Reliable and Precise WCET and Stack Size Determination for a Real-life Embedded Application

Philippe Baufreton*, Reinhold Heckmann**

*Hispano-Suiza, Etablissement de Réau BP 42
F-77551 Moissy-Cramayel Cédex, France
philippe.baufreton@hispano-suiza-sa.com
http://www.hispano-suiza-sa.com/
**AbsInt Angewandte Informatik GmbH
Science Park 1, D-66123 Saarbrücken, Germany
heckmann@absint.com
http://www.absint.com/

Abstract. Failure of a safety-critical application on an embedded processor can lead to severe damage or even loss of life. Here we are concerned with two kinds of failure: stack overflow, which usually leads to run-time errors that are difficult to diagnose, and failure to meet deadlines, which is catastrophic for systems with hard real-time characteristics. Classical software validation methods like simulation and testing with debugging require a lot of effort, are expensive, and do not really help in proving the absence of such errors.

AbsInt's tools **StackAnalyzer** and **aiT** (timing analyzer) provide a solution to these problems. They use abstract interpretation as a formal method that leads to statements valid for all program runs. Both tools have been used successfully at **Hispano-Suiza** to analyze applications running on a Motorola PowerPC MPC555. They turned out to be well-suited for analyzing large safety-critical applications developed at **Hispano-Suiza**. They can be used either during the development phase providing information about stack usage and runtime behavior well in advance of any run of the analyzed application, or during the validation phase for acceptance tests prior to the certification review.

1 Introduction

Failure of a safety-critical application on an embedded processor can lead to severe damage or even loss of life. Therefore, utmost carefulness and state-of-the-art machinery have to be applied to make sure that an application meets all requirements. Classical software validation methods like simulation and testing with debugging require a lot of effort and are very expensive. Furthermore, they cannot really guarantee the absence of errors. In contrast, *abstract interpretation* (Cousot and Cousot, 1977) is a formal verification method that yields statements valid for all program runs with all inputs, e.g., absence of violations of timing or space constraints, or absence of runtime errors.