Presentation

Wuzhenghui

2020/08/31

1. IDF Environment

- IDF Structrue
- Project Structrue
- IDF Commands
- Project Build Process

2. Bring Up

- ESP32-S3 Core System
- Boot Process
- Bring UP Steps

3. ROM Code (C3)

- Function
- Test

4. Cache Verification

- Cache Structrue
- Verification

5. RISC-V CPU

- RISC-V CPU
- ESP32 C3 ROM Code & Boot
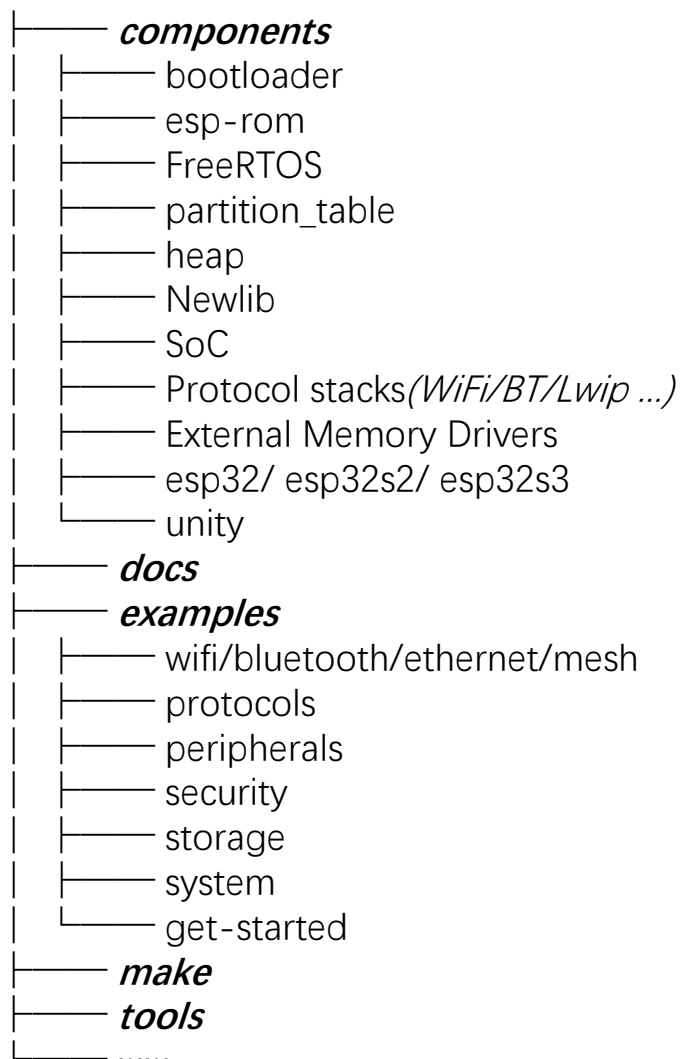
6. FreeRTOS

- RISC-V CPU
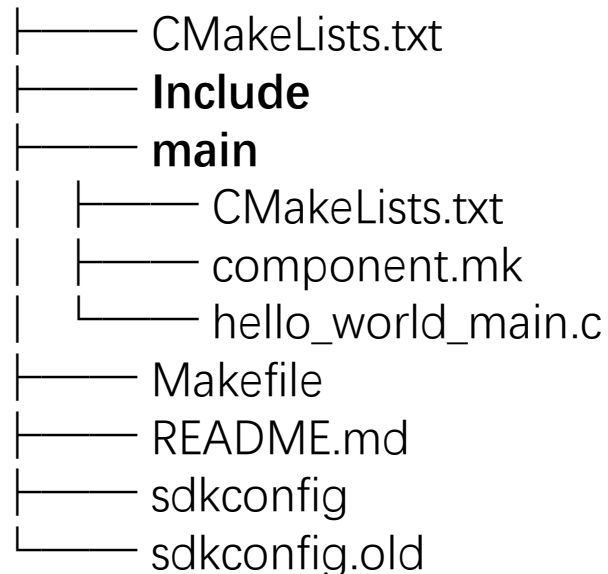- ESP32 C3 ROM Code & Boot

/01   IDF Environment

- IDF Structrue
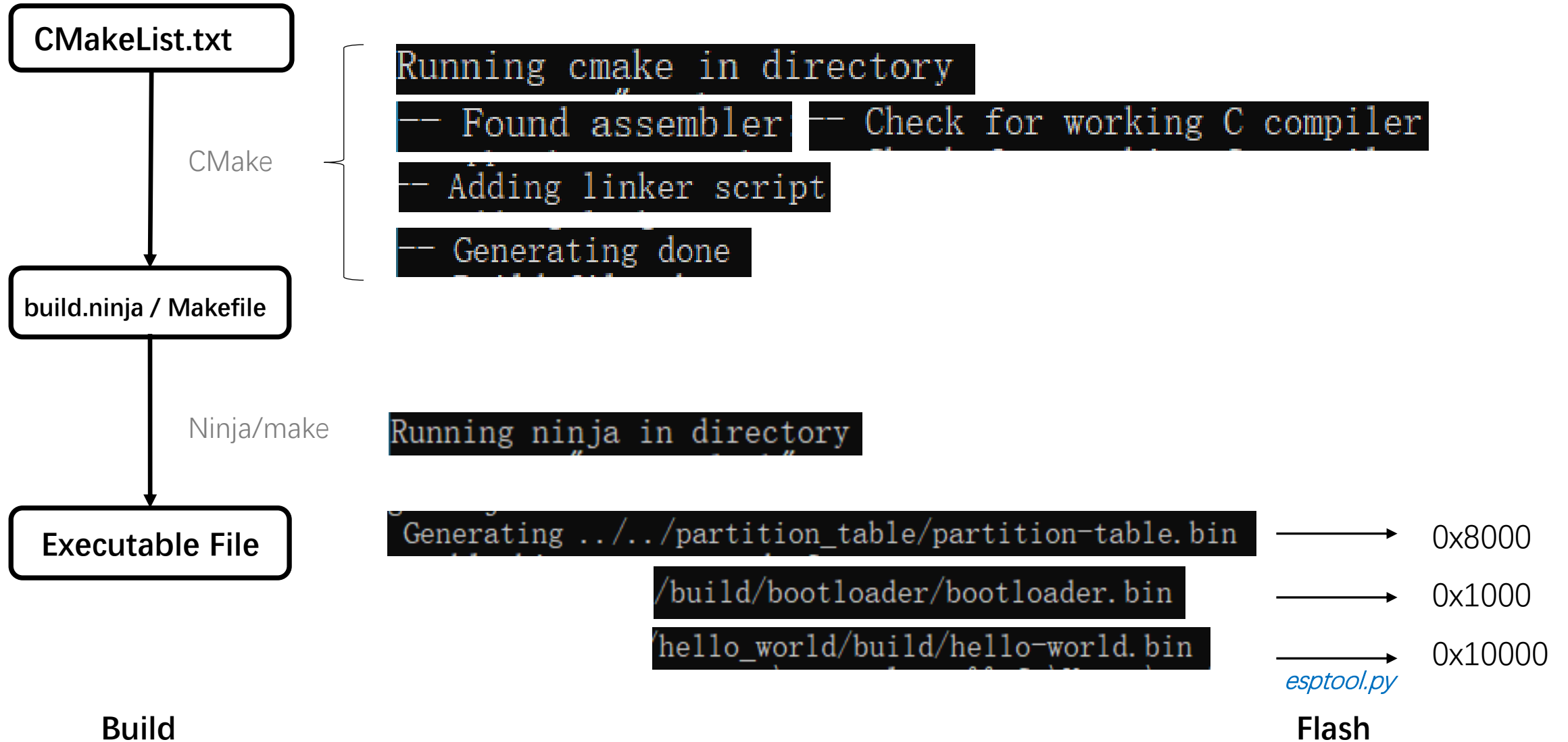- Project Structrue
- IDF Commands
- Project Build Process

IDF

ESPRESSIF

IDF Structure:

```
├──── components
│    ├──── bootloader
│    ├──── esp-rom
│    ├──── FreeRTOS
│    ├──── partition_table
│    ├──── heap
│    ├──── Newlib
│    ├──── SoC
│    ├──── Protocol stacks(WiFi/BT/Lwip …)
│    ├──── External Memory Drivers
│    ├──── esp32/ esp32s2/ esp32s3
│    └──── unity
├──── docs
├──── examples
│    ├──── wifi/bluetooth/ethernet/mesh
│    ├──── protocols
│    ├──── peripherals
│    ├──── security
│    ├──── storage
│    ├──── system
│    └──── get-started
├──── make
├──── tools
└──── .....
```

Project Structure:

```
├──── CMakeLists.txt
├──── Include
├──── main
│    ├──── CMakeLists.txt
│    ├──── component.mk
│    └──── hello_world_main.c
├──── Makefile
├──── README.md
├──── sdkconfig
└──── sdkconfig.old
```

Commands:

idf.py menuconfig
    build
    set-target <chip name>
    flash
    monitor
    clean / fullclean
    erase_flash
    gdbgui/gdbtui
    size
    . . .

Project Build Step

ESPRESSIF

CMakeList.txt

CMake

```
Running cmake in directory
-- Found assembler        -- Check for working C compiler
-- Adding linker script
-- Generating done
```

build.ninja / Makefile

Ninja/make

```
Running ninja in directory
```

Executable File

```
Generating ../../partition_table/partition-table.bin
/build/bootloader/bootloader.bin
hello_world/build/hello-world.bin
```

→ 0x8000

→ 0x1000

→ 0x10000

esptool.py

Build

Flash

/02 Bring Up

- ESP32-S3 Core System
- Boot Process
- ROM Code
- Bootloader
- Bring UP Steps

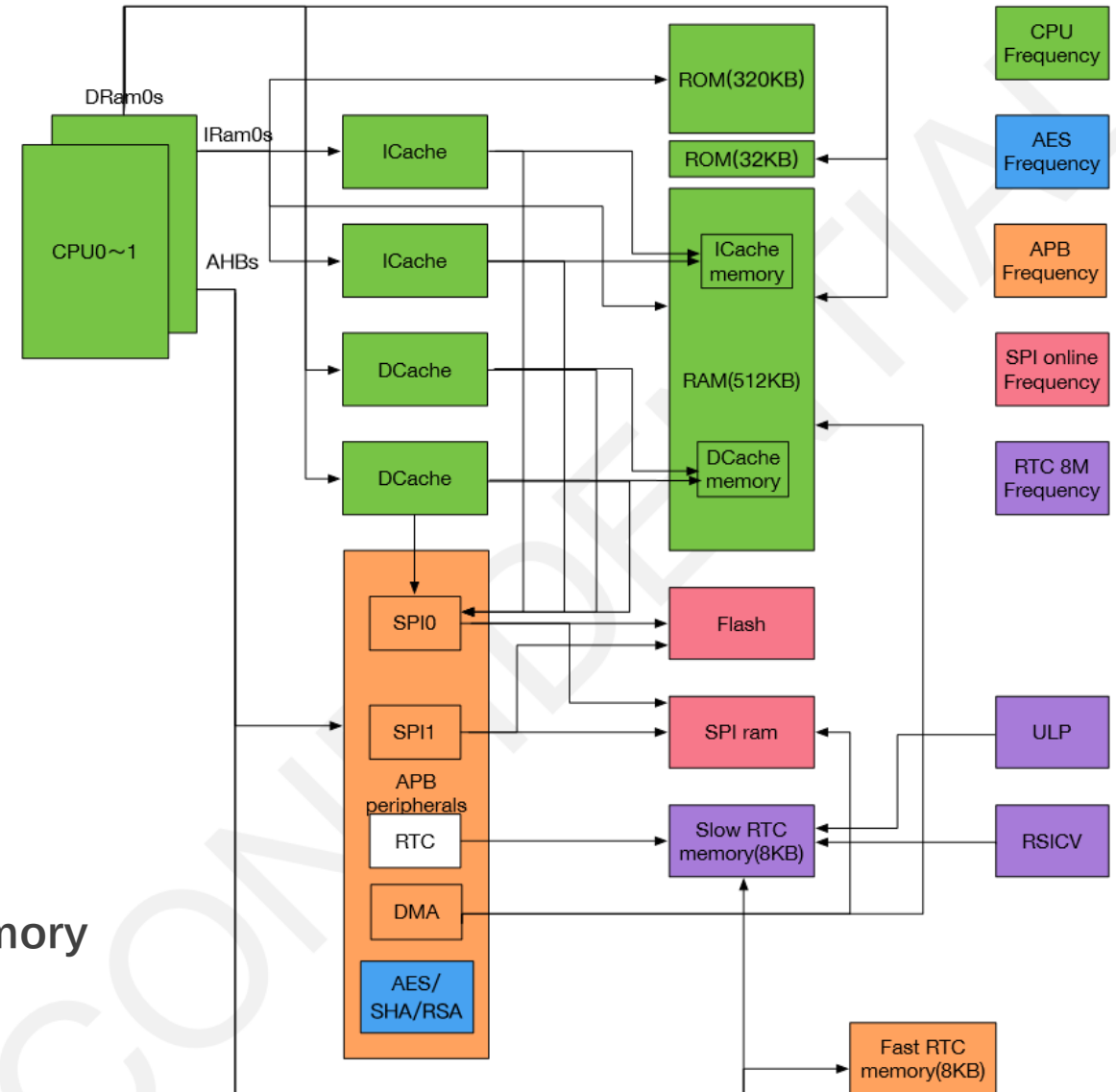# ESP32-S3 Core System

CPU:

- Two Lx7 Cores
- RISC-V Core
- ULP

Buses (Click jump to Bus-Memory Map)**:**

- IRam0
- DRam0
- AHB

Memory:

- 320KB ROM
- 512KB SRAM
- 8KB Fast RTC Memory & 8KB Slow RTC Memory
- Virtual memory: map to external memory
- Memory inside peripherals

ESP32-S3 Core System

Peripherals & DMA

- **High speed peripherals （ 1x，2x or 3x APB frequency)**

- **APB peripherals**

- **Peripherals**

DMA (arbitrate with CPU buses access) → Memory

EDMA (arbitrate with cache access) → External Flash & SPI Ram

Cache System

- Use part of SRAM as cache memory

- MMU Table : VA to PA

- Tag memory : Record cache memory states

ROM Code & Bootloader

1. ROM中的第一级引导加载程序将Flash中 offset = 0x1000 的第二级引导加载程序映像加载到RAM（IRAM和DRAM）

2. 第二级引导程序从闪存加载分区表(0x8000)和主应用程序映像(0x10000)。

esp-idf\components\bootloader\subproject\main\bootloader_start.c   ----   bootloader.bin   (0x1000)

```c
void __attribute__((noreturn)) call_start_cpu0()
{
    // 1. Hardware initialization
    if (bootloader_init() != ESP_OK) {
        bootloader_reset();
    }
    // 2. Select the number of boot partition
    bootloader_state_t bs = { 0 };
    int boot_index = select_partition_number(&bs);
    if (boot_index == INVALID_INDEX) {
        bootloader_reset();
    }
    // 3. Load the app image for booting
    bootloader_utility_load_boot_image(&bs, boot_index);
}
```

3. 应用程序 (offset = 0x10000) 启动阶段

esp-idf\components\esp32/cpu_start..c

Check CPU related flies

- **CPU archive**

  /*Has the Overlays we need to generate new toolchain for specific CPU*/

  - binutils：One set includes linker, assembler and other tools for object files and archives

  - config：Configuration files

  - cstub

  - gcc

  - gdb

- **CPU SDK**

  - HAL：Hardware abstract layer files

  - XTOS：A simple OS without context switch to make CPU boot

- **Header files**

  /*Works on specific CPU platform generated by Xtensa EDA*/

Bring Up Steps

ESPRESSIF

Toolchain

- Use script to make structure changes from the provided overlays

- Generate new toolchain

Newlib

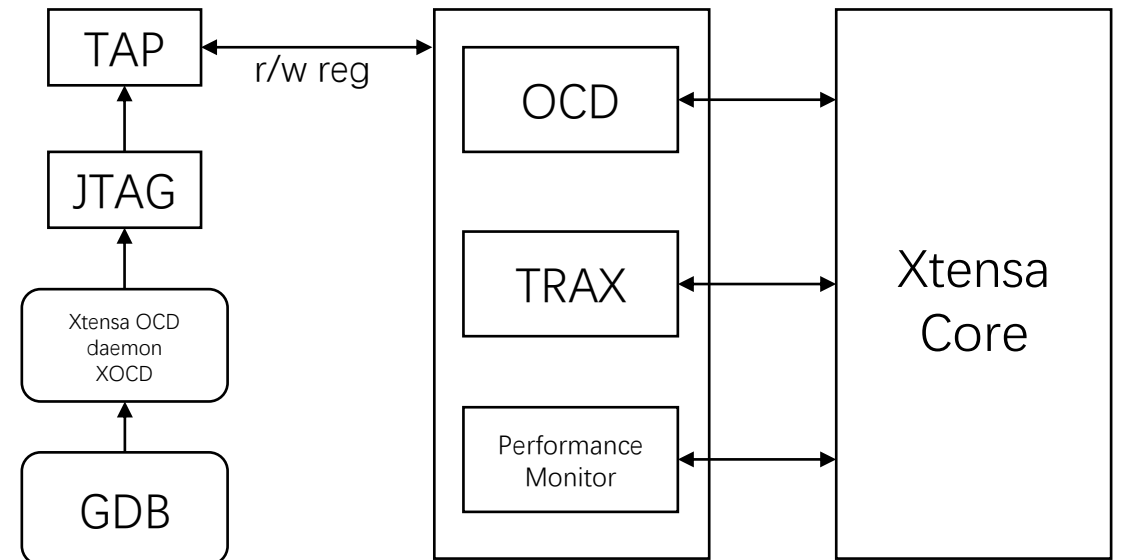*/*A C library intended for use on embedded systems*/*

- Make Newlib

OpenOCD

- Chip memory layout and registers modifications

  xtensa-config.c   *reg-xtensa.dat* → *esp108_regs.h*

- Config files

  */*describes the basic features about chip,*

  Including chip-name, endian and tapeid/*

TAP ← r/w reg →

OCD

JTAG

TRAX

Xtensa OCD daemon XOCD

Performance Monitor

GDB

Xtensa Core

Bring Up Steps

ESPRESSIF

Update SoC header flies

- Add/Remove registers

- Change register width

- Change register address and default value

- ……

Make ROM download work

- Modify makefile

- Build ROM & make it can be download

1. Download image to FPGA

2. Make Rom (eagle.pro.rom.out)

3. Generate link file for specific chip from eagle.pro.rom.out

4. Open OpenOCD Server

5. Use GDB Download ROM to FPGA by OpenOCD（Download Boot）

Bring Up Steps

ESPRESSIF

Make ROM boot work

esp-rom\rom\ets_kernel\ets_main.c

- Choose Reset Mode

- Initial SPI

- Initial Cache

- Parse Flash header

- Load { Text section, Data section, rodata }

```
/* ets_main.c */
main()
GET_CPUID        //获取CPUID
    APP_CPU:app_code_start()
    PRO_CPU:
        rtc_get_reset_reason()        //获取复位原因
        boot_prepare(reset_reson)
            Cache_Mask_All()          //屏蔽Cache
            rtc_unhold_all_pads()     //取消所有pad的hold状态
            uartAttach()              //设置uart模块的默认参数（不使能RX中断）
            ets_get_apb_freq()        //获取APB总线时钟频率
            ...                       //兼容性配置
        switch (reset_reason)         //判断boot模式
            DEEPSLEEP_RESET:
                    user_code_start()
            Others:
                    if(ETS_IS_FLASH_BOOT)
                        ets_flash_boot()
                    else
                        ...           //re-initialize the UART
                        ets_secure_download_mode()

ets_flash_boot()
    ets_efuse_get_spiconfig()
    spi_flash_boot_attach()           //初始化SPI
    ets_run_flash_bootloader()        //运行Flash中bootloader

/* ets_loader.c */
ets_run_flash_bootloader()
    ...                               //初始化Cache MMU，设置cache 模式，失效cache tags，使能Cache
    ets_loader_map_range()            //映射FLASH VA->PA
    memcpy(&fhdr, tmp, sizeof(struct flash_hdr));    //从FLASH中解析header
    ...
    /*Copy Text section, Data section, rodata to RAM and verify them*/
```
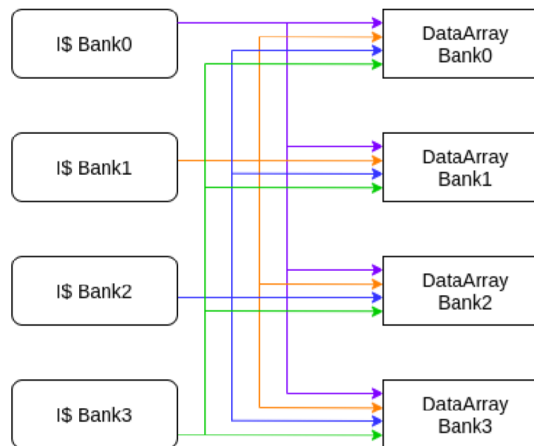
Bring Up Steps

- Initialize SPI

  spi_flash_boot_attach()

- Occupy cache bank to correct sram bank

  Cache_Occupy_DCache_MEMORY()

DBANK0 0001	IBANK0 0001
DBANK1 0010	IBANK1 0010
DBANK2 0100	IBANK2 0100
DBANK3 1000	IBANK3 1000

DCache   ICache

H1   H0   L1   L0   H1   H0   L1   L0
     HIGH   LOW   HIGH   LOW
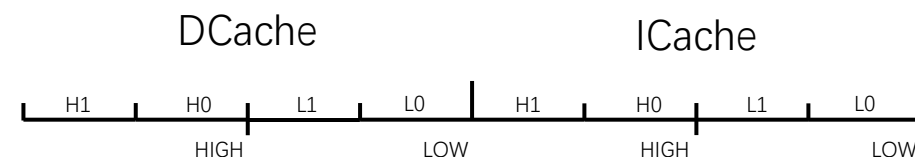
```c
void Cache_Occupy_DCache_MEMORY(cache_array_t dcache_low, cache_array_t dcache_high)
{
    uint32_t array_dcache = ((dcache_low << CACHE_DCACHE_LOW_SHIFT) | (dcache_high << CACHE_DCACHE_HIGH_SHIFT)) >> CACHE_DCACHE_LOW_SHIFT;
    uint32_t dcache_usage = (dcache_low | dcache_high) ^ SENSITIVE_INTERNAL_SRAM_DCACHE_FLATTEN;
    REG_SET_FIELD(SENSITIVE_CACHE_DATAARRAY_CONNECT_1_REG, SENSITIVE_DCACHE_DATAARRAY_CONNECT_FLATTEN, array_dcache);
    REG_SET_FIELD(SENSITIVE_INTERNAL_SRAM_USAGE_1_REG, SENSITIVE_INTERNAL_SRAM_DCACHE_FLATTEN, dcache_usage);
}
```

- Modify chip memory address MACRO

  /include/soc/soc.h   */include/soc/cache_memory.h*
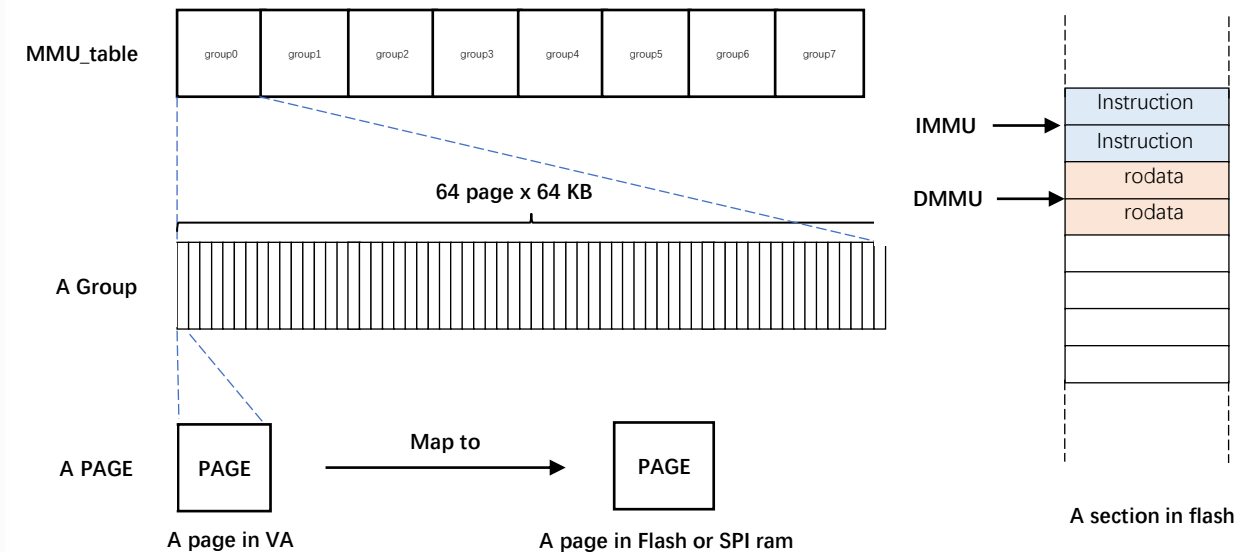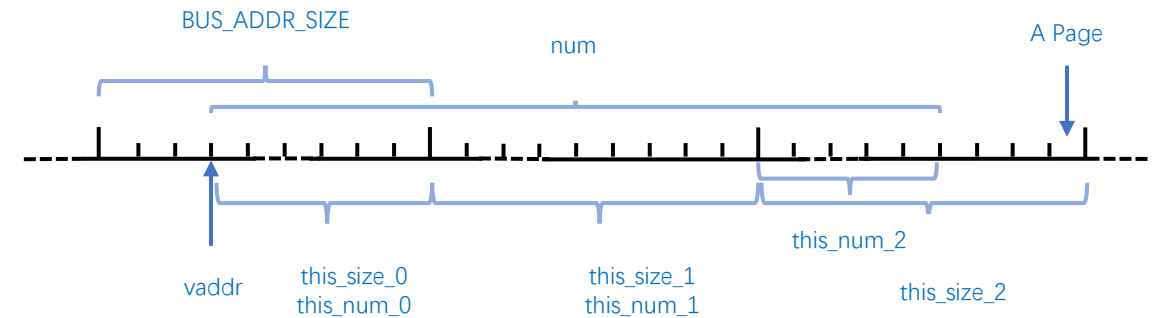
Bring Up Steps

ets_loader_map_range() ⟶ *Cache_Dbus_MMU_Set()*

```c
int Cache_Dbus_MMU_Set(uint32_t ext_ram,uint32_t vaddr, uint32_t paddr, uint32_t psize, uint32_t num, uint32_t fixed)
{
    uint32_t i,shift,mask_s;
    uint32_t mmu_addr;
    uint32_t mmu_table_val;

    //check if address page aligned
    if (ADDRESS_CHECK(vaddr,psize) || ADDRESS_CHECK(paddr,psize)){
        return MMU_SET_ADDR_ALIGNED_ERROR;
    }
    if (psize == 64) {
        shift  = 16;
        mask_s = 0;
    } else {
        return MMU_SET_PASE_SIZE_ERROR;
    }

    while (num > 0) {
        //split the vaddr room by the bus aligned address
        int this_size = ((vaddr + BUS_ADDR_SIZE) & ~(BUS_ADDR_SIZE - 1)) - vaddr;
        //calculate the pages to map this round
        int this_num = this_size >> shift;
        if (this_num > num) {
            this_num = num;
        }
        //calculate MMU value by physical address
        mmu_table_val = paddr >> shift;
        //calculate MMU address by virtual address
        if(ADDRESS_IN_DRAM0_CACHE(vaddr)) {
        mmu_addr = ((vaddr & (BUS_ADDR_MASK >> mask_s)) >> shift) * sizeof(uint32_t);
        } else {
            return MMU_SET_VADDR_OUT_RANGE;
        }
        //write the MMU values
        for ( i = 0; i < this_num; i++){
            if (!fixed) {
                *(uint32_t *)(DR_REG_MMU_TABLE + mmu_addr) = (mmu_table_val + i) | ext_ram;
            } else {//fixed mode always map to first physical page
                *(uint32_t *)(DR_REG_MMU_TABLE + mmu_addr) = (mmu_table_val) | ext_ram;
            }
            mmu_addr += 4;
        }
        //prepare for next round
        num -= this_num;
        if (!fixed) {
            paddr += this_size;
        }
        vaddr += this_size;
    }

    return 0;
}
```

Bring Up Steps

Bring Up bootloader & idf

- Update cache address MACRO *esp-idf/components/soc/soc/esp32s3/include/soc/soc.h & cache_memory.h*

- Update the address and Chip_ID in esptool.py *esp-idf/components/esptool_py/esptool/esptool.py*

- Update ld file in IDF *esp-idf/components/esp32s3/ld/*

1. bootloader_init()

```
memset(&_bss_start, 0, (&_bss_end - &_bss_start) * sizeof(_bss_start));
```

```
Cache_Suspend_DCache();
Cache_Invalidate_DCache_All();
Cache_MMU_Init();
```

```
flash_gpio_configure(&fhdr);
```

```
bootloader_clock_configure()
uart_console_configure()
wdt_reset_check()
```

2. select_partition_number()

```
bootloader_utility__load_partition_table();
selected_boot_partition();
```

3. bootloader_utility_load_boot_image()

```
try_load_partition();                    //check app partition validity
    esp_image_load();
        bootloader_flash_read();
        bootloader_sha256_start();
        bootloader_sha256_data();
        verify_image_header();
        process_segment();
        /*Read app image header first.
          We get to know the number of segments we should load/map.
          Then we use cache to copy load segment to sram*/

load_image()
    esp_secure_boot_permanently_enable(); //Check Secure Boot
    esp_flash_encryption_enabled();       //Checking flash encryption

    unpack_load_app();                    //Copy sections to RAM, Set Cache for sections
        set_cache_and_start_app(addr_parameters);
            cache_flash_mmu_set();
            Cache_Ibus_MMU_Set();
            Cache_Dbus_MMU_Set();
            /*Use Cache_Ibus_MMU_Set and Cache_Dbus_MMU_Set to map instruction and rodata respectively. Then
              jump to third boot stage*/
```

Make FreeRTOS work

```
call_start_cpu0();
    memset(&_bss_start, 0, (&_bss_end - &_bss_start) * sizeof(_bss_start));

    rom_config_instruction_cache_mode();
    rom_config_data_cache_mode();

    Cache_Set_IDROM_MMU_Size();
    cpu_configure_region_protection();

    esp_spiram_init_cache();

    call_start_cpu1();
    heap_caps_init();
    start_cpu0();
```

```
start_cpu():
    ......
    xTaskCreatePinnedToCore(&main_task, "main",
                            SP_TASK_MAIN_STACK, NULL,
                            ESP_TASK_MAIN_PRIO, NULL, 0);
    ......
    vTaskStartScheduler()

main_task():
    ......
    app_main();
```

*Dual Core：

SoC复位后：PRO CPU立即运行，执行复位向量代码

APP CPU仍然保持复位状态

启动过程中：PRO CPU 会执行所有的初始化操作
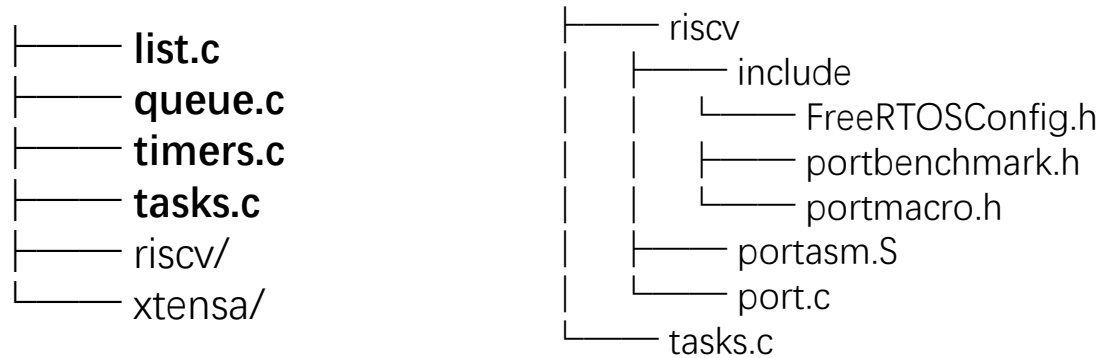
APP CPU 的被call_start_cpu1唤醒

Bring Up Steps

Make FreeRTOS work

- Config   *esp-idf/components/freertos/include/freertos/FreeRTOSConfig.h*

  #define config***: Configure API in FreeRTOS
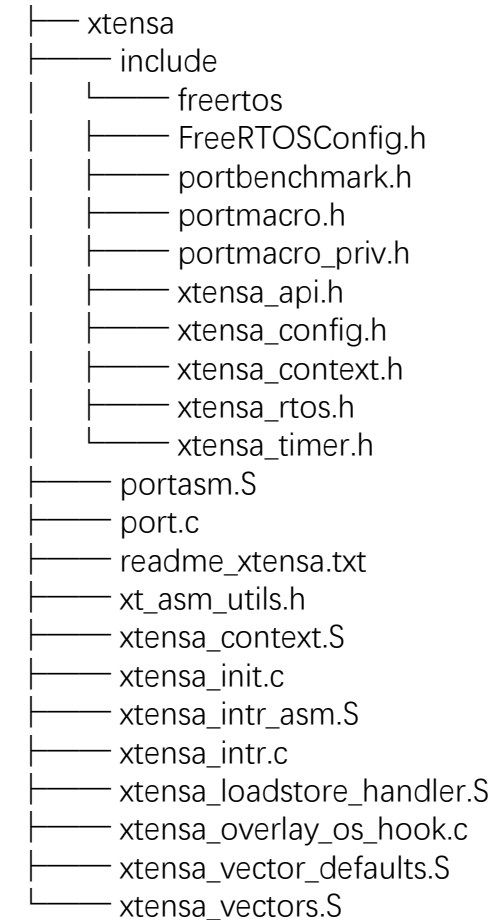 *#define include***: Enable or disable FreeRTOS*

- Structure   *esp-idf/components/freertos/*

```
├──── list.c                ├──── riscv
├──── queue.c               │    ├──── include
├──── timers.c              │    │    └──── FreeRTOSConfig.h
├──── tasks.c               │    │    ├──── portbenchmark.h
├──── riscv/                │    │    └──── portmacro.h
└──── xtensa/               │    ├──── portasm.S
                            │    └──── port.c
                            └──── tasks.c
```

- Implement functions defined in portable.h   *esp-idf/components/freertos/include/freertos/portable.h*

```
xPortStartScheduler()
vPortEndScheduler()
vPortYieldOtherCore()
vPortSetStackWatchpoint()
xPortInterruptedFromISRContext()
xPortGetTickRateHz()
...
```

- Check   *interrupt.csv*   with   *periph_defs.h*

```
├──── xtensa
├──── include
│    └──── freertos
│    ├──── FreeRTOSConfig.h
│    ├──── portbenchmark.h
│    ├──── portmacro.h
│    ├──── portmacro_priv.h
│    ├──── xtensa_api.h
│    ├──── xtensa_config.h
│    ├──── xtensa_context.h
│    ├──── xtensa_rtos.h
│    └──── xtensa_timer.h
├──── portasm.S
├──── port.c
├──── readme_xtensa.txt
├──── xt_asm_utils.h
├──── xtensa_context.S
├──── xtensa_init.c
├──── xtensa_intr_asm.S
├──── xtensa_intr.c
├──── xtensa_loadstore_handler.S
├──── xtensa_overlay_os_hook.c
├──── xtensa_vector_defaults.S
└──── xtensa_vectors.S
```

ESPRESSIF

/03 ROM Code Test (C3)

- Function
- Test

Function

ESPRESSIF

ROM Scheduler:

```
arch
├──── risc-v
└──── xtensa
```
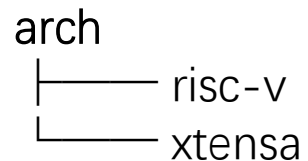
ROM API:

- Cache

- GPIO

- Newlib

- SPI/OPI_Flash

- UART

- USB

- ......

ROM Boot:

```
/*ets_main.c*/
main()
GET_CPUID    //获取CPUID
    APP_CPU:app_code_start()
    PRO_CPU:
            rtc_get_reset_reason()
        boot_prepare(reset_reson)
            Cache_Mask_All()        //Mask Cache
            uartAttach()            //Initialize uart（Without enable RX interrupt）
            ets_get_apb_freq()      //Get APB Bus Clock Frequency
            ...                     //Compatible Setting
        switch(reset_reason):
            DEEPSLEEP_RESET:
                user_code_start()
            Others:
                if(ETS_IS_FLASH_BOOT)
                    ets_flash_boot()
                else
                    if (ets_efuse_download_modes_disabled)
                        ETS_PRINTF_RESET("Download boot modes disabled\n")

                uartAttach(uart_rx_buff) //Initialize UART
                if(ets_efuse_security_download_modes_enabled)
                    ets_secure_download_mode()
                else
                    ets_contact_host()

ets_contact_host()
    UART_BOOT
    SDIO_BOOT
    SPI_DOWNLOAD_BOOT
    JOINT_DOWNLOAD_BOOT
```

ROM Test

ROM Scheduler Test:

- Main routine test

- Interrupt test

- Scheduler test

Switch[4 3 2 1]	Mode
1XXX	SPI Boot
01XX	UART Download
0001	SPI Download
0010	Diagnostic
0011	Test

ROM Boot Test:

SPI Flash boot modes:

- Fast SPI Flash Boot

- Legacy SPI Flash boot mode

RTC RESET
DEEP SLEEP RESET
DIGITAL RESET
CPU RESET

Download boot modes:

- Joint detect download boot modes test

- SDIO_Slave_v1.1 download boot mode

- Diagnostic download boot mode

- Security of download modes

/04 Cache Verification

- Structure
- Verification Method

Cache

A Cache Line

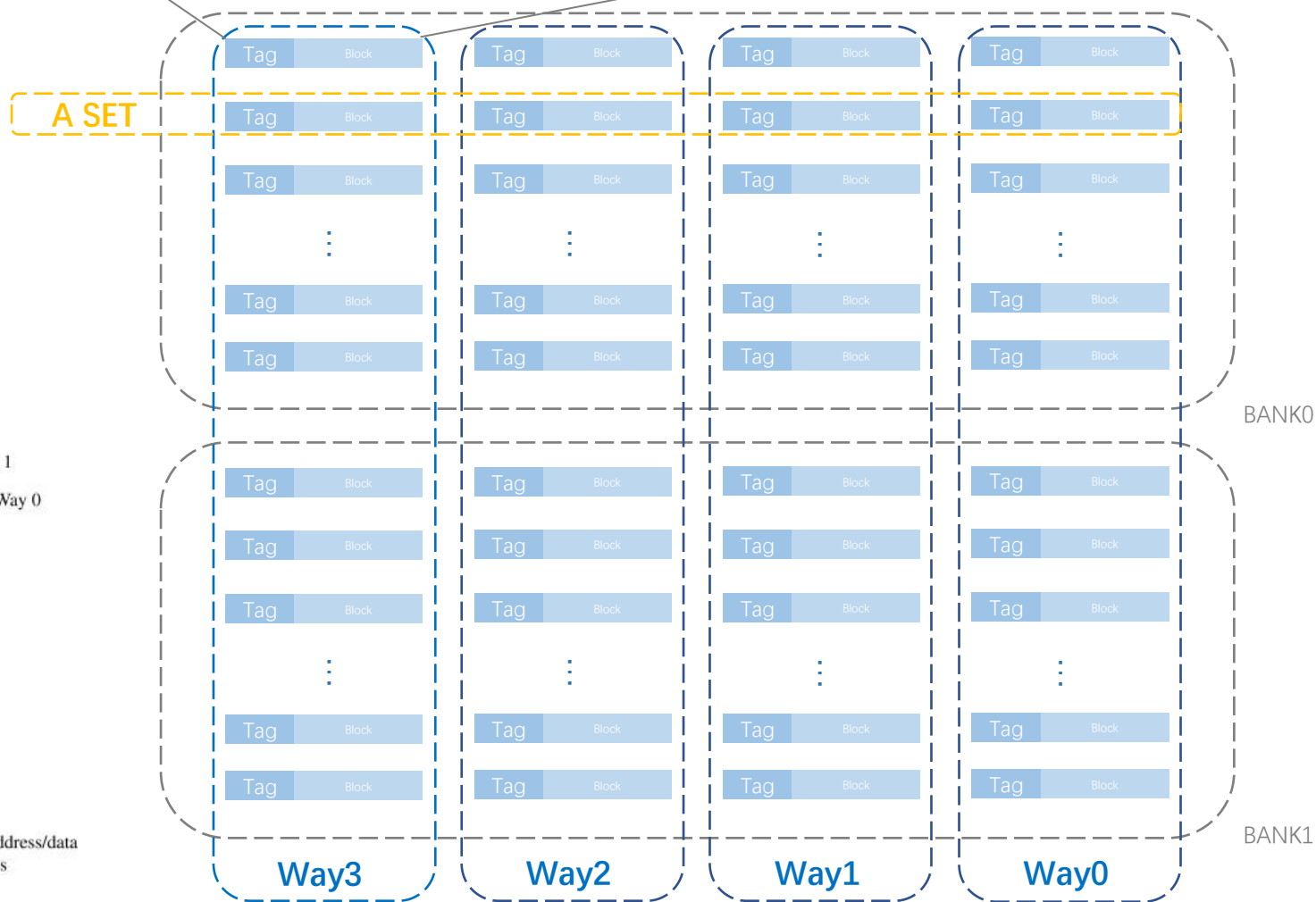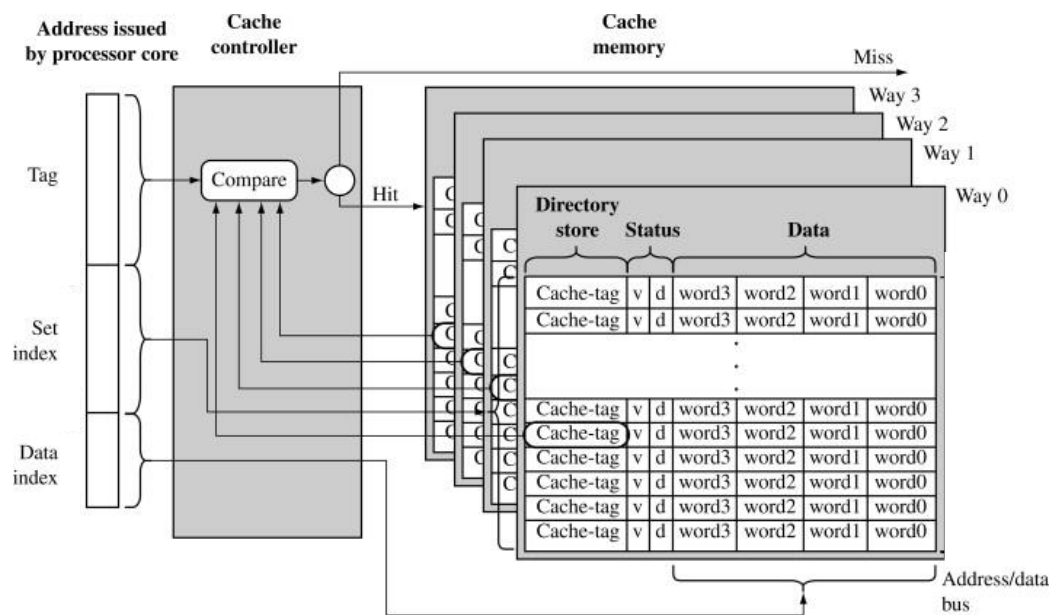| 10+ tag	10 9 count	8 evict	7~4 attribute	3 occupy	2 lock	1 valid	0 dirty	Set Index	Offset	Block

Cache Architecture:

Instruction cache size： 16KB / 32KB

Data cache size： 32KB / 64KB

Block Size： 16Bytes / 32Bytes / 64Bytes

Associated ways： 4way(Dcache/ICache) / 8way (ICache)

A SET

BANK0

BANK1

Way3   Way2   Way1   Way0

Cache Verification Methods

Cache address and access control test

1. IRam0 Data access test:

    • Use Cache/SPI to Read/Execute content/code in Flash/SPI RAM.

    • Use Cache/SPI read Content in whole Flash/SPI RAM.

2. DRam0 Data access test:

    • Use Cache/SPI Read/Write Content in whole SPI RAM.

    • Use Cache/SPI read Content in whole Flash.

Cache path test

4.3.1.1 Cache path integration test					
Instruction	Rodata	data	Priority	core0	core1
Flash	Flash	SPI ram	High		
Flash	SPI ram	SPI ram	High		
SPI ram	Flash	SPI ram	High		
SPI ram	SPI ram	SPI ram	High		
Flash	Flash	SPI ram , Flash (rodata)	High		

Cache Verification Methods

Cache mode test

1. **Instruction cache mode test:** Data cache parameters: size=16KB, block size=32Byte, set-associative=4

2. **Data cache mode test:** Data cache parameters: size=16KB, block size=32Byte, set-associative=4

4.4.1.1 Instruction cache mode test

ICache size(KB)	ICache ways	ICache block size(Bytes)	Priority	core0	core1
32	8	64	medium		
32	8	32	High		
32	8	16	High		
16	8	64	medium		
16	8	32	High		
16	8	16	High		
32	4	64	medium		
32	4	32	High		
32	4	16	High		
16	4	64	medium		
16	4	32	High		
16	4	16	High		

4.4.2.1 Data cache mode test

DCache size(KB)	DCache ways	DCache block size(Bytes)	Priority	core0	core1
64	8	64	medium		
64	8	32	High		
64	8	16	High		
32	8	64	medium		
32	8	32	High		
32	8	16	High		
64	4	64	medium		
64	4	32	High		
64	4	16	High		
32	4	64	medium		
32	4	32	High		
32	4	16	High		

3. **Cache Flash/SPI ram mode test:** Different Flash Mode, Flash clock divider, I/Dcache block size

4. **Cache SPI wrap mode test:** Different SPI ram clock divider, Flash clock divider, I/Dcache block size

5. **Cache mode integration test:** Different ICache size, DCache size, ICache block size, DCache block size

Cache Verification Methods

Cache operation test

1. ICache: Pre-lock, Lock/Unlock, Invalidation operation, time test

2. DCache: Lock/Unlock, Invalidation operation, clean/flush, time test

Cache replacement and write back policy test

1. ICache: replacement policy with 0/1/2/3/4 way locked,

   replacement policy with invalidate/lock & invalidate

2. Dcache: replacement policy with 0/1/2/3/4 way locked,

   replacement policy with invalidate/lock & invalidate

Cache preload & autoload test

1. ICache: autoload, preload

2. DCache: auto autoload, manually preload

3. Speed test

Cache Verification Methods

Cache encryption and decryption test

1. Cache encryption path test

2. Cache encryption SPI Flash mode test

3. Cache encryption SPI ram mode test

4. Cache encryption SPI wrap mode test

5. Cache encryption mode integration test

6. Encryption SPI write test

Cache miss delay test

1. Flash Cache miss delay

2. SPI ram Cache(without/with write back) miss delay

3. Dual core cache delay test

Cache error interrupt

MMU Tag and Cache memory utility

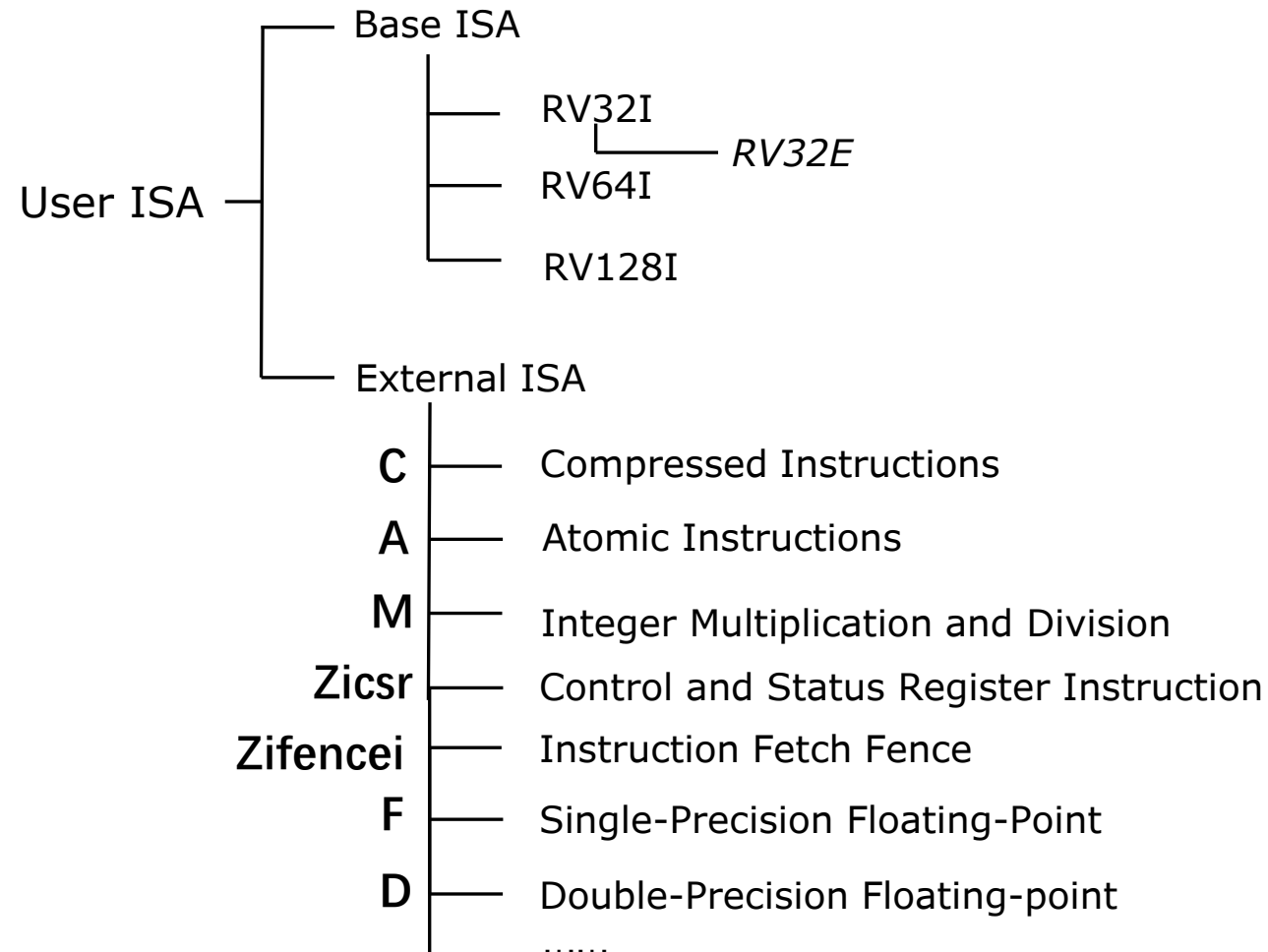**EDMA function test**

Cache & SPI arbiter test

Cache stress test

/05 RISC-V

- RISC-V CPU
- ESP32 C3 ROM Code & Boot

RISC-V CPU

ESPRESSIF

ISA

```
                    ┌─── Base ISA
                    │         ├──── RV32I
                    │         │        └──── RV32E
User ISA ───────────┤         ├──── RV64I
                    │         │
                    │         └──── RV128I
                    │
                    └─── External ISA

        C ──── Compressed Instructions

        A ──── Atomic Instructions

        M ──── Integer Multiplication and Division

     Zicsr ──── Control and Status Register Instruction

   Zifencei ──── Instruction Fetch Fence

        F ──── Single-Precision Floating-Point

        D ──── Double-Precision Floating-point

           ──── ......
```

RISC-V CPU

Privileged ISA

- *ABI: Application Binary Interface*
- *AEE: Application Execution Environment*
- *SBI: Supervisor Binary Interface*
- *SEE: Supervisor Execution Environment*
- *HBI: Hypervisor Binary Interface*
- *HEE: Hypervisor Execution Environment*

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	Hypervisor	H
3	11	Machine	M

| Number of levels	Supported Modes	Intended Usage
1	M	Simple embedded systems
2	M, U	Secure embedded systems
3	M, S, U	Systems running Unix-like operating systems
4	M, H, S, U	Systems running Type-1 hypervisors

RISC-V CPU

ESPRESSIF

31	0

x0 / zero	Hardwired zero
x1 / ra	Return address
x2 / sp	Stack pointer
x3 / gp	Global pointer
x4 / tp	Thread pointer
x5 / t0	Temporary
x6 / t1	Temporary
x7 / t2	Temporary
x8 / s0 / fp	Saved register, frame pointer
x9 / s1	Saved register
x10 / a0	Function argument, return value
x11 / a1	Function argument, return value
x12 / a2	Function argument
x13 / a3	Function argument
x14 / a4	Function argument
x15 / a5	Function argument
x16 / a6	Function argument
x17 / a7	Function argument
x18 / s2	Saved register
x19 / s3	Saved register
x20 / s4	Saved register
x21 / s5	Saved register
x22 / s6	Saved register
x23 / s7	Saved register
x24 / s8	Saved register
x25 / s9	Saved register
x26 / s10	Saved register
x27 / s11	Saved register
x28 / t3	Temporary
x29 / t4	Temporary
x30 / t5	Temporary
x31 / t6	Temporary

32

31	0
pc	

32

General register

Name	Description
sstatus	Supervisor status register
sie	Supervisor interrupt-enable register
stvec	Supervisor trap handler base address
sscratch	Scratch register for supervisor trap handlers
sepc	Supervisor exception program counter
scause	Supervisor trap cause
sbadaddr	Supervisor bad address
sip	Supervisor interrupt pending
sptbr	Page-table base register
mstatus	Machine status register
medeleg	Machine exception delegation register
mideleg	Machine interrupt delegation register
mie	Machine interrupt-enable register
mtvec	Machine trap-handler base address
mscratch	Scratch register for machine trap handlers
mepc	Machine exception program counter
mcause	Machine trap cause
mbadaddr	Machine bad address
mip	Machine interrupt pending

CSR

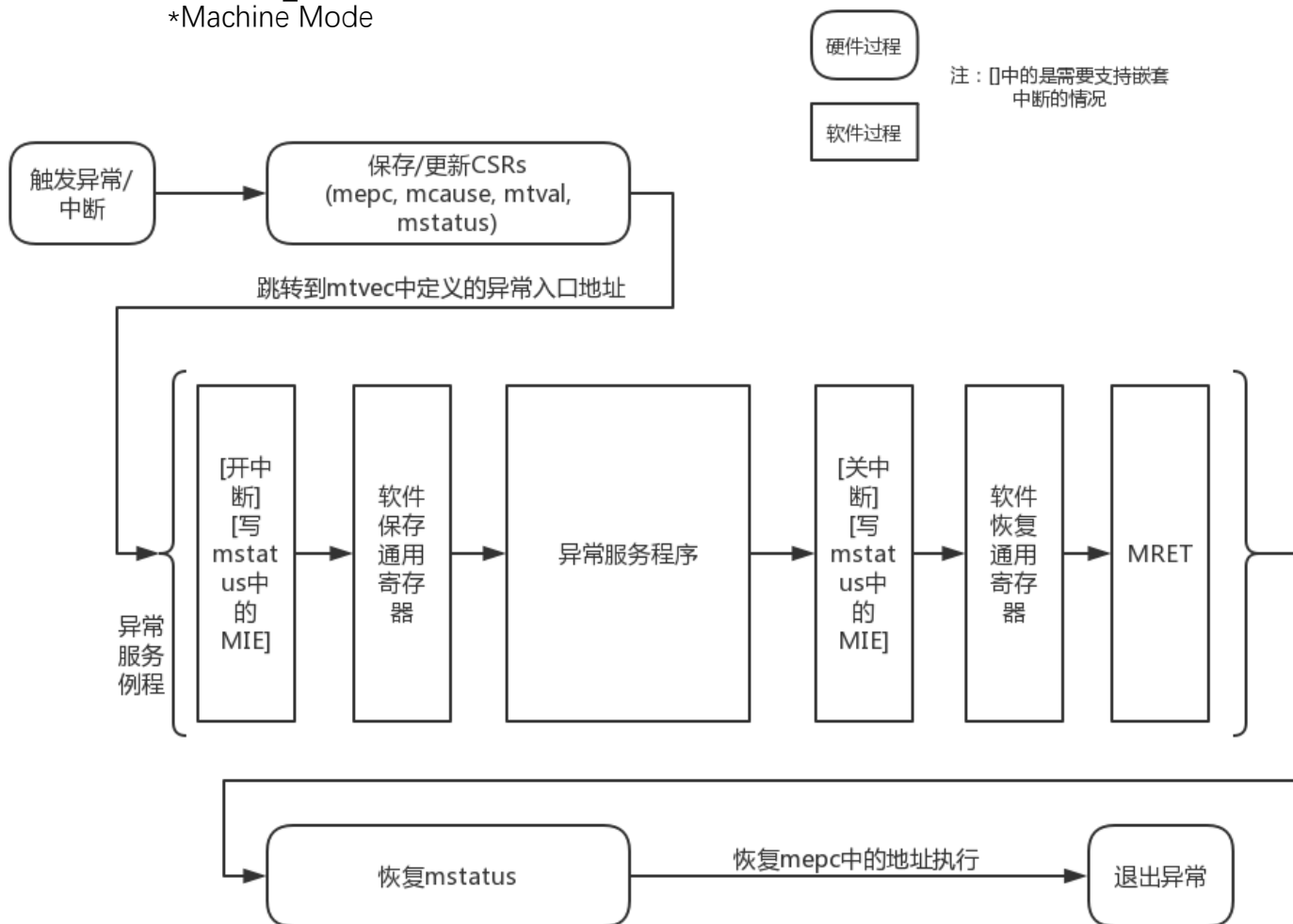# RISC-V CPU

Interrupts

RISC-V CPU

ESPRESSIF
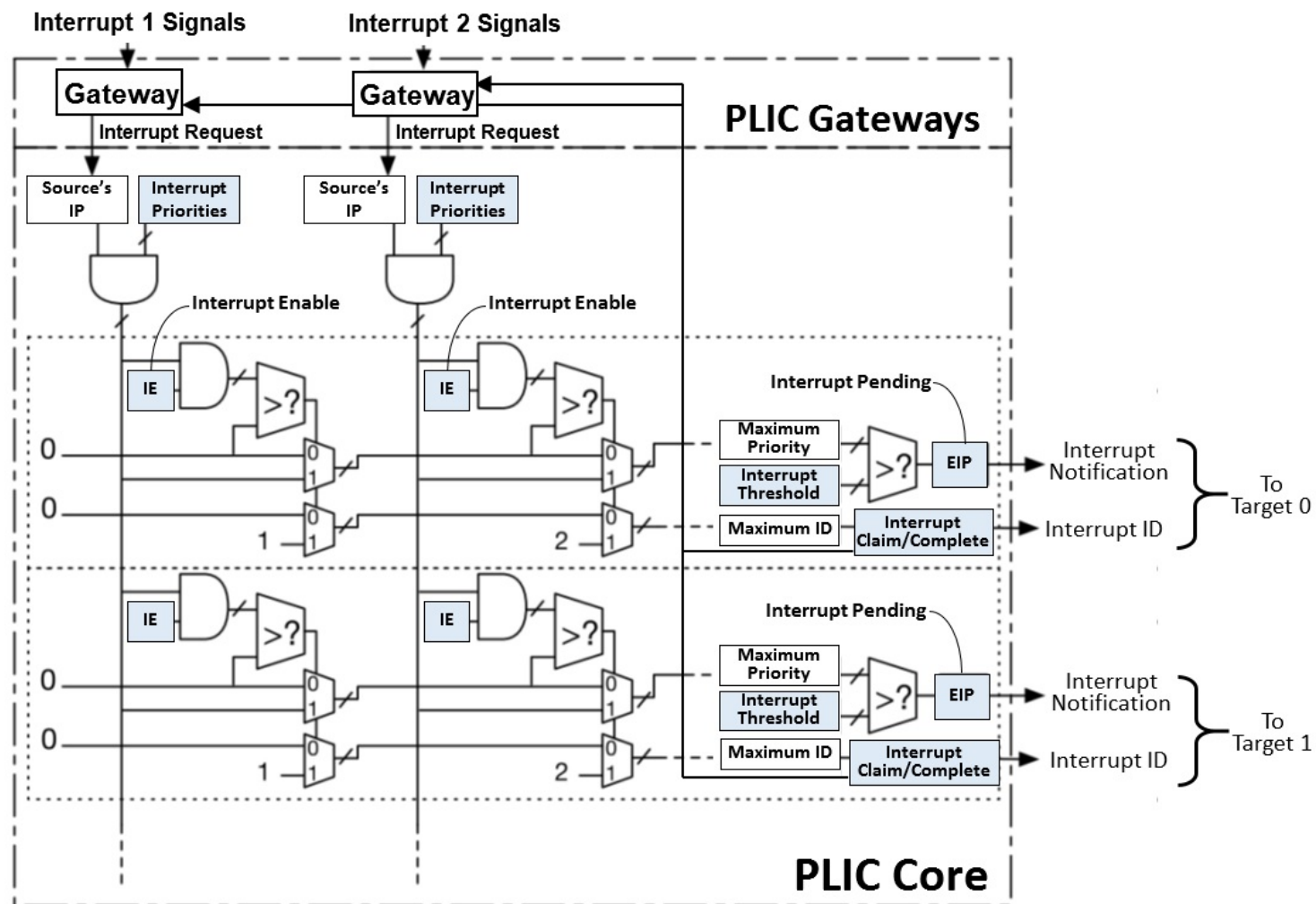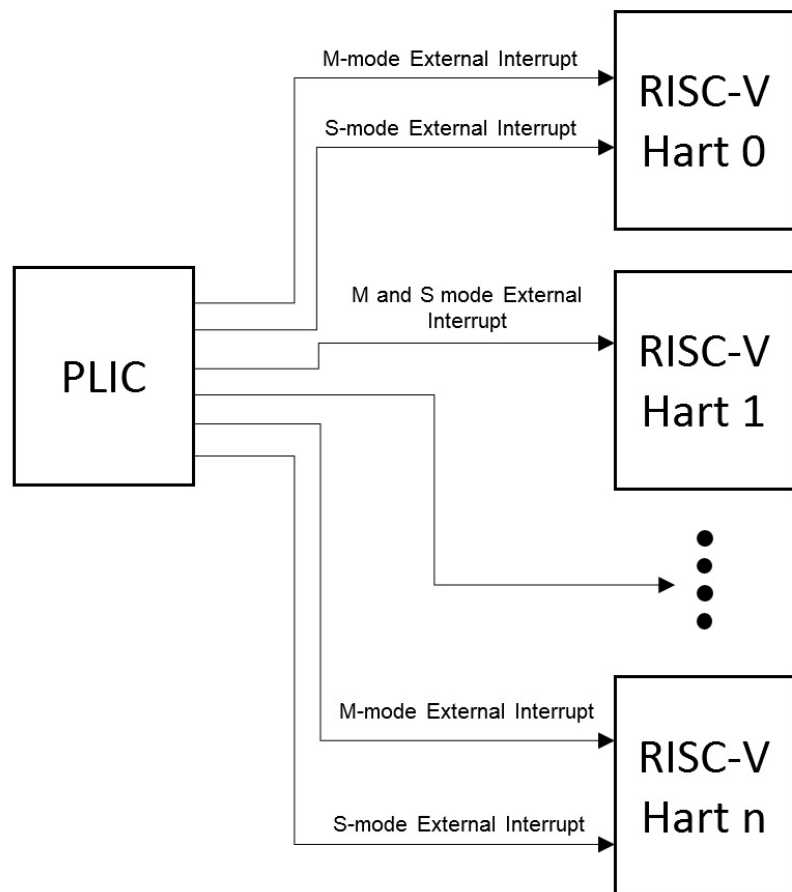
Exception
*Machine Mode

Enter the exception

1. **Update mepc:** If interrupt, write next instruction address to mepc

   If exception, write current instruction address to mepc

2. **Update mcause:** Update mcause according to the type of exception

3. **Update mtval:** Some exceptions need to write exception-related information into mtval

4. **Update mstatus:**

   Save the MIE to MPIE before the exception occurred

   Save the privilege level before the exception occurred in the MPP

   Update the value of MIE to 0

Exit the exception

1. Update mstatus

2. Execute from the address stored in mepc

Start.S

```asm
#include "riscv/csr.h"
#include "soc/interrupt_reg.h"
#include "soc/soc.h"
    .section    .startup.text,"ax"

    .align  4,0xff
    .global _start
_start:

    /* Mask all interrupts */
    csrw    mstatus, zero
    csrw    ustatus, zero
    /* Set interrupt enable to 0 */
    la  t0, INTERRUPT_CORE0_CPU_INT_ENABLE_REG
    sw  zero, (t0)

    /* Set performance mode to clock cycle count and enable*/
    li  t0, PCER_CYCLES
    csrw    pcer, t0
    li  t0, PCMR_GLOBAL_EN
    csrw    pcmr, t0

.option push
.option norelax

li      gp, 0
.option pop
```

```asm
    /* Set MTVEC */
    la  t0, _vector_table
    ori t0, t0, MTVEC_MODE_VECTORED
    csrw    mtvec, t0

    /* Set interrupt level to 1*/
    li  t1, 1
    la  t0, INTERRUPT_CORE0_CPU_INT_THRESH_REG
    sw  t1, (t0)

    /* Enable interrupts */
    li  t0, MSTATUS_MIE | MSTATUS_UIE
    li  t1, USTATUS_UIE
    csrw    mstatus, t0
    csrw    ustatus, t1

    /* init stack */
    la  sp, __stack

    /* init .data */
    la  a1, _data_start
    la  a2, _data_end
    ble a2, a1, 2f
```

```asm
unpackloop:
    lw  a3, 0(a1)           /* VMA of .data segment start */
    lw  a4, 4(a1)           /* VMA of .data segment end */
    lw  a5, 8(a1)           /* LMA of .data segment start */

    /*Copy .data from FLASH to RAM*/
1:  lw  t0, (a5)
    sw  t0, (a3)
    addi    a3, a3, 4
    addi    a5, a5, 4
    bltu    a3, a4, 1b

    addi    a1, a1, 16
    bltu    a1, a2, unpackloop

    /* init .bss */
2:  la  a1, _bss_start
    la  a2, _bss_end
    ble a2, a1, 2f
clearloop:
    lw      a3, 0(a1)           /* VMA of .bss segment start */
    lw      a4, 4(a1)           /* VMA of .bss segment end */

1:  sw  zero, (a3)
    addi    a3, a3, 4
    blt a3, a4, 1b

    addi    a1, a1, 12
    bltu    a1, a2, clearloop
2:

    tail    main
    /* Never returns here */

    .size   _start, . - _start
```

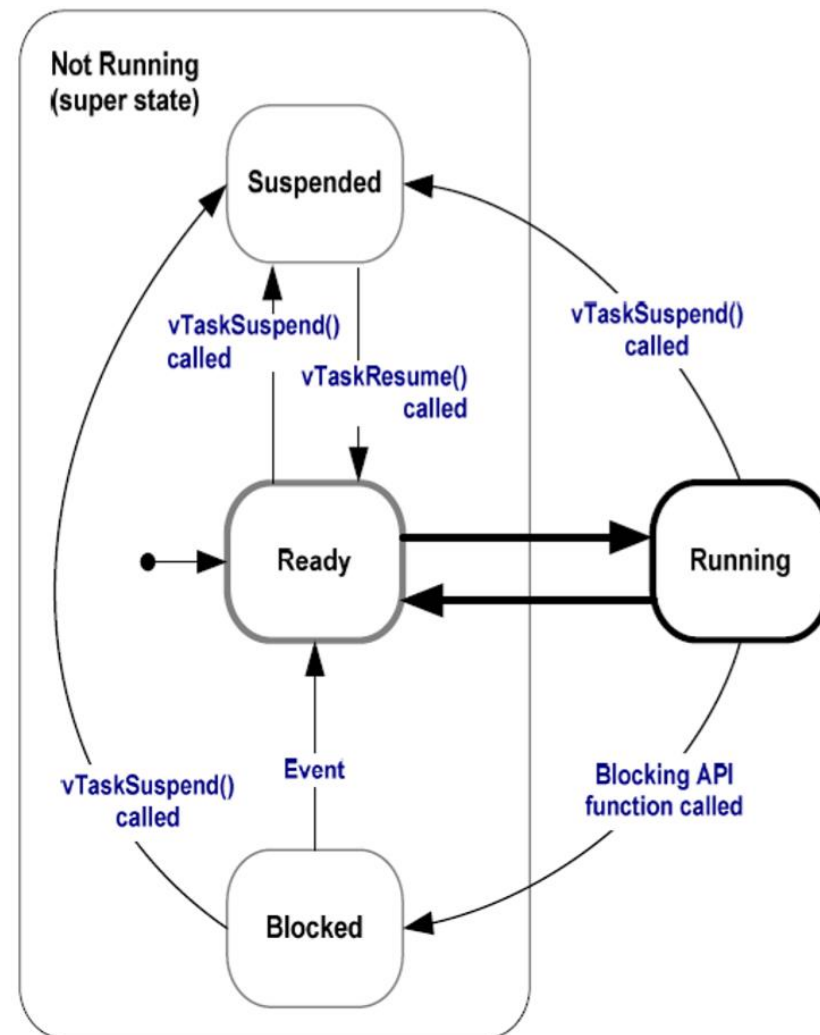# /06 FreeRTOS

Task

```c
void example_task(void *pvParameters)
{
    //Initialize local variables here

    while (1) {
        //Task main loop
    }

    //Delete task here
    vTaskDelete(NULL);
}
```
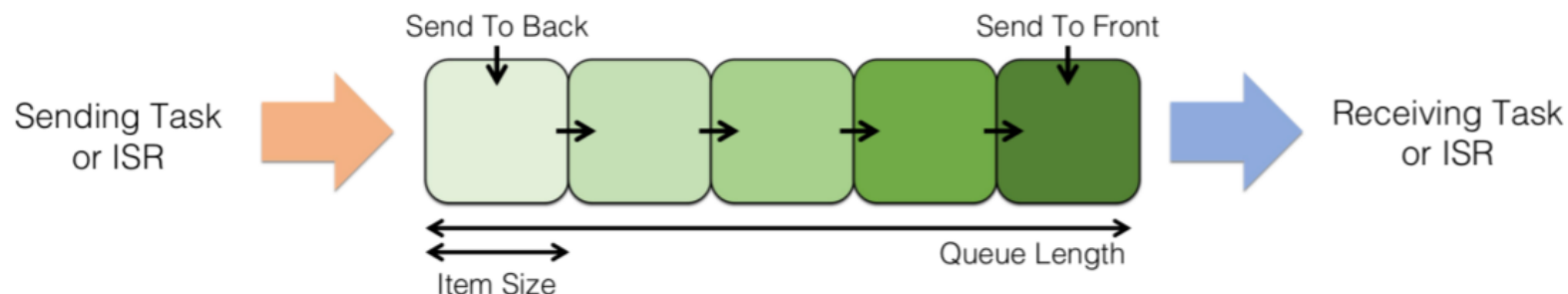
- A task function must be a function that accepts a void pointer parameter and returns void.

- User can use the parameter as a pointer to a structure containing parameter values, or cast the pointer directly as a value.
- Main body of task is usually implemented as an infinite loop.

- To prevent the loop constantly consuming CPU time, a task will usually **block** (calling a delay or waiting for an event).

- Breaking out of the loop usually means the task has run to completion.
- A task should **never return or execute past the end of the function**, doing so will likely lead to an unrecoverable error.

- A task should exit by deleting itself.

```c
xTaskCreate( vTask1,        /* 指向任务函数的指针 */

             "Task 1",      /* 任务的文本名字，只会在调试中用到 */

             1000,          /* 栈深度 – 大多数小型微控制器会使用的值会比此值小得多 */

             NULL,          /* 没有任务参数 */

             1,             /* 此任务运行在优先级1上. */

             NULL );        /* 不会用到任务句柄 */
```
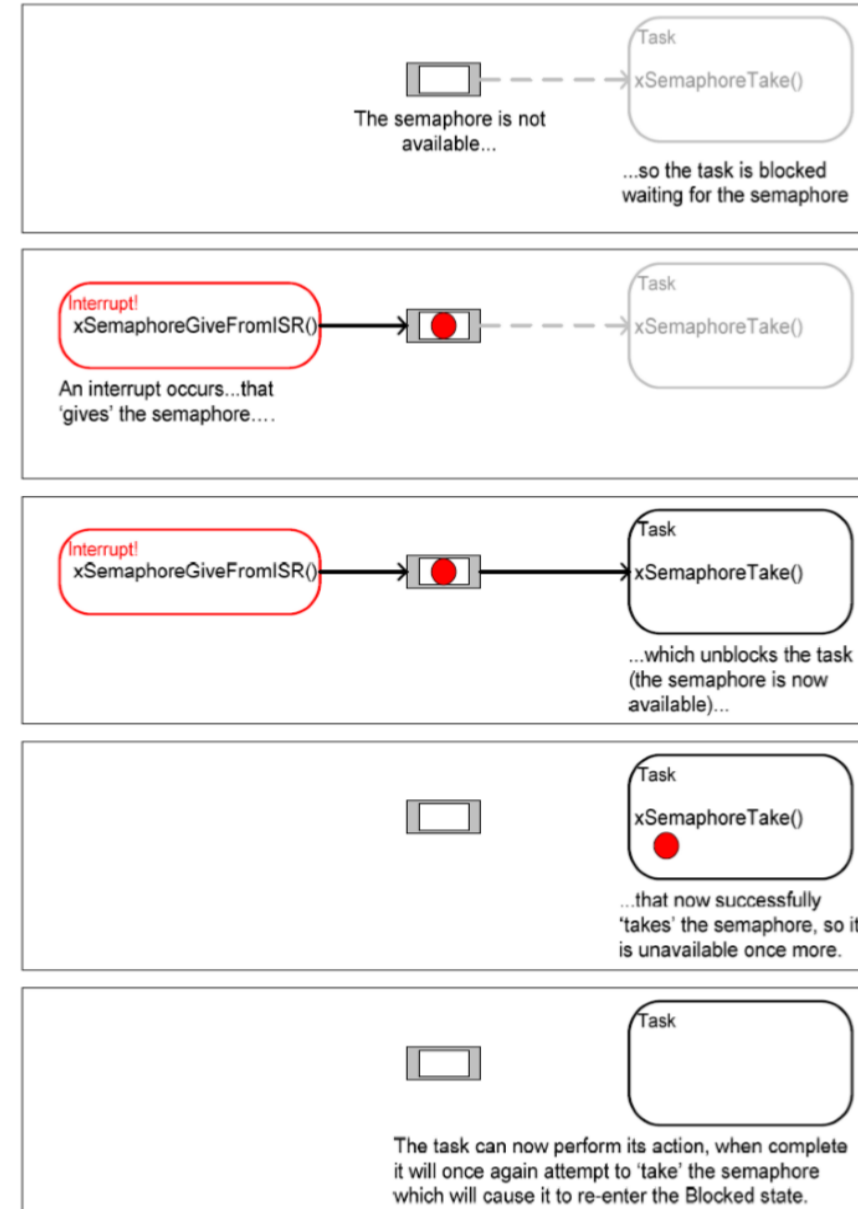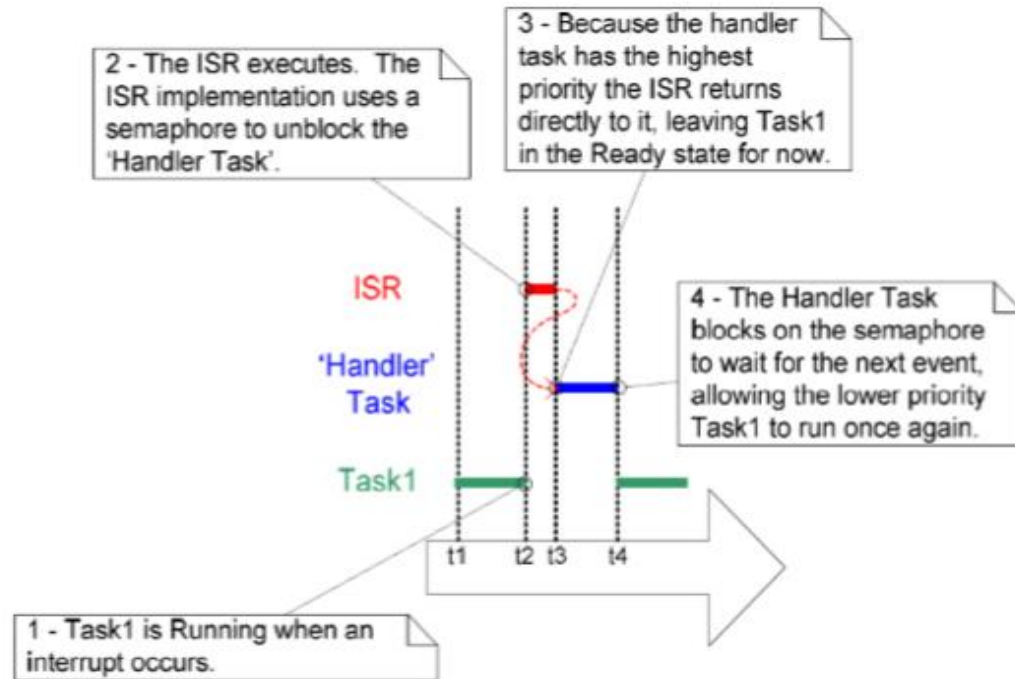
FreeRTOS API

Queue

Send To Back   Send To Front

Sending Task or ISR → [queue items] → Receiving Task or ISR

Item Size

Queue Length

Sending Functions	
xQueueSendToBack() xQueueSendToBackFromISR()	Copy an item to the back of the queue from a task/ISR. ISR version will not block.
xQueueSendToFront() xQueueSendToFrontFromISR()	Copy an item to the front of the queue from a task/ISR. ISR version will not block.
xQueueOverwrite() xQueueOverwriteFromISR()	Copy and item to the back of the queue from a task/ISR but overwrite if there is not space available. Only call on queues of length one.

Receiving Functions	
xQueueReceive() xQueueReceiveFromISR()	Receive and remove an item from the front of the queue. ISR version will not block.
xQueuePeek() xQueuePeekFromISR()	Receive but don't remove an item from the front of the queue. ISR version will not block.

Utility Functions	
xQueueCreate()	Creates a queue
vQueueDelete()	Deletes a queue. Only call when no tasks are blocked on the queue.

FreeRTOS API

![Espressif logo]

Semaphore

2 - The ISR executes. The ISR implementation uses a semaphore to unblock the 'Handler Task'.

3 - Because the handler task has the highest priority the ISR returns directly to it, leaving Task1 in the Ready state for now.

4 - The Handler Task blocks on the semaphore to wait for the next event, allowing the lower priority Task1 to run once again.

1 - Task1 is Running when an interrupt occurs.

ISR

'Handler' Task

Task1

t1 t2 t3 t4

The semaphore is not available...

Task
xSemaphoreTake()

...so the task is blocked waiting for the semaphore

Interrupt!
xSemaphoreGiveFromISR()

Task
xSemaphoreTake()

An interrupt occurs...that 'gives' the semaphore....

Interrupt!
xSemaphoreGiveFromISR()

Task
xSemaphoreTake()

...which unblocks the task (the semaphore is now available)...

Task
xSemaphoreTake()

...that now successfully 'takes' the semaphore, so it is unavailable once more.

Task

The task can now perform its action, when complete it will once again attempt to 'take' the semaphore which will cause it to re-enter the Blocked state.

Critical

```
taskENTER_CRITICAL()
     ......
taskEXIT_CRITICAL()
```

Mutex

FreeRTOS Kernel
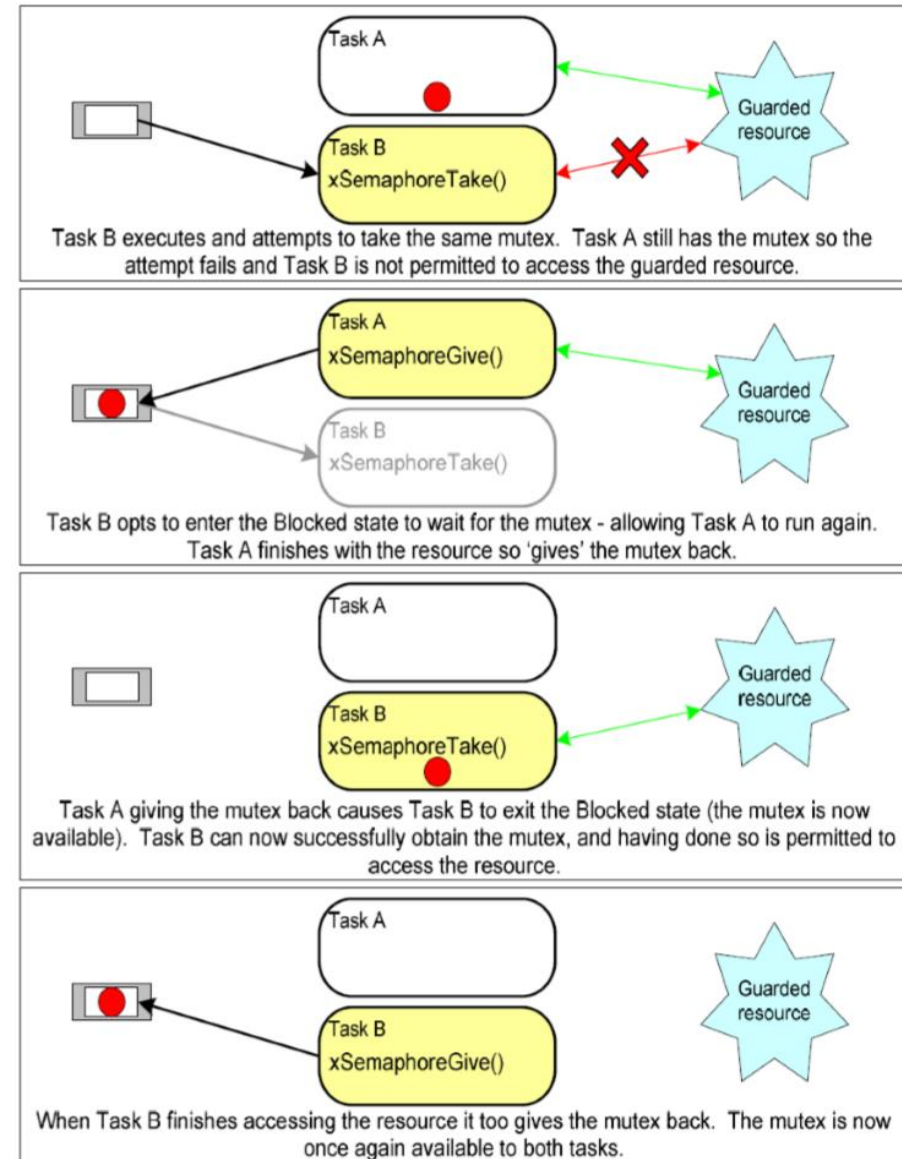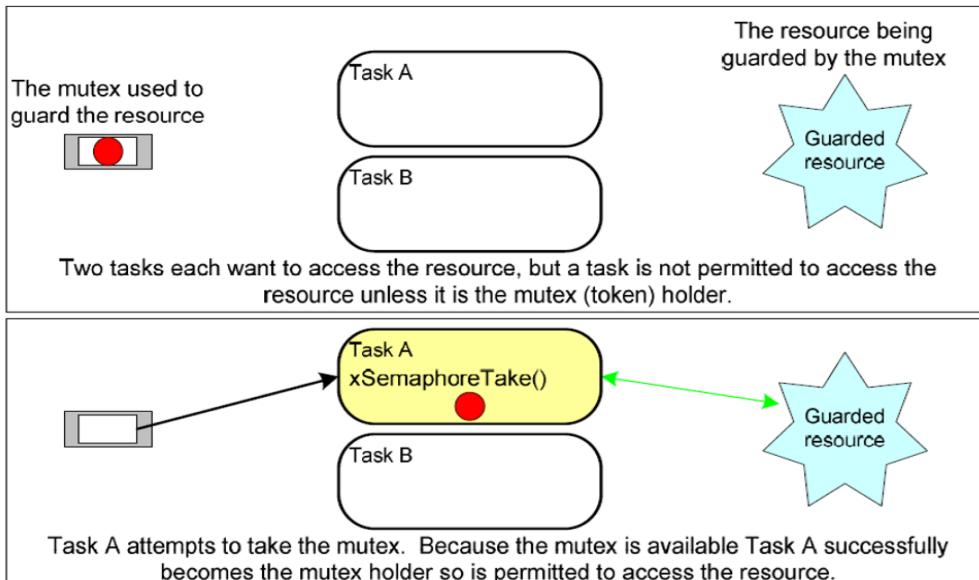
Task   *esp-idf/components/freertos/tasks.c*

- TCB (Task Control Block)

```
typedef struct tskTaskControlBlock
{
    volatile portSTACK_TYPE    *pxTopOfStack;    //指向任务堆栈结束处
    xListItem        xGenericListItem;              //用于把TCB插入就绪链表或等待链表
    xListItem        xEventListItem;                //用于把TCB插入事件链表（如消息队列）
    unsigned portBASE_TYPE    uxPriority;        //任务优先级
    portSTACK_TYPE            *pxStack;           //指向任务堆栈起始处
    signed portCHAR            pcTaskName[ configMAX_TASK_NAME_LEN ]; //任务名称
    unsigned portCHAR            uxTCBNumber;  //用于记录功能
}
```

- 任务堆栈

```
#define portSTACK_TYPE uint32_t
typedef portSTACK_TYPE StackType_t
```

- xTaskCreate()

- xTaskCreatStatic()

- xTaskCreateRestricted()

- vTaskDelete()

- vTaskSuspend()

- vTaskResume()

- vTaskResumeFromISR()

- portENABLE_INTERRUPTS()

- portDISABLE_INTERRUPTS()

- ......

FreeRTOS Kernel

List

`esp-idf/components/freertos/list.c`

```
typedef struct xLIST
{
    configLIST_VOLATILE UBaseType_t uxNumberOfItems;
    ListItem_t * configLIST_VOLATILE pxIndex;
    MiniListItem_t xListEnd;
} List_t;
```

xList_ITEM & xMINI_LIST_ITEM

```
typedef struct xMINI_LIST_ITEM
{
    configLIST_VOLATILE TickType_t xItemValue;
    struct xLIST_ITEM * configLIST_VOLATILE pxNext;
    struct xLIST_ITEM * configLIST_VOLATILE pxPrevious;
    void * pvOwner;
    void * configLIST_VOLATIE pvContainer;
} List_t;
```

- vListInitialise()
- vListInitialiseItem()
- vListInsert()
- vListInsertEnd()
- uxListRemove()
- listGET_OWNER_OF_NEXT_ENTRY()

FreeRTOS Kernel
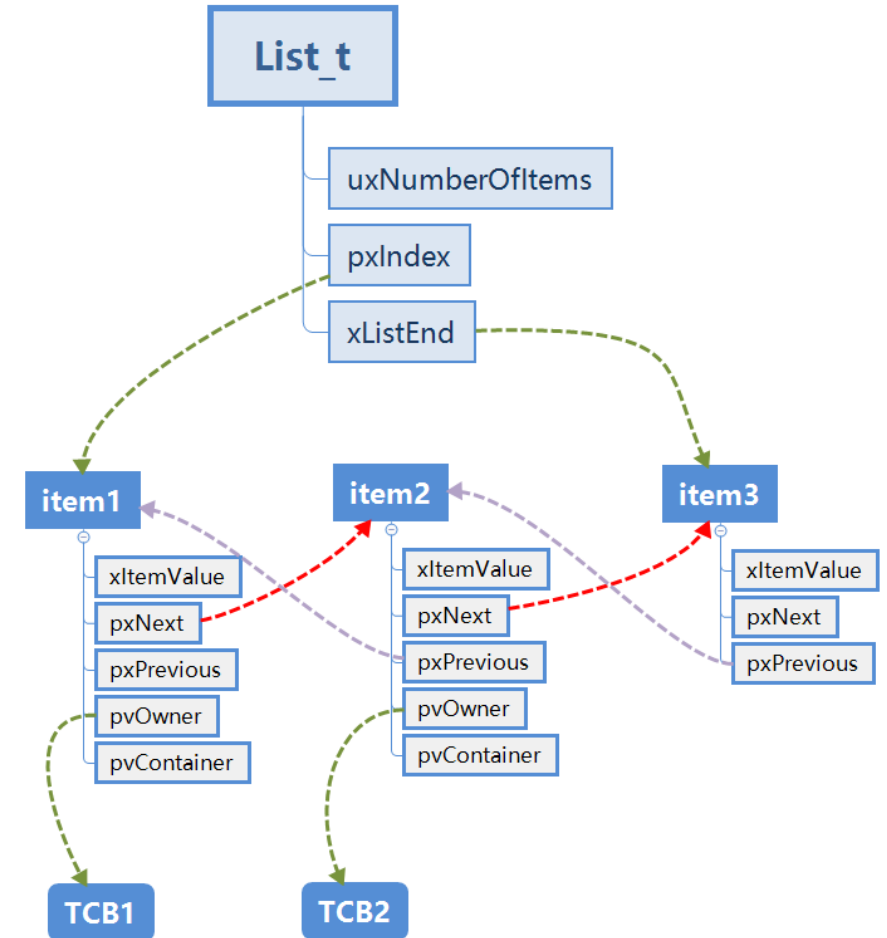
xTaskCreatePinnedToCore

```
BaseType_t  xTaskCreatePinnedToCore(  TaskFunction_t pxTaskCode,          //函数指针，指向任务函数的入口
                                      const char  *  const  pcName,        //任务描述，用于调试
                                      const uint32_t usStackDepth,         //任务堆栈大小
                                      void  *  const  pvParameters,        //传递任务参数指针
                                      UBaseType_t  uxPriority,             //任务优先级
                                      TaskHandle_t  *  const  pxCreatedTask,  //任务句柄
                                      const  BaseType_t  xCoreID）         //指定核
```

vTaskCreatePinnedToCore
- 创建任务堆栈 pvPortMallocStackMem() ①
- 创建任务TCB pvPortMallocTcbMem() ①
- 任务堆栈位置赋给TCB pxNewTCB->pxStack = pxStack
- 初始化Task及其TCB必要字段 prvInitialiseNewTask() ⑤
- 将新Task添加进就绪列表 ⑪

vTaskCreate Flow Chart

Scheduler *esp-idf/components/freertos/tasks.c*

- vTaskStartScheduler()

判断需不需要通过钩子函数得到空闲任务的句柄,静态创建空闲任务 — 通过钩子函数得到空闲任务的内存分配的信息,然后静态创建空闲任务 / 否则,动态创建空闲任务

VTaskStartScheduler

判断是不是开启了软件定时器 — 创建定时器任务 xTimerCreateTimerTask()

若执行正常 — 关闭全局中断 portDISABLE_INTERRUPTS()

把调度器的运行状态修改为运行态

开启调度器的函数 xPortStartScheduler()
- 设置并使能Systick中断 vPortSerupTimer() ⑤
- 设置软件中断 vPortSetupSoftwareInterrupt() ④
- 设置中断优先级阈值 esprv_intc_int_set_threshold(priority_threshold) — 低于此阈值中断被屏蔽
- 全局中断使能 riscv_global_interrupts_enable()
- 转让CPU使用权 vPortYield() ㉜

- vTaskSwitchContext

判断任务调度有没有挂起 — 挂起了就不能进行任务切换了

修改标志位为没有挂起

有没有使能任务信息统计 — 用总运行时间-上一次任务切换进来时的定时器时间,算出两次切换之间的时间间隔 / 更新上一次任务切换的时间为当前任务运行的总时间

检测堆栈溢出功能

选择下一个任务taskSELECT_HIGHEST_PRIORITY_TASK
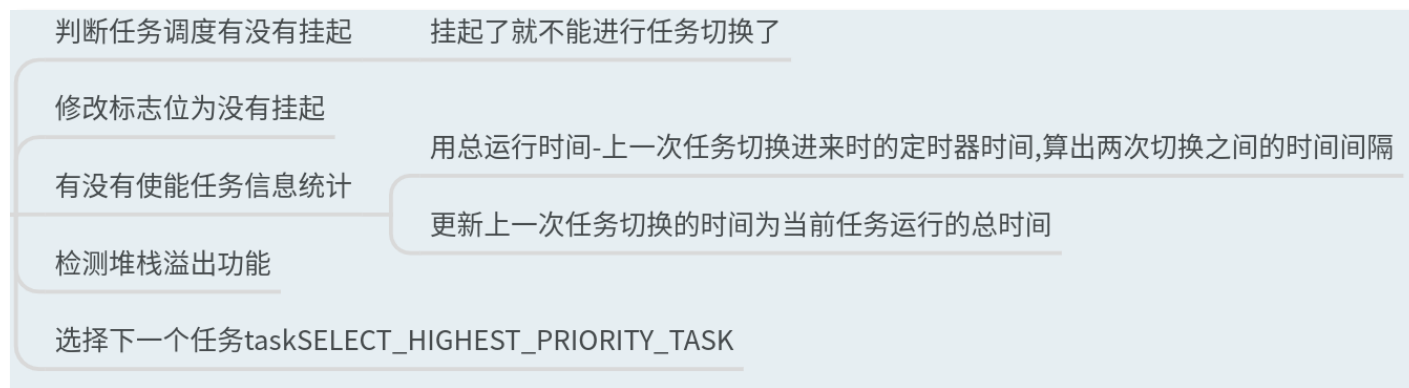
- xPortStartScheduler()
  - vPortSetupTimer
  - vPortSetupSoftwareInterrupt()
  - esprv_intc_int_set_threshold(1)
  - riscv_global_interrupts_enable()
  - vPortYield()
    - *REG_WRITE(SYSTEM_CPU_INTR_FROM_CPU_0_REG, 1)*

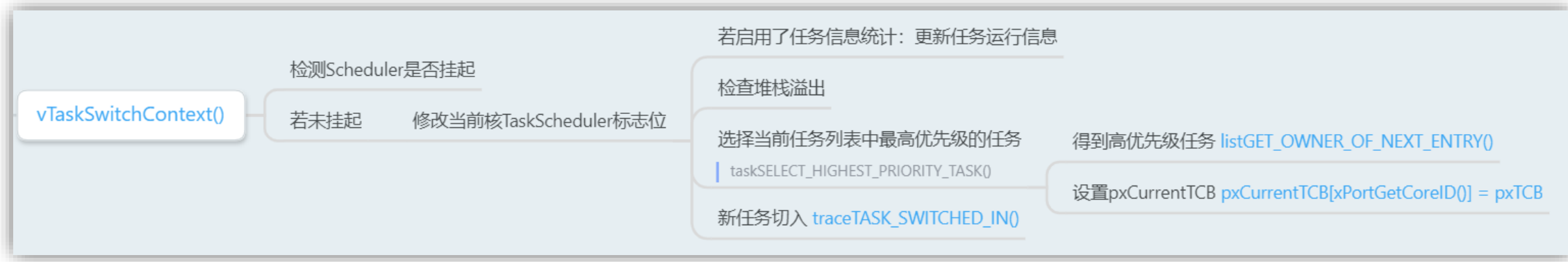[vTaskStartScheduler FlowChart](#)

FreeRTOS Kernel

TaskSwitching

- 执行系统调用切换

  taskYIELD() → portYIELD() → vPortYield() → 检查中断嵌套→ 触发软件中断 → vPortSoftwareInterrupt → vTaskSwitchContext()

- Systick定时器中断切换

  触发定时器中断 → vPortSysTickHandler() → vTaskSwitchContext()

IdleTask

- 当任务自己删除自己的时候,需要在空闲任务中释放掉任务申请的任务控制块和任务堆栈等内存
- 运行用户设置的空闲任务钩子函数
- 判断是否开启低功耗tickless模式,
- 用户可以创建与空闲任务优先级相同的应用任务，当宏configIDLE_SHOULD_YIELD为1时，空闲任务会让出时间片给相同优先级的应用任务。
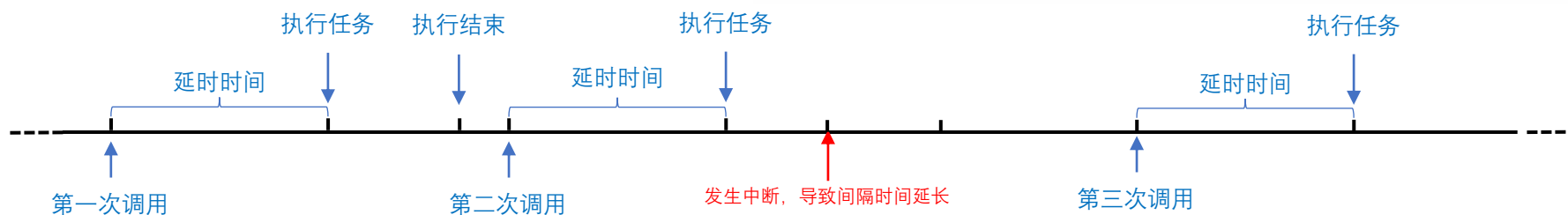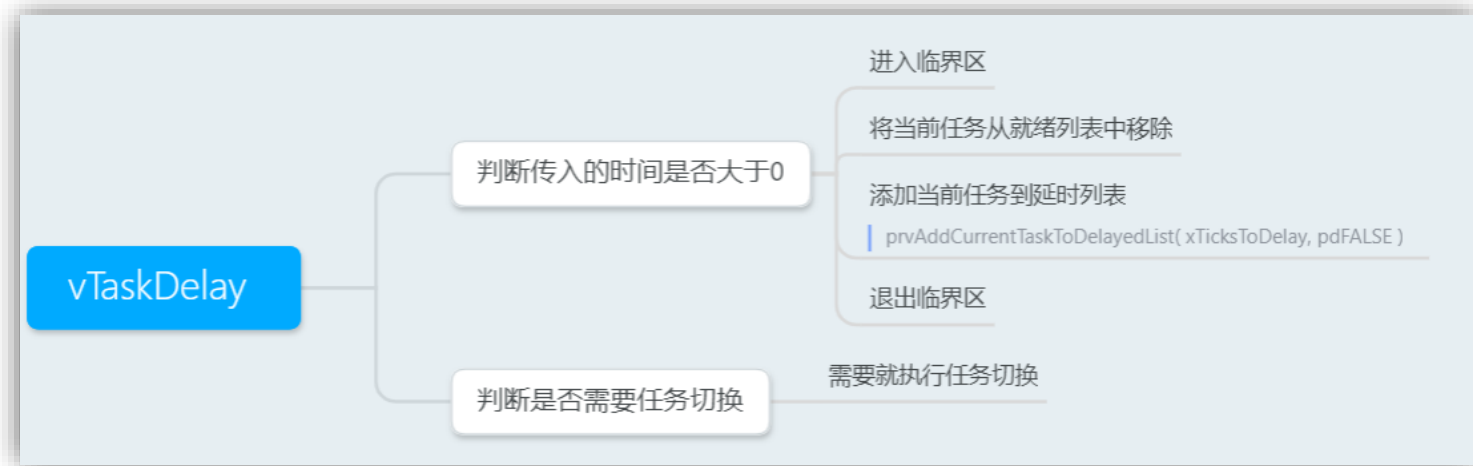
FreeRTOS Time management

- xTaskIncrementTick()

  xTickCount是FreeRTOS的系统节拍计数器，每个滴答定时器中断后加一

  xTickCount的具体操作是在函数xTaskIncrementTick中进行的

  - 系统节拍计数器的值加1

  - 判断是否有任务的延时等待时间已到，如果就就将其恢复

  - 处理时间片调度

- vTaskDelay(xTicksToDelay)

  - 相对于其调用者函数而言的延时

  - 不确保周期性

进入临界区

将当前任务从就绪列表中移除

添加当前任务到延时列表

prvAddCurrentTaskToDelayedList(xTicksToDelay, pdFALSE)

退出临界区

判断传入的时间是否大于0

vTaskDelay

判断是否需要任务切换 — 需要就执行任务切换

执行任务　执行结束　　执行任务　　　　　　　　执行任务

延时时间　　　　　　　延时时间　　　　　　　延时时间

第一次调用　　　　　第二次调用　　发生中断，导致间隔时间延长　　第三次调用

FreeRTOS Kernel

- vTaskDelayUntill(pxPreviousWakeTime, xTimIncrement)
  - 主调函数+延时任务的时间是确定的,可以实现周期性
  - 添加的延时值 = 唤醒时间点xTimeToWake - 当前时间值 xConstTickCount(动态变化的,主体任务变化而变化)
  - pxPreviousWakeTime 保存上一次该任务被唤醒的时刻值, xTimeIncrement是需要延时的时间
  - 首次调用,需要将pxPreviousWakeTime初始化为 进入while循环体的事件点的值
  - 在后续的运行中,任务会自动更新这个pxPreviousWakeTime

vTaskDelayUntil

- 记录下当前系统时间 xConstTickCount
- 计算唤醒时间xTimeToWake = pxPreviousWakeTime上一次唤醒的时间点+xTimeIncrement需要延时的时间
- 如果当前进入时刻xConstTickCount溢出
  [当前进入时刻xConstTickCount < *pxPreviousWakeTime上一次唤醒的时间点]
- 更新上一次的延时时间 *pxPreviousWakeTime = xTimeToWake;
- 判断要不要延时
  - 当前任务从就绪列表移除
  - 当前任务添加到延时列表 prvAddCurrentTaskToDelayedList
- 有没有任务切换 — 任务切换

执行指定任务　延时时间　执行指定任务　延时时间　执行指定任务

第一次调用　第二次调用　发生中断 但不影响指定任务下次唤醒时间　第三次调用