

Exploring High-Level Languages for Accelerator Design

Minsik Cho, Geert Janssen, Indira Nair and Jeonghee Shin
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
{minsikcho, geert, indira, jeonshin}@us.ibm.com

1. Introduction

Accelerators provide additional performance for specialized tasks such as encryption, compression and search [1][2]. However, developing an accelerator optimized for specific functionalities in a conventional manner can be expensive because **(a)** it has low reusability compared with a general-purpose processor, and **(b)** it still has to go through rigorous design steps such as modeling, synthesis and verification. Hence, it is more important than ever to have a fast and affordable design flow from modeling to silicon. Also, it should be easy to maintain and upgrade the already-developed accelerators with new functionalities. Considering the limited automation from architectural modeling to silicon in the current design flows and the advantages of high-level synthesis (e.g., powerful abstraction and high design productivity), a new design paradigm based on high-level synthesis can offer a promising solution for efficient accelerator development in the future. This paper describes an experiment underway at IBM Research to explore such a possibility.

The contributions of this paper consist of a study and discussions on the applicability of a high-level language, Bluespec SystemVerilog (BSV) [3] in our case, to the design of a high-performance processor that is designed to function as an accelerator in a larger system. In this study, we set our goal to be the implementation in BSV of a substantial unit of an IBM PowerPC processor with performance requirements equal to its counterpart that was previously designed.

2. Electronic System Level (ESL)

Among the many languages proposed for high-level hardware design such as SystemC, SystemVerilog and HardwareC, we adopted Bluespec SystemVerilog (BSV) for our project. BSV is a proprietary language with a simulator and RTL synthesis tool. BSV, in its current form, is the second generation of a language that has its roots in functional programming and term rewriting. Particularly, it borrows many ideas and constructs from Haskell for its combinational

expressiveness. To specify the dynamic aspects of the design such as the ordering of events and handling of state, BSV offers the concept of atomic actions inside rules, which is supported by an automatically generated scheduler to resolve conflicts. This is unusual for an HDL: most of them are imperative languages based on the C/C++ family of programming languages. On the other hand, BSV does hide most of its functional heritage in a more accessible SystemVerilog oriented syntax. BSV aims at productivity improvement especially in the area of ASIC and FPGA designs. Although many competitors also address the concept of generate statements (i.e., ways to express composition and repetition in a programmatic fashion), BSV's capabilities are very extensive, thanks to its functional background with polymorphic types and the fact that rules and interfaces can be manipulated as lists and vectors. It is good to note that BSV does not offer high-level synthesis in the style of silicon compilation. It requires all state elements to be explicitly instantiated by the designer and merely helps in managing access to these elements as dictated by their rule semantics.

It is our belief that high-level languages (like BSV) are crucial in enabling rapid prototype development. BSV claims that anything written in it is synthesizable; there are no awkward exceptions or subsets. The BSV compiler produces either a compiled C simulation model or a Verilog model. The C model is useful for quick verification through simulation; the Verilog model can be taken as input to a more conventional back-end flow of logic synthesis, placement and routing for an FPGA or custom chip implementation.

3. An experiment in using BSV

Although BSV has been successfully used in FPGA-based performance modeling of high-performance processors, its applicability to the design and implementation of high-performance microprocessors is largely unknown. It is unclear if the quality of hardware in terms of area and cycle time would be comparable to carefully hand-coded VHDL when the BSV generated Verilog is taken through a

synthesis flow. To address these questions, we decided to implement an entire unit of an existing IBM microprocessor in BSV and evaluate the BSV specification against the VHDL one in terms of designer productivity, hardware quality and the ability to easily make design modifications. As a test vehicle, we selected a somewhat complex instruction unit of an IBM PowerPC processor, which performs instruction cache access, branch prediction, instruction decode, dependency analysis and instruction issue to one or more execution units. For this experiment, we chose to preserve the existing VHDL logic boundaries and processor pipeline in our BSV design. This tight correspondence between the BSV modules and the existing VHDL macros allowed for easier comparisons and also provided the BSV designers with pre-defined interface definitions. In implementing the instruction unit, we exploited many of the abstractions and advanced types such as vectors and structs provided by BSV wherever possible. We also created a parameterized model and used BSV’s ability to specify polymorphism in order to allow easier design modifications.

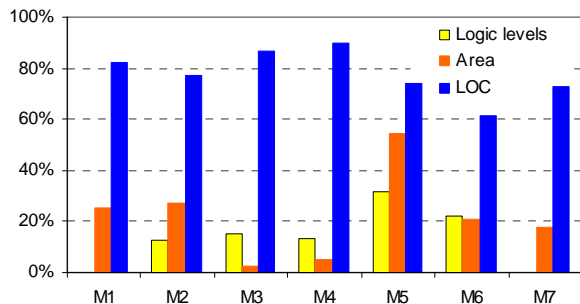


Figure 1 BSV area and logic levels in excess of VHDL; BSV LOC (lines of code) improvement over VHDL

Figure 1 shows the comparative results that we obtained by running both the VHDL and the BSV specifications for seven modules through a preliminary logic synthesis flow that did not include physical synthesis. The area results show the combinational area only – we did not consider arrays or latches in this comparison. Without applying timing assertions, our timing comparison is based on comparing the maximum number of logic levels in the post-synthesis networks. Designer productivity was measured loosely by comparing the number of lines of code in the two designs. It should be noted that the BSV specification did not include the pervasive logic (scan or performance monitoring) that was present in the original VHDL. As shown in Figure 1, the BSV specification is much more compact than the VHDL. The area and logic levels of the BSV design are

somewhat worse than the VHDL, although more intensive physical synthesis may improve these results.

Like other high-level description languages, BSV provides greater modularity and reusability than RTL design due to better support of parameterization, structures and polymorphism. BSV design is also self-documenting since the functionality of hardware can be described in an algorithmic way. However, the currently available version of BSV often reveals its limitations for implementing accelerators with more complex and flat structures. This is because BSV modules cannot be cross-instantiated to interface sibling macros talking to each another. The concept of transactions called rules and rule scheduling may be confusing initially. The automation of scheduling can be cumbersome, especially in designing fairly complex high-performance accelerators, due to implicit conditions and false cycles created by rule dependencies. In addition, mapping the generated Verilog code back to BSV is not straightforward if lower-level simulation and verification are required. A fine-tuning of area and timing is difficult, if at all possible, since the designers do not have control over RTL combinational logic and latch types generated by the BSV compiler.

4. Conclusion

From our BSV design experiment, we can conclude that languages like BSV have the potential to improve designer productivity, while delivering results that are comparable to hand-crafted VHDL specifications. In the accelerator design space, with a potentially large number of designs being created, we believe that improved designer productivity can be achieved by creating flexible designs with reusable components in a high-level language like BSV with continued development of the synthesis and simulation tools.

Acknowledgments

The authors would like to thank their colleague Haoxing Ren for helping out establishing the synthesis environment and gathering the experimental data, and John A. Darringer and their management for useful feedback on the paper.

References

- [1] J. Landman, “The Need for Acceleration Technologies to Achieve Cost-effective Supercomputing Performance for Advanced Applications”, AMD White Papers, 2007.
- [2] P. Wang et al., “Accelerator Exoskeleton”, Intel Technical Journal, 2007.
- [3] Bluespec, <http://bluespec.com>