

复位以后可以在 startup_apm32f10x_hd.s 中看到

```
127 Reset_Handler:
128 ldr r1, = _sidata
129     ldr r2, = _sdata
130     ldr r3, = _edata
131
132     subs r3, r2
133     ble fill_bss_start
134
135     loop_copy_data:
136     subs r3, #4
137     ldr r0, [r1, r3]
138     str r0, [r2, r3]
139     bgt loop_copy_data
140
141     fill_bss_start:
142     ldr r1, = __bss_start
143     ldr r2, = __bss_end
144     movs r0, 0
145     subs r2, r1
146     ble startup_enter
147
148     loop_fill_bss:
149     subs r2, #4
150     str r0, [r1, r2]
151     bgt loop_fill_bss
152
153     startup_enter:
154     bl SystemInit
155     bl entry
156
```

可以看见

初始化 Bss 段后

先执行系统时钟 72MHZ 再进入 entry

```
87  */
88 void SystemInit(void)
89 {
90     /* Set HSIEN bit */
91     RCM->CTRL_B.HSIEN = BIT_SET;
92     /* Reset SCLKSEL, AHBPSC, APB1PSC, APB2PSC, ADCPSC and MCOSEL bits */
93     RCM->CFG &= (uint32_t)0xF8FF0000;
94     /* Reset HSEEN, CSSEN and PLLEN bits */
95     RCM->CTRL &= (uint32_t)0xFE6FFFFF;
96     /* Reset HSEBCFG bit */
97     RCM->CTRL_B.HSEBCFG = BIT_RESET;
98     /* Reset PLLSRCSEL, PLLHSEPC, PLLMULCFG and USBDIV bits */
99     RCM->CFG &= (uint32_t)0xFF80FFFF;
100    /* Disable all interrupts and clear pending bits */
101    RCM->INT = 0x009F0000;
102
103    SystemClockConfig();
104
105    #ifdef VECT_TAB_SRAM
106        SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET;
107    #else
108        SCB->VTOR = FMC_BASE | VECT_TAB_OFFSET;
109    #endif
110 }
```

```

180 static void SystemClockConfig(void)
181 {
182     #ifdef SYSTEM_CLOCK_HSE
183         SystemClockHSE();
184     #elif defined SYSTEM_CLOCK_24MHz
185         SystemClock24M();
186     #elif defined SYSTEM_CLOCK_36MHz
187         SystemClock36M();
188     #elif defined SYSTEM_CLOCK_48MHz
189         SystemClock48M();
190     #elif defined SYSTEM_CLOCK_56MHz
191         SystemClock56M();
192     #elif defined SYSTEM_CLOCK_72MHz
193         SystemClock72M();
194     #elif defined SYSTEM_CLOCK_96MHz
195         SystemClock96M();
196     #endif
197 }
198

```

进入以后，调用 `rtthread_startup()` 函数

```

159 /* Add -eentry to arm-none-eab
160 int entry(void)
161 {
162     rtthread_startup();
163     return 0;
164 }
165 #endif

```

```

232 */
233 int rtthread_startup(void)
234 {
235     rt_hw_interrupt_disable(); 关中断
236
237     /* board level initialization
238      * NOTE: please initialize heap inside board initiali
239      */
240     rt_hw_board_init(); 板载外设初始化
241
242     /* show RT-Thread version */
243     rt_show_version(); 打印一些系统信息
244
245     /* timer system initialization */
246     rt_system_timer_init(); 系统定时器初始化
247
248     /* scheduler system initialization */
249     rt_system_scheduler_init();
250
251     #ifdef RT_USING_SIGNALS
252         /* signal system initialization */
253         rt_system_signal_init();
254     #endif /* RT_USING_SIGNALS */
255
256     /* create init_thread */
257     rt_application_init(); 用户代码
258
259     /* timer thread initialization */
260     rt_system_timer_thread_init(); 系统定时器
261
262     /* idle thread initialization */
263     rt_thread_idle_init(); 空闲任务
264
265     #ifdef RT_USING_SMP
266         rt_hw_spin_lock(&cpus_lock);
267     #endif /* RT_USING_SMP */
268
269     /* start scheduler */ 开启调度
270     rt_system_scheduler_start();

```

1、其中板载外设初始化包括

```

94  * This function will initial APM32 board.
95  */
96  RT_WEAK void rt_hw_board_init()  硬件系统时钟初始化
97  {
98      /* SysTick initialization */
99      rt_hw_systick_init();
100
101      /* Heap initialization */
102      #if defined(RT_USING_HEAP)
103          rt_system_heap_init((void *)HEAP_BEGIN, (void *)HEAP_END);
104      #endif  内存堆管理初始化
105
106      /* Pin driver initialization is open by default */
107      #ifdef RT_USING_PIN
108          rt_hw_pin_init();  GPIO初始化
109      #endif
110
111      /* USART driver initialization is open by default */
112      #ifdef RT_USING_SERIAL
113          rt_hw_usart_init();  串口初始化
114      #endif
115
116      /* Set the shell console output device */
117      #if defined(RT_USING_CONSOLE) && defined(RT_USING_DEVICE)
118          rt_console_set_device(RT_CONSOLE_DEVICE_NAME);
119      #endif  shell
120
121      /* Board underlying hardware initialization */
122      #ifdef RT_USING_COMPONENTS_INIT
123          rt_components_board_init();
124      #endif  其它外设
125  }
126

```

2、rt_show_version 打印一些信息

```

640  */
641  void rt_show_version(void)
642  {
643      rt_kprintf("\n \\ | /\n");
644      rt_kprintf("- RT -      Thread Operating System\n");
645      rt_kprintf(" / | \\      %d.%d.%d build %s %s\n",
646          RT_VERSION, RT_SUBVERSION, RT_REVISION, __DATE__, __TIME__);
647      rt_kprintf(" 2006 - 2022 Copyright by RT-Thread team\n");
648  }
649  RTM_EXPORT(rt_show_version);
650

```

3、rt_system_timer_init 链表初始化

```

832  */
833  void rt_system_timer_init(void)
834  {
835      int i;
836
837      for (i = 0; i < sizeof(_timer_list) / sizeof(_timer_list[0]); i++)
838      {
839          rt_list_init(_timer_list + i);
840      }
841  }
842

```

4、rt_system_scheduler_init 系统调度初始化，主要是排列线程优先级，放入线程优先级链表

```
196  */
197 void rt_system_scheduler_init(void)
198 {
199 #ifdef RT_USING_SMP
200     int cpu;
201 #endif /* RT_USING_SMP */
202     rt_base_t offset;
203
204 #ifndef RT_USING_SMP
205     rt_scheduler_lock_nest = 0;
206 #endif /* RT_USING_SMP */
207
208     RT_DEBUG_LOG(RT_DEBUG_SCHEDULER, ("start scheduler: max priority 0x%02x\n",
209                                         RT_THREAD_PRIORITY_MAX));
210
211     for (offset = 0; offset < RT_THREAD_PRIORITY_MAX; offset++)
212     {
213         rt_list_init(&rt_thread_priority_table[offset]);
214     }
215 #ifdef RT_USING_SMP
```

5、rt_application_init 创建主线程

```
205  */
206 void rt_application_init(void)
207 {
208     rt_thread_t tid;
209
210 #ifdef RT_USING_HEAP
211     tid = rt_thread_create("main", main_thread_entry, RT_NULL,
212                             RT_MAIN_THREAD_STACK_SIZE, RT_MAIN_THREAD_PRIORITY, 20);
213     RT_ASSERT(tid != RT_NULL);
214 #else
215     rt_err_t result;
216
217     tid = &main_thread;
218     result = rt_thread_init(tid, "main", main_thread_entry, RT_NULL,
219                             main_stack, sizeof(main_stack), RT_MAIN_THREAD_PRIORITY, 20);
220     RT_ASSERT(result == RT_EOK);
221
222     /* if not define RT_USING_HEAP, using to eliminate the warning */
223     (void)result;
224 #endif /* RT_USING_HEAP */
225
226     rt_thread_startup(tid);
227 }
228
```

6、rt_thread_idle_init 空闲线程初始化

```

305  */
306 void rt_thread_idle_init(void)
307 {
308     rt_ubase_t i;
309     char tidle_name[RT_NAME_MAX];
310
311     for (i = 0; i < _CPUS_NR; i++)
312     {
313         rt_sprintf(tidle_name, "tidle%d", i);
314         rt_thread_init(&idle[i],
315             tidle_name,
316             rt_thread_idle_entry,
317             RT_NULL,
318             &rt_thread_stack[i][0],
319             sizeof(rt_thread_stack[i]),
320             RT_THREAD_PRIORITY_MAX - 1,
321             32);
322 #ifdef RT_USING_SMP
323         rt_thread_control(&idle[i], RT_THREAD_CTRL_BIND_CPU, (void*)i);
324 #endif /* RT_USING_SMP */
325         /* startup */
326         rt_thread_startup(&idle[i]);
327     }
328

```

7、rt_system_scheduler_start 开启系统任务调度

主要是获取最高优先级线程，将 sp 指针指向它，当前面执行完成后系统会自动执行从 sp 指针这里执行

```

247  */
248 void rt_system_scheduler_start(void)
249 {
250     struct rt_thread *to_thread;
251     rt_ubase_t highest_ready_priority;
252
253     to_thread = _scheduler_get_highest_priority_thread(&highest_ready_priority);
254
255 #ifdef RT_USING_SMP
256     to_thread->oncpu = rt_hw_cpu_id();
257 #else
258     rt_current_thread = to_thread;
259 #endif /* RT_USING_SMP */
260
261     rt_schedule_remove_thread(to_thread);
262     to_thread->stat = RT_THREAD_RUNNING;
263
264     /* switch to new thread */
265 #ifdef RT_USING_SMP
266     rt_hw_context_switch_to((rt_ubase_t)&to_thread->sp, to_thread);
267 #else
268     rt_hw_context_switch_to((rt_ubase_t)&to_thread->sp);
269 #endif /* RT_USING_SMP */
270
271     /* never come back */
272 }
273

```