Design Methodologies for System Level IP

Grant Martin

Cadence Design Systems, Alta Business Unit

Abstract

System-chip design which starts at the RTL-level today has hit a plateau of productivity and re-use which can be characterised as a "Silicon Ceiling". Breaking through this plateau and moving to higher and more effective re-use of IP blocks and system-chip architectures demands a move to a new methodology: one in which the best aspects of today's RTL based methods are retained, but complemented by new levels of abstraction and the commensurate tools to allow designers to exploit the productivity inherent in these higher levels of abstraction. In addition, the need to quickly develop design derivatives, and to differentiate products based on standards, requires an increasing use of software IP. This paper will describe today's situation, the requirements to move beyond it, and sketch the outlines of near-term possible and practical solutions.

1. Introduction: System-Chip Design Today

If we classify the design process for system-chips and chipsets into four stages: behaviour, architecture, assembly and manufacturing (see Figure 1), then the current design methodology encompasses the assembly and manufacturing stages. Assembly can be defined as starting at the RTL capture/RTL-level floorplanning level of design, on the HW side, and writing C and assembly code on the SW side.

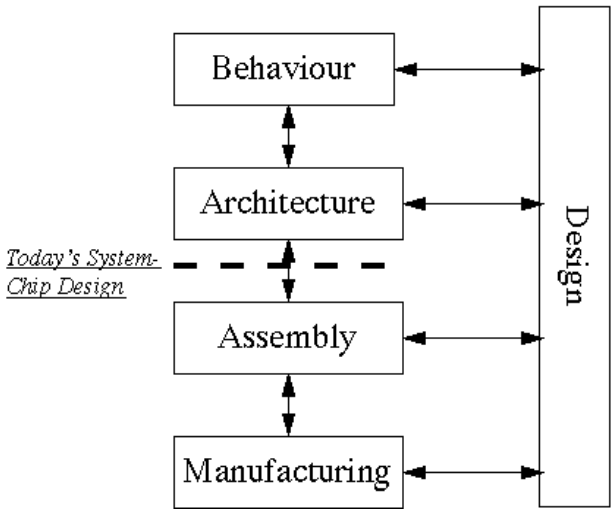

Figure 1: Stages of System Chip Design Process

Today's design methodologies vary widely. Although many design teams use HDL-based RTL-level synthesis extensively, others, for example in Japan, have just started in the last year or so [3]. Design re-use is limited by methodology: the Virtual Socket Interface (VSI) Alliance was formed in September 1996 to foster the development and recognition of standards which will enhance designers' capability to create and integrate re-usable blocks of IP. [1,5] The VSI Alliance is now one year old, and despite its desire to endorse existing standards activities in preference to creating new ones, re-use of synthesisable, soft RTL cores is still difficult. Re-use of hard layout cores is less flexible than soft cores, in that they have already been committed to a particular IC manufacturer and process; this has led to the idea of a "firm" IP block or Virtual Component, although this concept has not yet been fully developed.

Whatever the re-use and system-chip design paradigm, verification of design intent and implementation continues to be a problem, which is compounded for embedded products which have a significant Software component. Today, HW-SW co-verification is hard. HW-SW Co-design is essentially a collection of manual techniques, as described in [2].

In addition to the re-use and verification issues, progress in IC technology is forcing a move to the linkage of logical and physical design. Successful system-chip design processes include early floor-planning and forward estimation/prediction of performance effects (timing, power, cost/area), and the propagation of the resulting design constraints into synthesis and layout, in order to control the timing and power outcomes of the logic synthesis and layout stages.

2. Current Methodology Limits

System-chip architectures captured at the RTL-C level are hard to re-use and evolve. At the RTL level, architectures must be fully articulated or elaborated, with all signals instantiated on all blocks, all block pins defined, and a full on-chip clocking and test scheme defined. As well as a complete definition of each block's functionality, architectural designs at RTL-level have completely defined communications mechanisms. This makes it very hard to change the on-chip control structures and communications mechanisms between blocks. Because of the need to "rip-up" and "re-route" the communications mechanisms, and to re-wire any new or substituted functional blocks to the communica-

tions structures, an architectural change is very time-consuming; thus it is almost impossible to effectively explore the architecture space.

A corollary effect is that it is difficult to "drop-in" or substitute a new IP block choice. Dropping in a new microcontroller core requires detailed "rip-up" and "re-route" to link the block to the communications structure, making it hard to explore the space of available IP effectively.

Designs captured at the RTL level mix both behavioural and architectural design together. Often the only model of IP block function is the synthesisable RTL code which represents the implementation of the function. Similarly, the only model of a SW function may be the C or assembly language implementation of the function. This "intertwining" of behaviour and architectural components together makes it extremely difficult to evolve the behaviour of the design and its architectural implementation separately. If a design needs to conform to a particular standard which is evolving, or needs to be modified to conform to the next generation of a standard, the RTL/C level design which entangles both behavioural intent and implementation is a very clumsy and difficult representation to work with.

Finally, verification of embedded HW-SW designs at the RTL level requires:

- nearly-complete hardware design
- nearly-complete Software code for the hardware interface (drivers), part of the RTOS, and if the behaviour of the system is to be verified, the layered application(s)

Co-verification with today's ad hoc and emerging commercial tools at this level is slow [2,4]. It is clear that one cannot verify complete system application behaviour in an HDL/C simulation environment.

During such co-simulation, if major application problems are found, a time-consuming and tedious redesign process is required to repair the design. Re-partitioning is very difficult as the communications infrastructure will require detailed redesign. Substitution of better programmable HW IP blocks (new processors, controllers) or custom HW accelerators for part of the SW implementation requires significant changes to application SW.

The net result of today's limited methodology is that system partitioning and design is often done with manual, back-of-the envelope techniques, and carries a high risk of major problems emerging during downstream implementation and integration.

3. Where would we go by evolving current methodology?

As today's RTL-level tools and methodologies evolve, we will see more up-front system and chip design planning, better forward prediction of physical effects of layout (so that these effects can be incorporated into up-front design planning), and more robust HW-SW co-verification.

RTL-level, top-down floorplanning is emerging. It is important to remember, however, that not all IP blocks will be re-used at the RTL-level. Re-use of "hard" (essentially, laid out in an IC process), and "firm" (cell-level netlists) blocks will increase, and for many reasons, vendors of large complex programmable IP cores may prefer distribution to their end-customers of their blocks in these formats as opposed to synthesisable, "soft", RTL code. However, RTL-level floorplanning will give better control over the physical effects determined during the synthesis process, and will allow a reduction in the number of iterations required to converge on a feasible design.

Continued work within VSI will eventually result in a near-complete IP block interchange standard covering the "soft", "firm" and "hard" domains, from RTL level down through physical layout.

However, such an evolution of methodology will not touch some of the major limitations enumerated above:

- *architectures* will still be hard to re-use, and evolve
- it will still be difficult to explore IP block *alternatives*, especially programmable ones
- *verification* that an architecture and design will work for an application will still pose considerable difficulties.
- the *behaviour* and *architectural implementation* for a design will still be intertwined and difficult to *evolve separately*

4. Can we break through the "Silicon Ceiling"?

Providing solutions for these major limitations in today's methodology and tools requires a breakout from concentrating on chip-level design at today's RTL-C level - the "Silicon Ceiling".

Breaking through the ceiling requires a move to higher levels of design abstraction: the "architectural" and "behavioural" levels (Figure 1). Abstraction is required in 3 key areas:

- architectures
- models
- design exploration and verification

Architectural abstractions must be easy to capture, evolve and change. This requires that such abstractions must remove the fully elaborated detail that is not necessary for first- and second-order architectural exploration and evaluation.

One cannot explore architectural and IP choices with detailed cycle- and pin-accurate simulation models. They are fundamentally too slow to execute, and too difficult to easily manipulate during exploration. Articulated HDL-based signal and event-driven simulation, whether just used

for HW validation or as part of HW-SW co-verification, is too slow to allow the validation of system-level behaviour for embedded system-chip designs, or to allow the exploration of architectural and IP alternatives. The appropriate abstraction level is the use of *performance analysis techniques* to make first and second-order architectural trade-offs.

5. Abstractions

First-order architectural trade-offs don't need fully articulated architectures to be captured. Instead, the minimum requirement is to capture a "relaxed" view of the system-chip architecture. In this relaxed view, IP function blocks are instantiated with simple connections to abstract views of communications mechanisms. At the highest level of abstraction, communications can be described as moving frames, packets, tokens between function blocks over channels. The next lower level of communications abstraction is as a series of basic bus-transactions (acquire bus, bus-read, bus-write, burst-read or write), or SW communications methods (memory-mapped IO, semaphores, interrupts). [7]

The functional IP blocks can be classed into several categories: processors (control dominated or signal processing dominated), custom function blocks (e.g. MPEG decoders, filter blocks), memories, peripheral controllers, buses, etc. These functional IP blocks process tokens, frames, packets, or step through control and computational sequences under SW control. The basis system operation can be described by how fast blocks process tokens or run SW, and blocks transfer tokens to each other over communications mechanisms. The equations which describe block performance are called *delay equations*. Since shared resources (buses, processors) imply contention for the resource, this must be modeled in the appropriate *delay equations*, which need to invoke models of resource contention.

Making first-order architectural trade-offs does not require finely detailed models incorporating all signals, pins and HW details, and every detailed state transition or signal event to be simulated. Rather, system application behaviour can be modeled and verified at higher abstraction levels using faster simulation methods. A key technique suitable for making decisions is the use of architectural evaluation performance analysis.

6. Making IP Blocks Re-Usable

The current VSI definition helps in re-use of soft, firm and hard IP blocks. But a design methodology which focuses solely on RTL through hard layout makes IP re-use inherently hard. Abstraction makes IP block re-use and architectural exploration easier, so it must follow that: ***an effective IP re-use methodology will require abstract models of IP blocks***.

We define these abstract models to be a combination of

- architectural delay equations, appropriate for the class of IP block, and
- resource contention models for shared resources.

These models are tractable and can be constructed through developing characterisation methods appropriate to the classes of IP blocks.

SW IP represents an interesting class of IP. SW code is re-usable if it can be easily re-targeted. There are two essential kinds of SW IP:

- "Close-to-Hardware": RTOS, drivers, HW-dependent code, that is optimised to particular HW platforms, often written in assembly rather than a higher language such as C, and inherently hard to re-target
- "HW-independent": usually in C, and having adequate performance when kept HW-independent. For this software, re-targeting primarily requires an assurance that the SW will perform "adequately" on new target HW. We now have techniques (based on extensions to the SW estimation work in POLIS [6]), that permit this assurance to be derived not through detailed SW execution on a detailed HW model or actual HW, but by ***estimating SW performance automatically on target HW***.

7. Re-usable architecture

As functional IP blocks become easier to re-use, it is important to move on to the next step: the re-use of chip architectures. Architectures can be made re-usable if they have the following characteristics:

- simple to capture and modify: i.e., in a "relaxed" form, rather than a fully articulated form
- come with rich libraries of architectural IP components, from internal and 3rd party IP providers
- supported by central architecture groups, who create architectural derivative product design kits, containing reference architectures, IP block libraries and sample applications
- easy to modify system control and communications through use of abstract communications description and mechanisms for refining from abstract through bus-transaction through to articulated level: a methodology emerging as "Interface-Based Design" [7].
- easy to export to implementors of architectural derivatives. It must be possible to link architectural design to real HW and SW implementation flows, so that design information captured at the architectural level will be usable at subsequent design process stages.

8. Derivative Product Design: A conundrum

In today's embedded consumer communications and

multimedia products, original architectures created on a "blank sheet" are relatively rare. However, a base or platform architecture is often used to create a whole series or family of derivative products.

Derivative designs may rely on the same basic processor cores and communications buses, but:

- may vary peripherals depending on the application
- may add optional accelerating HW
- may do limited IP blocks substitution (for example, moving to a new micro-controller core which, via subsetting, is instruction-set compatible with the old core)
- will definitely involve significant SW changes, tailoring a global product for particular markets, adding special new User Interface capabilities, etc.

Since derivative design is a must for product variants and evolutions, they require rapid and risk-free design cycles. This presents a problem with an RTL/C-based methodology: at this level of abstraction, derivative design is either "hard" or "limited":

- if HW changes are needed, it's "hard". The design is fully articulated at the RTL level: it's hard to change blocks. Communications mechanisms are fully described at the pin and signal level - this makes it hard to add new IP blocks or modify the communications. The application SW must be re-validated, somehow, by ad hoc and slow HW-SW co-verification methods, and one cannot really be sure that the new design will work without exhaustive and exhausting regression testing. Furthermore, if programmable HW is changed, extensive SW changes may be required.
- if HW changes are too risky, or "hard", then a "limited" derivative design is the only one possible. Here one must only modify SW, and then must re-validate against HW using the ad-hoc and slow HW-SW co-verification methods. If one is always restricted to SW changes, then the architecture and partitioning cannot be optimised. If the new SW does not run fast enough, then the product will be infeasible.

9. A new methodology

To break out of the "Silicon Ceiling", RTL designers need a methodology and tool technology that supports:

- easy, rapid, low-risk architecture derivative design
- risk reduction for complex product designs: system designers must be able to develop key concepts at an abstract level and pass them to be rapidly implemented with little risk of architectural infeasibility
- abstract models that are easy to generate
- an ability to make first-order trade-offs and architectural and IP evaluations above RTL level
- real and effective HW-SW co-design and co-verification at a higher level: applications running on an architecture without the overhead of verifying complete details at the RTL level
- easy to create and modify on-chip communications schemes, and to hook up and change the interface of IP blocks to those mechanisms
- linkages to real HW and SW implementation flows

A description of new technology and methodology under development which has these characteristics can be found in [8].

10. Conclusions

Moving beyond today's RTL-based methodology for system-chip design requires the elaboration of a new re-use driven methodology and the provision of tools and technologies which support it. This paper has outlined the key characteristics of this new methodology, and discussed why they are necessary. Technology developments are currently underway which will allow designers to move through today's "Silicon Ceiling" to new levels of abstraction and design productivity through re-use and a focus on IP-based system-chip integration.

References

1. Virtual Socket Interface Alliance Architecture Document. V 1.0, March, 1997, p. 1. © VSI Alliance. See the VSI-A web site at URL: http://www.vsi.org.
2. G. Martin, "HW-SW Co-Design: A Perspective", *EDA Vision*, Volume 1, Number 1, October 1997 (on-line magazine: URL: http://www.dacafe.com/EDAVision/Issue1/EDAVision.1-3a.html).
3. Gary Smith, Dataquest, Private Communication, June 1997. Gary Smith notes that designers in Japan as of January 1997 are ramping up with RTL-level synthesis very fast, compared to 1.5 years earlier.
4. J. Rowson, "Virtual prototyping", *Proceedings of CICC 1997*, May, 1997, pp. 89-94.
5. M. Hunt and J. Rowson, "Blocking in a system on a chip", *IEEE Spectrum*, November 1996, pp. 35-41.
6. F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.
7. J. Rowson and A. Sangiovanni-Vincentelli, "Interface-based design". *Proceedings of the 34th. Design Automation Conference, 1997*, pp. 178-183.
8. G. Martin and B. Salefski, "Methodology and Technology for Design of Communications and Multimedia Products via System-Level IP Integration", *Proceedings of DATE 98*, February, 1998.