

Methods and Tools for Calculating the Flexibility of Automotive HW/SW Architectures

Liangpeng Guo
Univ of California-Berkeley

Arkadeb Ghosal
National Instruments

Haibo Zeng
General Motors Company

Paolo Giusto
General Motors

Alberto Sangiovanni-Vincentelli
Univ of California-Berkeley

ABSTRACT

To cope with the increasing number of advanced features (e.g., smart-phone integration and side-blind zone alert.) being deployed in vehicles, automotive manufacturers are designing flexible hardware architectures which can accommodate increasing feature content with as fewer as possible hardware changes so as to keep future costs down. In this paper, we propose a formal and quantitative definition of flexibility, a related methodology and a tool flow aimed at maximizing the flexibility of an automotive hardware architecture with respect to the features that are of greater importance to the designer. We define flexibility as the ability of an architecture to accommodate *future changes in features with no changes in hardware* (no addition/replacement of processors, buses, or memories). We utilize an optimization framework based on mixed integer linear programming (MILP) which computes the flexibility of the architecture while guaranteeing performance and safety requirements.

CITATION: Guo, L., Ghosal, A., Zeng, H., Giusto, P. et al., "Methods and Tools for Calculating the Flexibility of Automotive HW/SW Architectures," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 5(1):2012, doi: 10.4271/2012-01-0005.

INTRODUCTION

Due to the faster-than-ever pace of new feature deployment in vehicles, the need to quickly design and build a flexible and reliable HW/SW architecture (hereafter referred to as *architecture*) has never been so compelling. In this paper, we focus on architecture flexibility and provide a model-based method and tool to compute this metric. We define architecture flexibility as the measure of ability of the architecture to accommodate *future changes in features with no changes in hardware* (no addition/replacement of processors, buses, memories). As design complexity increases, designers will find it increasingly difficult to

measure flexibility solely based on their experience. To the best of our knowledge no methods or tools exist to assist designers in this task. The flexibility metric is one of a set of metrics that can be used to evaluate architecture alternatives during the early conceptual phase of the development cycle. Other metrics include scalability and expandability. We provided definitions for these metrics in [Ghosal2010]. In this paper, we focus on flexibility as it is of paramount importance for automotive manufactures to cut design and variable product costs, specifically for high volume/low contribution margin programs (e.g., sub-compact segment) that are increasingly strategic for the manufactures to stay competitive in the market place. Flexible architectures enable

to cut costs as they provide opportunities to avoid late and costly engineering changes when new features are deployed onto an existing architecture.

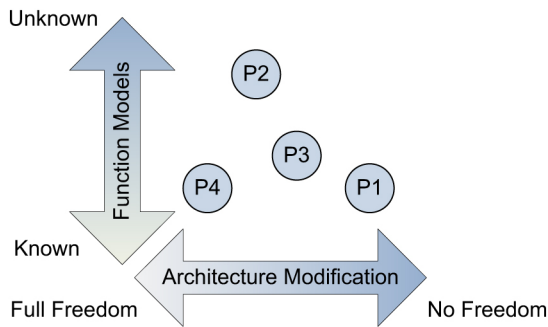


Figure 1. Two Axes of the Problem Space.

The computation model of the metric depends on the available information about the new features, and the degrees of freedom that designers have in modifying the available architecture. In the following, we assume that a *feature* is decomposed into *functions* and a *feature/function mapping* is available to the designer. A *function model* is a set of features to be implemented onto the architecture. Therefore, the problem space can be defined along two axes (Figure 1), where P1-P4 are problem instances with different degrees of freedom: function models (characterizing the available information on the new functions to be mapped onto the architecture), and degree of architecture modification (characterizing the freedom to modify the available architecture). Function models can be characterized as follows: (a) a set of features that is completely known, including the set of composed functions and signals, and their full characteristics (P4 in Figure 1); (b) a set of features with full knowledge of the functions and signals, but only with an estimation of the resource usage (CPU, memory or bus utilization) (P3 in Figure 1); (c) a set of features characterized only with the estimation of resource usage at the feature level (P2 in Figure 1). Also, because of uncertainties in the development of future features, the characterization of function and signal resource usage may not be determined with absolute confidence and therefore it may be described by a probability distribution function. The architecture modifications are as follows: (a) full freedom in mapping (P4 in Figure 1), (b) no changes to existing mapping when adding new functionality (P1 in Figure 1), (c) partial freedom to change mapping with the appropriate mapping constraints (P3 in Figure 1). For example, some functions may require particular resources (e.g. special instruction, I/O) that are only available on a particular processor, or some functions may be customized for a particular processor. Thus, mapping is fixed for some functions, while for other functions re-mapping is allowed although with some penalties associated with the cost of the re-mapping.

In this paper, we assume that the function models include a priori characterization of the estimated resource requirements, and we study whether it is possible to

accommodate new functions on the given architecture assuming that mapping freedom is available to the designer. We plan to work in the future on other aspects, including the impact of additional characterizations of feature properties (e.g., probabilistic models), restrictions on mapping, and broader freedom in modifying hardware (deletion and/or addition of parts).

We formulate our approach as an optimization problem using linear constraints that characterize the different aspects of function and architecture modification models. Although the model is extendable for different architecture and feature descriptions, with no loss in generality we have bounded the search space by making a few assumptions on the hardware and feature architecture components. We assume that the hardware portion of the design consists of Electronic Control Units (ECUs) connected via buses. Each ECU has one processor and one memory bank, and each bus consists of one bidirectional channel. We also assume that a feature consists of functions that can be distributed on multiple ECUs. A function is characterized by the required CPU computation time and memory, an activation period and computation deadline, and a signal interface. Signals between functions denote data dependencies - they do not represent activation signals. Each signal is characterized by its bus load. A function can be shared by different features. For instance, if feature F_1 and feature F_2 rely on the information provided by a function f , then function f is a part of the definition of both features F_1 and F_2 . Similarly, a signal can be shared by multiple features, e.g., features F_1 and F_2 may rely on a signal s . We use a simple fault tolerance concept as it is realized by replicating functions and signals; while mapping, different replicas of a function (resp. signal) cannot be mapped to the same ECU (resp. bus). Based on the above framework, we will formulate flexibility of a mapping (for an architecture that has pre-mapped existing features) to accommodate new features, as well as flexibility of an architecture (based on the largest number of new features that can be accommodated). The following sections will present the formal definition of an architecture, as well as translation of design information, function model and architecture modification into linear constraints.

RELATED WORK

The work presented in this paper is focused on the research performed in the area of metric-based architecture exploration of HW/SW automotive architectures ([Dinatale2007, Ghosal2007, Giusto2007, Popp2007, Ghosal2008, Ghosal2010]). Although a different terminology may have been used in the past, the general idea of metric definition and computation to capture the ability of an architecture to accommodate changes has been studied before. For SW architectures, [Weyuker2002] introduces the Performance Non-Scalability Likelihood (PNL) metric to predict likely loads at which the probability that performance problems arise will exceed acceptable levels. [Bass2003]

defines the metric scalability for software architecture as “the addition of any number of domain-specific components over time must have no impact on the performance of the application software, nor must it cause re-engineering of client applications.” Based on the observation that a long time is required to design, build, and test the individual components of a control system independently, [Fayek1993] proposes the metric expandability which captures the ability to build a control system incrementally. In [Yerraballi1996], the evaluation of extensibility with respect to changes in task execution times is studied when the system is characterized by end-to-end deadlines. [Zhu2009] looks at the issue of design robustness with respect to changes in task requirements for hard real-time distributed systems. [Zheng2005] introduces two metrics, extensibility and scalability, for tasks running on CPUs and messages communicating over buses. [Ghosal2010a] introduces the concept of robustness to uncertainty, an approach based on the info-gap decision theory, which provides a systematic scheme for analyzing robustness of (FlexRay) schedules by computing the greatest horizon of uncertainty that still satisfies the performance requirements. [Kim2011] proposes a Software Component allocation algorithm and a framework for AUTOSAR with reliability and timing constraints. This paper does not focus on defining a flexibility metric nor compares the flexibility of allocation options. To the best of our knowledge, *previous work focuses on defining metrics that are related to flexibility albeit missing one or more aspects or to optimizing the configuration of existing architecture/mapping for future extensions, without explicitly taking into consideration the anticipated deployment of new functions to the existing architecture.*

The rest of the paper is organized as follows. In the next section, we give an overview of the flexibility model using an example. Then, we formally describe the flexibility model. Then, we provide a case study on a simplified version of an automotive electronic architecture. Finally, in the last section, we provide some conclusions and opportunities for future work.

ANALYSIS OVERVIEW

We demonstrate the flexibility calculation method using a simple but representative example. In the example (Figure 2), an architecture consists of a hardware (HW) architecture, a functional (representing the software implementation) architecture, and a mapping of the functional architecture on the HW architecture. We show how the flexibility metric is computed with respect to a given library of new features.

The hardware architecture consists of three ECUs (e_1 , e_2 , and e_3) and two buses (b_1 and b_2). Bus b_1 connects ECUs e_1 , e_2 , and e_3 . Bus b_2 connects ECUs e_1 and e_2 . Each ECU has a single-core processor and a local memory. A bus is characterized by the available bandwidth. We assume that all buses in the model are bidirectional and single-channel (a multi-channel bus is modeled as a combination of single-

channel buses). The sizes of the rectangles representing ECUs in Figure 2 are proportional to the available memory in the ECUs.

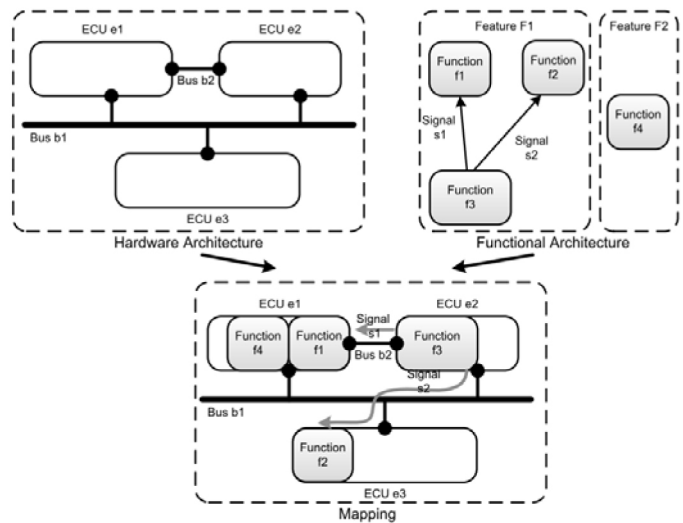


Figure 2. HW, Functional Architecture, and Mapping

The functional architecture consists of two features F_1 and F_2 . Each feature is decomposed into functions. Feature F_1 consists of functions f_1 , f_2 , and f_3 . Feature F_2 consists of function f_4 . Each function has some resource requirements (e.g. memory usage). The rectangle size representing a function in Figure 2 is proportional to the memory required by the function. Besides these requirements, functions also have data dependencies on one another, which are modeled as signals. For example, functions f_1 and f_2 receive signals from function f_3 . A signal is characterized by its communication bandwidth usage.

The mapping is a set of allocations of functions to ECUs and signals to buses. Functions f_1 and f_4 are mapped to ECU e_1 , function f_3 to ECU e_2 , and function f_2 to ECU e_3 . Signals s_2 and s_1 are mapped to buses b_1 and b_2 respectively. The mapping is performed according to the constraints on replicas, on the available memory and CPU utilization on each ECU, on the function deadlines, and on the available bandwidth on each bus. For example, function f_2 is a replica of function f_1 , and signal s_2 is a replica of signal s_1 . The replicas must be mapped onto different ECUs and buses to ensure fault tolerance. A signal can also be mapped to an ECU as long as all functions related to the signal are mapped to the same ECU.

Figure 3 shows three new features that are candidates to be mapped onto the existing HW architecture. Feature F_3 has three functions, and reuses functions f_9 and f_4 as well as signal s_5 . Typically, features are not equally important as their introduction into the market place is determined by competitive factors. Therefore, each feature has assigned a number which denotes its importance. The importance