QEMU 中 RISC-V 相关实现 —以 SiFive 为例

Gao Zhiyuan
Intern at PLCT, ISRC

About me

- Sophomore at Seoul National University, South Korea
- LV2 intern at PLCT, ISRC
- Interested in Serverless Computing with 3 papers (2 accepted, 1 ongoing)
- Mentee at Google Summer of Code 2019 with Apache Foundation
- Intern at SUSE Beijing in 2018, advocator of OpenSUSE
-
- github.com/alapha23
- medium.com/@alapha23

Some backstory telling

- QEMU supports emulation of SiFive RISCV SoCs
- Nuclei (芯来科技) has RISCV SoCs that we hope to be emulated by QEMU as well
- As a consequence, I have tried to dive into SiFive's riscv implementation.

Emulation Type

Functional: QEMU

binary translating emulator. No instruction-by-instruction trace. Binary translator leads to better simulation performance.

Trace-accurate: Spike aka riscv-isa-sim

Instruction-by-instruction trace with accurate simulation of a RISC-V processor. Spike is the "golden reference" simulator for the RISC-V ISA, and its behavior is the reference for hardware and software. The focus of an interpreter is typically behavioral accuracy for verification.

Cycle-accurate: RTL (Register Transfer Level) implementations

Provides the exact cycle-by-cycle behavior of one particular RISC-V implementation. For example, Rocket's Chisel can be compiled to Verilog, and then compiled to C++ by Verilator to create a cycle-accurate simulator.

RISC-V QEMU Boards

There are now 5 RISC-V boards accessible via QEMU's -machine command line option:

- spike_v1.9 - priv v1.9.1 (HTIF and config-string)
- spike_v1.10 - priv v1.10 (HTIF and device-tree)
- sifive_e300 - priv v1.10 (SiFiveUART, HiFive1 compatible)
- sifive_u500 - priv v1.10 (SiFiveUART and device-tree)
- virt - priv v1.10 (16550A UART, virtio-net, virtio-block and device-tree)

The RISC-V QEMU supports the following instruction set extensions:

- RV32GC with Supervisor-mode and User-mode (RV32IMAFDCSU)
- RV64GC with Supervisor-mode and User-mode (RV64IMAFDCSU)
- Any legal subset of the above (ie, RV32I or RV64IMACSU), as defined by current RISC-V specification

Note:

M: Integer Multiplication and Division instructions

C: Compressed Instructions as 16bits Encoding to reduce code size

A: Atomic Instructions

F: Single-Precision Floating-Point Instructions

D: Double-Precision Floating-Point Instructions

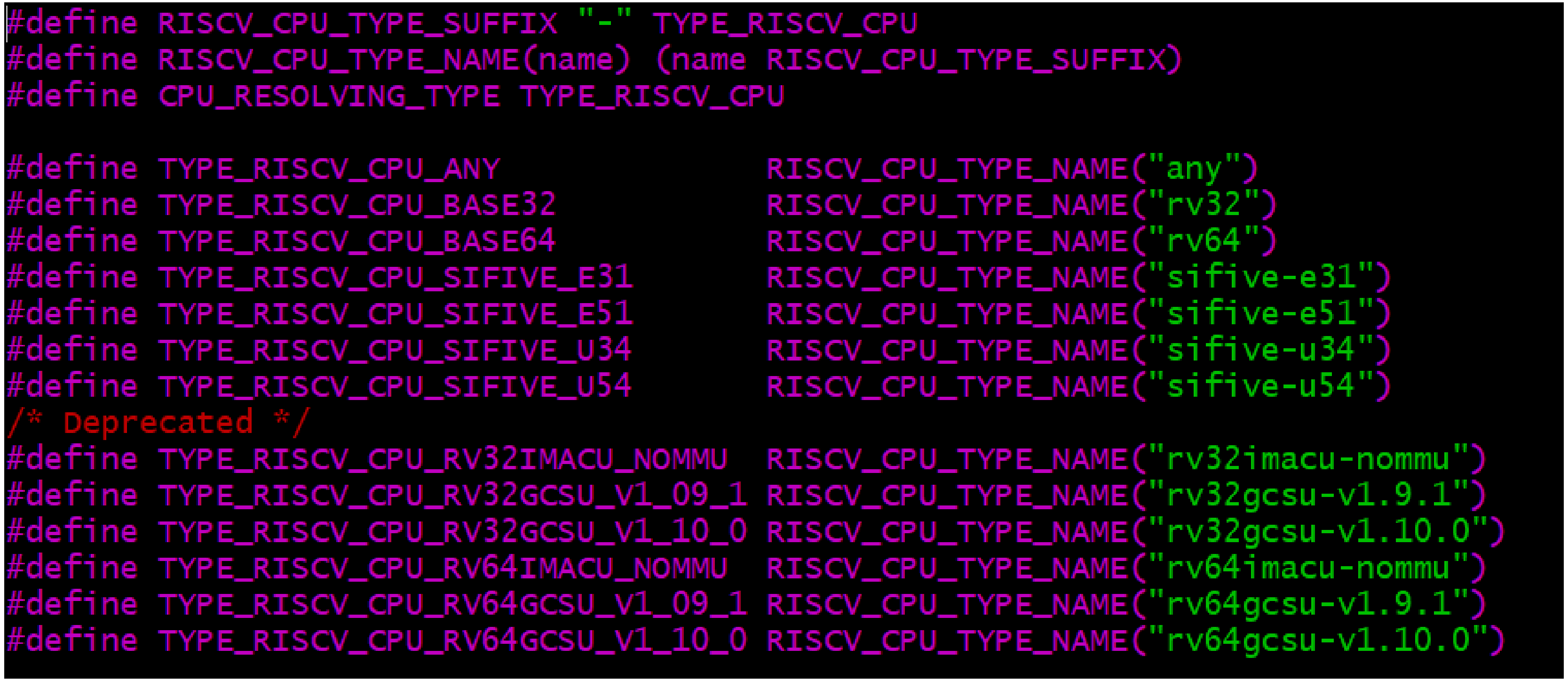
P: Packed-SIMD Instructions

target/riscv/cpu.h

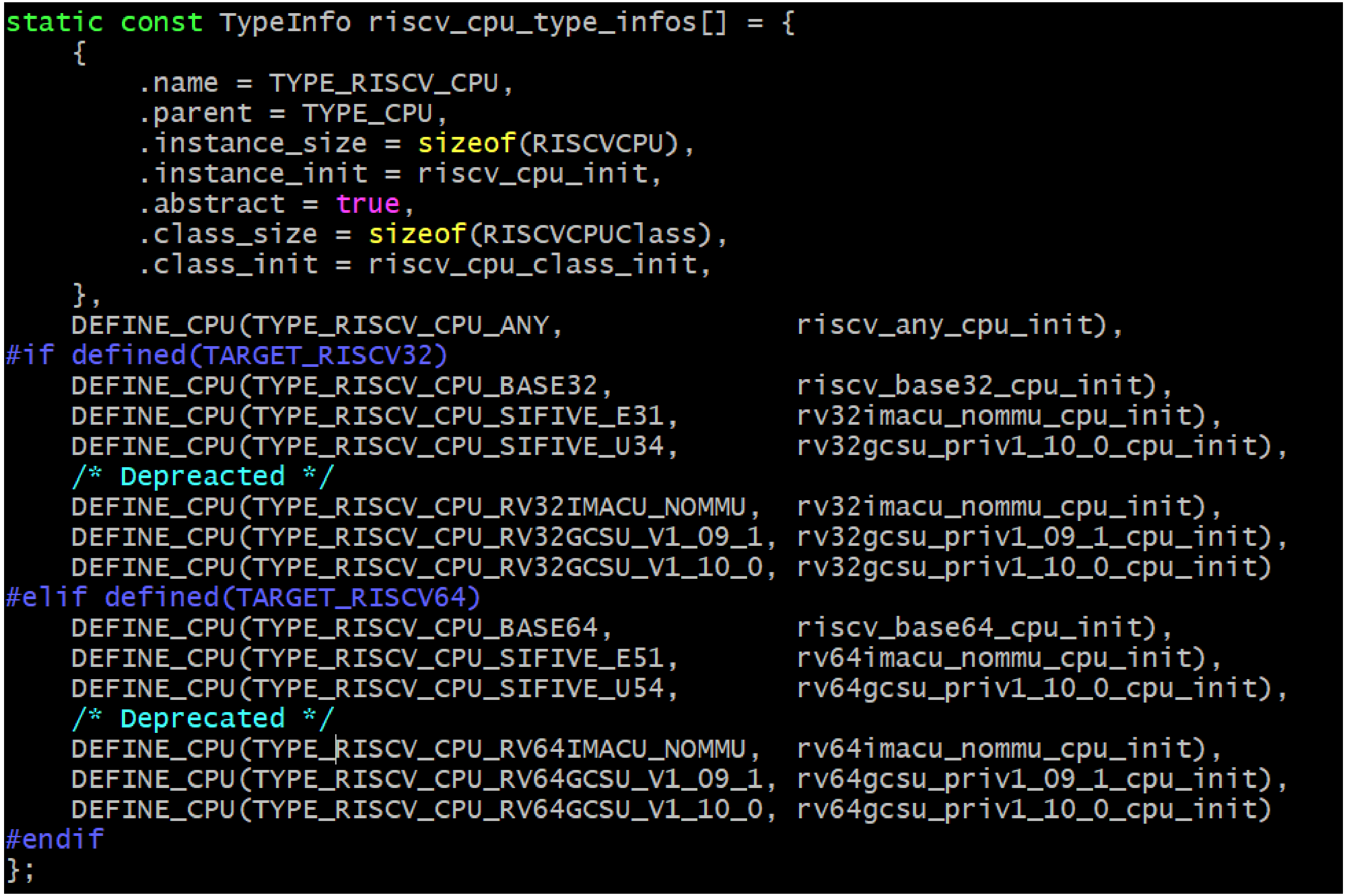
E31 — RV32IMAC Support

E51 — RV64IMAC

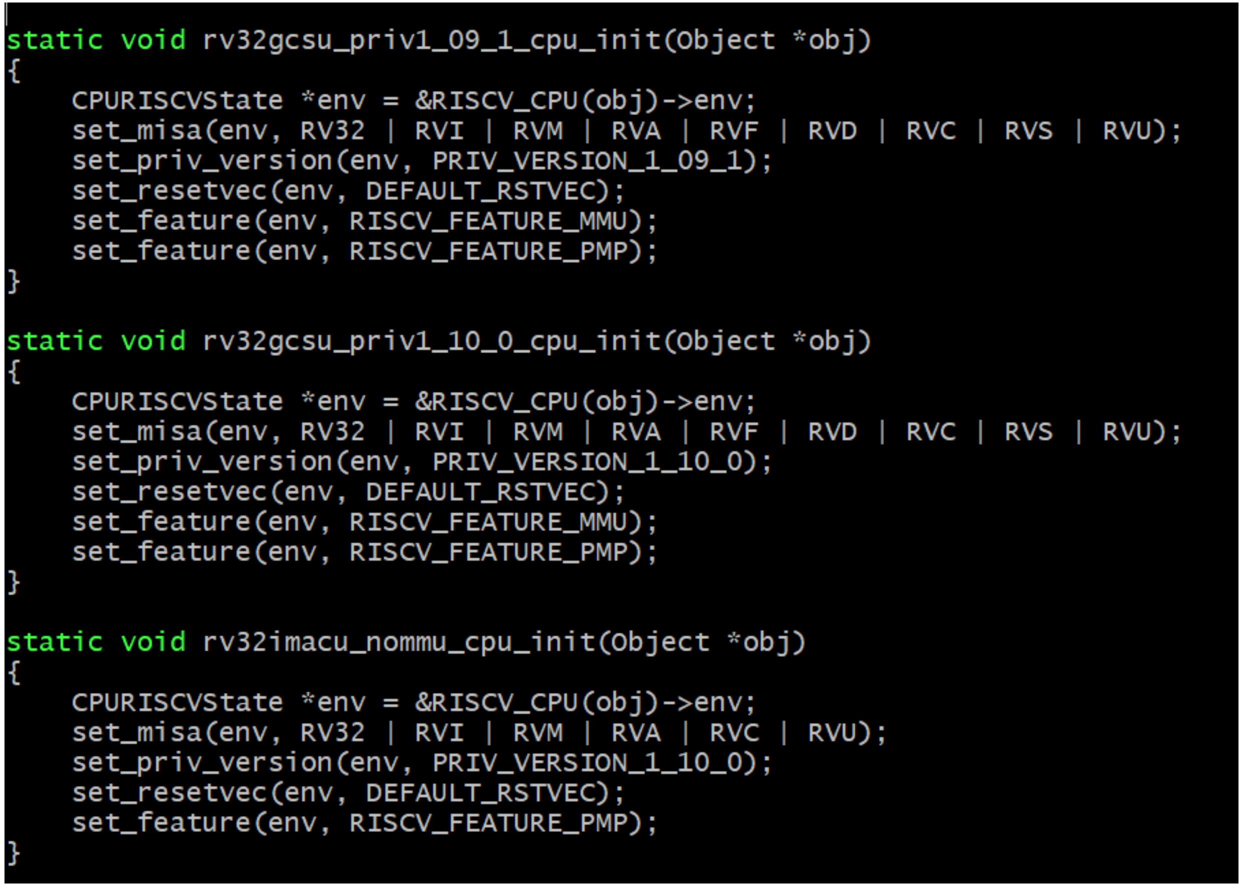
U54 —- RV64GC U54



```
#define RISCV_CPU_TYPE_SUFFIX "-" TYPE_RISCV_CPU
#define RISCV_CPU_TYPE_NAME(name) (name RISCV_CPU_TYPE_SUFFIX)
#define CPU_RESOLVING_TYPE TYPE_RISCV_CPU

#define TYPE_RISCV_CPU_ANY              RISCV_CPU_TYPE_NAME("any")
#define TYPE_RISCV_CPU_BASE32           RISCV_CPU_TYPE_NAME("rv32")
#define TYPE_RISCV_CPU_BASE64           RISCV_CPU_TYPE_NAME("rv64")
#define TYPE_RISCV_CPU_SIFIVE_E31       RISCV_CPU_TYPE_NAME("sifive-e31")
#define TYPE_RISCV_CPU_SIFIVE_E51       RISCV_CPU_TYPE_NAME("sifive-e51")
#define TYPE_RISCV_CPU_SIFIVE_U34       RISCV_CPU_TYPE_NAME("sifive-u34")
#define TYPE_RISCV_CPU_SIFIVE_U54       RISCV_CPU_TYPE_NAME("sifive-u54")
/* Deprecated */
#define TYPE_RISCV_CPU_RV32IMACU_NOMMU  RISCV_CPU_TYPE_NAME("rv32imacu-nommu")
#define TYPE_RISCV_CPU_RV32GCSU_V1_09_1 RISCV_CPU_TYPE_NAME("rv32gcsu-v1.9.1")
#define TYPE_RISCV_CPU_RV32GCSU_V1_10_0 RISCV_CPU_TYPE_NAME("rv32gcsu-v1.10.0")
#define TYPE_RISCV_CPU_RV64IMACU_NOMMU  RISCV_CPU_TYPE_NAME("rv64imacu-nommu")
#define TYPE_RISCV_CPU_RV64GCSU_V1_09_1 RISCV_CPU_TYPE_NAME("rv64gcsu-v1.9.1")
#define TYPE_RISCV_CPU_RV64GCSU_V1_10_0 RISCV_CPU_TYPE_NAME("rv64gcsu-v1.10.0")
```



```c
static const TypeInfo riscv_cpu_type_infos[] = {
    {
        .name = TYPE_RISCV_CPU,
        .parent = TYPE_CPU,
        .instance_size = sizeof(RISCVCPU),
        .instance_init = riscv_cpu_init,
        .abstract = true,
        .class_size = sizeof(RISCVCPUClass),
        .class_init = riscv_cpu_class_init,
    },
    DEFINE_CPU(TYPE_RISCV_CPU_ANY,                riscv_any_cpu_init),
#if defined(TARGET_RISCV32)
    DEFINE_CPU(TYPE_RISCV_CPU_BASE32,             riscv_base32_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_E31,         rv32imacu_nommu_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_U34,         rv32gcsu_priv1_10_0_cpu_init),
    /* Depreacted */
    DEFINE_CPU(TYPE_RISCV_CPU_RV32IMACU_NOMMU,   rv32imacu_nommu_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_RV32GCSU_V1_09_1, rv32gcsu_priv1_09_1_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_RV32GCSU_V1_10_0, rv32gcsu_priv1_10_0_cpu_init)
#elif defined(TARGET_RISCV64)
    DEFINE_CPU(TYPE_RISCV_CPU_BASE64,             riscv_base64_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_E51,         rv64imacu_nommu_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_U54,         rv64gcsu_priv1_10_0_cpu_init),
    /* Deprecated */
    DEFINE_CPU(TYPE_RISCV_CPU_RV64IMACU_NOMMU,   rv64imacu_nommu_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_RV64GCSU_V1_09_1, rv64gcsu_priv1_09_1_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_RV64GCSU_V1_10_0, rv64gcsu_priv1_10_0_cpu_init)
#endif
};
```


```
static void rv32gcsu_priv1_09_1_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV32 | RVI | RVM | RVA | RVF | RVD | RVC | RVS | RVU);
    set_priv_version(env, PRIV_VERSION_1_09_1);
    set_resetvec(env, DEFAULT_RSTVEC);
    set_feature(env, RISCV_FEATURE_MMU);
    set_feature(env, RISCV_FEATURE_PMP);
}

static void rv32gcsu_priv1_10_0_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV32 | RVI | RVM | RVA | RVF | RVD | RVC | RVS | RVU);
    set_priv_version(env, PRIV_VERSION_1_10_0);
    set_resetvec(env, DEFAULT_RSTVEC);
    set_feature(env, RISCV_FEATURE_MMU);
    set_feature(env, RISCV_FEATURE_PMP);
}

static void rv32imacu_nommu_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV32 | RVI | RVM | RVA | RVC | RVU);
    set_priv_version(env, PRIV_VERSION_1_10_0);
    set_resetvec(env, DEFAULT_RSTVEC);
    set_feature(env, RISCV_FEATURE_PMP);
}
```

What's in RISC-V QEMU

- Support for privileged ISA v1.10 and v1.9.1 (spike_v1.10 board)
- Backwards compatibility for privileged ISA v1.9.1 (spike_v1.9 board)
- Parameterizable CLINT (Core Local Interruptor)
- Parameterizable PLIC (Platform Level Interrupt Controller)
- sifive_e300 board that can run binaries targeting the SiFive E300 SDK
- sifive_u500 board that can run binaries targeting the SiFive U500 SDK
- virt board with VirtIO MMIO (virtio-net, virtio-block, etc)
- UART emulation on the 'virt' board
- HTIF Console (Host Target Interface) for Spike emulation
- Device-tree config for the spike_v1.10, virt and sifive_u500 boards

File structure

Implementation

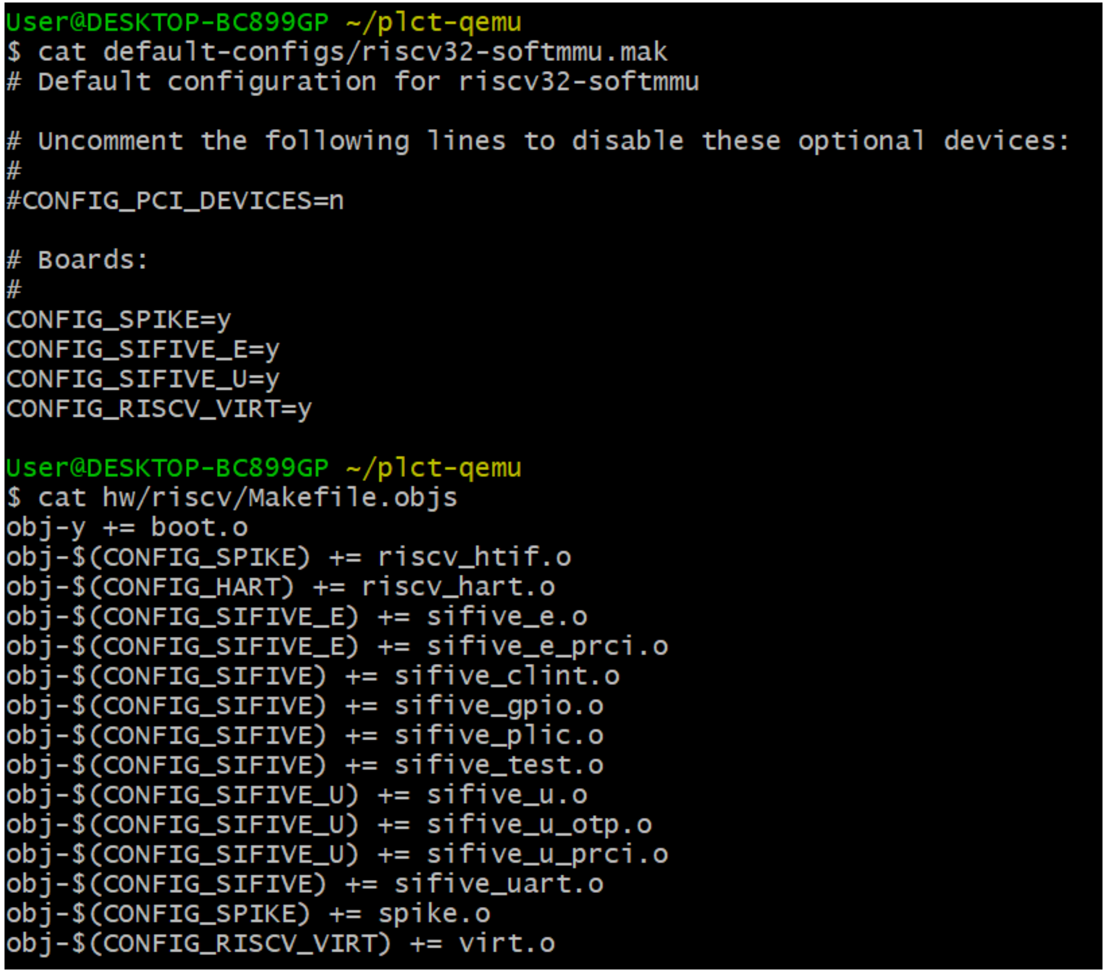
hw/riscv/sifive_u.c	— Provides a board compatible with the SiFive Freedom U SDK
hw/riscv/sifive_u_prci.c	— PRCI refers to Power, Reset, Clock, Interrupt
hw/riscv/sifive_u_otp.c	— SiFive U OTP (One-Time Programmable) Memory interface
hw/riscv/sifive_e.c	— Provides a board compatible with the SiFive Freedom E SDK
hw/riscv/sifive_e_prci.c	
hw/riscv/sifive_uart.c	— Models the UART on the SiFive E300 and U500 series SOC

Peripheral definitions

hw/riscv/sifive_plic.c

hw/riscv/sifive_gpio.c

hw/riscv/sifive_clint.c



```
User@DESKTOP-BC899GP ~/plct-qemu
$ cat default-configs/riscv32-softmmu.mak
# Default configuration for riscv32-softmmu

# Uncomment the following lines to disable these optional devices:
#
#CONFIG_PCI_DEVICES=n

# Boards:
#
CONFIG_SPIKE=y
CONFIG_SIFIVE_E=y
CONFIG_SIFIVE_U=y
CONFIG_RISCV_VIRT=y

User@DESKTOP-BC899GP ~/plct-qemu
$ cat hw/riscv/Makefile.objs
obj-y += boot.o
obj-$(CONFIG_SPIKE) += riscv_htif.o
obj-$(CONFIG_HART) += riscv_hart.o
obj-$(CONFIG_SIFIVE_E) += sifive_e.o
obj-$(CONFIG_SIFIVE_E) += sifive_e_prci.o
obj-$(CONFIG_SIFIVE) += sifive_clint.o
obj-$(CONFIG_SIFIVE) += sifive_gpio.o
obj-$(CONFIG_SIFIVE) += sifive_plic.o
obj-$(CONFIG_SIFIVE) += sifive_test.o
obj-$(CONFIG_SIFIVE_U) += sifive_u.o
obj-$(CONFIG_SIFIVE_U) += sifive_u_otp.o
obj-$(CONFIG_SIFIVE_U) += sifive_u_prci.o
obj-$(CONFIG_SIFIVE) += sifive_uart.o
obj-$(CONFIG_SPIKE) += spike.o
obj-$(CONFIG_RISCV_VIRT) += virt.o
```

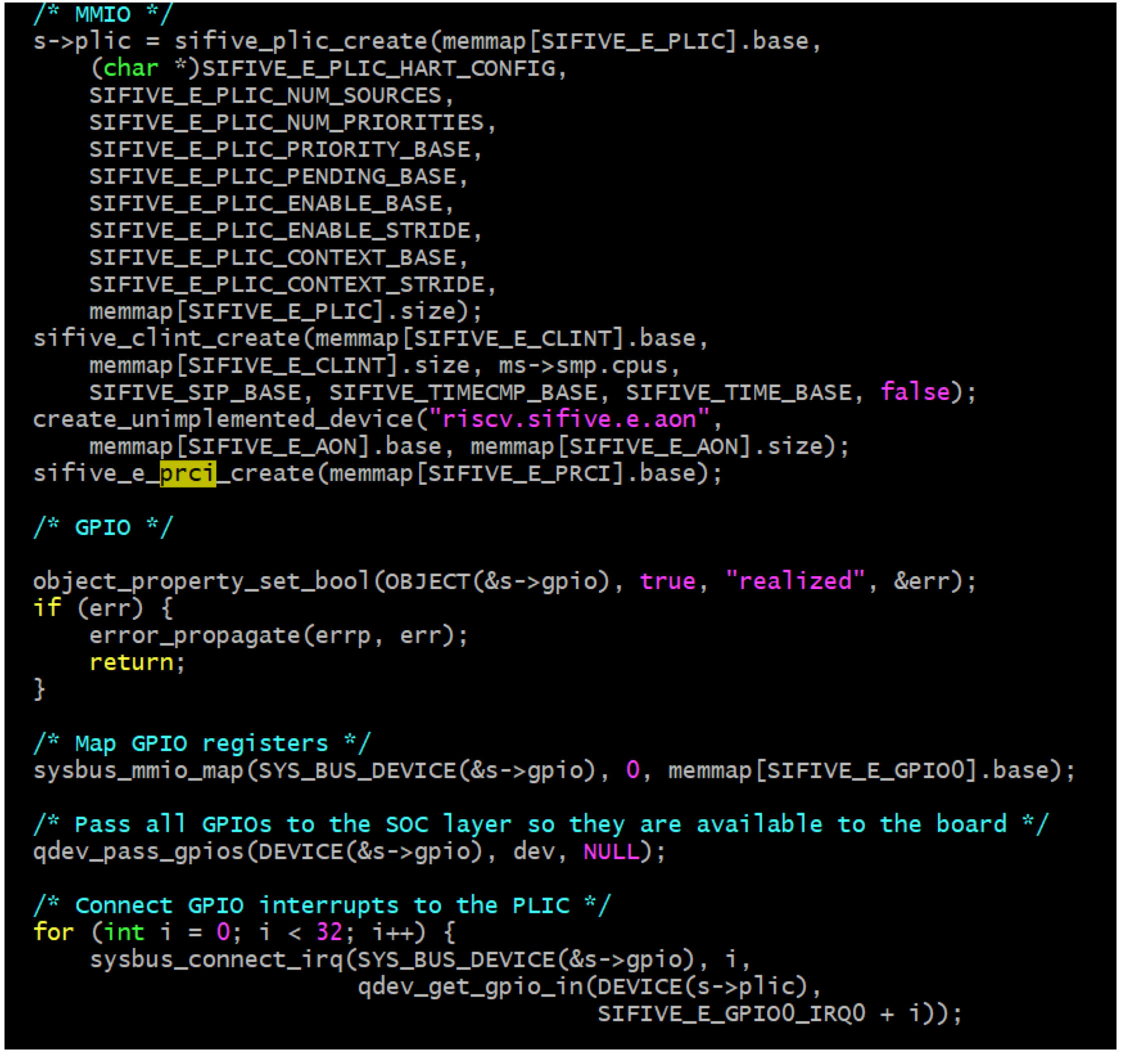
sifive_u vs sifive_e

Same

- UART CLINT (Core Level Interruptor) PLIC (Platform Level Interrupt Controller) PRCI (Power, Reset, Clock, Interrupt) Registers emulated as RAM: AON, GPIO, QSPI, PWM Flash memory emulated as RAM

Sifive_u has

- OTP (One-Time Programmable) memory with stored serial number
- GEM (Gigabit Ethernet Controller) and management block

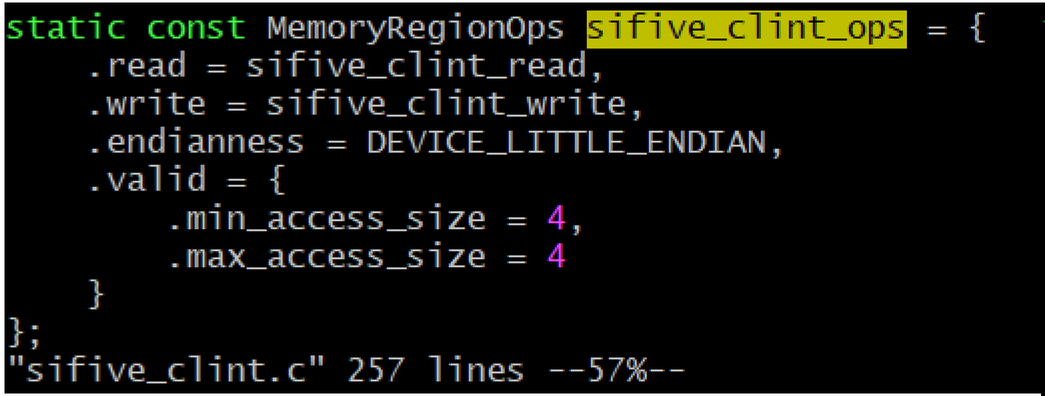


```
/* MMIO */
s->plic = sifive_plic_create(memmap[SIFIVE_E_PLIC].base,
    (char *)SIFIVE_E_PLIC_HART_CONFIG,
    SIFIVE_E_PLIC_NUM_SOURCES,
    SIFIVE_E_PLIC_NUM_PRIORITIES,
    SIFIVE_E_PLIC_PRIORITY_BASE,
    SIFIVE_E_PLIC_PENDING_BASE,
    SIFIVE_E_PLIC_ENABLE_BASE,
    SIFIVE_E_PLIC_ENABLE_STRIDE,
    SIFIVE_E_PLIC_CONTEXT_BASE,
    SIFIVE_E_PLIC_CONTEXT_STRIDE,
    memmap[SIFIVE_E_PLIC].size);
sifive_clint_create(memmap[SIFIVE_E_CLINT].base,
    memmap[SIFIVE_E_CLINT].size, ms->smp.cpus,
    SIFIVE_SIP_BASE, SIFIVE_TIMECMP_BASE, SIFIVE_TIME_BASE, false);
create_unimplemented_device("riscv.sifive.e.aon",
    memmap[SIFIVE_E_AON].base, memmap[SIFIVE_E_AON].size);
sifive_e_prci_create(memmap[SIFIVE_E_PRCI].base);

/* GPIO */

object_property_set_bool(OBJECT(&s->gpio), true, "realized", &err);
if (err) {
    error_propagate(errp, err);
    return;
}

/* Map GPIO registers */
sysbus_mmio_map(SYS_BUS_DEVICE(&s->gpio), 0, memmap[SIFIVE_E_GPIO0].base);

/* Pass all GPIOs to the SOC layer so they are available to the board */
qdev_pass_gpios(DEVICE(&s->gpio), dev, NULL);

/* Connect GPIO interrupts to the PLIC */
for (int i = 0; i < 32; i++) {
    sysbus_connect_irq(SYS_BUS_DEVICE(&s->gpio), i,
                       qdev_get_gpio_in(DEVICE(s->plic),
                                        SIFIVE_E_GPIO0_IRQ0 + i));
```

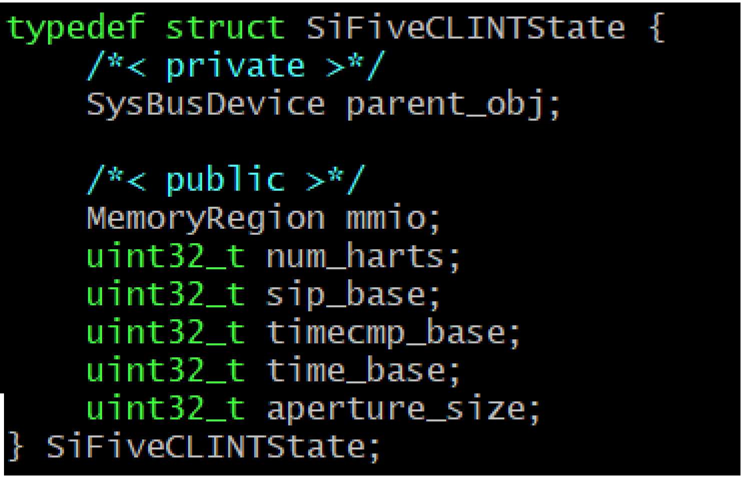
sifive_e

SiFive CLINT (Core Local Interruptor)

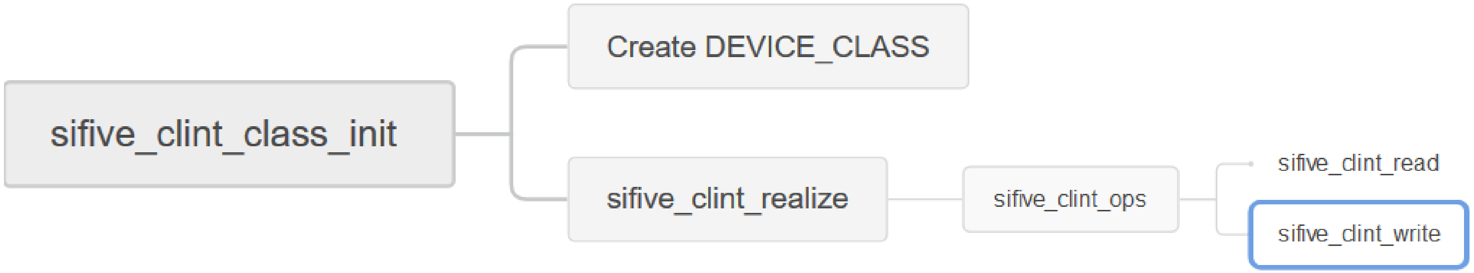
The SiFive CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. In prior versions of QEMU, there was a separate RTC and interruptor. In the latest version of QEMU, the IPIs (Inter-Processor Interrupts) have been implemented.

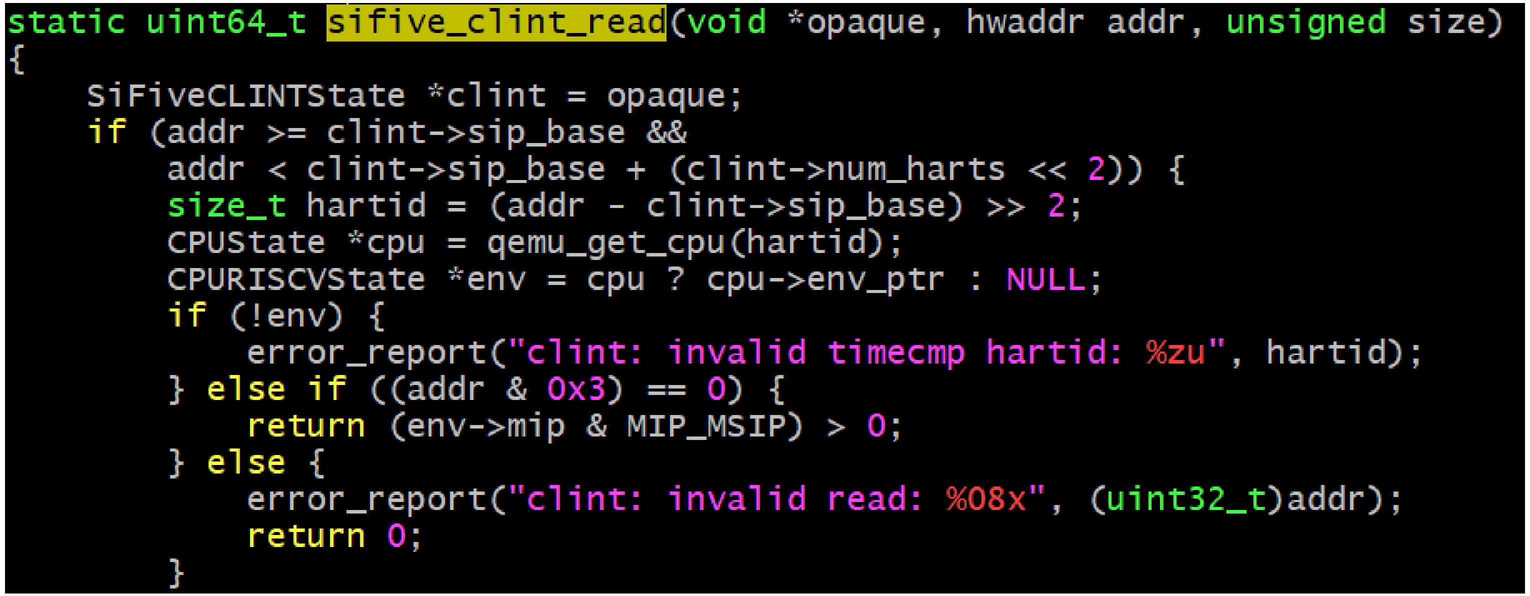


```
static const MemoryRegionOps sifive_clint_ops = {
    .read = sifive_clint_read,
    .write = sifive_clint_write,
    .endianness = DEVICE_LITTLE_ENDIAN,
    .valid = {
        .min_access_size = 4,
        .max_access_size = 4
    }
};
"sifive_clint.c" 257 lines --57%--
```



```
typedef struct SiFiveCLINTState {
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
    MemoryRegion mmio;
    uint32_t num_harts;
    uint32_t sip_base;
    uint32_t timecmp_base;
    uint32_t time_base;
    uint32_t aperture_size;
} SiFiveCLINTState;
```





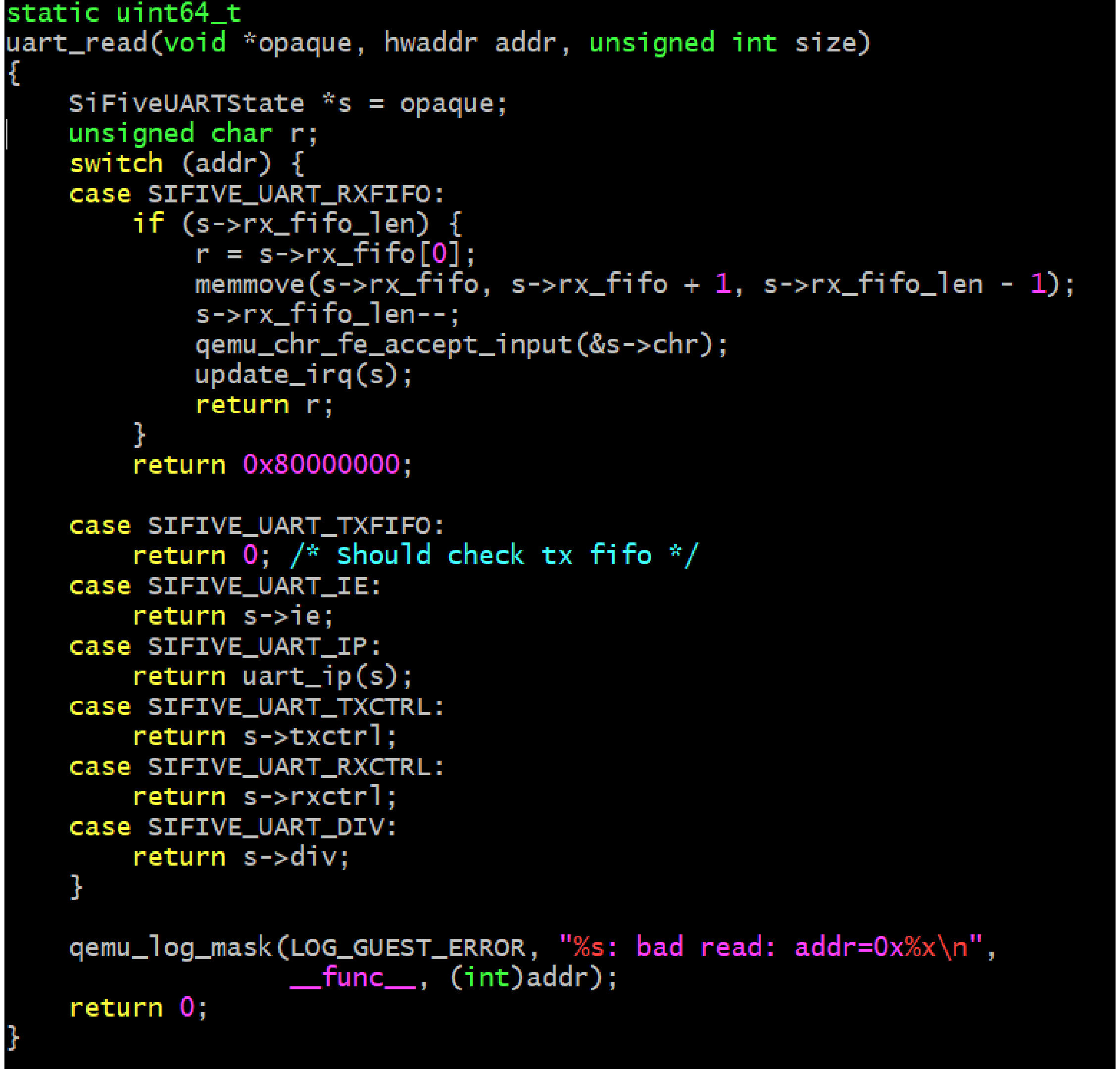
```
static uint64_t sifive_clint_read(void *opaque, hwaddr addr, unsigned size)
{
    SiFiveCLINTState *clint = opaque;
    if (addr >= clint->sip_base &&
        addr < clint->sip_base + (clint->num_harts << 2)) {
        size_t hartid = (addr - clint->sip_base) >> 2;
        CPUState *cpu = qemu_get_cpu(hartid);
        CPURISCVState *env = cpu ? cpu->env_ptr : NULL;
        if (!env) {
            error_report("clint: invalid timecmp hartid: %zu", hartid);
        } else if ((addr & 0x3) == 0) {
            return (env->mip & MIP_MSIP) > 0;
        } else {
            error_report("clint: invalid read: %08x", (uint32_t)addr);
            return 0;
        }
```

SiFive PLIC (Platform Level Interrupt Controller)

The SiFive platform-level interrupt controller (PLIC) prioritizes and distributes global interrupts in a RISC-V system.
The standard RISC-V platform-level interrupt controller (PLIC) provides centralized interrupt prioritization and routing for shared platform-level interrupts, and sends only a single external interrupt signal per privilege mode (meip/seip/ueip) to each hart.
The QEMU PLIC implementation supports a parameterizable number of interrupt sources and priorities. A priority can be set per each interrupt source and for each target context (hart and mode). Each context has a read-write enable bitmask and a read-only pending bitmask. The new QEMU PLIC implementation is compatible with hardware on the FE310G000 SOC, the SiFive Freedom-E-SDK and the SiFive Freedom-U-SDK.

UART

- Create UART device
- uart read
- uart write



```c
static uint64_t
uart_read(void *opaque, hwaddr addr, unsigned int size)
{
    SiFiveUARTState *s = opaque;
    unsigned char r;
    switch (addr) {
    case SIFIVE_UART_RXFIFO:
        if (s->rx_fifo_len) {
            r = s->rx_fifo[0];
            memmove(s->rx_fifo, s->rx_fifo + 1, s->rx_fifo_len - 1);
            s->rx_fifo_len--;
            qemu_chr_fe_accept_input(&s->chr);
            update_irq(s);
            return r;
        }
        return 0x80000000;

    case SIFIVE_UART_TXFIFO:
        return 0; /* Should check tx fifo */
    case SIFIVE_UART_IE:
        return s->ie;
    case SIFIVE_UART_IP:
        return uart_ip(s);
    case SIFIVE_UART_TXCTRL:
        return s->txctrl;
    case SIFIVE_UART_RXCTRL:
        return s->rxctrl;
    case SIFIVE_UART_DIV:
        return s->div;
    }

    qemu_log_mask(LOG_GUEST_ERROR, "%s: bad read: addr=0x%x\n",
                  __func__, (int)addr);
    return 0;
}
```