Verification of an Ad-hoc Serial Communication Protocol through Model-checking: A Case Study with Echo Sounder

Shakaiba Majeed[1], Kashif Saghar[2], Kashif Hameed [2], Minsoo Ryu[3*]
[1] Department of Computer and Software, Hanyang University, Seoul, South Korea,
[2] CESAT, 44000, Islamabad, Pakistan
[3]Department of Computer Science and Engineering, Hanyang University, Seoul, South Korea

* Corresponding author Email: msryu@hanyang.ac.kr
Manuscript submitted Feb 20, 2017; accepted April 2017.
doi: 10.17706/jsw.12.4.253-264

Abstract: Serial data transmission accounts for a considerable share of the overall communication involved in real-time embedded systems. Although there are some standard serial protocols, many systems still use ad-hoc serial protocols for communication between a control computer and a serial peripheral device. Such protocols may have flaws in them which cannot be revealed by computer simulations or testing only. To complement testing, formal methods are now widely used and have proved effective in the verification of various communication protocols. However, for serial communication specifically, most of the previous research is focused on applying formal methods for the verification of standard serial interfaces. In the current work instead, we use formal methods to verify an ad-hoc serial communication protocol between a control computer and an echo sounder. Through our case study, we show how we integrated formal modeling and model-checking methods in an existing system and as a result, we were able to discover a fault in the protocol design, which could have gone unnoticed without formal software verification.

Key words: Formal-modeling, model-checking, fault-injection, serial data transmission.

1. Introduction

Serial data transmission, being simple and cost effective, marks a considerable share of the overall communication involved in embedded systems. There are a number of safety critical applications that rely on the use of wired or wireless serial communication for the exchange of data between various components of the system. NMEA-0183 standard, for example, is used in GPS receivers for the transmission of positioning data to the host computer [1], radar altimeters communicate serially with the aircraft control computer via wired interfaces such as RS-232/422 [2]. Similarly, LIN provides a cheap serial alternative to CAN bus for communication between components in automobiles.

While there are a few standard serial communication protocols, many others are custom designed by different manufacturers that produce serial peripheral devices. These protocols provide very similar functionalities but may involve different processes required for successful software interfacing between the specific communicating devices. To guarantee the correctness and reliability of the critical information that is exchanged through such serial communication links, it is therefore very important to ensure the correctness of the design and implementation of the serial communication protocols.

Static reviews and testing (including computer simulations and HILS) are the most common methods used for the verification of software. However, it has been concluded through research that such methods of verification are limited to detecting specific classes of fault [3]. Therefore to expose subtle concurrency and

algorithmic errors, formal verification methods are needed. Moreover, serial devices such as those mentioned earlier are often deployed in harsh onboard environments, where attenuation of signals and interference from neighboring devices can cause the corruption of data on the serial links. To ensure that only correct and reliable information is conveyed to the destination device, the designed protocols must be able to deal with faults and continue error free functioning. This requires that the protocol be tested against all the relevant behaviors of the system including the effects of the onboard environment. Unfortunately, exhaustive testing of such systems with reasonable time and effort is not feasible.

Formal methods, such as model-checking, provide valuable support to verify the correct functioning of a system design model [3]-[5]. In contrast to traditional verification approaches, model-checking permits systematically and exhaustively examining the behavior of any system through its formal model. Over the last two decades, model-checking has proven to be effective for hardware verification [6], [7], analysis of real-time embedded systems [8], [9], and validation of data communication protocols [10]-[12]. However, to our knowledge, very few papers have been published on model-checking of serial communication protocols [13], [14] and those that do, focus on verifying data and control logic of the standard serial interfaces. In the current work, we aim to analyze and verify an ad-hoc serial communication protocol at the packet transfer level, using model-checking. We develop a formal model of the protocol with the aid of protocol state diagrams and/or event-sequence diagrams and derive specification properties for the model from the system requirements. To ensure the fault tolerant behavior of the protocol in a noisy environment, we integrate the system design model with an environment model. This environment model injects faults into the communication, emulating the effect of data corruption in a noisy onboard environment. The integrated model is then analyzed and verified through the automatic model-checker UPPAAL. If a specification property is not satisfied, a counter-example is generated by the model-checker. Since abstractions are involved in developing a formal model, it may sometimes lead to incorrect conclusions or false-positives. Therefore, we validate the counter-example through computer simulations to confirm the fault and fix it. Following the same procedure, the improved protocol design model is verified again, until all the faults have been removed.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the conventional serial protocols. Section 3 presents the proposed framework for the verification of such protocols, and Section 4 describes a case study on the serial interfacing of an echo sounder with a control computer where we used our proposed framework based on model-checking to reveal faults in the serial protocol design. Section 5 includes some of the research work related to the topic and we conclude with Section 6.

2. Review of Conventional Serial Communication Protocols

In many safety critical systems, serial communication at relatively low baud rates occurs between two modules that exchange data through asynchronous medium such as RS-232, RS-422 or RS-485 serial cables [15]. The smallest information unit that is exchanged in asynchronous serial communication is a data frame. Each data frame consists of start bits, data bits, stop bits and optional parity bits as indicated in Fig. 1.

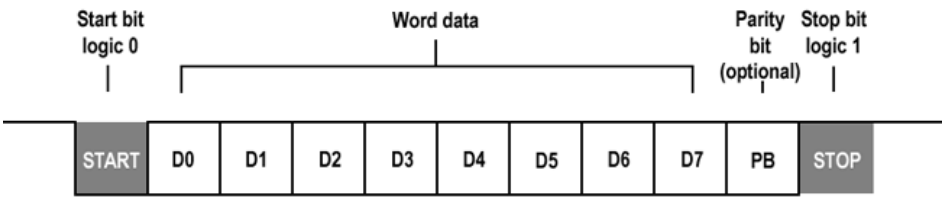


Fig. 1. Format of a typical serial data communication frame.

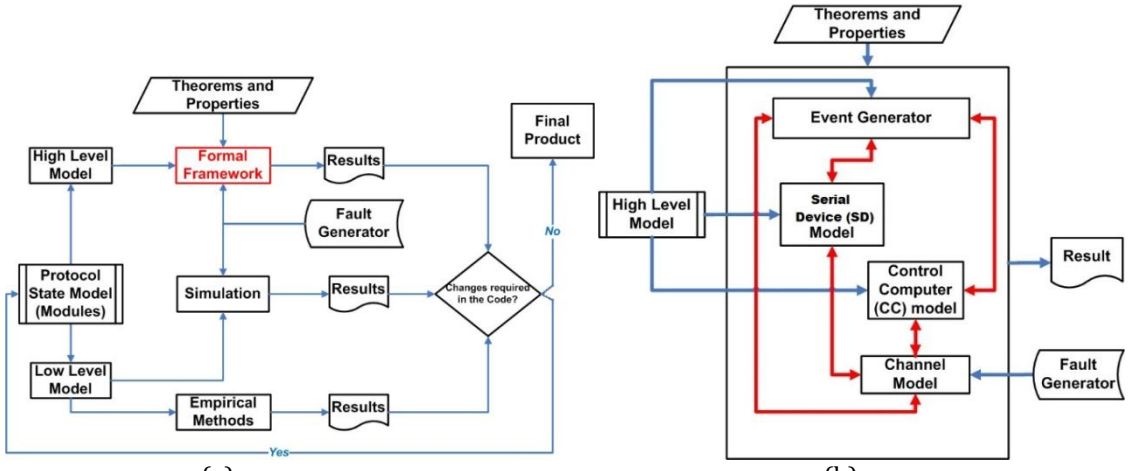
Table 1. Format of a Typical Serial Data Packet of Length n

Byte number	Description 1
Byte 0	Packet Header
Byte 1	Packet Length
Byte 2~Byte n-3	Data
Byte n-2~n-1	Check word for Packet

The information exchanged between two modules occurs in the form of packets consisting of multiple data frames as defined by a protocol. An example of a typical data packet is shown in Table 1. In this regard, a communication protocol is a well-designed agreement of data transmission between any two communicating entities that must be implemented to successfully comprehend the information. This protocol is often defined by the serial peripheral manufacturers and includes information, such as the rate of data transmission, types of packets, packet length, contents of packets, format of data included in the packets, and other handshaking mechanisms.

Serial communication protocols are generally verified through debugging, simulation, and testing [16, 17]. Initially, debugging tools help in revealing errors during the coding phase. Later, computer simulations are used to create a representation of the behavior of the protocol under consideration and to verify that it meets the desired requirements, but in the absence of the actual working environment. Finally, the implemented protocols in the communicating entities are subject to testing in an integrated working environment. It is worth noting that the effectiveness and the cost of the simulation and testing phase depends upon how well the test cases are designed to verify the functional requirements (black-box testing) and to cover the space of execution of the protocol (white-box testing).

3. Verification Through Model-Checking and Fault-Injection: The Proposed Framework

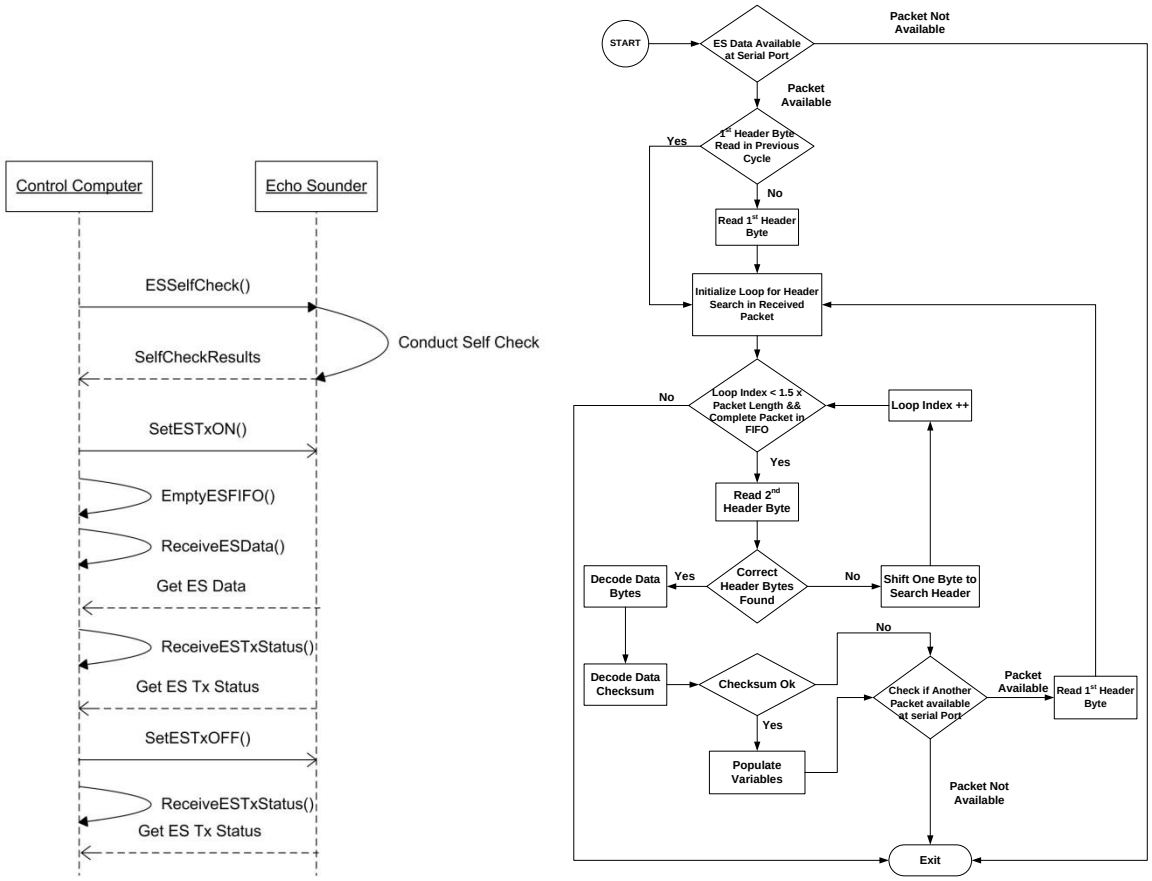


(a) (b)

Fig. 2. (a) Methodology Adopted (b) Formal framework block diagram.

The goal is to analyze and verify the design of a serial communication protocol in the presence of faults induced by channel noise. The method adopted to analyze serial communication protocols is shown in Fig. 2. (a). The protocol state model is a semi-formal definition of the protocol expressed as event sequence diagrams, flow charts, and the format of packets exchanged between two devices. To be more specific, this protocol state model is a skeleton of the modules that need to be implemented in each communicating

entity for the successful exchange of data on the serial communication link. This protocol state model is refined to generate a high-level and a low-level model. The high level model is transformed into a formal model and specification properties are defined to check the presence of any faults or bugs present in the protocol. The properties included basic sanity checks (confirmation that the model possesses some fundamental properties, debugging checks, etc.), the reachability checks (confirmation that all part(s) of the code are reachable), the liveness checks (something good will eventually occur) and the safety checks (nothing bad ever occurs). In case a property fails, the formal model-checker automatically generates a trace providing the reason as to how a bug occurred in the protocol. The low-level model is used to implement the protocol in a low-level simulator such as Matlab Simulink and also in actual communicating entities for verification through empirical testing methods.



(a) (b)

Fig. 3. (a) Echo sounder data acquisition event sequence diagram (b) Flowchart for ReceiveESData()

The formal framework, shown in Fig.2.(b), is comprised of four main parts: a control computer (CC) model, a serial peripheral device (SD) model, an event generator (EG) model, and a channel model. The CC model represents the host device, which may be performing some control operations based upon the information received from the serial peripheral device. CC is therefore, a collection of the formal models of all the modules that need to be implemented on the control computer side as defined by the protocol specifications. Similarly, the SD model is a collective representation of the formal models of every module on the peripheral side. The channel model represents the properties of the communication medium. All modules on each side communicate with one another through this channel model. To enable fault-injection based testing, a fault generator is used to introduce reception faults into this channel model. Finally, the event generator model is used to generate the different events required in the protocol that are external to

the communicating devices. These events are basically triggers to enable devices to perform tasks such as sense data from the environment, generate a timer's timeout, or finish a particular phase. In UPPAAL, sometimes, instead of using quantized time (clocks), an event generator is employed to generate timeouts to indicate that a phase has been completed. In the current context, readers may object to employing a timed-automata model-checker, UPPAAL, rather than using a simpler one such as SPIN. However, we prefer UPPAAL because of its rich graphical user interface that helps in constructing an abstract model of embedded system and simulating its dynamic behavior.

In the next section, we describe a case study to explain how we use the abovementioned framework to verify the serial communication protocol between the control computer and an echo sounder.

4. Case Study on Interfacing the Echo Sounder with a Control Computer

The current case study involves measurement of distance/depth under water by an echo sounder (a type of sonar). An echo sounder is mounted beneath or over the side of a boat, "pinging" a pulse of sound downward at the seafloor. The pulse travels down through the water, bounces off the seafloor, and then travels upwards until the reflection is heard by the echo sounder. To calculate the distance, the echo sounder measures the time that the pulse takes to travel to the seafloor and back to the ship. Such echo sounders are generally interfaced to a control computer that helps in correcting the attitude of the boat during the surveying exercise. The echo sounder under consideration provides data to the control computer through an RS-422 serial interface. To ensure that the design of the serial communication protocol between the control computer and echo sounder is free of flaws and that correct data is guaranteed with each reading, we verified the protocol using the proposed framework.

The subsections provide the details.

4.1. The Protocol State Model

The serial communication protocol was designed by the echo sounder manufacturer. This particular echo sounder generates an echo signal every 20 ms, and the corresponding undersea depth is transmitted to the control computer's receiving port at a baud rate of 20.833 kbps. The data transmission is realized in the form of data packets where a lower significant byte comes first, followed by a higher byte. Each packet is saved in the receiving port's FIFO until it is read by the control computer. The echo sounder transmits packets in the form of the following message:

$$Node_E \rightarrow Node_C : (H,\ L,\ DATA_{MSB,LSB},\ CS_{H,L,Data})$$

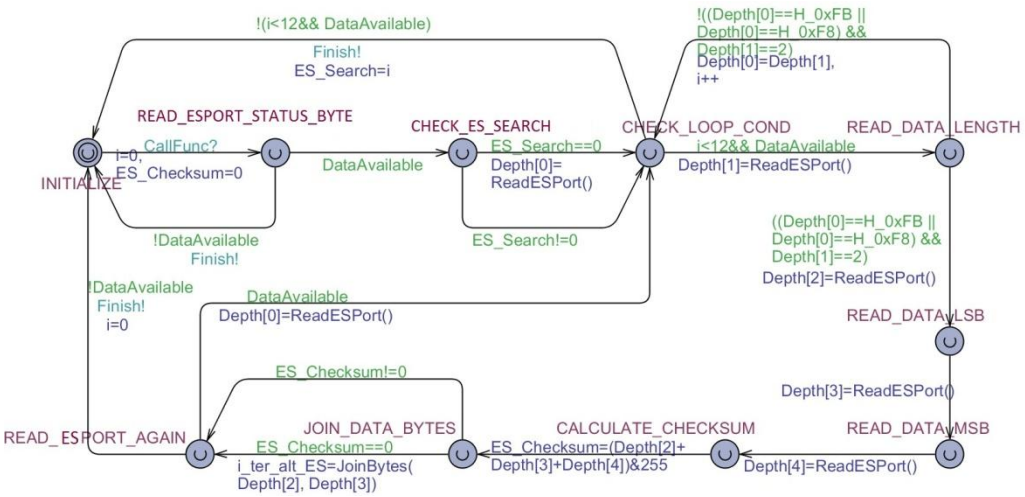
In this message 'H' is a one byte message header usually 0xF8 or 0xFB. 'L' is the length of data in bytes and is 0x02. The 'DATA' is a two byte sensed depth. The valid range of the depth is from 0 to 2000 m. Finally 'CS' is the 2's complement of the sum of data bytes. The sequence of tasks for the successful acquisition of data from the echo sounder is depicted by the event sequence diagram shown in Fig.3 (a).

Before acquiring the data from the echo sounder, it is necessary to ensure its correct operation and the health status of its internal circuitry. The first command, therefore, sent by the control computer is self-check. Upon receiving this command, the echo sounder undergoes a pre-defined self-check procedure, and the result is dispatched serially to the control computer. On receiving the desirable self-check response, a command to turn ON the echo sounder transmitter is initiated. The feedback of this command is obtained through a digital input signal on the control computer. Once the transmitter is turned ON, the echo sounder starts sending the current under water depth, which is periodically received by the control computer and stored in some non-volatile memory. The data acquisition may be stopped at any time by sending the transmitter OFF command which again can be confirmed through the same digital input signal.

The entire event sequence can be considered consisting of different modules that need to be implemented on each side of the transmission. The modules on the echo sounder side were pre-programmed by the manufacturer. The receiving modules on the control computer were implemented and tested using the proposed framework. Due to the lack of space, it is not possible to show the details for each module; therefore, we shall discuss here only one module called "ReceiveESData()" and also present its results.

4.2. Formal Modeling and Model Checking using UPPAAL

Based upon the event sequence diagram and the details of the packet information provided by the manufacturer, a flow chart of the ReceiveESData() is constructed, as shown in Fig.3 (b).This serves as a high-level model used to develop a formal model of the ReceiveESData() module using UPPAAL. The resulting formal model for the module and its event generator model are shown in Fig.4. The environment model is made a part of the event generator illustrated by Fig.4.(b). The state ADD_ERROR in the event generator model induces faulty bytes in a transmitted packet based upon an allowable threshold, ERROR_PERCENTAGE. The bytes to be replaced by faulty bytes in a packet are selected non-deterministically between indexes 0 to MAXBYTES-1, where MAXBYTES is the maximum number of bytes contained in a packet.



(a)

(b)

Fig. 4. Formal model for ReceiveESData module (a) Function model (b) Event generator model.

The properties of the formal model are checked against basic safety (S), liveness (L), sanity (T), and reachability (R) tests. Again due to a shortage of space we will only show the results of conducting safety and liveness tests:

4.2.1. Safety tests

S1) Theorem: *There is no deadlock in the software*

This test confirms that even if the function is called infinite times, no deadlock can occur. The property used is

A[]!deadlock

S2) Theorem: *The software function will always read the correct depth*

This test confirms that the function will read the correct depth when the desired number of bytes in a packet have been communicated, and there is no checksum error in the packet. The property used is

A[]EventGen.FINISH ⇒ (Function.iterdepth == Depth)

S3) Theorem: *When the received Packet's header and data length bytes are correct, the function always reads LSB of data*

This test confirms that the function will read the data's LSB byte when the first two bytes of the received packet are correct. The property used is

A[](((Function.Depth[0] == 0xF8 || Function.Depth[0] == 0xFB) && Function.Depth[1] == 2)⇒ Function.READ_DATA_LSB)

S4) Theorem: *The function always reads the MSB of the data when the received packet's first data byte has been read*

This test confirms that the function will read the data MSB byte when the first two bytes of the received packet are correct and the first data byte has been read. The property used is

A[](Function.READ_DATA_LSB == LSB⇒ Function.READ_DATA_MSB)

S5) Theorem: *When received Packet's header and data length bytes are correct the function always reads checksum byte of data and calculates checksum*

This test confirms that function will read data checksum byte to calculate checksum when first two bytes of the received packet are correct. The property used is

A[](((Function.Depth[0] == 0xF8 || Function.Depth[0] == 0xFB) && Function.Depth[1] == 2) ⇒ Function.CALCULATE_CHECKSUM)

S6) Theorem: *When received Packet's header and data length bytes are correct the function always joins the data bytes*

This test confirms that function will join the data bytes when first two bytes of the received packet are correct. The property used is:

A[](((Function.Depth[0] == 0xF8 || Function.Depth[0] == 0xFB) && Function.Depth[1] == 2) ⇒ Function.JOIN_DATA_BYTES)

S7) Theorem: *When the checksum of a packet has been calculated, the function will always read the port for next packet*

This test confirms that function will always check the receiving port for the availability of the next packet when a complete packet has already been received and its checksum computed. The property used is

A[](Function.CALCULATE_CHECKSUM ⇒ Function.READ_ESPORT_AGAIN)

S8) Theorem: *FIFO will never overflow*

This test confirms that FIFO will never overflow. The property used is:

A[] FIFOindex < FIFOSIZE

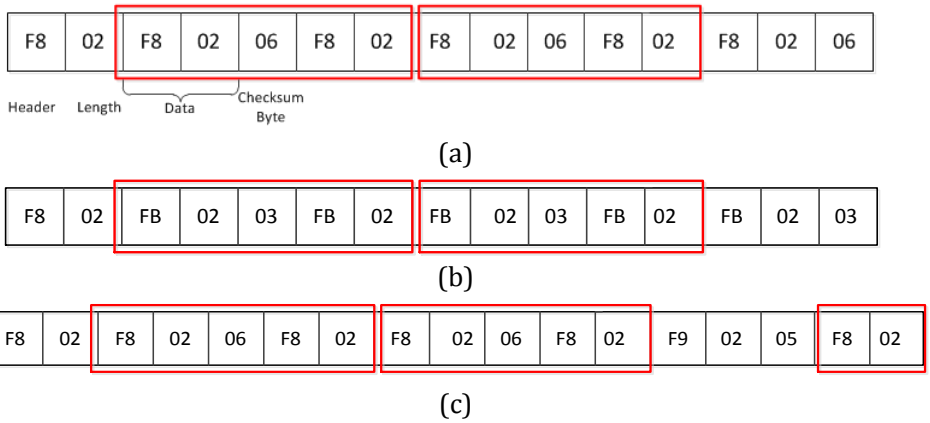


Fig. 5. ReceiveESData packets (a) Depth 760 m (b) Depth 763m (c) Synchronization of packets

4.2.2. Liveness tests

L1) Theorem: *In all paths eventually the FIFO will become empty, i.e. function will read all values of FIFO*

This test confirms that the code will read all values of FIFO, i.e. in all paths eventually the FIFO will become empty. The property used is

A◊ FIFOEmpty

L2) Theorem: *Event generator will write data on Echo Sounder port and function will be called to read the data*

This test confirms that in all paths, the event generator will be able to write desired data on the port, and consequently the corresponding function will be called to read the data from port. The property used is:

A◊ EventGen.FINISH

4.2.3. Results

In the abovementioned tests, Theorem S2 stating that "The software function will always read the correct depth" has failed. The trace has confirmed two special cases in which a single header byte corruption in a packet can cause an incorrect interpretation of the data bytes not only in the current packet but also in subsequent packets. The ReceiveESData() module according to protocol state model identifies the start of a packet by comparing the first two bytes received against a predefined header value. In this case, if the first byte is 0xF8 or 0xFB and the second byte is 0x02, the ReceiveESData() considers it as a valid header, calculates the check sum of packet, and reads the next two bytes from the header as the depth provided by the echo sounder. Consider a situation where the echo sounder is sending depth of 760 m for some consecutive readings, i.e. 0x2F8 as shown in Fig.5.(a). The LSB in that case is 0xF8, and MSB is 0x02. The complete packet is $(F8\ 02\ F8\ 02\ 06)_H$, if now a header byte is lost, the function will join bytes from the next packet and consider data as header as indicated by the red rectangle. Note that in this particular case the check sum will also be valid. The same problem arises in cases where the depth is 763, as shown in Fig.5.(b). Therefore, if the echo sounder is continuously sending these two values after the first header byte error, the

code will never report the correct depth value unless the value is changed from 760 m or 763 m. This is shown below in Fig.5.(c) where a change in depth will eventually synchronize the software with the header byte. Such a fault is impossible to be revealed by simulation or HILS testing as it requires exploring the behavior of the protocol using all the possible input values and in various faulty scenarios.

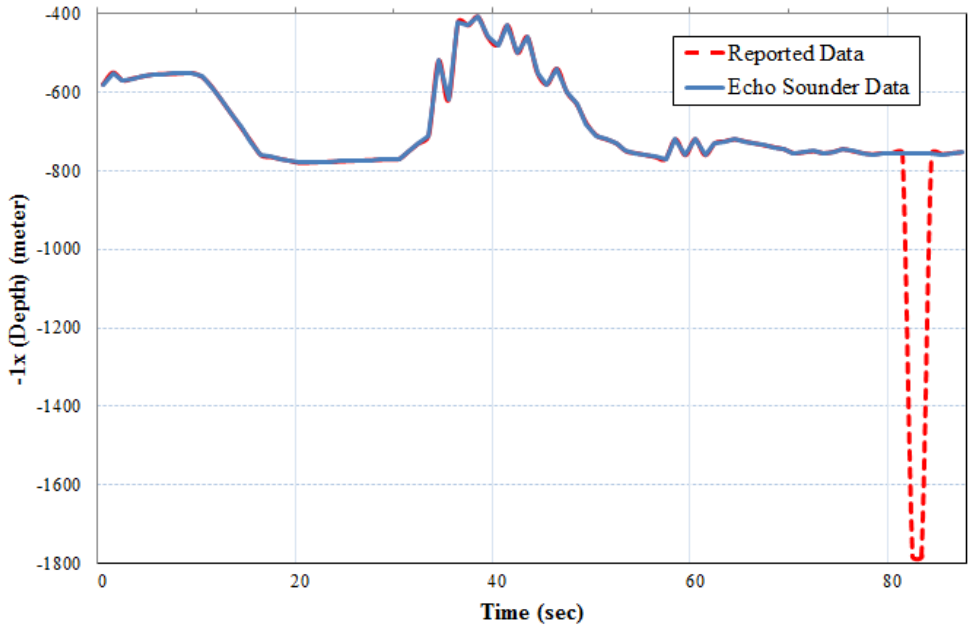


Fig. 6. Validation of error case through simulation

4.3. Validation through Simulation

In order to validate the error case suggested by the formal verification exercise, a real time simulation setup was established. The control computer has a dedicated RS-422 serial port for data acquisition from the echo sounder. A Simulink model was designed on a real-time computer to simulate the echo sounder. The true data stream from the echo sounder model was perturbed using the fault injection case suggested by model-checking. The underwater depth data is shown in Fig.6, where the blue graph was actually generated by the model. An injected header byte loss at the instant of 82 seconds, caused the next reading to be misinterpreted as the header. Thus, the depth reported at that particular instant and also in the subsequent frames is 1784 m (0x6F8) as indicated by the red graph. Note that when the depth is changed from 760 the frames are automatically synchronized by the protocol, and a correct depth is registered.

The results of the simulation exercise are consistent with those obtained through model-checking, thus confirming that the problem was not due to any modeling artifact but because of a deficiency in the protocol. The communication protocol between the echo sounder and control computer is weak in this regard as the value of the header and data length 0x2F8 overlaps with the range of depth being provided by the echo sounder. The fault was reported to the echo sounder manufacturer so that the header could be changed to some other reasonable value.

5. Related Work

Use of formal modeling and model-checking to verify communication protocols is not a new practice. A number of different protocols have been verified using formal modeling. Camara et al. [18] applied formal methods using SPIN to verify the design bugs in protocols for MANET networks such as LAR, DREAM and

OLSR. Fehnker et al. [19] and Henderson et al. [20] used UPPAAL to identify flaws in the WSN protocols. However, we see only a little evidence of the adoption of such approaches when ad-hoc serial communication protocols are in question. Most existing work in protocol design and verification in the serial communication domain is based upon computer simulations and testing [16, 17, 21]. However, we believe that such protocols, especially in safety critical systems, should be verified and tested three times: initially by formal verification through model checking, followed by a software simulator, and finally using a hardware prototype.

Similar to our approach, Bergenhem and Karlsson[22] also proposed a framework for fault-injection based testing of protocols that implement fault tolerance and redundancy management in safety-critical distributed real-time systems.

Saghar et al. [23] have also used a fault injection based formal framework to analyze the effect of Denial of Service (DoS) attacks on routing protocols in WSNs.

6. Conclusion

This paper presents a case study of modeling and analysis of a serial communication protocol for an echo sounder using the UPPAAL model checker. The use of formal modeling into our existing testing framework helped us in detecting the vulnerability of the protocol against faults induced by channel noise. We later use simulations to confirm the presence of this vulnerability by injecting the faulty scenario as directed by the trace information obtained from model-checking. The improvements done in order to remove the fault were further analyzed by the model-checker, as a feedback process to reach a qualified model and implementation.

In the future, we intend to apply this approach to the analysis of other ad-hoc protocols used in our setup to interface sensors to the control computer and employ the results to develop more robust protocol(s).

Acknowledgment

This work was supported by the ICT R&D program of MSIP/IITP, Korea [R01141600460001002, Software Black Box for Highly Dependable Computing] and by CESAT Pakistan.

References

[1] Cao, T., & Gao, Y. (2006). Application of NMEA-0183 Protocol for GPS. *Electronic Engineer, 32*, 8-11

[2] *Free Flight Systems-Radar Altimeter Specifications*. Retrieved August 9, 2016, from http://www.freeflightsystems.com/images/products/RADALT.pdf

[3] Baier, C., Katoen, J.-P., & Larsen, K. G. (2008). *Principles of Model Checking*. MIT Press.

[4] Clarke, E. M., Grumberg, O., & Peled, D. (1999). *Model Checking*, MIT Press.

[5] Visser, W., Havelund, K., Brat, G., Park, S., & Lerda, F. (2003). Model checking programs. *Automated Software Engineering.*

[6] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., & Hwang, L.-J. (1992). Symbolic model checking: 1020 states and beyond. *Information and Computation*.

[7] McMillan, K. L. (2000). A methodology for hardware verification using compositional model checking. *Science of Computer Programming*.

[8] Madl, G., Abdelwahed, S., & Schmidt, D. C. (2006). Verifying distributed real-time properties of embedded systems via graph transformations and model checking. *Real-Time Systems*.

[9] Shukla, S. K., & Gupta, R. K. (2001). A model checking approach to evaluating system level dynamic power management policies for embedded systems. *Proceedings of the Sixth IEEE International High-Level Design Validation and Test Workshop.*

[10] David, A., & Yi, W. (2000). Modelling and analysis of a commercial field bus protocol. *Proceedings of the 12th Euromicro Conference on Real-Time Systems*.

[11] Kwiatkowska, M., Norman, G., & Sproston, J. (2002). Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. *Process Algebra and Probabilistic Methods: Performance Modeling and Verification*.

[12] Marrero, W., Clarke, E., & Jha, S. (1997). Model checking for security protocols.

[13] Plessier, B., & Pixley, C. (1995). Formal verification of a commercial serial bus interface. *Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications*.

[14] Gorai, S., Biswas, S., Bhatia, L., Tiwari, P., & Mitra, R. S. (2006). Directed-simulation assisted formal verification of serial protocol and bridge. presented at the proceedings of the 43rd annual Design Automation Conference, San Francisco, CA, USA, 2006.

[15] Claus, M. J., & Nguyen, H. V. (2012). Serial communications protocol for safety critical systems.

[16] Feng, J., & Ding, Y. (2015). Design and implementation of embedded serial communication based on finite State machine. *International Journal of Hybrid Information Technology, 8*, 25-32.

[17] Wang, Q., Ge, H., & Huang, M. (2011). The implementation of serial communication technology in the vehicle alcohol lock. *Proceedings of the 2011 4th International Congress on Image and Signal Processing (CISP)*.

[18] Câmara, D., Loureiro, A. A. F., & Filali, F. (2007). Methodology for formal verification of routing protocols for ad hoc wireless networks. *Proceedings of the IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*.

[19] Fehnker, A., Van Hoesel, L., & Mader, A. (2007). Modelling and verification of the LMAC protocol for wireless sensor networks. in *International Conference on Integrated Formal Methods*.

[20] Henderson, W. D., & Tron, S. (2006). Verification of the minimum cost forwarding protocol for wireless sensor networks. *Proceedings of the 2006 IEEE Conference on Emerging Technologies and Factory Automation*.

[21] Lee, J.-H., Hwang, J.-G., & Park, G.-T. (2005). Performance evaluation and verification of communication protocol for railway signaling systems. *Computer Standards & Interfaces*.

[22] Bergenhem, C., & Karlsson, J. (2008). An environment for testing safety-critical protocols. *Proceedings of 2008 3rd IET International Conference on the System Safety*.

[23] Saghar, K., Henderson, W., & Kendall, D. (2009). Formal modelling and analysis of routing protocol security in wireless sensor networks. *Proceedings of the 10th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*.

Shakaiba Majeed received her BSc. engineering degree in electrical and electronic engineering from the University of Engineering & Technology Taxila, Pakistan, in 2004 and MS electronics engineering degree in 2010 from Hanyang University, South Korea. Currently, she is a PhD candidate at the Department of Computer Engineering at Hanyang University. Her research interests include software development, testing and debugging of embedded systems.

Kashif Saghar received his M.Sc. and PhD degrees in embedded computer systems from the University of Northumbria, Newcastle upon Tyne, England, UK in 2007 and 2010 respectively. He is currently working as project director (modelling & verification) in CESAT and is involved in research activities in National University of Sciences and Technology, Pakistan. His research interests include formal methods, model checking & verification, wireless communication & networks, and software quality& testing systems.

Kashif Hameed received his PhD (2010) degree in Software Engineering from UWE, Bristol, U.K. contributing additional knowledge in Aspect Oriented Software Development (AOSD), He did his M.Sc. (2001, UWE) in Software Engineering with dissertation at MOTOROLA Swindon U.K., where he developed a prototyping tool for doing GPRS Network Health Analysis. His current interest includes design & development of fault-tolerant soft systems and dependability/reliability assessment of real-time Systems.

Minsoo Ryu is a professor at Department pf Computer Science and Engineering, Hanyang University, South Korea. He received his PhD from School of Electrical Engineering and Computer Engineering, Seoul National University in 2002. His research is focused on real-time embedded systems. Specific research areas include real-time system design and analysis, real-time operating systems and middleware, and diverse software engineering issues for multicore and/or manycore embedded computing systems.