Platform Based Design at the Electronic System Level

Platform Based Design at the Electronic System Level

Industry Perspectives and Experiences

Edited by

Mark Burton
GreenSocs Ltd, France

Adam Morawiec
ECSI, Grenoble, France

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-5137-9 (HB)
ISBN-13 978-1-4020-5137-1 (HB)
ISBN-10 1-4020-5138-7 (e-book)
ISBN-13 978-1-4020-5138-8 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

All Rights Reserved
© 2006 Springer
No part of this work may be reproduced, stored in a retrieval system, or transmitted
in any form or by any means, electronic, mechanical, photocopying, microfilming, recording
or otherwise, without written permission from the Publisher, with the exception
of any material supplied specifically for the purpose of being entered
and executed on a computer system, for exclusive use by the purchaser of the work.

CONTENTS

FOREWORD: ENABLING PLATFORM-BASED DESIGN

My first exposure to the concept of Platform-Based Design (PBD) of integrated circuits, more particularly of the emerging "System-on-Chip" or SoC devices, was at lunch in the Cadence cafeteria in the mid 1990's, when a colleague who had worked for VLSI Technology, Andy McNelly, explained their concept of a design "platform". Among those who were most influential in advancing this concept at VLSI Technology was Bob Payne, who coined the still memorable phrase "deconfiguration is easier than configuration", in describing his concept of Rapid Silicon Prototyping (RSP). Indeed, VLSI's leadership in what came to be called "Platform-Based Design" was one key reason for them being bought by Philips in 1999.

At that time, platform-based design was an interesting concept that was far from being universally accepted as a key way to do SoC design. Indeed, the concept of SoC design, and the reuse of intellectual property (IP) in the design of complex chips, was itself just in its infancy. Many designers and observers of the design scene put their faith in other approaches such as block-based design: ad-hoc IP assembly of new and reused design blocks as the situation demanded, for integrated circuits that were designed new each time, with little sharing or reuse of underlying architectures.

But there was something just too compelling about the Platform-Based Design concept to ignore. Indeed, this whole idea of a design process promised four important advantages over the more conventional top-down and block-based design methodologies then in widespread use:

1. It offered a process for IC implementation that could become regularised, standardised, highly productive and with much lower risk. Basing designs on an "integration platform architecture" with libraries of pre-proven IP blocks was a powerful method to bring order to the chaos of the normal custom design-driven IC design process.
2. It brought order and methodology to the emerging field of IP reuse. Blocks could be designed for reuse in a known family of architectures, with more predictable interfaces, rather than being designed to fit in everywhere, usually by risky ad-hoc interface design for each new chip. Platform-Based Design promised to be the best enabler of IP reuse.
3. It gave a level playing field, or sandbox, in which hardware designers and software developers could interact. Already in the mid 1990's we saw the development of processor-based platforms, cellphone baseband chips, for example, used a combination of RISC, DSP, hardware blocks, memories and buses as the basic architecture. Platform-based design offered better-thought out and

controlled hardware variability, giving the software people less to worry about in their hardware-dependent firmware. And moving processor choice from free-form to a well-thought out evolution gave the hardware people less novelty with each new design, lowering risk and enhancing productivity.

4. Finally, it promised to evolve up towards new system-level design abstractions that would offer much greater productivity and an ability to design derivatives from platforms with much greater speed and lower risk than the classical RTL and C abstractions of existing design methods. Although system-level design existed before platform-based design, the marriage of the two seemed like a marriage made in heaven.

Advocating Platform-Based Design as a key methodology for SoC began to seem less like a wild and crazy new adventure, and more like "good old Northern common sense". Although there were skeptics among the industry analysts, mere observation indicated that more and more design teams and IC design companies were beginning to explore this methodology and adapt and adopt it to their particular product design philosophies. It was the basis for the SoC design methodology that Cadence developed for the Alba project in Scotland. It first seemed like a good way to "survive" the SoC revolution, and then seemed more and more like the best way to "win" this transition in the industry.

Now, a decade later, what is the current situation in complex SoC design? First, it seems clear that Platform-Based Design is a well-accepted methodology, even the skeptics have hopped aboard the train. If one looks out in the industry, one can see many many examples of this approach used by different companies and teams. Philips Nexperia, ST Nomadik, TI OMAP, Xilinx Virtex II/IV, Altera SOPC, Infineon, Freescale, Samsung, Toshiba, Sony, and many many more chips, design groups and companies are all out there designing and using platforms successfully.

From the 2006 viewpoint, then, what remains to be done? We can see evidence that the first three promises of Platform-Based Design have been met:

1. We see that IC design processes have become much more regularised. Indeed, we see design service companies offering IC implementation services as partners to global design companies with high-level handoffs and high success rates.
2. We see substantial, extensive and growing IP reuse. From processors of all kinds to memories, from buses to hardware blocks, from device drivers to middleware to software applications, hardware and software IP reuse has flourished. It may not have evolved in the directions first envisaged by some of the early dreamers – we don't see IP being designed in every garage in Silicon Valley by independent contractors, for example – but it flourishes nevertheless.
3. We see much more processor centric design. Instruction set processors, whether fixed or configurable and extensible, are the new lingua franca of complex electronic product design – the place where complex functionality is defined and realized. Creating architectures where processors can flourish and prosper is the true mission of hardware design.

But we're missing something . . . . . what about system level design (SLD) and the move to higher design abstractions? Has this occurred? Are all complex platforms

modeled at an abstract level? Do all derivative products get designed and verified at a high level so that the rest of the design flow becomes "mere implementation" or "an exercise left to the readers"? I think we can say no to this question, but offer a cautious maybe as the answer to this question a year or two from now.

Despite being a hypothetical marriage made in heaven, the union of PBD and SLD, to become the PBSLD family, seems to have been ever protracted. In particular, one can say that SLD (or "Electronic System Level" – ESL design, as it has been renamed) has been "always the bridesmaid, never the bride". Early efforts were made by busy matchmakers to bring this couple together, but they did not succeed. No doubt there were many reasons for this failure. Among them was surely the level of complexity of early platforms – they were not complex enough to design that use of SLD methods was essential. Rather, traditional methods of muddling through were good enough. Early baseband processors in cellphones may have had both a RISC and DSP – but these were relatively isolated subsystems with relatively simple interactions, and the software load was (relatively) small. Because the interactions were simpler and the amount of software smaller, the verification task was more tractable using simpler and traditional design methods.

Early SLD tools of the later 1990's and early in the new millennium also lacked standard specification languages and model interoperability, which discouraged models from being created, since they would need to be redone in each new proprietary language or format that came along. And every company or design group had a different notion of what abstractions were appropriate. Transaction level modeling started to be advocated, but in its early days, "it all depended on what your definition of a transaction was".

So, for many good reasons the PBD-SLD marriage was not necessary then. Why is it a good idea now, in 2006? Several reasons come to mind:

1. Designs are much more complex. Leading SoCs have many processors – often 3, 4 or more, as well as many memories, complex bus and memory access hierarchies, many hardware accelerators and peripheral blocks and many external interfaces. Verifying designs of this complexity with traditional methods has failed.
2. Software content of advanced SoCs has grown enormously. You don't put up to 10 processors or more on an SoC with many large memory blocks without wanting to fill that memory up with software. Software is increasingly the medium of choice for delivering advanced functionality to users. And you don't verify software of hundreds of thousands or millions of lines of code complexity using traditional RTL or cycle-accurate ISS models alone.
3. Despite fits and starts, EDA industry politics and competing languages, and the opacity of its steering body, a system level language standard has emerged, with the ratification of SystemC by the IEEE as IEEE standard 1666–2005, and the publication of its Language Reference Manual at the end of March, 2006. Mirabilis of mirabilis, we have a standard system level modelling language – and a pretty fair shake at one, too.
4. We are also stumbling or slouching towards an industry consensus on design abstraction levels as well. The notion of Transaction Level Modelling, advocated

for several years, may, within the year 2006, become a common definition among the disparate community of system modelers, design companies, IP companies and language experts. The mills have been grinding slowly on this one, but one hopes they have been grinding towards an exceedingly fine result.

Always an optimist even in my blackest moods, I would like to feel that the final roadblocks to the marriage of PBD and (now) ESL are being dismantled. And that leads us to the subject of this book.

This book crystallizes a snapshot in the evolution towards the standard modelling notions and design abstractions that will enable high-productivity, low-risk, ESL-based Platform-Based Design possible. It has been written by experts from a variety of perspectives (note to the gentle reader: I wrote a chapter, but would demur from the label "expert"). These include ESL tools companies, modelling companies, IP companies, semiconductor platform providers, independent consultants and academia. Naturally, as with any complex subject that has not quite reached a complete consensus, there are different notions and fine shadings and colouring on the standards, methods, and abstractions necessary for this emerging methodology. But the authors agree on several key points:

1. the need for standards in modelling and to enable model interoperability.
2. the idea of the Transaction as a key enabling abstraction level and enabler of the ESL methodology for PBD.
3. the need to give software writers a better and more realistic model of the target platform, and to try to offer them the right tradeoff options of speed vs. accuracy (speed being paramount).
4. the centrality of processor-based design and the corollaries of buses and other interconnect structures as being a key part of the SoC architecture.
5. the search for even higher level abstractions than the TLM model offers to allow platform providers to characterise their platforms in new ways, and designers to choose particular design tradeoffs in more efficient ways.

With agreement converging or near on these key points, it seems that the vision of interoperable TLM models and higher level software prototyping models seems much closer to realisation. 2006 seems likely to become an annus mirabilis after all.

I commend this book to the reader as the best way to dip their toes into the murky waters formed by the confluence of Platform-Based and Electronic System Level Design. Necessity may have forced the Shotgun Marriage of these two Partners, but Let Us Hope it is a Fruitful One, and gives Birth to Many New Opportunities.

Grant Martin
Santa Clara, California
April, 2006

CHAPTER 1

THE NEED FOR STANDARDS

MARK BURTON[1], AND ADAM MORAWIEC[2]
[1] *GreenSocs Ltd., www.greensocs.com,*
[2] *ECSI, www.ecsi.org*

1. CONTEXT AND SCOPE OF THE WORK

This book presents a multi-faceted view of the set of problems that the electronic industry currently faces in the development and integration of complex heterogeneous systems (including both hardware and software components). It analyses and proposes solutions related to the provision of integration platforms by SoC and IP providers in light of the needs and requirements expressed by the system companies: they are the users of such platforms which they apply to develop their next generation products. Further, the book tries to draw a comprehensive picture of the current "interfaces" between the platform providers and users, defined by technical requirements, current design methodology and flows, standards, and finally by the business context and relationships (which should not to be underestimated). These producer-consumer, shared "interfaces" enable (or should enable) the exchange of a well-understood and complete set of data between both parties to ensure design efficiency, high productivity and best use of domain-specific expertise and knowledge.

The problems to be solved are related to modelling of platform functionality and performance (formalisms, methods, metrics), interoperability of models, architecture exploration, early SW development in parallel to the HW platform instantiation, verification and debugging methods and flows, management of complexity at various abstraction levels, and the implications of the trade-offs between the accuracy and complexity of models. The solutions discussed by the contributors to this book have one common denominator: these are standards. In the general sense, the book provides views on why and what kind of standards are the prerequisite to the deployment of a platform based design ecosystem, in which cooperation is

M. Burton and A. Morawiec (eds.), Platform Based Design at the Electronic System Level, 1–11.
© 2006 *Springer.*

made possible between all parties involved in system development: system houses, platform and IP providers and EDA companies.

2. OVERVIEW OF CONTRIBUTIONS

The complex problem of Electronic System Level design is presented in the chapters of the book from various perspectives.

First, Densmore, Donlin and Sangiovanni-Vincentelli present in Chapter 2 an elegant overview of the design tasks associated with the Electronic System Level (see their Figure 2, Transcending RTL effort with ESL design technologies). They split the design task into essentially three main activities:

1. Early software development.
2. Functional verification and executable specification.
3. System-wide performance analysis and exploration.

What is interesting about this list is that while it is split up by different people in different ways, the core elements are invariably the same. The split becomes interesting as different companies have taken different approaches to minimising the number of models they need to write. Hence some try to service all the activities with one model, while others use targeted models for each activity.

They go on to focus on performance analysis, presenting a formal way in which they have characterised system performance. Addressing the same issue, Aldis presents in Chapter 3 a simulation framework that concentrates totally on the communication fabric. Again, he characterises the components of the system, or uses sampled data in order to drive traffic generation on the communication infrastructure. The interest of this approach is that he is able to build complete systems models of the OMAP platform in response to user requests in a very timely manner. The "modelling cost" is relatively low. His model only focuses on one aspect of design activity, but does it effectively. This style of modelling could be characterised as being just about the timing (often abbreviated as T).

In chapter 4, Maillet-Contoz addresses software development and functional verification aspects of the design activity with somewhat more costly models.

His models could be characterised as being just about the Programmers View (PV) of a system. He takes the standpoint that if the programmer cannot observe the effect, then it needs not be modelled. Indeed, somewhat stronger than this, if the programmer is not permitted to rely on an effect, then it may be modelled in any way that suits the model the best.

This opens up an area not covered by the submissions to this book: namely, the prospect that models of system performance can be more useful than real hardware implementations. The case of a programmer not being permitted to make use of a hardware effect, but doing so none the less is all too common. Such hardware related "bugs" are often invisible until the underlying hardware is changed and the software is intended to remain the same. The worst case of this is where the hardware feature becomes non-deterministic, and the bug can then remain hidden for hours of runtime. Models can play an important role here in providing platforms

that either check that the software is only relying on permissible features of the hardware, or "strain" the software to fail if it does not. An example of this would be a cache that either caches nothing or everything.

While several contributions to this book have looked at how to evaluate performance, Martin describes in chapter 5 some of the options that can be chosen to better achieve the required performance. In his case, this is very much focused on the processor core itself, however it is apparent from all the contributions, and it is a theme that will be returned to, that platforms are not monolithic: to be successful, they have to be configurable.

In turns Kogel shows in his contribution (chapter 6) how a consistent methodology can help in creating peripheral models that can be plugged into and out of a platform. This is essentially a part of the same platform configuration story. Kogel's work focuses on the simulation activity: to continue the theme above, he addresses a combination of Programmers View and Timing (PVT).

In chapter 7, Hellestrand shows how important this level of modelling is to achieve accurate simulation results for critical systems. He also points out how important it is to be able to achieve these results quickly, both in terms of the speed of writing a model and the speed of execution of the model, as typically, in order to fully characterise a system will take a large number of scenarios and a lot of code needs to be executed.

Throughout this book, the authors have repeatedly returned to the subject of standards. In this brief summary, we will give an impression of why this is such an important issue in the field of platform-based design. Also, some of the aspects of standards bodies that help standards to be created in a timely and effective manner will be discussed.

3. STANDARDS IN PBD?

If we stand back for a moment, it seems strange that standards are interesting at all for platform-based design. Indeed, one would expect that fewer, rather than more standards might be involved as the platforms themselves "envelop" more and more of the design space. Designers of IP blocks previously had to conform to standards on their peripheries, but, if the IP block is the platform, then the internal structure should not matter.

This is anything but the truth. In reality, platform based design is not about all consuming platforms, rather it is about the ability to construct customised and tailored designs quickly, based on a common kit of parts.

The debate about whether there will be one or many platforms misses the key issue. Each manufacturer may only support a single "platform" but the benefit of such a platform will be its ability to be highly configurable and customisable to individual applications. In order for this to become reality, standards become more important than ever. Now, IP blocks or entire sub-systems will be quickly (re)used in combinations that the original designers may not have thought about. It becomes important not only to have interconnectivity standards, but standards that support