

Constructing Electronic System Level Models Using Simulink

Cheng-Chien Chen

No. 5, Li-Hsin Rd. III, Hsinchu City,
Taiwan 300, R. O. C.
886+3-578-7888 Ext. 8768

ken_chen@faraday-tech.com

ABSTRACT

Our work aims at introducing a methodology that adopts nowadays existing tool to develop models that can be reused repeatedly in the future. This methodology enables MATLAB Simulink to involve in the entire SoC design phase from high level abstract algorithm design all the way down to cycle-based comprehensive hardware implementation, to consider simultaneously both control plane and data plane design aspects. Models constructed in this way offer versatile solutions that suit a wide range of electronic system level (ESL) design requirements, they can be integrated in pure software virtual platform, VHDL/Verilog simulation environment, or hardware accelerated simulation solutions. An intellectual property (IP) design project is used as a case study, showing a significant improvement on design flow and IP quality.

Keywords

Electronic system level (ESL), modeling, Simulink.

1. INTRODUCTION

As the design complexity increases, in order to secure the implementation feasibility and time-to-market, the design methodology has turned from transistor level, gate level, register transfer level (RTL), to (the well-known but un-unified) electronic system level (ESL). By definition, the ESL design methodology aims to offer a solution to designing from the system or application perspective. It tries to build up a design flow that can perform algorithm/architecture exploration, hardware/software co-simulation, and even verification in an efficient and effective way. However, resulting from the variety or diversity of the problem the system design may encounter, it is hardly to have one unified design flow that can meet those demands for all kinds of systems or applications.

In spite of the uncertainty and crudity of the ESL design flow and tool sets current existing, all ESL technologies essentially share two fundamental objectives: facilitating design reuse and supporting design abstraction. From this point of view, we find that a model with the following two objectives is the crucial constituent towards ESL methodology.

1.1 Scalability

Among different design stages, an ESL model should be able to support abstracting the design, validation analysis, and implementation processes. Seamless migration between different abstract levels links up high level system simulation with detail function verification reliably.

1.2 Reusability

We usually acquire different tool sets to deal with different system or application issues. Therefore, ESL model needs to be integrated in C/C++ environments as well as in VHDL/Verilog

environments. Easy transformation or wrapping of the model to fit to various tool sets or usage scenarios makes the model extensively applicable.

In this article, we present a modeling methodology that can be used to create models satisfying both of those two objectives. These models can be system models, software models or hardware models. Models constructed in this way are easy to be implemented and integrated with. Designers with different domains, system or IP element, hardware or software, digital or analog, can manipulate this methodology to design and share their models effectively.

2. SIMULINK-BASED MODEL DESIGN METHODOLOGY

MATLAB's algorithm developing capability and Simulink's model-based design methodology are well recognized and adopted across industry and the academic world. The wide spread of this tool makes EDA vendors support Simulink model as if it were a standard. For software implementation, Real-Time Workshop provides a very effective link with many software development environments. As to FPGA solution, there are many possibilities to generate synthesizable code from Simulink models. For hardware implementation, Simulink HDL Coder is quiet adequate for generating synthesizable Verilog and VHDL code. Models constructed on such environment possess high reusability in nature.

Furthermore, Simulink provides high flexibility in describing and simulating heterogeneous systems. The design methodology, such as state flow and graphical multi-domain simulation, makes the model easy to be implemented, discussed, and integrated. Those gripping features provide us with a fundamental environment for developing high-reusable and high-scalable ESL models. The outlook of our proposed Simulink model is as shown in Figure 1. Base on this shell, we propose the following design methodologies to guarantee high reusability and high scalability.

2.1 Handshaking Mechanism

Every module is activated according to the EventIn ports and the granularity of the events enables swift switch between transaction-level and cycle-level simulation. We maintain two signals, accept and complete, for each RequestIn Ack port and RequestOut Ack port. By utilizing these two signals, we can accomplish the functionality of performing outstanding and command buffering mechanism.

2.2 Interface Design

As design complexity continues to increase, the integration of the constituent components of the system is a non-neglectable task, having a significant impact on the quality of designs and on development schedules. Therefore, the target of the interface design is to make the modules be easily integrated, reorganized or

replaced for architecture exploration. To this end, we define a universal interface for the RequestIn and RequestOut ports that comprise three channels, slave index or address, function index or data size, and message or data.

2.3 Design Language

State flow design methodology provides us with a nimble way toward abstraction. We can simply redirect a state transition and skip a comprehensive design by designing corresponding cost functions or timing tables. It also provides a good presentation of the design for both hardware and software components. Therefore, we use Simulink Stateflow as the fundamental of the modeling language.

3. CASE STUDY: AN IP DESIGN PROJECT

The proposed methodologies can be applied to system models, software components, digital hardware designs, and analog designs. Nevertheless, here we use a digital IP design project of NAND FLASH controller as a case study. In this project, our objective is to perform the architecture exploration of a FLASH controller which bridges a high-speed IO and a FLASH array. The whole system is considered to operate under Windows or Linux operating systems.

We start with designing some basic and fundamental parts for hardware design constructions, such as buffer-controller, protocol handler, arbiter, and SRAM, to name a few. These components are throughout verified to meet real hardware design capabilities. Many benches and random tests are applied to ensure their functionality and robustness. Also, an AHB bus and FLASH memory models are modeled in this way.

Then we bundle up the characteristics of operating system, file system, and high-speed IO link and design a component that models how application system utilizes this controller. Context switch duration, interrupt latency, sector size of the file system, and the bulk size of the high-speed IO are all parameterized and measured to cover a wide range of application scenarios. These components, hardware design or system behavior, are configured, assembled, and encapsulated into bigger constructs. A system model is then quickly constructed out of these primitive programmable components.

Taking the advantage of our interface design, we quickly build up a simulation system that consists of an application workload, a DMA, an AHB bus, an arbiter, some channel managers, two shared SRAM FIFOs, and a FLASH farm, as depicted in Figure 2. The overall simulation speed is around 200 times faster than RTL simulation and thus we have the opportunity to investigate much more system configurations.

This system allows for repeated analysis of the IP from various application perspectives. Since many design aspects are parameterized, we efficiently change many major system conditions and generate lots of performance reports for architecture exploration. We evaluate the IP performance under SATA and USB3.0 use scenarios. Operating systems with different context duration and interrupt latency are evaluated, too. After a quick but overall exploration, we examine the performance trends and check the conditions and performances that look peculiar to us.

For example, through a throughout profiling, we observed that the performance for low operating frequencies is not as good as expected. As a result, we analyze the cycle behavior of the system, as shown in Figure 3, and quickly find the defect of the algorithm under such system condition. After modifying the arbitration policy and the behavior of smart buffer, we successfully preserve full bus utilization and gain 15% performance improvement against the original design.

Besides, while developing the ECC engine of the NAND controller, it is not easy to model the distribution of FLASH errors using traditional design methodologies. Consequently, in order to catch up with the schedule, designers are coerced to make their decisions in the worst case only and this always leads to over-designs. However, by adopting this methodology, we quickly form a statistical model for FLASH errors and explore the performance under nominal cases. We shortly find the over design issue and modify the architecture to reduce the gate count to 57% of the original design. Moreover, since the design is originally derived under high operating frequencies, we also observe the performance shortage in low frequencies and then enhance and verify the corresponding architecture thereby.

In addition to the FLASH controller design case, we utilize this methodology to develop DDR controller under customer's proprietary virtual platform and improve system performance over 33%. We also successfully porting this DDR memory subsystem, within one day, to Virtutech Simics simulation environment and provide more precise memory access behavior for multi-core project. Putting this model into Carbon simulation environment through ARM CASI interface is also done rapidly. Meanwhile, we manipulate such methodology to perform pipelining and retiming design of ECC engine and seamlessly provide a cycle-by-cycle verification environment for RTL design. All demonstrate the broad scope of this proposed methodology.

4. CONCLUSIONS

Whereas cycle-based architecture design is usually viewed as a missing part of Simulink, in this article, we have shown that adopting Simulink environment to develop a high-reusable and high-scalable model is practical and beneficial. Instead of integrating various modeling languages and environments, we effectively use Simulink to get all necessary design aspects done. Models so created can be easily piled up, reorganized, and configured on Simulink environment for architecture exploration. Also, they are adequate to be the implementation basis and as a communication bridge between different technology domains. Their seamlessly abstracted versions can serve as the constituents for higher level development. At the same time, they can be converted to HDL to aid verification or become the hardware components for emulation systems.

By distributing these models to various usage scenarios, we receive many design feedbacks and, accordingly, we can refine the abstract description of the design, the accuracy of the model, and the design implementation simultaneously. Eventually, we develop a set of multi-purpose models that can be used to explore the architecture, to assist the design of different technology domains, and accelerate the design and verification process. Thus, without waiting for the convergence of the ESL toolsets, the models and the corresponding designs help us to keep up with the ESL trends constantly.

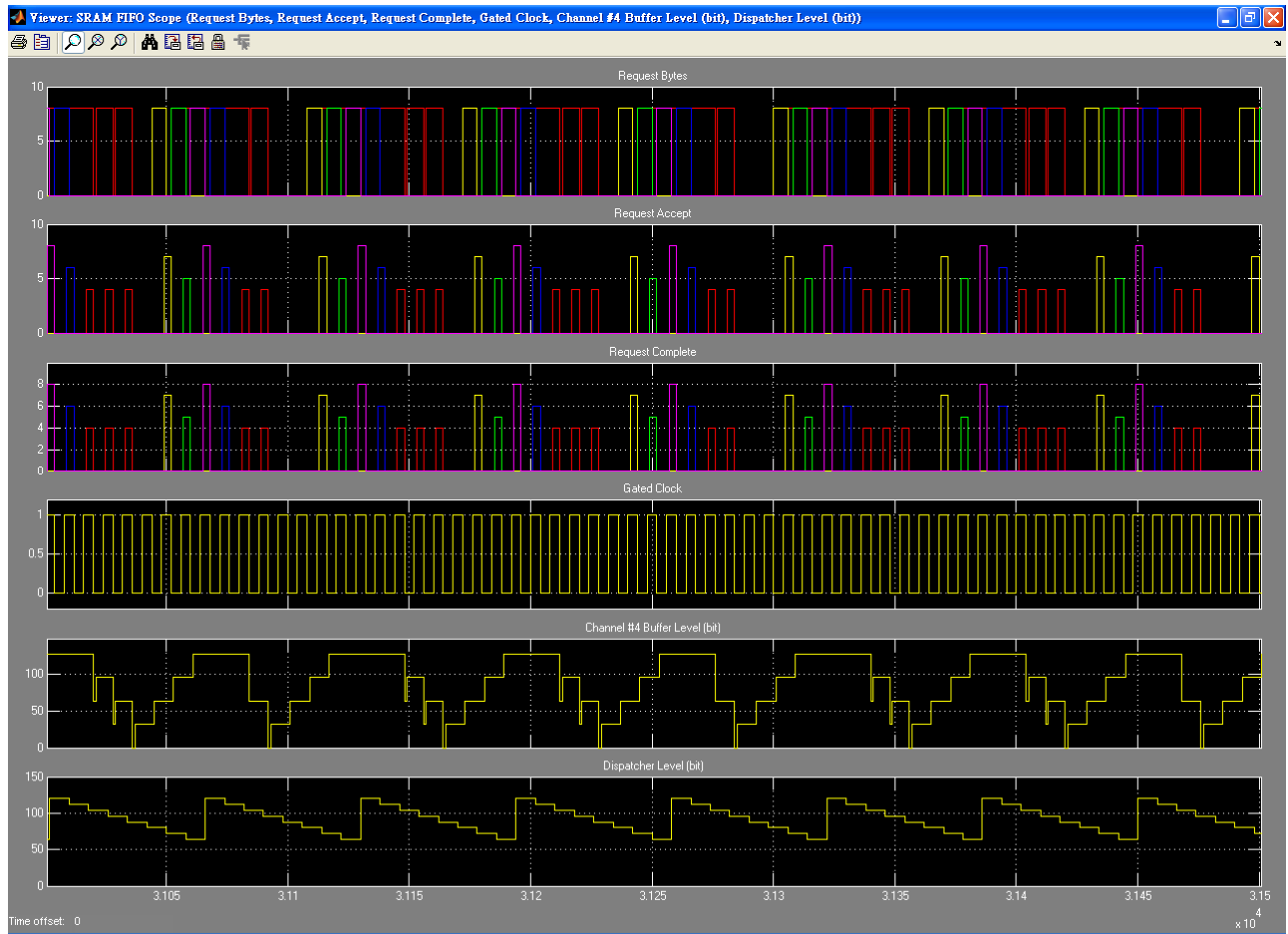


Figure 3. A snapshot of the cycle behavior of the FIFO of the FLASH controller.