从零开始的RISC-V模拟器开发

第10讲 QEMU篇之HelloWorld

中国科学院软件研究所
PLCT实验室

王俊强 wangjunqiang@iscas.ac.cn
李威威 liweiwei@iscas.ac.cn
吴伟 wuwei2016@iscas.ac.cn

本课内容

HelloWorld基础运行

- Uart设备初步实现
- SysTimer设备初步实现

Nuclei 内存模拟之加载运行

测试命令：

```
$qemu-system-riscv32  \
-nographic -M mcu_200t,msel=1  \
-d in_asm \
 -kernel ../hbird/ilm/helloworld.elf
```

$riscv-nuclei-elf-objdump -d helloworld.elf > hellowoeld.asm

```
800000cc <_start>:
800000cc: 30047073 csrci mstatus,8
800000d0: 10000197 auipc gp,0x10000
800000d4: 79018193 addi  gp,gp,1936 # 90000860 <__global_pointer$>
800000d8: 10010117 auipc sp,0x10010
800000dc: f2810113 addi  sp,sp,-216 # 90010000 <_sp>
800000e0: 20000293 li    t0,512
800000e4: 7d02a073 csrs  0x7d0,t0
800000e8: 00000297 auipc t0,0x0
800000ec: f1828293 addi  t0,t0,-232 # 80000000 <vector_base>
```

```
QEMU 5.2.90 monitor - type 'help' for more information
(qemu) ----------------
IN:
Priv: 3; Virt: 0
0x00001000:  00000297          auipc           t0,0            # 0x1000
0x00001004:  02028593          addi            a1,t0,32
0x00001008:  f1402573          csrrs           a0,mhartid,zero

----------------
IN:
Priv: 3; Virt: 0
0x0000100c:  0182a283          lw              t0,24(t0)
0x00001010:  00028067          jr              t0

----------------
IN:
Priv: 3; Virt: 0
0x80000000:  0cc0006f          j               204             # 0x800000cc

----------------
IN: _start
Priv: 3; Virt: 0
0x800000cc:  30047073          csrrci          zero,mstatus,8

----------------
IN: _start
Priv: 3; Virt: 0
0x800000d0:  10000197          auipc           gp,268435456    # 0x900000d0
0x800000d4:  79018193          addi            gp,gp,1936
0x800000d8:  10010117          auipc           sp,268500992    # 0x900100d8
0x800000dc:  f2810113          addi            sp,sp,-216
0x800000e0:  20000293          addi            t0,zero,512
0x800000e4:  7d02a073          csrrs           zero,0x7d0,t0
```

The Way of HelloWorld

- GDB
- QEMU -d选项
- 源码
- 反汇编源码
- 手册
- QEMU TCG plugin
- ……

```
$qemu-system-riscv32  -nographic -machine mcu_200t,msel=1 \
-kernel helloworld.elf -nodefaults -serial stdio \
-s -S
$riscv-nuclei-elf-gdb ./ helloworld.elf
(gdb)target remote :1234
(gdb)c
(gdb)n
```

```
Log items (comma separated):
in_asm          show target assembly code for each compiled TB
unimp           log unimplemented functionality
guest_errors    log when the guest OS does something invalid (eg accessing a
non-existent register)
```

gcc_demosoc_ilm.ld → ENTRY(_start) → startup_demosoc.S

system_demosoc.c

hellowoeld.asm

```
800000cc <_start>:
800000cc: 30047073 csrci mstatus,8
800000d0: 10000197 auipc gp,0x10000
800000d4: 79018193 addi  gp,gp,1936
```

```
-plugin ./build/tests/plugin/libinsn.so -D output -d plugin
```

使用GDB

```
$qemu-system-riscv32  -nographic -machine mcu_200t,msel=1 \
-kernel helloworld.elf -nodefaults -serial stdio \
-s -S
$riscv-nuclei-elf-gdb ./ helloworld.elf
(gdb)target remote :1234
(gdb)c
(gdb)n
```

出错流程：

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./hbird/ilm/helloworld.elf...
(gdb) target remote :1234
Remote debugging using :1234
0x00001000 in ?? ()
(gdb) b _start
Breakpoint 1 at 0x800000cc: file ../../../SoC/demosoc/Common/Source/GCC/startup_demosoc.S, lin
(gdb) c
Continuing.

Breakpoint 1, _start () at ../../../SoC/demosoc/Common/Source/GCC/startup_demosoc.S:176
176         csrc CSR_MSTATUS, MSTATUS_MIE
(gdb) n
181         la gp, __global_pointer$
(gdb) n
183         la sp, _sp
(gdb) n
_start () at ../../../SoC/demosoc/Common/Source/GCC/startup_demosoc.S:190
190         li t0, MMISC_CTL_NMI_CAUSE_FFF
(gdb) n
191         csrs CSR_MMISC_CTL, t0
(gdb) n
0x00000000 in ?? ()
(gdb) n
Cannot find bounds of current function
```

可能的错误： flen is 0 CPU没有支持FD扩展

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./hbird/ilm/helloworld.elf...
(gdb) target remote :1234
Remote debugging using :1234
bfd requires flen 8, but target has flen 0
```

```
set_misa(env, …… | RVF | RVD ……);
```

正确流程：

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./hbird/ilm/helloworld.elf...
(gdb) target remote :1234
Remote debugging using :1234
0x00001000 in ?? ()
(gdb) b _start
Breakpoint 1 at 0x800000cc: file ../../../SoC/demosoc/Common/Source/GCC/startup_demosoc.S, line 176.
(gdb) c
Continuing.

Breakpoint 1, _start () at ../../../SoC/demosoc/Common/Source/GCC/startup_demosoc.S:176
176         csrc CSR_MSTATUS, MSTATUS_MIE
(gdb) n
181         la gp, __global_pointer$
(gdb) n
183         la sp, _sp
(gdb) n
_start () at ../../../SoC/demosoc/Common/Source/GCC/startup_demosoc.S:190
190         li t0, MMISC_CTL_NMI_CAUSE_FFF
(gdb) n
191         csrs CSR_MMISC_CTL, t0
(gdb) n
197         la t0, vector_base
(gdb) n
198         csrw CSR_MTVT, t0
(gdb) n
206         la t0, irq_entry
(gdb) n
207         csrw CSR_MTVT2, t0
(gdb) n
208         csrs CSR_MTVT2, 0x1
```

辅助之源码

startup_demosoc.S

```
_start:
    /* ===== Startup Stage 1 ===== */
    csrc CSR_MSTATUS, MSTATUS_MIE
    ……
    csrs CSR_MMISC_CTL, t0
    ……
    csrw CSR_MTVT, t0
    ……
    csrw CSR_MTVT2, t0
    csrs CSR_MTVT2, 0x1
    ……
    csrw CSR_MTVEC, t0
    ……
    csrc CSR_MTVEC, t0
    csrs CSR_MTVEC, 0x3
    ……
    csrci CSR_MCOUNTINHIBIT, 0x5
    ……
    call SystemInit
    ……
    call __libc_init_array
    ……
    call _premain_init
    ……
    call main
```

system_demosoc.c

```
#ifndef SYSTEM_CLOCK
#define SYSTEM_CLOCK    (80000000UL)
#endif

void SystemInit(void)
{
    SystemCoreClock = SYSTEM_CLOCK;
}
```

```
void _premain_init(void)
{
    /* TODO: Add your own initialization code here, called before main */
    /* __ICACHE_PRESENT and __DCACHE_PRESENT are defined in demosoc.h */
#if defined(__ICACHE_PRESENT) && __ICACHE_PRESENT == 1
    EnableICache();
#endif
#if defined(__DCACHE_PRESENT) && __DCACHE_PRESENT == 1
    EnableDCache();
#endif
    SystemCoreClock = get_cpu_freq();
    gpio_iof_config(GPIO, IOF0_UART0_MASK, IOF_SEL_0);
    uart_init(SOC_DEBUG_UART, 115200);
    /* Display banner after UART initialized */
    SystemBannerPrint();
    /* Initialize exception default handlers */
    Exception_Init();
    /* ECLIC initialization, mainly MTH and NLBIT */
    ECLIC_Init();
}
```

辅助之源码

system_demosoc.c

```c
#define DEMOSOC_PERIPH_BASE                     (0x10000000UL)
#define UART0_BASE            (DEMOSOC_PERIPH_BASE + 0x13000)
#define UART0                   ((UART_TypeDef *) UART0_BASE)

#define SOC_DEBUG_UART      UART0

typedef struct {
    __IOM uint32_t TXFIFO;
    __IOM uint32_t RXFIFO;
    __IOM uint32_t TXCTRL;
    __IOM uint32_t RXCTRL;
    __IOM uint32_t IE;
    __IOM uint32_t IP;
    __IOM uint32_t DIV;
} UART_TypeDef;

int32_t uart_init(UART_TypeDef* uart, uint32_t baudrate)
{
    if (__RARELY(uart == NULL)) {
        return -1;
    }
    uart->DIV = SystemCoreClock / baudrate - 1;
    uart->TXCTRL |= UART_TXEN;
    uart->RXCTRL |= UART_RXEN;
    return 0;
}
```

```c
uint32_t measure_cpu_freq(uint32_t n)
{
    uint32_t start_mcycle, delta_mcycle;
    uint32_t start_mtime, delta_mtime;
    uint32_t mtime_freq = get_timer_freq();

    // Don't start measuruing until we see an mtime tick
    uint32_t tmp = (uint32_t)SysTimer_GetLoadValue();
    do {
        start_mtime = (uint32_t)SysTimer_GetLoadValue();
        start_mcycle = __RV_CSR_READ(CSR_MCYCLE);
    } while (start_mtime == tmp);

    do {
        delta_mtime = (uint32_t)SysTimer_GetLoadValue() - start_mtime;
        delta_mcycle = __RV_CSR_READ(CSR_MCYCLE) - start_mcycle;
    } while (delta_mtime < n);

    return (delta_mcycle / delta_mtime) * mtime_freq
           + ((delta_mcycle % delta_mtime) * mtime_freq) / delta_mtime;
}

uint32_t get_cpu_freq()
{
    uint32_t cpu_freq;

    // warm up
    measure_cpu_freq(1);
    // measure for real
    cpu_freq = measure_cpu_freq(100);

    return cpu_freq;
}
```

辅助之源码

nuclei-sdk\application\baremetal\helloworld\main.c

```c
int main(void)
{
    srand(__get_rv_cycle()  | __get_rv_instret() | __RV_CSR_READ(CSR_MCYCLE));
    uint32_t rval = rand();
    rv_csr_t misa = __RV_CSR_READ(CSR_MISA);

    printf("MISA: 0x%lx\r\n", misa);
    print_misa();

    for (int i = 0; i < 20; i ++) {
        printf("%d: Hello World From Nuclei RISC-V Processor!\r\n", i);
    }

    return 0;
}
```

CSR实现

target\riscv\cpu_bits.h

```
#define CSR_MMISC_CTL 0x07d0    MRW
#define CSR_MTVT      0x307   MRW
#define CSR_MTVT2     0x07ec MRW
/* Update to #define CSR_MCOUNTINHIBIT 0x320 for 1.11.0 */
#define CSR_MUCOUNTEREN     0x320  MRW
```

MMISC_CTL

N级别处理器内核 自定义 mmisc_ctl寄存器用于控制 NMI Misaligned Access和 BPU的相关功能。

MTVT

寄存器用于保存 ECLIC中断向量表的基地址，此基地址至少为 64byte对齐。

MTVT2

用于指定 ECLIC非向量模式的中断 common code入口地址。

MCOUNTINHIBIT(MUCOUNTEREN)

寄存器用于控制 mcycle和 minstret的计数

mtime

来自：Nuclei_N级别指令架构手册.pdf

CSR实现

表 7-20 mtvt2 寄存器各控制位

域	位	描述
CMMON-CODE-ENTRY	31:2	在 mtvt2.MTVT2EN=1 时，此域决定 ECLIC 非向量模式中断 common-code 入口地址。
Reserved	1	未使用的域为常数 0
MTVT2EN	0	mtvt2 使能位： ■ 0：ECLIC 非向量模式中断 common-code 入口地址由 mtvec 决定 ■ 1： ECLIC 非向量模式中断 common-code 入口地址由 mtvt2.COMMON-CODE-ENTRY 决定

表 7-12 mcountinhibit 寄存器各控制位

域	位	描述
Reserved	31:3	未使用的域为常数 0
IR	2	IR 为 1 时 minstret 的计数被关闭
Reserved	1	未使用的域为常数 0
CY	0	CY 为 1 时 mcycle 的计数被关闭

表 7-18 mmisc_ctl 寄存器各控制位

域	位	描述
Reserved	31:10	未使用的域为常数 0
NMI_CAUSE_FFF	9	控制 mnvec 及 NMI 的 mcause.EXCCODE： ■ 0：mnvec 的值等于处理器 reset 后的 PC， NMI 的 mcause.EXCCODE 为 0x1，此为默认值。 ■ 1：mnvec 的值与 mtvec 一致，NMI 的 mcause.EXCCODE 为 0xfff
Reserved	8:7	未使用的域为常数 0
MISALIGN	6	控制内核是否支持 Misaligned Access 功能： ■ 0：Misaligned Access 功能关闭，Misaligned Access 操作会产生异常 ■ 1：Misaligned Access 功能开启，此为默认值 注意：此域只有配置了非对齐数据访问功能才有效，否则为常数 0
Reserved	5:4	未使用的域为常数 0
BPU	3	控制分支预测器是否开启： ■ 0：分支预测器关闭 ■ 1：分支预测器开启，此为默认值 注意：此域只有配置了分支预测器才有效，否则为常数 0
Reserved	2:0	未使用的域为常数 0

来自：Nuclei_N级别指令架构手册.pdf

CSR实现

target\riscv\cpu.h

```
typedef struct {
    const char *name;
    riscv_csr_predicate_fn predicate;
    riscv_csr_read_fn read;
    riscv_csr_write_fn write;
    riscv_csr_op_fn op;
} riscv_csr_operations;
```

target\riscv\cpu.h

```
struct CPURISCVState {
    target_ulong gpr[32];
    uint64_t fpr[32];
……
    target_ulong mmisc_ctl;
    target_ulong mtvt;
    target_ulong mtvt2;
……
}
```

target\riscv\csr.c

```
/* Control and Status Register function table */
riscv_csr_operations csr_ops[CSR_TABLE_SIZE] = {
……
    [CSR_MTVT2] =               { "mtvt2", any,  read_mtvt2,       write_mtvt2       },
    [CSR_MTVT] =                { "mtvt", any,  read_mtvt,        write_mtvt        },
    [CSR_MMISC_CTL] =           { "mmisc_ctl", any,  read_mmisc_ctl,   write_mmisc_ctl   },
    [CSR_MUCOUNTEREN] =         { "mucounteren", any,  read_mucounteren, write_mucounteren },
……
}
```

```
static int read_mmisc_ctl(CPURISCVState *env, int csrno, target_ulong *val)
{
    *val = env->mmisc_ctl;
    return 0;
}

static int write_mmisc_ctl(CPURISCVState *env, int csrno, target_ulong val)
{
    env->mmisc_ctl = val;
    return 0;
}
……
```

CSR_* 宏定义 → CPU CSR变量定义 → csr_ops数组更新 → RWO函数实现

SysTimer(初步)实现

TIMER简介

计时器单元（Timer Unit TIMER），在 N级别处理器内核中主要用于产生计时器中断（ Timer Interrupt）和软件中断 Software Interrupt）。

来自：Nuclei_N级别指令架构手册.pdf 6.1

表 6-1 TIMER 寄存器的存储器映射地址

模块内偏移地址	读写属性	寄存器名称	复位默认值	功能描述
0x0	可读可写	mtime_lo	0x00000000	反映计时器 mtime 的低 32 位值，参见第 6.1.3 节了解其详细介绍。
0x4	可读可写	mtime_hi	0x00000000	反映计时器 mtime 的高 32 位值，参见第 6.1.3 节了解其详细介绍。
0x8	可读可写	mtimecmp_lo	0xFFFFFFFF	配置计时器的比较值 mtimecmp 低 32 位，参见第 6.1.5 节了解其详细介绍。
0xC	可读可写	mtimecmp_hi	0xFFFFFFFF	配置计时器的比较值 mtimecmp 高 32 位，参见第 6.1.5 节了解其详细介绍。
0xFF0	可读可写	msftrst	0x00000000	生成软件复位请求，参见第 6.1.7 节了解其详细介绍。
0xFF8	可读可写	mtimectl	0x00000000	控制计时器计数，参见第 6.1.4 节了解其详细介绍。
0xFFC	可读可写	msip	0x00000000	生成软件中断，参见第 6.1.6 节了解其详细介绍。

注意：

- TIMER 的寄存器只支持操作尺寸（Size）为 word 的对齐读写访问。
- TIMER 的寄存器区间为 0x0 ~ 0xFFF，除了上表中列出的寄存器之外的其他地址内的值为常数 0。

include\hw\intc\nuclei_systimer.h

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

    /*< public >*/
    MemoryRegion mmio;

    uint32_t mtime_lo;
    uint32_t mtime_hi;
    uint32_t mtimecmp_lo;
    uint32_t mtimecmp_hi;
    uint32_t msftrst;
    uint32_t mtimectl;
    uint32_t msip;
} NucLeiSYSTIMERState;
```

SysTimer(初步)实现

helloworld运行需求

```c
uint32_t measure_cpu_freq(uint32_t n)
{
    uint32_t start_mcycle, delta_mcycle;
    uint32_t start_mtime, delta_mtime;
    uint32_t mtime_freq = get_timer_freq();

    // Don't start measuruing until we see an mtime tick
    uint32_t tmp = (uint32_t)SysTimer_GetLoadValue();
    do {
        start_mtime = (uint32_t)SysTimer_GetLoadValue();
        start_mcycle = __RV_CSR_READ(CSR_MCYCLE);
    } while (start_mtime == tmp);

    do {
        delta_mtime = (uint32_t)SysTimer_GetLoadValue() - start_mtime;
        delta_mcycle = __RV_CSR_READ(CSR_MCYCLE) - start_mcycle;
    } while (delta_mtime < n);

    return (delta_mcycle / delta_mtime) * mtime_freq
           + ((delta_mcycle % delta_mtime) * mtime_freq) / delta_mtime;
}
```

```c
#define RTC_FREQ                        32768
// The TIMER frequency is just the RTC frequency
#define SOC_TIMER_FREQ                  RTC_FREQ
static uint32_t get_timer_freq()
{
    return SOC_TIMER_FREQ;
}
```

```c
typedef struct {
    __IOM uint64_t MTIMER;
    __IOM uint64_t MTIMERCMP;
    __IOM uint32_t RESERVED0[0x3F8];
    __IOM uint32_t MSFTRST;
    __IOM uint32_t RESERVED1;
    __IOM uint32_t MTIMECTL;
    __IOM uint32_t MSIP;
} SysTimer_Type;

#define SysTimer        ((SysTimer_Type *) SysTimer_BASE)
__STATIC_FORCEINLINE uint64_t SysTimer_GetLoadValue(void)
{
    return SysTimer->MTIMER;
}
```

SysTimer_BASE: offset 0 offset 4

mcycle :周期计数器的低32位(Lower 32 bits of Cycle counter)

- SysTimer_GetLoadValue 正常更新数据
- mcycle 运行正常
- cpu_freq 计算正确

```c
get_cpu_freq
measure_cpu_freq(1);
measure_cpu_freq(100);
```

SysTimer(初步)实现

QOM静态注册：

```c
static void nuclei_timer_class_init(ObjectClass *klass, void *da
{
    ……
    dc->realize = nuclei_timer_realize;
    dc->reset = nuclei_timer_reset;
    device_class_set_props(dc, nuclei_systimer_properties);
    ……
}

static const TypeInfo nuclei_timer_info = {
    .name = TYPE_NUCLEI_SYSTIMER,
    .parent = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(NucLeiSYSTIMERState),
    .instance_init = nuclei_timer_instance_init,
    .class_init = nuclei_timer_class_init,
};

static void nuclei_timer_register_types(void)
{
    type_register_static(&nuclei_timer_info);
}
type_init(nuclei_timer_register_types);
```

hw\intc\nuclei_systimer.c

include\exec\memory.h

```c
struct MemoryRegionOps {
    /* Read from the memory region. @addr is relative to @mr; @size is
     * in bytes. */
    uint64_t (*read)(void *opaque,
                     hwaddr addr,
                     unsigned size);
    /* Write to the memory region. @addr is relative to @mr; @size is
     * in bytes. */
    void (*write)(void *opaque,
                  hwaddr addr,
                  uint64_t data,
                  unsigned size);
}
```

```c
static const MemoryRegionOps nuclei_timer_ops = {
    .read = nuclei_timer_read,
    .write = nuclei_timer_write,
    .endianness = DEVICE_LITTLE_ENDIAN,
……
};

static void nuclei_timer_instance_init(Object *obj)
{
    NucLeiSYSTIMERState *s = NUCLEI_SYSTIMER(obj);

    memory_region_init_io(&s->mmio, obj, &nuclei_timer_ops,
                          s,TYPE_NUCLEI_SYSTIMER, s->aperture_size);
    sysbus_init_mmio(SYS_BUS_DEVICE(obj), &s->mmio);
}
```

MemoryRegionOps注册

SysTimer(初步)实现

- SysTimer_GetLoadValue 正常更新数据
- cpu_freq 计算正确
- mcycle 运行正常√

CSR mcycle读取：

```
static int read_instret(CPURISCVState *env,
           int csrno, target_ulong *val)
{
#if !defined(CONFIG_USER_ONLY)
    if (icount_enabled()) {
        *val = icount_get();
    } else {
        *val = cpu_get_host_ticks();
    }
#else
    *val = cpu_get_host_ticks();
#endif
    return 0;
}
```

```
static uint64_t nuclei_timer_read(void *opaque, hwaddr offset,unsigned size)
{
    ……
    switch (offset) {
    case NUCLEI_SYSTIMER_REG_MTIMELO:
        value = cpu_riscv_read_rtc(s->timebase_freq);
        s->mtime_lo =  value & 0xffffffff;
        s->mtime_hi = (value >> 32) & 0xffffffff;
        value = s->mtime_lo;
        break;
    case NUCLEI_SYSTIMER_REG_MTIMEHI:
        value =  s->mtime_hi;
        break;
    ……
    }
    return value;
}
static void nuclei_timer_write(void *opaque, hwaddr offset,uint64_t value, unsigned size)
{
    ……
    switch (offset) {
    case NUCLEI_SYSTIMER_REG_MTIMELO:
        s->mtime_lo = value;
        env->timer->expire_time |= (value &0xFFFFFFFF);
        break;
    case NUCLEI_SYSTIMER_REG_MTIMEHI:
        s->mtime_hi = value;
        env->timer->expire_time |= ((value << 32)&0xFFFFFFFF);
        break;
        ……
    }
}
```

```
#define NUCLEI_SYSTIMER_REG_MTIMELO 0x0000
#define NUCLEI_SYSTIMER_REG_MTIMEHI 0x0004
```

SysTimer(初步)实现

hw\intc\nuclei_systimer.c

```
static uint64_t cpu_riscv_read_rtc(uint64_t timebase_freq)
{
    return muldiv64(qemu_clock_get_ns(QEMU_CLOCK_VIRTUAL),
        timebase_freq, NANOSECONDS_PER_SECOND);
}
```

```
/* compute with 96 bit intermediate result: (a*b)/c */
static inline uint64_t muldiv64(uint64_t a, uint32_t b, uint32_t c)
```

```
typedef enum {
    QEMU_CLOCK_REALTIME = 0,
    QEMU_CLOCK_VIRTUAL = 1,
    QEMU_CLOCK_HOST = 2,
    QEMU_CLOCK_VIRTUAL_RT = 3,
    QEMU_CLOCK_MAX
} QEMUClockType;
```

QEMU_CLOCK_REALTIME: Real time clock
QEMU_CLOCK_VIRTUAL: virtual clock
QEMU_CLOCK_HOST: host clock
QEMU_CLOCK_VIRTUAL_RT: realtime clock used for icount warp

util\qemu-timer.c

```
int64_t qemu_clock_get_ns(QEMUClockType type)
{
    switch (type) {
    case QEMU_CLOCK_REALTIME:
        return get_clock();
    default:
    case QEMU_CLOCK_VIRTUAL:
        return cpus_get_virtual_clock();
    case QEMU_CLOCK_HOST:
        return REPLAY_CLOCK(REPLAY_CLOCK_HOST, get_clock_realtime());
    case QEMU_CLOCK_VIRTUAL_RT:
        return REPLAY_CLOCK(REPLAY_CLOCK_VIRTUAL_RT, cpu_get_clock());
    }
}
```

SysTimer(初步)实现

Machine中添加使用：

include\hw\riscv\nuclei_n.h

```
typedef struct NucLeiNSoCState {
    /*< private >*/
    DeviceState parent_obj;

    /*< public >*/
    RISCVHartArrayState cpus;

    NucLeiSYSTIMERState timer;
    ……
} NucLeiNSoCState;
```

```
static void nuclei_n_soc_instance_init(Object *obj)
{
    ……
    object_initialize_child(obj, "systimer", &s->timer,
                            TYPE_NUCLEI_SYSTIMER);
    ……
}
```

```
static void nuclei_n_soc_realize(DeviceState *dev, Error **errp)
{
    ……
    /* SysTimer */
    object_property_set_int(OBJECT(&s->timer), "aperture-size",
                            memmap[NUCLEI_N_TIMER].size, &error_abort);
    if (!sysbus_realize(SYS_BUS_DEVICE(&s->timer), errp)) {
        return;
    }
    sysbus_mmio_map(SYS_BUS_DEVICE(&s->timer), 0, memmap[NUCLEI_N_TIMER].base);
}
```

```
static Property nuclei_systimer_properties[] = {
    DEFINE_PROP_UINT32("aperture-size", NucLeiSYSTIMERState, aperture_size, 0x1000),
    DEFINE_PROP_UINT32("timebase-freq", NucLeiSYSTIMERState, timebase_freq, NUCLEI_NUCLEI_TIMEBASE_FREQ),
    DEFINE_PROP_END_OF_LIST(),
};
```

Uart(初步)实现

来自：https://static.dev.sifive.com/SiFive-E300-platform-reference-manual-v1.0.1.pdf

UART Overview

The UART peripheral supports the following features:

- 8-N-1 and 8-N-2 formats: 8 data bits, no parity bit, 1 start bit, 1 or 2 stop bits
- 8-entry transmit and receive FIFO buffers with programmable watermark interrupts
- 16× Rx oversampling with 2/3 majority voting per bit

Address	Name	Description
0x000	txdata	Transmit data register
0x004	rxdata	Receive data register
0x008	txctrl	Transmit control register
0x00C	rxctrl	Receive control register
0x010	ie	UART interrupt enable
0x014	ip	UART Interrupt pending
0x018	div	Baud rate divisor

Table 12.1: Register offsets within UART memory map.

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

    /*< public >*/
    MemoryRegion mmio;
    CharBackend chr;

    uint8_t rx_fifo[8];
    unsigned int rx_fifo_len;

    uint32_t txdata;
    uint32_t rxdata;
    uint32_t txctrl;
    uint32_t rxctrl;
    uint32_t ie;
    uint32_t ip;
    uint32_t div;
} NucLeiUARTState;
```

Uart(初步)实现

QOM静态注册：

hw\char\nuclei_uart.c

```
static Property nuclei_uart_properties[] = {
    DEFINE_PROP_CHR("chardev", NucLeiUARTState, chr),
    DEFINE_PROP_END_OF_LIST(),
};

static void nuclei_uart_class_init(ObjectClass *klass, void *data)
{
    ……
    dc->realize = nuclei_uart_realize;
    device_class_set_props(dc, nuclei_uart_properties);
}

static const TypeInfo nuclei_uart_info = {
    .name = TYPE_NUCLEI_UART,
    .parent = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(NucLeiUARTState),
    .instance_init = nuclei_uart_instance_init,
    .class_init = nuclei_uart_class_init,
};

static void nuclei_uart_register_types(void)
{
    type_register_static(&nuclei_uart_info);
}
type_init(nuclei_uart_register_types);
```

```
static const MemoryRegionOps uart_ops = {
    .read = uart_read,
    .write = uart_write,
    ……
};
static void nuclei_uart_instance_init(Object *obj)
{
    ……
    memory_region_init_io(&s->mmio, NULL, &uart_ops, s,
                          TYPE_NUCLEI_UART, 0x1000);
    sysbus_init_mmio(SYS_BUS_DEVICE(obj), &s->mmio);
}
```

```
static void nuclei_uart_realize(DeviceState *dev, Error **errp)
{
    NucLeiUARTState *s = NUCLEI_UART(dev);

    qemu_chr_fe_set_handlers(&s->chr, uart_can_rx, uart_rx,
                             uart_event,uart_be_change, s, NULL, true);
}
```

chardev\char-fe.c

```
void qemu_chr_fe_set_handlers(CharBackend *b,
                              IOCanReadHandler *fd_can_read,
                              IOReadHandler *fd_read,
                              IOEventHandler *fd_event,
                              BackendChangeHandler *be_change,
                              void *opaque,
                              GMainContext *context,
                              bool set_open)
```

Uart(初步)实现

Machine中添加使用：

include\hw\riscv\nuclei_n.h

```
typedef struct NucLeiNSoCState {
    /*< private >*/
    DeviceState parent_obj;

    /*< public >*/
    RISCVHartArrayState cpus;

    NucLeiUARTState uart0;
    NucLeiUARTState uart1;
    ……
} NucLeiNSoCState;
```

```
static void nuclei_n_soc_instance_init(Object *obj)
{
    ……
    object_initialize_child(obj, "uart0", &s->uart0,
                            TYPE_NUCLEI_UART);
    object_initialize_child(obj, "uart1", &s->uart1,
                            TYPE_NUCLEI_UART);
    ……
}
```

```
static void nuclei_n_soc_realize(DeviceState *dev, Error **errp)
{
    /* UART 0~1 */
    qdev_prop_set_chr(DEVICE(&s->uart0), "chardev", serial_hd(0));
    if (!sysbus_realize(SYS_BUS_DEVICE(&s->uart0), errp)) {
        return;
    }
    sysbus_mmio_map(SYS_BUS_DEVICE(&s->uart0), 0,
                    memmap[NUCLEI_N_UART0].base);
}
```

```
Chardev *serial_hd(int i)
{
    assert(i >= 0);
    if (i < num_serial_hds) {
        return serial_hds[i];
    }
    return NULL;
}
```

Uart(初步)实现

```
$qemu-system-riscv32  \
-nographic -machine mcu_200t,msel=1 \
-kernel ./hbird/ilm/helloworld.elf \
-nodefaults -serial stdio \
-d unimp
```

qemu-options.hx

```
DEF("nodefaults", 0, QEMU_OPTION_nodefaults, \
    "-nodefaults     don't create default devices\n", QEMU_ARCH_ALL)
SRST
``-nodefaults``
    Don't create default devices. Normally, QEMU sets the default
    devices like serial port, parallel port, virtual console, monitor
    device, VGA adapter, floppy and CD-ROM drive and others. The
    ``-nodefaults`` option will disable all those default devices.
ERST
```

```
DEF("serial", HAS_ARG, QEMU_OPTION_serial, \
    "-serial dev     redirect the serial port to char device 'dev'\n",
    QEMU_ARCH_ALL)
SRST
``-serial dev``
    Redirect the virtual serial port to host character device dev. The
    default device is ``vc`` in graphical mode and ``stdio`` in non
    graphical mode.

``stdio``
        [Unix only] standard input/output
```

```
case QEMU_OPTION_serial:
     add_device_config(DEV_SERIAL, optarg);
     default_serial = 0;
     if (strncmp(optarg, "mon:", 4) == 0) {
         default_monitor = 0;
     }
break;
```

```
foreach_device_config(DEV_SERIAL, serial_parse)
```

```
static int serial_parse(const char *devname)
{
    ……
    serial_hds = g_renew(Chardev *, serial_hds, index + 1);
    serial_hds[index] = qemu_chr_new_mux_mon(label, devname, NULL);
    ……
    return 0;
}
```

```
qemu_chr_new_noreplay
```

```
qemu_chr_parse_compat
```

```
qemu_chr_new_from_opts
```

```
if (……
     strcmp(filename, "stdio")   == 0) {
     qemu_opt_set(opts, "backend", filename, &error_abort);
     return opts;
}
```

CHARDEV_BACKEND_KIND_STDIO

```
#define TYPE_CHARDEV_STDIO "chardev-stdio"
```

Uart(初步)实现

```
#define TYPE_CHARDEV "chardev"
OBJECT_DECLARE_TYPE(Chardev, ChardevClass, CHARDEV)

#define TYPE_CHARDEV_NULL "chardev-null"
#define TYPE_CHARDEV_MUX "chardev-mux"
#define TYPE_CHARDEV_RINGBUF "chardev-ringbuf"
#define TYPE_CHARDEV_PTY "chardev-pty"
#define TYPE_CHARDEV_CONSOLE "chardev-console"
#define TYPE_CHARDEV_STDIO "chardev-stdio"
#define TYPE_CHARDEV_PIPE "chardev-pipe"
#define TYPE_CHARDEV_MEMORY "chardev-memory"
#define TYPE_CHARDEV_PARALLEL "chardev-parallel"
#define TYPE_CHARDEV_FILE "chardev-file"
#define TYPE_CHARDEV_SERIAL "chardev-serial"
#define TYPE_CHARDEV_SOCKET "chardev-socket"
#define TYPE_CHARDEV_UDP "chardev-udp"
```

include\chardev\char-fe.h

```
struct CharBackend {
    Chardev *chr;
    IOEventHandler *chr_event;
    IOCanReadHandler *chr_can_read;
    IOReadHandler *chr_read;
    BackendChangeHandler *chr_be_change;
    void *opaque;
    int tag;
    int fe_open;
};
```

include\chardev\char.h

```
struct Chardev {
    Object parent_obj;

    QemuMutex chr_write_lock;
    CharBackend *be;
    char *label;
    char *filename;
    int logfd;
    int be_open;
    GSource *gsource;
    GMainContext *gcontext;
    DECLARE_BITMAP(features, QEMU_CHAR_FEATURE_LAST);
};
```

QOM注册

```
static const TypeInfo char_type_info = {
    .name = TYPE_CHARDEV,
    .parent = TYPE_OBJECT,
    .instance_size = sizeof(Chardev),
    .instance_init = char_init,
    .instance_finalize = char_finalize,
    .abstract = true,
    .class_size = sizeof(ChardevClass),
    .class_init = char_class_init,
};
```

Uart(初步)实现

```
#define TYPE_CHARDEV "chardev"
OBJECT_DECLARE_TYPE(Chardev, ChardevClass, CHARDEV)

#define TYPE_CHARDEV_NULL "chardev-null"
#define TYPE_CHARDEV_MUX "chardev-mux"
#define TYPE_CHARDEV_RINGBUF "chardev-ringbuf"
#define TYPE_CHARDEV_PTY "chardev-pty"
#define TYPE_CHARDEV_CONSOLE "chardev-console"
#define TYPE_CHARDEV_STDIO "chardev-stdio"
#define TYPE_CHARDEV_PIPE "chardev-pipe"
#define TYPE_CHARDEV_MEMORY "chardev-memory"
#define TYPE_CHARDEV_PARALLEL "chardev-parallel"
#define TYPE_CHARDEV_FILE "chardev-file"
#define TYPE_CHARDEV_SERIAL "chardev-serial"
#define TYPE_CHARDEV_SOCKET "chardev-socket"
#define TYPE_CHARDEV_UDP "chardev-udp"
```

include\chardev\char.h

```
struct Chardev {
    Object parent_obj;

    QemuMutex chr_write_lock;
    CharBackend *be;
    char *label;
    char *filename;
    int logfd;
    int be_open;
    GSource *gsource;
    GMainContext *gcontext;
    DECLARE_BITMAP(features, QEMU_CHAR_FEATURE_LAST);
};
```

QOM注册：

```
static const TypeInfo char_type_info = {
    .name = TYPE_CHARDEV,
    .parent = TYPE_OBJECT,
    .instance_size = sizeof(Chardev),
    .instance_init = char_init,
    .instance_finalize = char_finalize,
    .abstract = true,
    .class_size = sizeof(ChardevClass),
    .class_init = char_class_init,
};
```

```
struct FDChardev {
    Chardev parent;

    QIOChannel *ioc_in, *ioc_out;
    int max_size;
};
```

```
static const TypeInfo char_stdio_type_info = {
    .name = TYPE_CHARDEV_STDIO,
#ifdef _WIN32
    .parent = TYPE_CHARDEV_WIN_STDIO,
#else
    .parent = TYPE_CHARDEV_FD,
#endif
    .instance_finalize = char_stdio_finalize,
    .class_init = char_stdio_class_init,
};
```

TYPE_OBJECT → TYPE_CHARDEV → TYPE_CHARDEV_FD → TYPE_CHARDEV_STDIO

Uart(初步)实现

include\chardev\char.h

```
struct ChardevClass {
    ObjectClass parent_class;

    bool internal; /* TODO: eventually use TYPE_USER_CREATABLE */
    void (*parse)(QemuOpts *opts, ChardevBackend *backend, Error **errp);
    void (*open)(Chardev *chr, ChardevBackend *backend,
                 bool *be_opened, Error **errp);
    int (*chr_write)(Chardev *s, const uint8_t *buf, int len);
    int (*chr_sync_read)(Chardev *s, const uint8_t *buf, int len);
    GSource *(*chr_add_watch)(Chardev *s, GIOCondition cond);
    void (*chr_update_read_handler)(Chardev *s);
    int (*chr_ioctl)(Chardev *s, int cmd, void *arg);
    int (*get_msgfds)(Chardev *s, int* fds, int num);
    int (*set_msgfds)(Chardev *s, int *fds, int num);
    int (*chr_add_client)(Chardev *chr, int fd);
    int (*chr_wait_connected)(Chardev *chr, Error **errp);
    void (*chr_disconnect)(Chardev *chr);
    void (*chr_accept_input)(Chardev *chr);
    void (*chr_set_echo)(Chardev *chr, bool echo);
    void (*chr_set_fe_open)(Chardev *chr, int fe_open);
    void (*chr_be_event)(Chardev *s, QEMUChrEvent event);
    void (*chr_options_parsed)(Chardev *chr);
};
```

chardev\char-fd.c

```
static void char_fd_class_init(ObjectClass *oc, void *data)
{
    ChardevClass *cc = CHARDEV_CLASS(oc);

    cc->chr_add_watch = fd_chr_add_watch;
    cc->chr_write = fd_chr_write;
    cc->chr_update_read_handler = fd_chr_update_read_handler;
}
```

FE封装接口：

- qemu_chr_fe_set_handlers
- qemu_chr_fe_set_handlers_full
- qemu_chr_fe_write
- qemu_chr_fe_accept_input
- qemu_chr_fe_ioctl

chardev\char-stdio.c

```
static void char_stdio_class_init(ObjectClass *oc,
                                  void *data)
{
    ChardevClass *cc = CHARDEV_CLASS(oc);

    cc->parse = qemu_chr_parse_stdio;
#ifndef _WIN32
    cc->open = qemu_chr_open_stdio;
    cc->chr_set_echo = qemu_chr_set_echo_stdio;
#endif
}
```

Uart(初步)实现

write函数实现：

```
static void
uart_write(void *opaque, hwaddr offset,
           uint64_t value, unsigned int size)
{
    ……
    unsigned char ch = value;

    switch (offset)
    {
    case NUCLEI_UART_REG_TXDATA:
        qemu_chr_fe_write(&s->chr, &ch, 1);
        //update_irq(s);
        break;
    ……
        break;
    default:
        break;
    }
}
```

read函数实现：

hw\char\nuclei_uart.c

```
static uint64_t
uart_read(void *opaque, hwaddr offset, unsigned int size)
{
    ……
    switch (offset)
    {
    case NUCLEI_UART_REG_TXDATA:
        return 0;
    case NUCLEI_UART_REG_RXDATA:
        if (s->rx_fifo_len)
        {
            fifo_val = 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 fifo_val;
        }
        return 0x80000000;
    }
    return value;
}
```

真正的读函数：

```
static int uart_can_rx(void *opaque)
{
    ……
    return s->rx_fifo_len < sizeof(s->rx_fifo);
}
```

```
static void uart_rx(void *opaque, const uint8_t *buf, int size)
{
    NucLeiUARTState *s = opaque;
    ……
    s->rx_fifo[s->rx_fifo_len++] = *buf;
    //update_irq(s);
}
```

运行测试

```
$qemu-system-riscv32  \
-nographic -machine mcu_200t,msel=1 \
-kernel ./hbird/ilm/helloworld.elf \
-nodefaults -serial stdio \
-d unimp
```

```
CSR_MTVEC: reserved mode not supported
riscv.nuclei.n.gpio: unimplemented device read  (size 4, offset 0x03c)
riscv.nuclei.n.gpio: unimplemented device write (size 4, offset 0x03c, value 0x00000000)
riscv.nuclei.n.gpio: unimplemented device read  (size 4, offset 0x038)
riscv.nuclei.n.gpio: unimplemented device write (size 4, offset 0x038, value 0x00030000)
Nuclei SDK Build Time: Apr 25 2021, 14:34:55
Download Mode: ILM
CPU Frequency 11855756 Hz
riscv.nuclei.n.eclic: unimplemented device write (size 1, offset 0x000b, value 0x00)
riscv.nuclei.n.eclic: unimplemented device read  (size 4, offset 0x0004)
riscv.nuclei.n.eclic: unimplemented device read  (size 1, offset 0x0000)
riscv.nuclei.n.eclic: unimplemented device write (size 1, offset 0x0000, value 0x00)
riscv.nuclei.n.eclic: unimplemented device read  (size 1, offset 0x0000)
riscv.nuclei.n.eclic: unimplemented device write (size 1, offset 0x0000, value 0x00)
MISA: 0x4010112d
MISA: RV32IMACFDU
0: Hello World From Nuclei RISC-V Processor!
1: Hello World From Nuclei RISC-V Processor!
2: Hello World From Nuclei RISC-V Processor!
3: Hello World From Nuclei RISC-V Processor!
4: Hello World From Nuclei RISC-V Processor!
5: Hello World From Nuclei RISC-V Processor!
6: Hello World From Nuclei RISC-V Processor!
7: Hello World From Nuclei RISC-V Processor!
8: Hello World From Nuclei RISC-V Processor!
9: Hello World From Nuclei RISC-V Processor!
10: Hello World From Nuclei RISC-V Processor!
11: Hello World From Nuclei RISC-V Processor!
12: Hello World From Nuclei RISC-V Processor!
13: Hello World From Nuclei RISC-V Processor!
14: Hello World From Nuclei RISC-V Processor!
15: Hello World From Nuclei RISC-V Processor!
16: Hello World From Nuclei RISC-V Processor!
17: Hello World From Nuclei RISC-V Processor!
18: Hello World From Nuclei RISC-V Processor!
19: Hello World From Nuclei RISC-V Processor!
```

下节课内容：

中断虚拟化

- QEMU RISCV IRQ中断介绍
- Timer与Clint
- Eclic与Clic

外设虚拟化

- Nuclei Eclic设备实现
- Nuclei Timer设备与中断
- Nuclei Uart设备与中断

谢谢

wangjunqiang@iscas.ac.cn