F2016-VESF-002

ENHANCING AUTOSAR SAFETY MECHANISMS FOR ISO 26262 FUNCTIONAL SAFETY REQUIREMENTS

Noh, Soonhyun; Kim, Myungsun; Hong, Seongsoo*
Department of Electrical and Computer Engineering, Seoul National University, Republic of Korea

KEYWORDS – ISO 26262, functional safety, ASIL, AUTOSAR, software fault detection

ABSTRACT

As the importance of functional safety in electric/electronic (E/E) systems increased in the modern automotive industry, a global functional safety standard named ISO 26262 was proposed. ISO 26262 provides functional safety requirements for each E/E system in order to avoid unreasonable risks. The AUTOSAR (AUTomotive Open System Architecture) has been actively introducing safety mechanisms to satisfy such requirements but it still falls short of expectation. Particularly, the current safety mechanisms of AUTOSAR cannot completely deal with software faults that must be detected according to ISO 26262. In this paper, we propose two enhanced safety mechanisms for AUTOSAR so that AUTOSAR can detect all types of software faults listed in ISO 26262. We first propose an enhanced deadline supervision mechanism to detect the indefinite blocking of a task. We also introduce an end-to-end protection mechanism that can detect the delayed transmission of data. We implemented our solution on TriCore™ Starter Kit and demonstrated that the proposed solution successfully detected faults as desired.

INTRODUCTION

In the modern automotive industry, the importance of functional safety in electric/electronic (E/E) systems is increasing. To extensively address automotive functional safety issues, a global functional safety standard named ISO 26262 was proposed [1]. ISO 26262 is an adaptation of IEC 61508, a functional safety standard for general electrical/programmable electronic safety related systems, to the automotive domain. It provides for a structured approach to achieving safety goals while developing E/E systems.

ISO 26262 offers a hazard analysis and risk assessment scheme that helps identify and categorize hazards caused by malfunctioning behaviors of E/E systems. As a result of applying the scheme, each E/E system can be assigned an ASIL (Automotive Safety Integrity Level) based on three factors: severity, probability of exposure and controllability. There are four ASILs: ASIL A, ASIL B, ASIL C and ASIL D. ASIL A has the lowest risk and ASIL D has the highest. Based on an assigned ASIL, each E/E system is assigned its own safety requirements for hardware and software development. These requirements should be satisfied while designing, implementing, verifying and validating the system.

Software fault detection is one of the key safety requirements for the software development in ISO 26262. Accordingly, each E/E system must have safety mechanisms to detect software faults that might occur during execution. ISO 26262 defines software faults that must be detected via such safety mechanisms. There are three types of faults: (1) *timing and execution faults*, (2) *memory faults* and (3) *exchange of information faults*. In ISO 26262,

E/E systems with ASIL C and ASIL D must have safety mechanisms to detect all the three types of faults.

AUTOSAR (AUTomotive Open System Architecture) is an automotive software platform standard introduced by the European automotive industry in 2002 [2]. It provides software platform architecture, a design methodology, templates and APIs for automotive software developers. Since many automotive industries are involved in the developing process [3], it is important for AUTOSAR to provide platform-level safety mechanisms. AUTOSAR has been actively introducing safety mechanisms since its release 4.0.3, but it still fails to satisfy all the requirements of ISO 26262.

Particularly, the current safety mechanisms of AUTOSAR are incapable of detecting all types of faults listed in ISO 26262. First, it fails to detect the indefinite blocking of a task at runtime. This prevents an AUTOSAR-based system from detecting some of the timing and execution faults. Second, it cannot detect delayed data transmission between two ECUs. This may conceal some of the exchange of information faults.

In this paper, we introduce two advanced safety mechanisms for AUTOSAR in order to detect all three types of software faults listed in ISO 26262. We first propose to enhance a deadline supervision mechanism so that it can monitor program execution flows and check a deadline miss between two checkpoints on the program: start checkpoint and end checkpoint. Compared to the original deadline supervision mechanism of AUTOSAR, the proposed mechanism can detect deadline misses caused by task blocking before the task reaches its end checkpoint. To do so, it periodically checks how much time has passed since the start checkpoint and immediately returns an error when the deadline is violated. We also propose an end-to-end protection mechanism that can detect the delayed transmission of data. The proposed mechanism keeps computing the time difference between sent time and received time for every message reception to check delayed transmission. We implemented the proposed mechanisms on TriCore™ Starter Kit and showed that our mechanisms successfully detected faults as desired.

The remainder of this paper is organized as follows. In the next section, we give the background of the proposed work. We then state our problem to solve and give an overview of the proposed approach. We then explain the proposed solution mechanism in detail along with its implementation. Finally, we report on the experimental evaluation and conclude the paper.

BACKGROUND

To aid in understanding the rest of this paper, we explain the fault detection requirements of ISO 26262 and the safety mechanisms of AUTOSAR.

Fault Detection Requirements of ISO 26262

The software fault detection is one of the essential requirements that ISO 26262 mandates for software development. ISO 26262 clearly specifies software faults that might occur at runtime. Such faults are classified into three categories: (1) timing and execution faults, (2) memory faults and (3) exchange of information faults. The detailed classification is given in Table 1.

Timing and execution fault	Memory fault	Exchange of information fault
(a) Blocking of execution (b) Deadlocks (c) Livelocks (d) Incorrect allocation of execution time (e) Incorrect synchronization between software component	(a) Corruption of content (b) Read or write access to memory allocated to another software element	(a) Repetition of information (b) Loss of information (c) Delay of information (d) Insertion of information (e) Masquerade or incorrect addressing information (f) Incorrect sequence of information (g) Corruption of information (h) Asymmetric information sent form a sender to multiple receivers (i) Information from a sender received by only a subset of the receivers (j) Blocking access to a communication channel

Table 1: Classification of the software faults in ISO 26262

In order to detect such software faults, ISO 26262 specifies necessary software safety mechanisms for E/E systems. They are (1) a range check of input and output data, (2) a plausibility check, (3) detection of data errors, (4) external monitoring facility, (5) control flow monitoring and (6) diverse software design. E/E systems with ASIL C or ASIL D must have all these safety mechanisms. Among them, mechanisms (1) to (5) must be used at runtime. In order to reduce the duplication of efforts in developing safety mechanisms, it is encouraged to provide these safety mechanisms at the platform level.

AUTOSAR Safety Mechanisms

AUTOSAR has been actively adopting safety mechanisms to satisfy functional safety requirements of ISO 26262. Starting from the release 4.0.3, AUTOSAR includes safety mechanisms such as program flow monitoring, end-to-end protection and memory partitioning in order to satisfy the fault detection requirements of ISO 26262. Software developers should use these safety mechanisms while developing software components.

Program flow monitoring is a mechanism that checks the correct execution of software components. Program flow monitoring is conducted by a task named a *watchdog manager*. A watchdog manager is a basic software module that supervises the execution of a program. Its logical unit of supervision is called a supervised entity, which is a part of code of a software component. Each supervised entity has a set of checkpoints. There are three monitoring methods: (1) *alive supervision*, (2) *deadline supervision*, and (3) *logical supervision*. In alive supervision, the watchdog manager counts how many times the supervised entity reaches its checkpoint in a given period of time. If the count is less than the minimum threshold or larger than the maximum threshold, the watchdog manager returns an error. In deadline supervision, the watchdog manager measures the execution time between two checkpoints and returns an error when the execution time exceeds its threshold. In logical supervision, the watchdog manager monitors the execution order of checkpoints. It returns an error when the unsafe order is detected.

End-to-end (E2E) protection is a mechanism for protecting data against the effects of faults within the communication link. Before sending data from a sender to a receiver, the sender adds an E2E header to the data. The E2E header includes the safety-related information such as CRC (cyclic redundancy check) bits and a sequence counter. When a receiver gets the data, it checks, using the E/E header, whether sent data is correctly transmitted. Three methods can be used for such a safety check: (1) CRC, (2) data ID check and (3) sequence counter check. In CRC, the E2E protection wrapper compares the CRC checksum of the sender and the receiver to check whether the data is changed during the transmission [4]. In the data ID check, the E2E protection wrapper checks whether the data is sent from the correct sender or not. This is done by checking the data ID, which is an ID given to each port of a sender and a receiver. In the sequence counter check, a sequence counter is incremented at the sender on every transmission request, and its value is checked at the receiver.

Memory partitioning is a mechanism that enables a software component to run in a separate memory partition so that it does not interfere with other software components. Several software components can be grouped together and placed into a separate memory region. As a result, a software component in one group cannot modify the memory contents of another software component in a different group.

PROBLEM STATEMENT

The current safety mechanisms of AUTOSAR are incapable of detecting all types of faults listed in ISO 26262. Specifically, AUTOSAR cannot detect blocking of execution or deadlocks, which leads to an instance of the timing and execution fault. In AUTOSAR, a watchdog manger is triggered when a supervised entity reaches a checkpoint. Thus, a watchdog timer cannot work properly when a program is blocked indefinitely between two checkpoints. Also, AUTOSAR is incapable of detecting the delay of information fault that is an instance of the exchange of information fault. AUTOSAR does not offer APIs to check the freshness of transmitted data. Table 2 shows which software faults are covered by the AUTOSAR safety mechanisms.

The problems we address in this paper are as follows. (1) For given two checkpoints in a supervised entity, a safety mechanism should detect blocking of execution that occurs between two checkpoints, within a predetermined amount of time. (2) A safety mechanism should detect a transmission delay between a sender and a receiver whose size is larger than a given threshold.

AUTOSAR safety mechanisms	Software faults in ISO 26262																
	Timing and execution fault					Mem. fault		Exchange of information fault									
	a	b	c	d	e	a	B	a	b	c	d	e	f	g	h	i	j
Program flow monitoring			√	√	√												
End-to-end protection								√	√		√	√	√	√	√	√	√
Memory protection						√	√										

Table 2: Relationship between AUTOSAR safety mechanisms and the software faults

SAFETY MECHANISMS FOR ISO 26262 FUNCTIONAL SAFETY REQUIREMENTS

In order to solve the problems stated above, we present two safety mechanisms for AUTOSAR: (1) a deadline supervision mechanism to detect blocking of execution within a predetermined amount of time and (2) an end-to-end protection mechanism that can detect the delayed transmission of data.

The deadline supervision mechanism monitors program execution flows and detects a deadline miss between two checkpoints in a program. The two checkpoints are called a start checkpoint and an end checkpoint, respectively. When a supervised entity reaches a start checkpoint, the watchdog manager invokes a function named WdgM_ChckpointReached(). This function records the time when the start checkpoint is reached. Then, the watchdog manger periodically computes how much time has passed since the control passed the start checkpoint. When the computed time exceeds a given deadline, the watchdog manger returns an error. Then a user-defined callback function is called to handle the fault.

In order to detect the delayed transmission of data, the end-to-end protection mechanism computes the time difference between sent time and received time. To do so, we modify the current E2E header of AUTOSAR. Specifically, we add two additional fields: sent time and delay threshold. A sender attaches an E/E header to the sending data and sends it to a receiver. When the receiver receives the data, it computes how much time has passed by subtracting the sent time stored in the header from the current time. It then compares the computed time with the threshold. An error is returned if the computed time is larger than the threshold. A user-defined callback function is invoked to handle the fault when an error occurs.

EXPERIMENTAL EVALUATION

We implemented our solution approach into AUTOSAR 3.1 on TriCore™ Starter Kit. We performed a series of experiments on the target system. Its detailed hardware and software specification is given in Table 3.

In one experiment, we created a software component in that a supervised entity was monitored by a watchdog manager. It had two checkpoints: a start checkpoint and an end checkpoint. We ran the supervised entity and blocked its execution between the start checkpoint and the end checkpoint. We then measured how much time the watchdog manager took for detecting such blocking. Our deadline supervision mechanism detected blocking of execution within a time limit.

Hardware (TriCore™ Starter Kit)	MCU	TC1797 (TriCoreTM v1.3.1 CPU 180MHz)
	Memory	4MB flash memory
	Communication	2 CAN, 2 FlexRay transceivers
Software	AUTOSAR	Release 3.1
	SWC modeling	SystemDesk 3.0
	ECU configuration	EBtresos 10.0
	Downloader	TASK VX-Toolset for Tricore v3.4r1

Table 3: Target system description

In the other experiment, we used two TriCore™ Starter Kit boards. A sender software component was placed on one board and a receiver software component was placed on the other. While the sender software component transmitted data to the receiver software component, we delayed the transmission. We checked whether our end-to-end protection mechanism could detect the delayed transmission. It successfully detected all delayed transmissions.

CONCLUSION

In this paper, we proposed two safety mechanisms for AUTOSAR in order to detect all three types of software faults listed in ISO 26262. First, we introduced an enhanced deadline supervision mechanism that could detect the indefinite blocking of a task. It periodically monitors how much time has passed since the start checkpoint and immediately returns an error when the deadline is violated. Second, we proposed an end-to-end protection mechanism that could detect the delayed transmission of data. The time difference between sent time and received time is computed for every message reception and an error is returned when delayed transmission occurs. We implemented our solution on TriCore™ Starter Kit and demonstrated that the proposed solution successfully detected faults as desired.

REFERENCES

[1] ISO 26262 Road Vehicles – Functional Safety, 2011.
[2] Specifications of AUTOSAR Release 4.2, 2015.
[3] Manish Kumar, Jonghun Yoo and Seongsoo Hong. Enhancing AUTOSAR Methodology to a COTS-based Development Process via Mapping to V-Model. IEEE Symposium on Industrial Embedded Systems. 2009.
[4] W. W. Peterson and D. T. Brown. Cyclic Codes for Error Detection. Proceedings of the IRE. 1961.