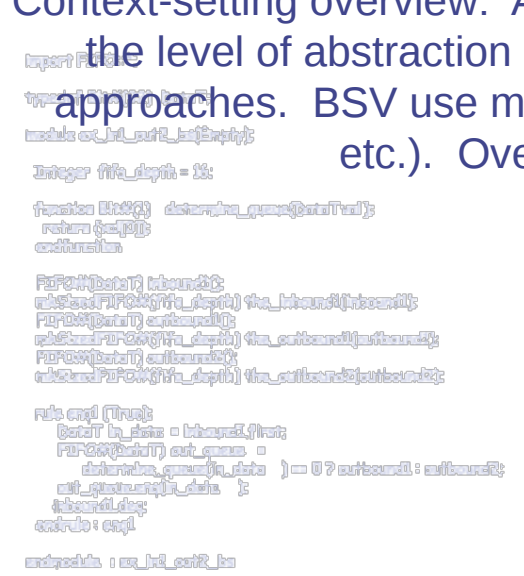
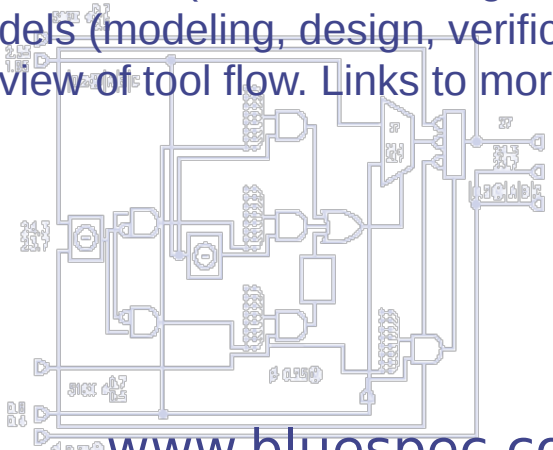
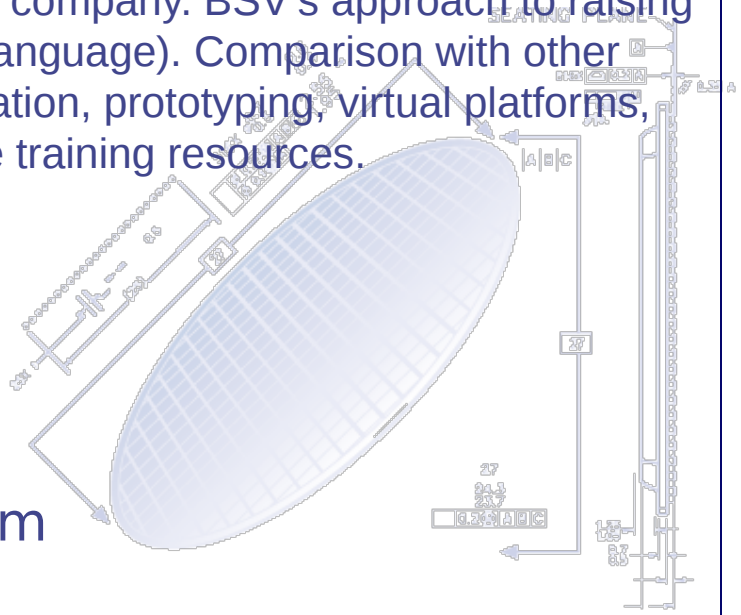
# BSV Training

## Lec_Intro

Context-setting overview: A word about Bluespec, Inc., the company. BSV's approach to raising the level of abstraction as an HDL (Hardware Design Language). Comparison with other approaches. BSV use models (modeling, design, verification, prototyping, virtual platforms, etc.). Overview of tool flow. Links to more training resources.







www.bluespec.com

# Bluespec, Inc. company overview

- Founded in 2003
- HQ and engineering in Framingham, Massachusetts; worldwide presence
- Patented technology: proven, mature, and shipping since 2005
  - Technology roots in MIT research (atomic rule synthesis) and Haskell, a modern functional programming language (for types, parameterization, static elaboration)

## Customer examples

Panasonic ideas for life, HITACHI Inspire the Next, Texas Instruments, IBM — Large networking company

FUJITSU, Alcatel·Lucent — Large search company

SAMSUNG, ST — Large microprocessor company

And, many more!

## Vibrant research partnerships

DARPA, SRI International, Open CPI, UCSD, Stanford, The University of Texas at Austin, Carnegie Mellon University, MIT, University of Washington, Cambridge, Oxford

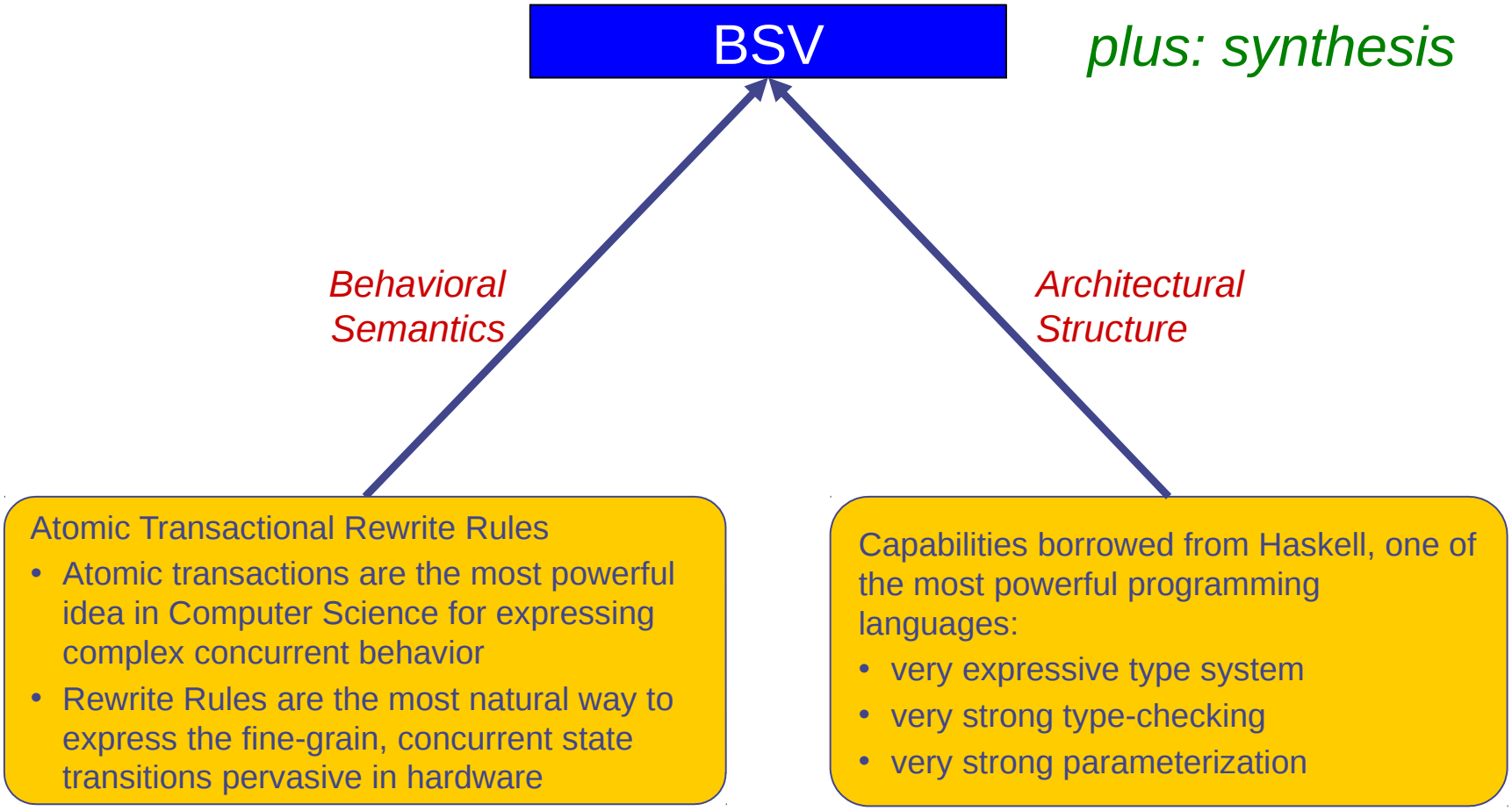
Over 50 universities teaching/doing research with Bluespec!





bluespec

# BSV: based on advanced Computer Science innovations



# BSV: fundamentally different approach to HW design

Structural expressiveness:

- Unlike most HDLs (other than Lava and Chisel), BSV is based on *circuit generation* rather than merely *circuit description*.
- 'Generate' is not an afterthought but an organic part of the language.
- 'Generate' is powerful—a full higher-order functional language with very powerful parameterization (~ Haskell)
- Expressive polymorphic types with strong static type-checking (~ Haskell)

Behavioral expressiveness:

- Unlike other HDLs (even Lava and Chisel), BSV is based on *atomic transactional rules* instead of a globally synchronous view of the world

- Atomic Transactional Rules
  - “Event centric”, “Reactive”, “elastic”, “Method-based module communication”
  - Almost ``asynchronous'' in mindset (even though compiled to synchronous logic)
  - Scalable reasoning, even across module boundaries
  - Compositional
  - Amenable to formal reasoning, proofs

- vs. Purely Synchronous view:
  - “State centric”, signal-based module communication
  - Globally synchronous reasoning: difficult to scale robustly, fragile composition

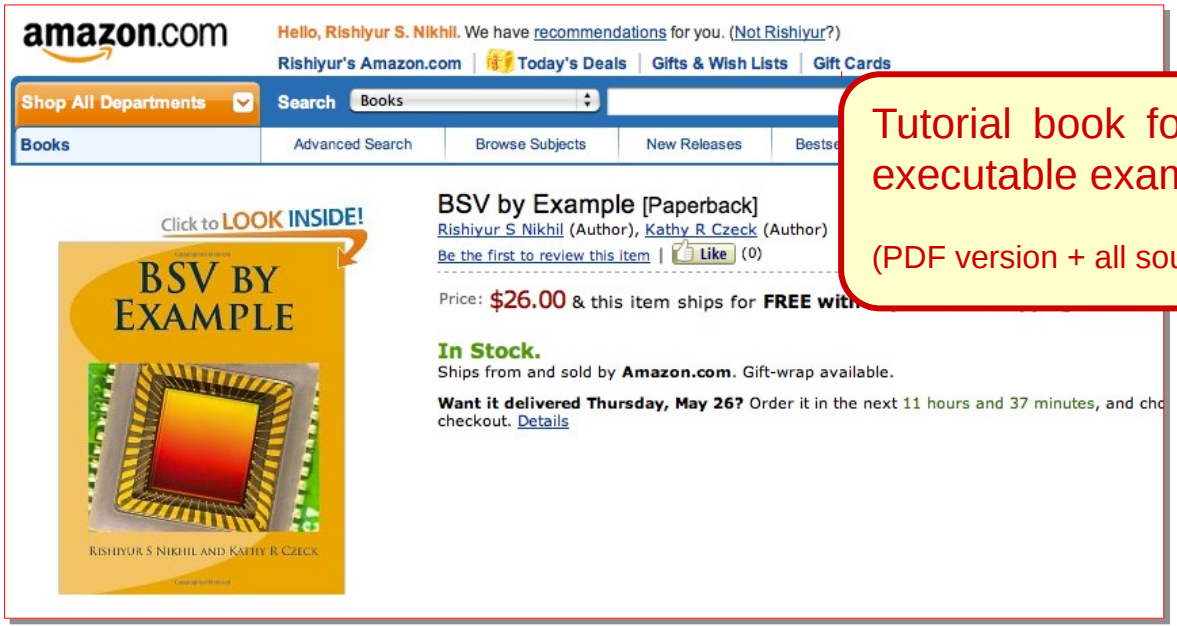
# BSV

- BSV is not like classical “HLS” (High Level Synthesis), where the source language (C/C++) has a quite different computation model (and algorithmic cost structure) from the target (hardware)
  - BSV is architecturally transparent: you are in full control of architecture and there are no architectural surprises
  - With BSV you think hardware, you think about architectures, you think in parallel

- BSV is “universal” in applicability (like traditional HDLs). BSV has been used for CPUs, caches, coherence engines, DMAs, interconnects, memory controllers, DMA engines, I/O devices, security devices, RF and multimedia signal processing, and all kinds of accelerators.

  - Since 2000, in several major companies and universities worldwide

# Learning effort is comparable to other modern languages




Tutorial book for self-learning, with small, fully executable examples.

(PDF version + all source codes available free from Bluespec)


UCSD

THE UNIVERSITY OF TEXAS AT AUSTIN

Carnegie Mellon University

Massachusetts Institute of Technology

UNIVERSITY of WASHINGTON

BSV is in use in over 50 universities worldwide, including top-tier universities.

(See article in Xilinx Xcell Journal 3Q 2011 on remarkable projects accomplished by students in a 1-semester MIT course.)

# Comparing BSV's approach to other HDLs

| *Behavioral Semantics* | BSV | Synthesizable RTL (Verilog, VHDL, SystemVerilog, SystemC) | "HLS" from C/C++/Matlab |
|---|---|---|---|
| Behavior | Rules (atomic) | Synchronous circuits | Sequential programming |
| Interfaces | Object-oriented methods (atomic) | Wires (few TLM interfaces) | N/A (few predefined interfaces for top-level) |

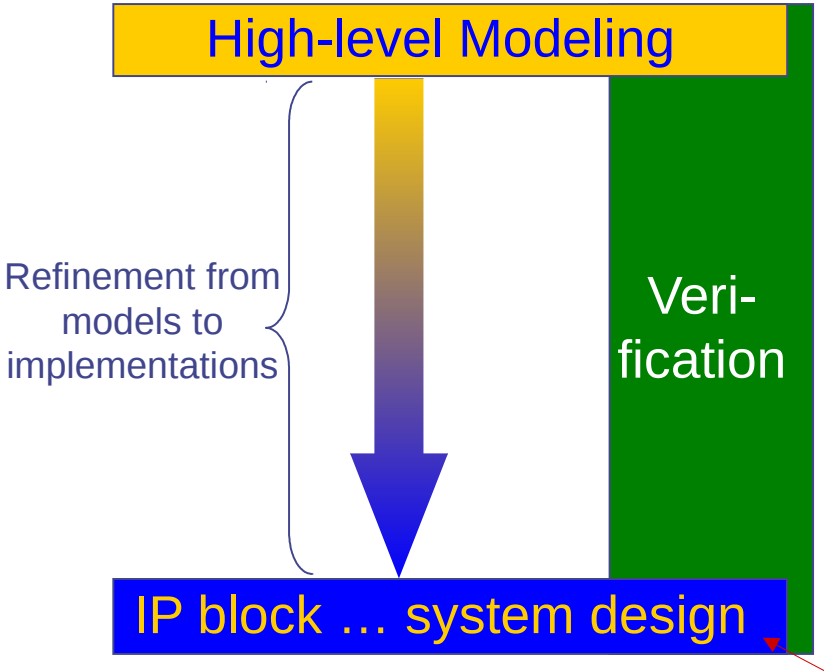
| *Structural abstractions* | BSV | Synthesizable RTL (Verilog, VHDL, SystemVerilog, SystemC) | "HLS" from C/C++/Matlab |
|---|---|---|---|
| Architectural transparency | Strong | Strong | Weak |
| Type-checking | Strong | Weak/Medium | Medium |
| Types | Powerful user-defined types | Bits, weak user-defined types | Weak user-defined types |
| Parameterization | Powerful | Weak | Weak |

Note: Last two columns only consider *synthesizable subsets* (e.g., not SystemC TLM). Higher-level constructs are available if you are only interested in simulation.

# BSV use models

*BSV is used for modeling:*

- *E.g., processor architecture models in BSV include MIPS, Sparc, x86, Itanium, ARM, PowerPC, TenSilica, RISC-V, JVM, and some more*
- *All of them synthesized and running on FPGAs*
- *Many of them executing real apps and OSs (Linux, …)*




*BSV is used for verification of complex IPs:*

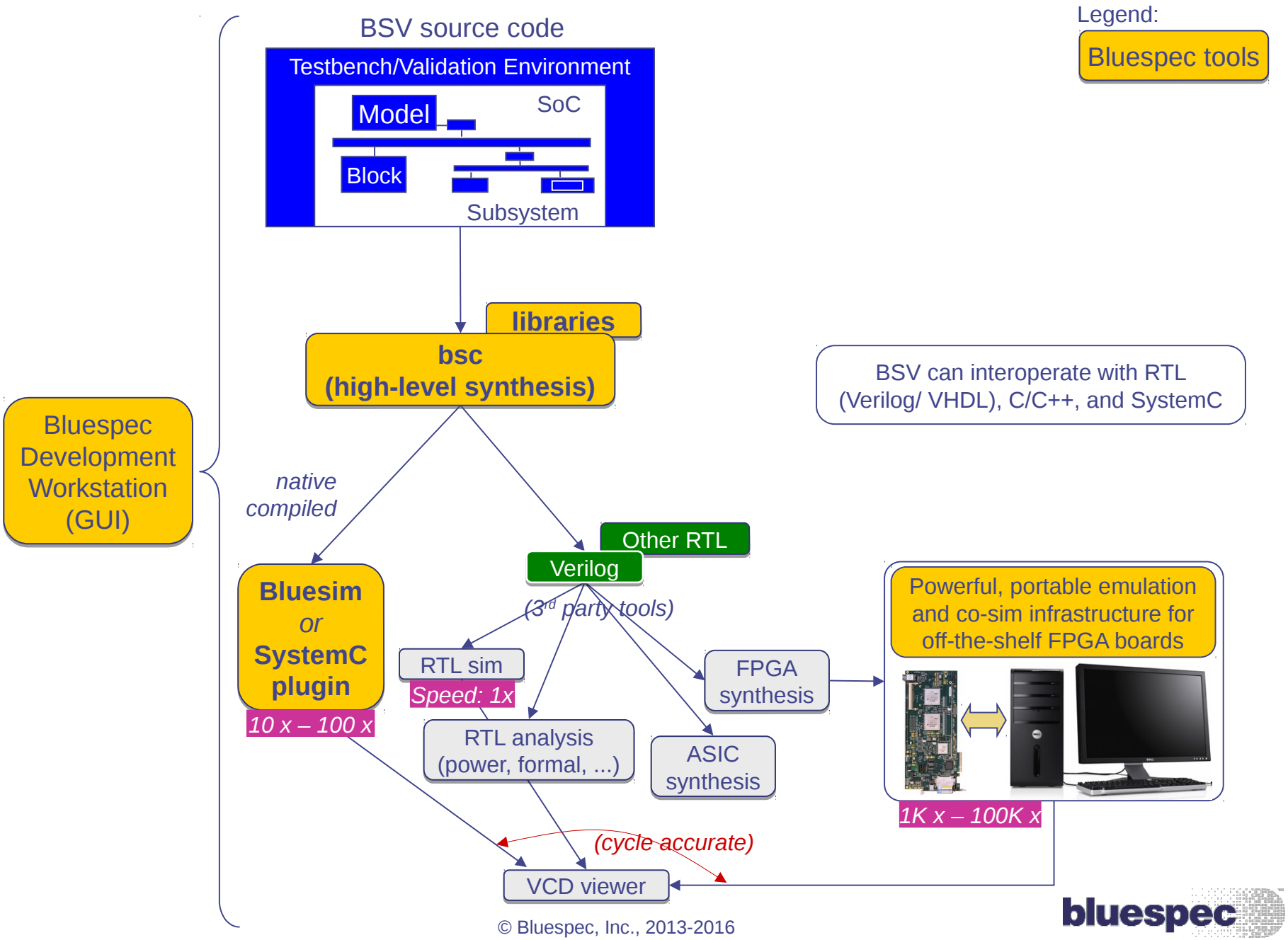
- *E.g., transactors for PCIe Gen 3, multi-core cache-coherent processor interconnect and AXI*
- *All synthesized, and running on FPGAs*

*BSV is used for complex IPs:*

- *E.g., IPs in commercial mobile devices (phones/ tablets) and set-top boxes, involving both high-speed datapaths and complex control*

*Note: unlike BSV's broad range of use models, other high-level synthesis tools are only used for IP design, and only for signal-processing IPs (datapath, not control)*

# BSV tool flow: core tools



# Resources

- Language reference guide: $BLUESPEC_HOME/doc/BSV/reference-guide.pdf
  - Complete reference on the BSV language (syntax, semantics, all language constructs, scheduling annotations, importing C and Verilog, extensive libraries
- Tool usage guide: $BLUESPEC_HOME/doc/BSV/user-guide.pdf
  - How to use BDW (Bluespec Development Workstation), how to compile and link using *bsc*, how to simulate using Bluesim and Verilog sim, how to generate and view waveforms, etc.
- Examples, lecture slides, training materials:
  - https://github.com/rsnikhil/Bluespec_BSV_Tutorial.git
- BSV-by-Example book (authors: Nikhil and Czeck):
  - Around 60 examples, each focusing on one topic, with ready-to-run source code
  - Hardcopy version: purchase at Amazon.com
  - Free PDF of book: $BLUESPEC_HOME/doc/BSV/bsv_by_example.pdf
  - All the example code: $BLUESPEC_HOME/doc/BSV/bsv_by_example_appendix.tar.gz
- General questions about BSV, the tools, anything:
  - User Forums at bluespec.com  (free, after registration)
  - E-mail to  'support@bluespec.com'

# Lecture slide decks reading guide

The topic-based lecture slide decks in the "Reference/" directory are intended as a reference, and need not be read sequentially.

However, people learning BSV on their own for the first time may wish to read them in the following order:

- Lec_Intro
- General intro to the Bluespec approach, and some comparisons to other Hardware Design Languages and High Level Synthesis.
- Lec_Basic_Syntax
  Gets you familiar with the "look and feel" of BSV code.
- Lec_Rule_Semantics, Lec_CRegs
- These two lectures describe BSV's concurrency semantics (based on rules and methods). This is the KEY feature distinguishing BSV from other hardware ands of software languages.
- Lec_Interfaces_TLM, Lec_StmtFSM
  These two lectures describe slightly advanced constructs: more abstract interfaces, and more abstract rule-based processes.
- Lec_Types, Lec_Typeclasses
  These two lectures describe BSV's type system, which is essentially identical to that of the Haskell functional programming language.
- Lec_BSV_to_Verilog
  Desribes how BSV is translated into Verilog by the bsc tool. Read this only if you are curious about this, or if you need to interface to other existing RTL modules.
- Lec_Interop_RTL
  How to import Verilog/VHDL code into BSV, and how to connect BSV into existing Verilog/VHDL.
- Lec_Interop_C
  How to import C code into BSV (for simulation only). How to export a BSV subsystem as a SystemC module (for use in a SystemC program).
- Lec_Multiple_Clock_Domains
  How to create BSV designs that use multiple clocks or resets.
- Lec_RWires

  A follow-up to Lec_CRegs, showing lower-level and stateless primitives for greater concurrency

# End