Efficient Circuit Design with uDL

STEVE HOOVER
Massachusetts Microprocessor Design Center, Intel
steve.hoover@intel.com

uDL (Microarchitecture Design Language) is an RTL-replacement concept language that promises many benefits through a long-list of synergistic features. It draws from decades of high-performance design experience in various disciplines. At its heart, is its ability to raise the level of abstraction without sacrificing control over physical implementation details. Languages like Esterel [1] and BlueSpec SV [2] introduce abstraction, but relinquish control of physical details to compilation technology, which yields unsatisfactory results for highly constrained designs. Lava [3] and Wired [4] introduce abstraction coupled with physical control, though for structured data-path logic only.

A uDL model encapsulates the same information as RTL, and, based on manual translation experiments, it does so with half as much code. It could (in theory) be compiled to RTL to feed downstream tools. A uDL model is comprised of two distinct (though interwoven) components: a cycle-accurate "behavioral model", and mappings from that model to physical details. Figure 1 illustrates the three main forms of mappings. 1) Behavioral logic is bucketed into **physical partitions**. Bucketing replaces explicit interface maintenance required by today's RTL. 2) Pipelined logic is bucketed into **pipe stages**. From this, sequential elements like flip-flops are inferred. 3) Values in the behavioral model are mapped to physical **encodings**. Encoders and decoders, for example, are absent from the behavioral model. Beyond mappings, a "refinement" capability allows portions of behavioral logic to be expressed in physically-oriented refinements and verified through formal equivalence verification or shadow simulation.

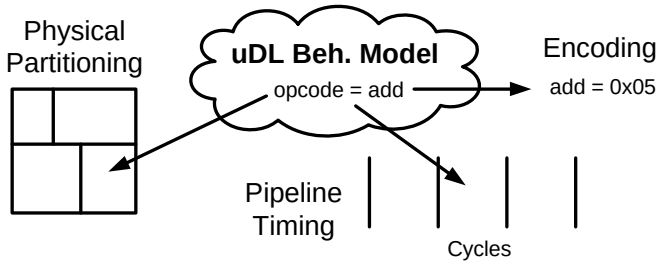

Figure 1: Forms of uDL Mappings

uDL methodology (Figure 2, right) represents substantial design efficiencies. The behavioral model is easily a quarter the size of RTL, which means models come alive faster, and there is less to verify. Performance modeling is more feasible with uDL models than with RTL, which can eliminate modeling effort. Later in the process, physically-oriented changes that plague designs today, such as repartitioning, retiming, and re-encoding, are absorbed in the uDL mappings and require neither regression testing nor involvement from the architecture team. The myriad verification models (transactors, checkers, reference models, protocol models, instruction set models, coverage monitors, visualization tools, and more) bind to a simpler and more stable model. Other methodology improvements enabled by experimental aspects of uDL represent exciting opportunities for discussion.

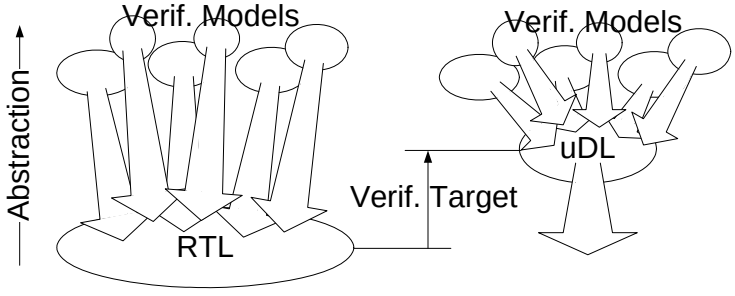

Figure 2: uDL Methodology

References

[1] http://www.esterel-technologies.com

[2] http://www.bluespec.com

[3] Per Bjesse, Koen Claessen, Mary Sheeran, Satnam Singh, “Lava: Hardware Design in Haskell,” International Conference on Functional Programming, September 1998.

[4] E. Axelsson, K. Claessen, and M. Sheeran, “Wired: Wire-aware circuit design,” Conference on Correct Hardware Design and Verification Methods, 2005.