RISC-V MCU启动文件分析

原创 傲过风景 于 2022-06-14 09:18:21 发布 255 收藏 4

文章标签： stm32 arm 嵌入式硬件

启动文件由汇编语言编写，是MCU上电复位后第一个执行的程序，主要执行以下内容：

- 初始化**gp**(**g**lobal **p**ointer)全局指针寄存器、**sp**(**s**tack **p**ointer)栈指针寄存器
- 将data数据从flash中加载至RAM中
- 清空bss段数据
- 初始化中断向量表
- 配置系统时钟
- 从Machine模式切换到User模式，进入main函数运行

CH32V103启动文件如下：

```
/********************************* (C) COPYRIGHT *******************************
* File Name          : startup_ch32v10x.s
* Author             : WCH
* Version            : V1.0.0
* Date               : 2020/04/30
* Description        : CH32V10x vector table for eclipse toolchain.
*******************************************************************************/

    .section    .init,"ax",@progbits     /* 声明section 为 .init */
    .global    _start                    /* 指明标签_start的属性为全局性的 */
    .align    1
_start:                                  /* 标签_start处 */
    j    handle_reset                    /* 跳转至 handle_reset处 */
    .word 0x00000013                     /* 内核设计需要，不用关注 */
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000013
    .word 0x00000073
    .section    .vector,"ax",@progbits
    .align  1
_vector_base:                            /* 中断向量表  */
    .option norvc;
        j   _start
    .word   0
        j   NMI_Handler                  /* NMI Handler */
        j   HardFault_Handler            /* Hard Fault Handler */
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
        j   SysTick_Handler              /* SysTick Handler */
    .word   0
        j   SW_handler                   /* SW Handler */
    .word   0
    /* External Interrupts */
        j   WWDG_IRQHandler              /* Window Watchdog */
        j   PVD_IRQHandler               /* PVD through EXTI Line detect */
        j   TAMPER_IRQHandler            /* TAMPER */
        j   RTC_IRQHandler               /* RTC */
        j   FLASH_IRQHandler             /* Flash */
        j   RCC_IRQHandler               /* RCC */
        j   EXTI0_IRQHandler             /* EXTI Line 0 */
        j   EXTI1_IRQHandler             /* EXTI Line 1 */
        j   EXTI2_IRQHandler             /* EXTI Line 2 */
        j   EXTI3_IRQHandler             /* EXTI Line 3 */
        j   EXTI4_IRQHandler             /* EXTI Line 4 */
        j   DMA1_Channel1_IRQHandler     /* DMA1 Channel 1 */
        j   DMA1_Channel2_IRQHandler     /* DMA1 Channel 2 */
        j   DMA1_Channel3_IRQHandler     /* DMA1 Channel 3 */
        j   DMA1_Channel4_IRQHandler     /* DMA1 Channel 4 */
        j   DMA1_Channel5_IRQHandler     /* DMA1 Channel 5 */
        j   DMA1_Channel6_IRQHandler     /* DMA1 Channel 6 */
        j   DMA1_Channel7_IRQHandler     /* DMA1 Channel 7 */
        j   ADC1_2_IRQHandler            /* ADC1_2 */
        .word   0
        .word   0
        .word   0
        .word   0
        j   EXTI9_5_IRQHandler           /* EXTI Line 9..5 */
        j   TIM1_BRK_IRQHandler          /* TIM1 Break */
        j   TIM1_UP_IRQHandler           /* TIM1 Update */
        j   TIM1_TRG_COM_IRQHandler      /* TIM1 Trigger and Commutation */
        j   TIM1_CC_IRQHandler           /* TIM1 Capture Compare */
        j   TIM2_IRQHandler              /* TIM2 */
        j   TIM3_IRQHandler              /* TIM3 */
        j   TIM4_IRQHandler              /* TIM4 */
        j   I2C1_EV_IRQHandler           /* I2C1 Event */
        j   I2C1_ER_IRQHandler           /* I2C1 Error */
        j   I2C2_EV_IRQHandler           /* I2C2 Event */
        j   I2C2_ER_IRQHandler           /* I2C2 Error */
        j   SPI1_IRQHandler              /* SPI1 */
        j   SPI2_IRQHandler              /* SPI2 */
        j   USART1_IRQHandler            /* USART1 */
        j   USART2_IRQHandler            /* USART2 */
        j   USART3_IRQHandler            /* USART3 */
        j   EXTI15_10_IRQHandler         /* EXTI Line 15..10 */
        j   RTCAlarm_IRQHandler          /* RTC Alarm through EXTI Line */
        j   USBWakeUp_IRQHandler         /* USB Wakeup from suspend */
        j   USBHD_IRQHandler             /* USBHD */

    .option rvc;

    .section    .text.  , "ax", @progbits     /* 中断服务程序宿定义  */
    .weak   NMI_Handler
    .weak   HardFault_Handler
    .weak   SysTick_Handler
    .weak   SW_handler
    .weak   WWDG_IRQHandler
    .weak   PVD_IRQHandler
    .weak   TAMPER_IRQHandler
    .weak   RTC_IRQHandler
    .weak   FLASH_IRQHandler
    .weak   RCC_IRQHandler
    .weak   EXTI0_IRQHandler
    .weak   EXTI1_IRQHandler
    .weak   EXTI2_IRQHandler
    .weak   EXTI3_IRQHandler
    .weak   EXTI4_IRQHandler
    .weak   DMA1_Channel1_IRQHandler
    .weak   DMA1_Channel2_IRQHandler
    .weak   DMA1_Channel3_IRQHandler
    .weak   DMA1_Channel4_IRQHandler
    .weak   DMA1_Channel5_IRQHandler
    .weak   DMA1_Channel6_IRQHandler
    .weak   DMA1_Channel7_IRQHandler
    .weak   ADC1_2_IRQHandler
    .weak   EXTI9_5_IRQHandler
    .weak   TIM1_BRK_IRQHandler
    .weak   TIM1_UP_IRQHandler
    .weak   TIM1_TRG_COM_IRQHandler
    .weak   TIM1_CC_IRQHandler
    .weak   TIM2_IRQHandler
    .weak   TIM3_IRQHandler
    .weak   TIM4_IRQHandler
    .weak   I2C1_EV_IRQHandler
    .weak   I2C1_ER_IRQHandler
    .weak   I2C2_EV_IRQHandler
    .weak   I2C2_ER_IRQHandler
    .weak   SPI1_IRQHandler
    .weak   SPI2_IRQHandler
    .weak   USART1_IRQHandler
    .weak   USART2_IRQHandler
    .weak   USART3_IRQHandler
    .weak   EXTI15_10_IRQHandler
    .weak   RTCAlarm_IRQHandler
    .weak   USBWakeUp_IRQHandler
    .weak   USBHD_IRQHandler

    .section    .text.handle_reset,"ax",@progbits
    .weak    handle_reset
    .align   1
handle_reset:                        /*  handle_reset起始位置 */
.option push
.option    norelax
    la gp, __global_pointer$         /* 将ld文件中的标签__global_pointer所处的地址值赋给gp寄存器 */
.option    pop
1:
    la sp, _eusrstack                /* 将ld文件中的标签_eusrstack所处的地址值赋给sp寄存器 */
2:
    /* Load data section from flash to RAM */
    la a0, _data_lma
    la a1, _data_vma
    la a2, _edata
    bgeu a1, a2, 2f
1:
    lw t0, (a0)
    sw t0, (a1)
    addi a0, a0, 4
    addi a1, a1, 4
    bltu a1, a2, 1b
2:
    /* clear bss section */
    la a0, _sbss
    la a1, _ebss
    bgeu a0, a1, 2f
1:
    sw zero, (a0)
    addi a0, a0, 4
    bltu a0, a1, 1b
2:
    /* enable all interrupt */  /* csrs ，根据寄存器中每个为1的位，把CSR寄存器对应位置位 */
  lif t0, 0x88
  csrs mstatus, t0          /* 状态寄存器mstatus赋值为0x88，打开所有中断，设置MPP值为00 */
    la t0, _vector_base
  orf t0, t0, 1
    csrw mtvec, t0          /* 将中断向量表的首地址赋值给mtvec寄存器（中断发生时PC的地址） */

  jal  SystemInit           /* 设置MCU系统时钟 */
    la t0, main
    csrw mepc, t0
    mret

/*
mret返回指令(M模式特有的指令)，调用该指令会进行如下操作：

- PC指针设置为mepc的值
- 将mstatus的MPIE域置制到MIE来恢复之前的中断使能
- 将权限模式设置为mstatus的MPP域中的值。

芯片上电默认进入的是机器模式，通过将mstatus中的MPP值设置为00（00: User, 01: Supervisor, 11: Machine），
并将main函数的地址赋值给mepc，调用mret，使得用户在进入main函数运行时，芯片由机器模式切换为用户模式。
*/
```