Conformance Testing of the STP-ISS Protocol Implementation by Means of Temporal Logic

Nadezhda Chumakova, Valentin Olenev, Irina Lavrovskaya
Saint-Petersburg State University of Aerospace Instrumentation
Saint Petersburg, Russia
{nadezhda.chumakova, valentin.olenev, irina.lavrovskaya}@guap.ru

***Abstract*—The paper considers a process of conformance testing by means of computational tree temporal logic. Authors present a Kripke structure, which is STP-ISS-13 protocol implementation under test and a compilation example of temporal logic formula for a protocol requirement. Finally, the paper gives an example of error detection.**

I. INTRODUCTION

On-board equipment always needs a proper testing. Before integration on-board equipment into a on-board network, we need to be sure that this equipment works as expected [1]. Especially if we talk about equipment, that operates according to the newly developed communication protocol. The conformance testing is used to ensure that equipment operation complies with specification of the protocol and expected behavior.

Conformance testing is a kind of an audit which is used to check that the external behavior of a given implementation of a protocol is equivalent to its formal specification [2]. This definition can be expressed formally:

$$(I \; sat \; S) \leftrightarrow (\forall r \in S: I \; sat \; r) \quad (1)$$

where I is an implementation under test (IUT), $\mathrm{S} = \{\mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3, \ldots\}$ – specification, r – conformance requirement. "*I sat S*" means that IUT *I* satisfies specification *S*.

Implementation under test is a system that represents implementation of one or more protocols and which is needed to be tested [3].

Nowadays, we participate in development, implementation and evolution of the STP-ISS transport protocol for the communication in onboard SpaceWire networks. In this project, we developed two revisions of STP-ISS protocol, simulated and investigated them. The first revision of STP-ISS (STP-ISS-13) is much simpler and compact, but the second one (STP-ISS-14) is more powerful. Nevertheless, the backward compatibility for these revisions is provided. The detailed description of the protocol is provided in [4]. After development of protocol specifications, we got the task to implement a tester for the STP-ISS rev.1 equipment, which could tell the manufacturer, that STP-ISS device operates correctly. This tester should examine a device, implementing STP-ISS, with a set of different testing scenarios; each scenario should test a particular STP-ISS mechanism. Therefore, after testing, the manufacturer will get information on what STP-ISS mechanism failed and then operation history from the log-files could be analyzed for the details of an error [5].

II. KRIPKE STRUCTURE

In conformance testing, IUT of tested protocol can be represented by any system, which implements that protocol. In this paper, we will use model checking as conformance testing method. In this case, IUT must be a formal model. The most often used formal model in model checking is a Kripke structure.

Kripke structure is a kind of a nondeterministic finite state machine. It can be represented as $\mathrm{M} = (\mathrm{S}, \mathrm{S}_0, \mathrm{R}, \mathrm{AP}, \mathrm{L})$, where:

- S is a non-empty set of states;
- $S_0 \subseteq S$ is a non-empty set of initial states;
- $R \subseteq S \times S$ is a set of transition relations. Every state must have one or more transitions;
- AP is a finite set of atomic propositions;
- $\mathrm{L}: \mathrm{S} \rightarrow 2^{\mathrm{AP}}$ is a labeling function. This function assigns set of true atomic propositions with every state.

Atomic proposition is a statement, that can be true or false and whose structure we can ignore [6]. We chose various service primitives, packages and some processes (storing and deletion from buffers, timers expiration, data transmission, etc.) as atomic propositions for STP-ISS-13 protocol. Examples of atomic propositions are presented in Table I.

A state of a Kripke structure is a snapshot of all information about modeled system in every specific moment of time. By means of the L function, each state is associated with a set of atomic propositions, which are true in this state. Besides the states, the Kripke structure must contain transitions. A transition is a change of states of a real system. Every state must have one or more transitions. Graphical representation of the Kripke structure of the STP-ISS-13 protocol IUT is presented in Fig. 1.

This Kripke structure contains:

- 103 states: $\{s_0, s_1, \ldots, s_{102}\}$;
- One initial state s_0;
- Set of atomic propositions. Examples of atomic propositions are given in Table I;
- Labeling function *L*.

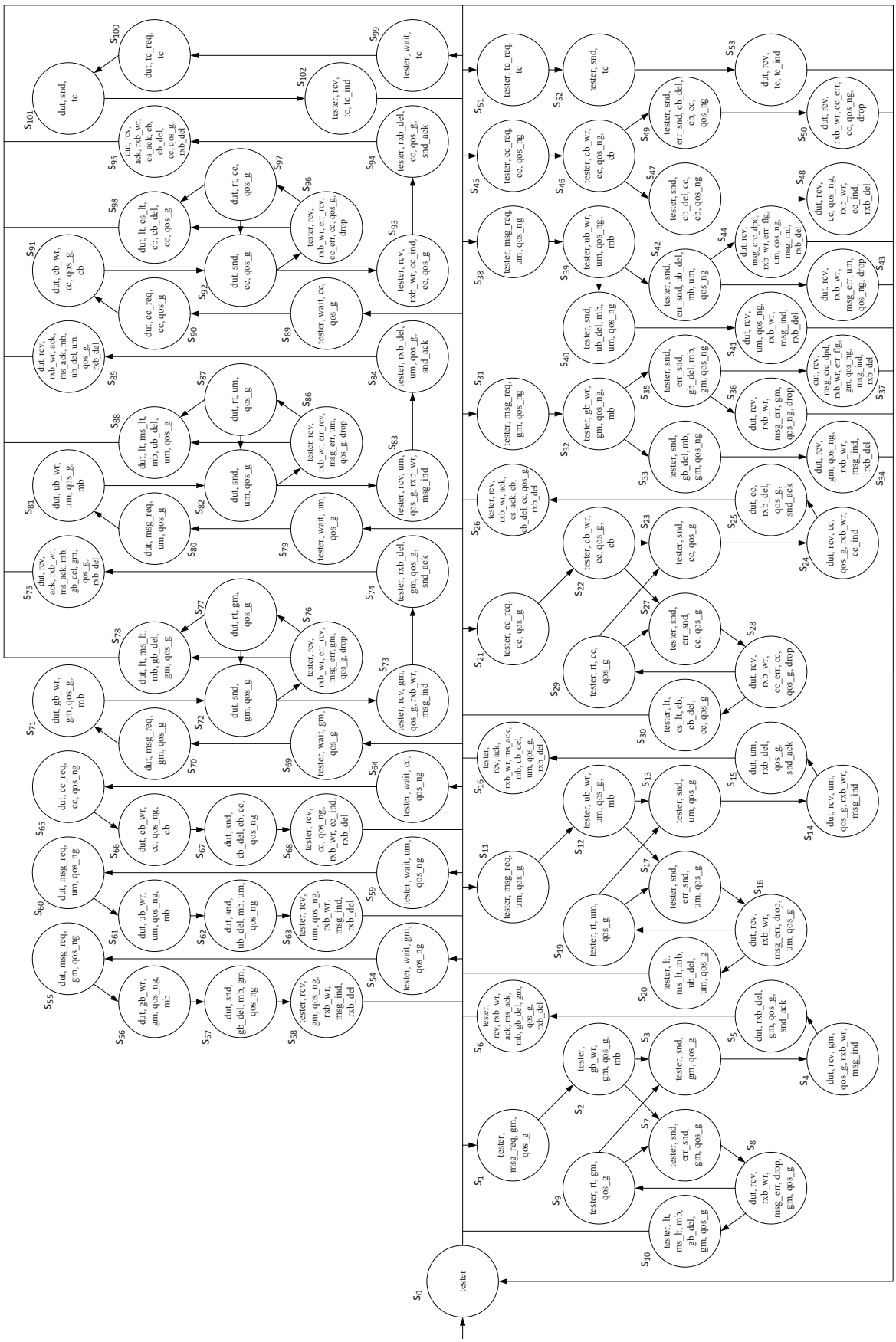

Fig. 1. The model of STP-ISS-13 protocol implementation

TABLE I. ATOMIC PROPOSITIONS

Atomic proposition	The meaning of atomic proposition
ack	Acknowledge
disc	Packet was discarded
iut	Implementation under test
err_snd	Packet was corrupted before transmission
gb_del	Packet was deleted from the general messages transmitter buffer
gm	General message
lt	Lifetime timer expiration
mb	Application layer was informed by STP_MessageBuffer.ind service primitive that the amount of free space in the general messages transmitter buffer has changed
ms_lt	STP-ISS-13 protocol indicated to the application layer by STP_MessageStatus.ind service primitive with "lifetime timer expiration" status about lifetime timer expiration
msg_err	Message is incorrect (other errors)
msg_ind	STP-ISS-13 protocol transferred the received message to the application layer by STP_Message.ind service primitive
msg_req	STP-ISS-13 protocol received message by STP_Message.req service primitive
qos_g	Guaranteed data delivery quality of service
rcv	Packet was received from a link
rt	Retry timer expiration
rxb_del	Packet was deleted from the receiver buffer
rxb_wr	Packet was stored in the receiver buffer
snd	Packet was sent to the link
snd_ack	Acknowledge was sent to the link
tester	Tester
um	Urgent message

States are observed in two nodes:

1) The first node is IUT of STP-ISS-13 protocol. States, which are observed in IUT, are marked by atomic proposition *iut* (in those states atomic proposition *iut* is true). For example:

- In state s_4, IUT received a general guaranteed message, stored this message in a receiver buffer and transferred that message to the application layer. Function L in this state has the following meaning: $L(s_4) = \{iut, rcv, gm, qos\_g, rxb\_wr, msg\_ind\}$.
- In state s_5, IUT sends an acknowledgement for packet receipt and deletes that packet from the buffer. Function L in this state has the following meaning: $L(s_5) = \{iut, rxb\_del, gm, qos\_g, snd\_ack\}$.
- In state s_8, IUT receives an incorrect general guaranteed message and discard that message. Function L in this state has the following meaning: $L(s_8) = \{iut, rcv, rxb\_wr, msg\_err, disc, gm, qos\_g\}$.

2) The Second node is a tester, which also implements STP-ISS-13 protocol. States, which are observed in the tester are marked by atomic proposition *tester*. For example:

- In state s_9, the retry timer in the tester expires for a guaranteed general message. Function L in this state has the following meaning: $L(s_9) = \{tester, rt, gm, qos\_g\}$.
- In state s_{10}, the lifetime timer in the tester expires for a guaranteed general message. This message is deleted from the general messages transmitter buffer. Application layer is informed about lifetime timer expiration and about the change of the amount of free space in the general messages transmitter buffer. Function L in this state has the following meaning: $L(s_{10}) = \{tester, lt, ms\_lt, mb, gb\_del, gm, qos\_g\}$.
- In state s_{11}, the application layer has an urgent guaranteed message, which is intended for transmission. Function L in this state has the following meaning: $L(s_{11}) = \{tester, msg\_req, um, qos\_g\}$.

The presence of a tester in this Kripke structure is due to the fact that the second node is required for modeling of the protocol operation. STP-ISS-13 protocol implementation in the tester is a reference implementation. Only IUT shall be a subject of testing.

III. TEMPORAL LOGIC

We created a set of testing scenarios for STP-ISS-13 protocol. Each scenario is intended for testing of a specific mechanism of STP-ISS-13. This set of scenarios was developed by means of temporal logic.

Temporal logic is a logic in which the statements take into account the time aspect [7]. Values of the formulas of temporal logic depend on the time at which the values of these formulas are calculated [6].

There are different temporal logics:

- ITL – Interval Temporal Logic [8];
- PSL – Property Specification Language [9];
- LTL – Linear Time Logic;
- CTL – Computation Tree Logic[6];
- Etc.

In order to implement a set of testing scenarios we chose the computation tree logic (CTL). CTL formulas are state formulas [10]. They describe computation trees properties. Computation tree is formed by sweeping the Kripke structure from the initial state [11].

CTL formulas can include:

- Atomic propositions (p, q, ...);
- Logical operators ($\vee$, $\wedge$, $\neg$, ...);
- Path quantifiers (A, E);
- Temporal operators (U, R, X, F, G);
- Parentheses.

CTL have two quantifiers for branching structure description:

- E (Exist). This quantifier means "in some computation path";
- A (All). This quantifier means "in all computation paths" [12].

Path quantifiers are used in every state to indicate that all or some paths, that outgoing from this state, possess the prescribed property.

Temporal operators describe the properties of the path in computation tree [11]. CTL have following temporal operators.

1) U (Until). U is a binary operator of conditional expectation.

Example of use:

$$p\ U\ q \tag{2}$$

In this example, atomic proposition p will be true until q is true. The diagram for this example is given in Fig. 2.

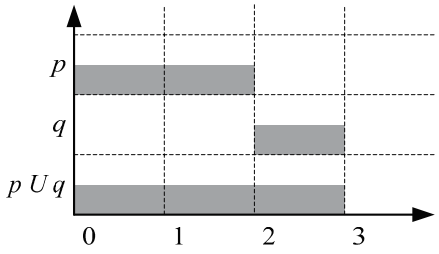

Fig. 2. Operator U

2) R (Release). R is a binary operator of unlocking.

Example of use:

$$p\ R\ q \tag{3}$$

In this example, atomic proposition q will be true until p is true. In contrast with the operator U, q becomes false only in the next state after the state in which p is true. The diagram for this example is given in Fig. 3.

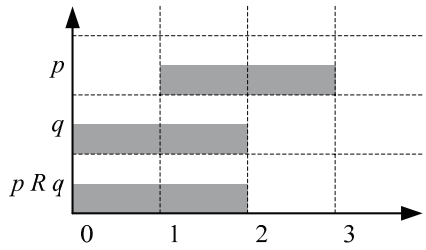

Fig. 3. Operator R

3) X (Next Time). X is a unary operator of time shift.

Example of use:

$$Xq \tag{4}$$

In this example, atomic proposition q will be true in the state, immediately following the considered state. Diagram for this example is presented in Fig. 4.

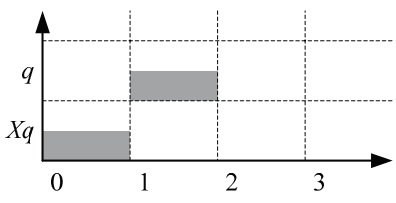

Fig. 4. Operator X

4) F (Future). F is a unary operator of an event.

Example of use:

$$Fq \tag{5}$$

In this example, atomic proposition q will be true at least in one state in future. Diagram for this example is presented in Fig. 5.

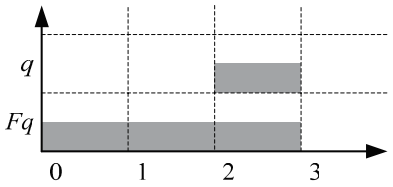

Fig. 5. Operator F

5) G (Globally). G is a unary operator of invariance.

Example of use:

$$Gq \tag{6}$$

In this example, atomic proposition q will be true in all future states. Diagram for this example is presented in Fig. 6.

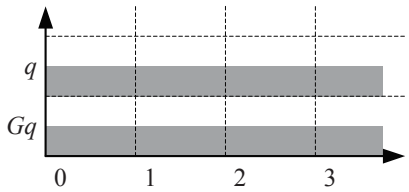

Fig. 6. Operator G

CTL formulas can include different combinations of path quantifiers and temporal operators, but each operator must be preceded by path quantifier [13].

IV. Model Checking

Let us consider an example of CTL formula drafting.

Step 1. Guaranteed general message receiving mechanism was chosen as the tested mechanism.

Step 2. According to the selected mechanism, we choose the following requirements from the STP-ISS-13 protocol specification [14]:

- An acknowledgement shall be sent if the following conditions are fulfilled simultaneously:
  - No CRC errors;
  - Correct data packet length.
  - Packet is transmitted with Guaranteed quality of service, i.e. there is an “Acknowledgement request” flag set to 1.

Step 3. According to the selected requirements, we choose the following atomic propositions from Table. I:

- *tester* – Tester. This atomic proposition is true in states, which are observed in the tester.

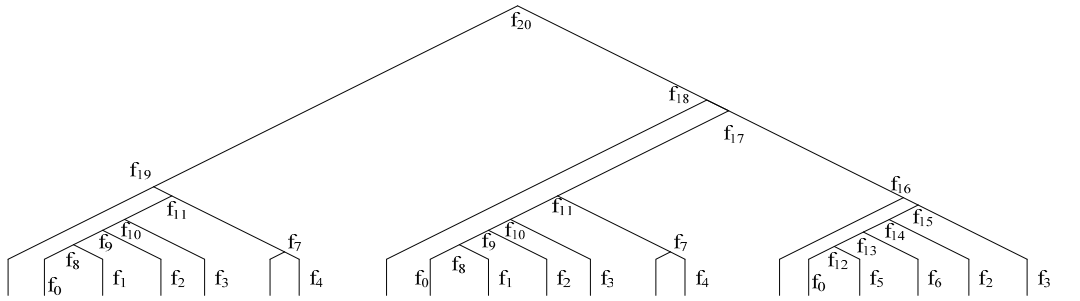

Fig. 7. Divided formula into subformulas

- *snd* – Packet was sent to the link. This atomic proposition means that packet was sent to the link.
- *rcv* – Packet was received from the link. This atomic proposition means that packet was received from the link.
- *gm* – General message.
- *qos_g* – Guaranteed delivery quality of service.
- *err_snd* – Packet was corrupted before transmission. A packet can be corrupted in order to test non-nominal cases. If a packet was not intentionally corrupted before transition, then err_snd atomic proposition will be false.
- *ack* – Acknowledge.

Construct the formula using selected mechanism, requirements and atomic propositions.

- In order to check that IUT implements the mechanism of guaranteed general message receiving correctly, the tester must send a guaranteed general message to the IUT, i.e., the model must include a state, in which atomic propositions *tester*, *snd*, *gm* and *qos_g* are true at the same time. Atomic proposition *err_snd* must be false in this state, because the packet was not corrupted before sending. This can be expressed by the formula:

$$\varphi_1 = tester \wedge snd \wedge gm \wedge qos\_g \wedge \neg err\_snd \quad (7)$$

- In future, after sending a correct guaranteed general message, the tester must receive an acknowledgement. This means that after the state, in which atomic propositions *tester*, *snd*, *gm*, *qos_g* are true and *err_snd* is false, must be a state, in which *tester*, *rcv*, *ack*, *gm*, and *qos_g* are true. This statement can be expressed by the formula:

$$\varphi_1 = (tester \wedge snd \wedge gm \wedge qos\_g \wedge \neg err\_snd) \Longrightarrow AF(tester \wedge rcv \wedge ack \wedge gm \wedge qos\_g) \quad (8)$$

- Formula φ_1 must be true in all states and all paths. Therefore, we add an appropriate temporal operator and quantifier:

$$\varphi_1 = AG((tester \wedge snd \wedge gm \wedge qos\_g \wedge \neg err\_snd) \Longrightarrow AF(tester \wedge rcv \wedge ack \wedge gm \wedge qos\_g)) \quad (9)$$

- Formula φ_1 have an implication logical operator ($\Longrightarrow$). Implication as function is true if and only if the left part of the function is true, and right part is false. In this example, formula φ_1 will be false if subformula tester ∧ snd ∧ gm ∧ qos_g ∧ ¬err_snd is true and subformula AF(tester ∧ rcv ∧ ack ∧ gm ∧ qos_g) is false. Consequently, if the model has no state, in which a correct guaranteed general message is received, the formula φ_1 will be true too. Therefore, we add an additional condition: the model must have at least one state, in which atomic propositions *tester*, *snd*, *gm* and *qos_g* are true and err_snd is false. This condition can be expressed by the formula:

$$\varphi_2 = EF(tester \wedge snd \wedge gm \wedge qos\_g \wedge \neg err\_snd) \quad (10)$$

- Formulas φ_1 and φ_2 can be united, because this formulas must be true at the same time. As a result, we obtain the following formula:

$$\varphi_1 = EF(tester \wedge snd \wedge gm \wedge qos\_g \wedge \neg err\_snd) \wedge AG((tester \wedge snd \wedge gm \wedge qos\_g \wedge \neg err\_snd) \Longrightarrow AF(tester \wedge rcv \wedge ack \wedge gm \wedge qos\_g)) \quad (11)$$

According STP-ISS-13 protocol specification, we developed a set of CTL formulas, which allow us to test all of STP-ISS-13 basic mechanisms and non-nominal cases. Here are examples of the formulas:

- For testing guaranteed control command receiving mechanism we used the following formula:

$$\varphi_3 = EF(tester \wedge snd \wedge cc \wedge qos\_g \wedge \neg err\_snd) \wedge AG((tester \wedge snd \wedge cc \wedge qos\_g \wedge \neg err\_snd) \Longrightarrow AF(tester \wedge rcv \wedge ack \wedge cc \wedge qos\_g)) \quad (12)$$

- For testing guaranteed urgent message receiving mechanism we used the following formula:

$$\varphi_8 = EF(tester \wedge wait \wedge um \wedge qos\_g) \wedge AG((tester \wedge wait \wedge um \wedge qos\_g) \Longrightarrow EF(tester \wedge rcv \wedge um \wedge qos\_g \wedge \neg msg\_err \wedge \neg msg\_crc\_dpd \wedge \neg err\_rcv \wedge AX(tester \wedge snd\_ack \wedge um \wedge qos\_g \wedge AX(iut \wedge qos\_g \wedge rcv \wedge ack \wedge um \wedge ub\_del)) \quad (13)$$

- For testing non-guaranteed incorrect control command receiving mechanism we used the following formula:

$$\varphi_{18} = EF(tester \wedge snd \wedge cc \wedge qos\_ng \wedge err\_snd) \wedge AG((tester \wedge snd \wedge cc \wedge qos\_g \wedge err\_snd) \Longrightarrow AX(iut \wedge rcv \wedge cc \wedge qos\_ng \wedge cc\_err \wedge disc)) \quad (14)$$

For formal verification of the model we chose the model checking method [15]. Depending on the formal models used, the model checking algorithms can be different. For the case, when the IUT is represented by a Kripke structure and requirements are represented by CTL formulas, we can use labeling algorithm.

Main idea of labeling algorithm is that for any subformula of the CTL formula we need to know in which states of the Kripke structure this subformula is true. CTL formula is true in the model if that formula is true in the initial state. In another case, the formula is false.

The labeling algorithm for CTL formula φ and Kripke structure M includes the following steps:

1) Formula φ is divided into subformulas, starting with the simplest.
2) For each subformula f:
   a. Set $\mathrm{Sat_f}$ of Kripke structure states, where subformula is true, is computed.
   b. Subformula is written as atomic proposition.
   c. All states from set $\mathrm{Sat_f}$ are marked by this atomic proposition.
3) If initial state of Kripke structure M is marked by atomic proposition $\mathrm{p_\varphi}$, then formula φ is true in M [6].

Consider an example of labeling algorithm for formula φ_1 and Kripke structure, presented in Fig. 1.

1) Divide formula φ_1 into 20 subformulas. Formula division is presented in Fig. 7.
2) For each subformula f:
   a. Compute a set $\mathrm{Sat_f}$. Results are presented in Table II.
   b. Each subformula is written as atomic proposition (subformula f_0 is written as atomic proposition f_0, subformula f_1 is written as atomic proposition f_1, etc.).
   c. All states from the set $\mathrm{Sat_{fi}}$ are marked by atomic proposition f_i. Part of the Kripke structure with new marks is presented in Fig. 8.

TABLE II. SET SAT

Subformula	Set Sat_f
f_0 = tester	Sat_{f0} = {s_0,…, s_3, s_6, s_7, s_9,…, s_{13}, s_{16}, s_{17}, s_{19},…, s_{23}, s_{26}, s_{27}, s_{29},…, s_{33}, s_{35}, s_{38}, s_{39}, s_{40}, s_{42}, s_{45}, s_{46}, s_{47}, s_{49}, s_{51}, s_{52}, s_{54}, s_{58}, s_{59}, s_{63}, s_{64}, s_{68}, s_{69}, s_{73}, s_{74}, s_{76}, s_{79}, s_{83}, s_{84}, s_{86}, s_{89}, s_{93}, s_{94}, s_{96}, s_{99}, s_{102}}
f_1 = snd	Sat_{f1} = {s_3, s_7, s_{13}, s_{17}, s_{23}, s_{27}, s_{33}, s_{35}, s_{40}, s_{42}, s_{47}, s_{49}, s_{52}, s_{57}, s_{62}, s_{67}, s_{72}, s_{82}, s_{92}, s_{101}}
f_2 = gm	Sat_{f2} = {s_1,…, s_{10}, s_{31},…, s_{37}, s_{54},…, s_{58}, s_{69},…,s_{78}}
f_3 = qos_g	Sat_{f3} = {s_1,…, s_{30}, s_{69},…, s_{98}}
f_4 = err_snd	Sat_{f4} = {s_7, s_{17}, s_{27}, s_{35}, s_{42}, s_{49}}
f_5 = rcv	Sat_{f5} = {s_4, s_6, s_8, s_{14}, s_{16}, s_{18}, s_{24}, s_{26}, s_{28}, s_{34}, s_{36}, s_{37}, s_{41}, s_{43}, s_{44}, s_{48}, s_{50}, s_{53}, s_{58}, s_{63}, s_{68}, s_{73}, s_{75}, s_{76}, s_{83}, s_{85}, s_{86}, s_{93}, s_{95}, s_{96}, s_{102}}
f_6 = ack	Sat_{f6} = {s_5, s_6, s_{15}, s_{16}, s_{25}, s_{26}, s_{74}, s_{75}, s_{84}, s_{85}, s_{94}, s_{95}}
$f_7 = \neg f_4$	Sat_{f7} = {s_0,…, s_6, s_8,…, s_{16}, s_{18},…, s_{26}, s_{28},…, s_{34}, s_{36},…, s_{41}, s_{43},…, s_{48}, s_{50},…, s_{102}}
$f_8 = f_0 \wedge f_1$	Sat_{f8} = {s_3, s_7, s_{13}, s_{17}, s_{23}, s_{27}, s_{33}, s_{35}, s_{40}, s_{42}, s_{47}, s_{49}, s_{52}}
$f_9 = f_8 \wedge f_2$	Sat_{f9} = {s_3, s_7, s_{33}, s_{35}}
$f_{10} = f_9 \wedge f_3$	Sat_{f10} = {s_3, s_7}
$f_{11} = f_{10} \wedge f_7$	Sat_{f11} = {s_3}
$f_{12} = f_0 \wedge f_5$	Sat_{f12} = {s_6, s_{16}, s_{26}, s_{58}, s_{63}, s_{68}, s_{73}, s_{76}, s_{83}, s_{86}, s_{93}, s_{96}, s_{102}}
$f_{13} = f_{12} \wedge f_6$	Sat_{f13} = {s_6, s_{16}, s_{26}}
$f_{14} = f_{13} \wedge f_2$	Sat_{f14} = {s_6}
$f_{15} = f_{14} \wedge f_3$	Sat_{f15} = {s_6}
$f_{16} = AF(f_{15})$	Sat_{f16} = {s_3, s_4, s_5}
$f_{17} = f_{11} => f_{16}$	Sat_{f17} = {s_0,…, s_{102}}
$f_{18} = AG(f_{17})$	Sat_{f18} = {s_0,…, s_{102}}
$f_{19} = EF(f_{11})$	Sat_{f19} = {s_0,…, s_{102}}
$f_{20} = f_{19} \wedge f_{18}$	Sat_{f20} = {s_0,…, s_{102}}

3) In the given example, the atomic proposition f_{20} is the atomic proposition $p_{\varphi 1}$. The initial state of the Kripke structure was marked by the example atomic proposition f_{20}. Therefore, the formula φ_1 is true in this Kripke structure. We can conclude that Kripke structure satisfies the requirement, which is represented by the formula φ_1.

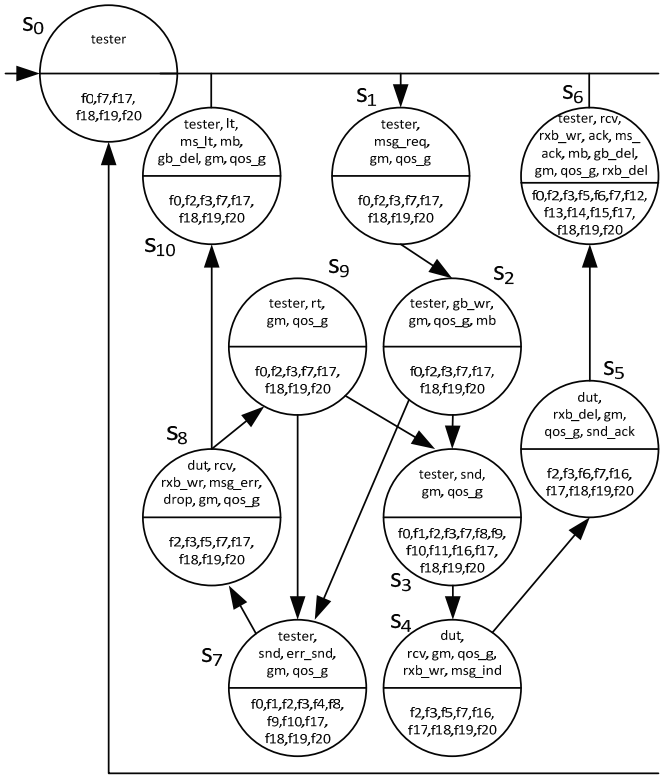

Fig. 8. Kripke structure with new marks

Let us consider an example of how we can identify errors in IUT by labeling algorithm. For this purpose, we added a transition from sate s_4 to state s_9 in Kripke structure, presented in Fig. 1. This transition means that the structure has a path, in which after receiving correct guaranteed general message IUT does not send an acknowledgement. A part of this structure presented in Fig. 9.

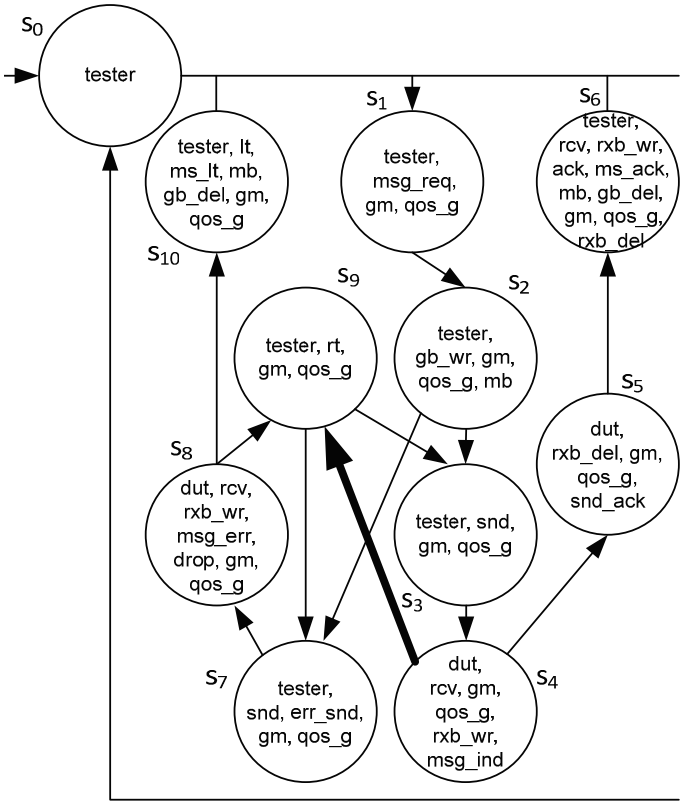

Fig. 9. Incorrect Kripke structure

We also use the labeling algorithm for formula $\varphi_1$checking on this model. The first step of the algorithm will be similar to the previous example. Let us consider the second and the third steps of the labeling algorithm:

2) For each subformula f:

   a. Compute a set $\mathrm{Sat_f}$. Meanings of sets Sat_{f0}, Sat_{f1}, ..., $Satf_{15}$ are matches the values in Table II. The rest of meanings is presented in Table III.

TABLE III. SET SAT FOR KRIPKE STRUCTURE WITH ERRORS

Subformula	Set Sat_f
$f_{16} = AF(f_{15})$	$Sat_{f16} = \{s_5\}$
$f_{17} = f_{11} => f_{16}$	$Sat_{f17} = \{\}$
$f_{18} = AG(f_{17})$	$Sat_{f18} = \{\}$
$f_{19} = EF(f_{11})$	$Sat_{f19} = \{s_0, \ldots, s_{102}\}$
$f_{20} = f_{19} \wedge f_{18}$	$Sat_{f20} = \{\}$

   b. Each subformula is written as atomic proposition (subformula f_0 is written as atomic proposition f_0, subformula f_1 is written as atomic proposition f_1, etc.).
   c. All states from the set Sat_{fi} are marked by the atomic proposition f_i. Part of the Kripke structure with new marks is presented in Fig. 10.

3) In the given example, the atomic proposition f_{20} is the atomic proposition $p_{\varphi 1}$. The initial state of the Kripke structure was not marked by the example atomic proposition f_{20}. Therefore, the formula φ_1 is false in this Kripke structure. We can conclude that the Kripke structure does not satisfy the requirement, which is represented by formula φ_1.

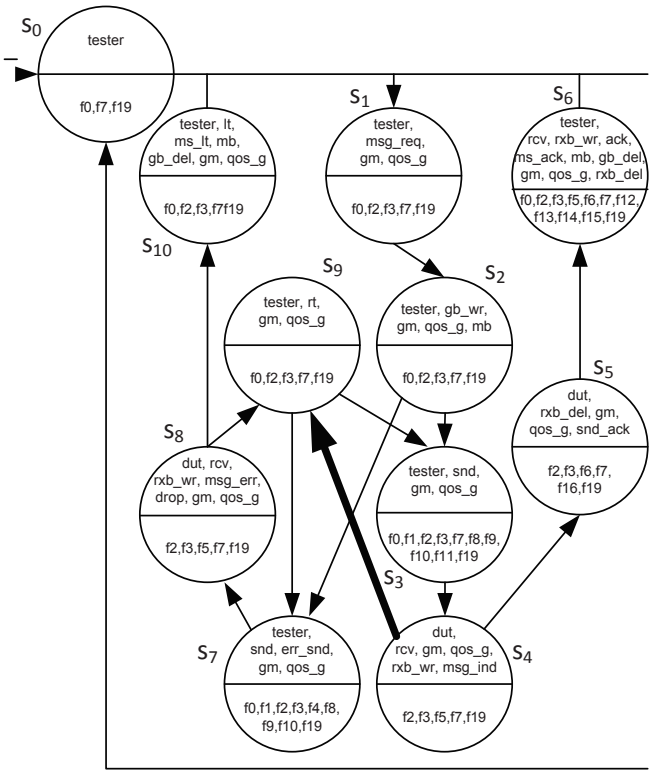

Fig. 10. Incorrect Kripke structure with new marks

V. CONCLUSION

The paper considers a process of conformance testing by means of computational tree temporal logic. Firstly, we created a Kripke structure, which represents a formal model of STP-ISS-13 protocol implementation. This model allows us to describe all possible states of IUT and tester. Also, we developed a set of CTL formulas, which is a formal description of requirements of the STP-ISS-13 protocol implementation. A set of CTL formulas allows us to test all of STP-ISS-13 basic mechanisms and non-nominal cases.

Moreover, in this paper we described the process of Kripke structure formal verification. We choose the labeling algorithm, which is one of a model checking algorithms, as a formal verification method. In this case, formal verification is a conformance testing method, where protocol implementation under test is a Kripke structure, and specification is a set of CTL formulas.

Described conformance testing method is suitable only for testing formal models of STP-ISS-13 protocol implementations. In order to be able to test software and hardware implementations of this protocol, we developed a Software-to-Hardware Tester. This tester is based on a Kripke structure and CTL formulas, which were developed in this paper.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation under the contract № 8.4048.2017/4.6.

REFERENCES

[1] P. Marwedel, *Embedded System Design*, Springer, 2006. 241 p.

[2] G. Holzmann, *Design and validation of computer protocols*. N.J: Murray Hill, 1991. 5433 p.

[3] ISO/IEC 9646-1:1994. Information technology. Open Systems Interconnection. Conformance Testing methodology and framework. Part 1: General concepts.

[4] V. Olenev, I. Lavrovskaya, Yu. Sheynin, I. Korobkov, E. Suvorova, E. Podgornova, D. Dymov, S. Kochura, "STP-ISS Transport Protocol for SpaceWire On-Board Networks: Development and Evolution", *International Journal of Embedded and Real-Time Communication Systems*, №5(4). Tampere: IGI Global, 2014. – pp. 45-76.

[5] V. Olenev, I. Lavrovskaya, N. Chumakova, D. Dymov, V. Shkolniy, S. Kochura "Software-to-hardware tester for SpaceWire oriented transport protocols", *proceedings of 2016 International SpaceWire Conference*. Yokohama, Japan. 2016.

[6] Y. Karpov, *MODEL CHECKING. Verification of parallel and distributed software systems*, St. P, 2016. 560 p.

[7] R. Caferra, *Logic for Computer Science and Artificial Intelligence*, N. J.: Willey & Sons, Inc; L.: ISTE Ltd, 2011. 537 p.

[8] K. D'Emanuele, G. Pace, "Runtime Validation Using Interval Temporal Logic, Computer Science Annual Workshop 2006", Department of Computer Science and AI, University of Malta, 2006.

[9] H. Foster, A. Krolnik, D. Lacey, *Assertion-Based Design*. Springer, 2005. 390 p.

[10] M. Browne, E. Clarke, O. Grumberg, "Characterizing finite Kripke structures in propositional temporal logic", *Theoretical Computer Science 59*, 1988, pp. 115-131.

[11] E. Clarke, O. Grumberg, D. Peled, *Model Checking*. MIT Press, 1999.

[12] F. Mogavero, A. Murano, "Branching-Time Temporal Logics with Minimal Model Quantifiers", *proceedings of Developments in Language Theory: 13th International Conference*. Stuttgart, Germany. 2009, pp. 396-409.

[13] S. Demri, P. Gastin, "Specification and Verification using Temporal Logic", World Scientific Review, 2010.

[14] Y. Sheynin, V. Olenev, I. Lavrovskaya, I. Korobkov, D. Dymov, "STP-ISS Transport Protocol for Spacecraft On-board Networks", *proceedings of 6th International Conference SpaceWire 2014 Program*, Athens, Greece, 2014. pp. 26-31.

[15] E. Clarke, *The Birth of Model Checking*. Springer, 2008.