Microprocessors and Microsystems 38 (2014) 539–551

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A framework for reliability-aware embedded system design on multiprocessor platforms

Jia Huang [a,*], Simon Barner [a], Andreas Raabe [a], Christian Buckl [a], Alois Knoll [b]

[a] fortiss GmbH, Guerickestr. 25, 80805 München, Germany
[b] TU München, Boltzmannstr. 3, 85748 Garching bei München, Germany

ARTICLE INFO

Article history:
Received 5 July 2013
Revised 30 October 2013
Accepted 18 February 2014
Available online 12 March 2014

Keywords:
Embedded systems
Reliability
Fault-tolerance
Design optimization
Model-driven development

ABSTRACT

This paper presents a model-driven framework that provides a tool-supported design flow for fault-tolerant embedded systems. Its system models comprise abstract descriptions of the application and the underlying execution platform. They provide the input to our analysis and optimization techniques that enable the automated exploration of design alternatives for applications with reliability requirements. The automated generation of source code and platform configuration files speeds up the development process. Our contribution is to advance reliability-aware design further into practice by providing an integrated tool framework and removing unrealistic assumptions in the analyzes. The case studies demonstrate the effectiveness of our approach.

1. Introduction

Reliability is becoming one of the most important concerns in today's embedded systems. However, as technology scales, modern devices are more susceptible to faults [5]. Such hardware faults could be permanent (hard errors) or transient (soft errors). Despite the efforts of the hardware community to enhance the hardware reliability, there is an increasing need to use system-level Fault-Tolerance Mechanisms (FTMs) to mitigate the impact of such faults.

Fault-tolerant embedded system design involves several challenging problems. First of all, the designer needs to reason about the system properties in the presence of FTMs to check if all requirements are met. Respective analysis techniques, such as reliability and timing analyses are needed. Second, since FTMs typically come with high overhead, it is critical to find the optimal design under given constraints. Therefore, a Design Space Exploration (DSE) problem arises. It is important to note that the configuration of FTMs must be considered jointly with other design parameters due to their correlation. In particular, the amount of redundancy highly influences the schedulability of the application. For multiprocessor systems, FTM configuration has to be considered together with the classical task mapping and scheduling problem. Finally, implementing the selected design on complex platforms such as Multiprocessor System-on-Chips (MPSoCs) is also challenging. Here, the designer faces both complex and labor-intensive problems such as multiprocessor programming, inter-core communication and the configuration of the execution platform. Therefore, tool support is highly desirable in order to reduce the design cost and the time-to-market.

Over the past decades, a lot of research effort has been devoted to the aforementioned challenges in the design of reliable systems. However, there is still a gap between theory and practice. On the one hand, due to the high complexity of the problem, the theoretical studies are typically based on certain assumptions and the according simplified system models. For instance, in many studies, it is assumed that any transient faults are detected at the end of the task using certain fault detectors so that all tasks have perfect fail-silent behavior [16,29,8,30]. Although a lot of important results have been achieved by studying this simplified version, such unrealistic assumptions limit the practical usability of the proposed techniques. On the other hand, most of the work focuses only on a single part of the overall problem and little effort is spent on integrated approaches providing tool support for the entire design process. In particular, current work mostly focuses on the "front-end" of the process, namely calculating the mapping/schedule under reliability constraints. The challenge of the "back-end", *i.e.* implementing the calculated schedule on a complex hardware platform, is underestimated.

This paper presents a model-driven development (MDD) framework to tackle the aforementioned problems. MDD is a well-known

* Corresponding author.
E-mail addresses: huang@fortiss.org (J. Huang), barner@fortiss.org (S. Barner), raabe@fortiss.org (A. Raabe), buckl@fortiss.org (C. Buckl), knoll@in.tum.de (A. Knoll).

http://dx.doi.org/10.1016/j.micpro.2014.02.007
0141-9331/

approach to cope with the rising complexity of embedded system design [26]. Our framework features modeling, analysis, optimization, and code generation tools in order to provide a complete integrated reliability-aware design flow. Fig. 1 compares the traditional development approach for reliable systems (left) to the design flow supported by our framework (right).

In the traditional approach, the designer extracts a scheduling model from the system specification in order to apply the reliability-aware scheduling algorithms. These algorithms may operate on different models (periodic task sets, task graphs, *etc.*) and the designer has to ensure the consistency. In parallel, the source code of the application is developed manually, including both functional code for the application and platform-specific structural code. Finally, the scheduling results and the source code are combined to an executable image and the platform configuration. The advantages of the proposed flow stem from the fact that models are first class citizens in MDD. They serve as central integration point for subsequent tasks such as analysis, DSE and code/configuration generation. Models speed up the development process and ensure the overall consistency by raising the level of abstraction and automation.

Our approach is based on the following:

- Platform-independent description of the application including timing and reliability requirements.
- Fine-grained platform model covering both hardware platform and system software (HW/SW stack).
- Multi-criteria design space exploration: mapping and scheduling w.r.t. timing and reliability constraints (consideration of permanent and transient faults).
- Automatic insertion of fault-tolerance techniques, including active redundancy, voting and fault detection to meet user-specified reliability goals.
- Generation of implementation artifacts such as application C source code and platform configuration files.

The remainder of the paper is organized as follows. The system models are presented in Section 3. The main contribution of this work, namely the reliability-aware design flow is presented in detail in Section 4. A case study is discussed in Section 5. Finally, Section 6 concludes this paper.

2. Related work

2.1. Fault-tolerant embedded system design

Using system-level FTMs has been addressed in a number of research studies. The approach presented in [36] handles transient faults by selectively inserting task re-executions. Girault and Kalla [8] consider fault-tolerant scheduling with active task replications and present a bi-criteria heuristic algorithm. Izosimov et al. [16] combine spatial and temporal redundancy and propose novel techniques to share the re-execution slack among multiple tasks. In the follow-up work [15], a more accurate probabilistic analysis is presented and hardware hardening is considered. Stralen and Pimentel present a DSE based approach for fault-tolerant deployment of applications on MPSoCs [33]. The FTMs are described as patterns that are applied to the application model. Only spatial redundancy patterns have been considered so far, *i.e., dual* and *triple modular redundancy* (DMR/TMR).

A recent work that is close to our approach is from Bolchini and Miele [4]. They also propose a generic DSE framework that supports a configurable set of FTMs, such as active redundancy, fault detection and voting. Moreover, they also synthesize time-triggered fault-tolerant schedules using genetic algorithms. One major difference between their work and ours is the fault model. They adopt a similar fault model as Izosimov et al. [16] and aim at handling a maximum number of concurrent faults. The reliability of the execution platform is modeled using a simple *qualitative* tag (*e.g.*, if the processor supports fault detection or fault-tolerance). Only coarse evaluation of the system reliability can be provided in this case. In contrast, our probabilistic reliability analysis provides precise *quantitative* results to guide the optimization process.

Tables 1 and 2 provide a qualitative comparison of representative related work. In Table 1, we first summarize the fault model utilized by the individual approaches. Some early work in the field [36,19] considers a single-fault model. This is a reasonable

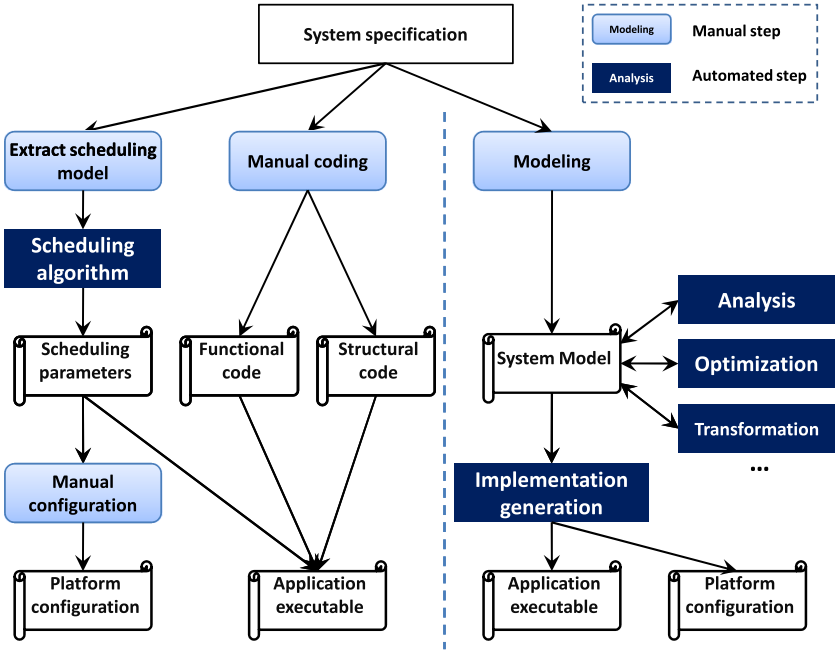

Fig. 1. Comparison of traditional and MDD flow.