Western Digital®

Getting started with RISC-V systems for free

Alistair Francis <alistair.francis@wdc.com>

Open Source Summit – Lyon, France

2019

Overview

- What is RISC-V?
- Why RISC-V?
- What is Western Digital doing with RISC-V?
- How can I use RISC-V?
- What is QEMU?
- What is supported in QEMU?
- Demos and Questions

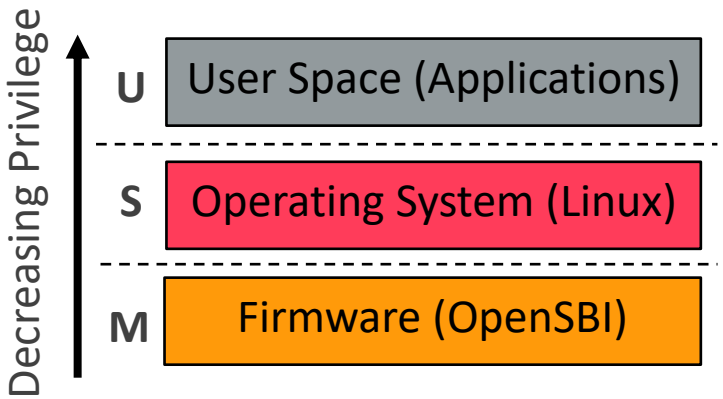
What is RISC-V?

- Pronounced as risk-five
- An open RISC Instruction Set Architecture (ISA)
- The base ISA is incredibly small
  - The ISA can be extended
    - Integer (I) and Embedded (E) are part of the base ISA
    - Multiply/Divide (M), Atomic (A), Single Floating Point (F), Double Floating Point (D), Quad Floating Point (Q) and Compressed Instructions (C) are frozen already
    - Virtualisation (H), Vector (V), Bit Manipulation (B) are examples of draft extensions
    - The General Extension (G) is shorthand for I, M, A, F and D
  - Aimed at everything from small embedded systems to high end HPC
- Multiple CPU implementations already available
  - SiFive, BOOM, Rocket64, Andes, QEMU, PULP, LowRISC, SweRV etc
- Currently heavily backed by SiFive, Microsemi, NVIDIA, Western Digital and others

What is RISC-V ?

Free and Open Instruction Set Architecture (ISA)

- **Clean-slate and Extensible ISA**
- **XLEN (machine word length)** can be 32 (RV32), 64 (RV64), and 128 (RV128)
- **32 general purpose registers**
- **Variable instruction length** (instruction compression)
- **Three privilege modes:** Machine (M-mode), Supervisor (S-mode), and User (U-mode)
- **Control and Status Registers (CSR)** for each privilege mode



General Purpose Registers	
zero	Hardwired-zero register
ra	Return address register
sp	Stack pointer register
gp	Global pointer register
tp	Thread pointer register
a0-a7	Function argument registers
t0-t6	Caller saved registers
s0-s11	Callee saved registers

S-mode CSRs (Used By Linux)	
sstatus	Status
sie	Interrupt Enable
sip	Interrupt Pending
stvec	Trap Handler Base
sepc	Trap Progam Counter
scause	Trap Cause
stval	Trap Value
satp	Address Translation
sscratch	Scratch
NOTE: **sedeleg**, **sideleg**, and **scounteren** not used currently	

Why RISC-V?

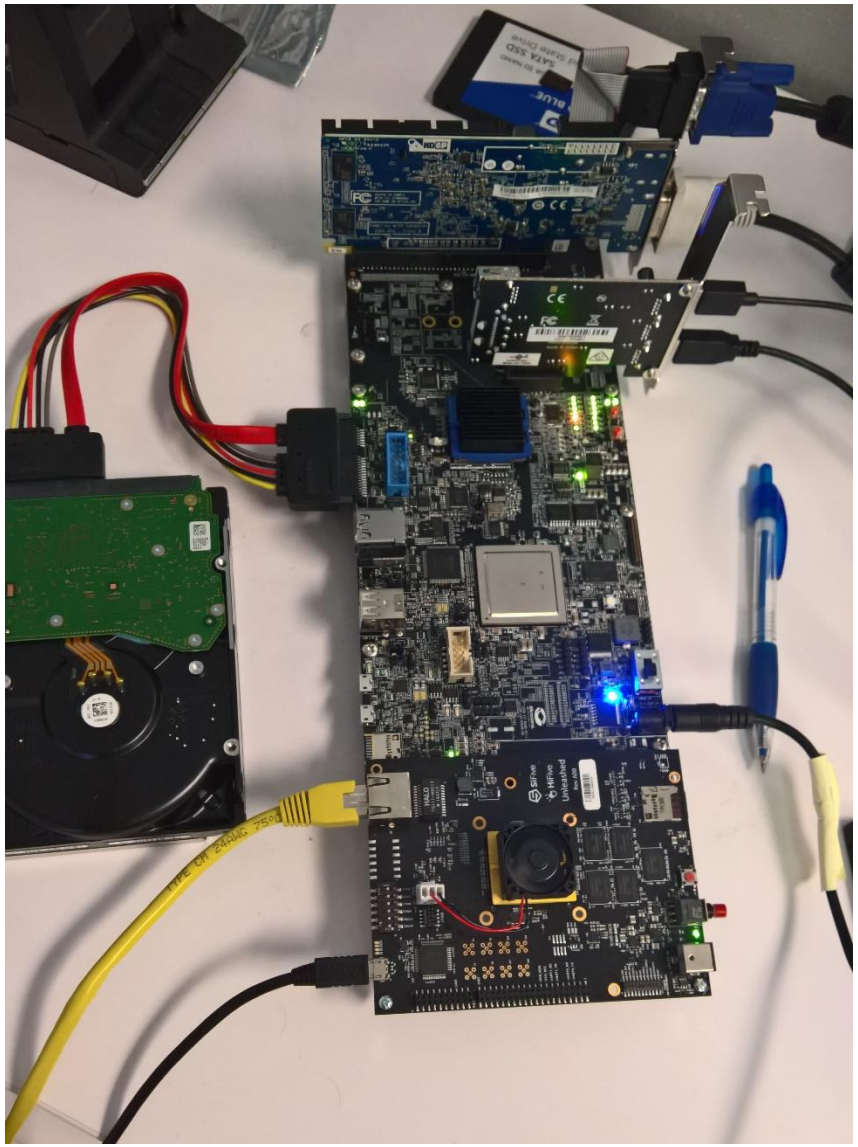
- An open ISA allows anyone to use and modify the ISA without any licensing cost
  - This is different to commercial companies which ship black boxes with strict NDAs
- Customisation for specific applications
- Security audits to verify security in critical applications
- Lower cost users (such as small companies or hobbyists) to use the ISA
- Community driven development approach allows input from anybody
  - The RISC-V foundation still decides on the final specifications

What is Western Digital doing with RISC-V?

- Nearly every Western Digital product has some kind of processing core included
  - Western Digital currently uses one billion cores every year in our products, and are publicly committed to transition our processor cores to RISC-V
  - This allows us to innovate more in our cores as we can customise them
  - Due to the modularity of RISC-V we can also standardise on a single ISA across a range of products
  - We are also working to develop the supporting RISC-V ecosystem
    - This allows us to leverage the open source technology
- RISC-V also allows creating processors that are purpose-built for data centric applications
  - Western Digital's open cache coherent interconnect OmniXtend is an example of this
  - We can connect general purpose RISC-V CPUs with high bandwidth low latency memory fabrics using a standarised memory protocol
    - Large number of RISC-V compute nodes can share a large pool of memory for multi-threaded applications

How can I use it?

- QEMU
  - Support for virt machine and basic models for SiFive machines in mainline
  - Fully supported in QEMU release 3.0
- HiFive Unleashed
  - Only RISC-V hardware capable of booting Linux (with a MMU)
  - MicroSemi expansion board is available to add PCIe and SATA connectivity
  - HiFive Unleashed: $1000
  - Microsemi Expansion Board: $1999



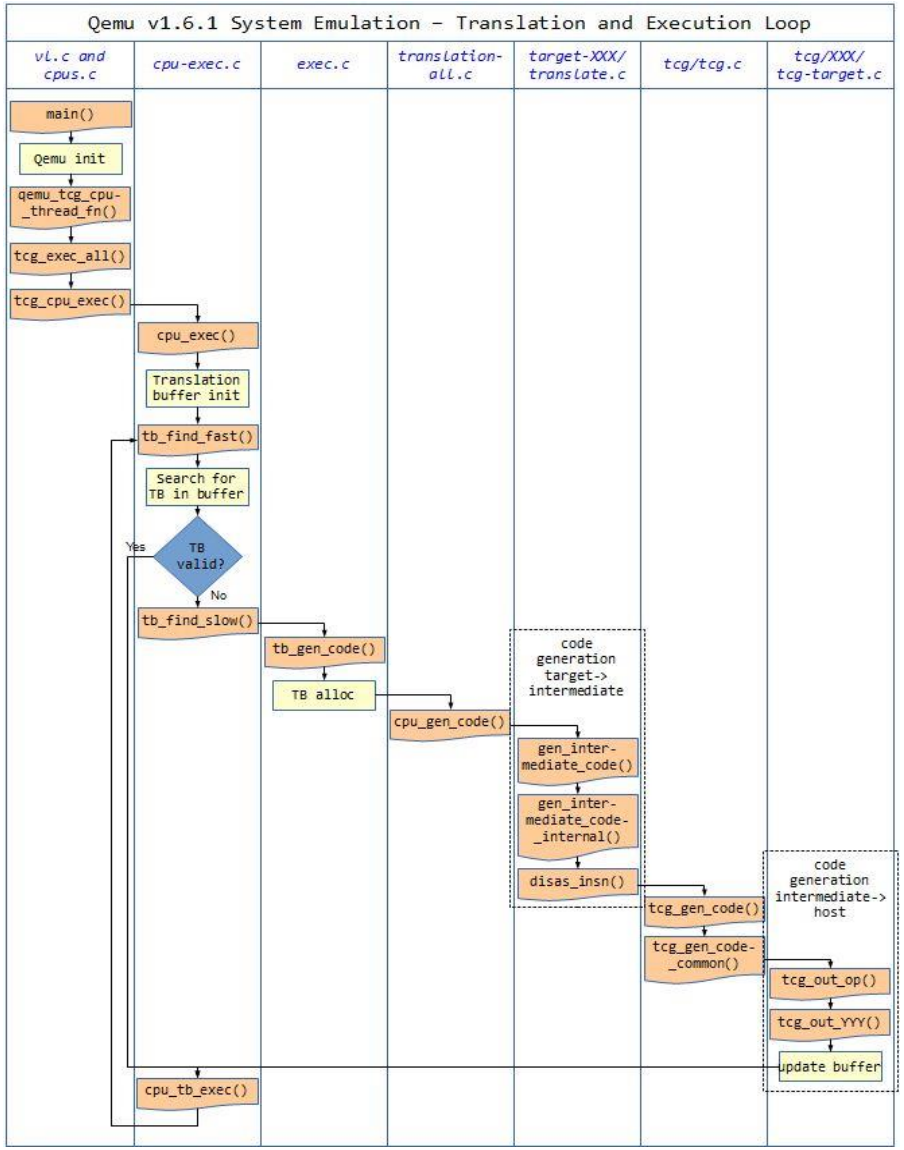
What is QEMU?

- QEMU is a very quick open source (mostly GPLv2) emulator and hypervisor
- It is not cycle accurate, but it is functionally accurate
- It uses the Tiny Code Generator (TCG) to translate different guest architecture instructions to host executable code
  - Supports full system (softMMU) emulation
  - Also supports just Linux/BSD user space translation
- Open source project, not written and maintained by a single company

Benoît Canet – wiki.qemu.org/Logo CC BY-SA

Basics of Tiny Code Generator (TCG)

- TCG began as a backend for a C compiler
- TCG can convert TCG ops to target (host) instructions
  - It also performs some optimisations and liveness analysis to improve performance
- TCG will combine blocks of guest code into a TB blocks
  - The end of a block occurs when a branch/jump instruction is encounted
- TCG currently natively supports these targets (hosts)
  - AArch64, ARMv7, x86, AMD64, MIPS, PPC, PPC64, S390, Sparc and RISC-V



VividD - https://stackoverflow.com/questions/20675226/qemu-code-flow-instruction-cache-and-tcg

Current QEMU Status

- Upstream QEMU fully supports RISC-V
  - Support for virt machine (32-bit and 64-bit)
  - sifive_u Machine (HiFive Unleashed)
  - sifive_e Machine (HiFive 1)
- Support for step by step debugging with GDB
- -bios support (useful for OpenSBI)
  - OpenSBI is even included as the “bios” by default
- RISC-V ISA strings can be customised with the –cpu argument
- Vector Extension and Hypervisor Extension support on list
- Getting started information available at: https://wiki.qemu.org/Documentation/Platforms/RISCV
- Jump on IRC if you have questions: https://freenode.logbot.info/riscv

Current QEMU Status cont.

- RISC-V support for all supported 32-bit and 64-bit hosts
  - Softmmu and Linux User mode are supported
  - RISC-V doesn’t have large guest support (64-bit RISC-V on 32-bit host)
  - RISC-V has support for MTTCG (multithreaded CPUs)
- QEMU support for 64-bit RISC-V hosts
  - Allows running other architecture Operating Systems or applications on RISC-V

QEMU Demo

Debugging OpenSBI with instruction output

```
qemu-system-riscv64 \
    -nographic -machine virt  -m 512 -serial mon:stdio -serial null \
    -bios ./fw_jump-virt.elf
```

QEMU Demo

Connecting GDB to QEMU (sifive_u) and setting break points

```
qemu-system-riscv64 \
    -nographic -machine sifive_u -smp 5 -m 512 -serial mon:stdio -serial null \
    -bios ./fw_jump-sifive_u.elf -s -S

riscv64-oe-linux-gdb
target extended-remote :1234
add-inferior
inferior 2
attach 2
set schedule-multiple
info threads
file ./fw_jump-sifive_u.elf
break sbi_init
c
watch coldboot
c
```

QEMU Demo

Using –pflash loader to develop bootloaders

```
qemu-system-riscv64 \
    -machine virt -m 1G -serial mon:stdio -serial null -nographic \
    -bios ./fw_jump-virt.elf \
    -kernel ./Image \
    -append "root=/dev/vda rw highres=off  console=ttyS0 mem=1G ip=dhcp earlycon=sbi" \
    -device virtio-net-device,netdev=net0,mac=52:54:00:12:34:02 -netdev user,id=net0 \
    -object rng-random,filename=/dev/urandom,id=rng0 -device virtio-rng-device,rng=rng0 \
    -drive id=disk0,file=./Yocto-rootfs.ext4,if=none,format=raw \
    -device virtio-blk-device,drive=disk0 \
    -pflash ./bootblob.bin
```

Western Digital®