Chapter 15
Model-Driven Development of Multi-Core Embedded Software

Shang-Wei Lin
National Chung Cheng University, Taiwan

Chao-Sheng Lin
National Chung Cheng University, Taiwan

Chun-Hsien Lu
National Chung Cheng University, Taiwan

Yean-Ru Chen
National Taiwan University, Taiwan

Pao-Ann Hsiung
National Taiwan University, Taiwan

ABSTRACT

Multi-core processors are becoming prevalent rapidly in personal computing and embedded systems. Nevertheless, the programming environment for multi-core processor based systems is still quite immature and lacks efficient tools. This chapter will propose a new framework called VERTAF/Multi-Core (VMC) and show how software code can be automatically generated from high-level SysML models of multi-core embedded systems. It will also illustrate how model-driven design based on SysML can be seamlessly integrated with Intel's Threading Building Blocks (TBB) and Quantum Platform (QP) middleware. Finally, this chapter will use a digital video recording (DVR) system to illustrate the benefits of the proposed VMC framework.

INTRODUCTION

With the proliferation of multi-core architectures (Akhter, 2006) for embedded processors, multi-core programming for embedded systems is no longer a luxury. We need embedded software engineers to be adept in programming such processors; however, the reality is that very few engineers know how to program them. The current state-of-the-art technology in multi-core

DOI: 10.4018/978-1-60960-215-4.ch015

programming is based on the use of language extensions such as OpenMP ("OpenMP," 2008), multi-core Java (Robert Eckstein, 2008) or libraries such as Intel Threading Building Blocks (TBB) (Reinders, 2007), Microsoft® Task Parallel Library (TPL)/ Parallel LINQ (PLINQ) ("Introduction to PLINQ"), (Daan Leijen & Judd Hall, 2007).

OpenMP, multi-core Java, TBB, and TPL/PLINQ are all very useful when programmers are already experts in multithreading and multi-core programming; however, there still exists a tremendous challenge in this urgent transition from unicore systems to multi-core systems. To aid embedded software designers in a smoother transition, we propose a framework that integrates software engineering techniques such as software component reuse, formal software synthesis techniques such as scheduling and code generation, formal verification techniques such as model checking, and multi-core programming technique such as TBB.

Several issues are encountered in the development of the integrated design framework. First and foremost, we need to decide upon an architecture for the framework. Since our goal is to integrate reuse, synthesis, and verification, we need to have greater control on how the final generated application will be structured, thus we have chosen to implement it as an object-oriented application framework (Fayad & Schmidt, 1997), which is a "semi-complete" application, where users fill in application specific objects and functionalities. A major feature is "inversion of control", that is the framework decides on the control flow of the generated application, rather than the designer. Other issues encountered in architecting an application framework for multi-core embedded software are as follows.

1. To allow software component reuse, how do we define the syntax and semantics of a reusable component? How can a designer uniformly and guidedly specify the requirements of a system to be designed? How can the existing reusable components with the user-specified components be integrated into a feasible working system?
2. What is the control-data flow of the automatic design and verification process? When do we verify and when do we schedule?
3. What kinds of model can be used for each design phase, such as scheduling and verification?
4. What method is to be used for verification? How do we automate the process? What kinds of abstraction are to be employed when system complexity is beyond our handling capabilities?
5. How do we generate portable code that not only crosses operating systems but also hardware platforms. What is the structure of the generated code?
6. How much and what kinds of explicit parallelism must be specified by a software engineer through system modeling? How can we automatically and correctly realize the user-specified models into multi-core embedded software code?

Briefly, our solutions to the above issues can be summarized as follows.

1. **Software Component Reuse and Integration:** A subset of the Systems Modeling Language (SysML) is used with minimal restrictions for automatic design and analysis. Precise syntax and formal semantics are associated with each kind of SysML diagram. Guidelines are provided so that requirement specifications are more error-free and synthesizable.
2. **Control Flow:** A specific control flow is embedded within the framework, where scheduling is first performed and then verification because the complexity of verification can be greatly reduced after scheduling (Hsiung, 2000).