Fault detection in safety-critical embedded systems

DOMEN VERBER[1], MATJAŽ COLNARIČ[1], AND WOLFGANG A. HALANG[2]
[1]University of Maribor, Faculty of Electrical Engineering and Computer Science, 2000 Maribor, Slovenia; [2]FernUniversität Hagen, Faculty of Electrical Engineering, 58084 Hagen, Germany

Abstract: In the paper, a proposition for a systematic approach to fault detection in building a dependable and fault-tolerant control system is presented. A network of simple monitoring cells that monitor and evaluate functioning of critical sub-processes of the system is proposed. Further, different approaches for the implementation of the monitoring cells are observed.

Key words: Safety-critical embedded systems, fault-tolerant control systems, fault detection, fault localization and isolation

1. INTRODUCTION

A typical present day control system consists of physical process components, sensors, actuators, distributed computers with communication networks, and several thousands of lines of code. Examples of these are control systems in industrial plants, nuclear reactors, avionics, etc. The size and the complexity of such systems increase every year and so does the probability of a fault in one of the many components. Each fault in such a system can cause severe material loss or even endanger human lives.

Most of the researchers working on the problem of fault-tolerant computing are only providing partial solutions (e.g. [1,2,3]). However, for appropriate consideration of all problems, a holistic approach is necessary.

The work presented in the paper is a part of the research founded by the Slovenian Ministry of Education, Science and Sport. The primary issue of this research is the study and the development of methods and tools for hierarchically organized fault-tolerant control systems. The proposed model

---

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35599-3_29

B. Kleinjohann et al. (eds.), *Design and Analysis of Distributed Embedded Systems*

should deal with faults in the environment, in the hardware and in the software of the embedded system. Basic research topics covered with the project are fault detection, fault localization and isolation, graceful degradation of the functionality of the system in a case of minor fault, and controlled and safe shut-down of the system in the case of severe defects.

In the article, a practical proposition of how to deal with fault detection of such systems is given. The basic idea is to use a network of simple monitoring cells that monitor and evaluate functioning of critical sub-processes of the system.

2. FAULT DETECTION BY MEANS OF MONITORING CELLS

A typical control system can be usually divided into a set of well-defined sub processes or tasks. Each task takes its inputs from sensors and from the results of other processes, and produces results that are used by other tasks, or influence the controlled system through actuators. Tasks are triggered by synchronous or asynchronous events.

There are several causes of faults in such a model. E.g., the components of the controlling process can breakdown due to hardware or software related errors. For the same reasons, values of input and output signals can be out of their predefined ranges or they do not apply to functional specifications of the system. In hard real-time systems, improper temporal behaviour is also considered as a fault. Another not so obvious source of errors can be temporal inconsistencies of the signals; e.g. the dynamics of changing of signals can be too steep, frequencies of event occurrence can be to high, etc.

Failure in the systems can be handled by redundancy and diversity, with reconfiguration etc. [2], but first the failure in the system must be detected. To do this, some sort of a dependable monitoring subsystem must be used that detects abnormalities in the system and triggers appropriate corrective actions. To lower the complexity, safety related issues of the system should be designed, evaluated and implemented independently and in parallel with the functional part. This is also true for fault detection. To achieve this, a set of monitoring components (we named it monitoring cell - MC) is introduced. In early phases of the development of the system each MC is considered as an abstract object. In later phases, the MCs can be implemented as hardware and/or software components. The basic task of the monitoring cell is to monitor the validity of input and output values of the sub process. A monitoring cells are only parts of larger fault detection and fault prevention system.

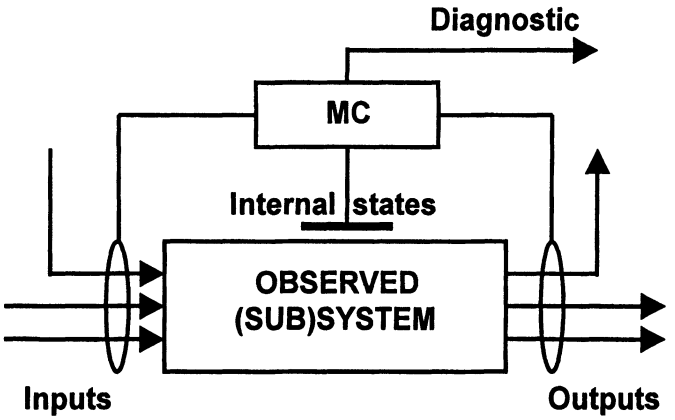

Figure 1. Concept of the Monitoring cell

Monitoring cell should be simple and should produce as little interference with the production environment as possible. It should be built from simple and robust components with low probability of failure. It should work with both custom built and Commercial Off-The-Shelf (COTS) components. It is supposed to be built in such way that it could be formally verified and possibly certified by a certification authority. Even if by this we cannot formally prove that the system will be dependable as a whole, we should prove that the safety-related subsystem is. When MC detects failure in the sub-process it usually does not react on its own. Instead, it signals the source and cause of error to the upper layers in the safety-related subsystem.

The proposed architecture of MC can be divided into four parts: decoder logic, validation logic, state machine and timing logic.

Decoder logic converts observed variables of the system into a form that can be used by monitoring systems. Usually it partitions the definition domain of each input and output value into a set of ranges and assigns to them appropriate discrete values. With this, discrete and continuous values can be mapped into a manageable set of states. Another possibility is that observed values are degraded by means of its accuracy to reduce complexity of the monitoring function (e.g. for A/D conversion of inputs fewer bits are used than that used by observed function). If it is convenient, all relevant internal variables that represent the current context of the task can also be observed in this way.

Decoder logic can also detect if observed values are out of predefined ranges and produce the appropriate diagnostic signals. Further, some physical characteristic of the hardware components can be measured to detect any abnormalities (e.g. measurement of the environment temperature or measurement of electric current consumption). Thus, early correction

actions can be taken by the system even before the controlled function is activated.

Decoder logic also latches all relevant values of the inputs to be used when outputs are produced and/or when a validation of dynamic changes is performed.

Validation logic makes sure that the outputs of the system are consistent with the inputs. In the basic case when input and output values are transcribed into small sets of discrete states, a list of all valid combinations can be enumerated and later observed. In more general cases, validation logic can be understood as a pattern-matching algorithm. It tests of a current combination of (mapped) inputs and outputs is valid. In this case validation logic can be represented as a Boolean function of the observed inputs, outputs, internal states and time. In addition to Boolean output of validation logic, other diagnostic signals can be generated that may give the hints as to what is the cause of the failure.

State machine is an optional component that moderates when evaluation logic should perform its evaluation. There are situations where behaviour of the monitored component changes significantly during the application execution (e.g. when the system can operate in several different modes). Also, if the controlled function is performed in several steps, it may be desired to observe and monitor each individual step separately. By this, inconsistencies in process execution can be detected early enough, and some correction measures can be taken. In a simple scenario, state machine logic can be left out because the evaluation is done only twice: first, when input values are evaluated and second when output values and input/output combinations are tested. This can be triggered by data arrival or by additional signals from controlled sub-process.

Timing logic is an optional component that can be used to validate temporal properties of the process. In the most basic scenario, this logic performs a similar function to a simple watchdog timer. When a task starts, the timer is set to the task's deadline. The timer is reset by the production of outputs. If no output is generated in a predetermined time, MC signals to the fault-tolerant subsystem that temporal requirements are violated. In more complex situations, timing logic is coupled with the state machine and additional checkpoints are added into the process. Again, this allows temporal inconsistencies to be detected early enough to react in time.

Other usage of the timing logic is in the monitoring of dynamic behaviour of the system. For example, by comparing two successive signals

in a given time interval, it can detect if the changes are too steep or if the frequency of arrival of the signals is to high.

3. IMPLEMENTATION OF MC

There are several possible approaches for the implementation of MCs, and when mapping is simple enough, its implementation can be automated and/or emulated in the early stages of development.

In simple cases software solutions can be applied. With this, each MC can be implemented as a set of monitoring routines that runs together with production software. Simple mapping tables or decision trees can be used by software implementation. These routines can be integrated into the operating system. In this case evaluation routines are divided into two parts. The first set of routines is called after inputs are acquired and before the code of the main function is called. It performs preparation and validation of inputs. After the code of the task is executed and before outputs are produced, another set of routines is called that evaluates outputs and input-output combinations.

The next example shows how these routines can be integrated into the code. Each input, output and other parts of the algorithm are associated with a constant that identifies them.

```
functon PerformTask:
AcquireInputs(i1,i2)
if not MCEvaluateInputInt(INPUT1_ID,i1) then
   HandleException
if not MCEvaluateInputFloat(INPUT2_ID,i2) then
  HandleException
MCSetDeadline(FUNCTION1_ID)
Compute(i1,i2,o1)
if not MCEvaluateOutputInt(OUTPUT1_ID,o1) then
  HandleException
if not MCEvaluateResults(FUNCTION1_ID) then
  HandleException
ProduceOutputs(o1)
```

A higher degree of dependability and agility can be achieved by using dedicated hardware solutions. In most cases they can be implemented with

simple discrete logic (i.e. FPGA): decoder logic can be implemented with a collection of comparators, validation logic is implemented by simple logical gates; the state machine and the timing logic are implemented by means of flip-flops and simple registers.

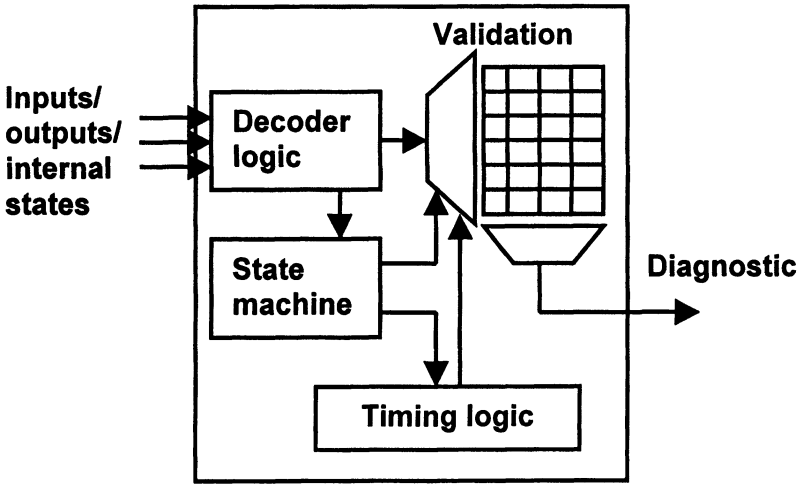

Figure 2. Hardware implementation of the MC

If the requirements are simple enough, automated generation of the VHDL description from the application specification is possible.

If the observed system is very complex or if it behaves in a non-deterministic way, then simple validation logic may not be good enough. In this case more sophisticated methods should be implemented. For example, validation logic can be observed as a pattern classification problem in the neural net paradigm [4], fuzzy functions can be used, etc.

4. OTHER POSSIBLE USE OF MC

There are some other possible uses of the MCs. When mapping functions are simple enough, MC can be implemented (emulated) before the actual process is built. By this, it can be used either as a surrogate of the process or as an additional test bed for then original task.

MC can also be used as a last resort in the situation of primary process failure. Based on the simplified knowledge of the mapping inputs into outputs, it can be used to generate rough output results. In this case, for each valid input combination, the most appropriate output must be noted.

5. CONCLUSION

The paper represents the work in progress. There are several aspects of the problem of fault detection that are not yet adequately resolved. One of the problems is how are the parameters for the MC gathered from specifications of the system. This can be difficult if fault-tolerant issues are not appropriately considered in the development (e.g. definition of pre- and post- conditions of tasks, physical descriptions of inputs and outputs, temporal requirements, etc.).

Appropriate mapping of inputs and outputs into simple states is also a difficult part of using MCs and is still an open issue. Some kind of learning technique may be used with a learning set gathered from mathematical models or testing runs.

To test different approaches to MC implementation several short-term subprojects are started that cover software implementation, hardware implementation and the use of machine learning techniques (neural networks and fuzzy logic).

In the future, other aspects of the development of the fault-tolerant system will be investigated.

REFERENCES

[1] N. Storey, 'Safety-Critical Computer Systems', Addison-Wesley Pub, august 1996.
[2] D. S. Herrmann, 'Software Safety an Reliability',
[3] J. P. Bowen in M. G. Hinchey, 'High-Integrity System Specification and Design', Springer, April 1999.
[4] L.W. Fausett, 'Fundamentals of Neural Networks', Prentice Hall, 2001.