Development of Data Integrity Testing Tool for Mission-Critical Systems

Bup-Ki Min, Yong Jun Park,Yongjin Seo and Hyeon Soo Kim*
Dept. of Computer Sc. & Eng. Chungnam National University, Daejeon, Republic of Korea
*Corresponding author

***Abstract*—Mission-critical systems are hard real-time systems that collect large amounts of information from diverse sensors for mission performance. For perfect performance of missions of systems, data should not be omitted or spoiled. To this end, data integrity in the systems should be ensured. Therefore, along with a strategy to test data integrity in mission-critical systems, this paper presents a method to test synchronization mechanisms of the shared memory that receives data from sensors and transmits the data to tasks.**

Keywords-data integrity; mission-ciriticalsystem; shared memory test

I. INTRODUCTION

Mission-critical systems collect large amounts of information from diverse sensors in order to perfectly perform missions and have a time constraint as the collected information should be accurately processed at accurate time points[1]. To this end, the collected data should not be omitted or spoiled and the data that have been collected the most recently should be used. These characteristics of data are defined as data integrity in mission-critical systems. If data integrity is not ensured, inaccurate data may be provided to the task so that unexpected errors may occur and consequently the mission cannot be successfully accomplished. This paper presents a test strategy to check data integrity in mission-critical systems and proposes a test environment for the testing.

II. RELATED STUDIES

Because situations where the data integrity of mission-critical systems is not ensured are closely related to the data read/write operation of shared memories, studies for testing shared memories are reviewed.

In a study conducted by [2], shared memories are tested using the SPIN which is a model checking tool. This is a method that steadily increases the processing speed and memory size through modeling while testing all possible conditions(shared memory occupation, data throughput, and data omission, etc.) and finding the optimum test coverage. In a study conducted by [3], a method of creating test sequences and test cases to use shared resources among diverse hardware units using a Real-Time Extended Finite State Machine was proposed as well as a method of automating the tests using the Time-Constrained Transition Equivalence Class method.

A study conducted by[4] shows a test coverage metric targeting simultaneous programs that use shared memories. The objectives of this metric are measuring how programs that access a shared memory simultaneously should be tested from the viewpoint of concurrency control and guiding to make a concurrency control scenario that would not be found through testing.

III. TEST STRATEGY TO ENSURE DATA INTEGRITY

A. Ensuring data Integrity in Mission-Ciritical Systems

To ensure data integrity in mission-critical systems, the following two items should be checked without fail[5, 6].

- Whether the synchronizing mechanism for shared memory management is accurate
- Whether the cycle of the sensor(writer) that writes data in the shared memory and the cycle of the task(reader) to read data from the shared memory have been set appropriately

To test these two items, whether the ISR(Interrupt Service Routine) and the shared memory synchronizing mechanism that collects sensor data and transmits the data to the task operate properly should be checked. In mission-critical systems, as shown in Fig 1, information is collected from sensors and an interrupt is caused to the CPU so that the information is processed through the ISR. In this process, the ISR stores the data in the shared memory through the synchronizing mechanism. When the data have been stored, the task reads the data necessary for execution and processes the data. Since data integrity violations occur due to the operation errors of the shared memory synchronizing mechanism and the ISR, to ensure data integrity, this study focused on testing whether the shared memory and the ISR operate accurately.

The simplest one among methods of conducting tests in situations as shown in Figure 1 is the random test method. That is, creating sensors randomly, having individual sensors randomly write data into the shared memory while having tasks randomly access the shared memory to read data. In other words, artificially making data race situations and testing whether the SUT(System Under Test) operates accurately in the situations. Although the random test has its own advantages, it cannot be easily applied in parallel system environments as shown in Figure 1 for the following reasons. First, if tests are randomly conducted, the possibility for data race situations to occur accidently will be high. Second, if

random tests are conducted, reproducing the same situations will be difficult. General test processes are correcting errors if errors are found while a test is conducted and the test is conducted again to observe whether the errors were resolved. As such, designing tests so that they can be reproduced while being conducted is more important than anything else.

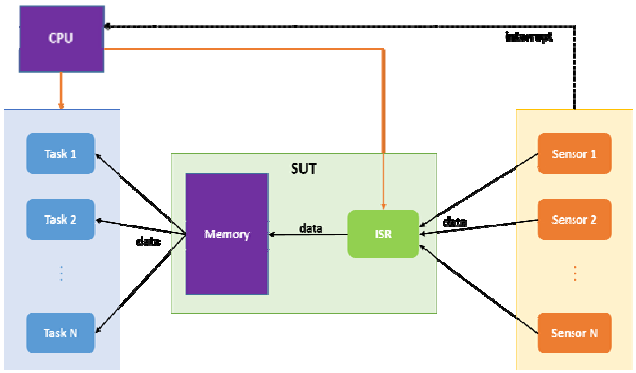

FIGURE I. EXECUTION SCENARIO OF ISR AND SHARED MEMORY IN THE MISSION-CRITICAL SYSTEMS.

B. *Test Starategy to Ensure Data Integrity*

1) Verigication of the accuracy of the shared memory synchronizing mechanism: To ensure data integrity in data race situations, the SUT prepares countermeasures such as double buffering. In terms of tests, whether such countermeasures of the SUT accurately operate and data integrity is actually ensured should be checked. Since the SUT can be implemented in diverse methods, the test should be conducted regarding the SUT as a black box.

2) Verification of the appropriateness of the set cycles of the sensor and the task: If the system has been designed so that data are transmitted between the sensor and the task through the shared memory, the cycles of the sensor and the task should be set appropriately to be able to ensure data integrity while avoiding data races. In terms of tests, whether the given set cycles ensure data integrity should be checked.

3) Test case design strategy: In the process of test case design, controllability which is an important characteristic of testing should be considered. Controllability is a characteristic that refers to the ability to control desired elements in the process of tests or test case design to test targeted matters. To this end, the following methods are used.

*a) Setting sensor data generation cycles:*Settingdata generation cycle is setting simulated sensors to be capable of generating data periodically as with actual sensors. The kinds of tests that can be conducted through this are as follows. Tests to increase the probability of the occurrence of data omission can be conducted by generating data faster than the speed of data storage in the shared memory while adjusting data generation cycles. In addition, tests to violate read and write operation schedule information by having the sensor generate new sensor data before a task that is interested in certain sensors data reads the data. Furthermore, tests to adjust the time point of accessing the shared memory for write operations and the time point of accessing the shared memory for read operations so that the time points overlap each other or become close to each other can be conducted.

*b) Setting task's data acquisition cycle:*Data acquisition cycle setting is setting simulated tasks to periodically read data from the shared memory as with actual tasks. The kinds of tests that can be conducted through this are making conditions under which data omission can occur or causing violations of read write operation schedule information by adjusting the task's data read operation cycles to overlap with the sensor's data write cycles.

*c) Setting the numbers of sensors and tasks:*This is a method that sets the numbers of sensors and tasks. The kind of tests that can be conducted through this is adjusting the numbers of sensors and tasks so that large numbers of sensors and tasks access the shared memory to increase the possibility of data races.

4) Test success identification method: This can be realized through observability which is another characteristic of tests. This characteristic is a concept that makes the results of test conducted are propagated to the time point intended to be observed by us.

a) Identification of data omission and spoilage: Whether generated sensors data have been omitted or spoiled in the process of being written in the shared memory can be identified by comparing the log of the generated data and the log of the data read by the task.

b) Application of data write and read patterns: If data write/read are performed in the normal patterns according to the set cycles of the sensor and the task, data at the same time point will be normally used so that data integrity can be ensured. For instance, in cases where $W_x$is assumed to mean the data write operation of sensors x, $R_y$is assumed to mean the data read operation of task y, and the two tasks 1 and 2 operate based on one sensor a, if a pattern in the form of $W_aR_1R_2$is repeated, read and write should have been done in accordance with the normal schedule.

In general, test case designs intended to test situations where sensors and tasks operate in non- deterministic for the shared memory are known as NP-complete problems. However, since the operations that access the shared memory operate based on schedule information in this study, they correspond to situations where sensors and tasks operate deterministic. Therefore, test cases intended to ensure data integrity can be designed by setting simulated sensors' data generation cycles and simulated tasks' data acquisition cycles.

IV. TEST ENVIRONMENT FOR DATA INTEGRITY VERIFICATION

For tests of the accuracy of the ISR and the shared memory synchronizing mechanism and the appropriateness of the set cycles of sensors and tasks, the test environment as shown is Fig 2 is proposed. The method proposed here is creating simulated sensors and simulated tasks to test the

SUT which is actually implemented software. In this case the test is conducted regarding the SUT as a black box.

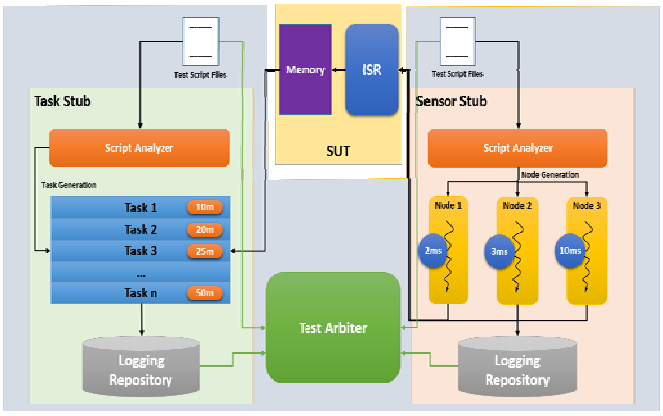

FIGURE II. ARCHITECTURE OF A DATA INTEGRITY TESTING TOOL.

Here, the sensor stub and the task stub simulate the sensors and tasks in actual environments. Since the internal logic of the SUT can vary depending on the implementation, we can verify whether the SUT operates accurately by manipulating the sensor stub and the task stub. Each sensor stub and task stub are automatically created through test script files and the test script files are prepared by a tester in accordance with the test strategy described in section 3.2. That is, the tester enters sensor and task information so that the contents desired to be identified (data omission, whether data at the same time point are used, and the appropriateness of the set cycle) can be verified. Fig.3 shows examples of sensor stub scripts and task stub scripts.

```
<SensorStubScript>
        <Sensor name="Sensor1">
                <Period value="5"/>
                <DataStructure type="1">
                        <Data name="temp" Min="-50" Max="100"/>
                </DataStructure>
        </Sensor>
        <Sensor name="Sensor2">
                <Period value="10"/>
                <DataStructure type="3">
                        <Data name="x" Min="000000000" Max="999999999"/>
                        <Data name="y" Min="000000000" Max="999999999"/>
                        <Data name="z" Min="000000000" Max="999999999"/>
                </DataStructure>
        </Sensor>
        <Sensor name="Sensor3">
                <Period value="20"/>
                <DataStructure type="3">
                        <Data name="x" Min="000000000" Max="999999999"/>
                        <Data name="y" Min="000000000" Max="999999999"/>
                        <Data name="z" Min="000000000" Max="999999999"/>
                </DataStructure>
        </Sensor>
```

```
<TaskStubScript>
        <Task name="Task1">
                <Period value="5"/>
                <UsingSensor>
                        <SensorData name="SensorData1" ownerID="Sensor1"/>
                </UsingSensor>
        </Task>
        <Task name="Task2">
                <Period value="20"/>
                <UsingSensor>
                        <SensorData name="SensorData2" ownerID="Sensor2"/>
                        <SensorData name="SensorData4" ownerID="Sensor4"/>
                </UsingSensor>
        </Task>
        <Task name="Task3">
                <Period value="30"/>
                <UsingSensor>
                        <SensorData name="SensorData2" ownerID="Sensor2"/>
                        <SensorData name="SensorData4" ownerID="Sensor5"/>
                </UsingSensor>
        </Task>
```

FIGURE III. STRUCTURE OF STUB SCRIPT.

Sensor test scripts describe information on the number simulated sensors and the data generation cycles of individual sensors and task test scripts contain information on the number of simulated tasks and the data acquisition cycles of individual tasks. Based on such information, threads for simulated sensors and simulated tasks are created. Test environments that simulate actual operation environments can be constructed by setting simulated sensors to generate sensors data in accordance with the set generation cycle and transmit the data to the SUT and setting simulated tasks to read data in accordance with the set acquisition cycle to perform works. The results of works of individual threads are stored in the log storage. The Test Arbiter analyzes test script files to identify items that must be verified and automatically makes judgment on data integrity utilizing the data stored in the log storage.

V. EXAMPLE OF DATA INTEGRITY TEST RESULTS

To conduct the test, a test environment as shown in Table 1 was set. Individual sensors and tasks generate data or perform works with their own cycle. To check the ability of the test tool, we intentionally implemented the SUT so that some errors would occur when the SUT operates.

TABLE I. EXAMPLE OF SETTING UP A TEST ENVIRONMENT.

Task Stub					Sensor Stub	
Task	Period	Dependent Sensors			Sensor	Period
T1	5ms	S1	-	-	S1	5ms
T2	20ms	S2	S4	-	S2	10ms
T3	30ms	S2	S5	-	S3	20ms

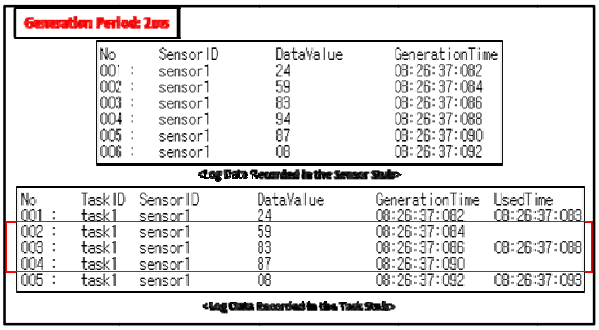

FIGURE IV. EXAMPLE OF LOG DATA.

Due to the restriction of papers, only the results of tests for data omission are shown here. To check data omission, we adjusted data generation cycles. Since the cycle of sensors S1 is 5ms, if task T1 is implemented at 5ms cycles, no omission will occur because the cycles of the task and the sensor are the same. However, if the sensor's data generation cycle is set to 2ms, the time point of data generation by the sensor and the time point of data reading by the task can collide with each other. Consequently, the fact that the data in the red box area in the Figure was omitted can be identified.

VI. Conclusion

In the case of mission-critical systems, real-time, schedulability, and data integrity should be ensured. Among them, we defined the data integrity of mission-critical systems, established a test strategy to test the systems, and implemented a test environment for the test. The simulated sensors and tasks were designed to be expandable and flexible so that they can be constructed to fit actual environments.

Acknowledgment

This research was supported by Agency for Defense Development (Contract No. UD130043CD).

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-R0992-15-1014) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

[1] S. Ostendorff, et. al., “Modeling Timing Constraints for Automatic Generation of Embedded Test Instruments", IEEE 17th Int’l Symp. on Design and Diagnostics of Electronic Circuits and Systems, 2014, pp.201-206

[2] G. J. Holzmann., R. Joshi, A. Groce, “Swarm Verification Technique", IEEE Trans. on Software Engineering, Vol..37, No.6, 2011, pp. 845-857

[3] Y. Yongfeng, “Real-time embedded software testing method based on extended finite state machine", Journal of Systems Engineering and Electronics, Vol.23, Issue.2, 2012, pp.276-285

[4] S. Tasiran, M.E. Keremoglu, K. Muslu, “Location pairs: a test coverage metric for shared-memory concurrent programs", Empirical Software Engineering, Vol.17, 2012, pp.129-165

[5] H. Kopetz, “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Springer, 2011

[6] R. I. Davis, A. Burns, “A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", ACM Computing Surveys, Vol.43, No.4, 2011. pp.35:1-35:44