DORNERWORKS

seL4 VMM on the RISC-V Rocket Chip

Robbie VanVossen

Michael Doran

Current Situation & Problems

Current Situation

- DornerWorks collaborates with multiple customers in various industries to utilize seL4 on real world systems
- RISC-V is an attractive platform for open-source solutions, such as seL4
- Virtualization is a great way to get the benefits of seL4 without needing to write a lot of components from scratch

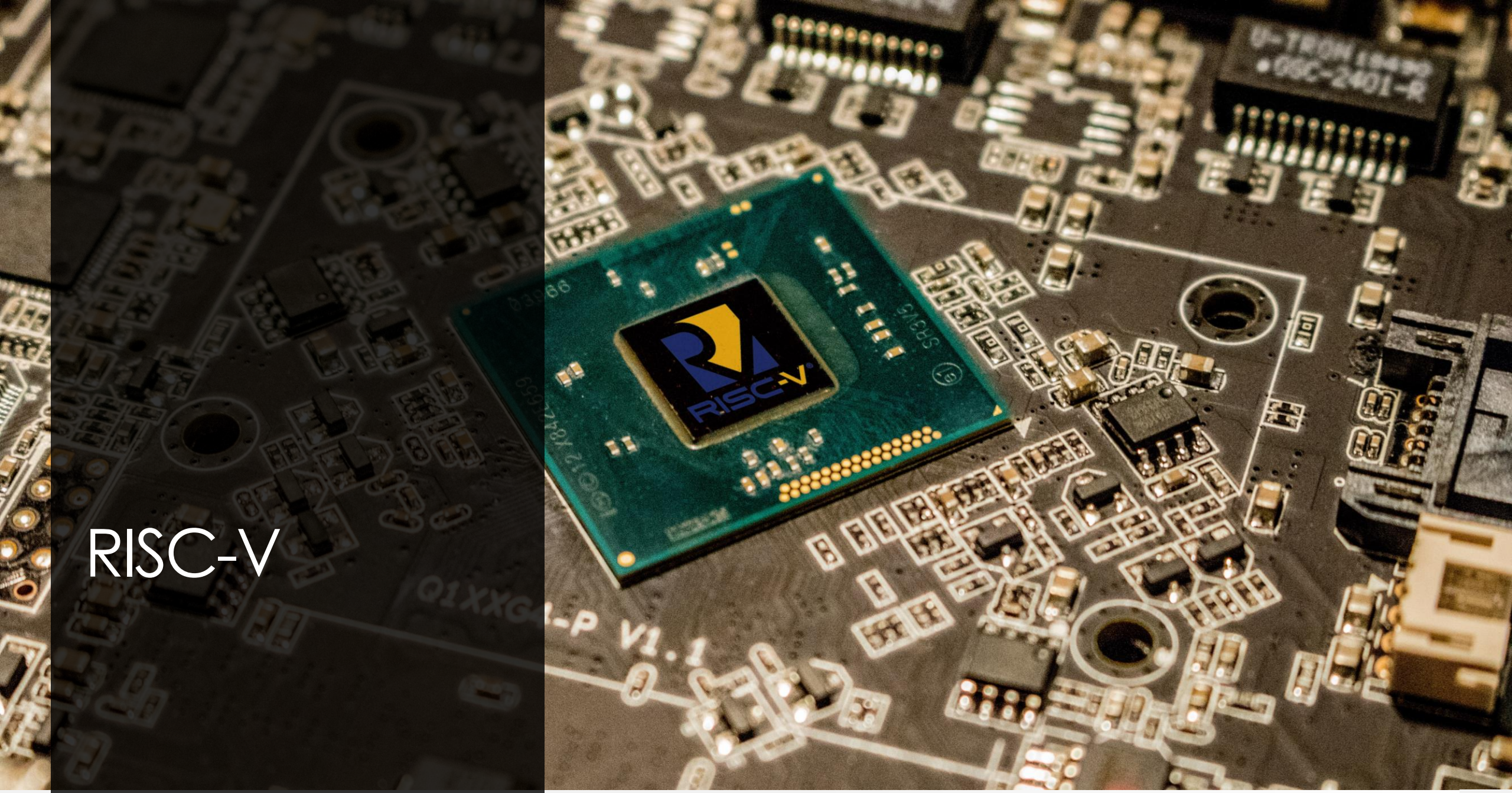
Problem 1

- RISC-V is lagging behind some other platforms for virtualization support
  - Software and hardware

Problem 2

- By not utilizing the formally-verified seL4 microkernel, RISC-V system designers are sacrificing strong isolation and security features

RISC-V

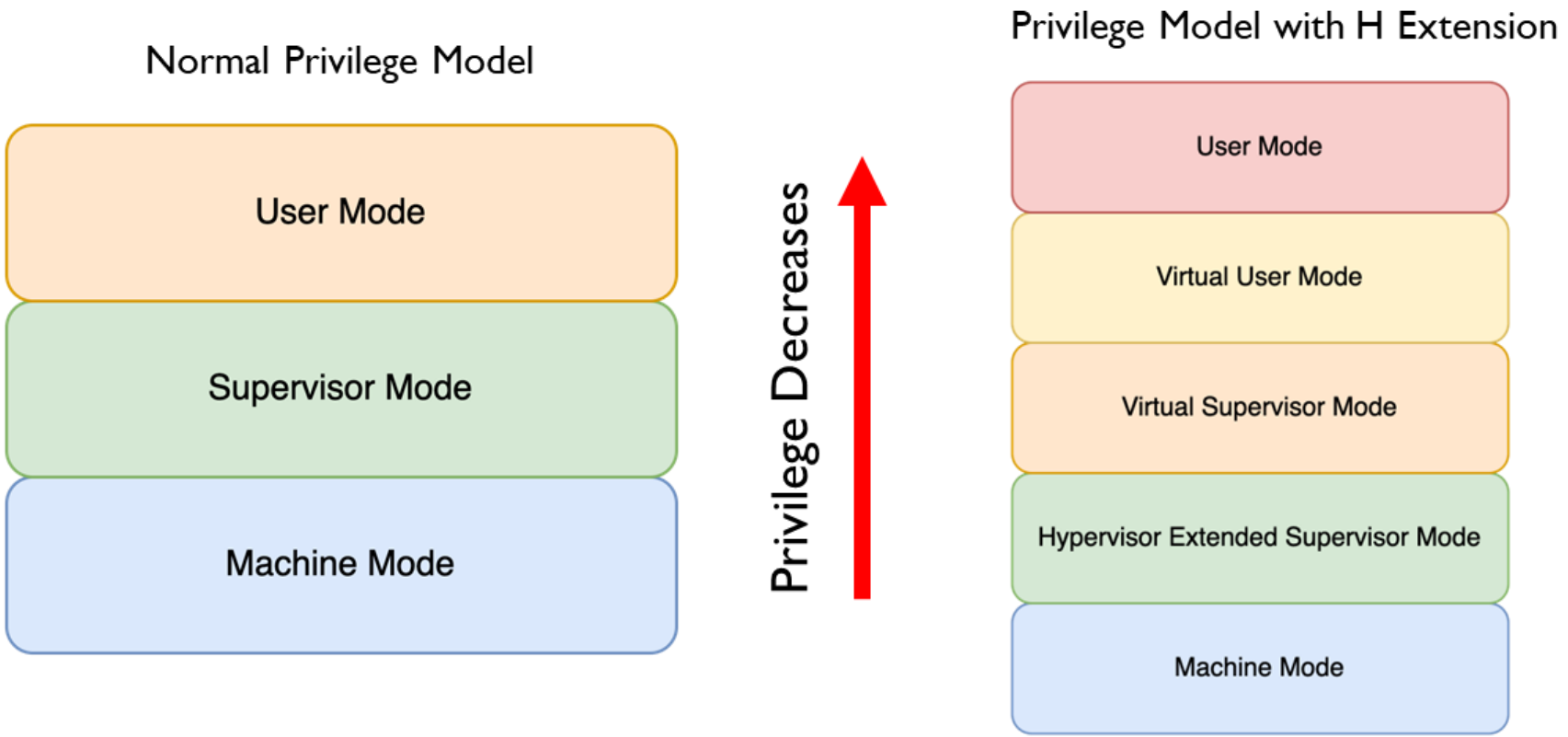


RISC-V: Background

- Open ISA based on RISC architecture
- Original developed by University of California Berkely in 2010
  - Standard is now managed by RISC-V International
- Functionality is expanded with the extension modules
  - Ex: the M module provides multiplication and division instructions
  - Custom modules can be easily created to better fit niche areas:
    - High-assurance security
    - High reliability
- Great fit for seL4
  - Open HW and SW stack

RISC-V ISA Extension	Description
M	Enables multiplication and division
A	Enables atomic instructions
F	Enables floating point instructions
D	Enables double precision floating point instructions
G	Enables modules M, A, F, D.
Q	Quad precision floating-point instructions
L	Decimal floating-point instructions
C	Compressed instructions
B	Bit manipulation instructions
J	Dynamically translated languages support
T	Transactional memory support
P	Packed-single Instruction Multiple Data
V	Vector operation instructions
N	User-level interrupt support
H	Hypervisor support
S	Supervisor level instructions

RISC-V Privilege Model



RISC-V: Hypervisor Extensions

- Hypervisor ISA ratified December 2021
- Updates privilege modes
- Adds hypervisor instructions
- Adds hypervisor and virtualized versions of registers
- Adds two-stage address translation
- Updates traps

Virtualization Mode (V)	Nominal Privilege	Abbreviation	Name	Two-Stage Translation
0	U	U-mode	User mode	Off
0	S	HS-mode	Hypervisor-extended supervisor mode	Off
0	M	M-mode	Machine mode	Off
1	U	VU-mode	Virtual user mode	On
1	S	VS-mode	Virtual supervisor mode	On

Table 8.1: Privilege modes with the hypervisor extension.

RISC-V: Previous Work on seL4 VMM

- We pitched this to our customer 1.5 years before the contract started
  - At the time no work had started in this area
  - We planned to do everything from scratch

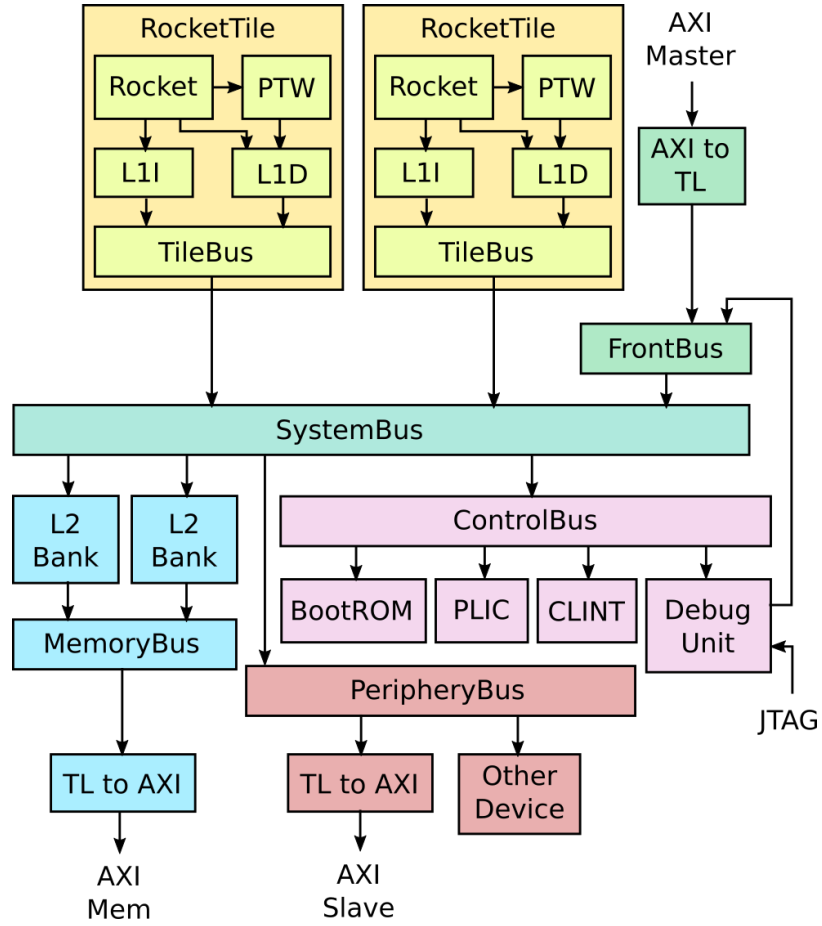
- Luckily, we found out that a lot of great work on QEMU was already completed by Yanyan Shen
  - Initial seL4 VMM on QEMU was working!
  - However, that was roughly what we had planned to do, so we needed to identify a way we could still provide value to our customer and the community
  - We ended up on porting the QEMU implementation to a soft-core RISC-V implementation with the Hypervisor extension

Rocket Chip

A RISC-V soft-core implementation

Rocket Chip: Background

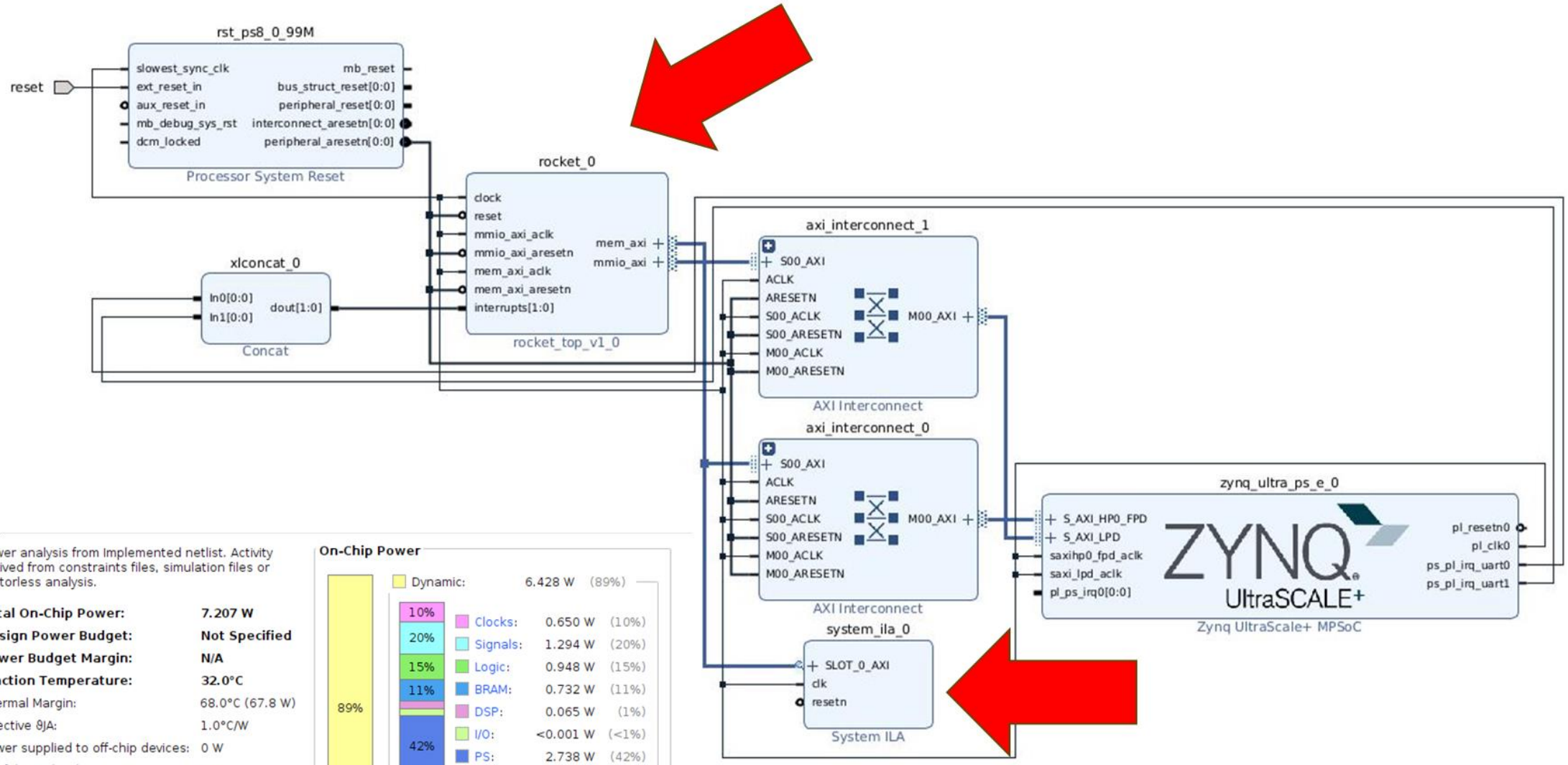
- Created by the EECS Department, University of California, Berkeley in 2016
- Chipyard -> Rocket Chip → Open-source Sysem-on-Chip design generator that emits synthesizable RTL
  - Leverages the Chisel hardware construction language to compose a library of generators for cores, caches, and interconnects into an integrated SoC
  - Generates general-purpose processor cores that use the open RISC-V ISA, and provides both an in-order core generator (Rocket) and an out-of-order core generator (BOOM)
- Other RISC-V generators and soft-cores are available
  - We have experience with Rocket Chip already
    - seL4test run on ZCU102 port of Rocket Chip w/o hypervisor extensions
  - Saw that someone had implemented the H-Extension on it



Rocket Chip: Our Implementation

- Found a fork that supported H-extensions and worked on the FPGA on a Xilinx ZCU104
  - Thanks to the Bao Hypervisor team!
  - Ported that to work on the FPGA on a Xilinx ZCU102
- Generated Rocket Chip RTL and imported that into the Xilinx Vivado tool
- Hooked up the Rocket Chip SoC to vital pieces of the system
  - Resets, Clocks and RAM
  - UART
    - IRQs
    - MMIO
- Built the bitstream and booted on the board
  - Was able to test with a native Linux and seL4Test

Rocket Chip: Programmable Logic



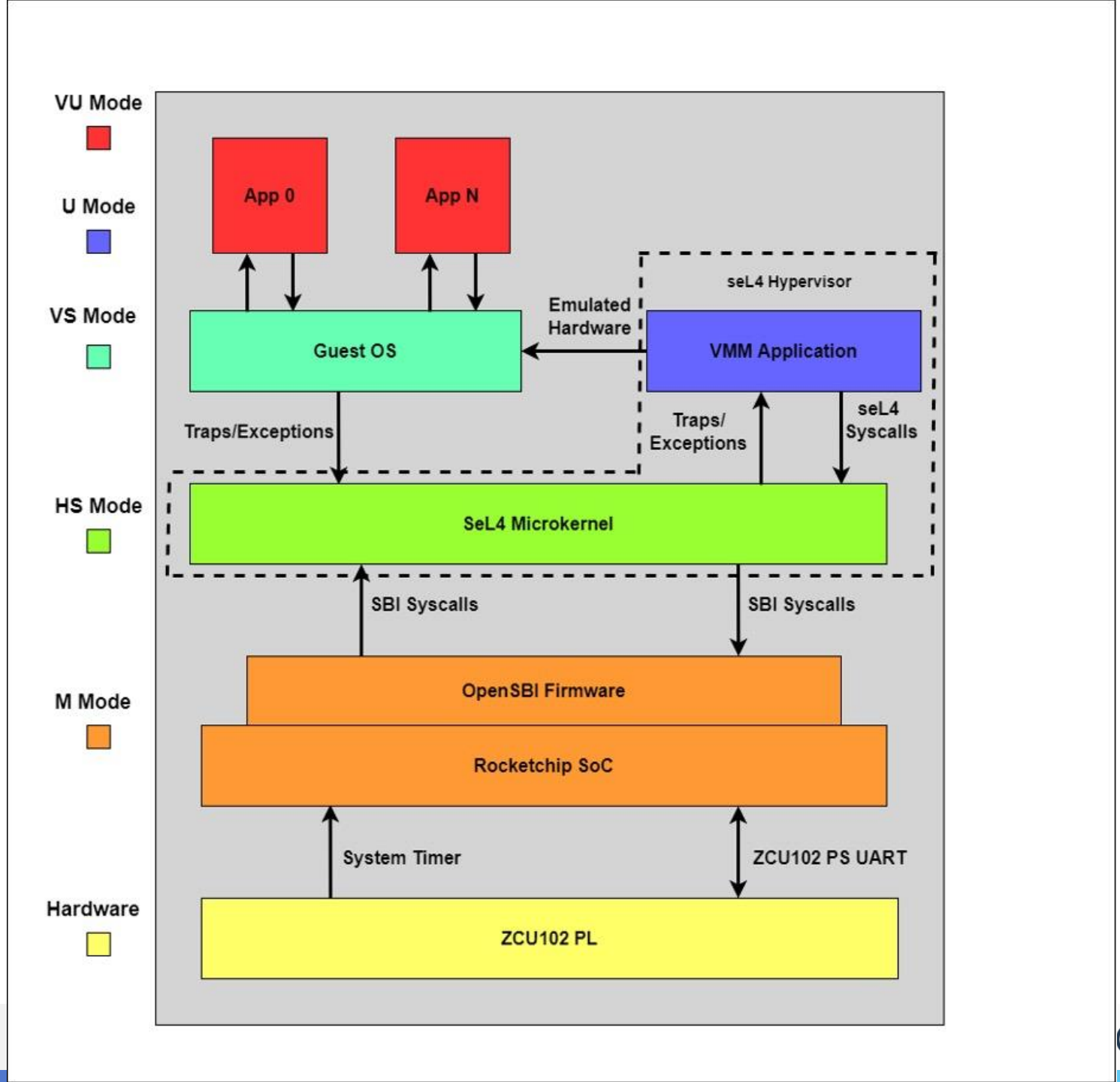
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**7.207 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**32.0°C**
Thermal Margin:	68.0°C (67.8 W)
Effective ϑJA:	1.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

Development

seL4 VMM on the Rocket Chip instantiation


VU Mode
U Mode
VS Mode
HS Mode
M Mode
Hardware
App 0
App N
seL4 Hypervisor
Emulated Hardware
Guest OS
VMM Application
Traps/Exceptions
Traps/ Exceptions
seL4 Syscalls
SeL4 Microkernel
SBI Syscalls
SBI Syscalls
OpenSBI Firmware
Rocketchip SoC
System Timer
ZCU102 PS UART
ZCU102 PL

Development: Major Issues/Changes on the Rocket Chip

- Compressed (C) mode (Instructions can be 32-bit or 16-bit)
  - The VMM's PLIC emulation worked for QEMU since C mode isn't enabled so all instructions could be assumed to be 32-bit
  - Updated the emulation code to check the size of each store/load instruction to emulate it accordingly
  - Updated program counter advancing logic to account for the correct size of the instruction
    - This was always assumed to be 32-bits as well, so a 16-bit instruction would jump over the next 16-bit instruction or start in the middle of the next 32-bit instruction

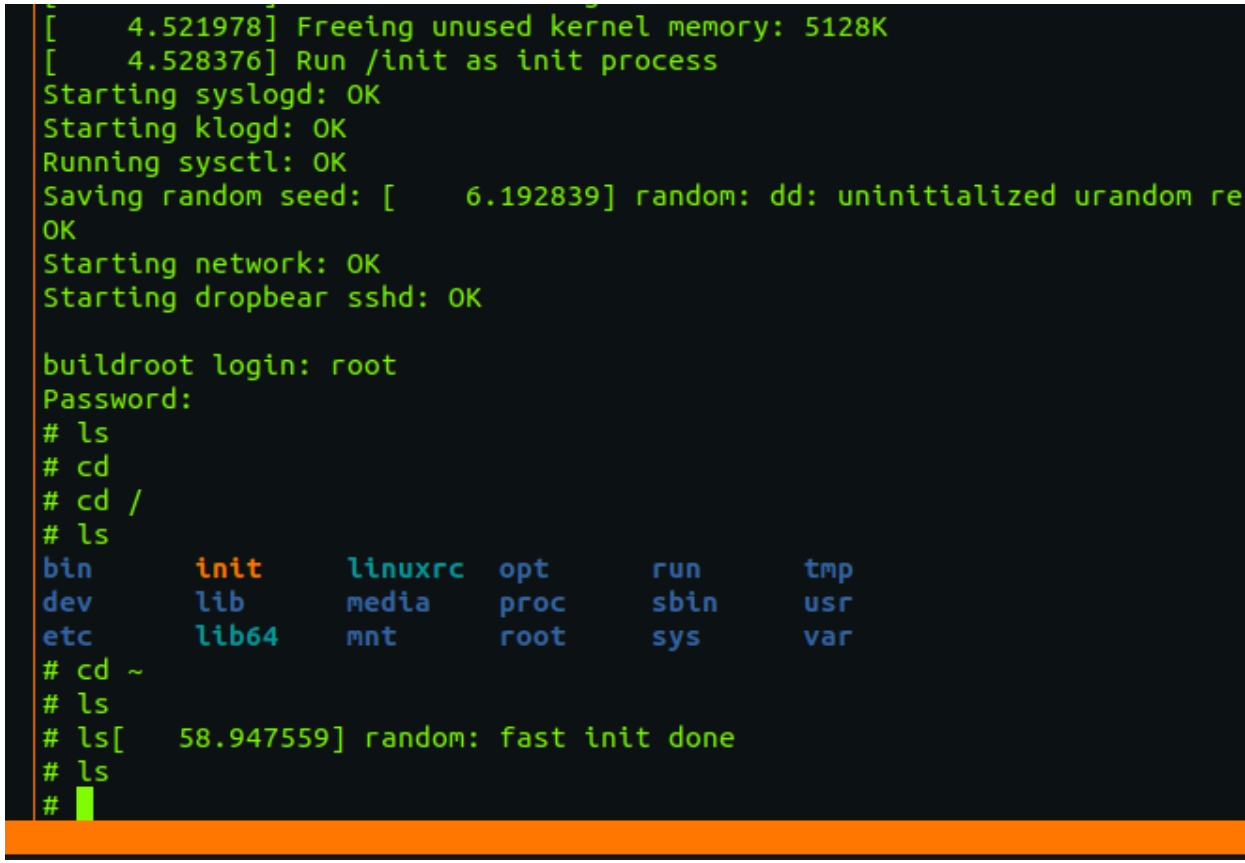
- QEMU PLIC Errata
  - Some PLIC handling code in the microkernel was written to address a bug in QEMU, therefore it didn't match up with the PLIC specification
  - That bug was not present on the Rocket Chip, so correct handling was added for the PLIC and ifdefs were used to make sure it still functioned for both platforms

Development: Minor Issues/Changes on the Rocket Chip

- New DTS specific to the Rocket-chip on ZCU102
  - Needs to be updated anytime you update devices for the soft-core
- Update memory configuration to support larger guest image
  - Our kernel and ramdisk ended up quite a bit bigger than the QEMU image, so increased Allocator Pool
- Pass-through a UART device instead of emulating OpenSBI serial API
- Implement SBI_REMOTE_FENCE_I to support OpenSBI spec
  - Expected by newer Linux guest

Development: Results

- Single Linux VM booting on the Rocket Chip with UART passed-through
- Mainlined and/or Open Sourced this work
- Added the Rocket Chip on ZCU102 with H-Extensions as a supported hardware platform for seL4:
  - https://docs.sel4.systems/Hardware/rocketchip-zcu102.html
  - Documentation included to replicate our results



Future Work: Match features of other architectures

- Support more advanced configurations:
  - Multiple VMs
  - Serial Server
  - Multicore
  - Pass-through of more complicated devices (Ethernet, USB, etc.)
- Run on real hardware
  - A real processor with the H-extension will be available at some point (hopefully soon)
- Use standardized VMM implementation
  - Lots of code ripped out from various mainline repos to speed up development
    - Relocate architecture specific VMM code into corresponding libraries
  - Port to a mainline VMM implementation:
    - CAmkES_VM
    - Microkit VMM


Questions?

References

- https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf
- https://chipyard.readthedocs.io/en/stable/Generators/Rocket-Chip.html
- https://dornerworks.com/blog/sel4-on-risc-v-rocket-chip/
- https://docs.sel4.systems/Hardware/rocketchip-zcu102.html
- https://riscv.org/technical/specifications/