Defence Science Journal, Vol. 73, No. 1, January 2023, pp. 41-50, DOI : 10.14429/dsj.73.18449

Formal Modelling and Verification of the Clock Synchronisation Algorithm of FlexRay

Shimmi Asokan[#,*], K.H. Kochaleema[$] and G. Santhosh Kumar[#]

[#]*Department of Computer Science, Cochin University of Science and Technology, Kochi - 682 022, India*
[$]*DRDO-Naval Physical and Oceanographic Laboratory (NPOL), Kochi - 682 004, India*
[*]*E-mail: shimmideepak@gmail.com*

ABSTRACT

The hundreds of electronic control devices used in an automotive system can effectively communicate with one another, thanks to an in-vehicle network (IVN) like FlexRay. Even though every node in the network will be running on its local clock, a global notion of time is essential. The clock synchronisation algorithm accomplishes this global time between the nodes in FlexRay. In this era of self-driving cars, the vehicle's safety is paramount. For the vehicle to operate safely and smoothly, timely communication of information is critical, and the clock synchronisation algorithm plays a vital role in this. It is essential to formally test the clock synchronisation algorithm's correctness. This paper attempts to model and verify the clock synchronisation algorithm of FlexRay using formal methods, which in turn enhance the reliability of safety-critical automotive systems. The clock synchronisation is modelled as a network of six timed automata in the UPPAAL model checker. Three system models were developed, a model for an ideal clock, another for a drifting clock, and a third model considering propagation delay. The precision of the clocks is verified to be within the prescribed limits. Simulation studies are also conducted on the model to ensure that the clock's drift is always within the precision.

Keywords: Formal verification; FlexRay; Model checking; UPPAAL; Clock synchronisation

1. INTRODUCTION

High-end automobiles manufactured today are equipped with several safety and performance features. Advanced driver assistance system, adaptive cruise control, anti-lock braking system, and engine control are a few to mention. Manufacturers implement these features as independent electronic control units (ECUs). Modern cars have hundreds of ECUs, and effective communication between them is essential for the electronic systems to provide functionalities. This communication is established with IVN technologies like Local Interconnect Network[1], Controller Area Network (CAN)[2], FlexRay[3], Media Oriented System Transport[4], etc. FlexRay provides a bandwidth of 10Mbps, which is better than other IVN[5]. FlexRay is reliable compared to other protocols and is used for brake-by-wire, steer-by-wire and shift-by-wire, which are safety-critical.

FlexRay was developed by the consortium consisting of Freescale Semiconductor, Robert Bosch, NXP Semiconductors, BMW, Volkswagen, Daimler and General Motors as its core members[3]. FlexRay standards are currently a collection of ISO standards, ISO 17458-1 to 17458-5[6]. Several works on FlexRay have been reported in the literature recently[7-12]. Various researchers have modelled the FlexRay startup mechanism and have formally verified its properties like reachability and liveness[13-15]. Guo *et al.*[16] modelled a framewuork for formally modelling automotive systems that used FlexRay and verified the features associated with sending and receiving frames. The authors also developed a reusable framework for systems that employ both CAN and FlexRay for communication[17]. The authors described an abstraction for the same and evaluated the validity of the abstraction and the performance of the framework[18]. The authors considered three topologies for implementing IVN systems and the timed properties were checked for all three cases. To ensure the safety of high-end cars, the ECUs should be synchronised to deliver messages at the appropriate time for carrying out various activities. Once the nodes begin running, the node's clocks start to drift. The clock synchronisation algorithm serves the purpose of synchronising between the cluster nodes[3]. A couple of works on the formal verification of a clock synchronisation algorithm for CAN were reported by Navas, *et al.*[19,20]. Time-Triggered CAN was modelled as a system of timed automata by Leen[21-22] *et al*. Steiner, *et al.* reported a fully automated proof of the Time-Triggered Ethernet (TTE) clock synchronisation[23]. They also modelled the compression function of TTE and formally verified its correctness[24]. The approach for clock synchronisation in CAN and TTE differs from the distributed approach used in FlexRay.

Hanzlik[25] assessed the performance and stability of the FlexRay clock synchronisation algorithm through simulation using the tool SIDERA. A cluster of 15 nodes was considered for the analysis, and the performance of the algorithm was determined for different clock drift rates. The clock drift rates considered were stable clock drift rates, immediate clock drift

Received : 22 August 2022, Revised : 16 December 2022
Accepted : 16 January 2023, Online published : 10 March 2023

rate change, linear clock drift rate change and oscillating clock drift rate change. An experimental evaluation of FlexRay clock synchronisation using the deterministic replay of a bus traffic approach was done by Armengaud[26]. These works are based on simulation and do not use formal methods. Simulation alone does not guarantee that the system is error-free. Formal verification guarantees the clock synchronisation algorithm's correctness and thus makes it reliable for use in the automotive industry for safety-critical functionalities.

The clock synchronisation algorithm of FlexRay is modelled, simulated and verified using formal methods in this paper. UPPAAL[27-28] model checker is used. The paper's contributions can be summarised as (i) formal modelling of the FlexRay clock synchronisation algorithm as a network of timed automata in UPPAAL. Three system models were developed: the first model mimics an ideal clock, the second simulates a drifting clock, and the third model integrates the propagation delay experienced when sending and receiving messages. (ii) verification of the precision property, as well as the system's safety property of being free of deadlock. (iii) simulation of the models, demonstrating that the clock's precision is always within the prescribed limits. The paper is organised as follows. Section 2 describes how FlexRay performs synchronisation of clocks. A summary of the UPPAAL model checker is also provided. Our model of the FlexRay clock synchronisation algorithm is detailed in section 3. The results and discussion in section 4 elaborate on the simulation and verification results. The conclusion of the work and some directions for future study are presented in Section 5.

2. BACKGROUND

Multiple ECUs or nodes connected by a bus make up a FlexRay cluster. FlexRay employs time division multiple access (TDMA) for triggering transitions. The communication cycle has a static segment, divided into static slots, and each slot is assigned to a node. Every node in a network will be running on its local clock, and the nodes should have a common understanding of time. In a cluster, it should be ensured that all the communication cycles are of equal length and begin at the same point. Also, all the static slots should start at the same point in the communication cycle. The clock synchronisation algorithm ensures this, and the following subsections describe the clock synchronisation process of FlexRay[3].

2.1 Clock Synchronisation in FlexRay

Cycles, macroticks and microticks are used to represent time in FlexRay. With the use of the variables vCycleCounter, vMacrotick and vMicrotick, a node's time can be represented. The basic unit of time is the microtick, whose source is the oscillator clock tick. A macrotick is made up of an integral number of microticks. An integral number of macroticks constitutes a cycle and is represented using the variable gMacroPerCycle. Each cycle and each node in a cluster will use the same value for this parameter. At any one time, a cluster's nodes should all have the same cycle number. The cycles are counted, and the variable vCycleCounter holds the current cycle number of a node[3].

There are two notions of time, global and local time. The local perspective of the global time from a node is called its global time. The clock time of the node obtained from the node's oscillator is its local time. The clock synchronisation algorithm enables the node to adjust its local time to global time. A distributed clock synchronisation mechanism is used in FlexRay. The algorithm for clock synchronisation ensures that the time difference between a cluster's nodes consistently remains within the precision. The time differences between two node's clocks are of two types, offset (phase) difference and rate (frequency) difference. The macrotick generation process (MTG) and the clock synchronisation process (CSP) are the two stages of FlexRay's clock synchronisation algorithm. The cycle and macrotick counters are controlled by MTG, which is also responsible for applying the rate and offset correction values. CSP is responsible for the initialisation of time at the start of the cycle, and it measures the deviation of the time between the node's clocks and stores the value in the deviation table. From these deviation values, the offset and rate correction values are also computed by the CSP process. The steps in the clock synchronisation of FlexRay are depicted in Fig. 1.

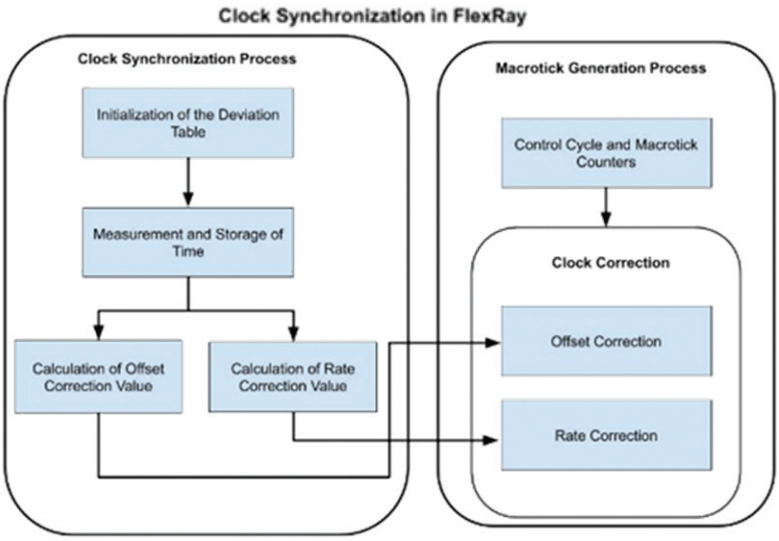

Figure 1. Clock synchronisation of FlexRay.

2.1.1 Initialisation

A significant step in calculating offset and rate correction values is to measure the deviation values between the local time of a node and the local time of every other node in the cluster. The deviation values measured during every cycle are stored in a data structure dev_table with N rows, where N is the number of SYNC nodes in the cluster. This table consists of an even part that stores the value measured during the even cycle and an odd part that stores the value measured during the odd cycle. During initialisation, the even part of dev_table is reset during the even communication cycle, and in the odd cycle, the odd part is reset.

2.1.2 Measurement and Storage of Time

Every slot in the static segment consists of a static slot action point (SAP). A transmitting node starts the frame transmission only when this point is reached. A signal is generated by the media access control (MAC) when its timer reaches the SAP. When CSP receives this signal, it takes a timestamp and saves it in a variable SAP.

The receiving node captures a timestamp as part of the frame reception, serving as a secondary time reference point (STRP). The primary time reference point (PTRP), the offset and the deviation of the clock are computed as in Eqn. (1), Eqn. (2) and Eqn. (3), respectively. The DecodingCorrection value can be 18 to 143 microticks, and DelayCompensation can be 0 to 200 microticks as per the specification[3].

$$PTRP = STRP - Offset \tag{1}$$

$$Offset = DecodingCorrection + DelayCompensation \tag{2}$$

$$Deviation = PTRP - SAP \tag{3}$$

The deviation value computed during a cycle will be stored in the corresponding part of the dev_table. Every node will have an array of measurement values after the static part. The values in this array are used to compute the correction value using the fault-tolerant midpoint algorithm (FTMPA)[3].

2.1.3 Calculation of Offset Correction Value

Offset correction ensures that the difference in real time between the points in time reached by different clocks should be as small as possible. The offset correction value is computed as follows.

- If the communication cycle is even, the deviation values computed during the even cycle are used to compute the offset correction value. Otherwise, the deviation values computed during the odd cycle are used.
- The corresponding deviation values are copied into an array, the FTMPA is executed on the array's contents, and the correction term, *vOffsetCorrection,* is computed.
- Check this value against the specified limits:
  a. If $vOffsetCorrection < -pOffsetCorrectionOut$, set $vOffsetCorrection = -pOffsetCorrectionOut$
  b. If $vOffsetCorrection > pOffsetCorrectionOut$, set $vOffsetCorrection = pOffsetCorrectionOut$

2.1.4 Calculation of Rate Correction Value

Rate correction ensures that all cycles of all clocks have the same length. The steps in calculating the rate correction value are as follows.

- The difference between the deviation values computed during each slot is found and stored in an array.
- The FTMPA is executed, and the correction term, *vRateCorrection,* is computed.
- Damping value, *pClusterDriftDamping* is applied to *vRateCorrection*
  a. If $vRateCorrection \geq pClusterDriftDamping$ set $vRateCorrection = vRateCorrection - pClusterDriftDamping$
  b. If $vRateCorrection \leq -pClusterDriftDamping$ set $vRateCorrection = vRateCorrection + pClusterDriftDamping$
  c. Else $vRateCorrection = 0$
- Check *vRateCorrection* against the specified limits.
  a. If $vRateCorrection < -pRateCorrectionOut$, set $vRateCorrection = -pRateCorrectionOut$
  b. If $vRateCorrection > pRateCorrectionOut$, set $vRateCorrection = pRateCorrectionOut$

2.2 UPPAAL

UPPAAL model checker takes two inputs, the model of the system to be verified and the properties the system has to satisfy. The properties that the system must satisfy are modelled using timed computation tree logic (TCTL), and the system is represented as a collection of timed automata. Rigorous verification of the property for all possible runs of the system is performed. If it is satisfied for all possible runs, the model checker produces as output "the property is satisfied". If the property is not satisfied for at least one run, it produces as output "the property is not satisfied", and a counterexample is generated[27-28]. The limitation of using UPPAAL in verifying clock synchronisation algorithms is that the value of a clock variable cannot be read. Huang, X describes an integer clock model and demonstrates it by verifying the correctness properties of the Timing-sync Protocol for Sensor Networks[29].

3. FORMAL MODELLING OF THE FLEXRAY CLOCK SYNCHRONISATION ALGORITHM

Starting a FlexRay cluster requires the presence of at least three cold-start nodes. In this work, a cluster consisting of four nodes is considered for modelling. The state space explosion is managed by restricting the nodes to four. The assumption is that all the nodes initially have a synchronised clock. The model developed comprises six automata, an Oscillator, Observer, MAC, Node, CSP and MTG.

All the automata are instantiated for every node in the network. Node(0) to Node(3) corresponds to the four nodes in the network. Oscillator(0), Observer(0), CSP(0), MTG(0) and MAC(0) are the instances of the various automata that correspond to the Node(0). Three different models are considered for simulation and verification. The first model assumes that all nodes are always synchronised, while the second considers that nodes 1, 2, and 3's clocks drift over time. In model 2, only the Oscillator and Observer automata are remodelled. Models 1 and 2 do not experience any delays while receiving the frame. The node automaton in the third

model is remodelled to incorporate the delay, which comprises the decoding correction and delay compensation.

3.1 Modelling the Clock in FlexRay

A node's clock is modelled using Oscillator and Observer automata.

3.1.1 Oscillator Automaton

The Oscillator automaton shown in Fig. 2 is used to model a node's clock pulse generator. The automaton has only a single state with an invariant t<=nsamp, where t is a clock variable, and nsamp is used to simulate the frequency of the node's clock. When t>=nsamp, a self-transition occurs, and the clock t is reset to 0. The transition generates a broadcast synchronisation clk_tck[i]! which controls the clock of the Observer(i) automaton. An instance of the Oscillator is created for every node in the cluster, and thus the existence of a local clock is modelled for each node.

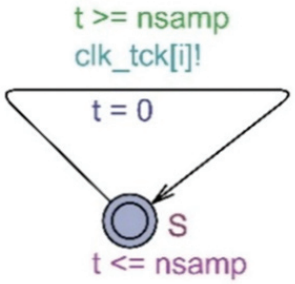

Figure 2. Oscillator Automaton.

The clocks in real-time may drift apart. The Oscillator and Observer automata are modified, as shown in Fig. 4 and Fig. 5, respectively, to model clock drifts. It is modelled such that the Observer ignores every m^{th} sample of the clock during specific cycles and does not increment the vMicrotck value during that pulse. The value of m for each node is stored in the array skip[N]. These modified automata are used in Model 2 and Model 3.

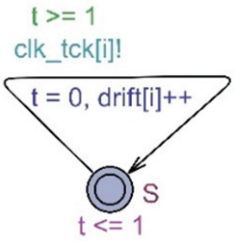

Figure 4. Oscillator for drifting clock.

3.2 MAC Automaton

The *MAC* automaton in Fig. 6 models the communication cycle of a node and controls the node's turn to access the media and send a frame. The shared media is implemented as a structure with two fields: a frame field and a status field that shows whether the bus is in use or not. A FlexRay communication cycle is made up of a static segment, a dynamic segment, a symbol window and network idle time (NIT)[3]. Most safety-critical communication is done during the static segment, and the MAC automaton models only this segment and NIT. MAC(i) automaton interacts with the Node(i), CSP(i) and

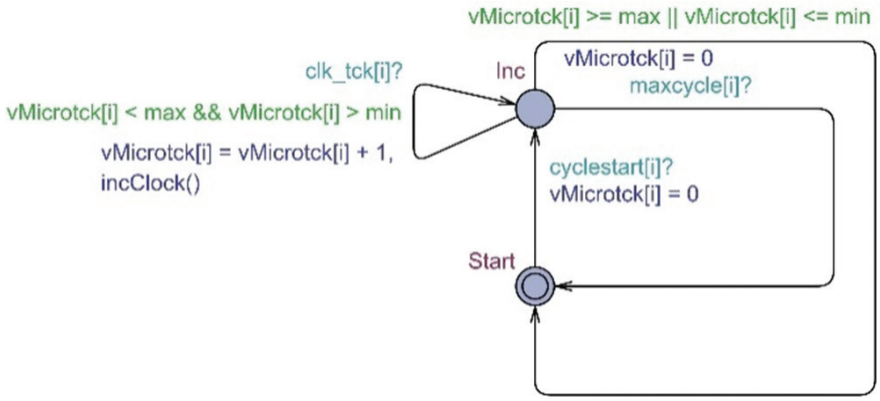

Figure 3. Observer automaton.

3.1.2 Observer Automaton

The Observer automaton depicted in Fig. 3 simulates the local time of a node's clock. UPPAAL does not permit to read the values of the clock variables. To implement the clock synchronisation it is required to take the timestamp at several instances. The variables vMicrotck[i] and vMacrotck[i] are the microtick and the macrotick counters and are of type integer. The value of *vMicrotck[i*] is read as the value of the clock for node i when a timestamp is to be taken. The variable vMicrotck[i] is incremented at every clock pulse when it receives a message via clk_tck[i]?. The variable vMacrotck[i] is incremented at every n^{th} microtick, where n microticks constitute a macrotick. Incrementing vMacrotck is implemented as a function in the Observer automaton.

MTG(i) automata. The MAC(i) automaton issues a cyclestart[i]! signal, which triggers the MTG(i) for the node. It also sets the values of firstcycle[i] and ecycle[i] to true, indicating cycle0. The automaton then spends in the location Wait_for_Slot for a time equal to slottime[i], and when this time expires, it is the i^{th} node's turn to access the media. The automaton waits in the location Wait_APO for gdAPO time before it issues the static slot action point signal, ssap!. The ssap! synchronises MAC(i) with the Node(i) and CSP(i) automata. MAC(i) then issues the slot[i]! signal, which Node(i) receives to send a frame. When the node's static slot expires, MAC(i) resets the contents on the bus by executing the resetBus() function. MAC(i) then waits in the Channel_Released location for a time vStSeg[i] for the static segment to complete. It then issues the sscompleted[i]!,

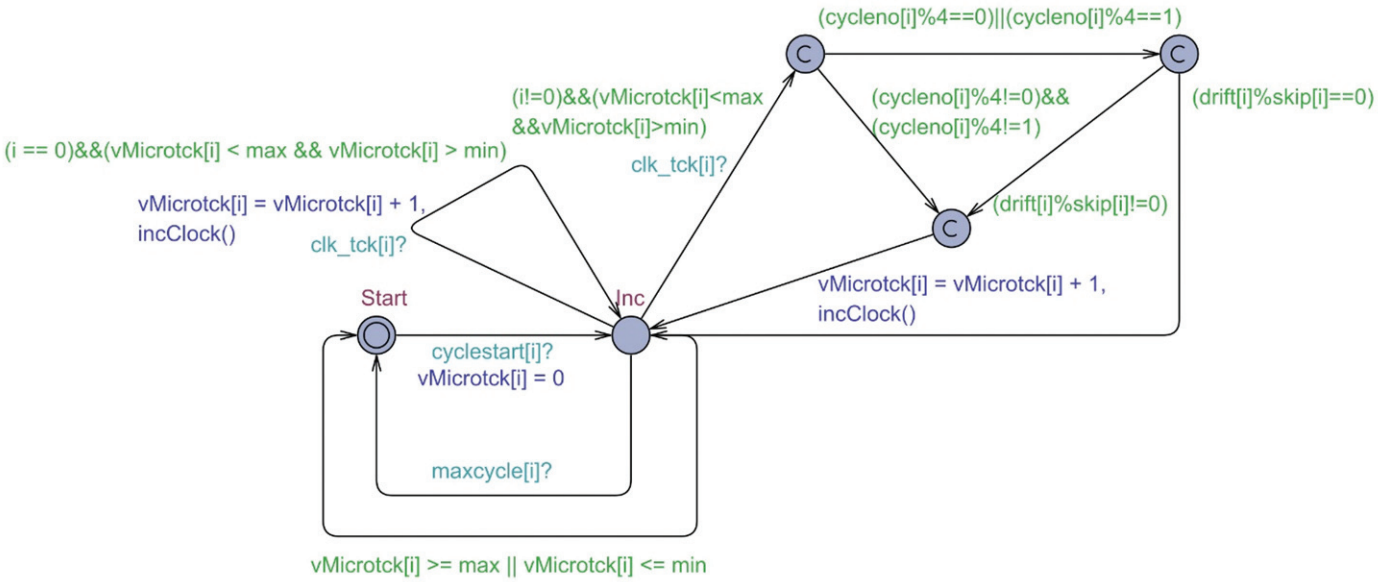

Figure 5. Observer for drifting clock.

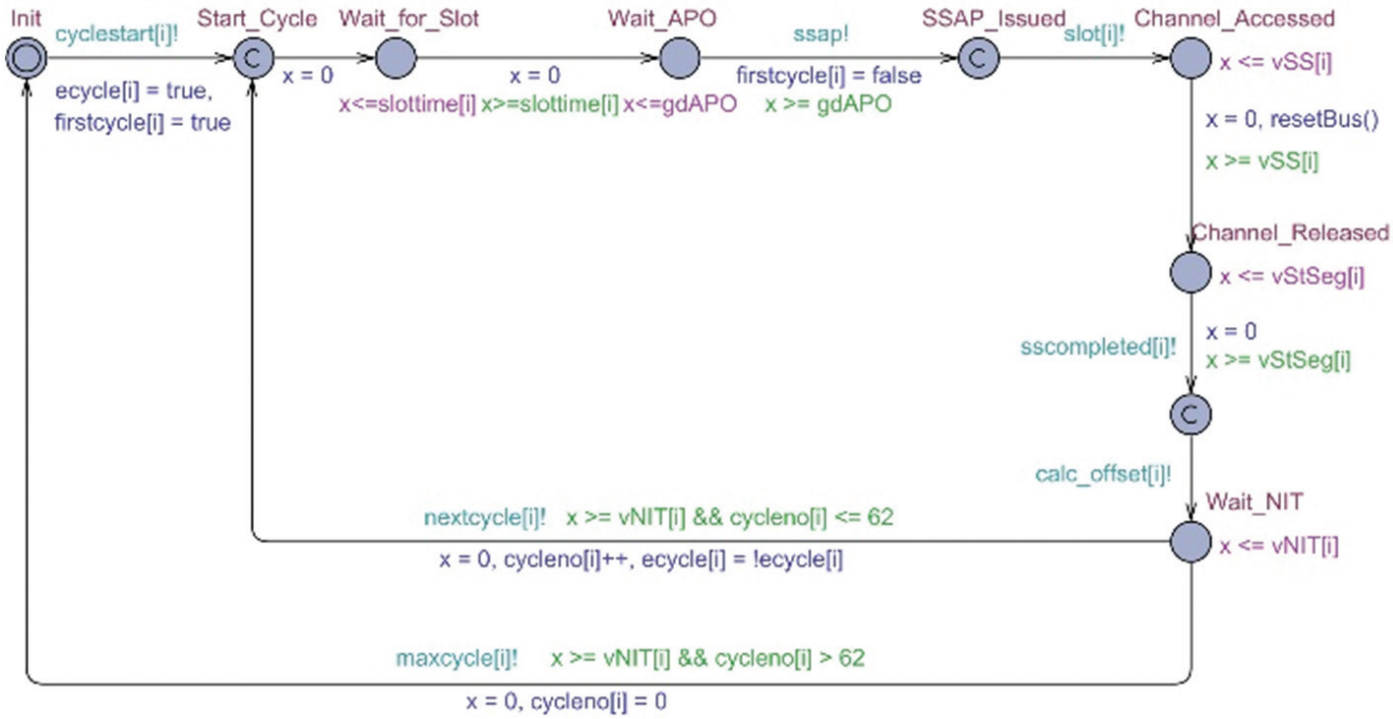

Figure 6. MAC automaton.

which will be received by the MTG(i) automaton. MAC(i) then generates the calc_offset[i]! to the CSP(i) to initiate the offset calculation. The automaton then spends time vNIT in the location NIT and then iterates back to the Start_Cycle location to start the next communication cycle. On this path, it resets the clock x to 0 and increments the cycleno[i]. It also negates the ecycle[i] because an even cycle will be followed by an odd cycle and vice versa. A FlexRay network permits a maximum of 63 cycles, so the cycleno[i] ranges from 0 to 62, and when the value is 63, the automaton resets to the Init location.

3.3 Node Automaton

The Node automaton shown in Fig. 7 models the node's behaviour of sending and receiving frames. When Node(i) receives the ssap? signal from MAC(i), it takes the timestamp and stores it in sap[i]. Node(i) then waits in the Ready state for either its slot to send a frame or till some other node accesses the media and sends a frame. If it is node i's turn to access the media, Node(i) will receive a slot[i]? signal from the MAC automaton and will transmit the frame. The node sends sendframe! signal while it transmits the frame, which is received by all other nodes in the cluster. The signal triggers the receivers to read the contents of the bus and record the timestamp at which

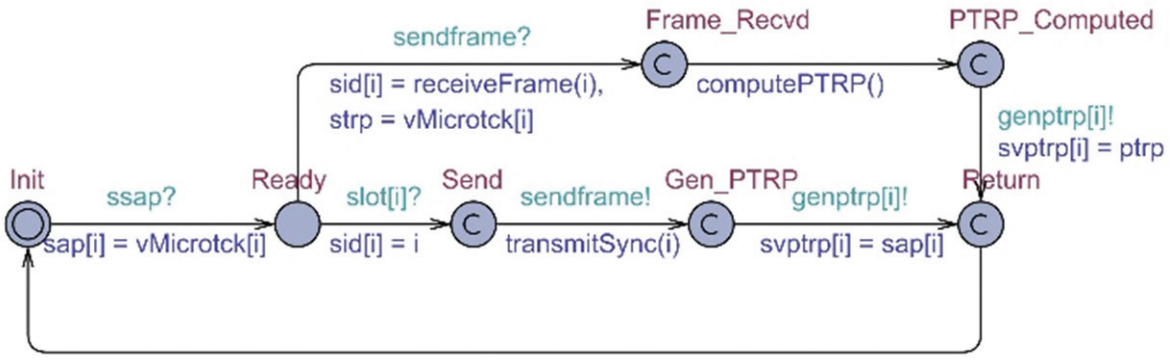

Figure 7. Node automaton.

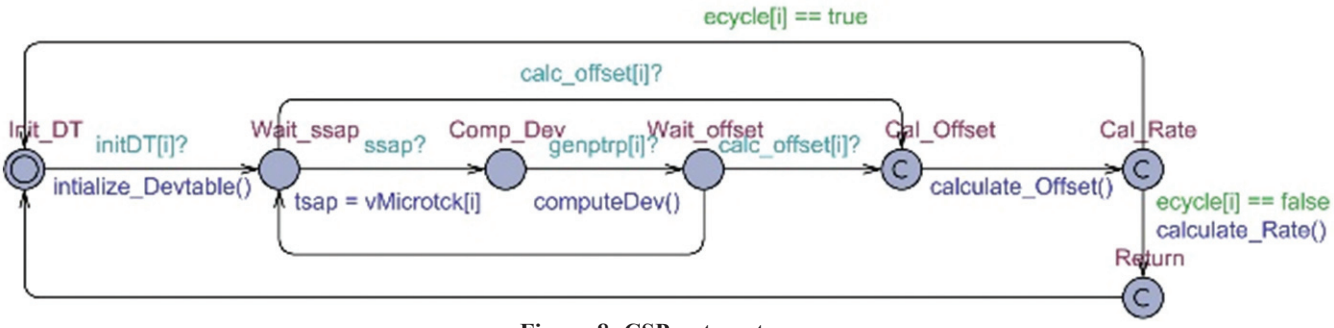

Figure 8. CSP automaton.

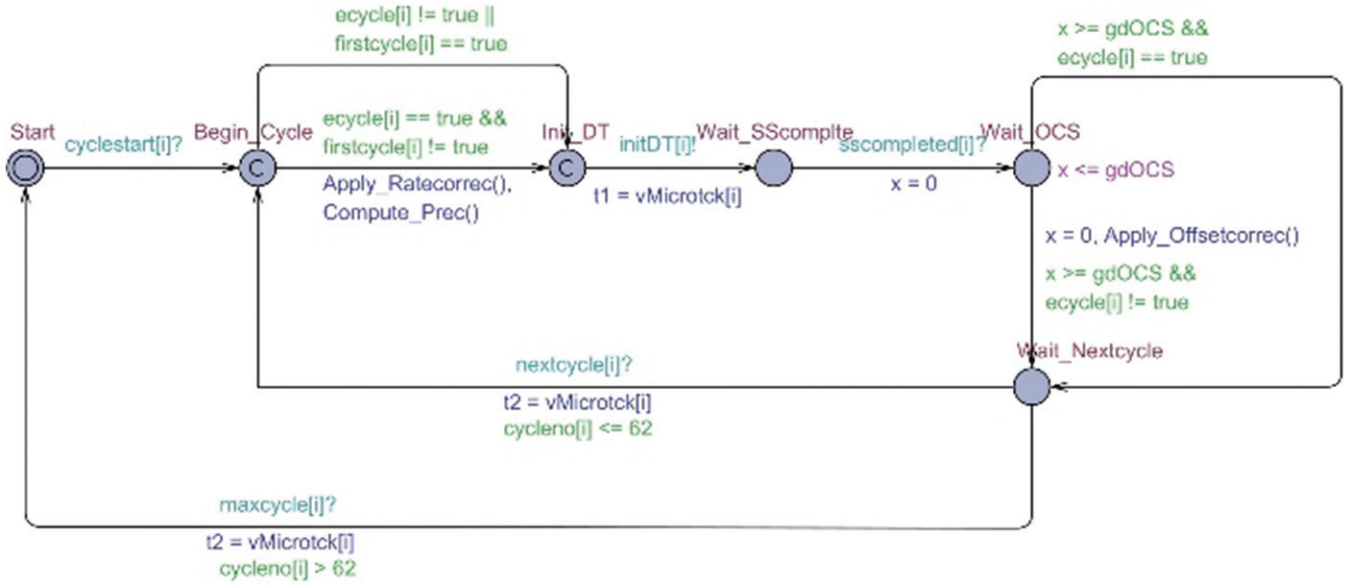

Figure 9. MTG automaton.

they receive the frame. This timestamp, STRP, is recorded in strp. The nodes that receive the frame execute the function computePTRP(). The node then issues genptrp[i]! to indicate that the computation of PTRP is completed and to synchronise with the CSP automaton for computing the deviation values.

The Node automaton is modified to consider the decoding delay and the propagation delay in Model 3. The receiver node on receiving the sendframe? moves to a wait location. This location has an invariant y<=offset + delta, and a transition from this location occurs when the value of y is between offset – delta and offset + delta. Thus, a delay is induced at the receiver while receiving the frame.

3.4 CSP Automaton

The *CSP* automaton depicted in Fig. 8 models the clock synchronisation process (Fig. 1). It initialises the deviation table when it receives the initDT? from the MTG automaton. The initialize_Devtable() function resets a part of the dev_table corresponding to the cycle. A timestamp is taken and is stored in tsap when the automaton receives the ssap? from MAC(i). It then computes the deviation when genptrp[i]? is received from the Node(i) as $svptrp[i] - tsap$. This value is stored in the appropriate part of dev_table based on the cycle being even or odd. The automaton waits for the calc_offset? signal

from MAC(i) and then calculates the offset correction value. It also calculates the rate correction value at the end of every odd cycle.

3.5 MTG Automaton

The main responsibility of the MTG process is to apply the offset and rate correction values. The automaton receives a cyclestart[i]? signal from MAC(i), indicating the start of the first communication cycle. The rate correction is done only at the beginning of an even cycle, except for the first cycle. The rate correction value computed during the previous cycle determines whether there are more or fewer microticks in the current cycle. The automaton then initiates initializing dev_table by issuing initDT! which will be received by CSP(i). It then waits for the completion of the static segment and receives sscompleted[i]? from MAC(i). A node has to wait for gdOffsetCorrectionStart (gdOCS) time to perform offset correction. Offset correction is performed by extending or shortening the NIT interval of the communication cycle. Figure 9 depicts the MTG automaton.

5. RESULTS AND DISCUSSION

The model is simulated and verified using the 4.1.25-5 version of UPPAAL on an Intel Xeon octa-core with 64-bit OS and 16 GB RAM. The values of vMicrotck from the Observer automaton were read during the simulation run and were used to compute vMacrotick and the precision.

The clock synchronisation of FlexRay modelled in this paper considers a bandwidth of 10Mbps. Since a cluster consisting of four nodes is considered, the static segment is slotted into four slots, one for each node. The values for the variables used in the model are computed assuming a clock period of 0.0125µs and a macrotick (MT) duration of 1µs. One sample of the clock pulse is considered as one microtick (µT), and one macrotick consists of 80µT. The maximum value of offset correction that is permitted, pOffsetCorrectionOut is computed as in equation 4. Where gOffsetCorrectionMax is the maximum necessary offset correction value (0.15µs) for the cluster globally, pdMicrotick is the node-specific duration of 1µT (0.0125µs), and cClockDeviationMax is the maximum clock frequency deviation. The cClockDeviationMax is a protocol constant, and the value is 1500 ppm. The computation of the maximum rate correction value, pRateCorrectionOut, is done as in Eqn. 5. The number of microticks per cycle, pMicroperCycle, is 1440. The values of the different variables computed for the simulation and verification are listed in Table 1. The range of values these variables can take for 10 Mbps is specified in the specification of FlexRay[3].

Table 1. Computed values of variables

Variable	Value
gdSampleClockPeriod	0.0125µs
gdMacrotick	1µs (80µT)
gdAPO	80µT
slottime[i]	0,320,640,960µT
gMacroPerCycle	18MT
cyclelength[i]	1440µT for all the nodes
vNIT[i]	160 for all the nodes
vSS[i]	4MT (320µT) for all the nodes
vStSeg[i]	960, 640, 320, 0
gdOCS	80µT
pDecodCorrec	18µT
pDelayComp	0 – 5µT
pOffsetCorrcOut	50µT
pRateCorrOut	5µT
pClustDriftDamp	0

Cycle No.	Node ID.	Offset Correction Value			Rate Correction Value			vNIT			Cycle Length		
		Model 1	Model 2	Model 3	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
0 – 1	0	0	2	0	0	2	1	160	162	160	1440	1442	1441
	1	0	-5	-11	0	-2	-3	160	155	149	1440	1438	1437
	2	0	0	-3	0	0	0	160	160	157	1440	1440	1440
	3	1	1	-2	0	0	0	161	161	158	1440	1440	1440
2 – 3	0	0	3	0	0	2	1	160	165	160	1440	1444	1442
	1	0	-8	-13	0	-2	-3	160	147	136	1440	1436	1434
	2	0	-1	-3	0	0	0	160	159	154	1440	1440	1440
	3	0	1	-1	0	0	0	161	162	157	1440	1440	1440
4 – 5	0	0	3	0	0	2	1	160	168	160	1440	1446	1443
	1	0	-8	-14	0	-2	-3	160	139	122	1440	1434	1431
	2	0	-1	-4	0	0	0	160	158	150	1440	1440	1440
	3	0	1	-2	-1	0	0	161	163	155	1439	1440	1440
6 – 7	0	0	2	0	0	1	1	160	170	160	1440	1447	1444
	1	0	-8	-13	0	-2	-3	160	131	109	1440	1432	1428
	2	0	-1	-3	0	-1	0	160	157	147	1440	1439	1440
	3	1	1	-1	-1	0	0	162	161	154	1438	1440	1440
8 - 9	0	0	3	0	0	1	0	160	173	160	1440	1448	1444
	1	0	-8	-12	0	-2	-3	160	123	97	1440	1430	1425
	2	0	0	-3	0	-1	0	160	157	144	1440	1438	1440
	3	1	1	-1	-1	0	1	163	165	153	1437	1440	1439

Figure 10. Simulation results of the three models.

$$pOffsetCorrectionOut = ceil\left(\frac{gOffsetCorrectionMax}{pdMicrotick*(1-cClockDeviationMax)}\right) \quad (4)$$

$$pRateCorrectionOut = ceil\left(\frac{pMicroperCycle*2*cClockDeviationMax}{(1-cClockDeviationMax)}\right) \quad (5)$$

The results obtained from performing simulation on the three models, synchronised clocks (Model 1), drifting clocks (Model 2), and the model with a delay at the receiver's end (Model 3), are in Fig. 10 and the observations are listed below.

- The duration of *vNIT* is reduced or extended based on the offset correction value computed during the odd cycle.
- The cycle duration is modified at the beginning of the even cycle considering the rate correction value computed during the previous double cycle.
- In all the models, the precision is between 0 - 2 MT (see Fig. 11), which is within the worst-case precision of a FlexRay network.

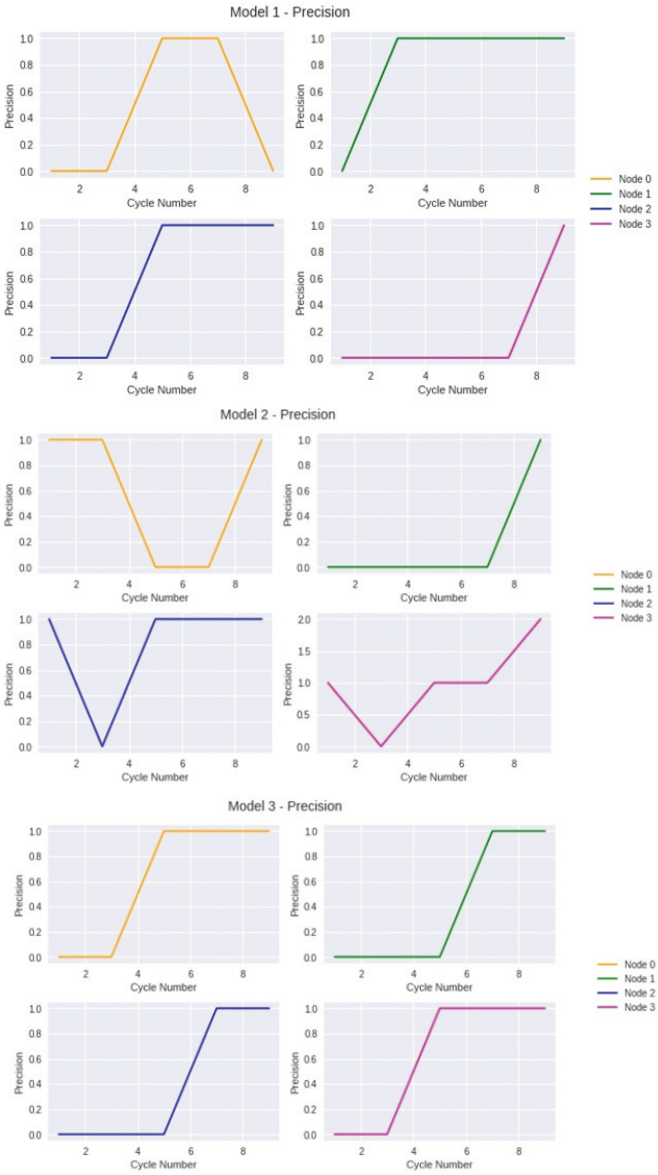

Figure 11. Precision observed for the models during simulation.

The precision values computed as the maximum deviation of the node's clock from the clocks of all other nodes in the cluster in terms of MT were verified in the UPPAAL model checker. The model checker checks whether the property is violated in any state in all the possible runs. If violated, the output produced by UPPAAL is 'property is not satisfied'; otherwise, it gives the output as 'property is satisfied'. The best and the worst-case precision as specified by the FlexRay specification is 0.15 µs (12 µT) and 6.675 µs (534 µT), respectively. The properties verified and the model checker's output are listed below. The precision values were formally verified to be within 0 – 6 MT for all the models using property 1 and property 2. The deadlock property, which states that in all possible runs of the system, no state may exist, in which no outbound transitions are triggered, is verified by property 3.

Properties Verified

Property 1: Precision is within the range 0-6MT.

Specification:

$$A[]\left(forall\left(i:nodes\right)prec[i]\geq 0\,and\ prec[i]\leq 6\right)$$

Result: Property is satisfied.

Property 2: Precision does not go beyond the range 0-6MT.

Specification:

$$E<>\left(exists\left(i:nodes\right)prec[i]<0\,or\ prec[i]>6\right)$$

Result: Property is not satisfied.

Property 3: System does not go into a deadlock state.

Specification:

$$A[]\ not\ deadlock$$

Result: Property is satisfied.

5. CONCLUSIONS

Formal modelling and verification of clock synchronisation of FlexRay make it reliable for in-vehicle communication by major automotive manufacturers. This paper developed a formal model of the FlexRay clock synchronisation algorithm using the UPPAAL model checker. The model developed consists of a network of six timed automata, Oscillator, Observer, MAC, Node, CSP and MTG. A cluster with four nodes was considered, and simulation and verification were done on three variants of the model, one with synchronised clocks, one with drifting clocks and one in which the decoding and propagation delays were modelled at the receiver node.

Simulation studies on all the models show that the clock correction is done by modifying the duration of the NIT and the cycle length. NIT is shortened or lengthened based on the offset correction value, and the cycle length is increased or decreased based on the rate correction value computed. Also, the precision is found to agree with the worst-case precision of 6 MT mentioned in the specification of FlexRay. It is also formally verified by the model checker that the precision is always within 0 MT and 6 MT. It is also verified that the system is free from deadlock. The models proposed in this work can also be applied to a dual-channel network by creating instances of the MAC automaton for the two channels and modifying the Node automaton to broadcast and receive frames on both channels.

The model can be extended for event-triggered communication by modifying the MAC automaton to incorporate dynamic segment and symbol window. Modelling the synchronisation of the nodes in multiple clusters, their behaviour and the modelling of drift between the clusters will be considered as future work.

REFERENCES

1. LIN Specification Package, Revision 2.0, LIN Consortium, 2003.
2. CAN Specification, Robert Bosch GmbH, Stuttgart, Germany, 1991.
3. FlexRay communications system protocol specification version 2.1., FlexRay Consortium, 2005.
4. MOST Specification Revision 2.3, MOST Cooperation, Karlsruhe, Germany, 2008.
5. Prasad, M.; Dey, R.K.; Sardar, A. & Goswami, G. Ethernet as an emerging trend in vehicle network technology—Part I. *Auto Tech. Rev.*, 2014, **3**(12), 18-23. doi: 10.1365/s40112-014-0805-5
6. International organisation for standardisation. Road vehicles–FlexRay communications system–Part 1: General information and use case definition. ISO I. 17458-1–2013 Jan.
7. Piao, J.H.; Wu, Y.J. & Xu, Y.N. A security framework for in-vehicle flexray bus network. *Int. J. Modeling Optimisation*, 2022, **12**(3). doi: 10.1007/978-3-030-66042-0_6
8. Pang, F.; Huang, M.; Mi, Z. & Zhang, H. A way to synchronize clocks with the FlexRay bus. *J. Physics: Conference Series*, IOP Publishing, 2022, **2187**(1), 012054. doi: 10.1088/1742-6596/2187/1/012054
9. Peng, L.; Jia, L. & Zefeng, Y. FlexRay bus data fault diagnosis based on Zynq. *J. Physics: Conference Series*, IOP Publishing, 2021, **1907**(1), 012029. doi: 10.1088/1742-6596/1907/1/012029
10. Wang, Y.; Chen, M.; Ma, J.; Zhang, J. & Fu, J. Predictive control of FlexRay vehicle-mounted network based on neural network. *J. Physics: Conference Series*, IOP Publishing, 2021, **1907**(1), 012062. doi: 10.1088/1742-6596/1907/1/012062
11. Xiong, W.; Ho, D.W. & Wen, S. A periodic iterative learning scheme for finite-iteration tracking of discrete networks based on FlexRay communication protocol. *Information Sciences*, 2021, **548**, 344-356. doi: 10.1016/j.ins.2020.10.017
12. Stojanović, Branka; Hofer-Schmitz, Katharina; Nahrgang, Kai; Vallant, Heribert & Derler, Christian. Formal modeling: A step forward to cyber secure connected car systems. Towards connected and autonomous vehicle highways. *Springer, Cham*, 2021, 131-167. doi: 10.1007/978-3-030-66042-0_6
13. Malinský, J. & Novák, J. Verification of flexray start-up mechanism by timed automata. *Metrology and Measurement Systems*, 2010, **17**(3), 461-480. doi: 10.2478/v10178-010-0039-z
14. Cranen, S. Model checking the FlexRay startup phase. *In* International Workshop on Formal Methods for Industrial Critical Systems, Springer, Berlin, Heidelberg, 27 August 2012, 131-145. doi: 10.1007/978-3-642-32469-7_9
15. Shimmi, Asokan & Santhosh Kumar, G. Modelling and verification of the FlexRay startup mechanism using UPPAAL model checker. *In* 2018 8th International Symposium on Embedded Computing and System Design (ISED), IEEE, 2018, **13**, 69-73. doi: 10.1109/ISED.2018.8704029
16. Guo, X.; Lin, H.H.; Yatake, K. & Aoki, T. An UPPAAL framework for model checking automotive systems with FlexRay protocol. *In* International Workshop on Formal Techniques for Safety-Critical Systems. Springer, Cham, 2013, October, 36-53. doi: 10.1007/978-3-319-05416-2_4
17. Guo, X.; Lin, H.H.; Aoki, T. & Chiba, Y. December. A reusable framework for modeling and verifying in-vehicle networking systems in the presence of CAN and FlexRay. *In* 2017 24th Asia-Pacific software engineering conference (APSEC), IEEE, 2017, 140-149. doi: 10.1109/APSEC.2017.20
18. Guo, X.; Aoki, T. & Lin, H.H. Model checking of in-vehicle networking systems with CAN and FlexRay. *J. Syst. Software*, 2020, **161**, 110461. doi: 10.1016/j.jss.2019.110461
19. Rodriguez-Navas, G.; Proenza, J. & Hansson, H. Using UPPAAL to model and verify a clock synchronisation protocol for the controller area network. *In* 2005 IEEE Conference on Emerging Technologies and Factory Automation, 2005, September, IEEE. **2**(8). doi: 10.1109/ETFA.2005.1612717
20. Rodriguez-Navas, G.; Proenza, J. & Hansson, H. An UPPAAL model for formal verification of master/slave clock synchronisation over the controller area network. *In* Proc. of the 6th IEEE International Workshop on Factory Communication Systems, Torino, Italy, IEEE Computer Society Press, Los Alamitos, 2006 June. doi: 10.1109/WFCS.2006.1704117
21. Leen, G. & Heffernan, D. Modeling and verification of a time-triggered networking protocol. *In* International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06), IEEE, 2006, April, 178-178. doi: 10.1109/ICNICONSMCL.2006.150

22. Leen, G. & Heffernan, D. Formally verifying aspects of time-triggered controller area network (Phases 1 \& 2a). Tech. report, PEI/CSRC report no. 20020603, main library, University of Limerick, 2002.
23. Steiner, W. & Dutertre, B. Automated formal verification of the TTEthernet synchronisation quality. *In* NASA Formal Methods Symposium. Springer, Berlin, Heidelberg, April 2011, 375-390.
doi: 10.1007/978-3-642-20398-5_27
24. Steiner, W. & Dutertre, B. SMT-Based formal verification of a TTEthernet synchronisation function. *In* International Workshop on Formal Methods for Industrial Critical Systems. Springer, Berlin, Heidelberg, September 2010, 148-163.
doi: 10.1007/978-3-642-15898-8_10
25. Hanzlik, A. A case study of clock synchronisation in FlexRay. Research Report 31/2006 Technische Universitat Wien, Institut fur Technische Informatik. 2006.
26. Armengaud, E. Experimental evaluation of the FlexRay clock synchronisation service. Proc. 20. ITG/GI/GMM Workshop Testmethoden und Zuverlssigkeit von Schaltungen und Systemen, 2008. pp. 85-90.
27. Behrmann, G., David, A. & Larsen, K.G. A tutorial on uppaal. Formal methods for the design of real-time systems, 2004, 200-236.
doi: 10.1007/978-3-540-30080-9_7
28. David, A.; Larsen, K.G.; Legay, A.; Mikučionis, M. & Poulsen, D.B. Uppaal SMC tutorial. *Int. J. Software Tools for Technology Transfer*, **17**(4), 2015, 397-415.
doi: 10.1007/s10009-014-0361-y
29. Huang, X.; Singh, A. & Smolka, S.A. Using integer clocks to verify clock-synchronisation protocols. *Innovations Syst. Software Engin.*, **7**(2), 2011,119-130.
doi: 10.1007/s11334-011-0152-5

CONTRIBUTORS

Ms Shimmi Asokan received her M. Tech degree in Software Engineering from Cochin University of Science & Technology (CUSAT), Kochi, Kerala. She is currently a Research Scholar in the Department of Computer Science at CUSAT. Her research interests include modelling and verification of software systems using formal methods.
In the present work, she is responsible for developing the formal model of the clock synchronisation algorithm of FlexRay, simulating the model and formally verifying the model.

Ms K.H. Kochaleema received her MTech in Software Engineering from Cochin University of Science and Technology, Kochi, Kerala. She is currently working as Scientist G at DRDO-NPOL, Kochi. She is heading the Quality and Reliability group of NPOL.
In the present work, she has contributed to developing the model and the specification for verification. She has also provided valuable ideas and feedback for completing the work.

Prof G. Santhosh Kumar obtained his Ph.D. and M. Tech in Computer and Information Science from Cochin University of Science & Technology (CUSAT), Kochi, Kerala. He is currently a Professor in the Department of Computer Science, CUSAT. His research interests include formal modelling, cyber-physical systems, computer vision, data science and NLP.
In the present work, he has provided guidance and constructive ideas for modelling and verifying the system. He has also offered valuable direction and complete support to carry out this study successfully.