A Hardware/Software Codesign Methodology and Workbench for Predictable Development of Hard Real-Time Systems

Jakob Axelsson*
Linköping University, SWEDEN

Abstract

The prime goal for the implementation of a real-time system is to guarantee that it fulfills its timing constraints. The response times which are limited by the constraints depend on what hardware is used to implement the system, and often violations of the constraints are discovered late in the design process, leading to major design revisions, thus making the development unpredictable. This is because different subsystems are developed quite independently, and it therefore becomes hard to validate the system-level timing constraints before the integration phase. This paper presents a hardware/software codesign methodology that addresses these issues by taking a unified view on the system description. It also describes a workbench of supporting tools.

1 Methodology

The main problem that needs to be addressed is that of monitoring timing constraints across technology boundaries and during different stages of development. We assume that the development starts with a specification of the system’s intended behaviour, expressed as a set of tasks whose response times are constrained by deadlines. There is a timing validation procedure, based on schedulability analysis, which will be applied to the final system, and which provides conservative bounds on the response times of the tasks. It is parametrized by values indicating resource usage and execution time for the different tasks.

During the development, the resource needs of the tasks are determined using estimators, which, given a component such as a microprocessor and a behavioural description of a task, calculates the expected execution time, number of memory accesses, etc., for each activation of the task. It also gives indications about the size of the implementation, which is important when tasks are implemented in ASICs.

Although such estimations may become quite inaccurate, they can be used constructively to make the design process more predictable. Consider the case where the timing validation procedure, when given estimated parameters, says that the constraints will be met. This indicates that if a final implementation can be found, which behaves according to the estimation, then this implementation will meet the deadlines. The estimated values can thus be used as timing constraints during later development steps, and we refer to these new constraints as resource budgets. The resource budgets localize the global timing constraints to each individual component, and are thus easier to check.

2 Workbench

During the early phases of the development, it is necessary to do a thorough design space exploration to find the most cost-effective solution which is likely to meet the constraints. We have developed a workbench of tools that can automate this exploration to a certain extent. In the workbench, the current design is captured in a data structure called a virtual prototype (VP). It contains information about the system’s behaviour (the task set), architecture (the hardware components and their interconnections), partitioning (how tasks are allocated in processing units), and schedule (defined by a fixed-priority relation). The components of the architecture are selected from a component library which also contains data for the estimation models.

The VPs are iteratively modified using a set of transformations, which can: move tasks between processing units; add, remove, or reconnect hardware components; and change the scheduling priority of tasks. The tools manipulate the VPs solely through such transformations, and at the time of writing, the workbench contains the following tools:

Architecture synthesis. This tool selects a number of components from the component library and connects them to form an architecture. Simultaneously, it also performs a partitioning, which is necessary to be able to evaluate the result. The tool is based on tabu search [2].

Partitioning. The architecture synthesis tool gives good but not optimal partitionings. Therefore, we have also developed an algorithm based on branch-and-bound search, which finds the optimal partitioning [1].

Scheduling. An optimal priority assignment algorithm [1] has been implemented.

References


*Email: jakax@ida.liu.se