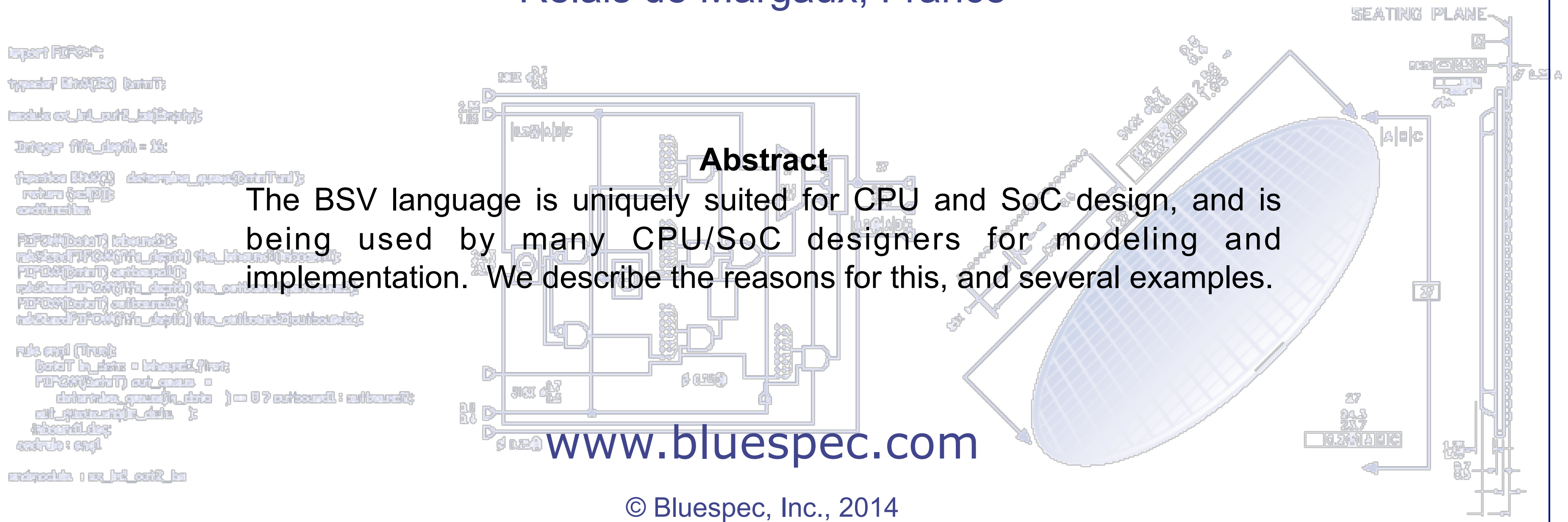
Bluespec BSV, the choice for CPU and SoC designers

Rishiyur Nikhil
CTO, Bluespec

MPSoC, July 2014
Relais de Margaux, France

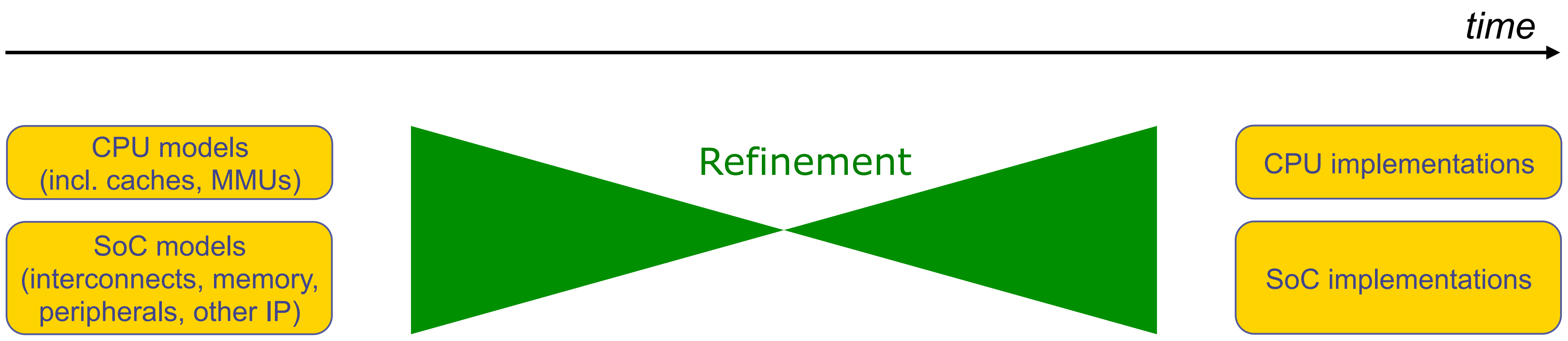


Abstract

The BSV language is uniquely suited for CPU and SoC design, and is being used by many CPU/SoC designers for modeling and implementation. We describe the reasons for this, and several examples.

www.bluespec.com

Range of CPU/SoC activities where BSV is used


time
CPU models (incl. caches, MMUs)
SoC models (interconnects, memory, peripherals, other IP)
Refinement
CPU implementations
SoC implementations

Activities throughout the design period

Debugging
Performance analysis and validation
Software/firmware development and testing

Ideally, with actual SW loads (full OS, apps, …)

Execution on FPGAs
(even for eventual ASIC targets)

Requirements for source language:

- *Natural expression of architectures, from models to implementation (high-level)*
- *Continuum from high-level to implementation-level (no disruptive changes)*
- *Full synthesizability (to FPGAs), from models to implementations*

Why people use BSV

What’s so special about BSV?

- Fundamentally parallel/concurrent language, no sequential “base language”
- Architectural Predictability and Controllability
  - Architectures (like algorithms) are the creative part of design; cannot be entrusted to a tool!
- BSV “Rules” are a natural abstraction of hardware behavior
  - Concurrent state machines on shared resources
  - Rules are ***composable, scalable, atomic state transitions, in object-oriented style***
- BSV “Rules” feed naturally into Formal Verification
  - Rules are the basic logic model in many formal verification systems

If you’d like more technical detail, see code examples, see a demo, … please catch me off-line.

- Most powerful data type system in any language for synthesizable hardware design
  - User-defined types, functional types, polymorphism, typeclasses (user-defined overloading)
  - Custom hardware bit-representations
- Most powerful static elaboration system in any language for synthesizable hardware design
  - Higher-order functions
  - Full orthogonality ➔ very powerful parameterization
    - Can parameterize {functions, modules, rules, actions} by {functions, modules, rules, actions}

What's so special about BSV (contd.)?

- Fully synthesizable (no concept of "synthesizable subset")
  - *All* high-level features of the language can be used in synthesizable code

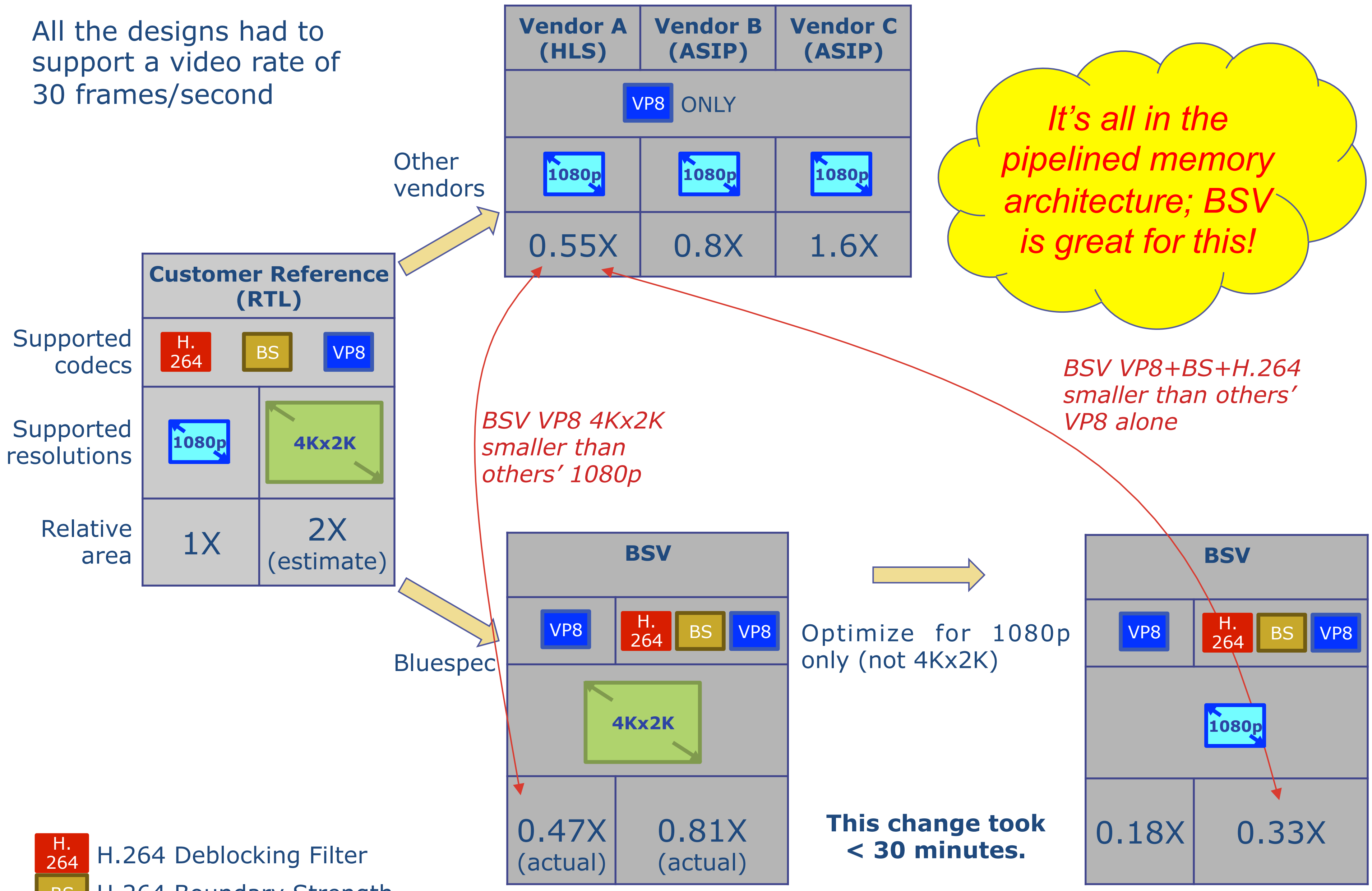
- Both computation-oriented and control-oriented designs handled very well
  - Computation: image, video, wireless, crypto, …
    - Highly complex pipelines, pipeline control, customized memory systems
  - Control: CPUs, caches, MMUs, DMAs, interconnects, memory controllers, high-speed I/O, network processing, …

If you'd like more technical detail, see code examples, see a demo, … please catch me off-line.

bluespec

H.264 Boundary Strength; H.264+VP8 Deblocking

All the designs had to support a video rate of 30 frames/second



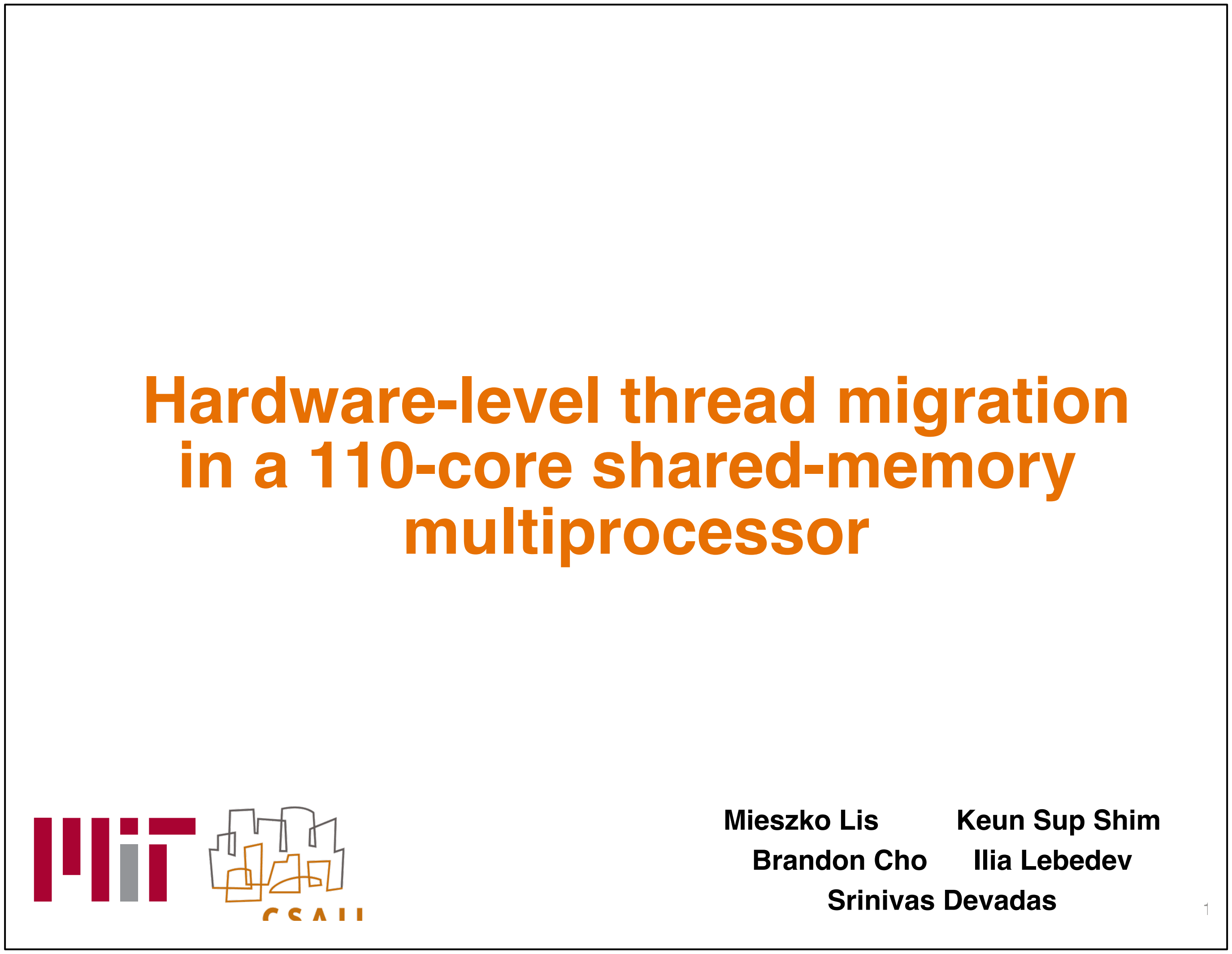
H. 264	H.264 Deblocking Filter
BS	H.264 Boundary Strength
VP8	VP8 Deblocking Filter

Examples of who's using BSV for CPU/SoC design

Organization	Type of model(s)	Comment	on FPGA?	OS?
Bluespec, Inc.	ARM (9, Cortex, …)	Synthesizable ISS model	Yes	Linux
	RISC-V	Synthesizable ISS model and pipeline implementation	Yes	Linux
Intel	[proprietary]	Cycle-accurate CPU pipeline models	Yes	Yes
IBM	Next-gen POWER	Components	Yes	
MIT/IBM	PowerPC	Cycle-accurate CPU pipeline models	Yes	Linux
MIT	110-core single-chip SMP	Exploring hardware-assisted thread migration	ASIC	Apps
U.of Cambridge (UK) and SRI (CA)	MIPS + security enhancements	With formal verification of security enhancements	Yes	FreeBSD
U.of Pennsylvania	Clean-slate design for security and formal proofs of correctness	With complete formal verification	Yes	Planned
IIT Madras (Chennai)	ANURAG	FPGA CPU implementation	Yes	?
	MIPS/RISC-V	Pipelined implementation	Yes	?
Carnegie-Mellon University	Itanium	CPU model, 4000x faster than Modelsim	Yes	No
	16 CPU UltraSparc III Sunfire 3800 Server	Synthesizable hyper-threaded ISS model	Yes	Solaris 8, Oracle 10g Enterprise DB Server
U.of Texas (Austin)	x86	Cycle-accurate CPU pipeline models	Yes	Windows and Linux
U.of Lund (Sweden)	JVM	Direct HW exec of JVM bytecodes, garbage collection	Yes	

Examples of who's using BSV for CPU/SoC design

Hot Chips 2013



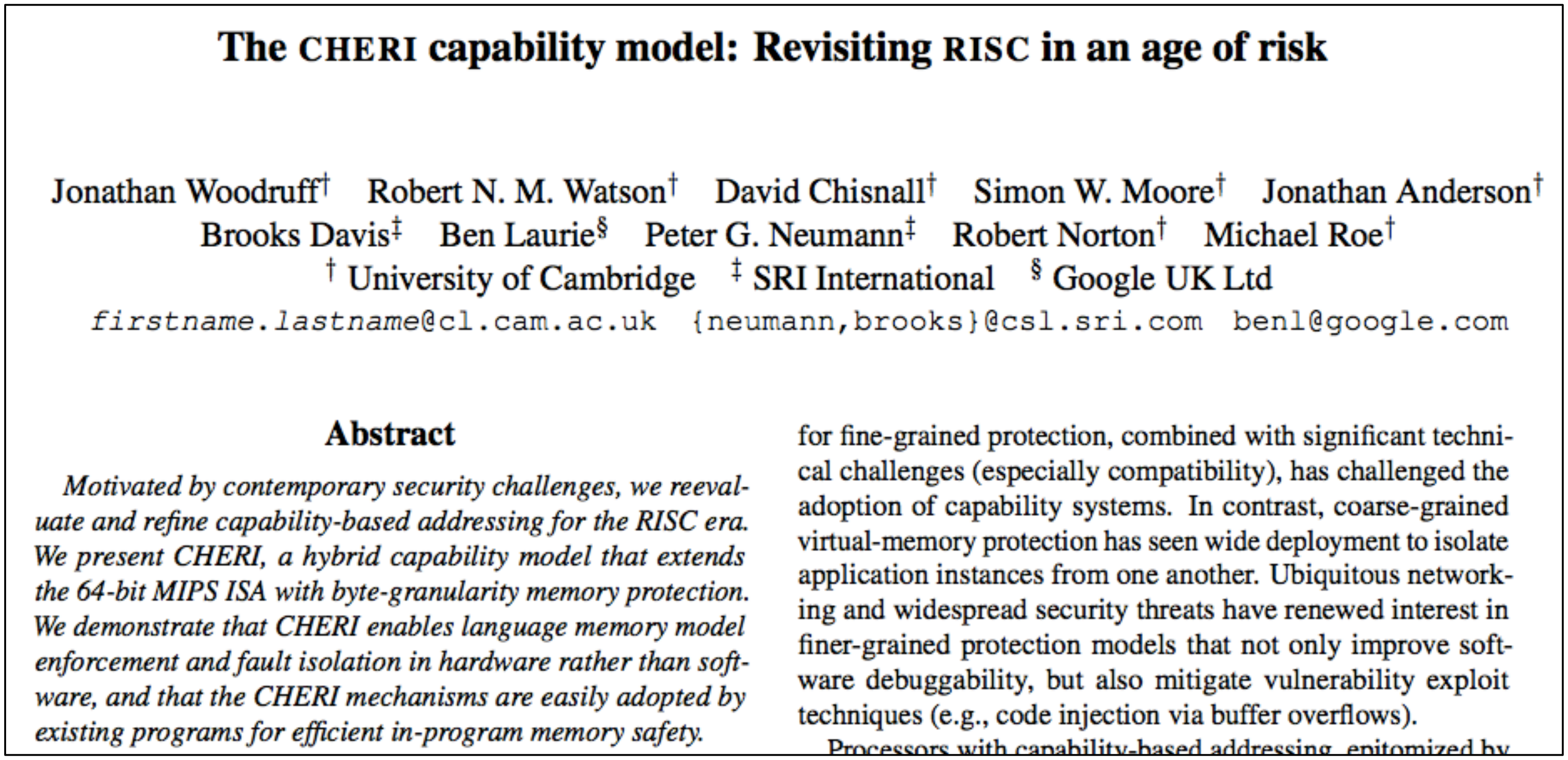
Hardware-level thread migration in a 110-core shared-memory multiprocessor

Mieszko Lis **Keun Sup Shim**
Brandon Cho **Ilia Lebedev**
Srinivas Devadas

1

Examples of who's using BSV for CPU/SoC design

Intl. Symp. on Computer Architecture (ISCA) 2014



The CHERI capability model: Revisiting RISC in an age of risk

Jonathan Woodruff[†] Robert N. M. Watson[†] David Chisnall[†] Simon W. Moore[†] Jonathan Anderson[†]
Brooks Davis[‡] Ben Laurie[§] Peter G. Neumann[‡] Robert Norton[†] Michael Roe[†]
[†] University of Cambridge [‡] SRI International [§] Google UK Ltd
firstname.lastname@cl.cam.ac.uk {neumann,brooks}@csl.sri.com benl@google.com

Abstract

Motivated by contemporary security challenges, we reevaluate and refine capability-based addressing for the RISC era. We present CHERI, a hybrid capability model that extends the 64-bit MIPS ISA with byte-granularity memory protection. We demonstrate that CHERI enables language memory model enforcement and fault isolation in hardware rather than software, and that the CHERI mechanisms are easily adopted by existing programs for efficient in-program memory safety.

for fine-grained protection, combined with significant technical challenges (especially compatibility), has challenged the adoption of capability systems. In contrast, coarse-grained virtual-memory protection has seen wide deployment to isolate application instances from one another. Ubiquitous networking and widespread security threats have renewed interest in finer-grained protection models that not only improve software debuggability, but also mitigate vulnerability exploit techniques (e.g., code injection via buffer overflows).

Processors with capability-based addressing, epitomized by

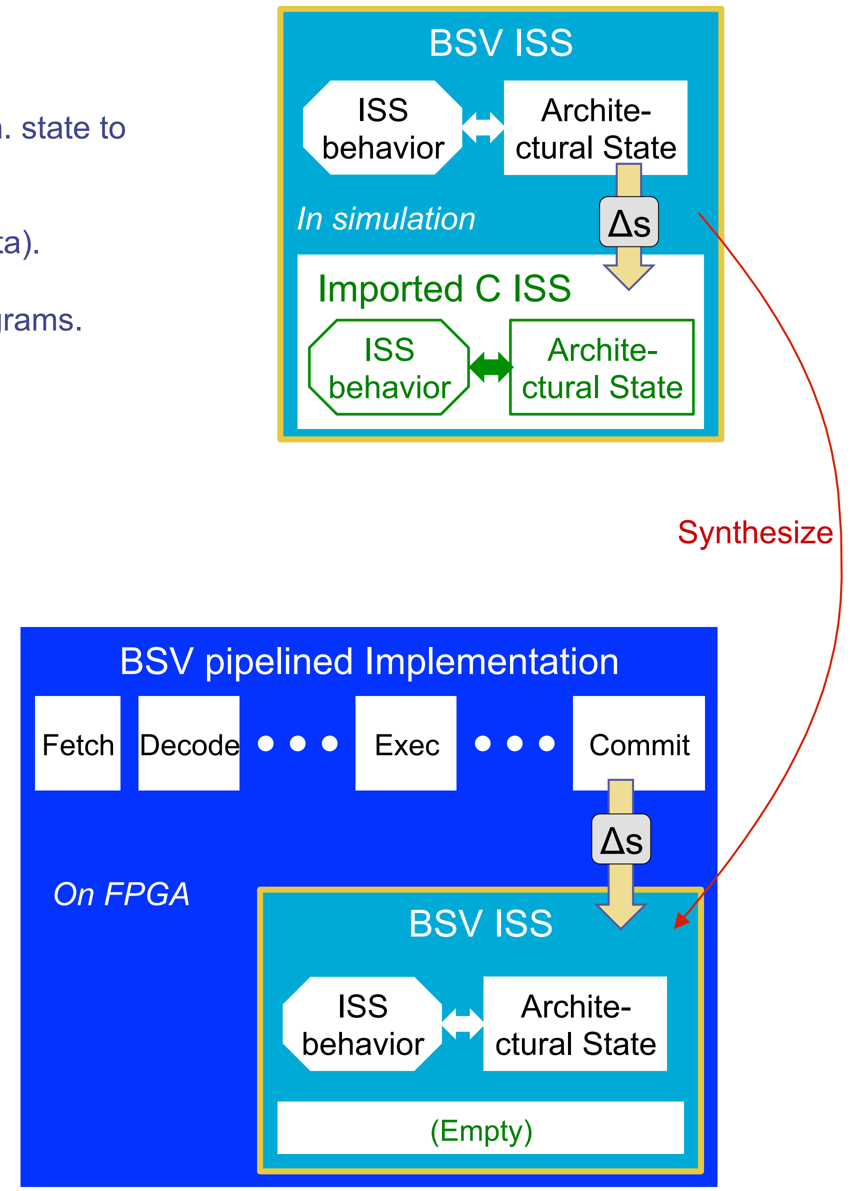
- Implementation of MIPS with extensions for fine-grain (object-granularity) protection and access controls (primarily U.Cambridge UK)
- Formal verification of security properties (primarily SRI)

Synthesizable ISS enables Tandem Verification

Bootstrap: Verify ISS in simulation

- For each instruction executed by the BSV ISS, send “deltas” of arch. state to C ISS, which also executes the instruction and verifies deltas.
- Immediately identifies divergent state!
- Cheap: per-instruction deltas are quite small (couple of words of data).
- This verifies the BSV ISS
- We do this for full OS boot and app runs, not just synthetic test programs.

- (Some tricky details about non-determinism: interrupts, cycle counts, …)



Use and re-use: BSV ISS is synthesized to FPGA, and used for tandem verification of actual CPU implementations (pipelined)

- Same game, at next level: BSV ISS verifies BSV complex pipelined implementation
- Fast: MHz speeds, everything is on FPGA
  - Many orders of magnitude faster than simulation
- Verify full OS boot, app execution, driver execution, …
- Re-use synthesizable ISS for verifying multiple pipelined implementations

RISC-V (basis for Bluespec's Customizable CPU and SoC Kit)

Bluespec offers a family of CPUs based on the **RISC-V** ISA (Instruction Set Architecture)

- 5th gen RISC ISA from Univ. of California, Berkeley
  - History: RISC-I (1981), RISC-II (1983), SOAR (1984), SPUR (1989) and various other specialized architectures
- Open ISA (no royalties), open software tools and systems (BSD license)
- Core spec: 32 bit and 64 bit integer instructions

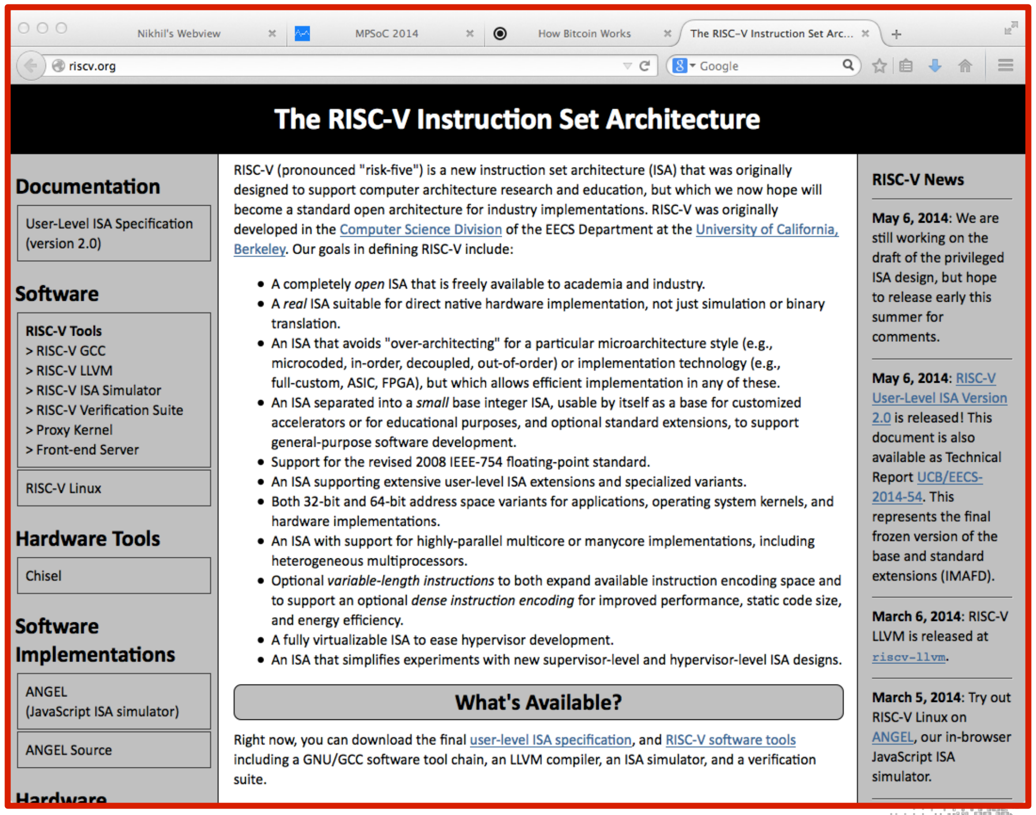
Extensible, customizable ISA

- Optional 16-bit instruction coding for small instruction footprint
- Standard extensions for integer multiply/ divide, floating point, atomic memory operations
- Standardized extension mechanism (vector, SIMD, Quad-precision floating point, bit manipulation, …)

Software (open source, BSD license, on github)

- GCC tools (gcc, objdump, gas
- GDB
- LLVM
- Linux

www.riscv.org



The RISC-V Instruction Set Architecture

Documentation

User-Level ISA Specification (version 2.0)

Software

RISC-V Tools
> RISC-V GCC
> RISC-V LLVM
> RISC-V ISA Simulator
> RISC-V Verification Suite
> Proxy Kernel
> Front-end Server

RISC-V Linux

Hardware Tools

Chisel

Software Implementations

ANGEL (JavaScript ISA simulator)

ANGEL Source

Hardware

RISC-V (pronounced "risk-five") is a new instruction set architecture (ISA) that was originally designed to support computer architecture research and education, but which we now hope will become a standard open architecture for industry implementations. RISC-V was originally developed in the Computer Science Division of the EECS Department at the University of California, Berkeley. Our goals in defining RISC-V include:

- A completely *open* ISA that is freely available to academia and industry.
- A *real* ISA suitable for direct native hardware implementation, not just simulation or binary translation.
- An ISA that avoids "over-architecting" for a particular microarchitecture style (e.g., microcoded, in-order, decoupled, out-of-order) or implementation technology (e.g., full-custom, ASIC, FPGA), but which allows efficient implementation in any of these.
- An ISA separated into a *small* base integer ISA, usable by itself as a base for customized accelerators or for educational purposes, and optional standard extensions, to support general-purpose software development.
- Support for the revised 2008 IEEE-754 floating-point standard.
- An ISA supporting extensive user-level ISA extensions and specialized variants.
- Both 32-bit and 64-bit address space variants for applications, operating system kernels, and hardware implementations.
- An ISA with support for highly-parallel multicore or manycore implementations, including heterogeneous multiprocessors.
- Optional *variable-length instructions* to both expand available instruction encoding space and to support an optional *dense instruction encoding* for improved performance, static code size, and energy efficiency.
- A fully virtualizable ISA to ease hypervisor development.
- An ISA that simplifies experiments with new supervisor-level and hypervisor-level ISA designs.

What's Available?

Right now, you can download the final user-level ISA specification, and RISC-V software tools including a GNU/GCC software tool chain, an LLVM compiler, an ISA simulator, and a verification suite.

RISC-V News

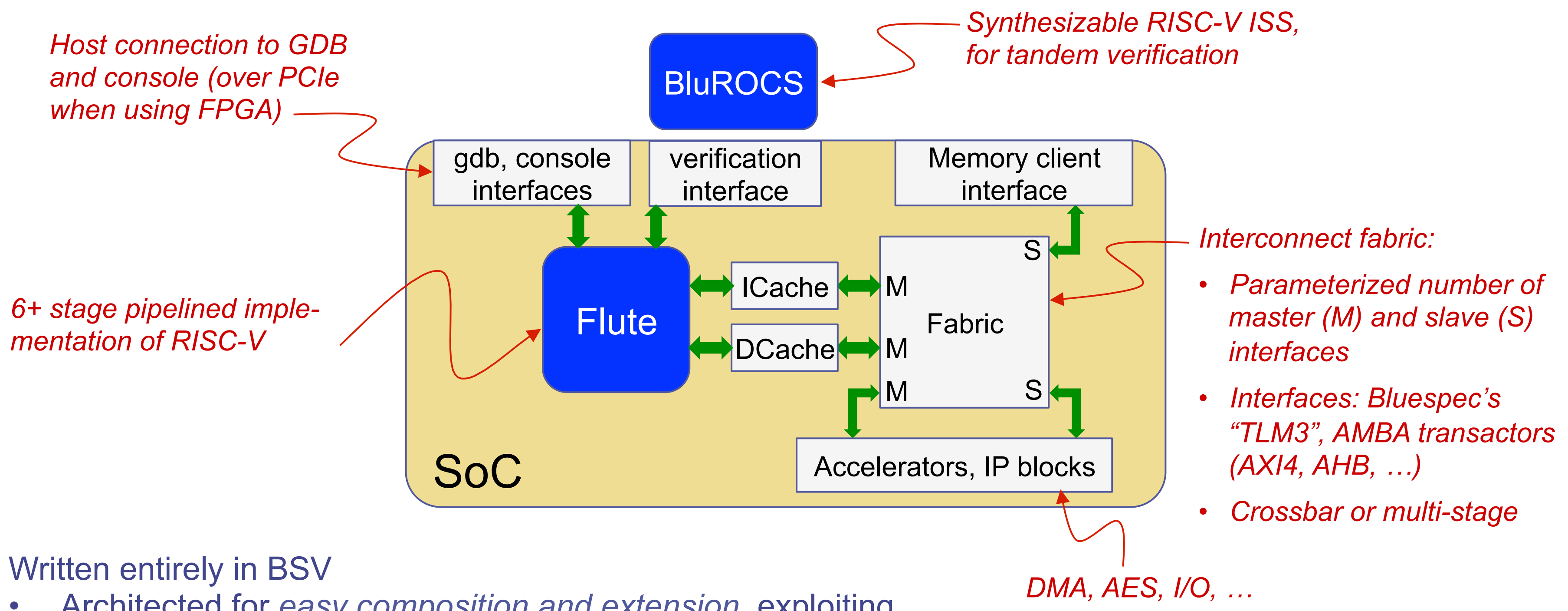
May 6, 2014: We are still working on the draft of the privileged ISA design, but hope to release early this summer for comments.

May 6, 2014: RISC-V User-Level ISA Version 2.0 is released! This document is also available as Technical Report UCB/EECS-2014-54. This represents the final frozen version of the base and standard extensions (IMAFD).

March 6, 2014: RISC-V LLVM is released at `riscv-llvm`.

March 5, 2014: Try out RISC-V Linux on ANGEL, our in-browser JavaScript ISA simulator.

Bluespec's Customizable CPU and SoC Kit


Host connection to GDB and console (over PCIe when using FPGA)
BluROCS
Synthesizable RISC-V ISS, for tandem verification
gdb, console interfaces
verification interface
Memory client interface
6+ stage pipelined imple-mentation of RISC-V
Flute
ICache
DCache
M
M
M
S
S
Fabric
Interconnect fabric:
• Parameterized number of master (M) and slave (S) interfaces
• Interfaces: Bluespec's "TLM3", AMBA transactors (AXI4, AHB, …)
• Crossbar or multi-stage
SoC
Accelerators, IP blocks
DMA, AES, I/O, …

Written entirely in BSV
- Architected for *easy composition and extension*, exploiting BSV features for this
- Open source community: alternative RISC-V implementations, fabrics, I/O blocks, IP blocks, accelerators, …

Software:
- Linux, BusyBox, …
- Tool chain: gcc and Gnu tools, LLVM, gdb, soft-float, …

Coming:
- Multi-core, with coherent caches

If you'd like more technical detail, see code examples, see a demo, etc., please catch me off-line.

Summary

- The BSV language is uniquely suited for CPU and SoC design
  - High level, natural expression of hardware structure and behavior
  - Powerful abstraction while remaining fully synthesizable
  - Synthesizable (even models, for fast FPGA execution)
  - From models to implementations

- Many organizations are using BSV for CPU/SoC design

- A RISC-V CPU and SoC kit is available from Bluespec
  - Lowers the barrier to entry in the SoC space because:
    - Open ISA, open source IP, open source software, …
    - Composable and extensible, due to unique BSV properties