Chapter 4

EMBEDDED SOFTWARE DEVELOPMENT
Through The TLM Approach

Eric Paire
STMicroelectronics, France

With special participation of Kshitiz Jain, Marc Harbonne, Maxime Fiandino, and Michel Bruant.

Abstract: Early embedded software development, covering coding, testing, integration and validation, is one of the most important targets of TLM platform methodology. This chapter describes mainly the close relationship between the TLM platform and the software running on it. The description illustrates how the software can benefit greatly from the early TLM platform availability. Reciprocally, hardware developers can also benefit from the early feedback on their design when used by the software developers. The TLM platform can therefore be considered as the meeting point between hardware and software development teams.

Key words: software; Operating Systems; firmware; device drivers; application; protocol stack.

1. INTRODUCTION

Nowadays, no hardware design of a system-on-chip is worth developing without any software to exercise its functions. The trend of “the smaller the better” in SoC design concept has rapidly pushed the role of software into prominence during SoC hardware design process. While hardware aspects are getting very tough to handle due to the ever-rising SoC complexity, the weight of software aspects becomes more and more important in the overall system to manage new hardware functionalities and to replace certain hardware features.

This chapter highlights the brand-new role of software in conjunction with TLM platforms. It underlines the core idea of how system embedded
software and TLM platforms could enhance and enrich each other in their respective missions.

The conventional design approach allows a significant amount of the software being developed, compiled and tested before any strict form of the hardware platform is made available. Only a specific part of software could be developed when the detailed information tightly associated with the hardware is accessible in the form of RTL or emulation platform. This part is usually the toughest and longest to test and debug. Unfortunately, software developers are always bound to wait quite long for such hardware platform in order to validate their development work. This is not only a costly time loss, but also an inefficient cooperation between hardware and software designers for lack of a common development base.

Despite the somewhat opposed design philosophies between hardware and software fellows, current SoC complexity is urging these two worlds to work together in a new way leading to concurrent hardware/software design. Time-to-market reduction and cost saving will be the successful culmination of such parallel hardware/software design.

The idea of hardware/software co-design and co-implementation can be realized through a unique reference -the TLM platform-. Indeed, TLM platforms provide adequate and accurate hardware information for software designers much earlier than the conventional platforms such as RTL platforms. This information must be sufficiently accurate for software designers to start developing, testing, and debugging the software code closely associated with the hardware \textit{without} pointless delay following the initial software development. In parallel, hardware designers can develop RTL platforms aimed at timing-accurate simulations, which are eventually employed for logic synthesis.

By the time the RTL design is complete, the software will have already been thoroughly verified on TLM platforms. The software design is thus ready to be integrated with the RTL hardware platform for system validation within a much shorter time than the traditional approach. As a result, sound and solid concurrent engineering is achieved through the unique reference of TLM platform.

A closer study clearly reveals that software running on TLM platforms can be classified into different categories according to their relationships with the hardware platform. This chapter will discuss extensively on the software categories ranging from design requirements to the mutual expectation of benefits between software and its hardware counterpart. Lastly, the chapter will draw a conclusion on how close collaboration between hardware and software developers could lead to a virtuous circle.