

## Imposing a Unified Design Methodology on Independent Rapid Prototyping Tools

Anders Hedberg, Hans Jacobson,  
Mats Einarsson and Glenn Jennings

Division of Computer Engineering  
Luleå Institute of Technology  
S-971 87 Luleå, Sweden

### Abstract

*The teaching of digital design requires a design platform which presents a coherent design methodology to the students. Yet we found ourselves with different tools and hardware subsystems, each having its own idea about what digital design was about. Here we record our effort to forge a single design platform for beginning designers from three separate software subsystems and two hardware subsystems. In the process we developed software which not only smoothed out the design flow, but also filled in some glaring gaps in the platform's functionality which had not been provided by any vendor. The insight gained from the exercise is reflected in a "wish-list" of features which we hope will be provided by vendors in the near future.*

### 1 Introduction

In this paper we record our practical experiences toward building up a teaching-oriented digital rapid prototyping facility for general-purpose synchronous circuits. It was an exercise in making at least three separate software packages and two separate hardware systems all appear as a single, top-down design environment to completely uninitiated users, namely, our beginning students in digital design. It was therefore of paramount importance to keep the "big picture" in front of us, rather than drowning ourselves or our students in distracting details and idiosyncrasies: these had to be suppressed.

There was an even more fundamental pedagogical

concern, which was to protect our students from being misled into thinking that they understood basic design principles just because they had learned a given set of design tools or design languages. As we worked with all of the various tools, all having beautiful color displays and endless options, we found this danger to be both seductive and acute. Even the authors, who have considerable collective design experience, were constantly tempted to ask "what next" rather than keeping the important questions "why" or "what is the purpose of this" in front of us. It is important for the reader to understand this pedagogical orientation, because it profoundly affected our implementation decisions and the items which we came to place onto our "wish-list."

Nevertheless we hoped that commercial tools could be used, partly because of the "reality factor" which we felt the students needed to be exposed to, and partly because experience with commercial tools increases the employment prospects of our graduates. So we chose the following components and subsystems. As our mainstake FPLD we chose AT&T's ORCA lookup-table-based family, even though it had only recently been introduced at that time. To provide a rapid prototyping interconnect we chose the Aptix AXB-GP4 field-programmable circuit board populated with four type "R" FPICs. For front-end capture, synthesis and simulation, we chose Viewlogic's tool suite as being "typical" of the genre. The ORCA components were supported by the ORCA Development System (ODS) from AT&T. To program the field-programmable Aptix interconnect, we needed the AXESS/FPCB Development System from Aptix.

Both the ORCA and Aptix hardware required host interfaces and cables which connected to our SUN workstations. This was the assembly of hardware and software which we sought to tame.

Some of our work was caused by our having to use first releases of support software. For example, we were using Version 1 of the ORCA software, which did not provide us with any support for logic synthesis under Viewlogic's *ViewSynthesis* tool. All of these early software releases contained their share of bugs. Even though future releases of the software may obviate the need for some of the steps we discuss below, yet by having had to do certain things ourselves we came face to face with what we *needed*, and from that we gained considerable insight into what we feel such tools *should* provide. It should be of interest to compare the experiences and wish-lists which we record in this paper, against what the tool vendors actually make available in the future.

## 2 Design Capture

We chose a design methodology in which we would describe larger combinational units as incomplete functions in a high-level language. For this we used VHDL, or perhaps we should write "Viewlogic's VHDL" since Viewlogic had libraries and embedded functions that were intended for interfacing to the synthesis tool. We made this VHDL work with the ORCA library (no synthesis interface to ORCA was provided with the software versions we were using) after a heroic hacking effort. However, after considerable disagreement among ourselves, we decided to make synchronizers explicitly visible to the students, which meant that as a first approximation we would not permit VHDL code which would emit synchronizers during synthesis. So, we removed latches and flipflops from the target gate library. It was not (primarily) an inertia to high-level synthesis which was behind this decision, but rather an objection to "black-boxism" which hid too many design principles from the students. We return to this shortly, as well as in Section 4 below.

Combinational units specified in VHDL were then interconnected using a graphical schematic editor. However we were frustrated in attempting to create simple RTL designs using the graphical tool, which required too much distracting logic and bit-level specification. For example, we found we had to give names to everything. We had to specify the widths of all busses explicitly since the tool could not reason out the

widths for itself.<sup>1</sup> There were no generic "wide components" whose width would automatically adapt to the width of the bus it was connected to, for example an " $n$ -bit wide flipflop" or an " $n$ -bit wide  $m$ -input multiplexer" where  $n$  and  $m$  were simply parameters which the tool could deduce for itself from the context. This was a real irritation to us, since we had been working with other RTL capture tools which had solutions to most of these problems [Jen91b] [BA92] [Jen91d]. Hierarchy was well-supported by the graphical editor, but the usual problems concerning functional hierarchy *vs* physical hierarchy raised their heads, as we discuss in Section 4 below.

Concerning the use of the synchronous synthesis capabilities of the synthesis tool, we had the following objections. First, at Luleå we have research interests in the use of transparent latches [Jen92b] [JJ94] [Jen92a] [JJ+93] [Jen93] so we wanted to teach our students to work with them. This is in fact one of the reasons why we selected the ORCA FPLD family in the first place. However the synthesis tool did not let us control how the resulting network would be synchronized, neither did it inform us about its clocking methodology or choice of synchronizers. This placed too great a distance between the students and the knowledge they need to master. Second, we discovered a totally unexpected feature in the synthesized netlists, in that the synthesis tool was asking for a tri-state buffer during the synthesis of incomplete functions. To our surprise we were face to face with an industrial example of something which Luleå has been addressing at the research level for over a year now [Jen95a] [Jen94] namely the widespread carelessness in defining what is meant by the "don't-care" value 'X' and the "don't-know" value 'X', that is, the need to *rigorously define which ternary logic system is being applied* [Hay86] [Jen95b] [Bra83]. We feared that the synthesis tool had taken the liberty of defining "don't-care" to mean *completely undriven*, which could cause downstream synchronizers to go metastable, CMOS circuitry to draw current heavily, and so forth. Therefore we refused to install the tristate gate into the technology mapping library. We were horrified.

---

<sup>1</sup>Subsequent tools could not discern these busses either, so we had to *add* the attribute "bus" to the busses as well!

### 3 Design Correctness

After the students had captured their design by using a combination of text and graphics, the only tools available for design validation were the simulators. In general, these could be applied to any circuit, and would produce results even in the presence of asynchronous constructs such as ring oscillators and cross-coupled NAND gates. This was true even of the “compiled” simulator included in our tool suite.

But this meant that the simulator had no constructive definition for what a “correct” synchronous circuit was [JJ94] and therefore the student was given no feedback as to whether he had *inadvertently* created a construct which was questionable for synchronous design. Therefore we had to write a separate program which accepted the resulting gate netlist and subjected it to topological analysis. We were surprised that no such tool was present in any of the packages we were working with. On the first day that the topological tool was complete (we named it ORCAMASH), the authors used a microprocessor as a test case design. This design had passed all simulations and subsequent mapping steps, yet when downloaded exhibited a failure during one particular instruction. ORCAMASH discovered a number of questionable timing constructs [BA92], several redundant synchronizers [Jen92b], even a NAND gate whose output was coupled to its input. The offending constructs were removed, and the microprocessor worked.

This same topology-checking design tool became an invaluable “glue” tool in our design flow, see Figure 1. For example, a design coming from the Viewlogic environment was required (by the ORCA software) to have pad buffers installed at the primary inputs and outputs. Since we did not always wish to commit a given Viewlogic design to be our ORCA chip boundary (the hierarchy problem again, see Section 4) we instead passed an uncommitted Viewlogic design into ORCAMASH, which easily identified the primary inputs and outputs and then *automatically* installed the required buffers as the design moved from Viewlogic to ORCA mapping. Other uses for ORCAMASH were discovered as we began to interconnect multiple FPLDs together with other standard components at the Aptix FPCB level. These will be discussed in Section 5.

Regarding simulation, we also found that the performance difference between the discrete-event scheduled simulator and the compiled simulator was surprisingly small, whereas we had expected the compiled simulator to be up to two orders of magnitude faster than the event-driven simulator [Jen91a] at the

expense of losing the transient waveforms prior to circuit quiescence. We do not yet fully understand these results, but they made it difficult to motivate to our students when they should choose one method over the other. We had hoped that the students could clearly understand the differences and tradeoffs among various simulation methods, so that they could gain some sophistication in choosing design tools.<sup>2</sup> Also we found that the ORCAMASH topological analyzer was an opportunity for us to experiment with logic simulation of our design. Luleå’s recent results in ‘X’-correct ternary simulation for example [Jen95a] [Jen94] could be implemented in ORCAMASH, and emitted as a compiled simulator. This is still being investigated.

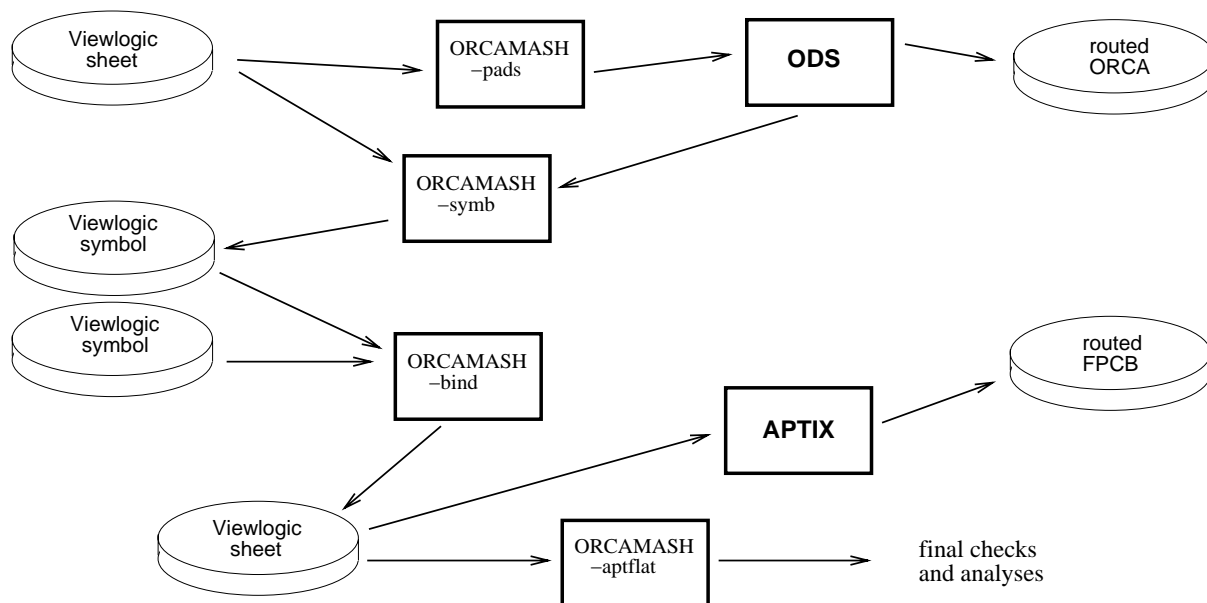
The point is that we were finding that ORCAMASH had in fact made us independent of our original choice of Viewlogic as synthesis platform and graphical front-end. For example, an interface from the GRTL RTL design tool [Jen91b] (with its innovations of self-deduced wide-component widths and bus widths [Jen91d] and of reversible simulation [Jen91c] [Jen92c]) to the ORCA and APTIX environment is now under development. This involves simply having ORCAMASH accept a topological netlist from GRTL instead of the netlist it presently accepts from Viewlogic. Other front-ends such as BBDS [BA92] or any other tool which emits a netlist of primitive components mappable to ORCA primitives [B<sup>+</sup>86] [SS<sup>+</sup>92] should be likewise relatively simple to adapt into our design environment. From a pedagogical viewpoint this is, of course, to our delight.

### 4 Partitioning over Multiple FPLDs

A problem which has long plagued VLSI designers is still with us, namely that the functional hierarchy used by the designer to reason about his design does not always correspond to the best physical hierarchy for laying out that design. In our case, a big design would be captured in a Viewlogic hierarchy which would reflect functional divisions. But to partition the entire design over several FPLDs, each and every Viewlogic sheet may need to be “cut,” that is, without regard to the sheet hierarchy. The only solution we had was to exercise forethought, compromising the functional hierarchy so that it could serve

---

<sup>2</sup>We are humbled by an anonymous referee, who writes: “[The authors] expressed disappointment that the similar performances of event-driven and compiled simulation cost the students an important lesson in choosing design tools. It sounds like they *did* get a good lesson! And so did the instructors!”



**Figure 1: The overall design flow of our system. ORCAMASH acquired an important “glue” function in making the various tools work smoothly together.**

as the physical hierarchy *at the same time*. We feel this interferes with the productivity of the designer, who already has enough simultaneous constraints on his mind as it is. We place the *automatic partitioning problem* very high on our list of things urgently requiring an effective solution. Although we had access to such a tool (which at the time did not support the ORCA FPLD) yet we felt there was more to be done: there are other cost functions to consider than simply packing individual FPLDs as full as possible.

Then comes the ORCA Development System (ODS). An apparent philosophy of this tool was to “protect” the user from the details of the mapping process. One can always debate a chauvinistic attitude that the tool knows more about the design than the designer does. But our further concern was pedagogical: even if we had no intention of interfering with the mapping, yet we saw no reason why ODS should hide its magic from us. It would have been of vast educational value, especially to computer engineering students, to see intermediate results after prunings and optimizations, mappings of incoming components to LUTs and cells, or whatever else might be contained in the tool. And of course for a research environment, the difficulty to affect, steer, or override the mapping process seemed excessive. Even though the ORCA is a LUT-based FPLD, yet its library was specified as a clumsy collection of gates: we could

not find how to specify 4-input or 5-input arbitrary functions, much less incompletely-specified functions, thereby limiting the expressive power of our capture and synthesis tools. The other extreme, however, must also be avoided, namely being given an “editor” where *everything* is laid across the designer’s back. We simply feel that the “black-boxism” of ODS is too severe. Furthermore ODS only deals with one ORCA circuit at a time, which leads to different Viewlogic schematics for each ORCA circuit, and that only reinforces the hierarchy problem.

Whatever partitioning tools may become available in the future, we hope that the total design can be managed as a unit, so that placing the “cuts” can be timing driven, so that there is still opportunity for logic and timing optimizations and pad minimization across the “cuts,” and so that there is the opportunity to retime the entire design [LS91] even if it contains latches, exploiting cycle borrowing in that case [Szy92] [LE92].

## 5 Board-Level Routing

For each ORCA FPLD to be mapped, it was a simple task to generate a Viewlogic schematic symbol for it *automatically*. This meant that we avoided building the symbol by hand, which would have been extremely

tedious given the hundreds and hundreds of pins (each having its own name!) emerging from the routing process. Once again we used ORCAMASH to do this job: it accepted the original Viewlogic sheet which was the input into the ORCA synthesis (this gave us the Viewlogic signal names), the pinmap file emerging from the ODS mapping (this told us which pin to assign to each Viewlogic signal), and the ORCA package type (so that we could also append the utility pins to the symbol). The result was a netlist which could then be given to Viewlogic's *ViewGen* tool, creating the final graphical symbol for the routed ORCA. With this, we had the symbol needed to interconnect the mapped ORCA circuit to other ORCAs, or to other components, at the board level.

Then came the problem of connecting multiple ORCAs together. Since each FPLD had many hundreds of pins, the task of doing this by hand with the tools we were given (graphically, pin by pin or at best bus by bus in the face of many busses) would also have become a bottleneck. Even worse, so much manual effort for specifying board-level interconnect set up a resistance to making the kind of sweeping high-level architectural changes which a programmable interconnect ought to encourage; simply put we began to fear having to change the board-level layout. Our solution was to impose a convention: at the board level, pins of the same name would be connected together. This meant that a netlist, binding all these hundreds to thousands of board-level connections, could be generated automatically. Once again, ORCAMASH was adapted for this function as well, creating a netlist which could then be accepted by the other tools. Its input in this mode was simply the names of the separate Viewlogic graphical symbols which were to be interconnected. However, we still preferred to *see* what we had gotten, rather than having only a netlist which could not be visualized. So we passed this netlist through *ViewGen* again, so that we could see (graphically) how the components were bound together. Then, any remaining "dangling" nets could be connected in the graphical tool, or by explicit command to ORCAMASH, since they were few in number if any.

Our complaints at this level included, foremost, a final timing analysis telling us how fast the entire aggregation would run, including all pad delays, routing and fanout effects. As far as validating correctness and function of the aggregation, it was not difficult to modify ORCAMASH once again to build the "flat" network of the entire aggregation, given the Aptix-level netlist. However, ORCAMASH could only recon-

struct the design as it has been entered/synthesized in Viewlogic, and not as it had actually been mapped by (for example) the ORCA mapping software. This frustrated our ability to carry out an accurate final global timing analysis on the entire resultant design. "Timing back-annotation" information was *not* what we wanted to get from our tools. For example, suppose the ODS mapping tool performed both logic resynthesis and retiming, both of which could *completely* change the topology of the original circuit [MS<sup>+</sup>91]. What possible confidence could we then give to "back-annotation" data delivered in terms of our original topology, a topology completely different from the one which was actually implemented inside of the FPLD? Therefore we would prefer instead to get a "dump" of the entire network *as it is actually mapped and routed* for each ORCA chip from ODS, similarly for the Aptix board, so that we could work with this information instead.

The Aptix GP4 uses a "type C" pin array requiring expensive adapters with a big footprint, which we found clumsy to work with. That footprint forced us to place an ORCA FPLD between two FPICs. In our initial inexperience we compounded this by placing the clock generator next to a third FPIC. The consequences in terms of skew and race hazards of clock and data feedback paths being routed through two to three FPICs, and of busses being broken up with the various nets routed through different numbers of FPICs, started to worry us. We hope the next generation of FPCBs provides more opportunity for using land grid array packages. A final complaint with the GP4 Aptix board was the need to connect utility power and ground connections on the board *by hand*. This was a real frustration. This should have been made far easier to do, even at the cost of some area on the board.

## 6 Conclusion

In short, we experienced that just going out and buying tools does *not* mean that one is buying a design methodology at the same time! Neither does it mean that one has purchased a smooth design flow from capture to final analyses. Yet when teaching students to reason about the entire design process, it is exactly this that one wants to give to them.

Part of our task was to prepare a tutorial for the students. This document is over 80 pages and under constant revision. Readers interested in obtaining this

document are encouraged to contact the authors.<sup>3</sup>

## Acknowledgements

We are grateful to Dr. Lennart Andersson, department chair, who arranged the funding for the purchase of the prototyping equipment and software, and for being flexible in scheduling vacation times so that this work could be carried out during Summer 1994.

## References

- [B<sup>+</sup>86] R. Brayton et al. Multiple-Level Logic Optimization System. In *Proceedings, Int'l Conference on Computer-Aided Design (ICCAD '86)*, 1986.
- [BA92] B. Breidegard and P. Andersson. BBDS – A Design Tool for Architectural Evaluation and Rapid Prototyping of Performance Critical Digital Systems. In *Third Int'l Workshop on Rapid System Prototyping*, 1992.
- [Bra83] Daniel Brand. Redundancy and Don't Cares in Logic Synthesis. *IEEE Transactions on Computers*, C-32(10):947–952, October 1983.
- [Hay86] John P. Hayes. Digital Simulation with Multiple logic Values. *IEEE Trans. Computer Aided Design*, CAD-5(2):274–283, April 1986.
- [Jen91a] G. Jennings. A Case against Event-Driven Simulation for Digital System Design. In Alan H. Rutan, editor, *Proceedings, 24th Annual Simulation Symposium*, pages 170–176, April 1991.
- [Jen91b] G. Jennings. GRTL – a Graphical Platform for Pipelined System Design. In *Proceedings of the 1991 European Conference on Design Automation (EDAC 1991)*, pages 424–428, February 1991.
- [Jen91c] G. Jennings. Reversible Functional Simulation for Digital System Design. In *Proceedings of the IEEE 1991 Custom Integrated Circuits Conference (CICC '91)*, pages 8.2.1–8.2.4, May 1991.
- [Jen91d] G. Jennings. Using Constraint Logic in a Graphical Electronics Tool Interface. In *Proceedings, Fourth Intl. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 91)*, pages 557–561, June 1991.
- [Jen92a] G. Jennings. On Cycle Borrowing Analyses for Interconnected Chips Driven by Clocks Having Different but Commensurable Speeds. In J. Fortes et al., editors, *Proc., Int'l Conference on Application Specific Array Processors (ASAP '92)*, pages 81–88. IEEE Computer Society Press, August 4-7 1992. Berkeley, California.
- [Jen92b] G. Jennings. On the Detection and Elimination of Superfluous Level-Sensitive Latches. In *Proceedings, Second Great Lakes Symposium on VLSI (GLSV '92)*, pages 176–182, February 28-29 1992.
- [Jen92c] G. Jennings. The GRTL Reversible Compiled Register Transfer Simulator. In John Stephenson, editor, *Modelling and Simulation 1992, Proceedings of the 1992 European Simulation Multiconference*, pages 480–484. Society for Computer Simulation International (SCSI), June 1-3 1992. York, England.
- [Jen93] G. Jennings. On Short Paths, Hold Times, Clock Skew, and the Linearity of the Cycle Borrowing Problem. In *Tau 93: 1993 Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Sept. 1993.
- [Jen94] G. Jennings. Experience Using Ordered Ternary Decision Diagrams for Compiled Logic Simulation. In *Proceedings, SIMS '94, Stockholm*. Scandinavian Simulation Society, COMSOL, Björnåsvägen 21, 11347 Stockholm, Sweden, August 1994.
- [Jen95a] G. Jennings. Accurate Ternary-Valued Compiled Logic Simulation of Complex Logic Networks by OTDD Composition. In *Proceedings, 28th Annual Simulation Symposium*, April 1995.
- [Jen95b] G. Jennings. Symbolic Incompletely Specified Functions for Correct Evaluation in the Presence of Indeterminate Input Values. In *Twenty-Eighth Annual Hawaii Int'l Conference on System Sciences, HICSS-28*, January 1995.
- [JJ<sup>+</sup>93] G. Jennings, A. Joshi, et al. Practical Cycle Borrowing Analysis Using Interior Point Algorithms. In *Tau 93: 1993 Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Sept. 1993. Also presented by Prof. P. Vaidya, UIUC, at the TMS/ORSA 34th Joint National Meeting as invited paper, Nov. 1992.
- [JJ94] G. Jennings and E. Jennings. A Discrete Syntax for Level-Sensitive Latched Circuits having  $n$  Clocks and  $m$  Phases. Technical report, Div. of Computer Eng., Luleå University, Luleå, Sweden, January 1994, to appear, *IEEE Transactions on Computer-Aided Design*, January 1996.
- [LE92] Brian Lockyear and Carl Ebeling. Optimal Retiming of Multi-Phase, Level-Clocked Circuits. In Thomas Knight and John Savage, editors, *Advanced Research in VLSI and Parallel Systems*, pages 265–280, 1992.
- [LS91] Charles E. Leiserson and James B. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 6:5–35, 1991.
- [MS<sup>+</sup>91] Sharad Malik, Ellen M. Sentovich, et al. Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques. *IEEE Transactions on Computer-Aided Design*, 10(1):74–84, January 1991.
- [SS<sup>+</sup>92] E. M. Sentovich, K. J. Singh, et al. SIS: A System for Sequential Circuit Synthesis. Memo UCB/ERL M92/41, Dept. of EECS, U. C. Berkeley, May 4 1992.
- [Szy92] Thomas G. Szymanski. Computing Optimal Clock Schedules. In *Proceedings of the 29th Design Automation Conference*, pages 399–404, June 1992.

---

<sup>3</sup> glenn@sm.luth.se