Platform-Based Design and the First Generation Dilemma

Jiang Xu and Wayne Wolf

Dept. of ELE, Princeton University
Jiangxu, Wolf@ee.Princeton.edu

Abstract

In this paper, we analyze system-level design methodologies for platform-based design. Platform-based design is popular as a way to reduce development time, but creating the platform is difficult. We introduce the ***first generation dilemma*** *problem --- initial designs are much harder because we do many of the components that will be used to build the system. We explain the origin of this problem and give a solution.*

1 Introduction

Due to smaller feature sizes, demand for more functional and cost-efficient products, and shorter times to market, design complexity is growing rapidly, but design productivity lags far behind. System-level design tools are introduced to absorb the growing complexity and accelerate larger designs. Many system-level design tools are announced in past two years, including CoWare N2C [1], Cadence VCC (Virtual Component Co-design), Innoveda Visual Elite [2], Elanix SystemView [3], and Synopsys CoCentric System Studio [4].

In this paper, we analyze platform-based design, which uses an existing base of components and architectures to reduce design time. One important discovery is that the platform-based design cannot solve the so-called **first generation dilemma**. Developing the first generation of a platform is much more difficult than using the platform to create a spin-off design. Tools and methodological research has emphasized using the platform, but more attention needs to be paid to developing the platform. We provide some solutions to this first generation dilemma and give an example. We will use the Cadence VCC tool to illustrate our observations, but our results apply to other system-level tools as well.

The next section introduces system-level design methodologies. In section 3, we analyze platform-based design methodologies in more detail and show our results. We will explain the first generation dilemma and give our solution and example in section 4. Finally, conclusions are given in section 5.

2 A typical system-level design methodology

A system-level tool helps users translate design specifications into chips and code running on hardware. Embedded systems are the main aim of this kind of tools. Some embedded systems are small enough to put onto a single chip, while others are implemented as chip sets because they are too large or have to be placed distributively. A typical design flow goes through steps shown in figure 1.

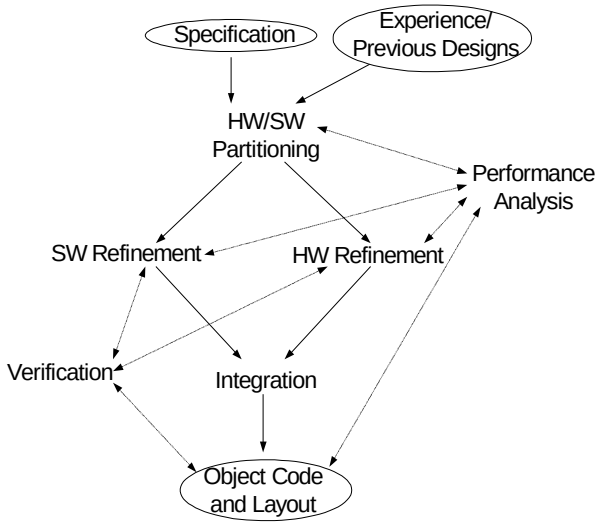

Figure 1: A typical system-level design flow.

2.1 Typical design flow

Usually a design starts with design specifications. The specifications describe requirements of a system in English. They include the expected behavior, performance, power, area, and testability of a system design. Designer experience is also an important input to the design. Sometimes a design is only a new generation of previous system designs, of which most parts could be reused.

Based upon experience with previous designs, designers choose an architecture and partition the design into HW and SW. If previous designs exist, HW/SW partitioning and choosing architecture are relative simple, and the main concern will be how to revise the previous designs. There are three kinds of revisions. Some redundant functions will be removed from the system. Most of the time more functions will be added. Sometimes some features of the system, including performance, power, area, or testability, need to be improved. Reusability is

gradually becoming the center of choosing architecture, where IP cores and customized modules are used. Recently platform-based design is becoming a main actor in system-level design [5][6][7][8]. It reuses the whole platform instead of individual IP cores and significantly improves the reusability and design efficiency. The impact of platform-based design to system-level design is that it converts the HW/SW partitioning into function mapping and platform revising. At the same time, early performance analysis is needed to direct the function mapping and platform revising.

After choosing architecture, designers select IP cores and customized modules for the architecture. Some architecture modules are not available, while some need to be revised. Hardware design will go through register transfer level design that breaks a module into smaller function units between registers, logic design that implements the units by gates, and physical design that partitions, floor-plans, and routes circuits. The software part of a design will go through software architecture analysis that chooses software architecture, dividing functions into modules, implementation of the modules, and generation of objective code. There are interfaces between SW-HW, HW-HW, and SW-SW. Usually in software architecture analysis SW-SW interfaces are solved, while SW-HW interfaces, also called device drivers, are partially solved and may need some change after real hardware are designed. HW-HW interfaces will become a big issue if IP cores are incompatible with each other. A platform-based design only needs a little interface design if any, which is a great merit. The integration step combines the software and hardware to generate the final design, which usually includes layouts for chips and object code.

2.2 Performance analysis and verification

Verification ensures that a design realizes all functions of a system under certain timing constraints as described by specifications. Performance analysis tries to reveal limitations of a design on timing. Performance analysis can take place in many steps in a typical system-level design flow. An accurate early performance analysis can help designers make right decisions and save efforts and time. Usually early performance analysis is fast because relative few details involve, while it also tends to be inaccurate due to the same reason. On the contrary, late performance analysis is slow and more accurate. The earliest step, which can take a performance analysis, is HW/SW partitioning.

In platform-based design, the performance models of previous designs are bases of performance analysis for a new design, and they make an early performance analysis possible. In the SW refinement step, HW refinement step, and object code/layout step, performance analysis ensures that a gradually detailed design satisfies specifications. Verification could also happen in the SW refinement step, HW refinement step, and object code/layout step.

3 Platform-based design methodology

Platform-based design uses IP blocks to build system architectures. Based upon Cadence VCC version 2.1, we will analysis its design flow and methodology in detail.

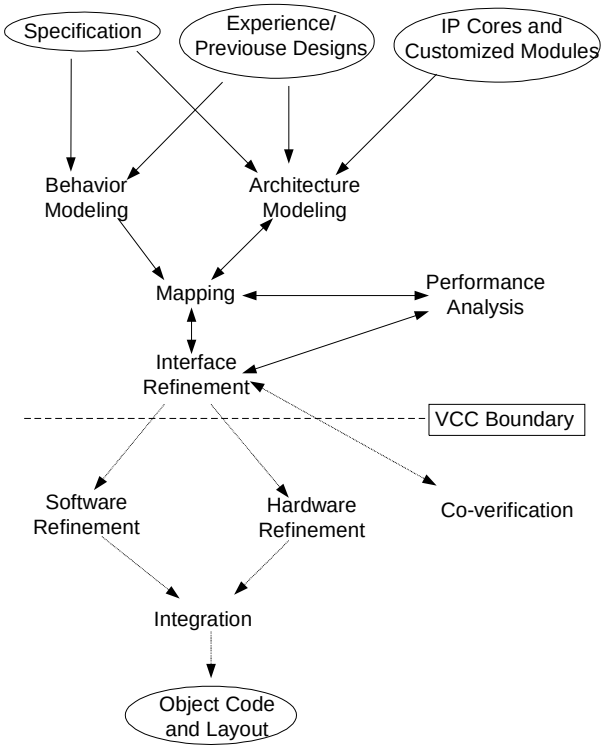

Figure 2: Platform-based design flow using Cadence VCC.

3.1 Design flow

The platform-based design flow using VCC is shown in figure 2. It begins with specifications and finally generates a logic-level design. Synthesis, physical design, and verification are accomplished by other tools. For example, VCC can export a design to Seamless Co-Verification Environment from Mentor Graphics [9]. So VCC must closely cooperate with other tools to complete a design. VCC divides the HW/SW partitioning step in the typical design flow into behavior modeling, architecture modeling, and mapping. This separation facilitates platform-based design and IP reuse, but it also poses some problems.

Behavior modeling is a key step for three reasons. First, behavior model details the function constraints of the

specifications. Second, it will be used to develop object code running on hardware. Third, it will also be used to develop hardware. Behavior model could be written in many languages, including C, C++, SDL, and STD. They are VCC versions and compatible with the standards. We find C and C++ are more convenient because a lot of experience from other designs and tools. When writing behavior model, designers need to choose a language style between software-oriented style and hardware-oriented style. Software-oriented style will ease the generation of an efficient object code, while it won't be good for hardware design, and vice versa. There is currently no programming technology that lets us easily translate between hardware-oriented and software-oriented descriptions in any language. There is unlikely to be such a solution any time soon because the software and hardware computational models are fundamentally different.

	Black Box	White Box	Clear Box
Language	C++, SPW, SDL, OMI	WhiteBox C	STD, Textual SDL
Simulated	Yes	Yes	Yes
Analyzed	No	Yes	Yes
Synthesizing	No	No	Yes

Table 1: Categories of behavior model.

VCC has three categories of behavior models, namely black box, white box, and clear box, which are distinguished by the language used (table 1). For example, a block box could be directly used to functional simulation, but not the performance analysis and hardware and software synthesis; it could be implemented by C++, SPW, SDL, or OMI. The last three language forms are used to import models from other tools. If using black box behavior model to analyze performance, designers must manually implement performance model, which captures the performance of the black box behavior model. And the capture easily betrays the performance of the original model. On the other hand, it is almost impossible to transform a behavior model between the three categories without an overhaul, because they use different languages. For this reason and also because C could easily be transformed to other languages, beginning from white box behavior model gives more flexibility.

The architecture model is simply implemented by choosing IP cores and customized modules and connecting them together. VCC doesn't need the implementation details of IP cores or customized modules to analyze performance; instead it needs performance models, which summarize the timing characteristics of IP cores and customized modules. For example, a performance model of a microprocessor is about the times needed for each instruction. Logic-level implementation is needed only when the design will be exported to other tools to continue a design.

By mapping a behavior model onto an architecture model, VCC partitions the specified functions, which are detailed by behavior model, into software and hardware. Then performance is analyzed based upon the mapping, and necessary adjustment is following. Interface refinement chooses proper communication methods among hardware and software to meet the performance requirements. It usually takes place after major mapping decisions have been made. Finally, the design can be exported to a co-verification tool and other tools to finish the synthesis and object code generation.

3.2 Performance analysis

Designers use performance analysis to evaluate architecture model, mapping, and interfaces. And designers adjust design decisions by looking at analysis results. The accuracy of performance directly affects the whole design. VCC needs a performance model for every IP core and customized modules in architecture model and for some module in behavior model. A performance model describes functions of a model and the timing properties of the functions. VCC first extracts the scheduling information from behavior model. Then based upon mapping, it counts on the timing properties of each module and gets the performance of the whole design.

Performance models decide the accuracy of performance analysis. VCC gives a complete structure for performance modeling, but it doesn't help designers estimate the fundamental timing properties of each modules. Designers are asked to give timing information themselves and by whatever means. For the IP cores and customized modules, this won't be a problem; but for many yet unavailable modules, timing information is hard to gather accurately. That inaccurate information finally affects the result of performance analysis.

4 First generation dilemma

The above methodology facilitates platform-based design and derivative design by emphasizing the central idea, which is to achieve performance constraints through mapping adjustment and architecture model refinement. It assumes that designers either have enough IP cores or customized modules for a design or have previous designs. But for many first generation designs, there are only some IP cores and a few customized modules available at the beginning, and most modules will be designed or purchased after several critical design

decisions are made. This poses a problem to the platform-based design. On one hand, at beginning, there are no enough IP cores and customized modules to show accurate performance of an architecture model. On the other hand, designers depend on the performance analysis results to make decisions to design customized modules and choose or purchase new IP cores. We call this problem the **first generation dilemma**. The dilemma reveals that platform-based design is not purely choosing and mapping to architectures and IP cores. Architecture design is still needed in some circumstances.

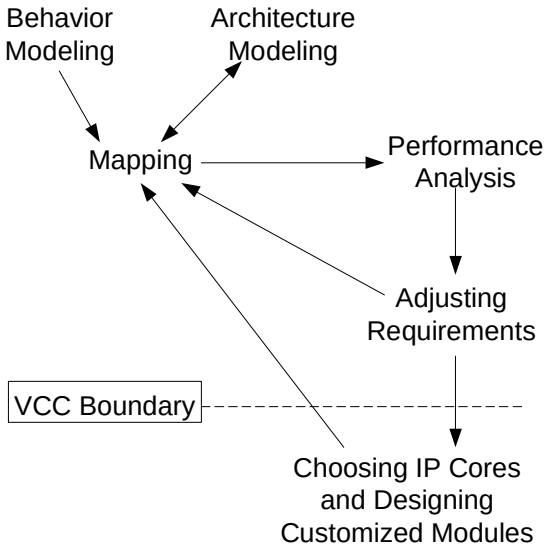

Figure 3: Revised platform-based design flow.

We can overcome this dilemma by modifying the platform-based design methodology. We need to change the central idea of the methodology. Rather than trying to achieve performance constraints through mapping adjustment and architecture model refinement, designers should find out what the performance constraints on the whole system means for new modules in a particular mapping and architecture model. To do this, first designers should try to decide an architecture model and a mapping, then through performance analysis designers can convert the performance constraints on the whole system to requirements for every single unavailable module (figure 3). Those requirements must be first screened by designer's experience before the next step. Any unrealistic requirements must be gotten rid of by either mapping adjustment or architecture revising. Based upon the requirements, designers can either purchase new IP cores or design customized modules. If some modules can't be realized because the requirements for them are too strict, designers need to adjust mapping or revise architecture model. To save the designs of modules that satisfy the requirements, they should be treated as the available customized modules without constrain the requirements. Extra modules may be added to relieve those too strict requirements. Then a second set of requirements can be found through performance analysis. This modification is compatible with the original design methodology, and it solves the first design dilemma. In this solution, system-level design will be a lot of easy, if designers have performance models in their libraries before purchasing any IP cores and they can use them in system-level performance analysis.

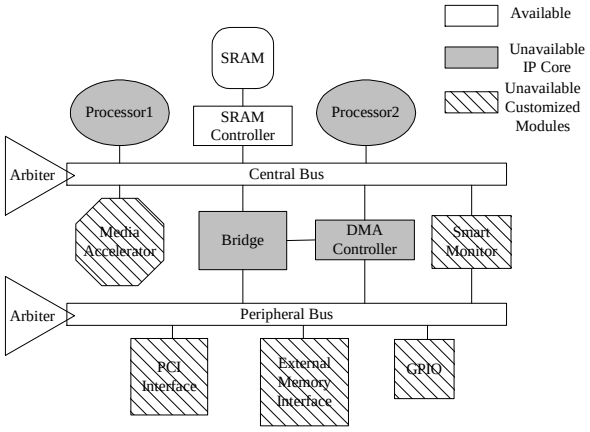

Figure 4: A multimedia embedded chip.

An example is the design of a multimedia embedded chip. We've already captured the details of its specification, and there are only several modules in our library. First, we design the architecture showed in figure 4. The white modules are in the library. We are going to use some IP cores for the dark ones and design the shadowed ones. Second, we initiate a performance model for each unavailable module to capture requirements on them. Then, performance analysis results are used to adjust those performance models. After several iterations, a set of performance models, which carry requirements on each module, is used to choose IP cores and design customized modules. During designing the media accelerator, our design cannot meet the requirements. We go back to adjust the performance model for the media accelerator and redirect some workload to a processor. After the adjustment, performance analysis shows the whole design meet the performance requirements. Without overhauling the media accelerator, we accomplish the chip design.

5 Conclusions

Platform-based design subdivides HW/SW partitioning into behavior modeling, architecture modeling, and mapping. In this way it supports IP reuse, PBD, and derivative design. Behavior model captures the details of design specifications and is used both in software design and hardware design, but the choice between a software-

oriented behavior model and a hardware-oriented one is difficult. Early performance analysis helps designers make design decisions at architecture-level, and this could save a lot of efforts and time for designers; but the accuracy of performance analysis depends on timing information given by designers. System-level design tools can provide structures to fill the timing information in, but it can't help much to guarantee the accuracy, and a timing mistake could affect the whole design. So to benefit from platform-based design designers must be very carefully to estimate timing properties of modules. System-level design tools must closely cooperate with other tools to complete a design. System-level tools can only generate gate-level result and do functional verification. So other tools are needed to do software refinement, hardware refinement, and system verification. So carefully choosing tools to cooperate with the system-level design system is important for a design.

Platform-based design methodology has difficulties to deal with the first generation dilemma, because it is optimized for IP reuse, platform-based design, and derivative design. We pose a modification to the methodology to solve this problem.

6 Acknowledgments

This paper was improved due to the valuable comments from reviewers.

References

[1] S. Tsasakou, N.S. Voros, M. Koziotis, D. Verkest, A. Prayati and A. Birbas, "Hardware-software co-design of embedded systems using CoWare's N2C methodology for application development", *Proceedings of IEEE International Conference on Electronics, Circuits and Systems*, 1999, pp. 59-62.

[2] www.innoveda.com

[3] A.P. Nash,G. Freeland, T. Bigg, "Practical W-CDMA receiver and transmitter system design and simulation", *1st International Conference on '3G Mobile Communication Technologies*, 2000, pp. 117-121.

[4] http://www.synopsys.com

[5] A. Sangiovanni-Vincentelli, G. Martin, "Platform-based design and software design methodology for embedded systems", *IEEE Design & Test of Computers*, 2001, pp. 23-33.

[6] K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2000, pp. 1523-1543.

[7] Jeroen A.J. Leijten, Jef L. Van Meerbergen, Adwin H. Timmer, Jochen A.G. Jess, "Prophid: a platform-based design method", *Design Automation for Embedded Systems*, SEP 2000, pp. 5-37.

[8] Masamichi Kawarabayashi, Jin-Qin Lu, Kazunori Goto, Patrick W. Fung, "System level design methodology for System on a Chip", *NEC Research and Development*, JUL 2000, pp. 248-252.

[9] Cadence VCC2.1 manuals