Chapter 13
Generating Process Network Communication Infrastructure for Custom Multi-Core Platforms

Peter Sørensen
Technical University of Denmark, Denmark

Jan Madsen
Technical University of Denmark, Denmark

ABSTRACT

We present an approach for generating implementations of abstraction layers implementing the communication infrastructure of applications modeled as process networks. Our approach is unique in that it does not rely on assumptions about the capabilities and topology of the underlying platform. Instead, a generic implementation is adapted to the particular platform based on information retrieved from analyzing the platform. At the heart of the approach is a novel method for analyzing the capabilities of custom execution platforms composed of components. The versatility and usefulness of the approach and analysis method is demonstrated through a case study.

1. INTRODUCTION

Modern embedded applications are increasingly relying on heterogeneous multi-processor platforms to provide the computational power and flexibility needed to meet the ever increasing demands on performance, features and reliability. By designing a platform towards executing a specific application it is in many cases possible to achieve a more efficient system in terms of performance, power consumption and resource usage than what could have been achieved using a more general-purpose platform. Designing such platforms is expensive, time consuming and in many cases infeasible because of budget limita-
tions and/or tight time-to-market constraints. In recent years, there has been an increasing interest in system-level design tools based on component-based design methodologies from academia (Bergamaschi et al., 2000; Nikolov, Stefanov, & Deprettere, 2008) as well as industry (Altera, 2009; Xilinx, 2009). These tools allow the designer to assemble complex platforms from libraries of existing hardware and software components. The task of designing a platform consists of deciding on which components to use, how to configure them and, finally, how to connect them. Together, the resulting component allocation (choice of components), component configuration and platform topology (organization of the components) determines how well suited the platform is for executing the application.

Compared to single processor systems, heterogeneous multi-processor systems are difficult to program. In the absence of a platform-wide programming solution, shielding the application developer from the details of the platform, the developer must program each processor individually. This is problematic for several reasons: First, the application becomes dependent on the particular platform making re-use hard and changes in the platform, application and/or mapping difficult. Second, a partitioning and mapping of the application must be decided upon before any code is written which leaves little room for errors and may lead to resource waste due to over-engineering. Finally, in addition to implementing the application, the application developer must also worry about low-level processor-to-processor interfacing that in turn will require intimate knowledge of the platform in particular and low-level programming in general.

Through the use of a parallel programming model, for example, TTL (van der Wolf, de Kock, Henrikson, Kruijtzer, & Essink, 2004), YAPI (de Kock et al., 2000), and OpenMP (www.openmp.org), it is possible to abstract away different aspects of the platform, such as communication and synchronization, and thus effectively de-couple the application from any actual platform (Jerraya, Bouchhima, & Frédéric, 2006; Paul, 2003; Martin, 2006). A programming model can be provided as an integral part of a programming language or as an add-on in the form of an application programming interface (API). In most cases, a given platform does not directly implement a given programming model and thus an abstraction layer, realizing the abstractions of the programming model, is needed. An application layer can be application-specific so that it only provides support for the functionality that is actually used by a particular application. The opposite of an application-specific layer is a general purpose layer where functionality is provided to support a broader range of applications.

Compared to the cost of designing a full custom ASIC platform the cost of developing an associated abstraction layer is negligible. However, in the context of system-level design tools, where a multi-processor platform can be designed (and instantiated on an FPGA) in a matter of hours, the cost of developing an abstraction layer is taking up a relativity larger portion of the design budget and thus becomes an important factor to consider in any attempt at further optimizing the design process for customized FPGA-based embedded systems.

**Problem Description**

For embedded systems, programming models based on the *Kahn Process Network* (KPN) (Kahn, 1974) model of computation (MoC) have been studied intensively. An application modeled as a KPN consists of processes communicating and synchronizing using unidirectional FIFO channels. In theory, the capacity of the channels is unbounded and writing to them is a non-blocking operation. In this work, we will consider a more practical version with bounded channel capacity and blocking write that we will refer to simply as a *process network*. The process network MoC is especially well suited for capturing streaming (e.g. image processing) applications and has previ-