The Hype around the RISC-V Hypervisor

Alistair Francis <alistair.francis@wdc.com>
Anup Patel <anup.patel@wdc.com>
KVM Forum 2019
Overview

- RISC-V H-Extension (Alistair)
- RISC-V H-Extension in QEMU (Alistair)
- KVM RISC-V (Anup)
- KVM RISC-V Status & Future Work (Anup)
- KVM RISC-V Demo (Anup)
- Questions
RISC-V H-Extension

The RISC-V Hypervisor Extension
RISC-V H-Extension: Spec Status

H-Extension spec close to freeze state

• Designed to suit both Type-1 (Baremetal) and Type-2 (Hosted) hypervisor

• v0.4-draft was released on 16th June 2019
  – This includes feedback from Open Source virtualisation projects
  – Additions have happened to the spec since:
    • htimedelta/htimedeltah CSR (Proposed by WDC – Merged)
    • Dedicated exception causes for Guest page table faults (Proposed by John Hauser – Merged)
    • htinst & htval2 CSRs for better MMIO emulation (Proposed by WDC and extended by John Hauser – Merged)
    • Separate HIE & HIP CSR for virtual interrupt injection (Proposed by WDC and extended by John Hauser – Merged)

• v0.5-draft released on 30th October 2019 (Today)

• Western Digital’s initial QEMU, Xvisor and KVM ports were based on v0.3

• They have all been updated to the new v0.4 spec
  – There were limited software changes required between v0.3 and v0.4
    • QEMU required more changes
RISC-V H-Extension: Privilege Mode Changes

New execution modes for guest execution

- HS-mode = S-mode with hypervisor capabilities and new CSRs
- Two additional modes:
  - VS-mode = Virtualized S-mode
  - VU-mode = Virtualized U-mode

![Diagram of execution modes]

- M-mode Software
- HS-mode Software
- VS-mode Software
- VU-mode Software
- U-mode Software

Decreasing Privilege Level

Virtualized World
- Guest User Space
- Guest Linux
- Hypervisor
- Firmware (OpenSBI)

Non-virtualized World
- Host User Space
- M
- HS
RISC-V H-Extension: CSR changes

More control registers for virtualising S-mode

- In HS-mode (V=0)
  - “s<xyz>” CSRs point to standard “s<xyz>” CSRs
  - “h<xyz>” CSRs for hypervisor capabilities
  - “vs<xyz>” CSRs contains VS-mode state

- In VS-mode (V=1)
  - “s<xyz>” CSRs point to virtual “vs<xyz>” CSRs

<table>
<thead>
<tr>
<th>HS-mode CSRs for hypervisor capabilities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hstatus</td>
<td>Hypervisor Status</td>
</tr>
<tr>
<td>hideleg</td>
<td>Hypervisor Interrupt Delegate</td>
</tr>
<tr>
<td>hedeleg</td>
<td>Hypervisor Trap/Exception Delegate</td>
</tr>
<tr>
<td>htimedelta</td>
<td>Hypervisor Guest Time Delta</td>
</tr>
<tr>
<td>hgatp</td>
<td>Hypervisor Guest Address Translation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HS-mode CSRs for accessing Guest/VM state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsstatus</td>
<td>Guest/VM Status</td>
</tr>
<tr>
<td>vsie</td>
<td>Guest/VM Interrupt Enable</td>
</tr>
<tr>
<td>vsip</td>
<td>Guest/VM Interrupt Pending</td>
</tr>
<tr>
<td>vstvec</td>
<td>Guest/VM Trap Handler Base</td>
</tr>
<tr>
<td>vsepc</td>
<td>Guest/VM Trap Program Counter</td>
</tr>
<tr>
<td>vscause</td>
<td>Guest/VM Trap Cause</td>
</tr>
<tr>
<td>vstval</td>
<td>Guest/VM Trap Value</td>
</tr>
<tr>
<td>vsatp</td>
<td>Guest/VM Address Translation</td>
</tr>
<tr>
<td>vsscratch</td>
<td>Guest/VM Scratch</td>
</tr>
</tbody>
</table>
RISC-V H-Extension: Two-stage MMU

Hardware optimized guest memory management

• Two-Stage MMU for VS/VU-mode:
  – VS-mode page table (Stage1):
    • Translates Guest Virtual Address (GVA) to Guest Physical Address (GPA)
    • Programmed by Guest (same as before)
  – HS-mode guest page table (Stage2):
    • Translates Guest Physical Address (GPA) to Host Physical Address (HPA)
    • Programmed by Hypervisor

• In HS-mode, software can program two page tables:
  – HS-mode page table: Translate hypervisor Virtual Address (VA) to Host Physical Address (HPA)
  – HS-mode guest page table: Translate Guest Physical Address (GPA) to Host Physical Address (HPA)

• Format of VS-mode page table, HS-mode guest page table and HS-mode host page table is same (Sv32, Sv39, Sv48, ....)
RISC-V H-Extension: I/O & Interrupts

I/O and guest interrupts virtualization

• Virtual interrupts injected by updating VSIP CSR from HS-mode

• Software and Timer Interrupts:
  – Hypervisor will emulate SBI calls for Guest

• HS-mode guest page table can be used to trap-n-emulate MMIO accesses for:
  – Software emulated PLIC
  – VirtIO devices
  – Other software emulated peripherals
# RISC-V H-Extension: Compare ARM64

How is RISC-V H-Extension compared to ARM64 virtualization?

<table>
<thead>
<tr>
<th>RISC-V H-Extension v0.4 draft</th>
<th>ARM64 (ARMv8.x) Virtualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No separate privilege mode for hypervisors.</strong> Extends S-mode with hypervisor capabilities (HS-mode) and Guest/VM run in virtualized S-mode/U-mode (VS-mode or VU-mode).</td>
<td><strong>Separate EL2 exception-level for hypervisors with its own &lt;xyz&gt;_EL2 MSRs. The Guest/VM will run in EL1/EL0 exception levels.</strong></td>
</tr>
</tbody>
</table>

| Well suited for both Type-1 (baremetal) and Type-2 (hosted) hypervisors. The S<xyz> CSRs access from VS-mode map to special VS<xyz> CSRs which are only accessible to HS-mode and M-mode. | **Special ARMv8.1-VHE Virtualization Host Extension for better performance of Type-2 (hosted) hypervisor.** Allows Host kernel (meant for EL1) to run in EL2 by mapping <xyz>_EL1 MSRs to <abc>_EL2 MSRs in Host mode. |
| Virtual interrupts for Guest/VM injected using VSIP CSR. The hypervisor does not require any special save/restore but it will emulate entire PLIC in software. | Virtual interrupts for Guest/VM injected using LR registers of GICv2/GICv3 with virtualization extension. The hypervisor will save/restore LR registers and emulate all GIC registers in software except GIC CPU registers. |

© 2019 Western Digital Corporation or its affiliates. All rights reserved.
## RISC-V H-Extension: Compare ARM64 (Contd.)

### How is RISC-V H-Extension compared to ARM64 virtualization?

<table>
<thead>
<tr>
<th>RISC-V H-Extension v0.4 draft</th>
<th>ARM64 (ARMv8.x) Virtualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Virtual timer events for Guest/VM using SBI calls emulated by hypervisor.</strong> The SBI calls trap to hypervisor so save/restore of virtual timer state not required.</td>
<td><strong>Virtual timer events for Guest/VM using ARM generic timers with virtualization support.</strong> The hypervisor will save/restore virtual timer state and manage virtual timer interrupts.</td>
</tr>
<tr>
<td><strong>Virtual inter-processor interrupts for Guest/VM using SBI calls emulated by hypervisor.</strong> The hypervisor does not require any special save/restore.</td>
<td><strong>Virtual inter-processor interrupts for Guest/VM by emulating ICC_SGI1R_EL1 (virtual GICv3) or GICD_SGIR (virtual GICv2).</strong> The save/restore will be handled as part of LR registers save/restore.</td>
</tr>
<tr>
<td><strong>Nested virtualization supported using HSTATUS.VTVM and HSTATUS.VTSR bits.</strong> The hypervisor will trap-n-emulate Guest hypervisor capabilities.</td>
<td><strong>Special ARMv8.3-NV for supporting nested virtualization on ARMv8.</strong> The hypervisor will trap-n-emulate Guest hypervisor capabilities. The ARMv8.4-NV further enhances nested virtualization support.</td>
</tr>
</tbody>
</table>
RISC-V H-Extension in QEMU

Emulating RISC-V Hypervisor Extension in QEMU
Current QEMU Implementation

• Patches on list to add support for v0.4 Virtualisation extension
  – Both for 32-bit and 64-bit
  – Includes all vs CSRs and support for swapping CSRs
  – Interrupts are correctly generated to the Hypervisor, which can then inject them to it’s guests
  – Floating point is correctly disabled by the Hypervisor
  – Two stage MMU is implemented and fully supported

• The Hypervisor extension is disabled by default
  – It can be enabled with: -cpu rv64, x-h=true
  – The patches can be found here until they are fully upstream: https://github.com/kvm-riscv/qemu
Changes made to QEMU in preparation

• Remove requirement on MIP CSR (pending interrupts) being updated atomically
  – Having MIP updated atomically posed a headache for swapping the VSIP and SIP CSRs

• Allow setting ISA extensions via command line
  – We need to have Hypervisor extensions disabled by default, and allow users to enable via command line
  – QEMU can now enable/disable extensions via command line

• Consolidate floating point enable/disable logic
Maintaining the Hypervisor State

• The Hypervisor state only changes on traps and returns
  – This makes it straight forward to keep track of

• M Mode and HS Mode can pretend to be Virtualised
  – This is used to access memory through the 2-stage MMU (to decode fault addresses for example)
  – QEMU needs to know when to do this

• Certain faults can not be delegated to the guests
  – QEMU needs to know if one of these happen
  – This is maintained as part of the virtualisation state (FORCE_HS_EXcep)
Two Stage MMU

- Two stages are always enabled when virtualisation is on

- Two stages can be turned on even when virtualisation is off
  - MSTATUS_MPRV in M mode and HSTATUS_SPRV and HSTATUS_SPV in HS mode
  - This doesn’t apply to instruction fetches, only loads/stores
  - This requires the translation to use vsatp instead of satp (guests page table)

- Second level translation failures must raise an exception with the Hypervisor
  - They can not be delegated

```c
/*
 * @env: CPURISCVState
 * @physical: This will be set to the calculated physical address
 * @prot: The returned protection attributes
 * @addr: The virtual address to be translated
 * @access_type: The type of MMU access
 * @mmu_idx: Indicates current privilege level
 * @first_stage: Are we in first stage translation?
 * @second_stage: Are we going to perform two stage translation
 */
static int get_physical_address(CPURISCVState *env, hwaddr *physical,
  int *prot, target_ulong *addr,
  int access_type, int mmu_idx,
  bool first_stage, bool two_stage)

bool riscv_cpu_tlb_fill(CPURISCVState *cs, vaddr address, int size,
...{
  if (riscv_cpu_virt_enabled(env) || m_mode_two_stage || hs_mode_two_stage) {
    /* Two stage lookup */
    ret = get_physical_address(env, &pa, &prot, address, access_type,
      mmu_idx, true, true);
    ...
  }
  if (ret == TRANSLATE_FAIL) {
    goto tlb_lookup_done;
  }
  /* Second stage lookup */
  im_address = pa;

  ret = get_physical_address(env, &pa, &prot, im_address, access_type, mmu_idx,
     false, true);
  ...
  if (ret != TRANSLATE_SUCCESS) {
    /* Guest physical address translation failed, this is a HS
     * level exception
     * /
    first_stage_error = false;
    address = im_address | (address & (TARGET_PAGE_SIZE - 1));
    goto tlb_lookup_done;
  }
  ...
} else {
...```
Handling Register Swapping in QEMU

- Using pointers to handle M-Mode CSRs that are exposed as S-Mode (mstatus, mie)
- Value swapping the S-Mode only CSRs
- mip CSR (no longer atomically accessed) is value swapped as well

![Diagram showing the process of handling register swapping in QEMU](image)
Future Work

- Upstream the current work
- Implement RISC-V H-Extension v0.5 draft
- Update QEMU's TLB caching index's to include Virtualisation state
  - Then we can support fine grain TLB flushing from sfence and hfence instructions
    - Allow sfence to only flush current virtualization TLBs
    - Allow hfence to flush only guest TLBs
  - Currently we flush everything on state changes which is slow and incorrect
- Update to the latest version of the spec as it is released
- Add support for nested virtualization
- Get 32-bit Linux guests running
KVM RISC-V

The RISC-V port of the KVM hypervisor
KVM RISC-V

World’s first Type-2 RISC-V hypervisor

- RISC-V H-extension is very well suited for KVM Hypervisor
- Host Linux runs unmodified in HS-mode
- H-extension CSRs only accessed by KVM RISC-V in Host Linux
- Guest Linux runs unmodified in VS-mode

Diagram:

- Virtualized World:
  - Guest User Space
  - Guest Linux
  - Linux KVM (Host Linux)

- Non-virtualized World:
  - QEMU/KVMTOOL
  - Firmware (OpenSBI)

Legend:
- M-mode Software
- HS-mode Software
- VS-mode Software
- VU-mode Software
- U-mode Software

Decreasing Privilege Level
KVM RISC-V: Key Aspects

What have we achieved so far?

- No RISC-V specific KVM IOCTL
- Minimal possible world-switch
- Full save-restore via vcpu_load()/vcpu_put()
- FP lazy save/restore
- KVM ONE_REG interface for user-space
- Timer and IPI emulation in kernel-space
- PLIC emulation is done in user-space
- Hugepage support
- SBI v0.1 interface for Guest
- Unhandled SBI calls forwarded to KVM userspace
KVM RISC-V: SBI Interface

Syscall style interface between Host and Guest

• SBI = Supervisor Binary interface
• SBI v0.1 in-use by Linux kernel
  (Refer, https://github.com/riscv/riscv-sbi-doc/blob/v0.1.0/riscv-sbi.md)
• SBI v0.2 in draft stage

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>sbi_set_timer</td>
<td>0</td>
</tr>
<tr>
<td>IPI</td>
<td>sbi_clear_i</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>sbi_send_i</td>
<td>4</td>
</tr>
<tr>
<td>Memory Model</td>
<td>sbi_remote_fence_i</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>sbi_remote sfence_vma</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>sbi_remote sfence vma asid</td>
<td>7</td>
</tr>
<tr>
<td>Console</td>
<td>sbi_console_putchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>sbi_console_getchar</td>
<td>2</td>
</tr>
<tr>
<td>Shutdown</td>
<td>sbi_shutdown</td>
<td>8</td>
</tr>
</tbody>
</table>
KVM RISC-V: RUN LOOP
The runtime loop for KVM RISC-V VCPUs

QEMU/KVMTOOL

Userspace RUN LOOP

ioctl(KVM_RUN)

Exit Handler

• MMIO Exit
• SBI Call Exit
• .... Others ....

VCPU Enter

VCPU Exit

KVM RISC-V

kvm_arch_vcpu_ioctl_run()
In-Kernel RUN LOOP

kvm_arch_vcpu_load()

__kvm_riscv_switch_to()

kvm_riscv_vcpu_exit()

kvm_arch_vcpu_put()

Host-to-Guest

Guest User Space

Guest Kernel

• Host Interrupt
• MMIO Trap
• SBI Call
• Stage2 Trap
• .... Others ....

Host-to-Host

• MMIO Exit
• SBI Call Exit
• .... Others ....
KVM RISC-V: RUN IOCTL

The In-Kernel RUN LOOP

```c
int kvm_arch_vcpu_ioctl_run(...)
{
    int ret = 1;
    ....
    /* Handle MMIO returned from userspace */
    ....
    /* Handle SBI returned from userspace */
    ....
    while (ret > 0) {
        ....
        kvm_riscv_vcpu_flush_interrupts(...);
        ....
        kvm_riscv_vcpu_sync_interrupts(...);
        ....
        ret = kvm_riscv_vcpu_exit(...);
    }
    ....
    return ret;
}
```

Update VCPU state for MMIO returned from userspace

Update VCPU state for SBI returned from userspace

Update VCPU VSIP CSR for pending VCPU interrupts

KVM RISC-V world switch

Sync-up VSIP CSR changes done by VCPU

Process VCPU traps
**KVM RISC-V: VCPU Interrupts**

Multiple producer and single consumer of VCPU interrupts

**In-Kernel RUN LOOP (Consumer)**

- `kvm_riscv_vcpu_flush_interrupts()` Update Guest VCPU VSIP CSR based on `irqs_pending` and `irqs_pending_mask`
- `__kvm_riscv_switch_to()`
- `kvm_riscv_vcpu_sync_interrupts()` Sync `irqs_pending` and `irqs_pending_mask`
  Guest VCPU updates to VSIP CSR

**VCPU Interrupt State**

- `irqs_pending` Atomic bitmap representing current state of VCPU interrupts
- `irqs_pending_mask` Atomic bitmap representing bits changed in `irqs_pending`

**KVM_INTERRUPT IOCTL (Producer)**

- VCPU Timer Expiry (Producer)
- IPI from other VCPUs (Producer)
KVM RISC-V: World Switch

The KVM world switch between Host and Guest

ENTRY(__kvm_riscv_switch_to)
    /* Save Host GPRs (except A0 and T0-T6) */....
    /* Save Host SSTATUS, HSTATUS, SSCRATCH and STVEC */....
    /* Change Host exception vector to return path */....
    /* Restore Guest HSTATUS, SSTATUS and SEPC */....
    /* Restore Guest GPRs (except A0) */....
    /* Save Host A0 in SSCRATCH */....
    /* Resume Guest */
    sret

__kvm_switch_return:
    /* Swap Guest A0 with SSCRATCH */....
    /* Save Guest GPRs (except A0) */....
    /* Save Guest A0 */....
    /* Save Guest HSTATUS, SSTATUS, and SEPC */....
    /* Restore Host SSTATUS, HSTATUS, SSCRATCH and STVEC */....
    /* Restore Host GPRs (except A0 and T0-T6) */....
    /* Return to C code */
    ret
ENDPROC(__kvm_riscv_switch_to)

Host-to-Guest (HS-mode)
- Save 24 GPRs
- Save 4 CSRs
- Restore 3 CSRs
- Restore 31 GPRs

In-Guest (VS-mode)
- HS-mode STVEC = __kvm_switch_return
- HS-mode SSCRATCH = VCPU context

Guest-to-Host (HS-mode)
- Save 31 GPRs
- Save 3 CSRs
- Restore 4 CSRs
- Restore 24 GPRs
## KVM RISC-V: VCPU Context

What things are saved/restored for a VCPU?

### `struct kvm_cpu_context`:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Save/Restore</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>Zero register</td>
<td>---</td>
</tr>
<tr>
<td>ra</td>
<td>Return address register</td>
<td>World switch</td>
</tr>
<tr>
<td>sp</td>
<td>Stack pointer register</td>
<td>World switch</td>
</tr>
<tr>
<td>gp</td>
<td>Global pointer register</td>
<td>World switch</td>
</tr>
<tr>
<td>tp</td>
<td>Thread pointer register</td>
<td>World switch</td>
</tr>
<tr>
<td>a0-a7</td>
<td>Function argument registers</td>
<td>World switch</td>
</tr>
<tr>
<td>t0-t6</td>
<td>Caller saved registers</td>
<td>World switch</td>
</tr>
<tr>
<td>s0-s11</td>
<td>Callee saved registers</td>
<td>World switch</td>
</tr>
<tr>
<td>sepc</td>
<td>Program counter</td>
<td>World switch</td>
</tr>
<tr>
<td>sstatus</td>
<td>Shadow SSTATUS CSR</td>
<td>World switch</td>
</tr>
<tr>
<td>hstatus</td>
<td>Shadow HSTATUS CSR</td>
<td>World switch</td>
</tr>
<tr>
<td>fp</td>
<td>All floating-point registers</td>
<td>Load/Put</td>
</tr>
</tbody>
</table>

### `struct kvm_vcpu_csr`:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Save/Restore</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsstatus</td>
<td>SSTATUS CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vsie</td>
<td>SIE CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vstvec</td>
<td>STVEC CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vsscratch</td>
<td>SSCRATCH CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vsepc</td>
<td>SEPC CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vscause</td>
<td>SCAUSE CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vstval</td>
<td>STVAL CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vsip</td>
<td>SIP CSR</td>
<td>Load/Put</td>
</tr>
<tr>
<td>vsatp</td>
<td>SATP CSR</td>
<td>Load/Put</td>
</tr>
</tbody>
</table>
KVM RISC-V: ONE REG Interface

How can KVM userspace access VCPU context?

- Only KVM_GET_ONE_REG and KVM_SET_ONE_REG IOCTLs available
- Five types of ONE_REG registers: CONFIG, CORE, CSR, FP_F and FP_D
- “isa” CONFIG register can only be written before running the VCPU
- “mode” CORE register has two possible values: 1 (S-mode) and 0 (U-mode)

<table>
<thead>
<tr>
<th>ONE_REG Name</th>
<th>Type</th>
<th>Width</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>isa</td>
<td>CONFIG</td>
<td>32/64</td>
<td>Read-n-Write-before-running</td>
</tr>
<tr>
<td>tbfreq</td>
<td>CONFIG</td>
<td>32/64</td>
<td>Read-Only</td>
</tr>
<tr>
<td>regs.pc</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>regs.ra</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>regs.sp</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>regs.gp</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>regs.tp</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>regs.t0 - regs.t6</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>regs.s0 - regs.s11</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>mode</td>
<td>CORE</td>
<td>32/64</td>
<td>Read-Write</td>
</tr>
<tr>
<td>f[0] - f[31]</td>
<td>FP_F</td>
<td>32</td>
<td>Read-Write</td>
</tr>
<tr>
<td>fcsr</td>
<td>FP_F/FP_D</td>
<td>32</td>
<td>Read-Write</td>
</tr>
<tr>
<td>f[0] – f[31]</td>
<td>FP_D</td>
<td>64</td>
<td>Read-Write</td>
</tr>
</tbody>
</table>
KVM RISC-V Status & Future Work

Where are we? and What next?
KVM RISC-V: Patches

Where are the patches?

• First version of KVM RISC-V series was send-out on July 29\textsuperscript{th} 2019
• Most of the patches are already Reviewed-n-Acked in v6 of KVM RISC-V series
• Recently, we send-out v9 of KVM RISC-V series on October 16\textsuperscript{th} 2019
• KVMTOOL/QEMU upstreaming on-hold until KVM RISC-V is merged in kernel
• Official KVM RISC-V repo on GitHub at: https://github.com/kvm-riscv/linux.git
• KVM RISC-V wiki at: https://github.com/kvm-riscv/howto/wiki
• To play with KVM RISC-V on QEMU refer: https://github.com/kvm-riscv/howto/wiki/KVM-RISCV64-on-QEMU
KVM RISC-V: TODO List

What next?

• Move to RISC-V H-Extension v0.5 draft
• Get 32-bit KVM working
• Bring-up on real-HW or FPGA
• SBI v0.2 base and replacement extensions support
• SBI v0.2 para-virtualized time accounting extension
• Trace points
• KVM unit test support
• Virtualize vector extensions
• Upstream KVMTOOL changes (blocked on KVM RISC-V kernel patches)
• QEMU KVM support (blocked on KVM RISC-V kernel patches)
KVM RISC-V: TODO List (Contd.)

What next?

• In-kernel PLIC emulation
• Guest/VM migration support
• Libvirt support
• Allow 32bit Guest on 64bit Host (*Defined in RISC-V spec*)
• Allow big-endian Guest on little-endian Host and vice-versa (*Defined in RISC-V spec*)
• ..... and more .....
KVM RISC-V Demo

KVM RISC-V running on QEMU RISC-V
Questions?