HML: A language for high-level design of high-frequency circuits

Andrew K. Martin

11 Nov, 2003

Abstract

This talk will present the hardware description language HML. HML is a hardware description language based on the OCaml dialect of the ML programming language. We have used it successfully at IBM’s Austin Research Laboratory in the design of a recent test-chip. The chip, which was intended to test a novel digital MOS circuit family designed to run at very high frequencies, was fabricated in mid 2003. The design of the entire chip, with the exception of the PLL and clock control circuitry, was expressed in HML. It was mechanically translated to VHDL for functional verification and to a net-list for layout. In all, about 1500 lines of HML code were translated into 331,000 lines of VHDL, of which 133,000 were instantiated decoupling capacitor cells, and the remaining 198,000 lines were a structural VHDL description of functional circuit elements. The use of HML allowed a very small team to design and integrate the chip in a period of less than four months. Logic design and verification was performed by only three people in this time-frame.

HML is the first step in a three part research effort leading toward a high-level design environment for very high frequency designs. Ultimately, we envisage the following design process. First, a design is expressed at the microarchitecture level of abstraction. Then the design space within the given microarchitecture is explored iteratively through a series of mappings that transform the high-level design representation into a low-level implementation. The designer writes an explicit map from the high-level representation to an implementation. A compiler then applies the map to the high-level design to produce a low-level (gate or switch) representation. This result is then analyzed using standard tools for timing, area, power etc. This information is then used by the designer to adjust the mapping in an iterative process. The expectation is that making the map from high-level representation to implementation explicit will give the designer sufficient control of the synthesis process that even designers of high-frequency components will be able to achieve their performance targets without resorting to designing at the gate level.

There are three main components to this research effort. A suitable design language must be devised that allows for the rapid expression of designs at the micro-architecture level. A notation must be devised for expressing the mappings from high-level design to low-level implementation. Finally a suitable format for presenting the results of the mapped design must be devised, so that the results can be interpreted in a meaningful way by the designer, and hence guide the search for an optimal implementation of the original design. HML addresses the first of these components.

HML is based on the Ocaml dialect of the programming language ML. The definition of HML is still somewhat fluid. Since, at present it is used only in a small research group, features can be added or removed from the language as dictated by our experience, both in using the language, and implementing tools that support it. This paper presents one snapshot of the language as used to design a recent test-chip. The language presentation will be followed by a discussion of our experience using the language, both positive and negative, and the implications of this experience on our future work.

HML was designed by the author at IBM’s Austin Research Laboratory. HML augments ML with a polymorphic primitive stream type, to represent the sequence of values that occur over time on a signal in a synchronous design. Hardware components are represented as functions from input streams to output streams. Component instantiation is represented by function application. As in ML, HML treats functions as first class values: They can be passed as parameters, to other functions, returned as results, and constructed anonymously. Since hardware components are represented as functions, they can be passed as arguments to functions, or to other hardware components, returned as results etc. To describe a piece of hardware in HML one simply writes a function.

The compiler performs HAWK-style lifting automatically. Functions that were not initially designed to work with streams are automatically promoted by the compiler to functions over streams as required. For example, the function fun x -> not x can be applied to a boolean argument, which it will complement, or it can be applied to a bool stream in which case it will return a stream of booleans – the point-wise complementation of its. State is represented in HML by the syntax delay(v1,v2) where v1 is a value of type a, and v2 is an a-stream. The result is a stream whose first value is v1, and whose subsequent values are those of v2 (delayed one unit of time).

For the test-chip an HML to net-list compiler was built that implemented a highly restricted subset of the language. No “synthesis” was supported in the usual sense. Instead, a library of primitive functions with hardware implementations was provided to the compiler. The resulting net-list consisted entirely of instantiations of these primitive circuit elements. Streams were restricted types whose atomic elements were booleans – thus integer valued signals were disallowed. In spite of these restrictions, the initial implementation turned out to have considerable value.

HML as described here represents a first step in an efficient system for logic design for high frequency circuits. The compiler used in the test site was essentially a net-lister. That is, the designer writes an ML program that is interpreted as a set of cell instantiations. Work is currently under way to build a second version of the compiler, which will include more comprehensive simulation and synthesis capabilities. In particular the second two research problems: a notation to express mappings from the high-level design to implementation, and a format for presenting the mapped design have yet to be addressed.