afore listed properties numerous additional properties have been verified in the course of the model design process, as the model itself required verification of behaviour at the various levels of completeness.

**Computational resources**

The individual verification computations consumed varying amounts of computational time and memory depending on the system configuration and requirements specification. The system was verified on an i86 platform, Xeon 1.7 GHz processor, with 4GB of RAM, running LINUX Red Hat V7.1, kernel version 2.4.15. The UPPAAL verification engine was version 3.2 Beta 4 (3.1.64), June 2001. Table 4 summarizes approximate resource consumption for various configurations. As expected, activating the verification engine from the command line significantly reduces the system memory requirements, as the GUI is not used in this mode. Also worth noting is the fact that incorrectly enabled verifier options may produce an undesirably large state-space, as the table illustrates. The concentrated interleaving of actions, as illustrated by the final row of Table 4, may result in a state-space explosion.

<table>
<thead>
<tr>
<th>TTCAN model</th>
<th>Property tested</th>
<th>Memory required</th>
<th>Time required for verification</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 nodes</td>
<td>1 to 11</td>
<td>971MB</td>
<td>Approx. 140 min/property</td>
<td></td>
</tr>
<tr>
<td>2 nodes</td>
<td>12 to 14</td>
<td>39MB</td>
<td>Approx. 1 min/property</td>
<td></td>
</tr>
<tr>
<td>3 nodes</td>
<td>$\forall \Box \neg (Er_H_S_2S.S39)$</td>
<td>2044MB</td>
<td>Approx. 1143 min</td>
<td>Inappropriate verifier options</td>
</tr>
<tr>
<td>3 nodes</td>
<td>$\forall \Box \neg$ deadlock</td>
<td>&gt;3400MB</td>
<td>&gt; 30 min (system crashed)</td>
<td>‘Tight’ packing of messages</td>
</tr>
</tbody>
</table>

Table 4 Approximate verification resource consumption

**Conclusions and further work**

In this paper we have presented a general overview of the new time-triggered protocol, TTCAN. We posed a number of arguments justifying the use of formal methods, not only in the verification of completed software/hardware designs but also in the verification of original specifications. A brief introduction to the UPPAAL formal specification and formal verification tool suite was presented.

We have provided a formal model of the TTCAN protocol, described as a network of timed automata. These automata capture the essence of the protocol behaviour and may help elevate any mis-interpretations of the textural specification. A number of key properties of the protocol have been formally examined, including deadlock free operation. The work described in this paper is novel in that, generally speaking, ISO protocols have hitherto not been formally verified during the design phase of the specification. The removal of errors and flaws in the early stages of design pays enormous dividends both economically and in effort expended. As J. Author et al. have pointed out ambiguities in the specification itself are often an ‘innocent’ source of error [1xxi]. Fortunately, a formal specification has little scope for ambiguity or mis-interpretation even across natural language boundaries. It is therefore not unreasonable to propose that future international specifications are formally verified prior to release and that a formal specification should be included in the specification release documentation. The formal verification of the protocol

---

24 Thus, in a correctly configured system there is no possible way to enter an error state. Note, this excludes the possibility of external system perturbation, as described in the model restrictions.

25 Effectively omitting the interframe space defined by the CAN protocol.
Formal Verification of the TTCAN protocol

specification is independent of whether the implementation is realised in software or in hardware. To date the protocol has been realised in by NEC in software and by Bosch and Hitachi in hardware.

The mathematical models presented in this paper will now form the basis of continued investigation into the performance of the TTCAN protocol. Issues such as clock drift between nodes will be examined. A number of the model restrictions will be removed and a more detailed examination will be conducted. For instance an automaton can be introduced into the system which generates a bounded number of transmission failures on the medium, and the subsequent protocol error containment behaviour may be observed. Indeed, as the models created are quite flexible, actual systems may be simulated and verified prior to implementation. Critical parameters such as worst-case message latency may be examined for specific messages in bounded error conditions. Different MC configurations may be compared and application specific control loop requirements may be formally verified to be satisfied, or otherwise. A future model of an extended Level 2 implementation may be examined, although some semantic restrictions of the UPPAAL language may make the specification of this feature a complex task.

It will be agreed that formal verification may not be a ‘panacea’ for all the pitfalls encountered in the design of software and hardware systems. However, formal methods are a powerful tool offering many unique opportunities. Some engineers may argue that software and hardware systems have not yet evolved far enough to achieve true artificial intelligence and to perform tasks, which have a similar level of complexity as those which humans accomplish. However, another equally plausible argument is that engineers have not evolved enough in the use of software and hardware tools and techniques to achieve such goals. After all software and hardware are indeed engineering creations and are by the same token limited or otherwise by engineers themselves. The ‘bandwidth’ of the human is somewhat fixed, while the complexity of systems which we desire to create are increasingly challenging our capacity. In the future, we believe that using formal methods and ‘logic as a tool’ will help engineers to ‘compress’ system complexity, remove mis-interpretation and accelerate the evolution of both software and hardware systems.

Acknowledgement

The authors wish to gratefully acknowledge the assistance of Prof. Robin Sharp, Ass. Prof. Hongyan Sun, Ass. Prof. Hans Henrik Løvengreen, Edward Todirica and Per Friis of the Computer Science and Technology Division, Informatics and Mathematical Modeling Department, Technical University of Denmark, for their guidance and support. The authors wish to thank Enterprise Ireland, PEI - Circuits & Systems Research Centre and the University of Limerick for their continued support on research into automotive technologies. Thanks to Linus Torvalds, Rik van Riel, and Andrea Arcangeli for their support on Linux. Thanks to the UPPAAL team especially Professor Wang Yi, Oliver Möller, Johan Bengtsson and Dr. Paul Pettersson.
**APPENDIX**

**Description of System Constants**

Table 5 and Table 6 list and briefly describe the system constants, variables, clocks and communication channels. A number of parameters, which have similar function to those of node 1 are not listed here, as they simply have a different numeric index, i.e. 2 or 3 and essentially perform the same function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>1</td>
<td>Boolean expression</td>
</tr>
<tr>
<td>Off</td>
<td>0</td>
<td>Boolean expression</td>
</tr>
<tr>
<td>Msg_1</td>
<td>80</td>
<td>Duration in NTU for message identifier 1</td>
</tr>
<tr>
<td>Msg_2</td>
<td>80</td>
<td>Duration in NTU for message identifier 2</td>
</tr>
<tr>
<td>Msg_3</td>
<td>120</td>
<td>Duration in NTU for message identifier 3</td>
</tr>
<tr>
<td>Msg_4</td>
<td>130</td>
<td>Duration in NTU for message identifier 4</td>
</tr>
<tr>
<td>Msg_5</td>
<td>120</td>
<td>Duration in NTU for message identifier 5</td>
</tr>
<tr>
<td>Msg_6</td>
<td>100</td>
<td>Duration in NTU for message identifier 6</td>
</tr>
<tr>
<td>Msg_7</td>
<td>130</td>
<td>Duration in NTU for message identifier 7</td>
</tr>
<tr>
<td>Msg_8</td>
<td>100</td>
<td>Duration in NTU for message identifier 8</td>
</tr>
<tr>
<td>Msg_9</td>
<td>120</td>
<td>Duration in NTU for message identifier 9</td>
</tr>
<tr>
<td>Msg_10</td>
<td>130</td>
<td>Duration in NTU for message identifier 10</td>
</tr>
<tr>
<td>Msg_11</td>
<td>130</td>
<td>Duration in NTU for message identifier 11</td>
</tr>
<tr>
<td>Msg_12</td>
<td>120</td>
<td>Duration in NTU for message identifier 12</td>
</tr>
<tr>
<td>Msg_13</td>
<td>130</td>
<td>Duration in NTU for message identifier 13</td>
</tr>
<tr>
<td>Array_End</td>
<td>13</td>
<td>End of MSC array</td>
</tr>
<tr>
<td>Tx_Enable_Win</td>
<td>8</td>
<td>Duration of Tx_Enable window</td>
</tr>
<tr>
<td>Ref_ID</td>
<td>2</td>
<td>Largest Reference Message identifier</td>
</tr>
<tr>
<td>Ref_ID_1</td>
<td>1</td>
<td>Reference ID for node 1</td>
</tr>
<tr>
<td>Ref_ID_2</td>
<td>2</td>
<td>Reference ID for node 2</td>
</tr>
<tr>
<td>Init_Ref_Trig_Offset_1</td>
<td>8</td>
<td>Initial Reference Trigger Offset for node 1</td>
</tr>
<tr>
<td>Init_Ref_Trig_Offset_2</td>
<td>10</td>
<td>Initial Reference Trigger Offset for node 2</td>
</tr>
<tr>
<td>Ref_Trig_Zero</td>
<td>780</td>
<td>Reference Trigger with zero offset</td>
</tr>
<tr>
<td>Ref_Trig_Zero_P_offset_1</td>
<td>788</td>
<td>Reference Trigger plus Initial offset for node 1</td>
</tr>
<tr>
<td>Ref_Trig_Zero_P_offset_2</td>
<td>790</td>
<td>Reference Trigger plus Initial offset for node 2</td>
</tr>
<tr>
<td>Gap_Time</td>
<td>1565</td>
<td>Maximum Gap time interval</td>
</tr>
<tr>
<td>Gap_T_O_1</td>
<td>1573</td>
<td>Gap_Time plus initial Offset for node 1</td>
</tr>
<tr>
<td>Gap_T_O_2</td>
<td>1575</td>
<td>Gap_Time plus initial Offset for node 2</td>
</tr>
<tr>
<td>Gap_T_O_3</td>
<td>1605</td>
<td>Gap_Time plus initial Offset for node 3</td>
</tr>
<tr>
<td>Watch_Trig</td>
<td>1700</td>
<td>Time out watch dog trigger</td>
</tr>
<tr>
<td>Synchronisation</td>
<td>0</td>
<td>Operation State</td>
</tr>
<tr>
<td>A_T_Master</td>
<td>1</td>
<td>Operation State: Actual Time Master</td>
</tr>
<tr>
<td>P_Master</td>
<td>2</td>
<td>Operation State: Potential time Master</td>
</tr>
<tr>
<td>Time_Receiver</td>
<td>3</td>
<td>Operation State: Time Receiving node</td>
</tr>
</tbody>
</table>

**Table 5 System constant declarations**
Formal Verification of the TTCAN protocol

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start_Tx_Col_0</td>
<td>0</td>
<td>Start of transmission column 0</td>
</tr>
<tr>
<td>Rx_Trigger_Col_0</td>
<td>126</td>
<td>Receive trigger column 0</td>
</tr>
<tr>
<td>Start_Tx_Col_1</td>
<td>130</td>
<td>Start of transmission column 1</td>
</tr>
<tr>
<td>End_Tx_Enable_Col_1</td>
<td>138</td>
<td>End of Tx_Enable Window for Tx. Column 1</td>
</tr>
<tr>
<td>Rx_Trigger_Col_1</td>
<td>286</td>
<td>Receive trigger column 1</td>
</tr>
<tr>
<td>Start_Tx_Col_2</td>
<td>290</td>
<td>Start of transmission column 2</td>
</tr>
<tr>
<td>End_Tx_Enable_Col_2</td>
<td>298</td>
<td>End of Tx_Enable Window for Tx. Column 2</td>
</tr>
<tr>
<td>Rx_Trigger_Col_2</td>
<td>446</td>
<td>Receive trigger column 2</td>
</tr>
<tr>
<td>Start_Tx_Col_3</td>
<td>450</td>
<td>Start of transmission column 3</td>
</tr>
<tr>
<td>End_Tx_Enable_Col_3</td>
<td>458</td>
<td>End of Tx_Enable Window for Tx. Column 3</td>
</tr>
<tr>
<td>Rx_Trigger_Col_3</td>
<td>606</td>
<td>Receive trigger column 3</td>
</tr>
<tr>
<td>Start_Tx_Col_4</td>
<td>610</td>
<td>Start of transmission column 4</td>
</tr>
<tr>
<td>End_Tx_Enable_Col_4</td>
<td>618</td>
<td>End of Tx_Enable Window for Tx. Column 4</td>
</tr>
<tr>
<td>Rx_Trigger_Col_4</td>
<td>766</td>
<td>Receive trigger column 4</td>
</tr>
<tr>
<td>Expected_Tx_Count_1</td>
<td>1</td>
<td>Expected number of exclusive transmissions for node 1</td>
</tr>
<tr>
<td>Expected_Tx_Count_2</td>
<td>5</td>
<td>Expected number of exclusive transmissions for node 2</td>
</tr>
<tr>
<td>Expected_Tx_Count_3</td>
<td>2</td>
<td>Expected number of exclusive transmissions for node 3</td>
</tr>
<tr>
<td>Max_Offset</td>
<td>10</td>
<td>Max. Ref Trigger Offset (Should be 127)</td>
</tr>
<tr>
<td>MSC_Max</td>
<td>7</td>
<td>Upper bound on MSC</td>
</tr>
<tr>
<td>Bus_Idle</td>
<td>14</td>
<td>Indicates network idle</td>
</tr>
<tr>
<td>Bus_Not_Idle</td>
<td>15</td>
<td>Indicates network busy</td>
</tr>
</tbody>
</table>

Table 6  System constant declarations (continued)

Description of System Clocks

Table 7 provides a list of the system clocks accompanied by a brief description.

<table>
<thead>
<tr>
<th>Clock</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus_Clock</td>
<td>Ensures progression in the physical medium</td>
</tr>
<tr>
<td>Cycle_Time_1</td>
<td>Counts Cycle Time in node 1</td>
</tr>
<tr>
<td>Bit_Clock_1</td>
<td>Ensures progression in the transceiver automata of node 1</td>
</tr>
<tr>
<td>Sync_1</td>
<td>Synchronising clocks for potential time master 1</td>
</tr>
</tbody>
</table>

Table 7  System Clocks
**Description of system synchronisation channels**

Table 8 provides a list of the system synchronisation channels accompanied by a brief description.

<table>
<thead>
<tr>
<th>Channel name</th>
<th>Channel type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOF</td>
<td>Urgent</td>
<td>Start Of Frame pulse from bus</td>
</tr>
<tr>
<td>TXFRAME</td>
<td>Urgent</td>
<td>Transceiver transfer of message to bus</td>
</tr>
<tr>
<td>EOF</td>
<td>Urgent</td>
<td>End Of Frame signal from bus</td>
</tr>
<tr>
<td>ENDFRAME</td>
<td>Urgent</td>
<td>Transceiver ending message to bus</td>
</tr>
<tr>
<td>REF_MARK_1</td>
<td>Urgent</td>
<td>Reference mark detected by node 1</td>
</tr>
<tr>
<td>TX_TRIGGER_1</td>
<td>Urgent</td>
<td>Transmit trigger from scheduler to node</td>
</tr>
<tr>
<td>TX_TRIGGER_E_1</td>
<td>Urgent</td>
<td>Increment Tx_trigger count in error handler</td>
</tr>
<tr>
<td>RX_TRIGGER_1</td>
<td>Urgent</td>
<td>Receive trigger, Scheduler to MSC automation</td>
</tr>
<tr>
<td>TX_OK_1</td>
<td>Urgent</td>
<td>Transmission completed without error</td>
</tr>
<tr>
<td>FAILED_ARB_1</td>
<td>Urgent</td>
<td>Arbitration failed signal from Bus</td>
</tr>
<tr>
<td>MSC_TX_OK_1</td>
<td>Urgent</td>
<td>Decrement MSC as transmission was successful</td>
</tr>
<tr>
<td>MSC_TX_NOK_1</td>
<td>Urgent</td>
<td>Increment MSC as Tx. was not successful</td>
</tr>
<tr>
<td>CHECK_MSC_1</td>
<td>Urgent</td>
<td>Signal to error handler to evaluate MSCs</td>
</tr>
</tbody>
</table>

**Table 8** System synchronisation channels
**Description of System Variables**

Table 9 provides a list of the system variables accompanied by a brief description.

<table>
<thead>
<tr>
<th>Integer bounds</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int[0,15]</td>
<td>Bus Frame Id</td>
<td>Identifier of message currently on physical medium</td>
</tr>
<tr>
<td>int[0,15]</td>
<td>N Frame Id 1</td>
<td>Message identifier entering arbitration, source node 1</td>
</tr>
<tr>
<td>int[0,17]</td>
<td>Ref Trigger 1</td>
<td>Node 1’s transmission trigger for the reference Message</td>
</tr>
<tr>
<td>int[0,10]</td>
<td>Ref Trig_Offset_1_P</td>
<td>Positive Offset component for the reference trigger of node 1</td>
</tr>
<tr>
<td>int[0,10]</td>
<td>Ref Trig_Offset_1_N</td>
<td>Negative Offset component for the reference trigger of node 1</td>
</tr>
<tr>
<td>int[750,795]</td>
<td>C_Tick 1</td>
<td>Artificial clock for synchronisation of node 1</td>
</tr>
<tr>
<td>int[0,4]</td>
<td>State 1</td>
<td>Node state, i.e. actual time master, potential, etc. node for node 1</td>
</tr>
<tr>
<td>int[0,3]</td>
<td>E State 1</td>
<td>Status of a node 1’s Error state machine</td>
</tr>
<tr>
<td>int[0,4]</td>
<td>Pt Tx Message 1</td>
<td>Pointer to the message next to be transmitted by node 1</td>
</tr>
<tr>
<td>int[0,9]</td>
<td>Pt Tx Message 2</td>
<td>Pointer to the message next to be transmitted by node 2</td>
</tr>
<tr>
<td>int[9,14]</td>
<td>Pt Tx Message 3</td>
<td>Pointer to the message next to be transmitted by node 3</td>
</tr>
<tr>
<td>int[0,2]</td>
<td>Cycle Count 1</td>
<td>Cycle count for node 1</td>
</tr>
<tr>
<td>int[0,2]</td>
<td>Ref Cycle Count</td>
<td>Distributed Cycle Count value in Reference Message</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>Next is Gap</td>
<td>Distributed Next is Gap bit in Reference Message</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>Disable Tx 1</td>
<td>Disable message transmission in node 1</td>
</tr>
<tr>
<td>int[0,2]</td>
<td>Tx Count 1</td>
<td>Counter of transmission triggers in a node 1 Matrix Cycle</td>
</tr>
<tr>
<td>int[0,6]</td>
<td>Tx Count 2</td>
<td>Counter of transmission triggers in a node 2 Matrix Cycle</td>
</tr>
<tr>
<td>int[0,3]</td>
<td>Tx Count 3</td>
<td>Counter of transmission triggers in a node 3 Matrix Cycle</td>
</tr>
<tr>
<td>int[0,13]</td>
<td>Rx Message 1</td>
<td>Indicates the message ID expected at an RX-trigger event for node 1</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>MSC DIFF 1</td>
<td>MSC difference &gt; 2 error for node 1</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>RX MSC7 1</td>
<td>Rx MSC is 7 error for node 1</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>TX MSC7 1</td>
<td>Tx MSC is 7 error</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>TX UNDER 1</td>
<td>Tx Underflow error</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>TX OVER 1</td>
<td>Tx Overflow error</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>WATCH 1</td>
<td>Reference trigger watchdog error</td>
</tr>
<tr>
<td>int[0,7]</td>
<td>Largest MSC 1</td>
<td>Variable used in Error state machine</td>
</tr>
<tr>
<td>int[0,7]</td>
<td>Smallest MSC 1</td>
<td>Variable used in Error state machine</td>
</tr>
<tr>
<td>int[0,13]</td>
<td>Pointer 1</td>
<td>Variable used in Error state machine</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>S One 1</td>
<td>Variable used in Error state machine</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>S Two 1</td>
<td>Variable used in Error state machine</td>
</tr>
<tr>
<td>int[0,1]</td>
<td>S Three 1</td>
<td>Variable used in Error state machine</td>
</tr>
</tbody>
</table>

**Table 9 System Integer Variables**

<table>
<thead>
<tr>
<th>Array of</th>
<th>Array Size</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int[0,1]</td>
<td>[14]</td>
<td>Rx Bit Array 1</td>
<td>Receive bit array for confirming message receipt</td>
</tr>
<tr>
<td>int[0,7]</td>
<td>[14]</td>
<td>MSC 1</td>
<td>Array of MSC requesters</td>
</tr>
</tbody>
</table>

**Table 10 System Variable Array**
References

[34] F. Hartwich, B. Müller, T. Führer, R. Hugel, “CAN Network with Time Triggered Communication” 7th International CAN Conference (ICC), Amsterdam, Netherlands, Oct. 24-25, 2000. Available at: http://www.can.bosch.com/content/Literature.html
T. Führer, B. Müller, F. Hartwich, and R. Hugel. Time Triggered CAN (TTCAN). Society of Automotive Engineers (SAE) paper order no. 2001-01-0073
SPIN Available at http://cm.bell-labs.com/cm/cs/what/spin/
The PVS specification and verification system. Available at http://pvs.csl.sri.com/
Kronos. Available at http://www-verimag.imag.fr/TEMPORISE/kronos/index-english.html
M. Lindahl, P. Pettersson, and W.Yi. Model-Checking of Real-Time Systems
Formal Verification of the TTCAN protocol

R. Alur and D. Dill. The Theory of Timed Automata. LNCS 600, pp 45 - 73