GALS-CMP: Chip-Multiprocessor for GALS Embedded Systems

Muhammad Nadeem, HeeJong Park, Zhenmin Li, Morteza Biglari-Abhari, and Zoran Salcic

Department of Electrical & Computer Engineering,
University of Auckland, Private Bag 92019,
Auckland 1142, New Zealand
{m.abhari,z.salcic}@auckland.ac.nz

Abstract. In this paper we present a novel multi-processor architecture for concurrent execution of programs that follow the Globally Asynchronous Locally Synchronous (GALS) formal model of computation. Programs are specified using the SystemJ concurrent programming language, suitable for modeling heterogeneous embedded applications that contain reactive and control driven parts and interact with the external environment. The proposed architecture is based on separating the control-driven and data-driven operations and executing them on distinct cores that support both types of operations, implemented as two modes within the single processor core. Each core can switch between two modes without any overhead. The core as the basic building block of the multiprocessor extends Java Optimized Processor (JOP), suitable for data-driven transformational operations, with control-oriented constructs that implement concurrency, reactivity, and control flow in SystemJ. Experimental evaluation over a range of benchmarks shows significant performance improvements over the existing platforms developed for the execution of the SystemJ program.

Keywords: GALS Processors, Reactive Processors, Chip-multiprocessors, Concurrent embedded systems.

1 Introduction

A wide range of embedded systems consist of multiple concurrent behaviors containing both control-dominated and data-dominated operations. These behaviors also interact with each other and with the environment repeatedly reading inputs, doing computations and generating outputs. They typically have different requirements on response times; hence, they need to run concurrently at different speeds. These systems are often modeled using GALS (Globally Asynchronous Locally Synchronous) model of computation. Although Java can be used to program such systems, it has never been an obvious choice due to the presence of a large piece of software in the form of Java Virtual Machine (JVM), as an additional layer, and garbage collector resulting in increased memory and execution cost. Java is not suitable for the hard real-time systems due to garbage collector which typically causes non-deterministic pauses in the application due to unpredictable invocation times and length of these pauses. In addition, Java threads and concurrency constructs introduce non-determinism in program execution and make it hard to analyze the execution behavior of the program. Also the lack of statements for modeling reactive control structures, support of synchronous reactive model of computation, and frequent occurrence of context switching to support thread-based concurrency reduce its efficiency.

SystemJ [1] is a system-level programming language which extends Java with synchronous and asynchronous concurrency and reactivity, making it suitable for designing complex embedded programs. The language allows use of full Java as usual object oriented sequential programming language, and does not recommend using Java concurrency (threads), but relies on its own concurrency model based on formal GALS model of computation (MoC). It extends Java with Esterel-like [2] constructs for the synchronous concurrency and reactivity, and CSP-like [3] constructs for the asynchronous concurrency. Synchronous parts of SystemJ programs are deterministic and suitable for real-time applications if the continuous blocks of java code embedded into those programs have bounded execution times.

A SystemJ program consists of multiple asynchronous processes, called clock-domains (CD), which are described at the top design level. The clock-domains are composed together with the asynchronous parallel operator (><). Each clock-domain consists of a number of synchronous concurrent processes, called reactions, which execute in lock-step, driven by a logical clock, called tick. A synchronous program reacts to its environment in a sequence of ticks, and computations within a tick are assumed to be instantaneous, i.e., as if the processor executing them was infinitely fast (synchrony hypothesis). The reactions communicate within a clock-domain, as well as with the external environment (input/output) through signals, which are broadcast and present within the current tick and comply with synchronous reactive MoC [2]. Communication between reactions in different clock-domains is carried out through the exchange of messages over channels, which are semantically the same as channels used in CSP MoC [3]. Besides operations on signals and channels, SystemJ allows free use of Java data objects and sequential statements in its reactions, and those statements are considered instantaneous in terms of logical time (i.e. they do not consume logical time or ticks).

Control flow of a SystemJ program incorporates scheduling of all reactions and clock-domains, as well as communication between reactions, and communication with the external environment. The data-driven computations and transformations are performed in Java.

This paper presents a homogeneous chip-multiprocessor architecture for executing programs described in SystemJ referred to as GALS-CMP. The multiprocessor is based on the use of multiple JOP-Plus [4] cores, which are particularly suitable for separation of SystemJ reactivity and concurrency control flow on one hand, and sequential Java computations, on the other hand. This is used as the