The Future of High-Level Modelling and System Level Design:
Some Possible Methodology Scenarios

Grant Martin
Cadence Design Systems
2001 Addison Street, Third Floor
Berkeley, California 94704 U.S.A.
+1-510-647-2804

gmartin@cadence.com

ABSTRACT

We discuss the future of high-level modelling in the context of system-level design, as the two concepts are inextricably interlinked. This is described as several possible methodology scenarios for the future of system level design as it unfolds within the electronics industry. Although these scenarios are presented as orthogonal, of course in actual fact the future may present us several of these simultaneously. The key methodology challenges that prevent the widespread adoption of high-level modelling and system-level design will be discussed.

1. INTRODUCTION

The fate of high-level modelling within the electronics industry is inextricably intertwined with the fate of system-level design. System-level design has been discussed for many years, yet it has hardly become the natural starting point for the design process for most electronics design engineers. In the hardware domain, most design activities for individual designers start at the RTL level with a textual specification passed from on high. In the software domain, most design activities start with bashing C code out in an integrated development environment.

In fact, system-level design, as practiced today, has many of the attributes of a cult. Arcane notations, obscure mathematical jargon, the prevalence of gurus and acolytes, robes and sandals, weird incantations and self-flagellating rituals are all attributes of system-level design shared with cult religions. The EDA industry would like to break system level design, and high level modelling, out of its cult-like status, primarily to grow the EDA industry and revenues, but has not yet found the appropriate magic formula to do this. A good survey of the current state is found in [1].

We can analyse the future possibilities for high level modelling and associated system level design along six orthogonal scenarios or predictive axes. Although these are presented in an orthogonal fashion, it is of course much more likely that the real future will contain a mix of these possibilities along with others as yet undreamed of. Thus we could view these as six 'corner cases' against which we can test our hypotheses on system level design.

These scenarios can be briefly summed up as:

1. The Cult scenario, in which high level modelling and system level continues to be the preserve of a few gurus and acolytes. Here ritual replaces results.
2. The Niche scenario, in which high level modelling and system level design has successes in niche design domains only – for example, dataflow algorithms, or the design of finite state machines. This is the current predominant state.
3. The Platform-Based Design scenario, in which almost all electronics product design is based on developing derivatives of complex fixed and reconfigurable, application-oriented platforms provided by a number of suppliers.
4. The Hardware scenario, in which design moves radically from software and soft-designed reconfigurable logic to more efficient hardware implementations.
5. The Software scenario, in which almost all design is done by developing software variations on a few fixed platforms (which may also contain reconfigurable logic, but designed in a 'soft' way). This is a close variant of the third scenario.
6. The Optimistic System Level Design Scenario, in which most design engineers do indeed move up in abstraction to the system level, and design involves the building, modification and reuse of high-level models in an overall system design methodology, with appropriate design flows through to implementation.

We will describe these scenarios in more detail, especially with respect to their methodology implications and implications for wider adoption, and then conclude with some ideas about the likely future of high level modelling and system level design.

Several driving factors or enablers will have a big impact on the unfolding of the future. The relative importance of design drivers will have a large effect on determining which scenario(s) emerge. These drivers include: cost, which will drive towards platforms; energy, which will drive towards dedicated hardware implementations; schedule, which will drive towards platforms or software; and manufacturing cost, which will drive towards dedicated hardware. Different applications will likely emphasise different drivers; thus we may not see one scenario prevail.

Whichever scenario(s) prevail will have a big impact on the state of modelling and system design tools. The current state is discouraging. As indicated in [2], "electronic system level is becoming an issue. As more design work moves up to this level, design engineers are finding the available tools inadequate."

2. THE CULT SCENARIO

In the cult scenario, system level design is the preserve of a few high priests or gurus, and their acolytes. High-level models are created in a variety of modelling languages and notations, to be used by a few system architects in making key architectural design decisions. However, the high level models are surrounded by what, to ordinary designers, are obscure mathematical notations: "models of computation", "formal methods", "denotational semantics", "satisfiability", and the like. These surround the models with an air of mysticism that many designers find impenetrable. In addition, in this scenario, the models do not "flow" – that is, there is no well-defined design flow or methodology in which the models can be either re-used at lower levels of design abstraction, or transformed into designs which can be implemented using automated tools, or generate useful artefacts which are valuable downstream. As a result, high-level models of systems remain perpetually disconnected from the activities and methods used by the bulk of those implementing the system; they grow out of date and are soon discarded if used at all.

The net result of the Cult scenario is that high-level modelling and system level design remain the property of the cult and never propagate into the design mainstream.

3. THE NICHE SCENARIO

The niche scenario is actually the prevalent best practice current usage of high-level modelling and system level design today. In particular design domains, for example dataflow algorithm modelling and implementation, or the design and implementation of finite state machines, some high level modelling/capture notations together with associated tools, methodologies and design flows have achieved reasonable success.

For example, in the dataflow algorithmic area, tools such as Ptolemy and SPW have been used to build complete methodology flows moving from high level models through to verified silicon implementations [3]. The high level model consists of a dataflow block diagram using blocks drawn from various libraries together with user defined blocks. Block functions are captured in a structured form of C++. The verification environment may draw on standards-based models of various wireless, wired communications and multimedia applications standards. Once a dataflow algorithm is captured in a high level model, such tools and methodologies allow co-simulation with embedded SW running on control or DSP processors; they also allow structured refinement of the algorithm from an untimed dataflow model to a clocked HDL implementation, using the system model as a golden reference model to validate an HDL-based implementation. Generated HDL code can be synthesised, laid out and validated using standard RTL to silicon design flows. The relative age of [3] – 1997 – indicates that this kind of niche design flow based on high-level models is well proven.

The problem with the niche scenario is that high-level modelling and associated system design tools and methods remain ghettoised in their niches. These specialised flows are the preserve of domain experts and can hardly be expected to appeal to a mass market of designers. Indeed, one can argue that such markets are saturated by today's tools and libraries.

4. THE PLATFORM-BASED DESIGN SCENARIO

Platform-based design has been written about with increasing frequency in recent years [4], despite some considerable controversy about its definition. Here I define a platform as a co-ordinated family of hardware-software architectures developed to promote high levels of re-use of hardware and software components in the rapid, low-risk design of application-oriented derivative products. These could take the form of System-on-Chip (SoC) or more complex electronic systems, and the platforms will be offered by a number of different vendors working in various product application domains, in the form of both relatively fixed platforms and ones incorporating reconfigurability.

In this scenario, platform-based design succeeds in capturing most of the electronic product design space. We further hypothesise that most designs will be relatively straight-forward derivatives, based on configuring platforms, on selecting HW and SW IP blocks from well-characterised libraries, and possibly by mapping some functions into reconfigurable logic. The design approach will be primarily based on configuration, and on "soft" design – either into a SW form or programming of reconfigurable HW.

In this scenario, then, where is high level modelling required? No doubt the platform creators would need to do some considerable modelling to support their trade-off analysis – i.e. the choices of which application functions to map into SW and which into HW. The community of platform creators will form in its own way a relatively small 'cult' of gurus and system architects. However such modelling may continue to be informal, using spreadsheets, hacked-up C/C++ models, and a variety of ad-hoc analysis methods that do not require much formalism about high level modelling. Most of the platform creation could continue to use RTL based design flows and methods, just as they do today.

Most derivative design will be done using simple platform configurators, and software programming. Verification of derivatives may well be done using HW-assisted rapid prototyping and emulators, given that the communications schemes within the platforms will be well-proven and most IP blocks will be pre-wrapped and validated to interface correctly to the on-chip or in-system communications architecture. Thus the need for elaborate system level design space exploration using high level models will be greatly reduced, if not eliminated entirely, in this platform-based scenario.

If the complexity of platforms increases, of course, then system level design space exploration may start growing in importance, both for platform definition and for derivative design. This will drive high level modelling and system design tools to emerge.

5. THE HARDWARE SCENARIO

This scenario is an interesting one because it calls designers to abandon their infatuation with software, and 'soft solutions' such as reconfigurable logic, and return to the true faith of hardware design and implementation. One leading exponent of the hardware approach is Bob Broderson of UC Berkeley and the Berkeley Wireless Research Centre (BWRC). As he correctly pointed out [5] at the Design Automation Conference in 2000, "Software architectures are at least 100 times less efficient in power and area than hardware. That gap will increase." In this

view, there is no "hardware-software codesign", because with at least two orders of magnitude difference in power and area between hardware and software (and often at least 10-100X in performance as well), there is nothing to trade-off. The advantage of direct hardware implementations of complex algorithms, vs. time-multiplexing central processing resources, are clear.

The natural methodological implementation of this scenario is an increasing move towards some kind of behavioural synthesis, or direct mapping (as in the BWRC work) from a natural high level description of the algorithms involved in a design, to hardware implementations. Relative inefficiencies of behavioural synthesis vs. RTL-level synthesis of hardware may become less important as we move to deeper and deeper submicron processes. However, the correct notation for high level modelling becomes vital in this scenario. Classic sequential programming languages are not at all appropriate as they lose the natural concurrency in computation and communications, which algorithms, especially dataflow processing, possess, and which are naturally implemented in hardware. This motivates a variety of algorithmic notations, from block-oriented dataflow networks as in Matlab, SPW and other tools, to programming languages such as C/C++ enhanced with natural means for expressing concurrency (e.g. SystemC, SpecC).

In this kind of mapping or synthesis flow, the high level model IS the high level design and the desired tool flow is total automation from high level algorithmic capture to optimised hardware implementation. Although such scenarios have begun to make progress in the signal processing and dataflow domains, control-oriented behavioural synthesis has much progress still to make.

The primary problem with the hardware scenario is that it assumes a return to application-specific IC (ASIC) design despite the growing mask costs and NRE. Given that ASIC design starts are dropping, the desire for many derivative products to be designed, and NRE and mask costs projected at one million dollars or more for 100 nanometre or smaller process technologies, it is unclear if this 'return to hardware' can remain any more than an academic pipe-dream.

6. THE SOFTWARE SCENARIO

The Software scenario is predicated on a future in which software is the primary means for product differentiation and design. The actual implementation form for 'software' can be in embedded software running on various processor(s), and in reconfigurable logic, which is defined, designed and implemented, in a 'soft' fashion. As the 2001 CANDE meeting concluded in one of its infamous predictions, "hardware/software co-design is 'irrelevant', and … the real problem is one of mapping system functions into programmable resources" [6]. The software scenario is actually a variant of the Platform-Based Design scenario, in which the platform is relatively fixed, as discussed earlier.

We can counterpoise the arguments used for the Hardware scenario – that hardware implementations of functions are 1-2 orders of magnitude better in power, performance, and cost (silicon die area) than software ones, with what one might call the 4 "M's" that favour the software approach. Hardware implementations will cost 1 Mask set, 1 Million U.S. dollars for NRE, 1 Month for manufacturing; and SW offers far better Management of risk, in that mistakes can be much more easily corrected in the field. In this scenario, all these factors combine to steer the design community increasingly towards software-based solutions for product differentiation.

Even if high level modelling is useful for SW design and implementation, it is likely to be a more traditional SW-oriented high level modelling language and notation. SDL, and UML seem the most likely approaches to achieve wide favour in the SW-dominated world. UML, with its multiple modelling notations and ability to be extended through profiling, seems the most likely high-level notation to be used. Interestingly, use of high level models for SW design remains a distinct minority taste at this time (2002) – for example, almost all use of UML is for high level graphical documentation of basic software architecture, and use of UML and SDL based code generation is quite low. Most embedded software is created using hand-coded and optimised C or assembler, with some use of Java and C++, but following classical capture-compile-run-iterate development methods.

For battery-powered, handheld devices, energy consumption is a dominant design issue. SW is inherently much less power efficient than hardware, as discussed in the previous scenario. The relative importance of battery life and power consumption may have a large impact on whether the hardware or software scenario, or some combination, is more likely.

7. THE OPTIMISTIC SYSTEM-LEVEL DESIGN SCENARIO

We've had enough of pessimistic views of high level modelling and system level design in our first five scenarios. Let's return to first principles and discuss what an optimistic and feasible vision for high level models and system level design might be.

In this vision, as described at many times and places [7,8], high level models of subsystems become the new lingua franca of design, not just by a cult, but by the mass of designers who move upwards in abstraction level because to do so gives them distinct advantages in the design process: productivity, time to design, and lowered risk. This can involve many technologies, for reuse, synthesis, automated code generation, formal verification, etc., which together with appropriate tools and methods, form an integrated design flow from system specification through to optimised implementation. High level modelling works with all design styles, from customised hardware implementation through platform design and configuration through to software-based methods. High-level models permit design space exploration – the appropriate amount of trade-off analysis and optimisation of design over alternative mappings and configurations [9]. High level models and system design tools hide the mathematical complexities of models of computation and systematic composition as well as formal methods, presenting design interfaces which are tractable and understandable by a wide community of users.

But what are the barriers to adoption of this more optimistic scenario? They are several, and they are not new.

7.1 A Lingua Franca

First, modellers and designers require a common 'lingua franca' for design modelling. This can take two forms:

- A small set of common well-defined languages and notations within which most of the semantics of design for various application domains can be expressed. For example, extensions to C/C++ such as SystemC and SpecC may fit here.
- An underlying compositional theory which allows models captured in different 'models of computation' to be combined into an overall system level model without either ***losing*** important aspects of the system specification (due to inappropriately basic compositional theories) and without ***confronting*** the designer with the underlying mathematical theories.

With such a well-defined lingua franca, models become *reusable* and *interoperable*, two vital characteristics for designers. If the investment in system level models is not reusable, the effort expended in building them is wasted and designers will be incredibly reluctant to build and validate them. If there are competing toolsets and the models are not interoperable between them, then again designers will be reluctant to be tied to one set of proprietary tools, even if the models are theoretically open. In this regard, SystemC seems the closest to a relatively open, widely adoptable and popular high level modelling language based on extensions to C++.

Another important note is that much of the common substructure for high level modelling need not be directly written or read by human beings. It is sufficient that tools can read and write the language(s) and present usable views of the models contained within to designers. For example, C++ is hardly an elegant, terse or formal language, yet its ubiquitousness makes it an obvious choice for a system design language. In my view, human beings need to be well insulated from the more ridiculous syntax and semantics of C++ and extensions such as SystemC. This opens up tremendous possibilities for appropriate methodologies, high level modelling standards, and tools, to be developed, that insulate designers from details which stand in their way of expressing system design intent in a natural and reliable manner.

7.2 A Consensus on Methodologie(s)

Although it is not necessary that only one methodology prevail in the system level design space, it is important that enough designer interest coalesce around a very few critical methodologies to create a critical mass of design interest that will attract the commercial EDA industry to create compelling tools to support it. Just as the RTL to place and route flows in IC design have become the overwhelming method for design of digital blocks and chips, to be used in custom, ASIC and SoC implementations, similar methods must be defined and adopted in the high level modelling and system design world. We addressed above the need for a common lingua franca for modelling – this defines as it were the common ***syntax*** for high-level design. Consensus methodologies define the common ***semantics*** that permit tools to operate in a defined flow – so that abstract models defined using tool X will be meaningful for analysis by tool Y.

This consensus on methodology might arise due to a single vendor/tool arising and setting the common methodology standards in high level modelling, or through pan-industry standardisation efforts, or through other mechanisms. But it is important to reach a consensus and reduce the fragmentation that is a key characteristic of today's system level design approaches. In this sense, consensus on verification aspects of the methodology is as important as consensus on design abstraction levels and appropriate semantics.

7.3 Flow Across the Systems to Implementation Divide

We discussed above the need to reuse models in system level design in order to justify their creation. It is also essential that high level models have reuse across the implementation process, either as inputs for automated synthesis of HW or SW; as verification environments or golden models at lower levels of abstraction; as guides for template-driven generation of implementation, or as refinement steps in a well-defined methodology, supported by tools, that moves designers from specification through to implementation.

It is the lack of use of these high level models in the past that have restricted modelling to the 'cult' status [8]. The 'gap' between high level and implementation designers must be crossed in order to establish a real design flow from one level (or multiple levels of system abstraction) to another (the RTL-C level of implementation).

We can hardly see that the world has yet achieved this state. High-level synthesis is itself a niche possibility. More comprehensive tools in this area are restricted in use to a very few design groups.

8. THE LIKELY FUTURE

I discussed at the beginning the fact that the likely future will contain a combination of scenarios. In fact, high level modelling is not really a cult, given that the niche scenario is widely used and adopted. The hardware scenario is unlikely to return, and yet designers will continue to want to optimise hardware implementations for some designs and portions of design platforms. Thus there are two likely futures we can predict:

1. A combination of the Niche and Platform-Based design scenarios. Here, use of niche high-level models in domains such as dataflow and state machines will continue with incremental improvement. Platform-based design will grow in usage, and methods will be found so that most derivative design can avoid high level modelling and system design approaches.
2. The fulfilment of the optimistic system level design and high level modelling scenario, which subsumes all the other scenarios. This is the true 'Radiant Future' of system design to which we aspire. But to realise this future, the "integuments must be burst asunder".

Is this second scenario the likely one? Enquiring minds would like to know…and fervently hope so. One point of setting up a group of straw men, and then knocking them all down, is to convince the readers, and listeners, of both the inevitability and desirability of the future which is left standing…

What actually will happen in this future will depend on the key properties and system level drivers which must be optimised. Again, if different application domains drive us in different directions, the likely future will consist of a combination of scenarios.

9. ACKNOWLEDGEMENTS

I would like to thank Frank Schirrmeister for his insightful feedback and suggestions for improvement, and Roberto Passerone for his review of the paper.

10. REFERENCES

[1] Gabe Moretti, "System level design merits a closer look", *EDN Europe*, February 2002, pp. 22-28.

[2] Gary Smith, Daya Nadamuni and Wei Tang, "ASIC Design Times Spiral Out of Control (Executive Summary)", *Gartner Dataquest Report* 103694, January 30, 2002, p. 2,

[3] Frank S. Eory, "A core-based system-to-silicon design methodology", *IEEE Design & Test of Computers*, Volume 14, Issue 4, Oct.-Dec. 1997, pp. 36-41.

[4] Alberto Sangiovanni-Vincentelli and Grant Martin, "A Vision for Embedded Systems: Platform-Based Design and Software Methodology", *IEEE Design and Test of Computers*, Volume 18, Number 6, November-December, 2001, pp. 23-33.

[5] Chris Edwards, "Panel weighs hardware, software design options", *EE Times*, June 8, 2000. URL: http://www.eetimes.com/story/OEG20000607S0043

[6] Richard Goering, "Cande charts EDA's future", *EE Times*, October 3, 2001. URL: http://www.eetimes.com/story/OEG20011003S0071

[7] G. Martin and B. Salefski, "Methodology and Technology for Design of Communications and Multimedia Products via System-Level IP Integration", *Designer Track of Design Automation and Test in Europe (DATE-98)*, Paris, France, February, 1998, pp. 11-18.

[8] G. Martin, "Design Methodologies for System Level IP" ", *Design Automation and Test in Europe (DATE-98)*, Paris, France, February, 1998, pp. 286-289.

[9] E.A. de Kock, G. Essink, W.J.M. Smits, R. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers, "YAPI: application modeling for signal processing systems", *Design Automation Conference*, 2000, pp. 402-405.