Panel: "System-Level Design and High-Level Synthesis"

ESWeek, Montreal, September 30, 2013

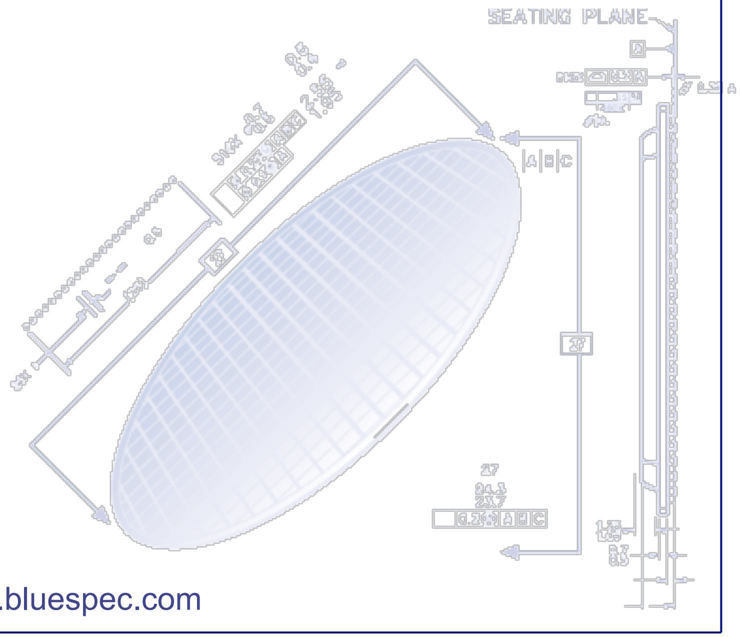
Rishiyur S. Nikhil, Ph.D.
CTO and co-founder, Bluespec, Inc.

Contents:

0. Brief background on Bluespec approach to HLS, for context and contrast

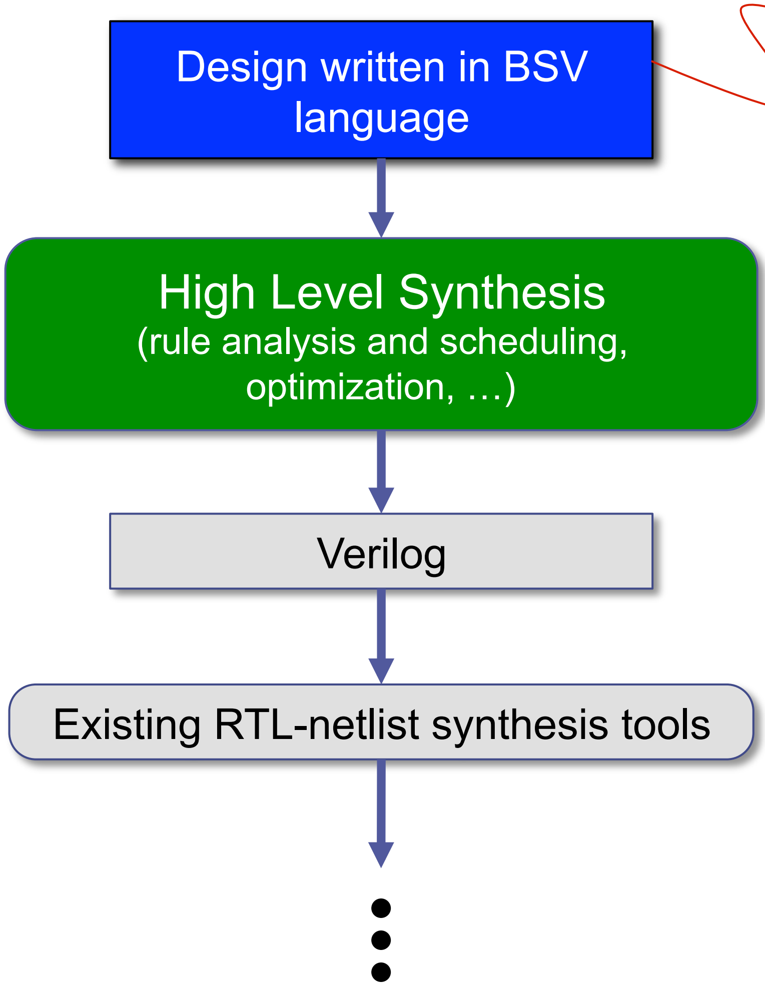
Responses to the moderators' three questions:

1. *What works? What doesn't?*
2. *Future evolution? Growth? Into new areas?*
3. *Future research opportunities?*



0: Background about Bluespec BSV approach to HLS

Borrow best modern ideas from programming languages, formal verification systems, and concurrency.
Don’t be hamstrung by sequential von Neumann legacy.



Design written in BSV language

High Level Synthesis
(rule analysis and scheduling, optimization, …)

Verilog

Existing RTL-netlist synthesis tools

Behavior spec: Guarded Atomic Transaction Rules

- cf. Guarded Commands, TLA+, UNITY, EventB, …
- Fundamentally parallel/concurrent

Architecture spec:

- cf. Haskell functional programming language
- strong type-checking, polymorphic types, typeclasses, higher-order functions, modularity, parameterization

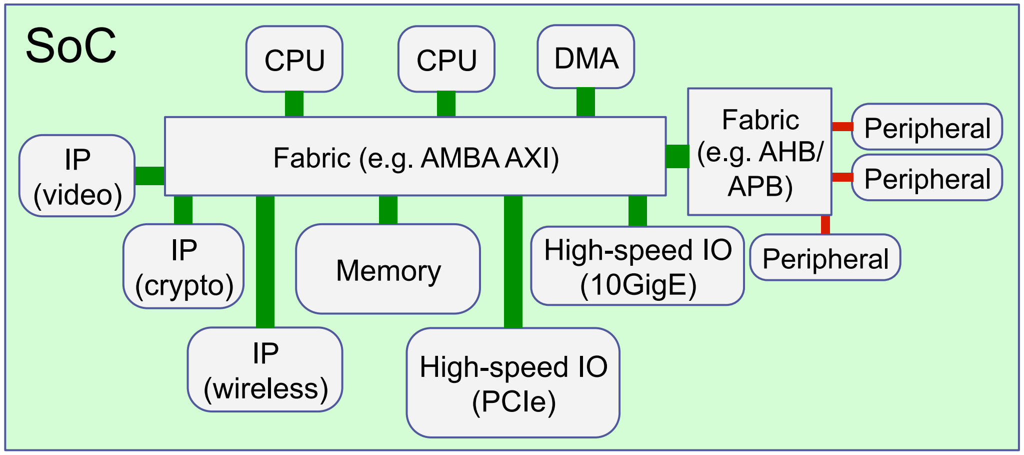
All HL language features available in synthesizable code (no subsetting for synthesis)

- In HW design, no fundamental separation between algorithm design and architecture design
  - Architecture ⬄ cost model ⬄ Algorithm Design
- BSV approach is completely architecturally neutral (no bias towards sequential von Neumann)
- Suitable for high performance data processing and complex control (in short: for anything for which you might previously have used RTL)

Q1: What works? What doesn't?

BSV HLS has worked well for designing IP blocks of every kind (expected).

BSV users don't (voluntarily!) go back to RTL.



Plus: several new use models for BSV HLS:

- *Synthesizable* models, for
  - Architecture exploration
  - Early firmware development and testing
  - Early SW development and testing
- *Synthesizable* Verification Environments

E.g., Existing CPU/SoC models in BSV:

- ARM (many versions), x86, PPC, Power, MIPS, Sparc, Alpha, RISC-V, JVM, Itanium, …
- All synthesized to FPGA, many of them booting a full OS (Linux/Sparc/xBSD/…)

Why?

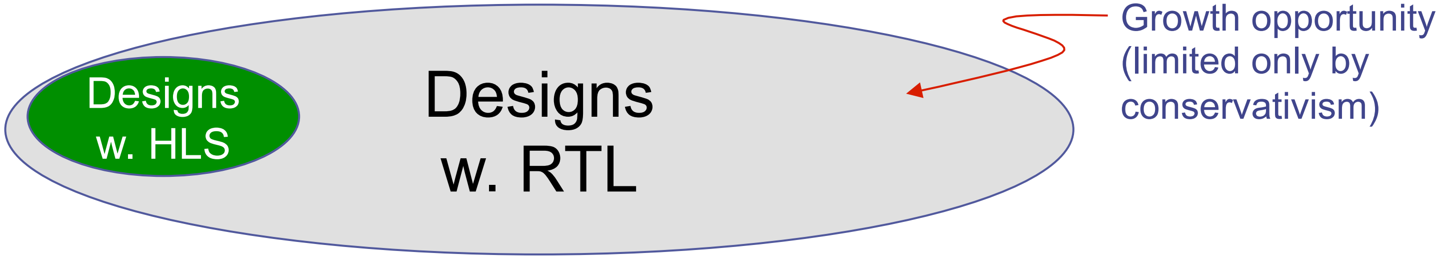
- Old way (simulation) is 1Kx – 1Mx too slow! New way: run on FPGA ➔ need synthesizability from High-Level Language
- Models need architectural credibility; not easy to achieve in non-synthesizable language

Both points achievable with BSV

Only conservatism prevents wider adoption

- Risk perception (esp. for small vendor)
- "Unfamiliar" language
- Unfamiliarity with modern ideas in programming languages (types, abstraction, advanced parameterization)

Q2: Future evolution? Growth? Into new areas?



Central rôle in whole-SoC design, using FPGAs:
- Modeling and architecture exploration
- Verification
- Early SW development and testing

This cannot happen without HLS

High-Performance Computing (HPC) using FPGAs
- FPGAs can be just as valuable as GPGPUs have become for HPC in science and engineering

This cannot happen without HLS

Q3: Future research opportunities?

Atomic Transaction Rules: Language, Synthesis

- Higher-level languages than BSV using Rules
- Better, more expressive scheduling of concurrent rules
- Pay the communications piper by moving towards more asynchronicity (GALS/GALA/latency-insensitivity/ dataflow). Rules are a natural fit for this.

Automatic Power Management:

- Rule semantics provides high-level information about when circuits are active/idle

Make FPGA environments easier to use than GPGPUs:

- “instant” synthesis
- incrementality
- service APIs
- full visibility)

[No fundamental technical obstacle]

Use Rule Semantics to change HW/SW interface:

- Device drivers are notoriously difficult, buggy. Perhaps because DDs are highly reactive, concurrent programs, for which C may be the wrong model.

Libraries and IP generators:

- BSV’s powerful parameterization and type system permit creation of STL-like libraries
- BSV makes an excellent target for IP generators. See Papamichael et. al. (CMU) for generators for NoCs (“CONNECT”) and Memory Hierarchies (“CoRAM”) using BSV as a back end

Formal methods exploiting Rule semantics:

- Automation of “design by refinement” from models to implementations
- Formal verification
- Formal testing, like Haskell Quickcheck

[Application] CPU and System Architecture research enabled by BSV:

- Fast execution on FPGAs with credible accuracy
- Already happening at Intel, IBM, DARPA CRASH/ SAFE project, Supercomputing Center Barcelona, IIT Chennai, …

bluespec™

Thank you!