2019 2nd International Conference on Mechanical, Electronic and Engineering Technology (MEET 2019)

Analysis of SpaceWire Link Initialization Time based on Timed Automata

Ping Luo,DaiWu Chen,Dong Xie,Yan Zhang
Department of Information Science and Engineering
Hunan University of Humanities,Science and Technology
Hunan, China
Email: luoping0718@163.com

***Abstract*—SpaceWire provides a full-duplex, bidirectional, point-to-point, high speed data link for on-board network. It is significant to make sure of the link connection successfully and reliably. Based on timed automata, the paper presents the modeling of the modules of Controller, Timer, Transmitter and Receiver implemented in SpaceWire link interface, and the simulation of the link initialization. The tool we adopted is Uppaal, a timed automata based model checker for real-time system. In Uppaal, we analyzed some time properties during the link initialization process. It turned out that there were details of the implementation inconsistent with the SpaceWire protocol standard. The state machine of the Controller for the link interface didn't wait enough time in some states as expected. Finally, we found out the reason caused the problem and revised the implementation.**

***Key Words*—SpaceWire;Timed Automata;Model Checking**

I. Introduction

SpaceWire[1] is a serial link standard applied to onboard network, publicly proposed by ESA(short for European Space Agency)in 2003[2]. It provides a full-duplex, bidirectional, point-to-point, high speed data link.The data signalling rate at which a SpaceWire link shall operate ranges from 2Mb/s to 400Mb/s. It has been widely applied to many aerospace projects by ESA, NASA, JAXA, and many other organizations. Considering the severe environment and special demand of high reliability, the study is significant on the test and verification of SpaceWire design and implementation.

Model Checking[3][4], put forward by Edmund Melson Clarke etc., is one of the most effective method of formal verification which uses formal methods of mathematics to prove or disprove the correctness of target systems with respect to a certain formal property. Extracting a model of a finite-state system, model checking shall exhaustively and automatically check whether the model satisfies a given specification. The models extracted can be finite state machine, timed automata, petri nets and Hoare logic and so on.Many tools are developed for model checking, like SPIN[5], SMV [6], Uppaal [7], etc. This paper chooses Uppaal as model checker.

Uppaal[8], available at www.uppaal.com, is a toolbox for verification of real-time systems modeled as networks of timed automata.It has been applied successfully in case studies ranging from communication protocols to multimedia applications. In Uppaal, a timed automaton is actually a state machine extended with clock variables. A system is modeled as a network of several such timed automata in parallel. Like the model, the properties to be checked are formally expressed in a simplified version of TCTL(timed computation tree logic). The definition of timed automata will be introduced in section II in this paper. In section III, we will establish the timed automata of the module of Timer of the SpaceWire link interface and test its function. And then we will model and check the link initialization with a brief state machine of SpaceWire link interface. At last in section IV comes conclusion.

II. Timed Automaton

Timed Automaton[9][10],is an extended finite automaton with clock variables. Before giving its definition of the syntax and semantics, we notate a set of clock variables as *C* and the set of formulas as *F(C)* over atomic constraints of the form $x \bowtie n$ or $x - y \bowtie n$, where $x, y \in C$, $n \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, >, \geq\}$. For all x $\in$ *C*, we define a clock valuation, a non-negative real-valued function $\nu : C \to \mathbb{R}_{\geq 0}$. Let $\mathbb{R}^C$ denote the set of all clock valuations. Let $\nu_0(x) = 0, \forall x \in C$.

(Definition.1) Timed Automaton (TA)
A timed automaton $\mathcal{A}$ is a six-tuple $< L, l_0, C, A, E, I >$, where

- L is a set of locations,
- $l_0 \in L$ is the initial location,
- C is the set of clock variables,
- A is a set of actions and reactions,
- $E \subseteq L \times A \times F(C) \times 2^C \times L$ is a finite set of edges between locations, which contains a source location, a guard, an action, a set of clocks to be reset, and a target location.
- I is a function: $L \to F(C)$, giving a location an invariant of clock condition.

(Definition.2) Semantics of Timed Automaton
A timed automaton $\mathcal{A} = < L, l_0, C, A, E, I >$ can be semantically defined as a labeled transition system $< S, s_0, \to >$.

- $S \subseteq L \times \mathbb{R}^C$ is the set of states,
- $s_0 = (l_0, \nu_0)$ is the initial state,

- $\rightarrow \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$ signifies the transition relations defined by:
  - $(l, \nu) \xrightarrow{d} (l, \nu + d)$, for $\forall d' \in \mathbb{R}_{\geq 0}$, if $0 \leq d' \leq d, \nu + d'$ satisfies *I(l)*
  - $(l, \nu) \xrightarrow{a} (l', \nu')$, if $\exists e = (l, a, g, r, l') \in E,\ s.t.\ \nu \in g,\ \nu' = \nu_{r \rightarrow 0}$, and ν' satisfies *I(l')*

A system generally runs multiple interactive processes. It comes natural to introduce the conception of a network of Timed Automata, comprised of n timed automata sharing a common set of clocks and one of actions. Let $\mathbb{A}_i = (L_i, l_i^0, C, A, E_i, I_i), 1 \leq i \leq n$, be a network of n timed automata. The location vector can be written as $\overline{l} = (l_1, l_2, ..., l_n)$. The initial location vector $\overline{l^0} = (l_1^0, l_2^0, ..., l_n^0)$.

III. Model Checking of SpaceWire Link Interface

This paper discusses the transactions at the exchange level, which is responsible for making a connection across a link and for managing the flow of data across the link. We shall focus on the link initialization before making connection successfully. The link interface designed mainly consists of modules such as Controller, Timer, Transmitter, Receiver, CreditCounter, BaudrateCounter, Recovery, and ErrorNotification. Considering the complexity of modeling and what we care about here is the time property of the Controller during the link initialization, a reduction of link interface is presented including reduced modules of Controller, Timer, Transmitter and Receiver. There are data and control characters across the link, which are separated into two types: link-characters (L-Char) and normal-characters (N-Char). L-Chars are only used in the exchange level, including the flow control token (FCT, 4-bit) character and escape (ESC, 4-bit) character. A FCT, which is sent out by a link interface, indicates that there is space for eight more N-Chars in the host receive buffer. In addition, the NULL control code (ESC + FCT) is escape sequence and may be regarded as L-Chars. NULLs shall be sent by transmitter to indicate that the link is still active, if there are no other characters to be delivered. N-Chars shall be passed on to the packet level including data characters (10-bit) and end-of-packet markers (EOP and EEP, 4-bit both).

A. Modeling

1) Controller: The Controller controls the overall operation of the link interface. The state transition diagram is illustrated in Figure.2.

The state machine includes six states: ErrorReset, ErrorWait, Ready, Started, Connecting and Run. After a reset signal from the system, the ErrorReset state shall be entered. The Transmitter and Receiver shall be reset at the same time. After a 6.4μs delay the ErrorReset state shall be unconditional left and the ErrorWait state is entered. In the Errorwait state, the Receiver is enabled in order to receive NULLs from the other link end. The state machine shall move into the Ready state after a delay of 12.8μs, which indicates that the link interface is ready to initialize. The Started state shall be entered if the link interface is enabled, where the Transmitter is able

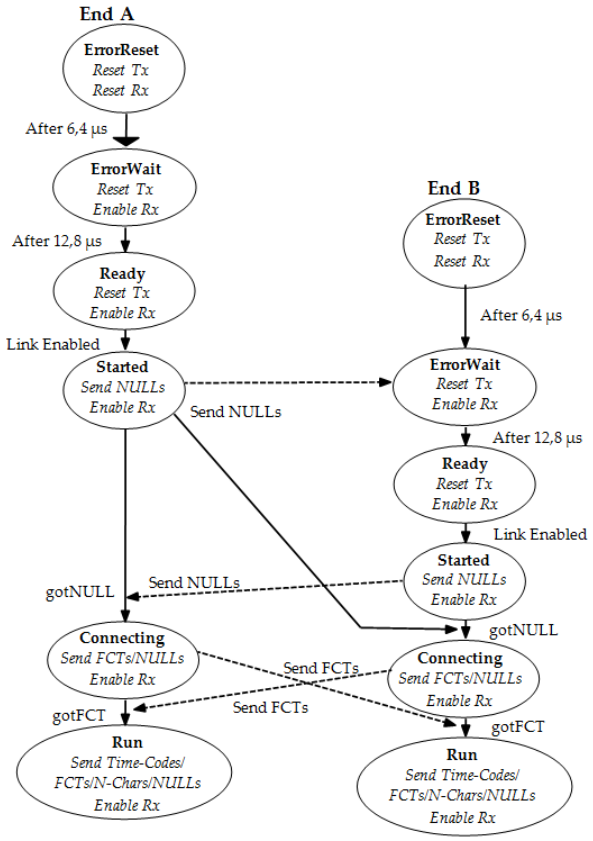


Fig. 1. Typical Link Initialization sequence

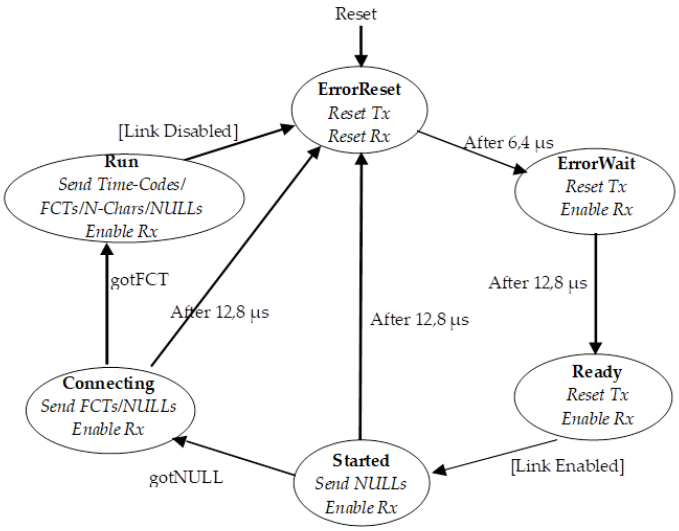


Fig. 2. Basic state diagram for SpaceWire link interface

to send NULLs and NULLs are expected to be received. If no NULLs are arrived after a 12.8μs timeout, the state machine shall be reset moving to the ErrorReset state. Otherwise, it shall move to the Connecting state and send at least one NULL. Both NULLs and FCTs are allowed to be sent. And FCTs are expected to be received. Once the arrival of an

FCT, the state machine shall move to the Run state, where the link initialization is made successfully. If an FCT fails to arrive within 12.8μs, there may be something wrong with the link connection. The state machine shall move back to the ErrorReset state and attempt to make connection once again. Resorting to the Uppaal, the Controller is modeled as a timed automaton illustrated as Figure.3.

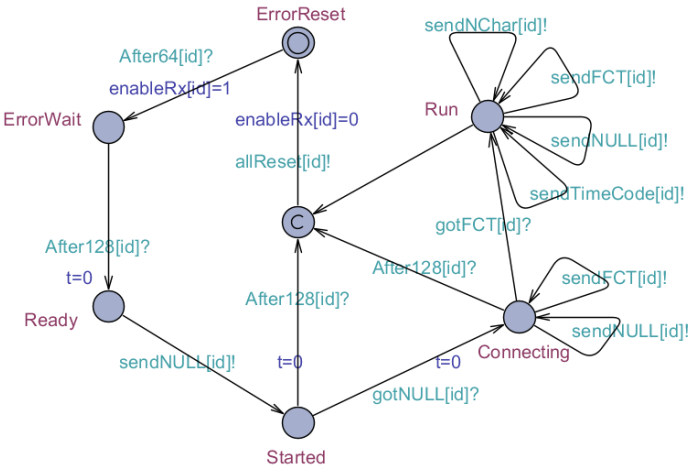


Fig. 3. Timed Automaton of the Controller in Uppaal

2) Timer: The timer provides the 6.4μs and 12.8μs timeouts in link initialization. Its functional block diagram is illustrated as Figure.4.(a)

A timed automaton of timer is modeled as Figure.4.(b). A clock variable *clk* is defined as the system clock. The time unit is set to be 100ns, due to the data signalling rate of 10M/s for link initialization, which means the transmitter shall send a bit per unit of time (100ns). A bool variable *trigger* defined to trigger the *After64* or *After128*: 0 meaning *After64* waiting for trigger, 1 meaning *After128*.

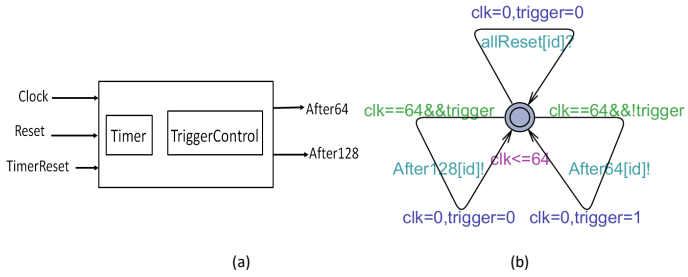


Fig. 4. Diagram of the Timer module interface(a),Timed Automaton of the Timer in Uppaal(b)

3) Transmitter and Receiver: The transmitter is responsible for encoding data using Data-Strobe(DS) encoding technique. It transmits N-Chars, FCTs, NULLs and Time-Codes requested to be sent from the host system. The receiver decodes the DS signals(Din and Sin) to produce N-Chars, FCTs, NULLs, and Time-Codes. When detecting the arrival of the NULLs or FCTs, it shall report to the state machine of Controller.

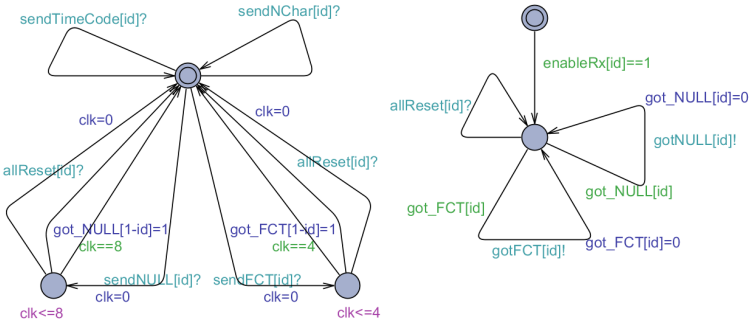


Fig. 5. Timed Automata of Transmitter and Receiver

Considering these two modules are not the key issue we care about in this paper, we extract a high level model assuming they function well. See the Figure.5.

B. Verification

Uppaal uses a simplified version of CTL to express the requirement specification. Path formulae and state formulae are both supported as shown in the following Tabular.

Property	Expression	Interpretation
Reachability	E<>p	p is satisfied in reachable states
Safety	A[]p, E[]p	something bad never happens
Liveness	A<>p, p− >q	p is eventually satisfied
Extrema	infexpression:list supexpression:list	infima of the list suprema of the list

We specify several properties to verify the correctness of the timer module and controller module as follows:

1) The signal *After64* of Timer should be triggered after 64 units of time since reset.

```
E<>StateMachine(0).ErrorWait && Timer(0).clk_test==64
Verification/kernel/elapsed time used: 0s / 0s / 0.017s.
Resident/virtual memory usage peaks: 9,120KB / 29,944KB.
Property is satisfied.
```

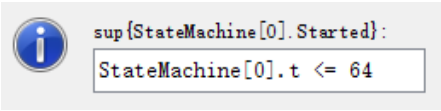
2) The signal *After64* of Timer should be triggered after 128 units of time since reset.

```
E<>StateMachine(0).Ready && Timer(0).clk_test==128
Verification/kernel/elapsed time used: 0s / 0s / 0.015s.
Resident/virtual memory usage peaks: 9,916KB / 31,536KB.
Property is satisfied.
```

3) Link connection can be made successfully.

```
E<>StateMachine(0).Run &&StateMachine(1).Run
Verification/kernel/elapsed time used: 0.016s / 0s / 0.013s.
Resident/virtual memory usage peaks: 8,736KB / 30,548KB.
Property is satisfied.
```

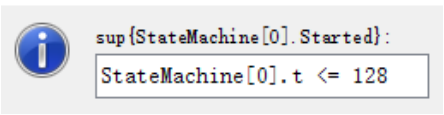
4) How long will the Controller stay in the state of Started at most?



The result turns out to be 64(6.4μs). However, the answer should be 128(12.8μs). If a NULL fails to arrive within 12.8μs in the Started state, the state machine shall unconditionally move into the ErrorReset state. Accordingly, we find the timer is not reset before entering the Started state. The same problems are also occur in the state of Ready and Connecting. Thus a process of ResetTimer is added as follow, which is responsible for resetting the timer when in the ErrorReset, Ready, Started and Connecting state.

```
CASE ResetTimer_state IS
WHEN ErrorResetDetected => TimerReset <= '1';
ResetTimer_state <= ErrorResetLeft;
WHEN ErrorWaitDetected => TimerReset <= '1';
ResetTimer_state <= ErrorWaitLeft;
WHEN StartedDetected => TimerReset <= '1';
ResetTimer_state <= StartedLeft
```

The models of the Controller and Timer are modified. We check the property 4) again, the result is shown as follow.



IV. Conclusion

Resorting to the model checker Uppaal, this paper established the network of timed automata of SpaceWire interface, including modules of Controller, Timer, Transmitter and Receiver. Both the interface to end A and end B were modelled. We paid attention to link initialization across the SpaceWire link. As a real-time communication network, SpaceWire link should provide real-time end-to-end data transfer. This paper focused on the time performance during the link initialization. Through the verification of time property of the link initialization, some errors in detail were found. Thus, it turns out that model Checking is an effective method to model and test. As one of the method of formal verification, model checking is promising. Future work may focus on the application to more real-time system using Uppaal.

Acknowledgment

We thank the group of formal verification of Capital Normal University for the work done before. Thank X.J. Li for answering the question about SpaceWire protocol, R. Wang for helping with Uppaal, B.Q.Zhou for the paper writing and revising. S.L. Xie did a lot of favor in this work.

References

[1] http://www.spacewire.esa.int/content/Home/HomeIntro.php
[2] European Cooperation for Space Standardization, *SpaceWire-Links, nodes, routers and networks*, ECSS-E-ST-50-12C, July 2008, available from http://www.ecss.nl/.
[3] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 2000
[4] Edmund M. Clarke, *The Birth of Model Checking*, Lecture Notes in Computer Science, 2008, Volume 5000/2008.
[5] http://spinroot.com/spin/whatispin.html
[6] A. Cimatti, E. Clarke, F. Giunchiglia, NUSMV: a new symbolic model checker, International Journal on Software Tools for Technology Transfer, 2000, Volume 2, Issue 4, pp 410-425.
[7] http://www.uppaal.org/
[8] Gerd Behrmann, Alexandre David, and Kim G. Larsen, *A tutorial on Uppaal*, In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS 3185.
[9] R. Alur and D. L. Dill. *A theory of timed Automata*. Theoretical Computer Science 126(2):183-235, 1994
[10] Johan Bengtsson and Wang Yi, *Timed Automata: semantics, algorithms and tools*, In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.
[11] C. McClements, S.M. Parkes, and A. Leon, *The SpaceWire CODEC*, International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003
[12] Christel Baier and Joost-Pieter Katoen, *Principles of Model Checking*, London, England: The MIT Press .2008
[13] R. Wang, X. Song and M. Gu, *Modelling and verifcation of program logic controllers using timed automata*, The Institution of Engineering and Technology, 2007, 1, (4), pp.127-131
[14] Kim G. Larsen, *Formal methods for meal mime systems: automatic verification and validation*, presented at the ARTES summer school, August 1998.