

```
#define LV_MEM_SIZE (48U * 1024U)
```

```
/*Size of the memory available for lv_mem_alloc() in bytes (>= 2kB)*/
#define LV_MEM_SIZE (10U * 1024U) /*[bytes]*/
```

解决方法: 检查魔法棒里右下角例选第二个RAM1的第二个参数Size是否被CubexDx自动修改超出了芯片内存上限, 超出上限则改为上限值, 并修改在CubexDx中给FreeRTOS任务分配的空间以及手动分配的堆栈空间, 写入一个适当的值, 确保不会超出内存上限

Read/Write Memory Area				
default	off-chip	Start	Size	Unit
<input type="checkbox"/>	<input checked="" type="checkbox"/>	RAM1: <input type="text"/>	<input type="text"/>	<input type="text"/>

RAM: 0x25000000 0x1000

[illegible]

```

/* USER CODE BEGIN 3 */
void wApplicationTickHook(void)
{
    lw_tick_increment();
}

```

[FreeRTOS任务方法测试](#)

问题：在STM32F103VET6平台上移植的FreeRTOS，一开始运行好好的，添加了一个小任务，然后程序无法正常运行

— 2022 —

- [illegible]

任务切换是在 PendSV 处理程序中执行的。该处理程序从代码中切换到更高优先级任务的函数中断。除非该处理程序已嵌套需要 PendSV 处理程序未执行。如果有其他优先级相同或更高的任务，则 configIDLE\_SHOULD\_YIELD 设置为 1 将在每次空闲任务运行时

我假设`configUSE_TIMELICING`未定义,因为它不在原始帖子中。应该意味着如果任何其他就绪状态任务具有与空闲任务相同的优先级,所以找假说同任务的优先级高于空闲任务。

如果我的理论是正确的，我们需要缩小为什么 `Perf25V` 没有执行。  
`configUSE_TICKLESS_IDLE` 设置为 0 运行，看看是否有区别 - 如  
在链接寻找我的线索。

1279  
1280

1282  
1283  
1284



	1290
	1291

xTaskGetSchedulerState() (任务调度器状态)  
xPortSysTickHandler() (系统滴答)

[illegible]

```
1374 the
1375 if(
1376 {
1377 t
```

1378 v  
1379 p  
1380 i  
1381 /

1382  
1383  
1384  
1385

```

1386
1387
1388 }
1389 else

```

1390

```

    }
    else
    {
        // ...
    }
}

```

```

back and at
prvUnlockQ
( void ) x)

```

```

    if( prvIsQue
    {
        traceQUEL
        return pr
    }

```

```

else

```

```

1279      post to the queue? If so, unlock the highest priority waiting
1280      task.
1281      1281 if (!list_is_empty( & paQueue->xTaskWaitingToSend )) == PDFALSE )
1282      {
1283          if ( xTaskRemoveFromEventList( & paQueue->xTaskWaitingToSend ) != PDFALSE )
1284              queue_unblock_if_using_preemption();
1285      }
1286      else
1287      {
1288          mtCOVERAGE_TEST_MARKER();
1289      }
1290      1291

```

`xTaskGetSchedulerState()`函数用来获取当前系统的调度器的状态。判断系统调度器是否已经启动,如果系统调度器已经启动,就会调用`xPortSysTickHandler()`函数来处理系统节拍。`xPortSysTickHandler()`函数的作用是给系统节拍进行计数,并将计数值写入到系统节拍中。通过该函数对时标,可以保证每个任务都能按时间顺序执行。

```

1374 the task on the list of tasks waiting to receive from the queue
1375 {
1376     if (prvQueueSemaphore == 0) {
1377         traceBLOCKING_ON_QUEUE_RECEIVE( prvQueue );
1378         vTaskSuspendEvent1( & prvQueue );
1379         prvQueueSemaphore = 0;
1380     }
1381     if (xTaskResumeAll() == pdFALSE) {
1382         portYIELD_WITHIN_API();
1383     }
1384     else {
1385         mtCOVERAGE_TEST_MARKER();
1386     }
1387 }
1388 }
1389 }
1390 }

```

```

    /* Timed out. If there is no data in the queue exit, otherwise loop
       back and attempt to read the data. */
    prvInQueueQueue( pxQueue );
    ( void ) xTaskResumeAll();

    if( prvIsQueueEmpty( pxQueue ) != pdFALSE )
    {
        traceQUEUE_RECEIVE_FAILED( pxQueue );
        return errQUEUE_EMPTY;
    }
    else

```

```

BaseType_t val;

/* 查询系统空闲堆最小内存大小 */
printf("FreeOfTOS总内存大小: %d 字节\n", configTOTAL_HEAP_SIZE); //打印FreeOfTOS
总内存大小 (单位字节)
printf("当前空闲堆内存大小: %d 字节\n", xPortGetFreeableSize()); //查询当前空闲堆
内存大小
printf("固定空闲堆最小内存大小: %d 字节\n", xPortGetMinimumFreeableSize()); //查询固定空闲堆最小内存大小
printf("可变空闲堆最小内存大小: %d 字节\n", xPortGetMinimumFreeableSize()); //查询可变空闲堆最小内存大小

/* 查询任务空闲堆最小内存大小 */
val = uxTaskGetStackHighWaterMark(SPTask_Handler); //查询SPTask任务空闲堆最小内存
大小 (单位字节)
printf("SPTask任务空闲堆最小内存: %d 字节\n", (int)val+4);
printf("uxTaskGetStackHighWaterMark_A6Task_Handler"); //查询A6Task任务空闲堆最小内存
大小 (单位字节)
printf("A6Task任务空闲堆最小内存: %d 字节\n", (int)val+4);
val = uxTaskGetStackHighWaterMark(MAINTask_Handler); //查询MAINTask任务空闲堆最小
内存大小 (单位字节)
printf("MAINTask任务空闲堆最小内存: %d 字节\n", (int)val+4);
val = uxTaskGetStackHighWaterMark(TCPSENETTask_Handler); //查询TCPSENETTask任务
空闲堆最小内存大小 (单位字节)
printf("TCPSENETTask任务空闲堆最小内存: %d 字节\n", (int)val+4);
val = uxTaskGetStackHighWaterMark(SDTask_Handler); //查询SDTask任务空闲堆最小内存
大小 (单位字节)
printf("SDTask任务空闲堆最小内存: %d 字节\n", (int)val+4);

```

```

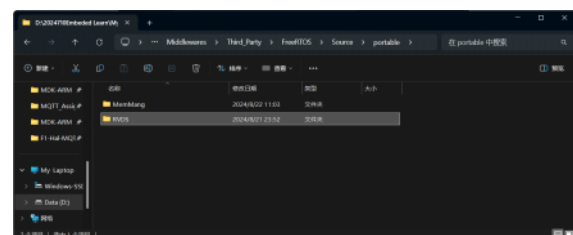
42         {lockType_t}portPMW_DELAY;
43
44     if (xQueueReceive(G_messageQueueToMQTT, data_buffer + strlen(data_buffer), 10))
45     {
46         printf("Received memory size %d\n", (int*)xTaskGetStackHighWaterMark(G_messageQueueToMQTT));
47         while (xQueueReceive(G_messageQueueToMQTT, data_buffer + strlen(data_buffer), 10))
48         {
49             printf("state is %d\n", (int*)xTaskGetSchedulerState());
50         }
51     }
52     msg.payloadLen = strlen(msg.payload);
53     mqtt_publish(client, "mcu_text", &msg); // publish the message to mqtt server
54     memset(data_buffer, 0, sizeof(data_buffer)); // reset data buffer
55     platform_async_unlink(&vTaskDelete);
56     printf("Send data to MQTT server successfully\n");
57     vTaskDelay(1000);
58 }

```

[illegible]

The diagram shows a vertical stack of four colored boxes representing software layers. From top to bottom, they are: a yellow box labeled 'UI App', a blue box labeled 'PageManager', a green box labeled 'LVGL', and an orange box labeled 'HWAcess'. Below these is a grey box that is partially cut off.

Name	Location/Value	Type
♥ <code>InterDefault_Handler</code>	0x00001118	void (*)
♥ <code>PendSV_Handler</code>	0x0000203E	void (*)
♥ <code>vQueueSemaphoreTake</code>	0x00005A86	long (*)
♥ <code>USART3_Read</code>	0x00042E84	void (*)
♥ <code>ProcessQueueOfStrings</code>	0x00000000	void (*)



分区 快速笔记 的第 2 页



實驗公報稿:

```

60 * user CODE END SMARTS_page < */
61 #if 0
62 /* brief This function handles smart's global interrupt.
63 */
64 void SMARTS_INTERRUPT(void)
65 {
66     * user CODE BEGIN SMARTS_page < */
67
68     * user CODE END SMARTS_page < */
69     SMART_INTERRUPT_HANDLER();
70     * user CODE BEGIN SMARTS_page < */
71 }
72 #endif
73 * user CODE BEGIN < */

```

```

( void ) xtaskResumeAll();

if( prvIsQueueEmpty( pxQueue ) != pdFALSE )
{
    traceQUEUE_RECEIVE_FAILED( pxQueue );
    return errQUEUE_EMPTY;
}
else
{
    mtCOVERAGE_TEST_MARKER();
}
}

```

```
0x20006f8: .....  
0x20006fc: .....  
0x2000700: .....  
0x2000704: .....  
breakEnable breakFill breakLink breaker breakDowns COVERAGE_DEFINE DIR
```

```
Size: Code=19340 EO-data=60132 PW-data=554 TF-data=19372
I creating hex file...
C:\MSD\Fl_FIRM\199L.amt* - 0 Error(s), 0 Warning(s).
Time: 0'14.4444444 sec
```

```

#!/bin/bash

#(1): Show CPU usage and PPS count" # is high count
define LV_USE_PPP_PERMISSION 0
if [ LV_USE_PPP_PERMISSION
    define LV_USE_PPP_PERMISSION_POS LV_ALIGN_BOTTOM_RIGHT
endif

#(2): Show the used memory and the memory fragmentation
# Requires LV_MEM_CUSTOM = 0"
define LV_USE_MEM_PERMISSION 0
if [ LV_USE_MEM_PERMISSION
    define LV_USE_MEM_PERMISSION_POS LV_ALIGN_BOTTOM_LEFT
endif

```

adc

Search (Ctrl+F)

Data Alignment: Right alignment

Scan Conversion Mode: Enabled

Continuous Conversion Mode: Disabled

Discontinuous Conversion Mode: Disabled

ADC\_Regular\_ConversionMode: Enable Regular Conversions

Number Of Conversions: 2

External Trigger Conversion Source: Rank 1

Regular Conversion launched by:

```

1427 int rc = MQTT_FAILED_ERROR;
1428 platform_timer_t timer;
1429 MQTTString topic = MQTTString_initializer;
1430 topic.cstring = (char *)topic_filter;
1431 /*if client is not connected, clear the payloadlen and return error (-1)*/
1432 if (client.state != CONNECTED) return client_state(-1);
1433 msg.payloadlen = 0; // clear
1434 rc = MQTT_NOT_CONNECTED_ERROR;
1435 RETURN_ERROR(rc);
1436 /* goto exit; */ *task = 0;
1437
1438 /*if message is not null, but payloadlen is 0, set the payloadlen to strlen(msg)
1439 if (NULL != msg->payload) && (0 == msg->payloadlen)
1440     msg.payloadlen = strlen(msg->payload);

```

```

50 // state
51 static int next_val(int* p, int* c, int* limit, int* m)
52 {
53     int rc = NEXT_SUCCESS_ERROR;
54     *limit = *m;
55     platform_time_t timer;
56
57     if (m1 == c)
58         return ERROR_NOT_PAID_ERROR;
59
60 if (c == *limit, m)
61     return, m0 = c - 1, m1 = *limit, m;
62
63 platform_time_t start_time;
64 platform_time_t end_time;
65
66 while ((platform_time_t) is expired(m0)) {
67     state = NEXT_SUCCESS_ERROR;
68
69 if (ALREADY_STATE_ERROR_SUCCESS == state) {
70     return, m0 = c - 1, m1 = NEXT_SUCCESS_ERROR;
71 } else if (IS_STATE_ERROR_SUCCESS == state) {
72     // will not return, and recover
73     rc = m1 != 0 ? recover(c) : 0;
74
75 if (NEXT_SUCCESS_ERROR_SUCCESS == rc)
76     return, success(rc);
77
78     continue;
79 }
80 }

```

```

17 | }
18 |
19 | static client_state_t mqtt_get_client_state(mqtt_client_t* c)
20 | {
21 |     return c->mqtt_client_state;
22 | }
23 |

```

Use regular conversion	1 channel
Start Of Conversion	0
Start Trigger Conversion Source	Regular Conversion launched by software
Task	0
Channel	Channel 1
Sampling Time	1.5 Cycles
Task	2
Channel	
Sampling Time	1.5 Cycles
AI_ConversionMode	
AI_Injected Conversions	Disable

因为模块上电不稳定，会乱发一些数据过来，然后我提前开启了中断，这时候我在自己写uart3handler里提前释放了信号量，导致rtos卡死在队列获取中断函数里

[关于FreeRTOS函数xSemaphoreGiveFromISR卡死的问题，xSemaphoreGiveFromISR导致卡死-CSDN博客](#)

分区 快速笔记 的第 7 页





