中国科学院软件研究所
Institute of Software Chinese Academy of Sciences
智能软件研究中心
Intelligent Software Research Center

QEMU 中 RISC-V SoC 的新增与实现

——基于NucLei SOC 在QEMU中的扩展

PLCT 王俊强

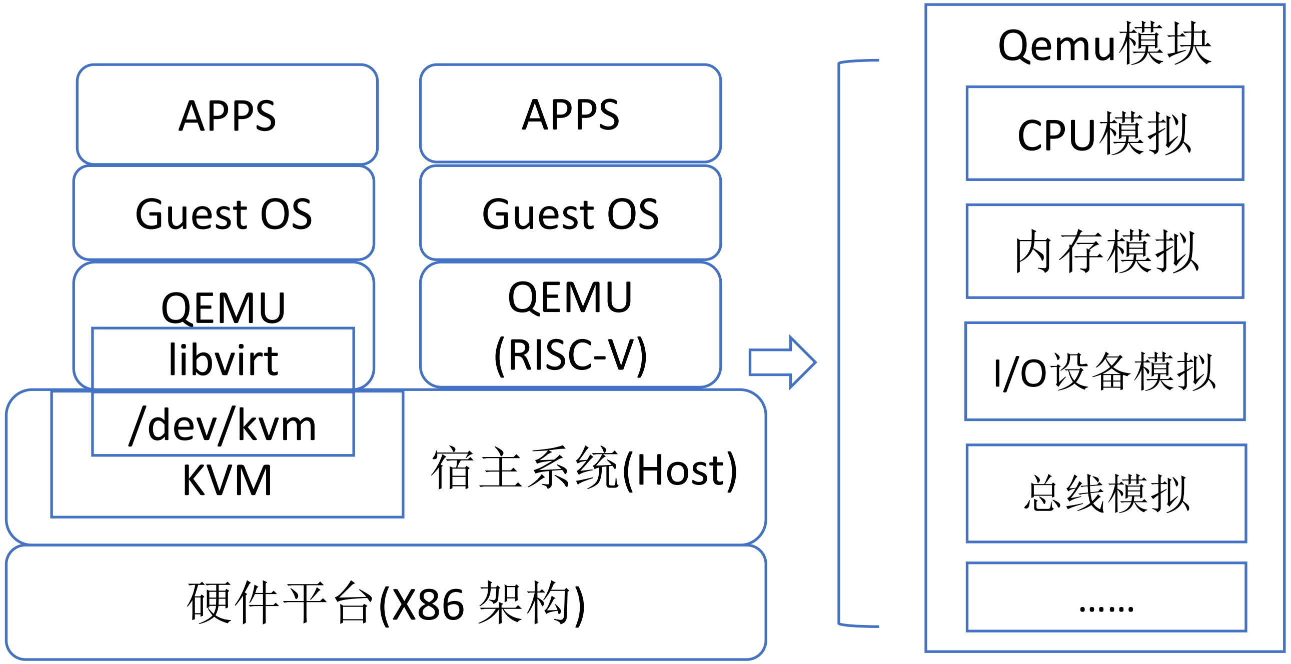
wangjunqiang@iscas.ac.cn

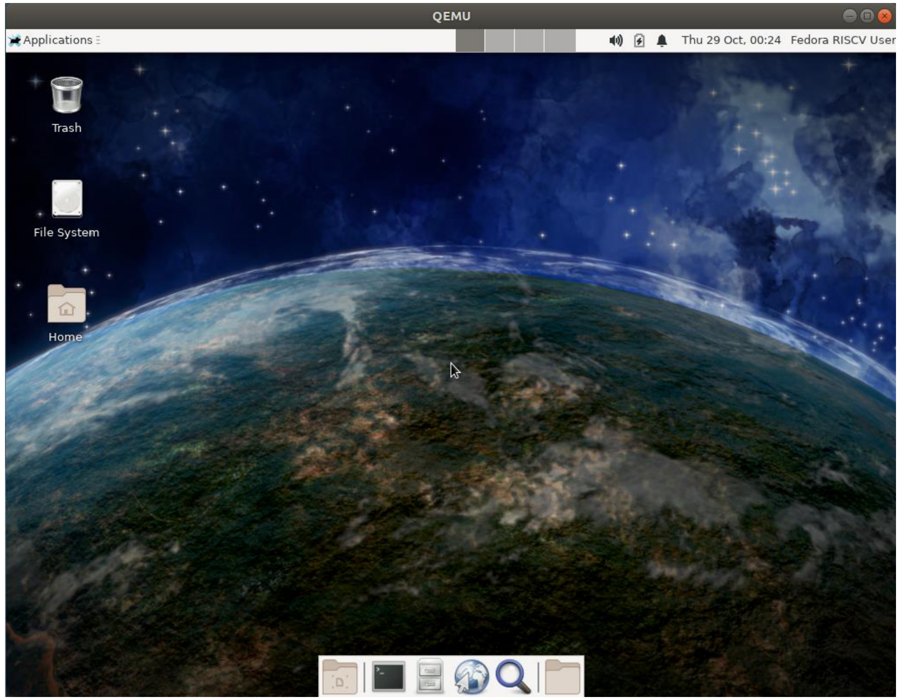
目录

- QEMU与RISC-V
- NucLei Soc简述
- QEMU中NucLei Soc的添加
- GD32V103F Board for RTOS
- HummingBird Board for Linux

QEMU RISC-V

- QEMU是一款开源的**模拟器**及**虚拟机监管器**(Virtual Machine Monitor, VMM)
- RISC-V是一个基于精简指令集（RISC）原则的**开源指令集架构**（ISA）





Fedora Desktop for RISC-V

QEMU RISC-V

QEMU对RISC-V支持

Features	support
Hardware Virtualization	soon* kvm-riscv
TCG Guest	Yes
MTTCG	Yes
TCG Host	No

Booting Linux	support
Debian	64
Fedora	64
OpenEmbedded	32/64
Buildroot	32/64

来自:Platforms Features

```
32bit Supported machines are:
none            empty machine
opentitan         RISC-V Board compatible with OpenTitan
sifive_e          RISC-V Board compatible with SiFive E SDK
sifive_u          RISC-V Board compatible with SiFive U SDK
spike            RISC-V Spike board (default)
virt           RISC-V VirtIO board
```

```
cpu type:
any
lowrisc-ibex
rv32
sifive-e31
sifive-e34
sifive-u34
```

```
64bit Supported machines are:
microchip-icicle-kit Microchip PolarFire SoC Icicle Kit
none            empty machine
sifive_e          RISC-V Board compatible with SiFive E SDK
sifive_u          RISC-V Board compatible with SiFive U SDK
spike            RISC-V Spike board (default)
virt           RISC-V VirtIO board
```

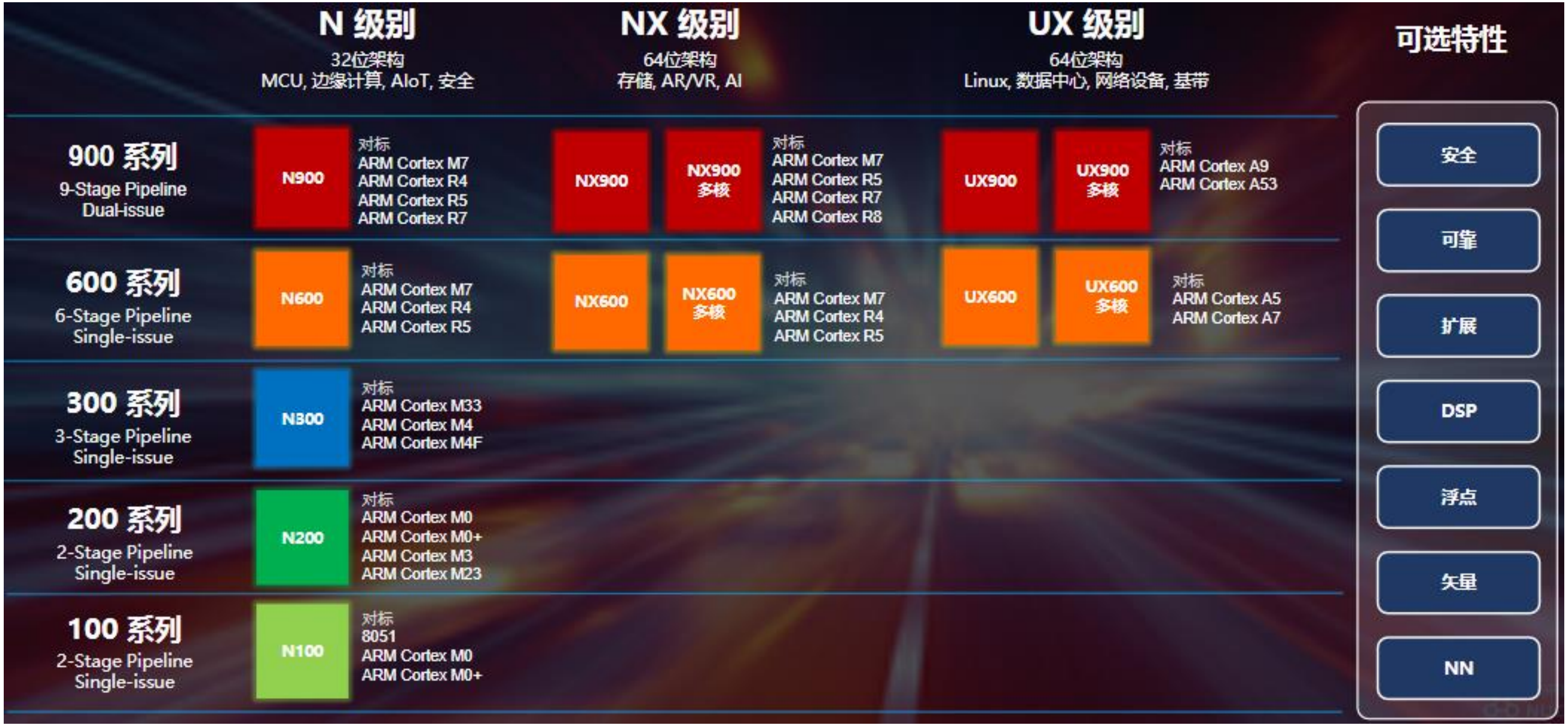
```
cpu type:
any
rv64
sifive-e51
sifive-u54
```

privileged 1.10.0
privileged 1.11.0
v extension 0.7.1

Extension	Version
I	2.1
E	2.1
M	2.0
A	2.1
F	2.2
D	2.2
V	0.7.1
C	2.0
H	0.3?
Counters	2.0
Zifencei	2.0
Zicsr	2.0

Extension支持(参考)

NucLei Soc简述



	N 级别 32位架构 MCU, 边缘计算, AIoT, 安全		NX 级别 64位架构 存储, AR/VR, AI			UX 级别 64位架构 Linux, 数据中心, 网络设备, 基带		
900 系列 9-Stage Pipeline Dual-issue	N900	对标 ARM Cortex M7 ARM Cortex R4 ARM Cortex R5 ARM Cortex R7	NX900	NX900 多核	对标 ARM Cortex M7 ARM Cortex R5 ARM Cortex R7 ARM Cortex R8	UX900	UX900 多核	对标 ARM Cortex A9 ARM Cortex A53
600 系列 6-Stage Pipeline Single-issue	N600	对标 ARM Cortex M7 ARM Cortex R4 ARM Cortex R5	NX600	NX600 多核	对标 ARM Cortex M7 ARM Cortex R4 ARM Cortex R5	UX600	UX600 多核	对标 ARM Cortex A5 ARM Cortex A7
300 系列 3-Stage Pipeline Single-issue	N300	对标 ARM Cortex M33 ARM Cortex M4 ARM Cortex M4F						
200 系列 2-Stage Pipeline Single-issue	N200	对标 ARM Cortex M0 ARM Cortex M0+ ARM Cortex M3 ARM Cortex M23						
100 系列 2-Stage Pipeline Single-issue	N100	对标 8051 ARM Cortex M0 ARM Cortex M0+						

可选特性

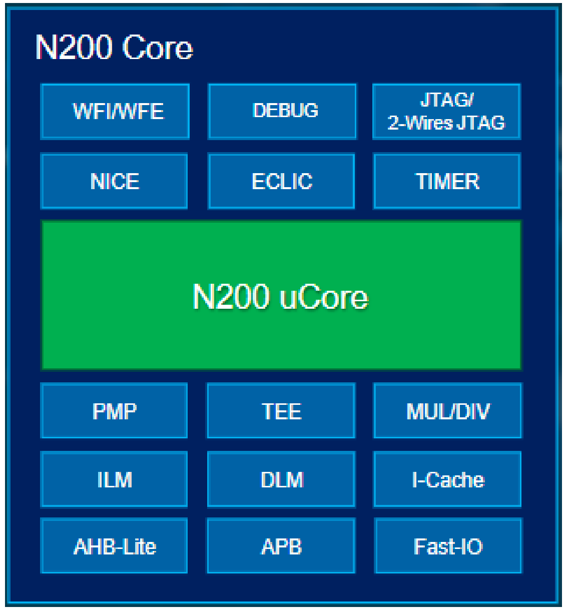
- 安全
- 可靠
- 扩展
- DSP
- 浮点
- 矢量
- NN

来自: www.nucleisys.com/product.php

NucLei Soc简述

SOC系列	RISC-V支持
N200	RV32I/E/M/A/C
N300	RV32I/E/M/A/C/F/D/P
N600	RV32I/M/A/C/F/D/P
NX600	RV64I/M/A/C/F/D/P
UX600(MMU)	RV64I/M/A/C/F/D/P
N900	RV32I/M/A/C/F/D/P/V
NX900	RV64I/M/A/C/F/D/P/V
UX900(MMU)	RV64I/M/A/C/F/D/P/V

- ➢工具链： RISC-V GNU Toolchain
- ➢软件环境—验证环境
  - ➢Nuclei SDK
  - ➢Nuclei Linux SDK
  - ➢RT-Thread Nuclei BSP
- ➢文档来源： Nuclei User Center



N200 Core

WFI/WFE | DEBUG | JTAG/ 2-Wires JTAG

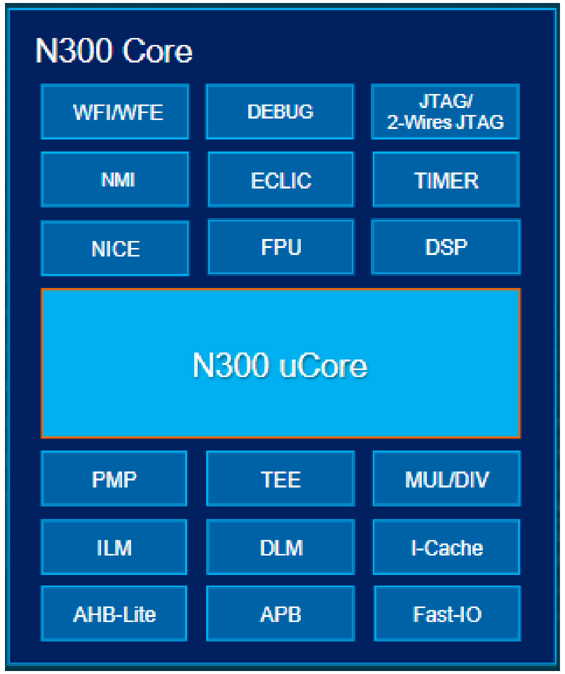
NICE | ECLIC | TIMER

N200 uCore

PMP | TEE | MUL/DIV

ILM | DLM | I-Cache

AHB-Lite | APB | Fast-IO



N300 Core

WFI/WFE | DEBUG | JTAG/ 2-Wires JTAG

NMI | ECLIC | TIMER

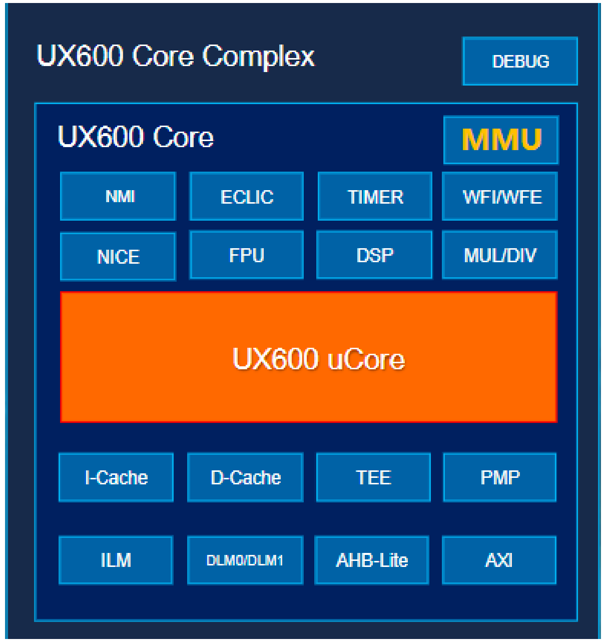
NICE | FPU | DSP

N300 uCore

PMP | TEE | MUL/DIV

ILM | DLM | I-Cache

AHB-Lite | APB | Fast-IO



UX600 Core Complex | DEBUG

UX600 Core | MMU

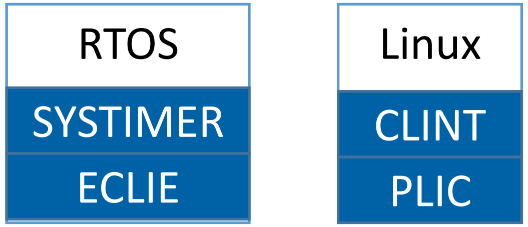
NMI | ECLIC | TIMER | WFI/WFE

NICE | FPU | DSP | MUL/DIV

UX600 uCore

I-Cache | D-Cache | TEE | PMP

ILM | DLM0/DLM1 | AHB-Lite | AXI

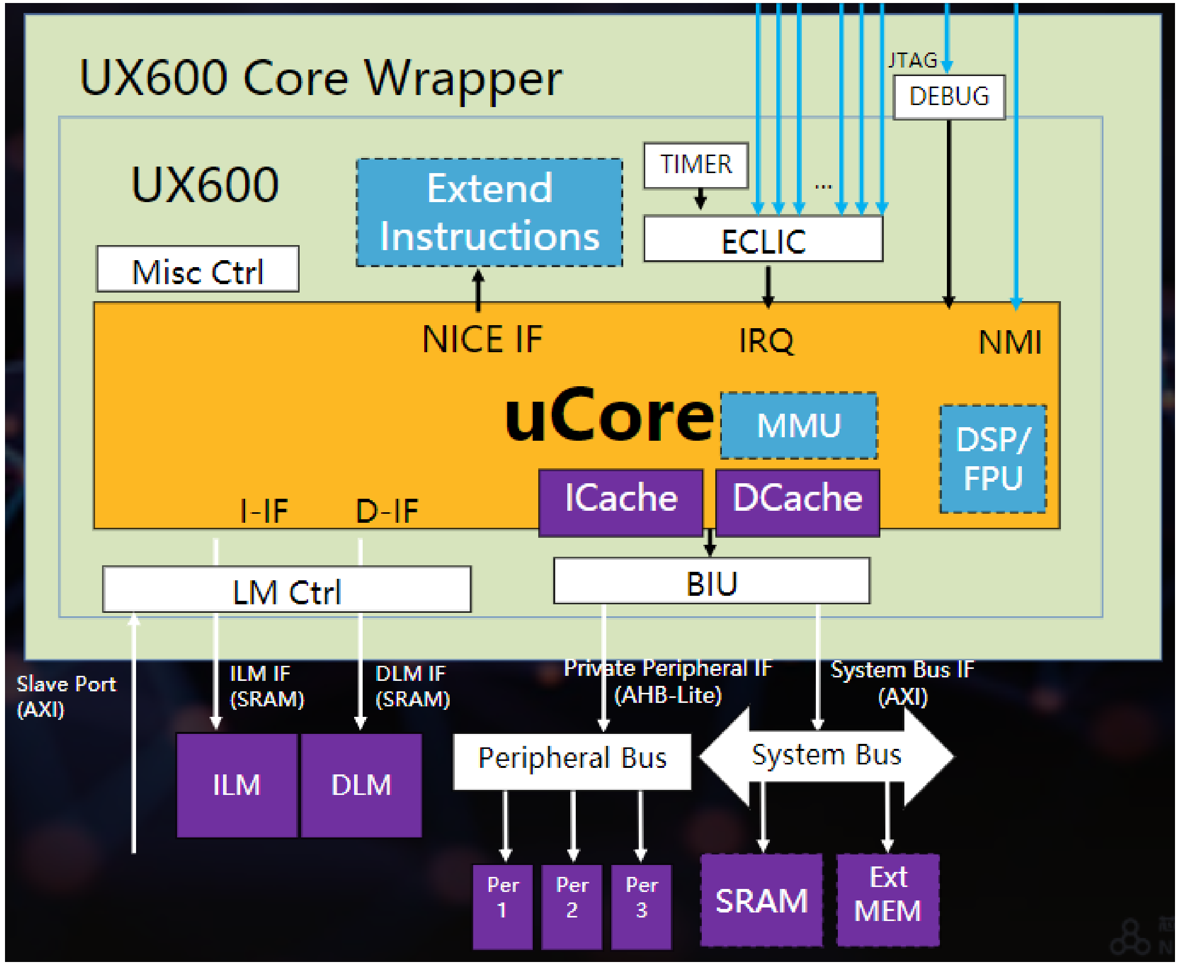


RTOS
SYSTIMER
ECLIE

Linux
CLINT
PLIC

QEMU中NucLei Soc的添加

- ➢CPU模拟
  - CPU Model
  - 指令与CSR
- ➢内存模拟
- ➢中断模拟
  - ECLIC
- ➢外设模拟
  - rcu
  - systimer
  - uart(示例)



UX600系统框架图示例

QEMU Object Model (QOM)

Features:

- System for dynamically registering types
- Support for single-inheritance of types
- Multiple inheritance of stateless interfaces

来自: QOM DOC

注册：

type_init(function)

module_init(function, type)

register_module_init(function, type);

QTAILQ_INSERT_TAIL(l, e, node);

```
typedef struct MyDevice
{
    DeviceState parent;
    int reg0, reg1, reg2;
} MyDevice;
```

```
typedef struct MyDeviceClass
{
    DeviceClass parent;
    void (*init) (MyDevice *obj);
} MyDeviceClass;
```

```
static void my_device_class_init(ObjectClass *oc, void *data)
{
}

static void my_device_init(Object *obj)
{
}
```

```
#define TYPE_MY_DEVICE "my-device"
static const TypeInfo my_device_info = {
    .parent = TYPE_DEVICE,
    .name = TYPE_MY_DEVICE,
    .instance_size = sizeof(MyDevice),
    .instance_init = my_device_init,
    .instance_finalize = my_device_finalize,
    .class_size = sizeof(MyDeviceClass),
    .class_init = my_device_class_init,
};
```

```
static void my_device_register_types(void)
{
    type_register_static(&my_device_info);
}
type_init(my_device_register_types)
```

```
DEFINE_TYPES(device_types_info)
```

CPU虚拟化

Task 1
CPU Name 建立

Supported Cores:
n201 n201e
n203 n203e
n205 n205e
n305
n307 n307fd
n600 n600f n600fd
nx600 nx600f nx600fd
ux600 ux600f ux600fd

Task 2
ADD Instructions

Instructions Extension:
V-Extension for 1.0.0
P-Extension
NICE(Nuclei Instruction Co-Unit Extension)

Task 3
ADD CSRs

CSR地址	读写属性	名称
0x307	MRW	mtvt
0x7eb	MRW	pushmsubm
0x7ec	MRW	mtvt2
0x7ed	MRW	jalmnxti
0x7ee	MRW	pushmcause
0x7ef	MRW	pushmepc
……		

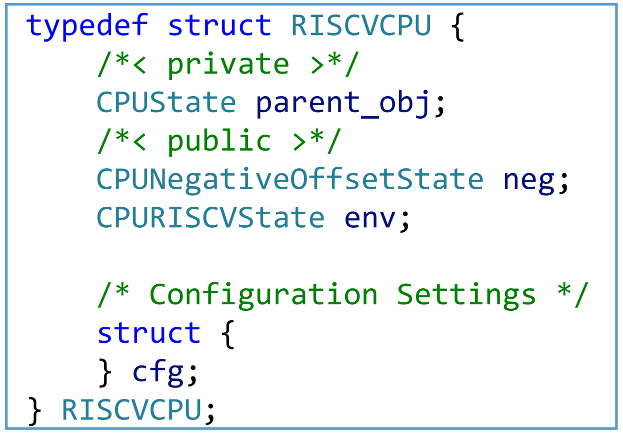
CPU模型

```
typedef struct RISCVCPU {
    /*< private >*/
    CPUState parent_obj;
    /*< public >*/
    CPUNegativeOffsetState neg;
    CPURISCVState env;

    /* Configuration Settings */
    struct {
    } cfg;
} RISCVCPU;
```

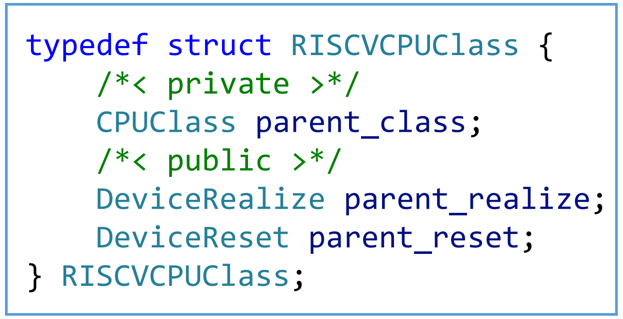
```
typedef struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
} RISCVCPUClass;
```

```
#define DEFINE_CPU(type_name, initfn)       \
    {                                       \
        .name = type_name,                  \
        .parent = TYPE_RISCV_CPU,           \
        .instance_init = initfn             \
    }

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),
    ……
#elif defined(TARGET_RISCV64)
    DEFINE_CPU(TYPE_RISCV_CPU_BASE64,           riscv_base64_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_U54,       rv64gcsu_priv1_10_0_cpu_init),
    ……
#endif
};

DEFINE_TYPES(riscv_cpu_type_infos)
```

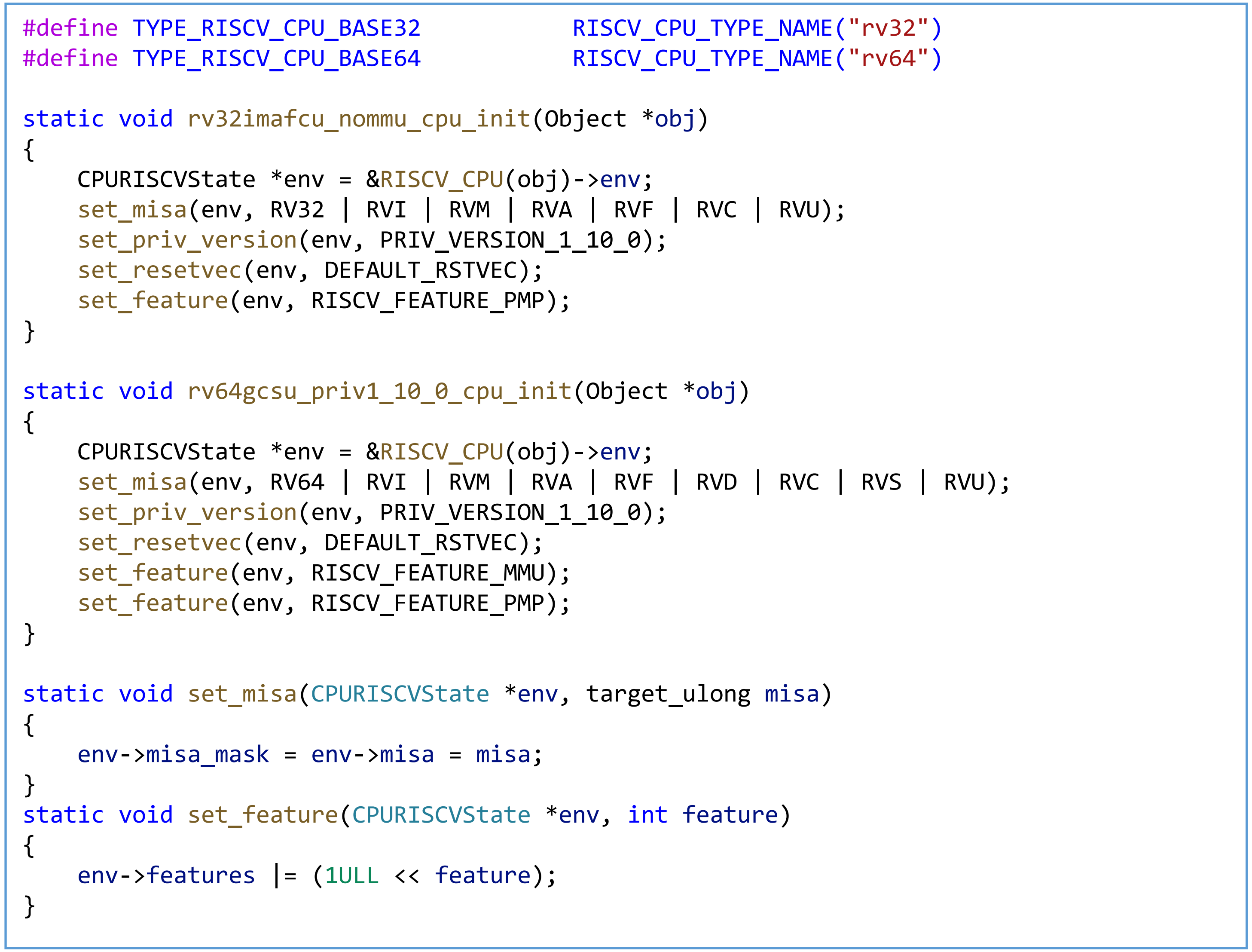
CPU模型



```
typedef struct RISCVCPU {
    /*< private >*/
    CPUState parent_obj;
    /*< public >*/
    CPUNegativeOffsetState neg;
    CPURISCVState env;

    /* Configuration Settings */
    struct {
    } cfg;
} RISCVCPU;
```



```
typedef struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
} RISCVCPUClass;
```



```
#define TYPE_RISCV_CPU_BASE32           RISCV_CPU_TYPE_NAME("rv32")
#define TYPE_RISCV_CPU_BASE64           RISCV_CPU_TYPE_NAME("rv64")

static void rv32imafcu_nommu_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV32 | RVI | RVM | RVA | RVF | RVC | RVU);
    set_priv_version(env, PRIV_VERSION_1_10_0);
    set_resetvec(env, DEFAULT_RSTVEC);
    set_feature(env, RISCV_FEATURE_PMP);
}

static void rv64gcsu_priv1_10_0_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV64 | 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 set_misa(CPURISCVState *env, target_ulong misa)
{
    env->misa_mask = env->misa = misa;
}
static void set_feature(CPURISCVState *env, int feature)
{
    env->features |= (1ULL << feature);
}
```

CPU模型

```
typedef struct RISCVCPU {
    /*< private >*/
    CPUState parent_obj;
    /*< public >*/
    CPUNegativeOffsetState neg;
    CPURISCVState env;

    /* Configuration Settings */
    struct {
    } cfg;
} RISCVCPU;
```

```
typedef struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
} RISCVCPUClass;
```

```
struct CPURISCVState {
    target_ulong gpr[32];
    uint64_t fpr[32]; /* assume both F and D extensions */
    target_ulong pc;

    target_ulong priv_ver;
    target_ulong misa;
    target_ulong misa_mask;
    ……
    uint32_t features;

    target_ulong mstatus;
    target_ulong mip;

#ifdef TARGET_RISCV32
    target_ulong mstatush;
#endif
    ……
    target_ulong mie;
    target_ulong mideleg;
    ……
    target_ulong sptbr;  /* until: priv-1.9.1 */
    target_ulong satp;   /* since: priv-1.10.0 */
    ……
    /* Fields from here on are preserved across CPU reset. */
    QEMUTimer *timer; /* Internal timer */
};
```

CPU模型

```
typedef struct RISCVCPU {
    /*< private >*/
    CPUState parent_obj;
    /*< public >*/
    CPUNegativeOffsetState neg;
    CPURISCVState env;

    /* Configuration Settings */
    struct {
    } cfg;
} RISCVCPU;
```

```
typedef struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
} RISCVCPUClass;
```

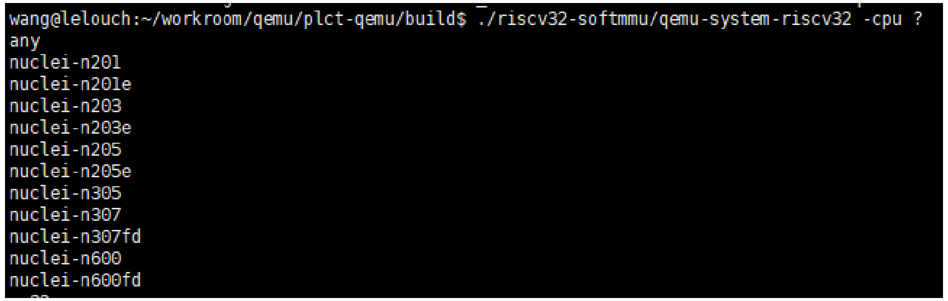
```
typedef struct CPUClass {
    /*< private >*/
    DeviceClass parent_class;
    /*< public >*/

    ObjectClass *(*class_by_name)(const char *cpu_model);
    void (*parse_features)(const char *typename, char *str, Error **errp);

    int reset_dump_flags;
    bool (*has_work)(CPUState *cpu);
    void (*do_interrupt)(CPUState *cpu);
    ……
    const VMStateDescription *vmsd;
    const char *gdb_core_xml_file;
    gchar * (*gdb_arch_name)(CPUState *cpu);
    const char * (*gdb_get_dynamic_xml)(CPUState *cpu, const char *xmlname);
    void (*cpu_exec_enter)(CPUState *cpu);
    void (*cpu_exec_exit)(CPUState *cpu);
    bool (*cpu_exec_interrupt)(CPUState *cpu, int interrupt_request);

    void (*disas_set_info)(CPUState *cpu, disassemble_info *info);
    vaddr (*adjust_watchpoint_address)(CPUState *cpu, vaddr addr, int len);
    void (*tcg_initialize)(void);

    /* Keep non-pointer data at the end to minimize holes.  */
    int gdb_num_core_regs;
    bool gdb_stop_before_watchpoint;
} CPUClass;
```

CPU模型

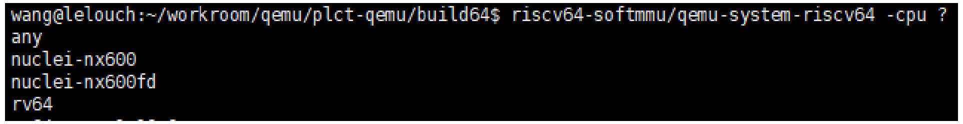
以N307FD为示例：

```
#define TYPE_RISCV_CPU_NUCLEI_N307FD    RISCV_CPU_TYPE_NAME("nuclei-n307fd")

static const TypeInfo riscv_cpu_type_infos[] = {
  ……
  DEFINE_CPU(TYPE_RISCV_CPU_NUCLEI_N307FD,    rv32imafdcu_nuclei_cpu_init),
  ……
};

static void rv32imafdcu_nuclei_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV32 | RVI | RVM | RVA | RVF | RVD | RVC | RVU);
    set_priv_version(env, PRIV_VERSION_1_10_0);
    set_resetvec(env, DEFAULT_RSTVEC);
    set_feature(env, RISCV_FEATURE_PMP);
}
```

运行结果：



```
wang@lelouch:~/workroom/qemu/plct-qemu/build$ ./riscv32-softmmu/qemu-system-riscv32 -cpu ?
any
nuclei-n201
nuclei-n201e
nuclei-n203
nuclei-n203e
nuclei-n205
nuclei-n205e
nuclei-n305
nuclei-n307
nuclei-n307fd
nuclei-n600
nuclei-n600fd
```

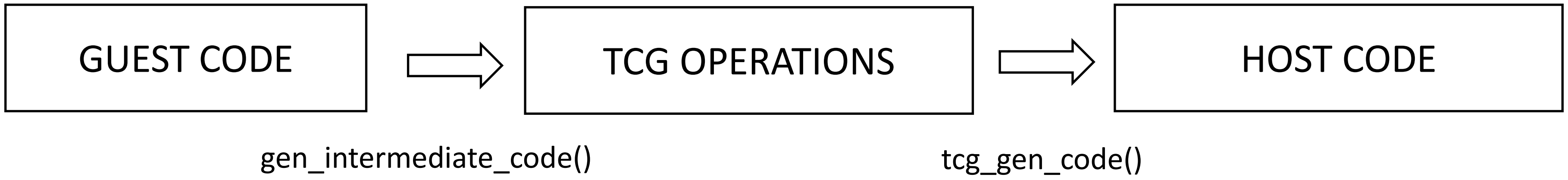


```
wang@lelouch:~/workroom/qemu/plct-qemu/build64$ riscv64-softmmu/qemu-system-riscv64 -cpu ?
any
nuclei-nx600
nuclei-nx600fd
rv64
```

RISC-V CPU指令扩展

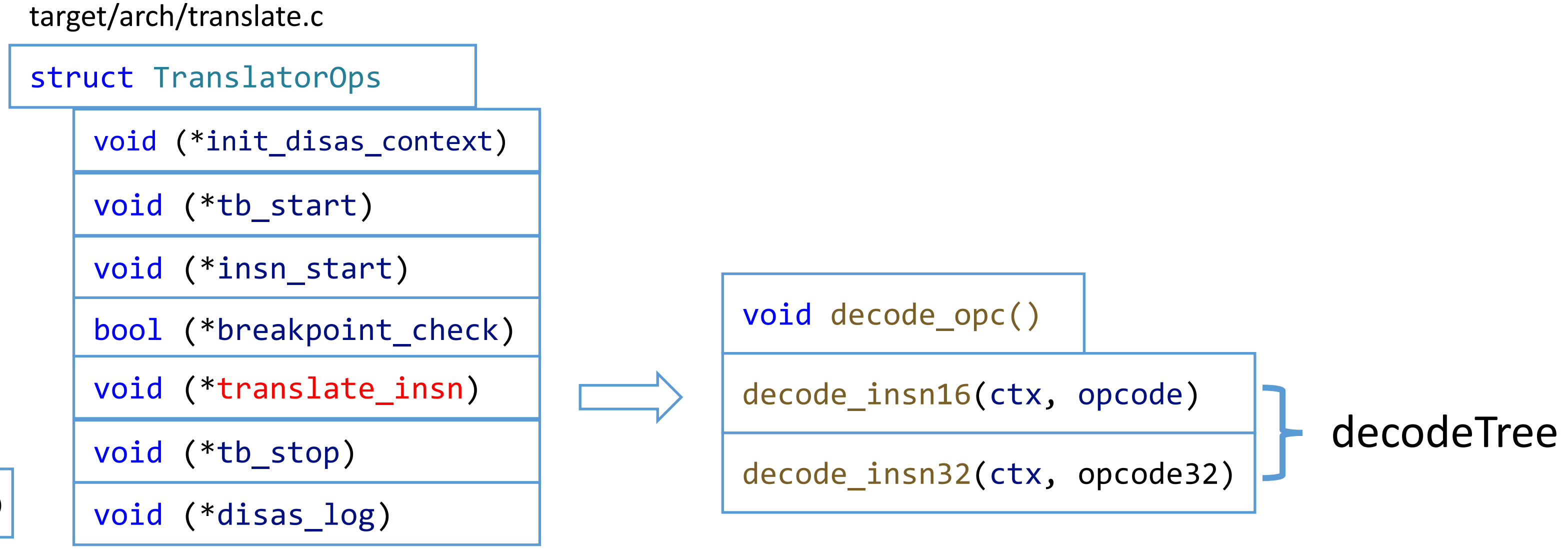
Tiny Code Generator（TCG)

将源处理器机器代码转换为虚拟机运行所需的机器代码块(如x86机器代码块)



主要调用栈:

```
cpu_loop(CPURISCVState *env)
cpu_exec(CPUState *cpu)
TranslationBlock *tb_find(……)
TranslationBlock *tb_gen_code(……)
void gen_intermediate_code(……)
translator_loop(const TranslatorOps *ops, ……)
tcg_gen_code(TCGContext *s, TranslationBlock *tb)
```


target/arch/translate.c

```
struct TranslatorOps
    void (*init_disas_context)
    void (*tb_start)
    void (*insn_start)
    bool (*breakpoint_check)
    void (*translate_insn)
    void (*tb_stop)
    void (*disas_log)
```

```
void decode_opc()
decode_insn16(ctx, opcode)
decode_insn32(ctx, opcode32)
```

RISC-V CPU指令扩展

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

ADDI Instructions

ADDI adds the sign-extended 12-bit immediate to register rs1. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result.

ADDI rd, rs1, 0 is used to implement the MV rd, rs1 assembler pseudoinstruction.

来自: Unprivileged ISA

addi rd, rs1, immediate
x[rd] = x[rs1] + sext(immediate)

```
typedef arg_i arg_addi;
static bool trans_addi(DisasContext *ctx, arg_addi *a);
```

target/riscv/insn32.decode ⟹ target/riscv/decode_insn32.inc.c

```
# Fields:
%rs1      15:5
%rd       7:5

# immediates:
%imm_i    20:s12

# Argument sets:
&i    imm rs1 rd

# Formats 32:
@i      ............    ..... ... ..... ....... &i     imm=%imm_i    %rs1 %rd

# *** RV32I Base Instruction Set ***
addi    ............    ..... 000 ..... 0010011 @i
```

```
static void decode_insn32_extract_i(D
isasContext *ctx, arg_i *a, uint32_t
insn)
{
    a->imm = sextract32(insn, 20, 12);
    a->rs1 = extract32(insn, 15, 5);
    a->rd = extract32(insn, 7, 5);
}
```

```
typedef struct
 {
    int imm;
    int rd;
    int rs1;
} arg_i;
```

Decodetree Specification

RISC-V CPU指令扩展

target/riscv/decode_insn32.inc.c

```c
static bool decode_insn32(DisasContext *ctx, uint32_t insn)
{
    switch (insn & 0x0000007f) {
        ……
        case 0x00000013:
            /* ........ ........ ........ .0010011 */
            switch ((insn >> 12) & 0x7) {
            case 0x0:
                /* ........ ........ .000.... .0010011 */
                decode_insn32_extract_i(ctx, &u.f_i, insn);
                if (trans_addi(ctx, &u.f_i)) return true;
                return false;
                ……
            }
        ……
    }
}
```

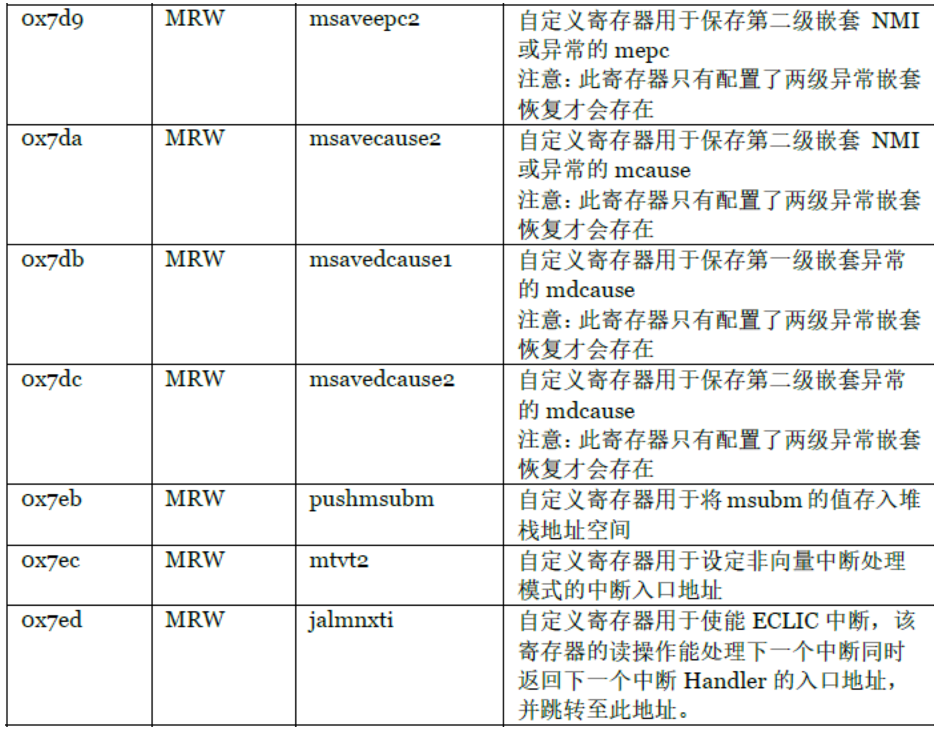
target/riscv/insn_trans/trans_rvi.inc.c

```c
static bool trans_addi(DisasContext *ctx, arg_addi *a)
{
    return gen_arith_imm_fn(ctx, a, &tcg_gen_addi_tl);
}

#define tcg_gen_addi_tl tcg_gen_addi_i32

void tcg_gen_addi_i32(TCGv_i32 ret, TCGv_i32 arg1, int32_t arg2)
{
    /* some cases can be optimized here */
    if (arg2 == 0) {
        tcg_gen_mov_i32(ret, arg1);
    } else {
        TCGv_i32 t0 = tcg_const_i32(arg2);
        tcg_gen_add_i32(ret, arg1, t0);
        tcg_temp_free_i32(t0);
    }
}
```

参考:
Backend Ops
Frontend Ops

Nuclei Soc CSR扩展

N级别处理器内核自定义CSR 部分:



0x7d9	MRW	msaveepc2	自定义寄存器用于保存第二级嵌套 NMI 或异常的 mepc 注意：此寄存器只有配置了两级异常嵌套恢复才会存在
0x7da	MRW	msavecause2	自定义寄存器用于保存第二级嵌套 NMI 或异常的 mcause 注意：此寄存器只有配置了两级异常嵌套恢复才会存在
0x7db	MRW	msavedcause1	自定义寄存器用于保存第一级嵌套异常的 mdcause 注意：此寄存器只有配置了两级异常嵌套恢复才会存在
0x7dc	MRW	msavedcause2	自定义寄存器用于保存第二级嵌套异常的 mdcause 注意：此寄存器只有配置了两级异常嵌套恢复才会存在
0x7eb	MRW	pushmsubm	自定义寄存器用于将 msubm 的值存入堆栈地址空间
0x7ec	MRW	mtvt2	自定义寄存器用于设定非向量中断处理模式的中断入口地址
0x7ed	MRW	jalmnxti	自定义寄存器用于使能 ECLIC 中断，该寄存器的读操作能处理下一个中断同时返回下一个中断 Handler 的入口地址，并跳转至此地址。

来自:《Nuclei_N级别指令架构手册》

```
* csrr   <->  riscv_csrrw(env, csrno, ret_value, 0, 0);
* csrrw  <->  riscv_csrrw(env, csrno, ret_value, value, -1);
* csrrs  <->  riscv_csrrw(env, csrno, ret_value, -1, value);
* csrrc  <->  riscv_csrrw(env, csrno, ret_value, 0, value);
```

target/riscv/insn32.decode

```
# Fields:
%rs1      15:5
%rd       7:5
%csr    20:12
# Formats 32:
@csr    ...........   .....  ... ..... .......            %csr    %rs1 %rd

# *** RV32I Base Instruction Set ***
csrrw    ...........    ..... 001 ..... 1110011 @csr
csrrs    ...........    ..... 010 ..... 1110011 @csr
csrrc    ...........    ..... 011 ..... 1110011 @csr
csrrwi   ...........    ..... 101 ..... 1110011 @csr
csrrsi   ...........    ..... 110 ..... 1110011 @csr
csrrci   ...........    ..... 111 ..... 1110011 @csr
```

```
static bool trans_csrrw(DisasContext *ctx, arg_csrrw *a)
static bool trans_csrrs(DisasContext *ctx, arg_csrrs *a)
static bool trans_csrrc(DisasContext *ctx, arg_csrrc *a)
static bool trans_csrrwi(DisasContext *ctx, arg_csrrwi *a)
static bool trans_csrrsi(DisasContext *ctx, arg_csrrsi *a)
static bool trans_csrrci(DisasContext *ctx, arg_csrrci *a)
```

```
gen_helper_csrrw
gen_helper_csrrs
gen_helper_csrrc
```

```
DEF_HELPER_3(csrrw, tl, env, tl, tl)
DEF_HELPER_4(csrrs, tl, env, tl, tl, tl)
DEF_HELPER_4(csrrc, tl, env, tl, tl, tl)
```

Nuclei Soc CSR扩展

target/riscv/cpu_bits.h

```
#define CSR_PUSHMSUBM      0x07eb
#define CSR_MTVT2      0x07ec
#define CSR_JALMNXTI      0x07ed
#define CSR_PUSHMCAUSE      0x07ee
#define CSR_PUSHMEPC      0x07ef
```

target/riscv/csr.c

```
/* Control and Status Register function table */
static riscv_csr_operations csr_ops[CSR_TABLE_SIZE] = {
    /* User Floating-Point CSRs */
    [CSR_FFLAGS] =              { fs,   read_fflags,      write_fflags      },
    [CSR_FRM] =                 { fs,   read_frm,         write_frm         },
    [CSR_FCSR] =                { fs,   read_fcsr,        write_fcsr        },

    /* User Timers and Counters */
    [CSR_CYCLE] =               { ctr,  read_instret                        },
    [CSR_INSTRET] =             { ctr,  read_instret                        },
    ……
};
```

struct riscv_csr_operations
riscv_csr_predicate_fn predicate
riscv_csr_read_fn read
riscv_csr_write_fn write
riscv_csr_op_fn op

CSR PUSHMCAUSE:

处理器 定义了通过 pushmcause 寄存器 csrrwi 操作实现的 CSR 指令，存储 mcause 的值到堆栈指针作为基地址的memory 空间

以如下指令为例介绍此CSR指令：

csrrwi x0，PUSHMCAUSE，1

该指令的操作是将mcause寄存器的值存到SP（堆栈指针）+1*4的地址。

```
static int rmw_pushmcause(CPURISCVState *env, int csrno, target_ulong *ret_value,
                          target_ulong new_value, target_ulong write_mask)
{
    uint64_t notify_addr = new_value * 4 + env->gpr[2];
    ……
    cpu_physical_memory_rw(notify_addr, &env->mcause,  4, 1);
    ……
    return 0;
}
```

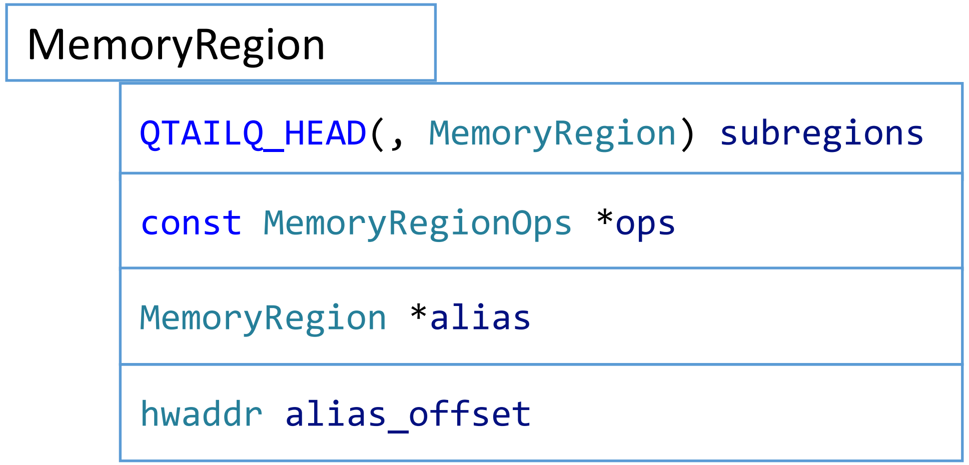
内存虚拟化

The memory API models the memory and I/O buses and controllers of a QEMU machine.

- ordinary RAM
- memory-mapped I/O (MMIO)
- memory controllers that can dynamically reroute physical memory regions to different destinations

来自: memory API

Types of regions	initialize
RAM	memory_region_init_ram()
MMIO	memory_region_init_io()
ROM	memory_region_init_rom().
ROM device	memory_region_init_rom_device()
IOMMU region	memory_region_init_iommu()
container	memory_region_init()
alias	memory_region_init_alias()
reservation region	memory_region_init_io()



MemoryRegion

```
QTAILQ_HEAD(, MemoryRegion) subregions
const MemoryRegionOps *ops
MemoryRegion *alias
hwaddr alias_offset
```

内存虚拟化

MemoryRegion ROOT — system_memory:

```
MemoryRegion *get_system_memory(void)
```

RAM:

```
void memory_region_init_ram(MemoryRegion *mr,
                            struct Object *owner,
                            const char *name,
                            uint64_t size,
                            Error **errp)
```

ROM:

```
void memory_region_init_rom(MemoryRegion *mr,
                            struct Object *owner,
                            const char *name,
                            uint64_t size,
                            Error **errp)
```

分配/挂载:

```
void memory_region_add_subregion(MemoryRegion *mr,
                                 hwaddr offset,
                                 MemoryRegion *subregion)
```

IO:

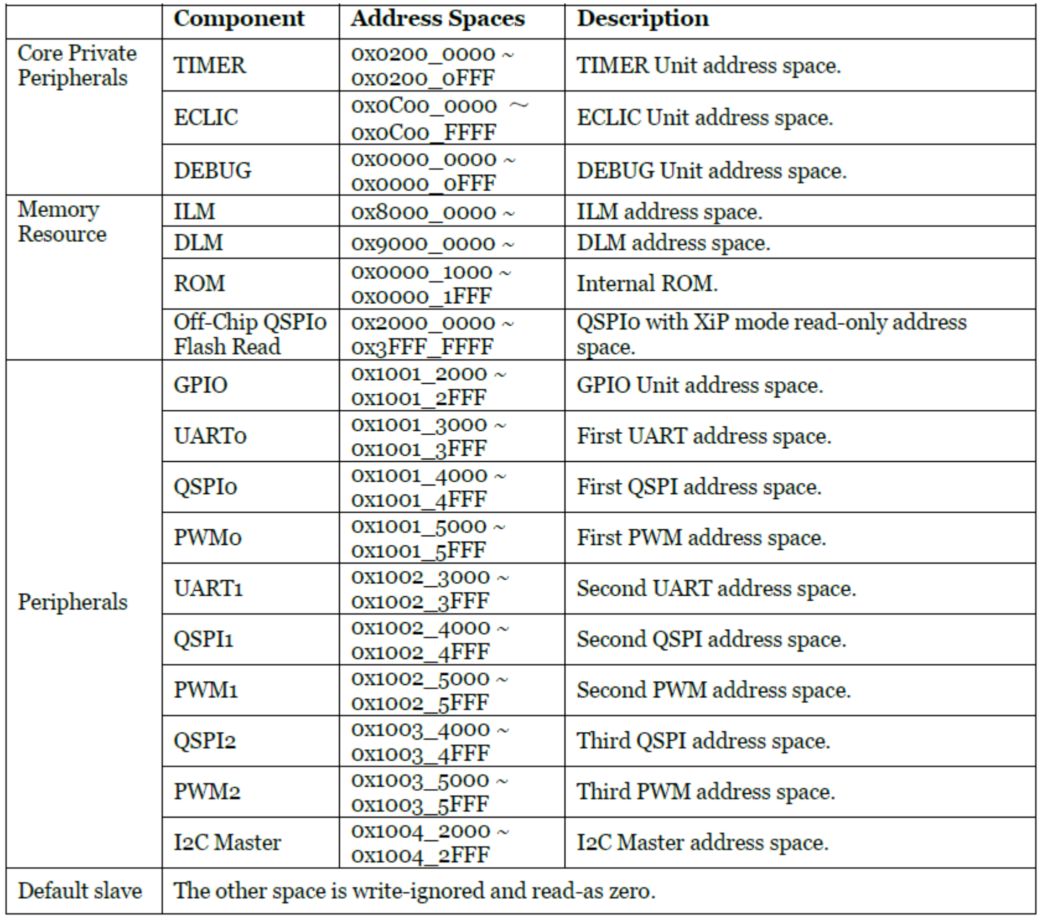
```
void memory_region_init_io(MemoryRegion *mr,
                           Object *owner,
                           const MemoryRegionOps *ops,
                           void *opaque,
                           const char *name,
                           uint64_t size)
void sysbus_init_mmio(SysBusDevice *dev, MemoryRegion *memory)
```

```
void sysbus_mmio_map(SysBusDevice *dev, int n, hwaddr addr)
```

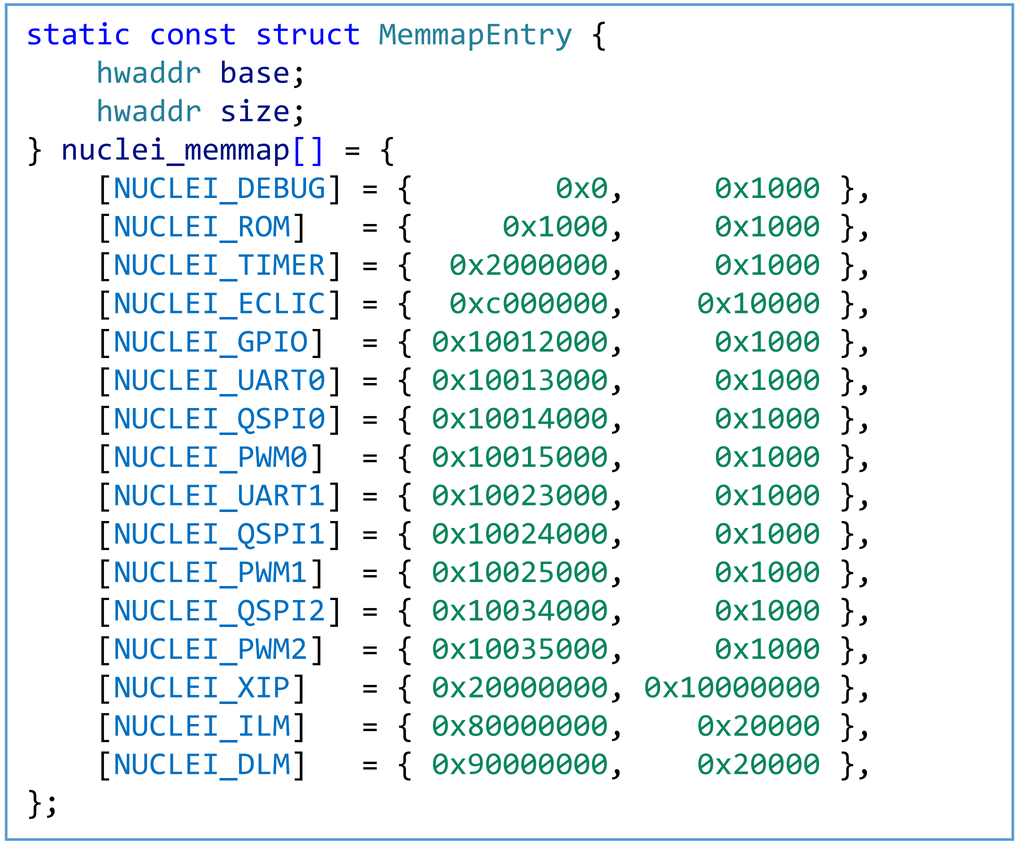
```
/*
 * Memory region callbacks
 */
struct MemoryRegionOps {
    /* Read from the memory region. @addr is relative to
@mr; @size is
     * in bytes. */
    uint64_t (*read)(void *opaque,
                     hwaddr addr,
                     unsigned size);
    /* Write to the memory region. @addr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

内存虚拟化

Table 5-1 Address Allocation of SoC



	Component	Address Spaces	Description
Core Private Peripherals	TIMER	0x0200_0000 ~ 0x0200_0FFF	TIMER Unit address space.
	ECLIC	0x0C00_0000 ~ 0x0C00_FFFF	ECLIC Unit address space.
	DEBUG	0x0000_0000 ~ 0x0000_0FFF	DEBUG Unit address space.
Memory Resource	ILM	0x8000_0000 ~	ILM address space.
	DLM	0x9000_0000 ~	DLM address space.
	ROM	0x0000_1000 ~ 0x0000_1FFF	Internal ROM.
	Off-Chip QSPI0 Flash Read	0x2000_0000 ~ 0x3FFF_FFFF	QSPI0 with XiP mode read-only address space.
Peripherals	GPIO	0x1001_2000 ~ 0x1001_2FFF	GPIO Unit address space.
	UART0	0x1001_3000 ~ 0x1001_3FFF	First UART address space.
	QSPI0	0x1001_4000 ~ 0x1001_4FFF	First QSPI address space.
	PWM0	0x1001_5000 ~ 0x1001_5FFF	First PWM address space.
	UART1	0x1002_3000 ~ 0x1002_3FFF	Second UART address space.
	QSPI1	0x1002_4000 ~ 0x1002_4FFF	Second QSPI address space.
	PWM1	0x1002_5000 ~ 0x1002_5FFF	Second PWM address space.
	QSPI2	0x1003_4000 ~ 0x1003_4FFF	Third QSPI address space.
	PWM2	0x1003_5000 ~ 0x1003_5FFF	Third PWM address space.
	I2C Master	0x1004_2000 ~ 0x1004_2FFF	I2C Master address space.
Default slave	The other space is write-ignored and read-as zero.		



```
static const struct MemmapEntry {
    hwaddr base;
    hwaddr size;
} nuclei_memmap[] = {
    [NUCLEI_DEBUG] = {        0x0,     0x1000 },
    [NUCLEI_ROM]   = {     0x1000,     0x1000 },
    [NUCLEI_TIMER] = {  0x2000000,     0x1000 },
    [NUCLEI_ECLIC] = {  0xc000000,    0x10000 },
    [NUCLEI_GPIO]  = { 0x10012000,     0x1000 },
    [NUCLEI_UART0] = { 0x10013000,     0x1000 },
    [NUCLEI_QSPI0] = { 0x10014000,     0x1000 },
    [NUCLEI_PWM0]  = { 0x10015000,     0x1000 },
    [NUCLEI_UART1] = { 0x10023000,     0x1000 },
    [NUCLEI_QSPI1] = { 0x10024000,     0x1000 },
    [NUCLEI_PWM1]  = { 0x10025000,     0x1000 },
    [NUCLEI_QSPI2] = { 0x10034000,     0x1000 },
    [NUCLEI_PWM2]  = { 0x10035000,     0x1000 },
    [NUCLEI_XIP]   = { 0x20000000, 0x10000000 },
    [NUCLEI_ILM]   = { 0x80000000,    0x20000 },
    [NUCLEI_DLM]   = { 0x90000000,    0x20000 },
};
```

内存虚拟化

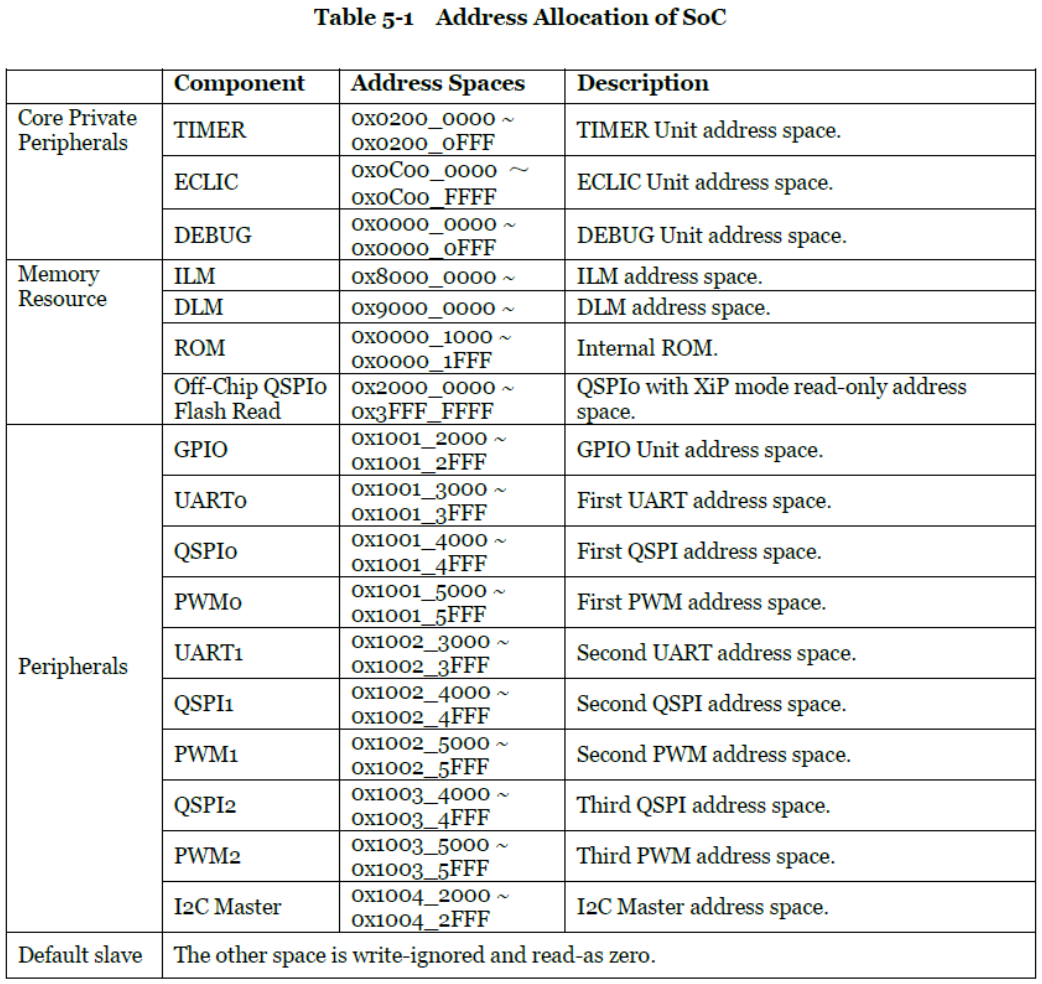


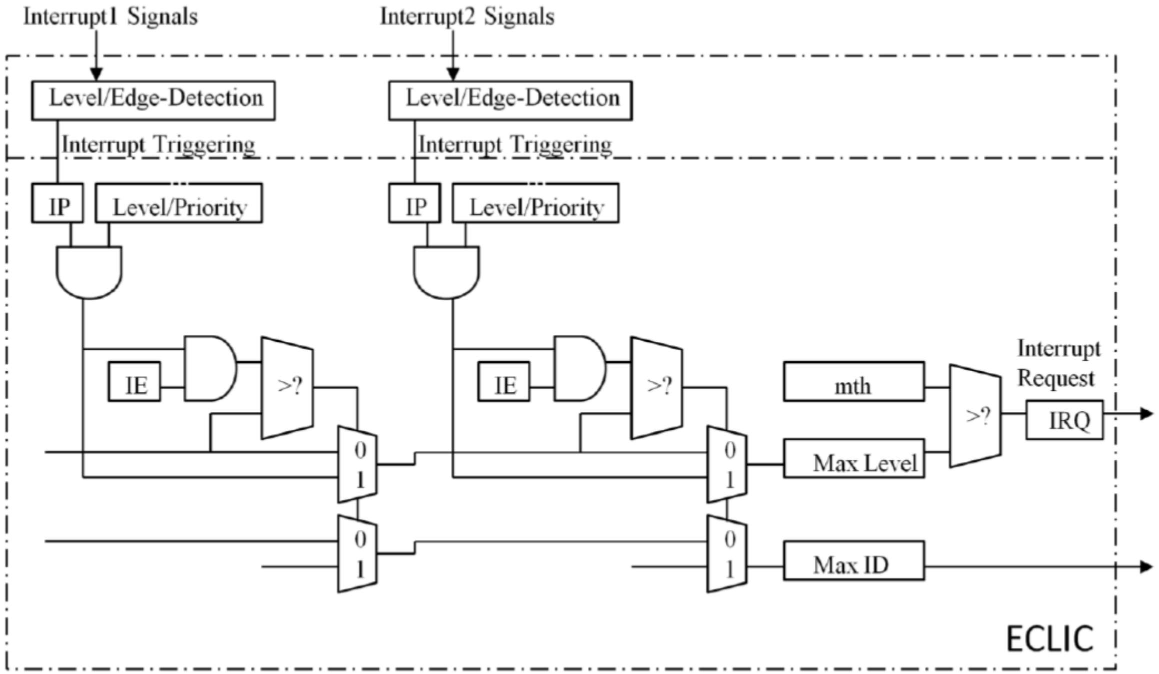
Table 5-1 Address Allocation of SoC

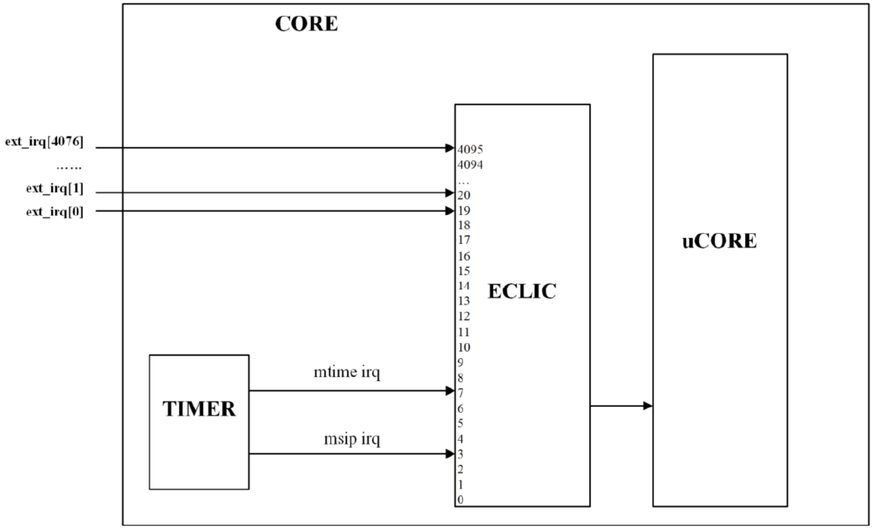
	Component	Address Spaces	Description
Core Private Peripherals	TIMER	0x0200_0000 ~ 0x0200_0FFF	TIMER Unit address space.
	ECLIC	0x0C00_0000 ~ 0x0C00_FFFF	ECLIC Unit address space.
	DEBUG	0x0000_0000 ~ 0x0000_0FFF	DEBUG Unit address space.
Memory Resource	ILM	0x8000_0000 ~	ILM address space.
	DLM	0x9000_0000 ~	DLM address space.
	ROM	0x0000_1000 ~ 0x0000_1FFF	Internal ROM.
	Off-Chip QSPI0 Flash Read	0x2000_0000 ~ 0x3FFF_FFFF	QSPI0 with XiP mode read-only address space.
Peripherals	GPIO	0x1001_2000 ~ 0x1001_2FFF	GPIO Unit address space.
	UART0	0x1001_3000 ~ 0x1001_3FFF	First UART address space.
	QSPI0	0x1001_4000 ~ 0x1001_4FFF	First QSPI address space.
	PWM0	0x1001_5000 ~ 0x1001_5FFF	First PWM address space.
	UART1	0x1002_3000 ~ 0x1002_3FFF	Second UART address space.
	QSPI1	0x1002_4000 ~ 0x1002_4FFF	Second QSPI address space.
	PWM1	0x1002_5000 ~ 0x1002_5FFF	Second PWM address space.
	QSPI2	0x1003_4000 ~ 0x1003_4FFF	Third QSPI address space.
	PWM2	0x1003_5000 ~ 0x1003_5FFF	Third PWM address space.
	I2C Master	0x1004_2000 ~ 0x1004_2FFF	I2C Master address space.
Default slave	The other space is write-ignored and read-as zero.		

```
ILM DLM示例:
MemoryRegion *sys_mem = get_system_memory();

memory_region_init_ram(&s->soc.ilm, NULL, "riscv.nuclei.ram.ilm",
                      memmap[NUCLEI_ILM].size, &error_fatal);
memory_region_add_subregion(sys_mem,
                      memmap[NUCLEI_ILM].base, &s->soc.ilm);

memory_region_init_ram(&s->soc.dlm, NULL, "riscv.nuclei.ram.dlm",
                      memmap[NUCLEI_DLM].size, &error_fatal);
memory_region_add_subregion(sys_mem,
                      memmap[NUCLEI_DLM].base, &s->soc.dlm);
/* Mask ROM */
memory_region_init_rom(&s>internal_rom, OBJECT(dev), "riscv.nuclei.
irom",memmap[NUCLEI_ROM].size, &error_fatal);
memory_region_add_subregion(sys_mem,
                      memmap[NUCLEI_ROM].base, &s->internal_rom);

hw/riscv/sifive_gpio.c
memory_region_init_io(&s->mmio, obj, &gpio_ops, s,
            TYPE_SIFIVE_GPIO, SIFIVE_GPIO_SIZE);
sysbus_init_mmio(SYS_BUS_DEVICE(obj), &s->mmio);

hw/riscv/nuclei_hbrid.c
object_property_set_bool(OBJECT(&s->gpio), true,"realized", &err);
sysbus_mmio_map(SYS_BUS_DEVICE(&s>gpio),0,memmap[NUCLEI_GPIO].base);
```

中断虚拟化

ECLIC:Enhanced Core Local Interrupt Controller



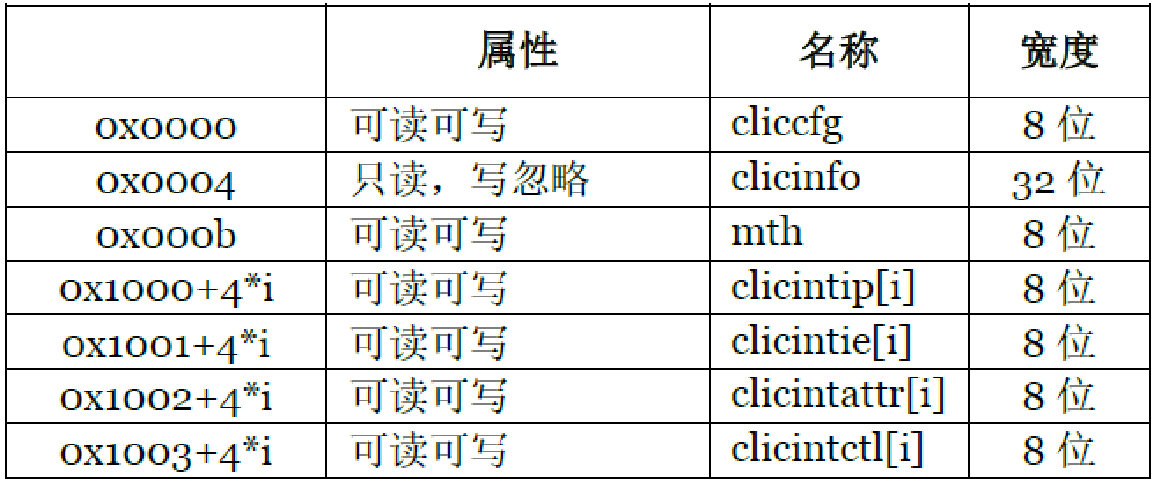


- ECLIC 只服务于一个处理器内核，为该处理器内核私有
- 兼容CLIC
- 支持4096个中断源
- 支持向量和非向量处理
- 支持中断响应、嵌套、咬尾
- ……

高志远[video][slides]

中断虚拟化

表 6-6 ECLIC 寄存器的单元内地址偏移量



	属性	名称	宽度
0x0000	可读可写	cliccfg	8 位
0x0004	只读，写忽略	clicinfo	32 位
0x000b	可读可写	mth	8 位
0x1000+4*i	可读可写	clicintip[i]	8 位
0x1001+4*i	可读可写	clicintie[i]	8 位
0x1002+4*i	可读可写	clicintattr[i]	8 位
0x1003+4*i	可读可写	clicintctl[i]	8 位

```
struct IRQState {
    Object parent_obj;

    qemu_irq_handler handler;
    void *opaque;
    int n;
};
typedef struct IRQState *qemu_irq;
```

```
static inline void cpu_interrupt(
CPUState *cpu, int mask)

void cpu_reset_interrupt(CPUState
 *cpu, int mask)
```

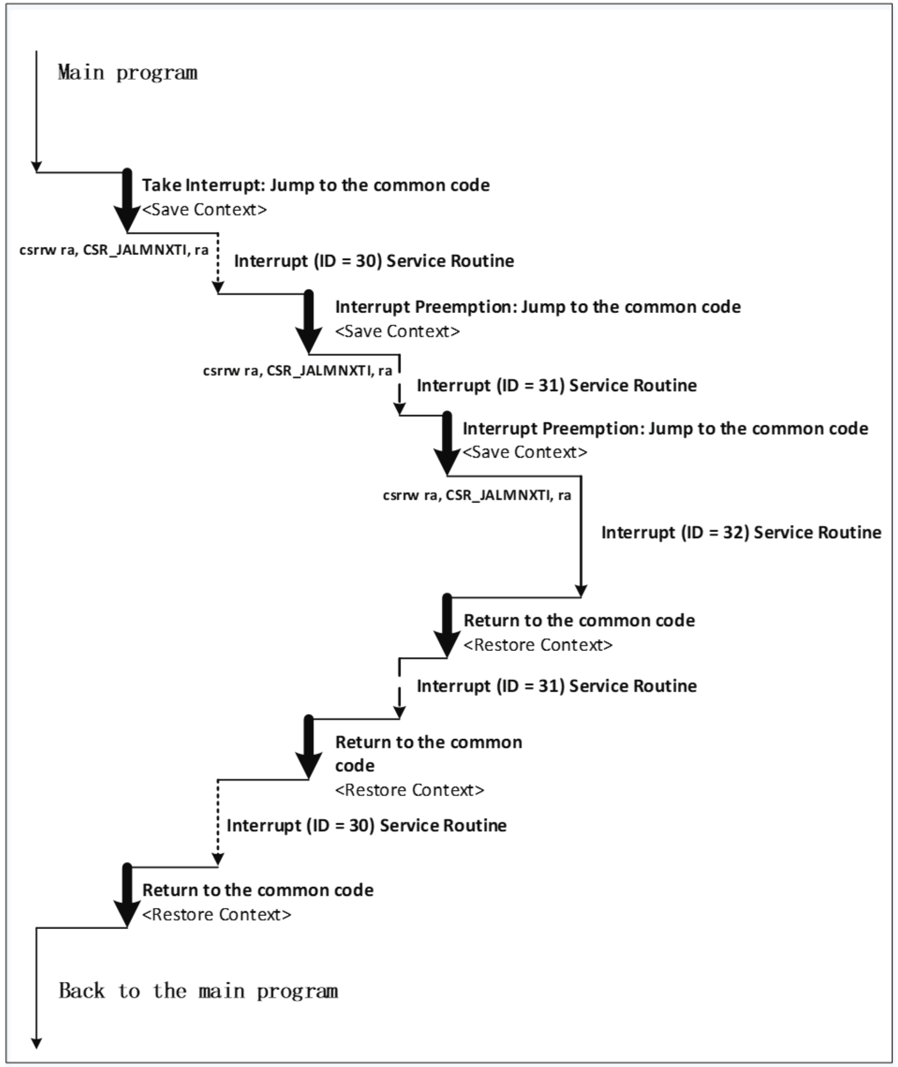
```
qemu_irq qemu_allocate_irq(qemu_irq_handler handler, void *opaque, int n)
```

```
void riscv_cpu_do_interrupt(CPUState *cs)
```

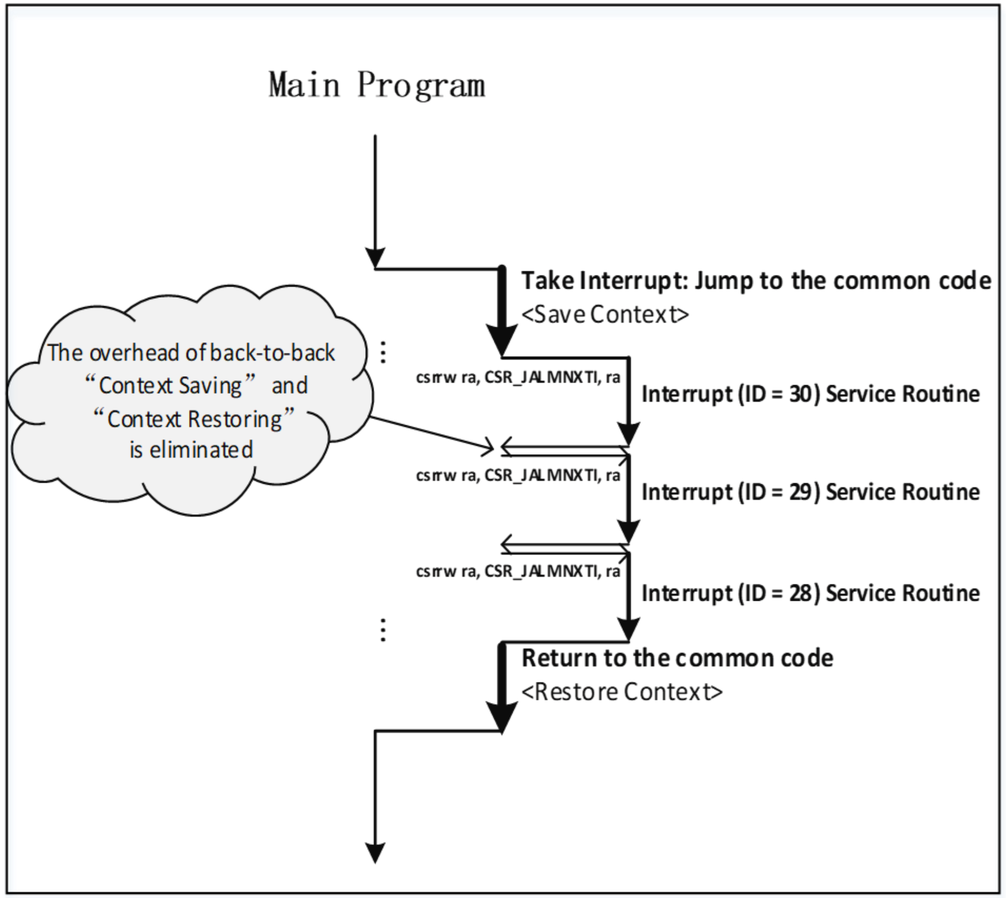
```
typedef struct NucLeiECLICState {
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
    MemoryRegion mmio;
    uint32_t num_sources;

    /* config */
    uint32_t sources_id;
    uint8_t  cliccfg;
    uint32_t clicinfo;
    uint8_t  mth;
    uint8_t  *clicintip;
    uint8_t  *clicintie;
    uint8_t  *clicintattr;
    uint8_t  *clicintctl;
    ECLICPendingInterrupt *clicintlist;
    uint32_t aperture_size;

    QLIST_HEAD(, ECLICPendingInterrupt) pending_list;
    size_t active_count;

    /* ECLIC IRQ handlers */
    qemu_irq *irqs;

} NucLeiECLICState;
```

```
void qemu_set_irq(qemu_irq irq, int level);
```

中断虚拟化(非向量化)


Main program
Take Interrupt: Jump to the common code
<Save Context>
csrrw ra, CSR_JALMNXTI, ra
Interrupt (ID = 30) Service Routine
Interrupt Preemption: Jump to the common code
<Save Context>
csrrw ra, CSR_JALMNXTI, ra
Interrupt (ID = 31) Service Routine
Interrupt Preemption: Jump to the common code
<Save Context>
csrrw ra, CSR_JALMNXTI, ra
Interrupt (ID = 32) Service Routine
Return to the common code
<Restore Context>
Interrupt (ID = 31) Service Routine
Return to the common code
<Restore Context>
Interrupt (ID = 30) Service Routine
Return to the common code
<Restore Context>
Back to the main program

嵌套流程


Main Program
Take Interrupt: Jump to the common code
<Save Context>
The overhead of back-to-back “Context Saving” and “Context Restoring” is eliminated
csrrw ra, CSR_JALMNXTI, ra
Interrupt (ID = 30) Service Routine
csrrw ra, CSR_JALMNXTI, ra
Interrupt (ID = 29) Service Routine
csrrw ra, CSR_JALMNXTI, ra
Interrupt (ID = 28) Service Routine
Return to the common code
<Restore Context>

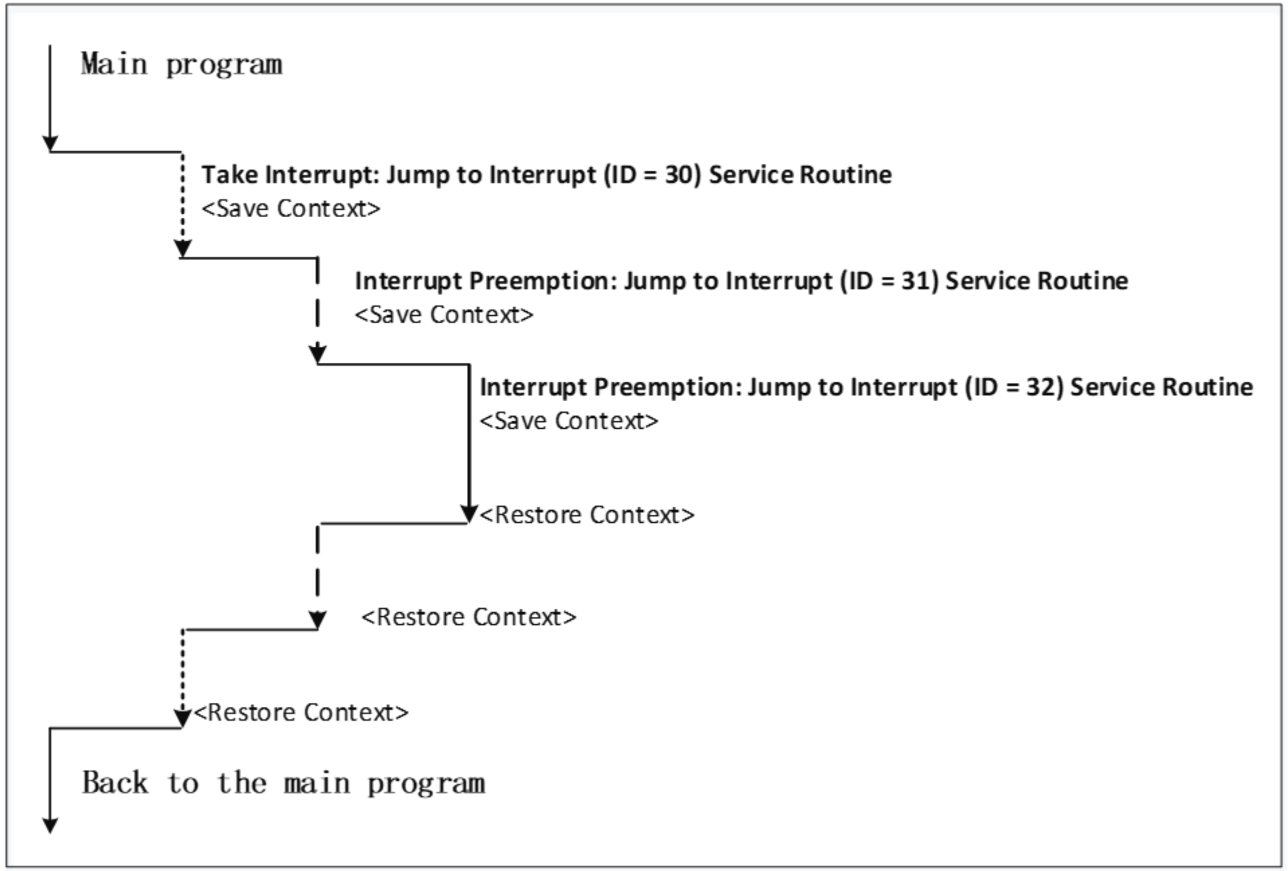
咬尾流程

irq_entry入口函数：

```
sw    ra,0(sp)
......
csrwi  0x7ee,11
csrwi  0x7ef,12
csrwi  0x7eb,13
csrrw  ra,0x7ed,ra
csrci  mstatus,8
csrw   0x7c4,t0
csrw   mepc,t0
......
lw    ra,0(sp)
......
mret
```

0x7c4	MRW	msubm
0x7eb	MRW	pushmsubm
0x7ec	MRW	mtvt2
0x7ed	MRW	jalmnxti
0x7ee	MRW	pushmcause
0x7ef	MRW	pushmepc

中断虚拟化(向量化)


Main program
Take Interrupt: Jump to Interrupt (ID = 30) Service Routine
<Save Context>
Interrupt Preemption: Jump to Interrupt (ID = 31) Service Routine
<Save Context>
Interrupt Preemption: Jump to Interrupt (ID = 32) Service Routine
<Save Context>
<Restore Context>
<Restore Context>
<Restore Context>
Back to the main program

嵌套流程

mstatus的MIE域

mstatus
寄存器中的 MIE 域表示机器模式下的全局中断使能：
当MIE 域的值为 1 时，表示中断的全局开关打开，中断能够被正常响应；
当MIE 域的值为 0 时，表示全局关闭中断，中断被屏蔽，无法被响应。

nuclei-sdk/demo_eclic

```
__INTERRUPT void eclic_msip_handler(void)
{
    static uint32_t int_sw_cnt = 0;   /* software i
nterrupt counter */
    // save CSR context
    SAVE_IRQ_CSR_CONTEXT();
    SysTimer_ClearSWIRQ();
    ……
    // restore CSR context
    RESTORE_IRQ_CSR_CONTEXT();
}
```

eclic_msip_handler

```
8000115a <eclic_msip_handler>:
8000115c:    df86                    sw  ra,252(sp)
)
8000117e:    dd7e                    sw  t6,184(sp)
80001180:    b502                    fsd ft0,168(sp)
800011a6:    a87e                    fsd ft11,16(sp)
800011aa:    342027f3            csrr    a5,mcause
800011b6:    341027f3            csrr    a5,mepc
800011c2:    7c4027f3            csrr    a5,0x7c4
800011ce:    40a1                    li  ra,8
800011d0:    3000a073            csrs    mstatus,ra
80001224:    3004b073            csrc    mstatus,s1
8000122e:    7c449073            csrw    0x7c4,s1
80001236:    34149073            csrw    mepc,s1
8000123e:    34249073            csrw    mcause,s1
80001244:    50fe                    lw  ra,252(sp)
80001266:    5fea                    lw  t6,184(sp)
80001268:    302a                    fld ft0,168(sp)
8000128e:    2fc2                    fld ft11,16(sp)
80001292:    30200073            mret
```

外设虚拟化—UART

寄存器名称	偏移地址	描述
UART_TXDATA	0x000	发送数据寄存器
UART_RXDATA	0x004	接收数据寄存器
UART_TXCTRL	0x008	发送控制寄存器
UART_RXCTRL	0x00C	接收控制寄存器
UART_IE	0x010	UART中断使能寄存器
UART_IP	0x014	UART中断等待标志寄存器
UART_DIV	0x018	波特率生成分频系数

UART描述：

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

    /*< public >*/
    qemu_irq irq;
    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;
```

```
static const MemoryRegionOps uart_ops = {
    .read = uart_read,
    .write = uart_write,
    .endianness = DEVICE_NATIVE_ENDIAN,
    .valid = {
        .min_access_size = 4,
        .max_access_size = 4
    }
};
```

UART后端设置：

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

Uart 实现

寄存器读操作：

```
static uint64_t
uart_read(void *opaque, hwaddr offset, unsigned int siz
e)
{
    NucLeiUARTState *s = opaque;
    uint64_t value = 0;
    uint8_t fifo_val;

    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;
    case NUCLEI_UART_REG_TXCTRL:
        value = s->txctrl;
        break;
 ……
}
```

寄存器写操作：

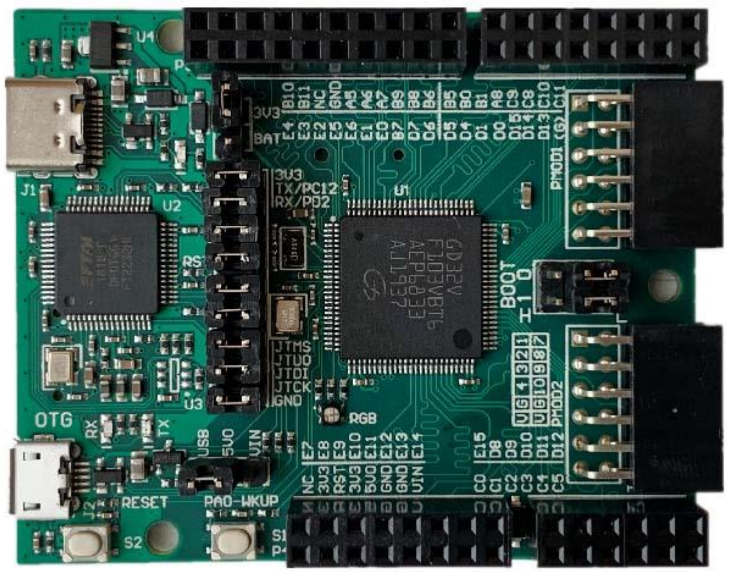
```
static void
uart_write(void *opaque, hwaddr offset,
           uint64_t value, unsigned int siz
e)
{
    NucLeiUARTState *s = opaque;
    unsigned char ch = value;

    switch (offset)
    {
    case NUCLEI_UART_REG_TXDATA:
        qemu_chr_fe_write(&s->chr, &ch, 1);
        update_irq(s);
        break;
    case NUCLEI_UART_REG_TXCTRL:
        s->txctrl = value;
        break;
……
}
```

UART中断：

```
static void update_irq(NucLeiUARTState *
s)
{
    int cond = 0;
    s->txctrl |=  0x1;
    if (s->rx_fifo_len)
        s->rxctrl &= ~0x1;
     else
        s->rxctrl |= 0x1;

    if ((s->ie & NUCLEI_UART_IE_TXWM) ||
        ((s-
>ie & NUCLEI_UART_IE_RXWM) && s-
>rx_fifo_len)) {
        cond = 1;
    }

    if (cond ) {
        qemu_irq_raise(s->irq);
    } else  {
        qemu_irq_lower(s->irq);
    }
}
```

李威威 I2C模拟 [video]

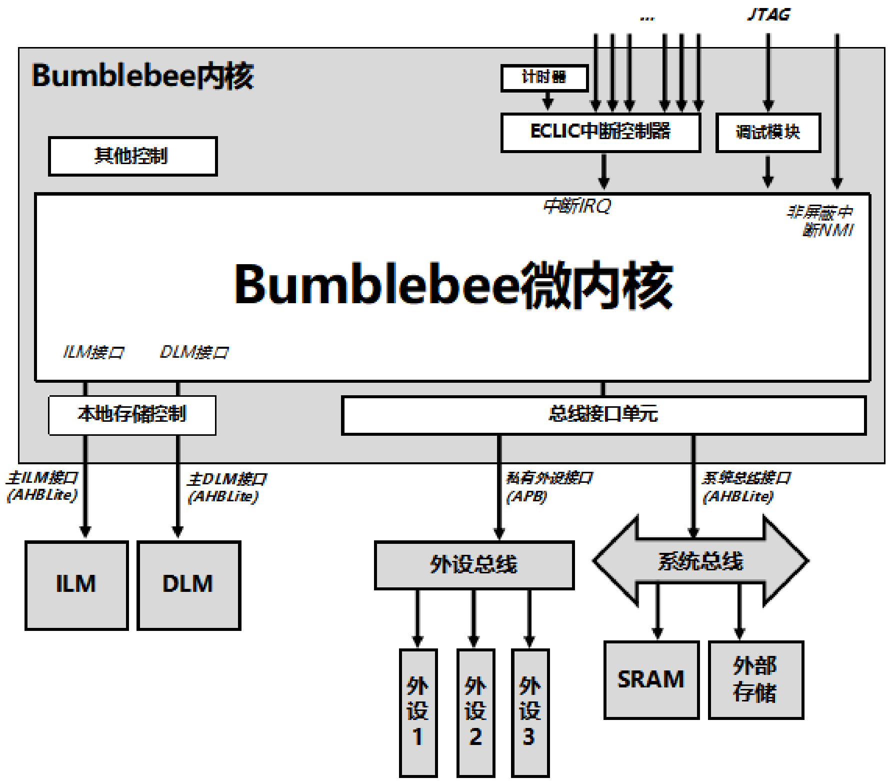
GD32V103F Board



微控制器：GD32VF103VBT6（32位RISC-V处理器）
内核：芯来科技Bumblebee内核（RV32IMAC）
主频：108MHz
内存：内置128KB Flash、32KB SRAM
工作电压：2.6~3.6V
外设资源：Timer（高级16位定时器*1，通用16位定时器*4）
U(S)ART*5、I2C*2、SPI*3、CAN*2、USBFS*1、
ADC*2（16路外部通道）、DAC*2、EXMC*1

gd32v103f

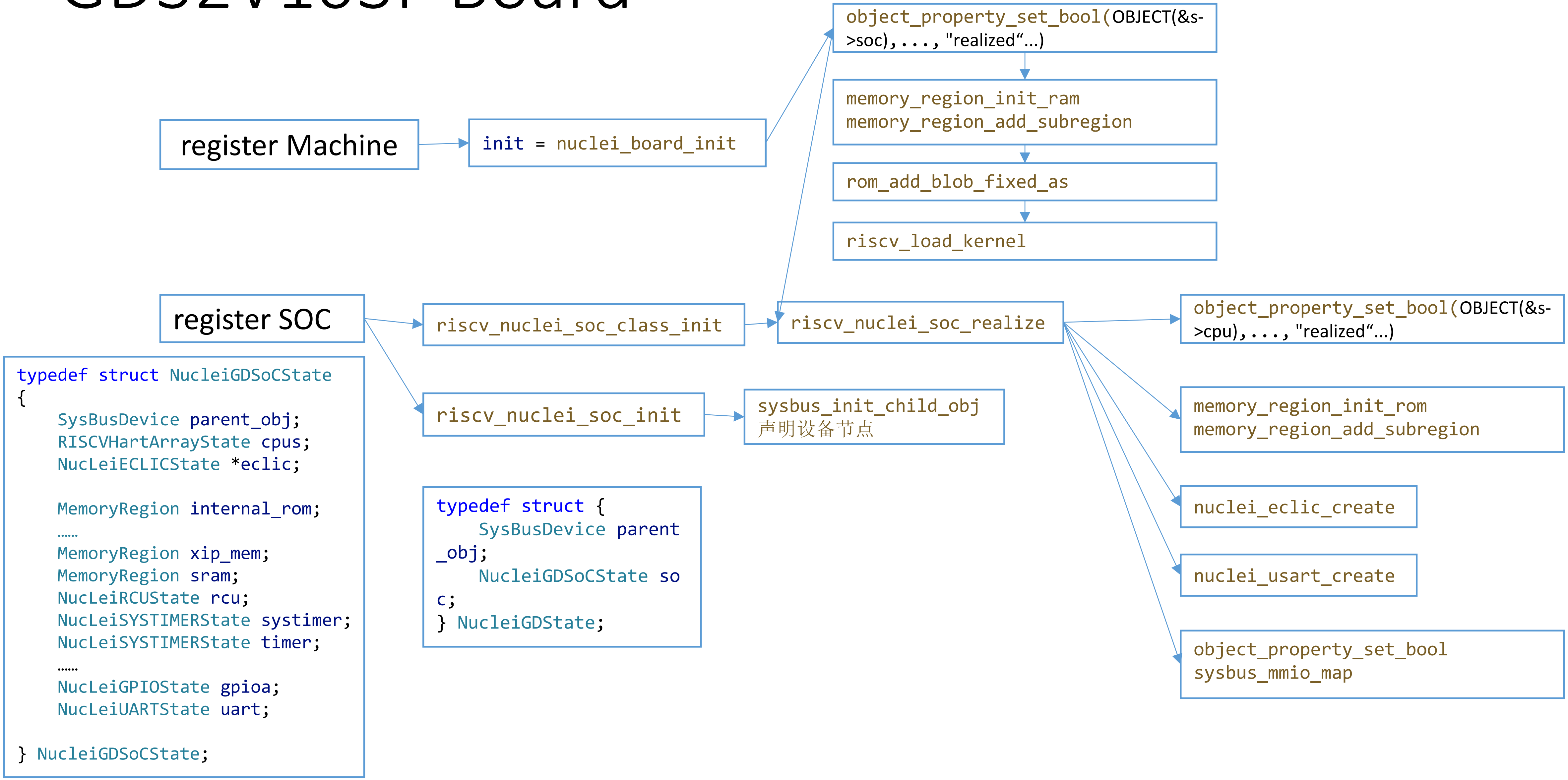
GD32V103F Board



gd32v103f

微控制器：GD32VF103VBT6（32位RISC-V处理器）
内核：芯来科技Bumblebee内核（RV32IMAC）
主频：108MHz
内存：内置128KB Flash、32KB SRAM
工作电压：2.6~3.6V
外设资源：Timer（高级16位定时器*1，通用16位定时器*4）
U(S)ART*5、I2C*2、SPI*3、CAN*2、USBFS*1、
ADC*2（16路外部通道）、DAC*2、EXMC*1

GD32V103F Board



register Machine

init = nuclei_board_init

object_property_set_bool(OBJECT(&s->soc), . . . , "realized“…)

memory_region_init_ram
memory_region_add_subregion

rom_add_blob_fixed_as

riscv_load_kernel

register SOC

riscv_nuclei_soc_class_init

riscv_nuclei_soc_realize

object_property_set_bool(OBJECT(&s->cpu), . . . , "realized“…)

memory_region_init_rom
memory_region_add_subregion

nuclei_eclic_create

nuclei_usart_create

object_property_set_bool
sysbus_mmio_map

riscv_nuclei_soc_init

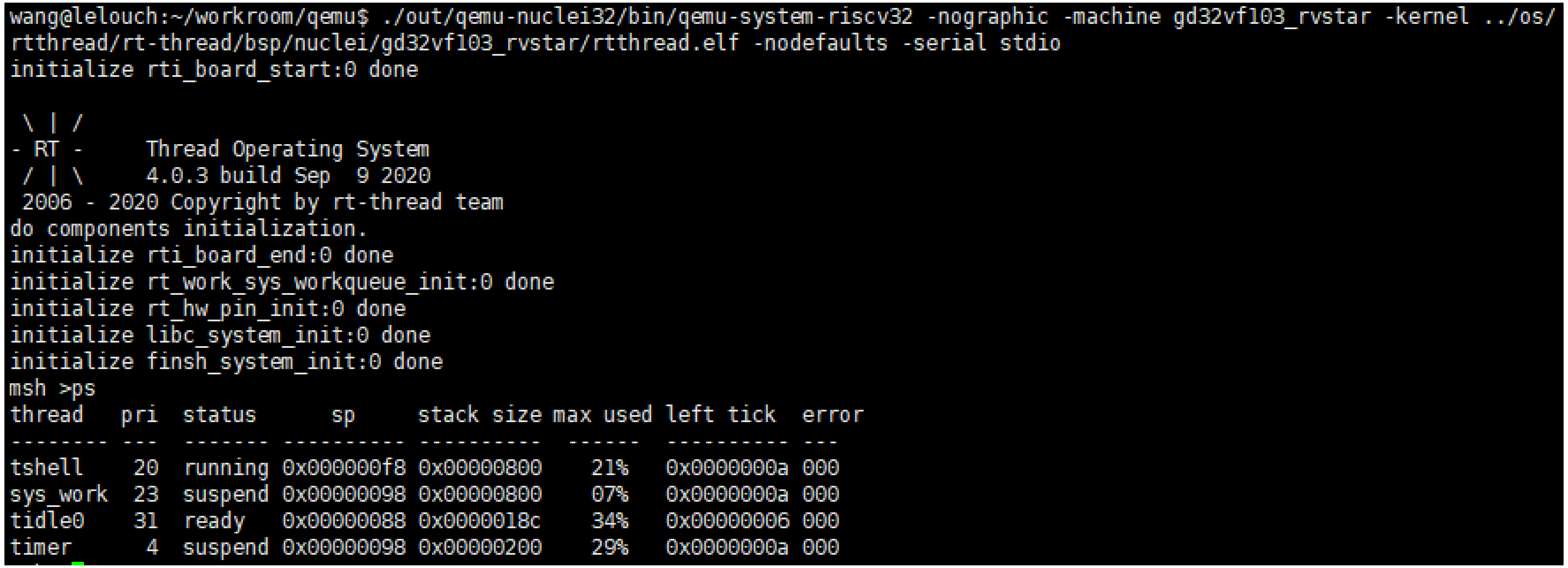
sysbus_init_child_obj
声明设备节点

```
typedef struct NucleiGDSoCState
{
    SysBusDevice parent_obj;
    RISCVHartArrayState cpus;
    NucLeiECLICState *eclic;

    MemoryRegion internal_rom;
    ……
    MemoryRegion xip_mem;
    MemoryRegion sram;
    NucLeiRCUState rcu;
    NucLeiSYSTIMERState systimer;
    NucLeiSYSTIMERState timer;
    ……
    NucLeiGPIOState gpioa;
    NucLeiUARTState uart;

} NucleiGDSoCState;
```

```
typedef struct {
    SysBusDevice parent
_obj;
    NucleiGDSoCState so
c;
} NucleiGDState;
```

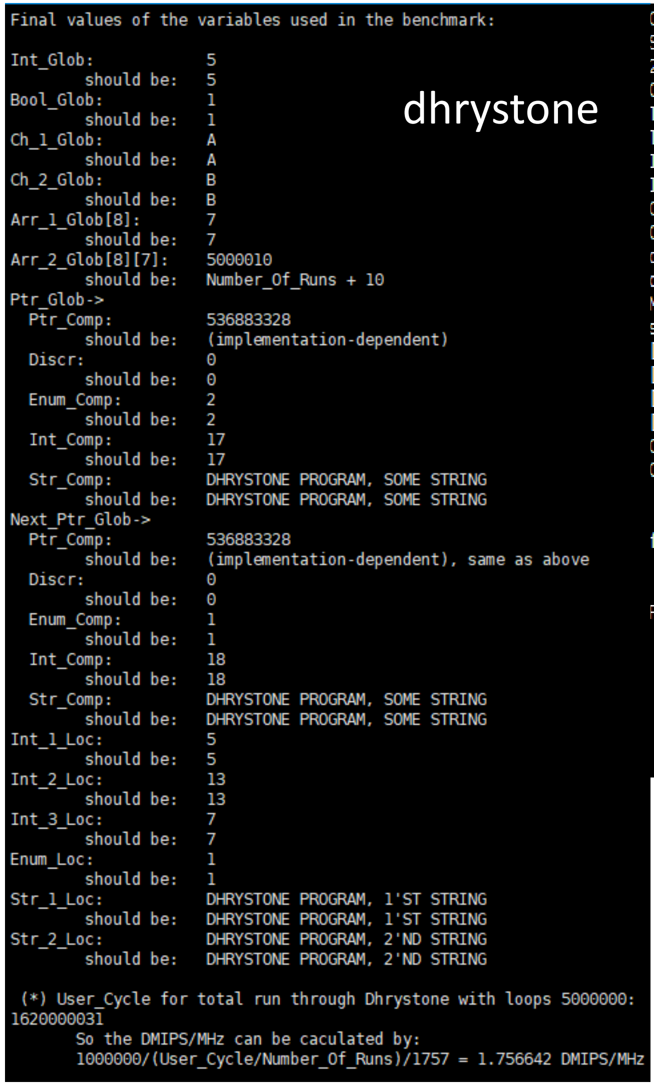
GD32V103F Board

```
qemu-system-riscv32 -nographic -machine gd32vf103_rvstar \
-kernel ../os/rtthread/rt-thread/bsp/nuclei/gd32vf103_rvstar/rtthread.elf -nodefaults -serial stdio
```

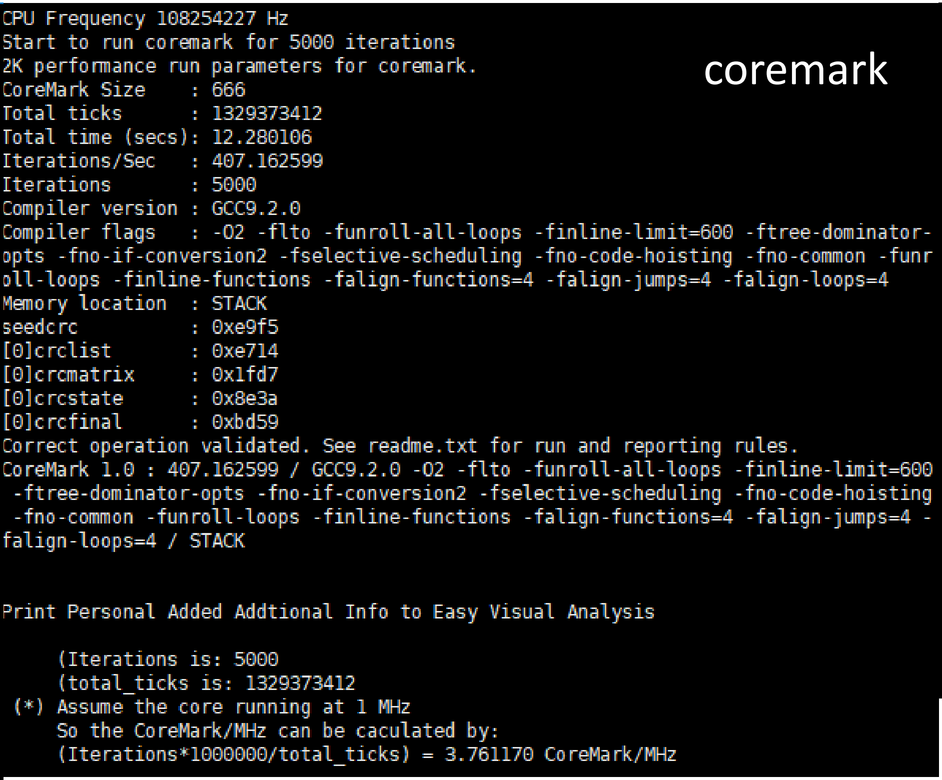


```
wang@lelouch:~/workroom/qemu$ ./out/qemu-nuclei32/bin/qemu-system-riscv32 -nographic -machine gd32vf103_rvstar -kernel ../os/
rtthread/rt-thread/bsp/nuclei/gd32vf103_rvstar/rtthread.elf -nodefaults -serial stdio
initialize rti_board_start:0 done

 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Sep  9 2020
 2006 - 2020 Copyright by rt-thread team
do components initialization.
initialize rti_board_end:0 done
initialize rt_work_sys_workqueue_init:0 done
initialize rt_hw_pin_init:0 done
initialize libc_system_init:0 done
initialize finsh_system_init:0 done
msh >ps
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
tshell    20  running 0x000000f8 0x00000800    21%   0x0000000a 000
sys_work  23  suspend 0x00000098 0x00000800    07%   0x0000000a 000
tidle0    31  ready   0x00000088 0x0000018c    34%   0x00000006 000
timer      4  suspend 0x00000098 0x00000200    29%   0x0000000a 000
```

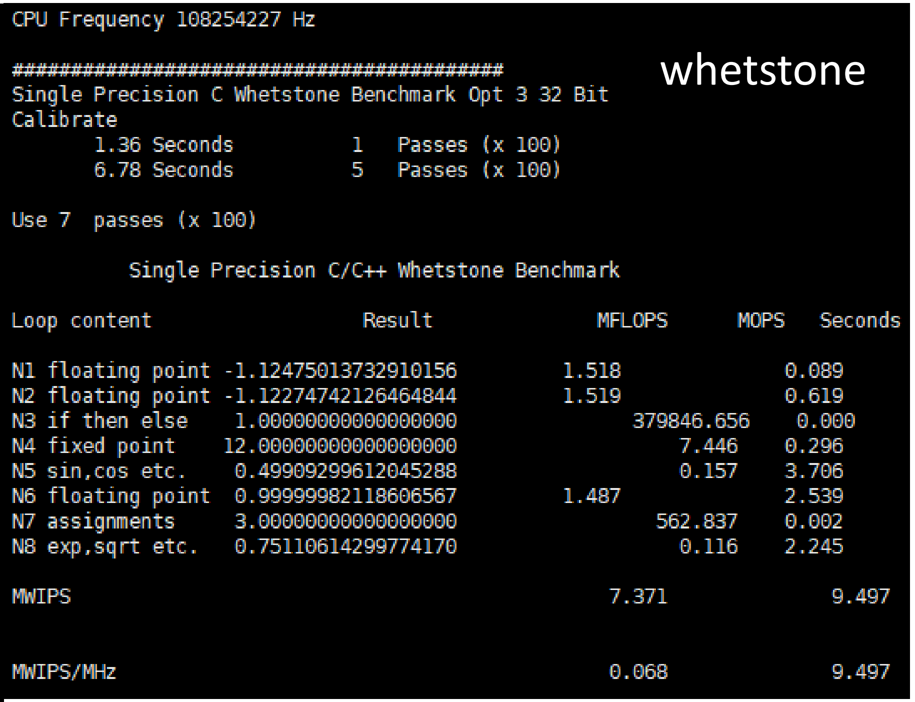
GD32V103F Board



```
Final values of the variables used in the benchmark:

Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    5000010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          536883328
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          536883328
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:           5
        should be:   5
Int_2_Loc:           13
        should be:   13
Int_3_Loc:           7
        should be:   7
Enum_Loc:            1
        should be:   1
Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
        should be:   DHRYSTONE PROGRAM, 2'ND STRING

 (*) User_Cycle for total run through Dhrystone with loops 5000000:
1620000031
       So the DMIPS/MHz can be caculated by:
       1000000/(User_Cycle/Number_Of_Runs)/1757 = 1.756642 DMIPS/MHz
```



```
CPU Frequency 108254227 Hz
Start to run coremark for 5000 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 1329373412
Total time (secs): 12.280106
Iterations/Sec   : 407.162599
Iterations       : 5000
Compiler version : GCC9.2.0
Compiler flags   : -O2 -flto -funroll-all-loops -finline-limit=600 -ftree-dominator-
opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -fno-common -funr
oll-loops -finline-functions -falign-functions=4 -falign-jumps=4 -falign-loops=4
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xbd59
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 407.162599 / GCC9.2.0 -O2 -flto -funroll-all-loops -finline-limit=600
 -ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting
 -fno-common -funroll-loops -finline-functions -falign-functions=4 -falign-jumps=4 -
falign-loops=4 / STACK


Print Personal Added Addtional Info to Easy Visual Analysis

     (Iterations is: 5000
     (total_ticks is: 1329373412
 (*) Assume the core running at 1 MHz
     So the CoreMark/MHz can be caculated by:
     (Iterations*1000000/total_ticks) = 3.761170 CoreMark/MHz
```

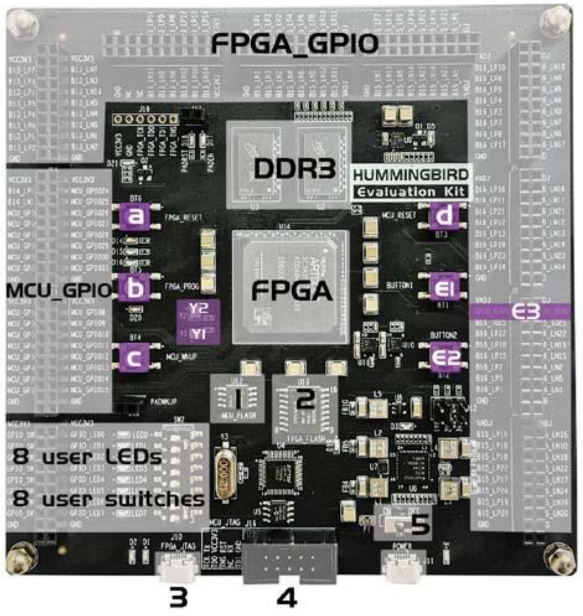


```
CPU Frequency 108254227 Hz

##########################################
Single Precision C Whetstone Benchmark Opt 3 32 Bit
Calibrate
       1.36 Seconds          1   Passes (x 100)
       6.78 Seconds          5   Passes (x 100)

Use 7  passes (x 100)

          Single Precision C/C++ Whetstone Benchmark

Loop content                  Result              MFLOPS      MOPS   Seconds

N1 floating point -1.12475013732910156       1.518              0.089
N2 floating point -1.12274742126464844       1.519              0.619
N3 if then else    1.00000000000000000            379846.656    0.000
N4 fixed point    12.00000000000000000                 7.446    0.296
N5 sin,cos etc.    0.49909299612045288                 0.157    3.706
N6 floating point  0.99999982118606567       1.487              2.539
N7 assignments     3.00000000000000000               562.837    0.002
N8 exp,sqrt etc.   0.75110614299774170                 0.116    2.245

MWIPS                                         7.371                  9.497


MWIPS/MHz                                     0.068                  9.497
```

	dhrystone	coremark	whetstone
QEMU benchmark	1.756642 DMIPS/MHz	3.761170 CoreMark/MHz	0.068MWIPS/MHz
SDK/app.rst	1.273270 DMIPS/MHz	3.081076 CoreMark/MHz	0.046MWIPS/MHz

HummingBird Board

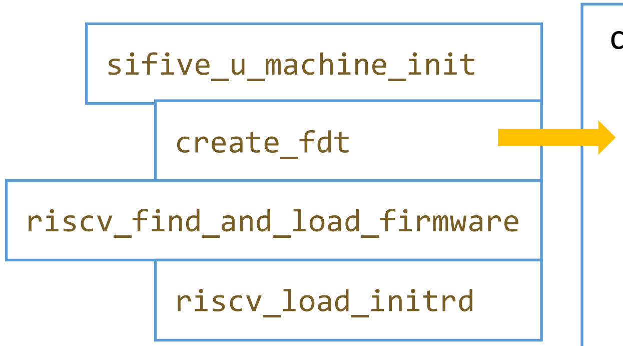
a: FPGA_RESET
b: FPGA_PROG
c: MCU_WKUP
d: MCU_RESET
e1: User button 1
e2: User button 2
e3: User button header
Y1: GCLK
Y2: RTC_CLK
1: MCU_FLASH
2: FPGA_FLASH
3: FPGA_JTAG
4: MCU_JTAG
5: Power switch



蜂鸟FPGA评估板

蜂鸟FPGA评估板
主芯片FPGA型号为Xilinx XC7A100T
板载双晶振设计：100MHz主时钟和32.768KHz RTC时钟
配备两颗MT41K128M16JT-125K DDR III 颗粒
备独立的MCU_FLASH芯片，此Flash用于存储RISC-V内核运行的程序文件
配备独立的FPGA_Flash芯片，此Flash用于存储mcs格式的比特流文件
配备用户LEDs、按键及拨码开关
配备独立的32个MCU GPIO
配备多达126个引出的FPGA GPIO，用于用户自定义使用
配备了FPGA_GPIO电平选择跳线接口，可以选择1.8V\2.5V\3.3V接口电平

HummingBird Board

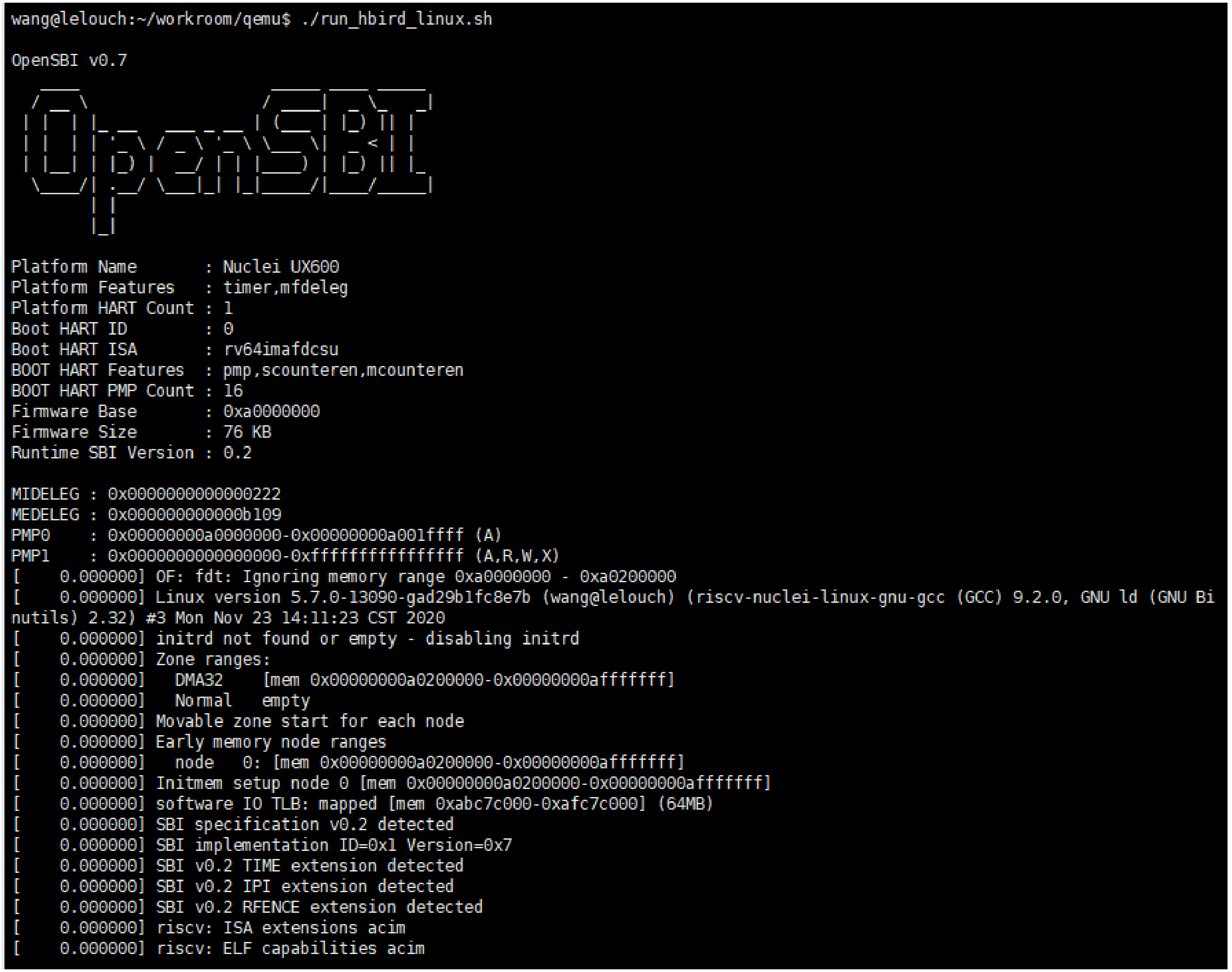


```
cpus {
  #address-cells = <1>;
  #size-cells = <0>;
  timebase-frequency = <32768>;
  cpu0: cpu@0 {
   device_type = "cpu";
   reg = <0>;
   status = "okay";
   compatible = "riscv";
   riscv,isa = "rv64imac";
   mmu-type = "riscv,sv39";
   clock-frequency = <8000000>;
   cpu0_intc: interrupt-controller {
    #interrupt-cells = <1>;
    interrupt-controller;
    compatible = "riscv,cpu-intc";
   };
  };
};
```

```
    qemu_fdt_add_subnode(fdt, "/cpus");
    qemu_fdt_setprop_cell(fdt, "/cpus", "timebase-frequency",
        SIFIVE_CLINT_TIMEBASE_FREQ);
    qemu_fdt_setprop_cell(fdt, "/cpus", "#size-cells", 0x0);
    qemu_fdt_setprop_cell(fdt, "/cpus", "#address-cells", 0x1);

    for (cpu = ms->smp.cpus - 1; cpu >= 0; cpu--) {
        int cpu_phandle = phandle++;
        nodename = g_strdup_printf("/cpus/cpu@%d", cpu);
        char *intc = g_strdup_printf("/cpus/cpu@%d/interrupt-controller", cpu);
        char *isa;
        qemu_fdt_add_subnode(fdt, nodename);
        /* cpu 0 is the management hart that does not have mmu */
        if (cpu != 0) {
            qemu_fdt_setprop_string(fdt, nodename, "mmu-type", "riscv,sv39");
            isa = riscv_isa_string(&s->soc.u_cpus.harts[cpu - 1]);
        } else {
            isa = riscv_isa_string(&s->soc.e_cpus.harts[0]);
        }
        qemu_fdt_setprop_string(fdt, nodename, "riscv,isa", isa);
        qemu_fdt_setprop_string(fdt, nodename, "compatible", "riscv");
        qemu_fdt_setprop_string(fdt, nodename, "status", "okay");
        qemu_fdt_setprop_cell(fdt, nodename, "reg", cpu);
        qemu_fdt_setprop_string(fdt, nodename, "device_type", "cpu");
        qemu_fdt_add_subnode(fdt, intc);
        qemu_fdt_setprop_cell(fdt, intc, "phandle", cpu_phandle);
        qemu_fdt_setprop_string(fdt, intc, "compatible", "riscv,cpu-intc");
        qemu_fdt_setprop(fdt, intc, "interrupt-controller", NULL, 0);
        qemu_fdt_setprop_cell(fdt, intc, "#interrupt-cells", 1);
        g_free(isa);
        g_free(intc);
        g_free(nodename);
    }
```

HummingBird Board

```
$qemu-system-riscv64  \
-M nuclei_u \
-nographic \
-m 1G \
-bios none \
-icount shift=0 \
-append "root=/dev/ram rw console=ttyS0 earlycon=sbi" \
-kernel  ~/workroom/risc-v/nuclei/nuclei-linux-sdk/work/opensbi/platform/nuclei/ux600/firmware/fw_payload.elf \
-initrd ./rootfs_riscv64.img
```



```
wang@lelouch:~/workroom/qemu$ ./run_hbird_linux.sh

OpenSBI v0.7
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name       : Nuclei UX600
Platform Features   : timer,mfdeleg
Platform HART Count : 1
Boot HART ID        : 0
Boot HART ISA       : rv64imafdcsu
BOOT HART Features  : pmp,scounteren,mcounteren
BOOT HART PMP Count : 16
Firmware Base       : 0xa0000000
Firmware Size       : 76 KB
Runtime SBI Version : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x00000000a0000000-0x00000000a001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
[    0.000000] OF: fdt: Ignoring memory range 0xa0000000 - 0xa0200000
[    0.000000] Linux version 5.7.0-13090-gad29b1fc8e7b (wang@lelouch) (riscv-nuclei-linux-gnu-gcc (GCC) 9.2.0, GNU ld (GNU Bi
nutils) 2.32) #3 Mon Nov 23 14:11:23 CST 2020
[    0.000000] initrd not found or empty - disabling initrd
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x00000000a0200000-0x00000000afffffff]
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x00000000a0200000-0x00000000afffffff]
[    0.000000] Initmem setup node 0 [mem 0x00000000a0200000-0x00000000afffffff]
[    0.000000] software IO TLB: mapped [mem 0xabc7c000-0xafc7c000] (64MB)
[    0.000000] SBI specification v0.2 detected
[    0.000000] SBI implementation ID=0x1 Version=0x7
[    0.000000] SBI v0.2 TIME extension detected
[    0.000000] SBI v0.2 IPI extension detected
[    0.000000] SBI v0.2 RFENCE extension detected
[    0.000000] riscv: ISA extensions acim
[    0.000000] riscv: ELF capabilities acim
```

Future Work

GitHub: https://github.com/isrc-cas/plct-qemu

- 代码版本更新和上游推送
- UX600系列, UX900系列支持
- SPI,ADC,QSPI,DMA等必要外设实现
- CLINT,PLIC,ECLIC等配置模块切换
- P-Extensions实现
- ……

欢迎实习生加入(。·∀·)ノ゛

https://zhuanlan.zhihu.com/p/271625260

谢谢

wangjunqiang@iscas.ac.cn