# Co-Design Techniques for Fault-Tolerant Real-Time Systems using Imperfect Fault Detectors

## 1 Introduction

To meet the reliability requirements of safety-critical embedded systems, fault tolerance techniques such as active redundancy are widely adopted. Fault-tolerant system design using active redundancy is a very challenging task that involves solving two major problems, namely finding the optimal utilization of temporal and/or spatial redundancy and the scheduling of tasks (including replicas) under timing constraints. Over the past decades, a lot of research efforts have been devoted to this field.

To cope with the high problem complexity, many state-of-the-art studies make simplifying assumptions on the fault models and modes. Perfect fail-silent behavior is one assumption that is often used in literature. It is assumed that all faults are detected within a certain time interval and that the fault detection overhead is contained in the tasks' Worst-Case Execution Times (WCETs), e.g., in fault-tolerant task scheduling [1, 2, 3, 4, 5], in reliability-aware energy management [6, 7, 8] and in error-aware system design [9, 10]. With this assumption, each task will produce either a correct output or no output at all. Although fail-silence is a highly desirable property, it is difficult to implement in practice. The prerequisite is the existence of a perfect fault detector that achieves 100% coverage under the given fault hypothesis.

In the previous study [11], we have explained the major problems of this assumption. On the one hand, this assumption is very impractical; on the other hand, even if it is implementable, using perfect fault detection is often a suboptimal design decision, due to the fact that good fault detectors usually come with high timing overheads [12, 13]. Actually, when active redundancy is concerned, there is a tradeoff about whether the available resources should be spent on implementing better fault detection or realizing more redundancy. We have developed new analysis and optimization techniques to tackle these issues. Experimental results show that certain designs involving imperfect fault detectors combined with task replication can outperform other designs assuming perfect fault detection.

So far, only software-implemented fault detection is considered. However, as shown in [10], fault detection could also be implemented in hardware to reduce the time overhead, e.g. using on-chip reconfigurable FPGA fabric. This not only contributes to reducing the schedule length but also allows more options for redundancy. Unfortunately, hardware fault detection increases the overall system cost. In particular, the on-chip resources are often not sufficient to implement hardware fault detectors for all tasks. Hence, it is a major design decision to select which fault detector to implement for each task and where to implement them.

In Figure 1 we show an example scenario extended from the motivating example of [11]. Figure 1a depicts the schedule using the perfect detector (it is assumed that perfect fault detection incurs 300% timing overhead). Figure 1b is another possible schedule, in which the task is replicated twice and the remaining time (200% task execution time in this case) is used to implement two partial fault detectors. Figures 1c and 1d show two similar schedules with higher number of replications. Figure 2 compares the reliability of those schedules, in terms of the probability of detectable ($DUF$ in the figure) and undetectable ($SDC$ in the figure) faults. As it can be seen, the design with perfect fault detection can detect all faults. However, only detecting the faults is often not sufficient, e.g. for fail-operational applications. When multiple replicas of the same task are available, we have another mean of fault detection, that is, to compare the output from different instances (voting). Actually, certain faults might even be corrected, e.g. a single faulty input out of three inputs will be masked by voting. In this case, the schedule with partial fault detectors might have higher reliability (see [11]). Figure 1e depicts one schedule that has not been considered so far. In this schedule, the fault detector is implemented in hardware to reduce the timing overhead. This allows us to schedule two instances of the task, both

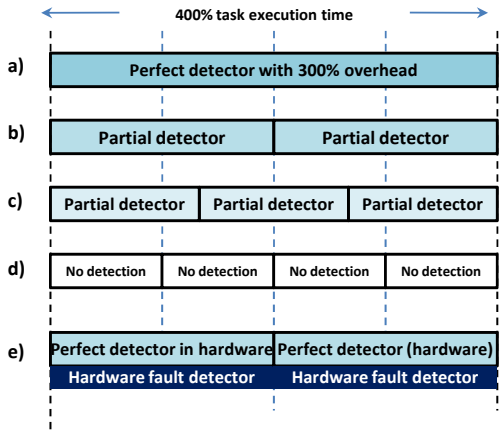


Fig. 1: Example Scenario

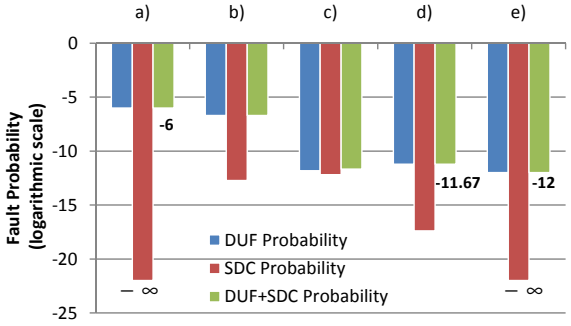


Fig. 2: Reliability of the Example Schedules

using a perfect fault detector. The resulted schedule achieves the highest reliability among the example schedules.

## 2 Problem Statement

In the previous paper [11], the author tried to achieve sound fault detection performance using pure software implementations only. But in reality, the overhead of implementing the fault detectors in software is so high that the system deadline might be violated. Therefore, we want to take hardware-acceleration for fault detection into our consideration in this project. Thus, we can provide good fault detection for the systems that have very tight resource limitations, or we can use the new approach to achieve better fault coverage than pure software-implemented fault detections. We formulate the problem as follows.

### 2.1 Problem input

A system consists of two parts: an execution platform, which is a heterogeneous multiprocessor platform and an application modeled as an acyclic task graph. The application has to be mapped to the execution platform. Besides each processor, there exists an FPGA co-processor sharing the same memory space as the processor. Memory access time and overhead of the processor and FPGA are ignored in this study.

We have a library of implementable fault detectors for each task (both software and FPGA implementations) characterized with detection coverage and overhead. The latter has two aspects, time overhead and hardware (FPGA area) overhead. In the software-implemented versions, the area overhead is 0.

### 2.2 Two optimization directions

#### 2.2.1 Problem 1

**Constraints**

1. The global deadline constraint of the application. The application has to finish its execution before the global deadline $D$.
2. The total hardware budget (FPGA area) is limited by a fixed total amount $A$. The total amount of on-chip reconfigurable fabric should not be more than this constraint.

**Optimization objectives** We want to maximize the reliability of the system and at the same time minimize the length of the schedule.

#### 2.2.2 Problem 2

**Constraints**

1. The global deadline constraint of the application. The application has to finish its execution before the global deadline $D$.
2. The reliability of the whole system, given as a percentage $R$ (e.g. 99.99999%), specifying the probability that the application completes successfully.

**Optimization objectives** Minimize the total hardware area added into the system for implementing the error detectors and at the same time minimize the length of the schedule.

### 2.3 What is a solution

1. The number of replications for each task;
2. The mapping and schedule of the application, including all task replicas;
3. The selection of fault detectors for each task, considering both overhead and coverage and their implementation method, i.e. in software or hardware.

## 3 Proposed Techniques

## 4 Experimental Results

## 5 Conclusion

## References

[1] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems," in *DATE*, 2005.

[2] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Trans. VLSI*, 2009.

[3] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *Dependable and Secure Computing, IEEE Transactions on*, 2009.

[4] P. K. Saraswat, P. Pop, and J. Madsen, "Task mapping and bandwidth reservation for mixed hard/-soft fault-tolerant embedded systems," in *RTAS*, 2010.

[5] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *CODES+ISSS*, Oct 2011.

[6] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *CODES+ISSS*, 2007.

[7] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *ICCAD*, 2009.

[8] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Trans. Computers*, 2009.

[9] J. Lee, I. Shin, and A. Easwaran, "Online robust optimization framework for qos guarantees in distributed soft real-time systems," in *EMSOFT*, 2010.

[10] A. Lifa, P. Eles, Z. Peng, and V. Izosimov, "Hardware/software optimization of error detection implementation for real-time embedded systems," in *CODES+ISSS*, 2010.

[11] J. Huang, K. Huang, A. Raabe, C. Buckl, and A. Knoll, "Towards fault-tolerant embedded systems with imperfect fault detection," in *49th Design Automation Conference (DAC)*, San Francisco, CA, USA, June 2012.

[12] G. Lyle, S. Chen, K. Pattabiraman, Z. Kalbarczyk, and R. Iyer, "An end-to-end approach for the automatic derivation of application-aware error detectors," in *DSN*, 2010.

[13] U. Schiffel, A. Schmitt, M. Süßkraut, and C. Fetzer, "Software-implemented hardware error detection: Costs and gains," in *Third International Conference on Dependability*, 2010.