MANIFESTATIONS OF HETEROGENEITY IN HARDWARE/SOFTWARE CODESIGN

Asawaree Kalavade and Edward A. Lee

Dept. of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720 USA
{kalavade,eal}@EECS.Berkeley.EDU

1. Types of Codesign

We identify three distinct types of hardware/software codesign:

1. Joint design of an instruction-set architecture and its program.
2. Synthesis of hardware and/or software from a common specification.
3. Specification, synthesis, and simulation of heterogeneous systems.

These can be viewed as manifestations of heterogeneity in the design methodology.

The first type concerns the joint synthesis of a data-path and its controller, two conceptually distinct parts of a processor [Pau93]. The heterogeneity in this approach is limited to distinguishing the datapath from its controller.

In type 2, a unified representation of an application is mapped into some mixture of implementation technologies. For example, a dataflow graph could be partitioned, and the separate partitions could be fed to hardware and software synthesis tools [Kal93]. The software components execute on commodity processors, or possibly on processors synthesized as in type 1. A hierarchical finite state machine could also be converted to hardware or software implementations [Har90]. If concurrent FSMs are used, then a mixture of hardware implementations might be attractive. In type 2, the heterogeneity is limited to distinguishing implementation technologies; the model of computation is unified.

In type 3, the design representation need not be unified. Distinct models of computation can be used for different parts of the design. The model of computation for each component would be chosen on the basis of the properties of that component. For instance, dataflow would be used for signal processing, and hierarchical FSMs for control. A typical application could combine both models of computation. Within each model of computation, types 1 or 2 might apply in synthesizing the implementations.

Type 3 addresses a large cross section of system-level design problems. In the signal processing domain, for example, most applications combine some hard-real-time numerical processing of signals with much more dynamic control. In applications such as modems, wireless communication systems, video phones, voice mail systems, voice recognition systems, disk-drive controllers, and data compression, one or more signal processing kernels are invoked in response to outside stimuli. For example, a modem needs to negotiate a common transmission format with another modem when a connection is established, or when a wireless device moves from one service area to another.

† This research was sponsored by the Semiconductor Research Corporation (92-DC-008) and the National Science Foundation (MIP-9201605).

2. System-Level Design

Type 1 is fundamentally still a chip-level design problem. It can be viewed as an approach to high-level synthesis. In this paper, we concentrate on system-level problems, and hence on types 2 and 3. In this context, four key problems emerge:

- partitioning
- synthesis
- cosimulation
- design methodology management

2.1. Partitioning

The partitioning problem has been broached, though somewhat less successfully so far, for hardware/software codesign [Gup92] [Hen93] [Bar92]. We have been investigating the partitioning problem for hardware/software codesign by concentrating on the synchronous dataflow (SDF) model of computation [Lee87]. Synthesis of both hardware and software from SDF graphs has been demonstrated [Pin94] [Rab91], and the model has proven useful for a reasonable set of signal processing applications. Thus, we focus on type 2 codesign, in which a unified representation is mapped onto some mixture of implementation technologies.

Even a unified model such as SDF does not preclude a bias towards one type of implementation or another. Certain properties of a graph lead to a preference:

- the mix of operations,
- the speed requirements,
- the regularity of the graph,
- the parallelism in the graph,
- the word length requirements, and
- the need for programmability.

In the last of these, where the end-product requires some degree of programmability, the bias is obviously towards software. In the others, the bias is less obvious.

If the mix of operations is dominated by bit manipulations, as in an error-correcting coder or decoder, the bias is towards hardware. If it is dominated by conditionals, the bias is towards software. If it is dominated by inner product computations, then the bias is towards software, since DSP architectures are highly tuned for this type of operation.

The regularity and parallelism of the graph couple with the other sources of bias [Rab93]. An irregular graph is biased towards software. A graph lacking parallelism is also biased towards software.

Widely disparate word length requirements for different parts of the graph imply a bias towards hardware, although this effect can get quite complicated. Applications requiring more precise numerical representations are biased towards software, if for no other reason than that the cost of the hardware is high, in both silicon area and design time. It had best be reused.

Systematic partitioning techniques require quantification of these factors that introduce a bias. A good start on some of them is described in [Rab93].

In order to make these bias values meaningful, it is important to partition at a task or a process level of granularity. This can be accomplished by clustering the original graph to identify natural clusters for mapping, taking into account communication overhead, control structures, and concurrency. Subsequently, the graph may be partitioned by an iterative assignment and scheduling process, taking these biases into consideration.

2.2. Synthesis of Hardware and Software

The hardware and software synthesis problem has received considerable attention (see [Kal93], [Rab91], and [Pin94], for example). We will not address it further in this paper.

2.3. Cosimulation

Cosimulation has two distinct manifestations. In a design that mixes models of computation (as in type 3), these models of computation must be made to operate together [Buc94]. Alternatively, in a design that mixes implementation technologies, as in types 2 and 3, or Figure 1), the models of designs using these technologies must interact [Kal93].

2.4. Design Management

Design management has been addressed at the chip level (see [All91] for example), but the extensions to the system level are not yet well understood (at least not by these authors). We are investigating a design methodology that treats the design flow as part of the design itself. Thus, for example, the choice of a partitioning tool is a part of a design that uses it. The choice of a synthesis tool for embedded software is similarly a part of the design.

The interconnection of all the tools used to construct a design, from the high-level algorithmic simulation tools, down to hardware simulation used for timing validation, is considered part of the design. Representation of this design flow, thus, becomes as important as representation of components in the design. Version management for tools gains equal priority with version management for library elements.

3. References

[All91] W. Allen., D. Rosenthal, K. Fiduk, “The MCC CAD Framework Methodology Management Systems”, *Proceedings of the 28th Design Automation Conference*, June 1991, pp 694-698.

[Bar92] E. Baros, W. Rosential, “A Method for hardware/software partitioning”, *Proceedings of COMPEURO’92, IEEE International Conference on Computer Systems and Software Engineering*, May 1992, The Hague, The Netherlands, pp 580-585.

[Buc94] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, “Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems”, *International Journal of Computer Simulation*, special issue on “Simulation Software Development,” January, 1994.

[Gup92] R. Gupta, Giovanni DeMicheli, “System-level synthesis using re-programmable components”, *Proceedings of the European Conference on Design Automation*, Brussels, Belgium, February 1992, pp 2-7.

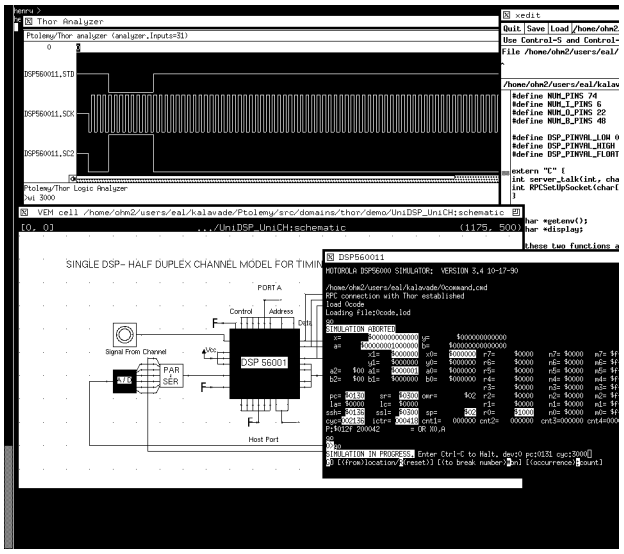

Figure 1 An illustration of a heterogeneous simulation environment. A hardware design (bottom left) containing a programmable DSP mixed with other hardware is developed using event-driven and SDF semantics. The DSP Model (corresponding C++ code in the upper right window) invokes the Motorola DSP56000 Simulator (bottom right) that executes software generated by software synthesis tools. Timing verification is possible by using a Logic Analyzer (top left).

[Har90] D. Harel, et. al., “STATEMATE: A Working Environment for the Development of Complex Reactive Systems”, *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, April 1990.

[Hen93] J. Henkel, T. Benner, R. Ernst, “Hardware Generation and Partitioning Effects in the COSYMA system”, Handouts from the *Second Intl. Workshop on Hardware/Software Codesign*, Boston, MA, Nov. 1993.

[Kal93] Asawaree Kalavade, Edward A. Lee, “A Hardware/Software Codesign Methodology for DSP Applications”, *IEEE Design and Test of Computers*, Sept. 1993.

[Lee87] E. A. Lee and D. G. Messerschmitt, “Synchronous Data Flow,” *IEEE Proceedings*, September, 1987.

[Pau94] Pierre G. Paulin, Cliff Liem, Trevor C. May, Shailesh Sutarwala, “DSP Design Tool Requirements for the Nineties: An Industrial Perspective”, Submitted to the *Journal of VLSI Signal Processing*, special issue on “Synthesis for DSP”, **to appear**, 1994.

[Pin94] J. Pino, S. Ha, E. Lee, J. Buck, “Software Synthesis for DSP Using Ptolemy”, invited paper in the *Journal on VLSI Signal Processing*, special issue on “Synthesis for DSP”, **to appear**, 1994.

[Rab91] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, “Fast Prototyping of Datapath-Intensive Architectures”, *IEEE Design and Test of Computers*, pp 40-51, June 1991.

[Rab93] Jan M. Rabaey and Lisa M. Guerra, “Exploring the Architecture and Algorithmic Space for Signal Processing Applications,” ICVC - 3rd Intl. Conference on VLSI and CAD 1993 (ICVC ‘93), Taejon, Korea, November 15-17, 1993.