

It's Not The Tools, It's the Language

The Effective Depreciation of RTL through Bluespec SystemVerilog

Shep Siegel

shepard.siegel@atomicrules.com

Previously: "An Individual Engineer's Opinion of the Most Usefully-
Disruptive Change to the Business of Circuit Design since RTL"



Agenda

- Motivation
- Landscape: RTLs and ESLs
- Language Features
- Target Independence
- Exploration, Productivity and Correctness
- Abstract Inside
- Platform Blue
- Q&A

Motivation

- Digital Systems Design: Need to do more with more, in less time...
- “The Productivity Gap”
- Evolution of schematics, netlists, PALs, ASICs, FPGAs, RTLs...
- Now what?

Productivity

Time-to-Proficiency → Time-to-Solution

Correctness and Scalability

Enjoyable Practice: Innovation can be fun!

Let the compiler do the mundane, error-prone parts

RTLs

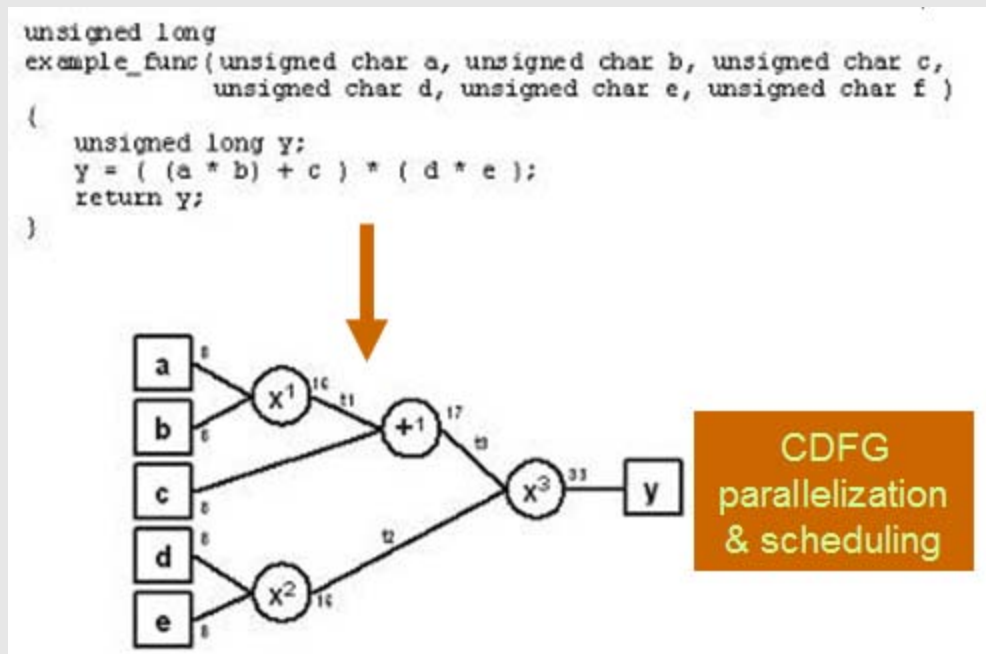
- VHDL and Verilog have been around for a long time
 - System Verilog (SV) more contemporary
- Established and Stable: are not going to go away (or change much)
- Can describe behavior and/or structure
- An excellent intermediate platform (stable and sufficient)
 - Think of RTL as a technology-independent *"byte code"*
- Lack contemporary software-language features
- Will that signal/variable/wire be a register's output?
 - RTLs infer registers based on assignments from clocked, concurrent processes
- Clock cycle-based abstraction scales poorly
 - Fine for simple circuits in isolation
 - System-level reasoning about clocks can be challenging

RTL Balance Sheet

- Pros
 - Established and stable
 - Can express behavior or structure
 - Today's mainstream design expression for FPGA/ASIC
 - Sequential, imperative process semantics for verification
- Cons
 - Lack contemporary (software) language features
 - Abstracting beyond the clock-cycle (e.g. TLM) is difficult
 - Not so concise (perhaps 5x as many source lines vs. 'C')
 - Sequential, imperative expression difficult to synthesize

ESL: “C to Gates”

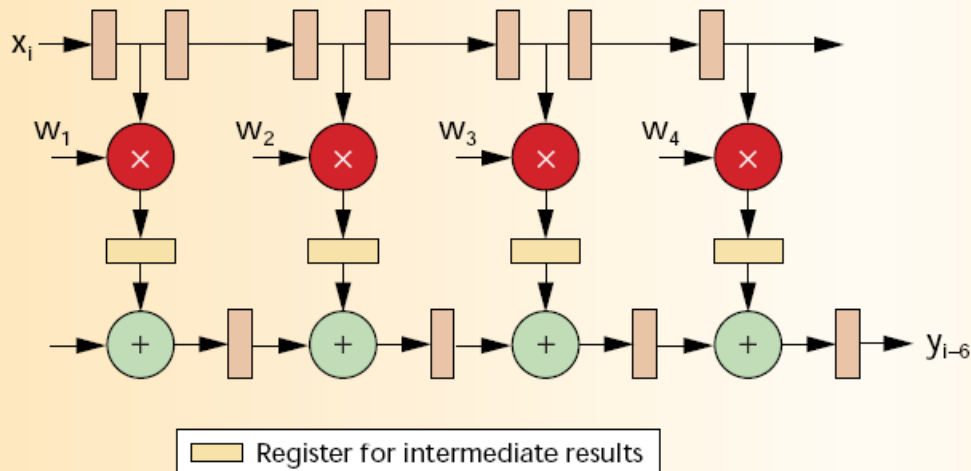
- Most C to Gates ESLs are CDFG elaborators
 - Function signature translated to circuit inputs and outputs
 - Constraint-driven CDFG parallelization and scheduling



“C-to-Gates” ESL Balance Sheet

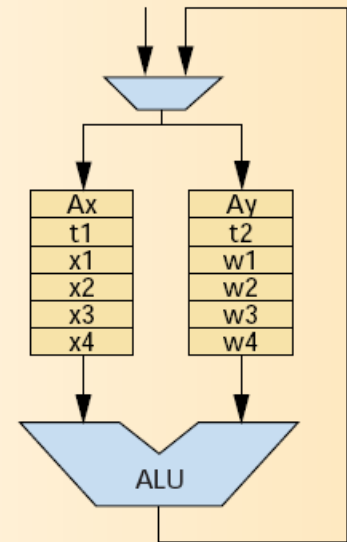
- Pros
 - Familiar syntax (C or C-like)
 - Familiar behavioral expression
- Cons
 - (Possibly) Non-ANSI language features (e.g. pragmas)
 - Difficult to predict F_{MAX} and Area (historically poor)
 - A sequential, imperative expression of a concurrent behavior...
 - C-to-gates does not bring “scalable atomicity” any more than an imperative language supports threads. In some cases may propagate “*The Problem with Threads*” [Lee2006] into the hardware camp.

Sidebar: Spatial v. Temporal



```

x4 ← x3 // x[i-3]
x3 ← x2 // x[i-2]
x2 ← x1 // x[i-1]
Ax ← Ax + 1
x1 ← [Ax] // x[i]
t1 ← w1 × x1
t2 ← w2 × x2
t1 ← t1 + t2
t2 ← w3 × x3
t1 ← t1 + t2
t2 ← w4 × x4
t1 ← t1 + t2
Ay ← Ay + 1
[Ay] ← t1
    
```



[DeHon2000]

- Digital design is a trade-space balancing act
 - Revisit your motivation for a h/w vs. s/w solution
- Most FPGA applications *require* some spatial expression to achieve sufficient speedup over common (x86) ISAs

Segue and Analogy

- If not RTL, then what?
 - And why?
- “balance” and “simplicity” often matter
- Just because a new technology excels at something doesn't make it necessary
- RTL is certainly *good enough* for many things;
 - like 'C' is *good enough* for Hello, world.
- But what if you have a tough problem that needs to scale?
 - A Sudoku solver, for example: 'C', or your favorite OOP-lang?

Problem: Abstraction and Detail

- How does one reconcile the seemingly conflicting objectives of
 - A) high-level, abstract, scalable, component-based design with
 - D) complete control over every single bit of state in the system?
 - Why? Both are important concerns!
 - A) Correct-by-Construction, Scalability, Reuse
 - D) F_{MAX} and Area (LUTs, FFs) matter; μ Architecture matters
- (Note: Quick dive into the deep end, hold on...)

Solution: Scalable Atomicity (1/2)

- *Scalable Atomicity* is the ability to reason about discrete, step-wise, “atomic” behaviors independent of scale
- Consider what actions govern the change of state of the following circuit elements:
 - A single flip-flop
 - A counter
 - A FIFO
 - A completion buffer
 - A endpoint or root-complex
 - A chip, board, or system’s level aggregation of these and other circuits
- RTL View: Clock-cycle reasoning is a super-linear effort
 - With more bits, modules, and interactions; comes an exponentially greater burden on the design and verification

Solution: Scalable Atomicity (2/2)

- *Scalable Atomicity* in a design language is the ability to describe atomic, state-changing, rule-step actions at any level of hierarchy
 - Updating the state of a flip-flop
 - ENQ or DEQ from a FIFO
 - An interface transaction
 - A system-level transaction
- Rules either execute completely (“fire”) or not, consequently
 - Behavior (and changes to behavior) are concise
 - Complex interactions are no-longer intractable

(We dove into the deep-end, now back to basics...)

Bluespec SystemVerilog (BSV)

- Research in term rewriting systems (TRS), λ -calculus, and functional programming helped set the stage for BSV
- Bluespec Inc. sells a compiler that translates the BSV language into technology-agnostic IEEE Verilog RTL
- BSV looks similar to SV syntax
 - Uses as much SV syntax as practical
 - No procedural `always@(posedge CLK)` blocks (rules instead)
 - Interfaces are composed of methods (rule semantics again)

Rules

- Rules make it easier to reason about segments of code that can change the state of a circuit
- They either FIRE or they don't – all or nothing → Atomic!
- Rules can have an *explicit* conditions, like a predicate
- And *implicit* conditions, such as the readiness of a method within the rule's context
- The BSV compiler generates a deterministic schedule to resolve rule and resource conflicts

Atomicity and Energy Efficiency

- Consider any sequential circuit that has a requirement for computational energy efficiency
- If you could wrap-up the costly operation in a rule, which only “fires” when there is real work to be done...
- Reasoning about energy efficiency might be easier
 - If you actually produce a circuit that is more or less efficient is up to your design
- Generally, expect some improvement
 - Easy to apply VSS or VDD when not fired

BSV Language

“... there are two ways of constructing a design: One way is to make it so simple that there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies. The first method is far more difficult.”

–Tony Hoare, 1980 ACM Turing Award Lecture

- BSV brings many of the features of its Haskell-based roots
 - It is concise
 - It has powerful type-classes (polymorphic interfaces)
 - It is functional programming with explicit state
 - As a result, initial language “Time-to-Proficiency ” issue for some

(almost) Everything Synthesizes

- Unlike RTL, BSV does not have a “synthesizable subset”
- If the interface can be ripped to bits, it can be synthesized
 - Minor exceptions ($\$display$ calls, file I/O)
- Allowing the designer to easily perform unit-level regressions as part of the edit-compile-build cycle provides:
 - Reduction in the number of unit-level defects
 - Continuous feedback on F_{MAX} and area
 - A comfort zone “sandbox” in which to explore

Target Substrate Independence

- The BSV compiler emits technology-agnostic RTL
- If you create a vector of registers and incrementally nudge up the width and depth of the register file, the RTL will look substantially the same...
- ...but if you observe the technology mapping from the RTL synthesis tool { *Synplify* | *XST* | *Quartus* }, you will see the implementation change from discrete registers, to distributed ram, to BRAM or MRAM
- BSV source codes are insulated from substrate-specific features, unless they wish to expose them

Exploration, Productivity, and Correctness

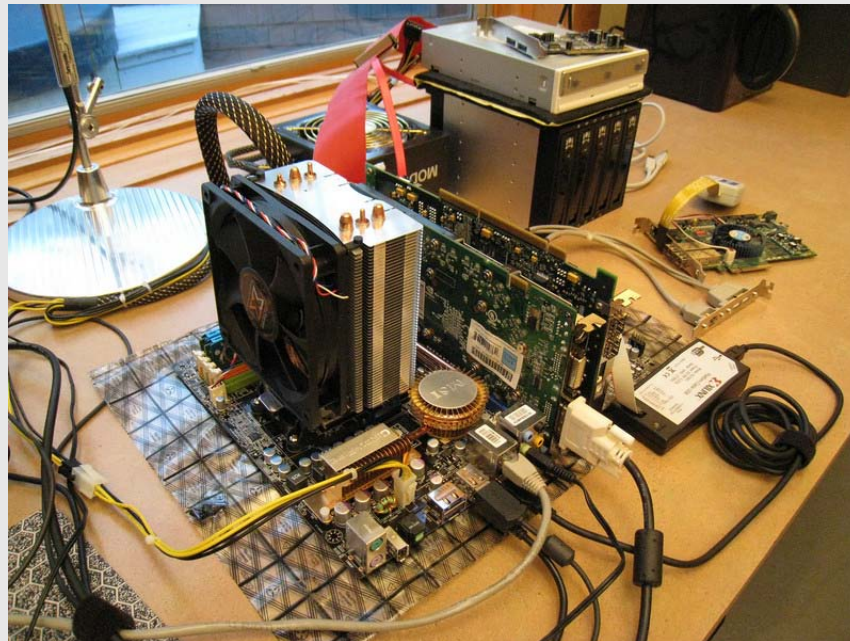
- Beyond any specific feature, BSV allows the digital circuit designer to easily explore and measure alternatives
 - With RTL, you often have so much “housekeeping”, that a specific μ Architecture has all but been pre-selected from the start
 - you get what you get
 - With BSV, I find myself frequently exploring different architectures for the same module – and often gaining valuable insights I would otherwise have motored past
- BSV allows the designer to separate correct “*function*” from correct “*performance*”
 - Verification Impact: Fewer crosscutting concerns

Abstract Inside

- Get-Put semantics allow SDF/KPN behavior without presupposing any particular (e.g. FIFO) implementation
- Core IP may be coded agnostic to any particular protocol
- Healthy exercise to help precipitate the designer's distinction of interface and implementation characteristics
- Still more helpful orthogonalization
 - Adjust interfaces separately from implementation
 - Verification: Test interfaces separately from implementation
- Late-stage ECO and tech refresh costs lowered

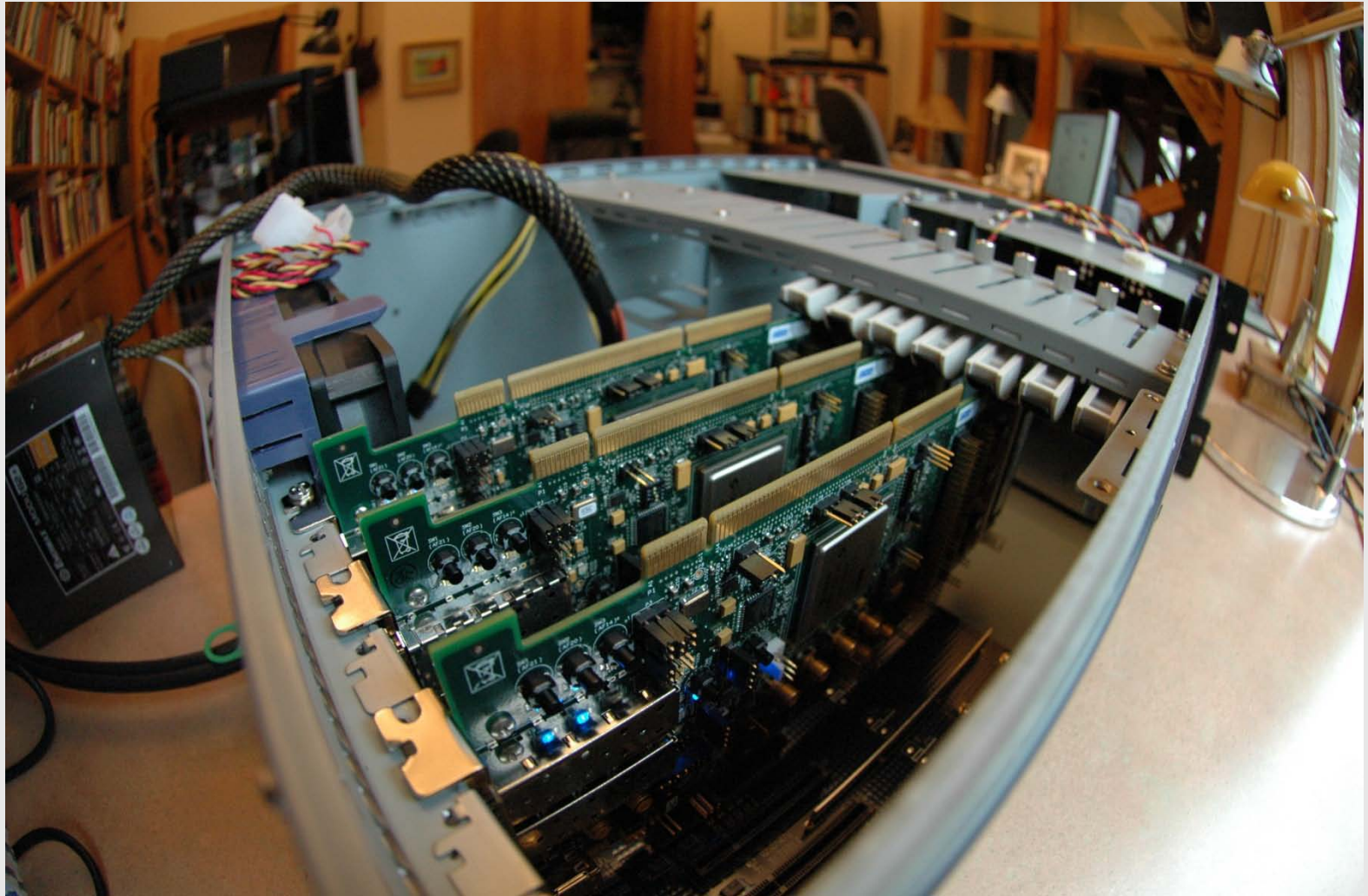
Platform Blue

- Interoperability and performance test platform
- Common BSV source on different vendor's FPGA silicon
- Measure latency and throughput in different deployment scenarios



[Siegel2008]

3-Node V5 ML555 Scale-Up



Learning More

- The materials on the Bluespec website [BSV] are good
- The 13-lecture, 3-day BSV training is much better
 - Appreciate the lectures and read the “reference-guide”
- MIT teaches BSV in [Digital Systems 6.375](#)
 - Online courseware and [projects](#)
- Actually solving your own (sufficiently hard) problem is best

Homework Question

- Give one way in which describing a design in Bluespec is superior to writing RTL level code. And give one way in which describing a design at the RTL level is superior to Bluespec.

Thank you!

Shepard Siegel, CTO

Atomic Rules LLC

287 Chester Rd

Auburn, NH 03032

Shepard.Siegel@atomicrules.com



Acknowledgements

These companies (alphabetically) have provided support for this work:

- Altera – fpga dev kit and Quartus software
- Bluespec – BSV compiler and Bluespec workstation
- Mentor – ModelSim simulator
- PLX - PCIe switch models
- Synopsys – Synplicity/SynplifyPro synthesis
- Xilinx – fpga dev kit and ISE software



References

[BSV] Bluespec Inc. www.bluespec.com

[Lee2006] [“The Problem With Threads”](#)

[DeHon2000] [“The Density Advantage of Configurable Computing”](#), Andre DeHon, IEEE Computer April 2000

[Rishiyur2007] [“Bluespec Sudoku”](#)

[Siegel2008] Author’s blog www.atomicrules.blogspot.com