The Open-Source Bluespec *bsc* Compiler and Reusable Example Designs

Julie Schwartz, Niraj N. Sharma, Darius Rad, Ken Takusagawa, Joe Stoy, Rishiyur S. Nikhil
Bluespec, Inc., Framingham MA, USA
{quark,niraj.sharma,darius,ken,stoy,nikhil}@bluespec.com

***Abstract*—The *bsc* compiler, which has been in commercial use for two decades, was released as a free and open-source artefact in January 2020. Here we describe briefly *bsc* and its flows, available tutorial materials, and several reusable open-source designs using *bsc*, many of them centered on RISC-V (from embedded to Linux-capable CPUs and systems), all FPGA-ready.**

***Index Terms*—EDA, BSV, BH, HDL, HLHDL, RISC-V**

I. Introduction

The *bsc* compiler and its libraries have been developed for commercial use since 2000. It was a licensed product from Bluespec, Inc. for two decades, although academic and research licenses have always been available at no cost. A couple of top-ten semiconductor vendors and one major search company have used *bsc* to design complex IPs in state-of-the-art ASIC SoCs. It has been used by many companies for FPGA-based prototyping, and in many universities for research on architecture and complex IPs.

In January 2020, Bluespec, Inc. released the compiler, libraries, and a GUI as free and open-source artefacts under the BSD 3 Clause license. These are available at:

https://github.com/B-Lang-org/bsc
https://github.com/B-Lang-org/bsc-contrib
https://github.com/B-Lang-org/bdw

The *bsc* compiler takes as input designs written in the HLHDLs (High Level Hardware Description Languages) BSV and BH (described below), and generates vanilla, synthesizable Verilog, which is then processed by standard RTL tools (simulation, synthesis, formal analysis, etc.).

In this paper, we describe briefly BSV and BH, *bsc* and its flows, and several free and open-source designs written in BSV/BH. Many of the designs are RISC-V artefacts (CPUs, IPs and systems). All are highy reusable and run on FPGAs.

II. Background on BSV and BH, High Level Hardware Design Languages

BSV and BH originated in research at MIT in the 1990s [10], when James Hoe and Arvind established the feasibility of compiling high quality Verilog from behavior expressed as *concurrent atomic transactions*. This is attractive because it is the behavioral model of choice for formal specification and analysis of complex concurrent systems (e.g., TLA+ [11], Event-B [12], UNITY [7]). Being able to compile such specs to quality hardware therefore enables an automation bridge between formal specifications and actual hardware.

Enabling compositional formal verification has always been a central motivation and feature of BSV/BH (e.g., see MIT's Kami project at https://github.com/mit-plv/kami). Atomic-transactional semantics also enhances everyday informal reasoning about correctness by the practicing engineer.

In 2000, Lennart Augustsson implemented these ideas in a new language, BH (for Bluespec Haskell), which uses Haskell's syntax and semantics, including its type system with polymorphism and type classes, monads and higher-order functions, resulting in powerful static elaboration and very strong type abstraction and type-checking [2], [14].

In 2004-2005 we created BSV, which was just a new, SystemVerilog-ish front end syntax [6], but which otherwise retained all the Haskell-like expressive power of BH.

In 2005 we introduced support for multiple clock and reset domains. Strong type-checking ensures the impossibility of mixing clocks and reset with ordinary signals. Static checking by *bsc* enforces proper clock domain discipline.

A. Comparison with RTL, Chisel and other HDLs, and HLS

BSV/BH, VHDL, Verilog, SystemVerilog and Chisel [3] are all HDLs: designers explicitly describe architecture and microarchitecture. In this, they are all fundamentally different from so-called "HLS" (High Level Synthesis) [9], where tools synthesize architectures and microarchitectures, possibly with high-level steering by the designer. This doesn't mean HDLs must be low-level: powerful abstraction and composition mechanisms allow describing high-level, parameterized architectures just as fluently as low-level microarchitectures [14]. BSV/BH gets this by mimicking Haskell; Chisel gets this by embedding in Scala.

BSV and BH also differ from languages like Lava [5] which focus on powerful, correct composition mechanisms for circuit structure and not on behavior.

As with other HDLs that directly express architecture, BSV/BH is general-purpose, and not targeted to any particular application domain.

Perhaps the most fundamental, broad-reaching and significant difference between BSV/BH and all other HDLs is its choice of concurrent atomic transactions as its central (and only) way to express behavior.

III. BSV/BH Designs, the *bsc* Compiler and Flows

As in other HDLs, a BSV/BH design is a hierarchy of module instantiations, albeit with much more powerful static elaboration compared to other HDLs because of the power of language semantics inspired by Haskell.

Unlike other HDLs, where behaviour is expressed as synchronous clocked processes, in BSV/BH it is expressed as *rules* which are globally atomic transactions.

Unlike other HDLs, where inter-module communication is based on input and output signal buses, in BSV/BH it is expressed using *methods* which are invoked from rules (or transitively from other methods). Methods extend atomic-transaction semantics of rules across module boundaries, i.e., atomicity *composes* as designs scale. A module's interface methods constitute a first-class *interface type*,

As in SystemVerilog, modules, interfaces and types can be partitioned into source files called *packages*. *bsc* takes BSV or BH source files (packages) and produces synthesizable Verilog, i.e., immediately acceptable to existing FPGA and ASIC synthesis tools. Separate package compilation by *bsc* enables fast incremental rebuilds in large systems.

bsc-generated Verilog runs on most well-known simulators, both open-source (Icarus, Verilator, CVC etc.) and commercial (Synopsys, Cadence, Mentor, Xilinx). It can be synthesized by most well-known synthesis tools (Design Compiler, Vivado, Quartus and those of other FPGA vendors).

A. *Interoperability with existing RTL and with C*

Verilog generated by *bsc* from BSV/BH can of course be instantiated in existing VHDL/Verilog/SystemVerilog modules. In the opposite direction, BSV/BH has import mechanisms to instantiate existing Verilog modules inside BSV/BH designs. Thus, one can freely mix and match BSV/BH with existing RTL in existing flows. For simulation, BSV/BH also has DPI mechanisms to import arbitrary C code.

B. bsc *internals and optional GUI*

Although BSV/BH borrows semantic and type ideas heavily from Haskell, it is not a DSL inside an existing Haskell compiler or system. *bsc* is a completely standalone, purpose-built compiler (that just happens to be written in Haskell). A central difference between *bsc* and other HDL compilers (such as for Verilator and Chisel) is *rule scheduling*, i.e., production of synchronous, clocked Verilog with control logic that preserves the atomic-transaction behavioral semantics. *bsc* uses open-source SAT solvers to analyze relationships between rule conditions, which results in optimized control logic.

In addition to Verilog, *bsc* can also generate C code and compile that into a standalone, executable simulator (*Bluesim*) that is perfectly cycle-accurate with RTL simulation and can generate VCD files.

Compiler object files can be queried and controlled from an API. This API has a Tcl binding (*bluetcl*); in fact Bluesim is just a Tcl wrapper exploiting this.

Although all BSV/BH development can be done from the command-line, *Bluespec Development Workstation* (BDW) (https://github.com/B-Lang-org/bdw) provides a GUI from which one can explore source hierarchy, compile, build and run simulations, and observe waveforms on an attached VCD viewer such as GtkWave. BDW can configure GtkWave to show waveforms at BSV source-level types (enums, structs, unions, vectors) instead of the flat signal buses of Verilog. BDW presents graphical displays of rule schedules to help understand how atomic-transaction rules have been mapped into clocked logic by *bsc*.

IV. Tutorials and Books

A variety of free and open-source tutorial resources are available for learning BSV/BH and how to use the tools. https://github.com/BSVLang/Main contains Bluespec, Inc.'s free and open training course for BSV/BH, as well as a free PDF copy of the *BSV by Example* book [15].

Links to tutorial materials and a video recording for the tutorial "Designing Hardware Systems and Accelerators with Open-Source BH" at the Intl. Conf. on Functional Programming 2020 are given in reference [16].

MIT (USA), Carnegie Mellon U. (USA), U. Cambridge (UK), Seoul National U. (S. Korea), Indian Institute of Technology Madras (India) and T. U. Darmstadt (Germany) have run courses on BSV for their students for many years, at both the undergraduate and graduate level; some of their materials are available on their respective public web sites.

The open-source designs listed in the next section are also a rich source of examples for people learning BSV/BH.

V. Example designs in BSV/BH

Over the years, BSV and BH have been used by a number of companies for complex IP subsystems in product ASIC SoCs. These include a many-to-many high-speed video data mover for set-top boxes, a display controller for handhelds, and an AI chip accelerator. Many companies exploited BSV's powerful abstraction mechanisms and type systems for rapid prototyping and modeling, on FPGAs, of future ASIC designs. While these examples demonstrate the power, scalability and maturity of BSV/BH and *bsc*, they are not available in open-source form.

However, many designs from Bluespec, Inc. and from leading universities are available under free and open-source licenses; a sampling of these are described next. Most of them are fairly large and may be of interest as tools in their own right. Although written in BSV, *bsc* compiles them to Verilog and they can be used as IPs in designs that are otherwise designed with RTL (see "interoperability" above).

A. *Open-source RISC-V CPUs (Bluespec, Inc.)*

Bluespec, Inc. has open-sourced a range of RISC-V processor designs, ranging from very small (for embedded and microcontroller applications) to very large and complex. They are all available at https://github.com/bluespec, under the Apache License, Version 2.0.

- Piccolo: 3-stage in-order pipeline, separate I- and D-channels.
- Flute: 5-stage in-order pipeline with branch prediction, separate I- and D- channels.

- Toooba: a packaging of MIT's RISCY-OOO design [20]: out-of-order, superscalar, deeply pipelined, branch prediction, separate cache-coherent I- and D- channels, multicore, cache-coherence.

Piccolo and Flute are parameterized to build any combination of the standard RISC-V unprivileged ISA choices: RV32 or RV64, I (integer), M (integer multiply/divide), A (atomics), C (16-bit compressed instructions) and FD (single- and double-precision IEEE floating point). They can optionally be configured for Privileged ISA levels M (machine), S (Supervisor) and U (User). For S, they support standard Sv32 and Sv39 Virtual Memory schemes. Thus, they are both Linux-capable. In addition, the CPUs are parameterized for a simple-to-complex range of memory systems: TCM (tightly-coupled memory), L1-only (writeback or writethrough), and cache coherent I-L1 + D-L1 + shared L2.

MIT's RISCY-OOO CPU inside Toooba is parameterized for number of cores, superscalarity, degree of speculation, size and organization of reorder buffers and branch prediction, size and organization of MMUs and caches, and more.

B. *Open-source IPs for RISC-V systems (Bluespec, Inc.)*

The following are available at https://github.com/bluespec, under the Apache License, Version 2.0:

- Debug Module: RISC-V spec'd hardware module adjacent to a RISC-V CPU enabling remote GDB control.
- PLIC (Platform Level Interrupt Controller): RISC-V spec'd interrupt controller arbitrating interrupts from external devices to one or more RISC-V cores.
- AXI4 and AXI4 Lite interfaces and interconnects.
- The memory systems described in the previous section (Tightly Coupled Memory, L1 and L2 caches with or without cache coherence) can also be reused as IPs in other RISC-V designs.

C. *AWSteria_Infra and Connectal for host+FPGA Systems*

AWSteria_Infra is a system written in BSV to simplify design of applications comprising software on a host computer communicating with hardware on an FPGA. Fig. 1 shows the structure of AWSteria_Infra and an application. C code on the host, and BSV code on the FPGA provide simple interfaces to the application (AXI4 and AXI4 Lite). BSV code on the FPGA also provides simple interfaces (AXI4) to DDR memory on the FPGA board. These interfaces are similar to the so-called "shell" provided in Amazon's aws-fpga development kits [1], but are available across a wider set of platforms.

AWSteria_Infra implementations exist for RTL simulation (using TCP/IP for communication), and for Xilinx VCU118 and AWS F1 FPGA systems (both use PCIe for communication). More platforms will be supported in the future. It is available at https://github.com/bluespec/AWSteria_Infra under the Apache License, Version 2.0.

Connectal, also written in BSV, has the same overall purpose, but provides a "remote method" model for host-FPGA communication, in both directions. It supports a wider set of platforms and is available at https://github.com/cambridgehackers/connectal, under the MIT License.

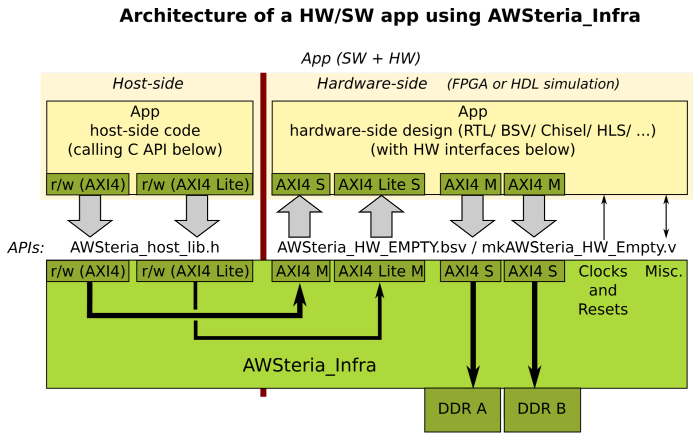
Architecture of a HW/SW app using AWSteria_Infra
App (SW + HW)
Host-side
Hardware-side (FPGA or HDL simulation)
App host-side code (calling C API below)
App hardware-side design (RTL/ BSV/ Chisel/ HLS/ ...) (with HW interfaces below)
r/w (AXI4)
r/w (AXI4 Lite)
AXI4 S
AXI4 Lite S
AXI4 M
AXI4 M
APIs:
AWSteria_host_lib.h
AWSteria_HW_EMPTY.bsv / mkAWSteria_HW_Empty.v
AXI4 M
AXI4 Lite M
AXI4 S
AXI4 S
Clocks and Resets
Misc.
AWSteria_Infra
DDR A
DDR B

Fig. 1. AWSteria_Infra for SW+FPGA systems.

D. *AWSteria-RISCV-Virtio (Bluespec, Inc.)*

AWSteria-RISCV-Virtio is a RISC-V system running on FPGAs that boots multi-user FreeBSD (it is Linux-capable as well, but this has not been tested yet). It has access to networks and block-storage devices, even on FPGA boards in the cloud which do not have (accessible) on-board network or storage devices. It accomplishes this using "Virtio" [18], an open standard originally developed for virtualization to provide portable device services for guest OSes across multiple host hypervisors.

Fig. 2 shows components in AWSteria-RISCV-Virtio. The FPGA-side is a RISC-V system with a BSV CPU (Flute, Toooba), Debug Module, PLIC (see above), DDR memory, a UART, and MMIO-to-host access for Virtio. Host-side code provides console TTY I/O for the RISC-V CPU, GDB control of the RISC-V CPU, and Virtio device services (networking, block storage, etc.) to the OS on the RISC-V CPU. Host-side Virtio devices are provided by TinyEMU, Fabrice Bellard's open-source system emulator for RISC-V [4] (a smaller and simpler version of QEMU). The host has cache-coherent access to FPGA DDRs, required for Virtio.

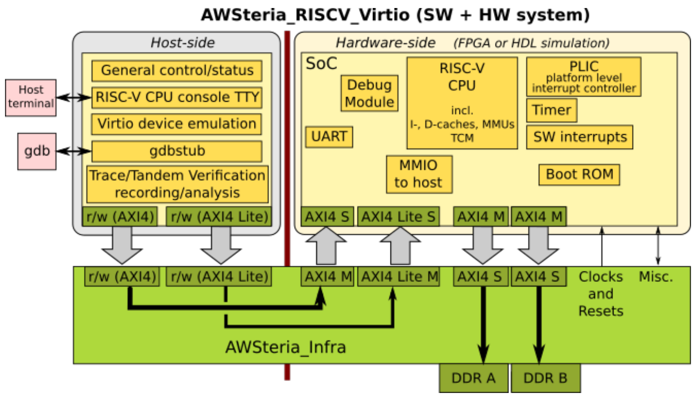

Fig. 2. AWSteria_RISCV_Virtio.

The system is built atop AWSteria_Infra (Sec. V-C) and hence is immediately portable to any supported platform (currently RTL simulation, Amazon AWS F1, Xilinx VCU118 boards). AWSteria-RISCV-Virtio is available at https://github.com/GaloisInc/BESSPIN-CloudGFE/tree/rsn3/AWSteria_RISCV_Virtio (will move soon to https://github.com/GaloisInc/BESSPIN-CloudGFE).

E. Secure RISC-V (U. Cambridge)

For a number of years Cambridge has been researching CHERI: Capability Hardware Enhanced RISC Instructions. Instructions and memory systems are enhanced with "capabilities" that enable secure computing, i.e., eliminate security vulnerabilities in traditional designs [19]. The designs are written in BSV and are available at https://github.com/CTSRD-CHERI under a license based on Apache License 2.0.

F. Shakti RISC-V Processors (IIT Madras)

The Shakti initiative at Indian Institute of Technology Madras is building a family of production-grade processors, SoCs, development boards and software platforms around RISC-V. The processors and SoCs are written in BSV, and are available at https://gitlab.com/shaktiproject under a BSD 3-Clause License.

G. Network-on-a-Chip Generator (Carnegie Mellon U.)

Papamichael and Hoe at CMU developed CONNECT, a NoC generator for FPGA-tuned multi-node NoCs of arbitrary topology [17]. This was the basis of their award-winning entry in the MEMOCODE 2011 Hardware/Software Codesign contest. This has recently been open-sourced at https://github.com/crossroadsfpga/connect with a BSD 3 Clause license.

H. BlueCheck Generic Hardware Testbench (U. Cambridge)

BlueCheck is an implementation of Haskell's QuickCheck [8] in BSV, exploiting BSV's inclusion of the same Haskell features used by QuickCheck: polymorphic types and type classes, monads and higher-order functions. Like QuickCheck, it has:

- automatic test-sequence generation from interface types, with support to override defaults;
- iterative deepening: generating longer and longer test sequences;
- shrinking: automatically shortening test sequences when a failure is found;
- full synthesizability (being written in BSV): the testbench can run at speed on FPGA with a DUT.

It is available at https://github.com/CTSRD-CHERI/bluecheck under the BERI Hardware-Software License, Version 1.0.

Acknowledgment

Thanks to Bluespec, Inc. for open-sourcing *bsc* and its libraries, and for providing ongoing resources to maintain and improve them. From 2000 to 2020 many people at Sandburst Corp. and Bluespec, Inc. contributed to BSV/BH language design, *bsc* implementation, BSV/BH documentation and usage. From a long list of universities with loyal, enthusiastic and sophisticated users, we mention especially MIT, Carnegie Mellon, U. Texas at Austin, U. Cambridge, Indian Institute of Technology Madras, Indian Institute of Technology Bombay, Seoul National U., Georgia Institute of Technology, T. U. Darmstadt and U. Toronto. The most recent feature added to BSV/BH and *bsc* ("Generics") was designed and implemented by Lucas Kramer at U. Minnesota and Ravi Nanavati at Google. At various times, BSV has been used by R&D and production teams in many companies, including Intel, IBM, ST Micro, Texas Instruments, Qualcomm and Google. Some of the open-source artefacts from Bluespec, Inc., described here were created in DARPA-funded projects.

References

[1] Amazon, Inc., "AWS EC2 FPGA Development Kit" https://github.com/aws/aws-fpga, 2021

[2] L. Augustsson, J. Schwartz, and R. S. Nikhil, "Bluespec Language Definition," Technical Report, Sandburst Corp., 95 pp., 2001

[3] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek and K. Asanovic, "Chisel: Constructing Hardware in a Scala Embedded Language," Proc. Design Automation Conferene (DAC), 2012

[4] F. Bellard, "TinyEmu," https://bellard.org/tinyemu/, 2019

[5] P. Bjesse, K. Claessen, M. Sheeran, and S. Singh, "Lava: Hardware Design in Haskell," Proc. ACM Intl. Conf. on Functional Programming (ICFP), 1998

[6] "Bluespec SystemVerilog Reference Guide," https://github.com/B-Lang-org/bsc Originally published by Bluespec, Inc. in 2005, with many subsequent updates. 462 pp. 2021

[7] K. Mani Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison Wesley, 516 pp. 1988

[8] K. Claessen and J. Hughes, "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," Proc. Intl. Conf. on Functional Programming (ICFP), pp. 268-279, 2000

[9] P. Coussy and A. Morawiec (eds.), *High Level Synthesis*, Springer, 297 pp., June 2008

[10] J. Hoe and Arvind, "Synthesis of operation-centric hardware descriptions," Proc. IEEE/ACM Intl. Conf. on Computer Aided Design (ICCAD), pp. 511-518, 2000

[11] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley Professional (Pearson Education), 2002

[12] C. Metayer, J.-R. Abrial and L. Voisin, "RODIN Deliverable 3.2: The Event-B Language," 147 pp., Tech.l Report Project IST-511599, School of Computing Science, U. Newcastle, Newcastle, May 31, 2005

[13] M. Naylor and S. Moore, "A Generic Synthesisable Test Bench," Proc. ACM/IEEE Intl. Conf. on Formal Methods and Models for Codesign (MEMOCODE), 2015

[14] R. S. Nikhil, "Abstraction in Hardware System Design," Communications of the ACM, *54*:10, pp. 36-44, October 2011.

[15] R. S. Nikhil and K. R. Czeck, *BSV by Example*, CreateSpace, 302 pp., December 2010 (avail. on Amazon, or free PDF from B-Lang-org repo)

[16] R. S. Nikhil, "Designing Hardware Systems and Accelerators with Open-Source BH (Bluespec Haskell)," Tutorial at Intl. Conf. on Functional Programming (ICFP), August 2020. Tutorial materials: https://github.com/rsnikhil/ICFP2020_Bluespec_Tutorial Video recording: https://www.youtube.com/watch?v=JCxE3JQAXY0

[17] M. K. Papamichael and J. C. Hoe, "The CONNECT Network-on-Chip Generator," IEEE Computer, December, 2015

[18] "Virtual I/O Device (VIRTIO) Version 1.1, Committee Specification 01," https://docs.oasis-open.org/virtio/virtio/v1.1/virtio-v1.1.html, April 2019

[19] J. Woodruff, R. N. M. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton and M. Roe, "The CHERI capability model: Revisiting RISC in and age of risk," Proc. Intl. Symp. on Computer Architecture (ISCA), 2014

[20] S. Zhang, "Constructing and Evaluating Weak Memory Models," Ph.D. Thesis, Massachusetts Institute of Technology, June 2019