南昌大学机器人队 Passion 战队

RM_Frame 程序说明文档

高云海

2020.12.21

目录

第一章　写在前面

1.1 软件开发环境

Toolchain：MDK-ARM Plus　Version：5.26.2.0 【MDK 版本信息】

Software Pack：Keil.STM32F4xx_DFP.2.9.0 【STM32 固件库】

STM32CubeMx：Version 5.5.0【 STM32CubeMx 版本信息】

Firmware Package：STM32Cube FW_F4 V1.24.2 【CubeMx 固件包】

FreeRTOS version：10.0.1 【FreeRTOS 操作系统版本信息】

CMSIS-RTOS version：1.02 【操作系统标准软件接口版本信息】

注：由于 STM32CubeMx 默认设置会自动检测更新并升级，所以建议关闭自动升级，操作如图。

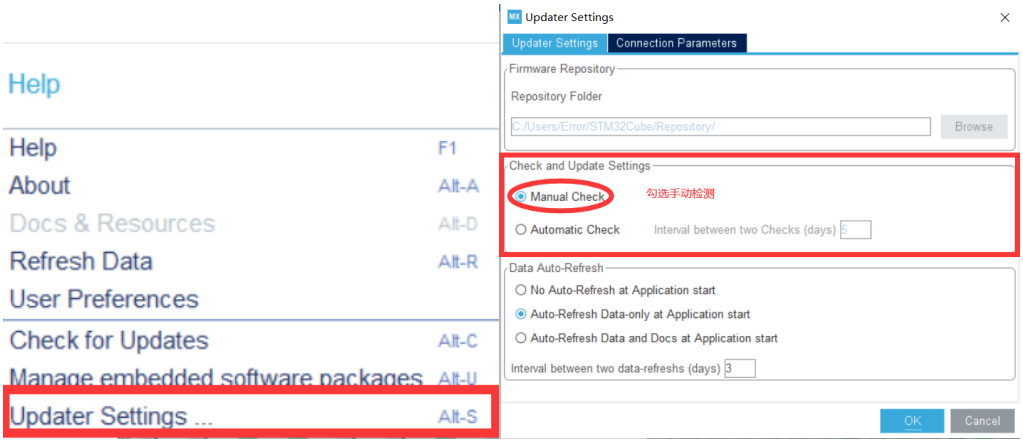


图 1-1 取消自动检测更新操作

1.2 注意事项

若使用该工程模板，请务必仔细阅读该章节内容，注意以下几点内容。

1、本工程是基于 stm32F427IIH6 主控芯片的 RoboMaster 开发板 A 型建立的，故若不是同款开发板，请根据实际芯片重新使用 CubeMX 配置底层文件，所有模块层文件均可直接直接复制使用。

2、若使用 CubeMX 进行修改底层配置时并重新生成工程时，由于 CubeMX 重新生成工程时对于系统生成的文件会清除未写在指定位置的程序，请务必注意以下五点：

① 请确保 main.c 文件中进入临界段【taskENTER_CRITICAL()】与退出临界段【taskEXIT_CRITICAL()】代码书写与正确位置。

```
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
  taskENTER_CRITICAL();  //进入临界段
  //由于退出临界段代码无法写在指定位置，故当使用CubeMX重新生成工程时，
  //需要将taskEXIT_CRITICAL()退出临界段重新写入
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_CAN1_Init();
  MX_CAN2_Init();
  MX_SPI5_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_TIM5_Init();
  MX_UART7_Init();
  MX_UART8_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  MX_USART6_UART_Init();
  /* USER CODE BEGIN 2 */
  BSP_Init();
  /* USER CODE END 2 */

  /* Call init function for freertos objects (in freertos.c) */
  MX_FREERTOS_Init();

  taskEXIT_CRITICAL();//退出临界段
  /* Start scheduler */
  osKernelStart();
```

图 1-2 进入及退出临界段

② 请确保将 main.c 文件中 CubeMX 自动生成的 TIM 回调函数注释掉，原因是在 offline_check.c 文件中自己重新写了 TIM 回调函数，若不将 main.c 文件中的 TIM 回调函数注释掉就是重复定义，则会报错。

```
/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM1 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
//void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
//{
//  /* USER CODE BEGIN Callback 0 */

//  /* USER CODE END Callback 0 */
//  if (htim->Instance == TIM1) {
//    HAL_IncTick();
//  }
//  /* USER CODE BEGIN Callback 1 */

//  /* USER CODE END Callback 1 */
//}
```

图 1-3 main.c 文件 TIM 回调注释部分

```c
/**
  * @brief          TIM中断回调函数（需要将main.c中cubemx生成的TIM中断回调函数注释）
  * @param[out]
  * @param[in]
  * @retval
*/
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM1)
  {
    HAL_IncTick();
  }

  else if(htim == (&OL_htim))
  {
    Refresh_SysTime();
  }
}
```

图 1-4　offline_check.c 文件 TIM 回调自写部分

③ 当使用 PID 无线调参时,首先确保使用的串口正确性，以串口 2 为例，其次需要注释掉 stm32f4xx_it.c 文件中的 USART2_IRQHandler(void)函数中的 HAL_UART_IRQHandler(&huart2)语句。

```c
/**
  * @brief This function handles USART2 global interrupt.
  */
void USART2_IRQHandler(void)
{
  /* USER CODE BEGIN USART2_IRQn 0 */

  /* USER CODE END USART2_IRQn 0 */
//  HAL_UART_IRQHandler(&huart2);
  /* USER CODE BEGIN USART2_IRQn 1 */
//stm32Mxcube重新配置生成代码后需要注释掉函数void USART2_IRQHandler(void)中的HAL_UART_IRQHandler(&huart2)。
//原因为此处不按照HAL_UART_IRQHandler函数的方式接收数据，重新编写了一段接收函数，方便按照自己要求进行数据接收。
  PW_IRQHandler();
  /* USER CODE END USART2_IRQn 1 */
}
```

图 1-5　串口中断服务函数需要注释部分

④ 若需要在系统生成的文件中添加代码，务必添加在指定位置即 USER CODE BEGIN 与 USER CODE END 两注释之间。

```c
/**
  * @brief This function handles USART1 global interrupt.
  */
void USART1_IRQHandler(void)
{
  /* USER CODE BEGIN USART1_IRQn 0 */

  /* USER CODE END USART1_IRQn 0 */
  HAL_UART_IRQHandler(&huart1);
  /* USER CODE BEGIN USART1_IRQn 1 */
  DR16_UART_IRQHandler();
  /* USER CODE END USART1_IRQn 1 */
}
```

图 1-6　系统文件代码书写指定位置

⑤ 对底层配置进行修改后，需要修改 README 文件夹下的资源分配表，同时重新生成工程报告即 rm_frame.pdf 文档。生成方法如图 1-7 所示，STM32CubeMX 中最上面的菜单栏中的 File 栏，点击 Generate Report 即可生成工程报告 rm_frame.pdf 文档。

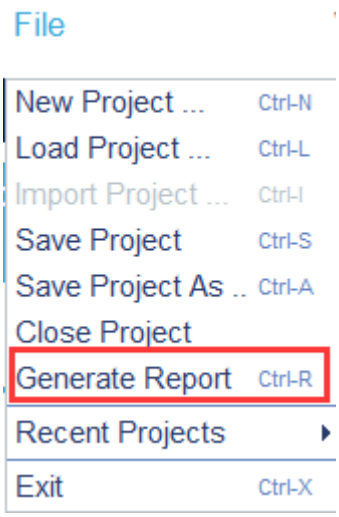


图 1-7 生成工程报告

3、所有外设使用情况均在工程中 README 文件夹下的资源分配表（讲解见 3.1）中有简略说明，详细说明见工程目录下 rm_frame.pdf 文档。

第二章　程序框架分析

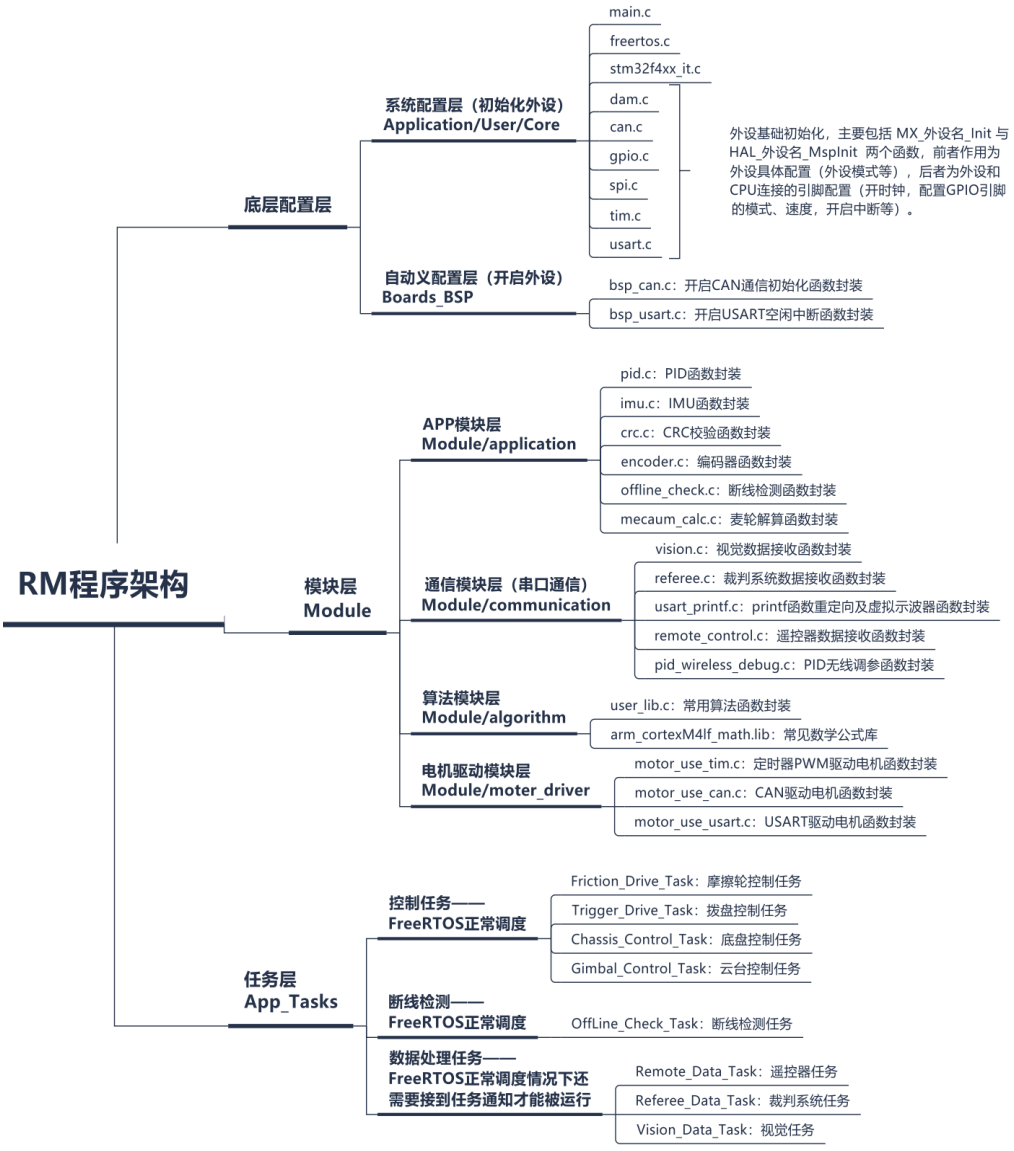


图 2-1　rm_frame 程序整体架构图

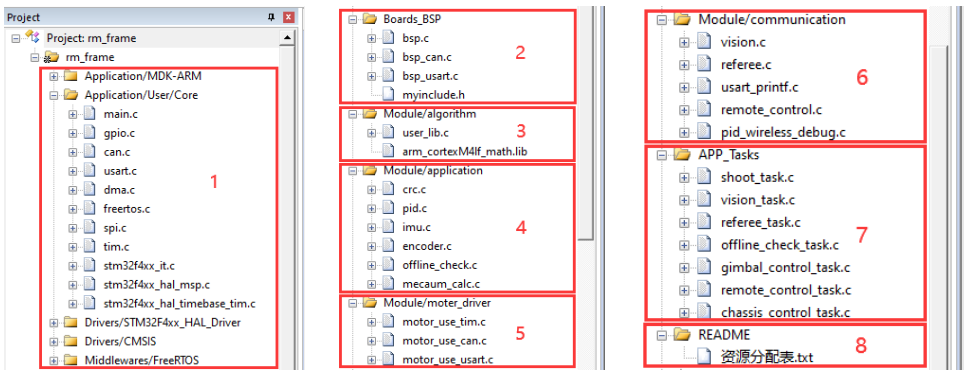


图 2-2　rm_frame 程序目录图

2.1 程序框架分析

程序共可分为三大板块：底层配置层、模块层、任务层。同时又分为八个小板块即图 2-2 所示，分别为系统文件、启用外设初始化文件、算法模块文件、APP 应用模块文件、电机驱动模块文件、通信模块文件（串口通信）、任务文件、readme 文件。

① 系统文件： STM32CubeMX 生成的工程文件。

② 外设初始化文件：启用外设的函数封装文件（bsp_can.c、bsp_usart.c）以及所有使用到的外设开启函数的汇总文件（bsp.c）。

③ 算法模块文件： lib 库以及算法文件。

④ APP 应用模块文件：不同功能的应用文件。

⑤ 电机驱动模块文件：不同方式驱动电机文件。

⑥ 通信模块文件：使用串口进行通信文件。

⑦ 任务文件：所有任务汇总文件。

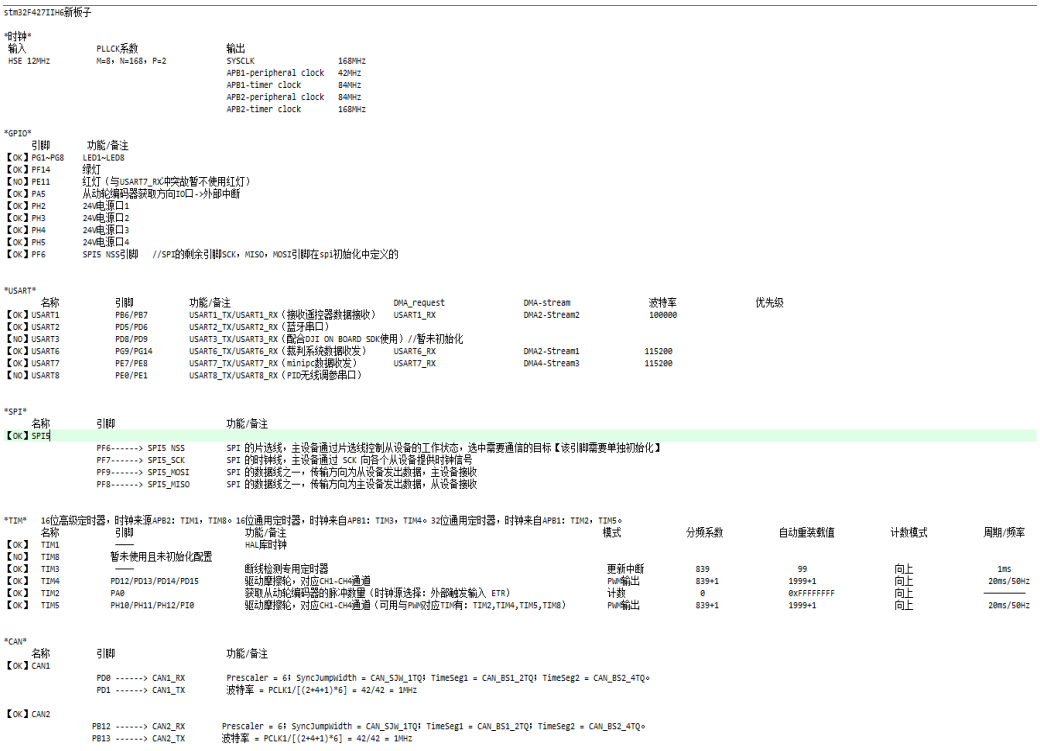
⑧ readme 文件：资源分配表即外设使用情况表。

对于算法模块文件、APP 应用模块文件、电机驱动模块文件、通信模块文件四个模块层文件，里面所有封装好的功能模块均可直接复制对应功能的.c 与.h 文件，并添加至自己工程中，按照注释完成相关操作后方可直接使用。

第三章　程序模块分析

3.1 README 文件

目前 readme 文件中只有资源分配表，其详细记录外设资源使用情况如图 3-1 所示。



stm32F427IIH6新板子

时钟

输入	PLLCK系数	输出	
HSE 12MHz	M=8, N=168, P=2	SYSCLK	168MHz
		APB1-peripheral clock	42MHz
		APB1-timer clock	84MHz
		APB2-peripheral clock	84MHz
		APB2-timer clock	168MHz

GPIO

	引脚	功能/备注
【OK】	PG1-PG8	LED1-LED8
【OK】	PF14	绿灯
【NO】	PE11	红灯（与USART7_RX冲突故暂不使用红灯）
【OK】	PA5	从动轮编码器获取方向IO口->外部中断
【OK】	PH2	24v电源口1
【OK】	PH3	24v电源口2
【OK】	PH4	24v电源口3
【OK】	PH5	24v电源口4
【OK】	PF6	SPI5 NSS引脚　//SPI的剩余引脚SCK，MISO，MOSI引脚在spi初始化中定义的

USART

	名称	引脚	功能/备注	DMA_request	DMA-stream	波特率	优先级
【OK】	USART1	PB6/PB7	USART1_TX/USART1_RX（接收遥控器数据接收）	USART1_RX	DMA2-Stream2	100000	
【OK】	USART2	PD5/PD6	USART2_TX/USART2_RX（蓝牙串口）				
【NO】	USART3	PD8/PD9	USART3_TX/USART3_RX（配合DJI ON BOARD SDK使用）//暂未初始化				
【OK】	USART6	PG9/PG14	USART6_TX/USART6_RX（裁判系统数据收发）	USART6_RX	DMA2-Stream1	115200	
【OK】	USART7	PE7/PE8	USART7_TX/USART7_RX（minipc数据收发）	USART7_RX	DMA4-Stream3	115200	
【NO】	USART8	PE0/PE1	USART8_TX/USART8_RX（PID无线调参串口）				

SPI

	名称	引脚	功能/备注
【OK】	SPI5		
		PF6------> SPI5 NSS	SPI 的片选线，主设备通过片选线控制从设备的工作状态，选中需要通信的目标【该引脚需要单独初始化】
		PF7------> SPI5_SCK	SPI 的时钟线，主设备通过 SCK 向各个从设备提供时钟信号
		PF9------> SPI5_MOSI	SPI 的数据线之一，传输方向为从设备发出数据，主设备接收
		PF8------> SPI5_MISO	SPI 的数据线之一，传输方向为主设备发出数据，从设备接收

TIM　16位高级定时器，时钟来源APB2: TIM1，TIM8。16位通用定时器，时钟来自APB1: TIM3，TIM4。32位通用定时器，时钟来自APB1: TIM2，TIM5。

	名称	引脚	功能/备注	模式	分频系数	自动重装载值	计数模式	周期/频率
【OK】	TIM1	————	HAL库时钟					
【NO】	TIM8	暂未使用且未初始化配置						
【OK】	TIM3	————	底线检测专用定时器	更新中断	839	99	向上	1ms
【OK】	TIM4	PD12/PD13/PD14/PD15	驱动摩擦轮，对应CH1-CH4通道	PWM输出	839+1	1999+1	向上	20ms/50Hz
【OK】	TIM2	PA0	获取从动轮编码器的脉冲数量（时钟源选择：外部触发输入 ETR）	计数	0	0xFFFFFFFF	向上	————
【OK】	TIM5	PH10/PH11/PH12/PI0	驱动摩擦轮，对应CH1-CH4通道（可用与PWM对应TIM有：TIM2,TIM4,TIM5,TIM8）	PWM输出	839+1	1999+1	向上	20ms/50Hz

CAN

	名称	引脚	功能/备注
【OK】	CAN1		
		PD0 ------> CAN1_RX	Prescaler = 6; SyncJumpWidth = CAN_SJW_1TQ; TimeSeg1 = CAN_BS1_2TQ; TimeSeg2 = CAN_BS2_4TQ。
		PD1 ------> CAN1_TX	波特率 = PCLK1/((2+4+1)*6] = 42/42 = 1MHz
【OK】	CAN2		
		PB12 ------> CAN2_RX	Prescaler = 6; SyncJumpWidth = CAN_SJW_1TQ; TimeSeg1 = CAN_BS1_2TQ; TimeSeg2 = CAN_BS2_4TQ。
		PB13 ------> CAN2_TX	波特率 = PCLK1/((2+4+1)*6] = 42/42 = 1MHz

图 3-1 开发板外设资源分配

图中标明外设的基本配置情况，以及其功能。同时最左边标有【YES】与【NO】,【YES】表示已完成测试，且测试情况正常，能够正常使用;【NO】表示未使用 STM32CubeMX 进行初始化配置，或表示未进行测试不能保证能够正常使用需要先进行测试再使用。

未增加程序的规范管理后续可添加工程修改记录文档：modify_notes.txt 用以记录何人与何时在何处进行了何修改。

3.2 APP_Task 文件

APP_Task 文件中包含所有任务，目前所有任务仅完成任务的创建，并未完成任务内部算法程序的添加，后续需要根据不同兵种的实际情况对各任务进行添加代码以实现相关功能及达到预期目标。

若需要另外增加任务，请完成以下几步：

注：不建议使用 STM32CubeMX 创建任务并重新生成工程。

① 创建对应任务.c 文件与.h 文件，保存于工程路径下的 APP_Task 文件夹中，并添加至 APP_Task 文件中。

② 按照规定格式要求完成.c 与.h 文件注释的添加，具体见后续章节。

③ 在该.c 文件中按照规定创建任务函数，名命为 xxx_Task()，例如 Trigger_Drive_Task()拨盘任务，如图 3-2 所示。

```
/**
  * @brief        拨盘任务
  * @param[out]
  * @param[in]
  * @retval
*/
void Trigger_Drive_Task(void const * argument)
{
  portTickType xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();

  for(;;)
  {
    Refresh_Task_OffLine_Time(TriggerDriveTask_TOE);//记录任务运行的时间点


    osDelayUntil(&xLastWakeTime,TRIGGER_PERIOD);

  }

}
```

图 3-2 拨盘任务函数的创建

④ 在 freertos.c 中定义该任务的句柄，名命为 xxxTaskHandle，例如 TriggerDriveTaskHandle，并于 myinclude.h 中进行声明，便于其他文件使用该任务句柄，如图 3-3 所示。

```
/* USER CODE BEGIN Variables */
osThreadId RemoteDataTaskHandle;
osThreadId VisionDataTaskHandle;
osThreadId RefereeDataTaskHandle;
osThreadId TriggerDriveTaskHandle;
osThreadId FrictionDriveTaskHandle;
osThreadId GimbalControlTaskHandle;
osThreadId ChassisControlTaskHandle;
osThreadId OffLineCheckTaskHandle;

/* USER CODE END Variables */
```

```
extern osThreadId RemoteDataTaskHandle;
extern osThreadId VisionDataTaskHandle;
extern osThreadId RefereeDataTaskHandle;
extern osThreadId TriggerDriveTaskHandle;
extern osThreadId FrictionDriveTaskHandle;
extern osThreadId GimbalControlTaskHandle;
extern osThreadId ChassisControlTaskHandle;
```

图 3-3 任务句柄定义及声明

⑤ 在 freertos.c 中声明之前创建的任务函数，如图 3-4 所示。

```
void Remote_Data_Task(void const * argument);
void Vision_Data_Task(void const * argument);
void Referee_Data_Task(void const * argument);
void Friction_Drive_Task(void const * argument);
void Trigger_Drive_Task(void const * argument);
void Gimbal_Control_Task(void const * argument);
void Chassis_Control_Task(void const * argument);
void OffLine_Check_Task(void const *argument);
```

图 3-4 任务函数的声明

⑥ 在 freertos.c 中创建任务，要创建任务时，只需要调用 osThreadDef 和 osThreadCreate 即可。例如创建拨盘任务如图 3-5 所示，其中 TriggerDriveTask 代表任务名称，Trigger_Drive_Task 代表任务函数的入口即任务函数名称，TriggerDriveTaskHandle 代表任务句柄。

```
osThreadDef(TriggerDriveTask, Trigger_Drive_Task, osPriorityNormal, 0, 256);        //拨盘控制任务
TriggerDriveTaskHandle = osThreadCreate(osThread(TriggerDriveTask),  NULL);
```

图 3-5 拨盘任务创建

完成以上六步即完成了一个任务的创建，下面对创建任务中使用到的两个关键 osThreadDef 和 osThreadCreate 宏于函数进行分析。

首先介绍 osThreadDef，实际上这不是一个函数，而是一个由 CMSIS 提供的宏定义，用于对要创建的任务进行设置。osThreadDef 详细介绍如图 3-6 所示。

```
#define osThreadDef(name, thread, priority, instances, stacksz)   \
extern const osThreadDef_t os_thread_def_##name
```

名称	osThreadDef
功能	对要创建的任务进行设置
参数 1	name，要创建的任务的名称
参数 2	thread，要创建的任务代码的入口名称
参数 3	priority，要创建的任务的优先级
参数 4	instances，任务下可以创建的线程的数量
参数 5	stacksz，任务栈大小

图 3-6 osThreadDef 详细介绍

参数 3 任务优先级 priority 参数共有 7 个优先级，由低到高分别为 osPriorityIdle、osPriorityLow、osPriorityBelowNormal、osPriorityNormal、osPriorityAboveNormal、osPriorityHigh、osPriorityRealtime。默认优先级为 osPriorityNormal。

参数 4 线程的数量 instances 参数默认为 0。

参数 5 任务栈的大小 stacksz 参数由任务所需内存空间决定（可以通过

代码量来决定），可选值有 128，256，512 等，一般默认为 128。

接着通过 CMSIS 提供的 osThreadCreate 函数来创建任务，该函数详细介绍如图 3-7 所示。

```
osThreadId osThreadCreate (const osThreadDef_t *thread_def, void *argument);
```

函数名称	osThreadCreate
函数功能	创建一个任务
返回值	osThreadId，任务 ID，ID 是一个任务的重要标识，当在创建完任务后需要执行修改这个任务的优先级，或者销毁该任务时，就需要调用任务 ID，需要提前声明一个类型为 osThreadId 的变量在此处存储返回值。
参数 1	const osThreadDef_t *thread_def，我们通过 osThreadDef 所设置的任务参数，采用强制转换+任务名的方式进行输入，比如在 osThreadDef 中设置任务名为 LED_RED，则在此处输入 osThread(LED_RED)
参数 2	void *argument，任务需要的初始化参数，一般填为 NULL

图 3-7　osThreadCreate 详细介绍

通过以上两步，一个任务就成功创建了。操作系统就会通过创建的任务找到我们之前创建的任务函数，并按照优先级高低进行调度任务。

3.3 Communication Module 文件

3.3.1 视觉数据接收 vision.c

该文件是视觉数据接收与解码及发送，包括 USART 初始化、UASRT 中断处理、数据解码、数据发送函数。

可以通过该文件开头注释栏查看该文件功能、函数列表及各函数调用情况，如图 3-8 所示。所有.c 文件均有该注释，后续不再进行讲解。

```
/*******************************************************************************
                    版权所有 (C), 2020-,NCUROBOT
********************************************************************************
 文 件 名   : vision.c
 版 本 号   : V1.0
 作    者   : 高云海
 生成日期   : 2020.12.8
 最近修改   :
 功能描述   : 视觉数据接收与解码及发送【USART初始化、UASRT中断处理、数据解码、数据发送】
 函数列表   : 1) VisionDate_Receive_Init()   【外部调用：bsp.c】
              2) Vision_UART_IRQHandler()    【外部调用：stm32f4xx_it.c的USART7中断服务函数】
              3) SBUS_To_Vision()            【内部调用：Vision_UART_IRQHandler()】
*******************************************************************************/
```

图 3-8 文件注释栏

默认情况与视觉通信 USART 为 USART7，若有改变，请直接改变视觉通信所有 USART 的宏，如图 3-9 所示。所有 Module 文件中的功能文件均设有类似宏，当需要修改外设时，同样修改宏即可。

```
#define VD_huart  huart7 //与视觉通信所用串口
```

图 3-9 视觉通信所用串口宏

1、USART 初始化

通过调用封装好的 API 函数完成 DMA 的开启+ USART 空闲中断的开启。封装好的 API 函数详情请见 3.7.3 节。

2、UASRT 中断处理

首先创建视觉 USART 中断处理函数来完成重新设置 USART 与 DMA 参数+数据解码+发送任务通知等操作，详细注释如图 3-10 所示。

```
/**
  * @brief        视觉USART中断处理函数（重新设置DMA参数+数据解码+发送任务通知）
  * @param[in]
  * @param[out]
  * @retval       none
*/
void Vision_UART_IRQHandler(void)
{
  static  BaseType_t  pxHigherPriorityTaskWoken;
  /* 判断是否为空闲中断 */
  if (__HAL_UART_GET_FLAG(&VD_huart, UART_FLAG_IDLE))
  {
      /* 清除空闲标志，避免一直处于空闲状态的中断 */
      __HAL_UART_CLEAR_IDLEFLAG(&VD_huart);

      /* 关闭DMA */
      __HAL_DMA_DISABLE(VD_huart.hdmarx);

      /* 计算接收到的视觉数据长度与发送的是否一致 */
      if ((VISION_RX_BUFFER_SIZE - VD_huart.hdmarx->Instance->NDTR) == VISION_DATA_LEN)
      {
        /* 数据解码 */
        SBUS_To_Vision(vision_rx_buf,&minipc_rx);
        /* 断线检测刷新时间 */
        Refresh_Device_OffLine_Time(Vision_TOE);
        /* 任务通知 */
        vTaskNotifyGiveFromISR(VisionDataTaskHandle,&pxHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(pxHigherPriorityTaskWoken);
      }
      /* 重新启动DMA传输 */
      /* 设置DMA可传输最大数据长度 */
      __HAL_DMA_SET_COUNTER(VD_huart.hdmarx, VISION_RX_BUFFER_SIZE);
      /* 开启DMA传输 */
      __HAL_DMA_ENABLE(VD_huart.hdmarx);

  }
}
```

图 3-10 视觉 USART 中断处理函数

(1) 对于 USART 与 DMA 外设方面设置需要注意以下几点：

① 必须先判断是否为空闲中断

② 必须清除空闲标志位，避免一直处于空闲状态的中断。查询

STM32F4xx 中文开发手册可知状态寄存器(USART_SR)的位 4 IDLE（检测到空闲线路）是通过硬件置 1，软件清零的。

位 4 **IDLE**：检测到空闲线路 (IDLE line detected)

检测到空闲线路时，该位由硬件置 1。如果 USART_CR1 寄存器中 IDLEIE = 1，则会生成中断。该位由软件序列清零（读入 USART_SR 寄存器，然后读入 USART_DR 寄存器）。

0：未检测到空闲线路

1：检测到空闲线路

注意：直到 RXNE 位本身已置 1 时（即，当出现新的空闲线路时）IDLE 位才会被再次置 1。

图 3-11　USART_SR 状态寄存器位 4　IDLE 说明

③ 关闭 DMA 数据传输。原因为 DMA 数据流 x 数据项数寄存器(DMA_SxNDTR)需要禁止数据流才能设定参数。

位 15:0 **NDT[15:0]**：要传输的数据项数目 (Number of data items to transfer)

要传输的数据项数目（0 到 65535）。只有在禁止数据流时，才能向此寄存器执行写操作。使能数据流后，此寄存器为只读，用于指示要传输的剩余数据项数。每次 DMA 传输后，此寄存器将递减。

传输完成后，此寄存器保持为零（数据流处于正常模式时），或者在以下情况下自动以先前编程的值重载：

– 以循环模式配置数据流时。

– 通过将 EN 位置“1”来重新使能数据流时

如果该寄存器的值为零，则即使使能数据流，也无法完成任何事务。

图 3-12　DMA_SxNDTR DMA 数据流 x 数据项数寄存器位 15:0　NDT 说明

④ 重新设置要传输的数据项数目，一般设置值是所需要接收的数据长度的两倍。

⑤ 开启 DAM 数据传输

(2) 断线检测时间刷新，即刷新每次进入该函数的时刻，若未接收到数据，则不会刷新时间。具体断线检测流程详见 3.5.5 节

(3) 对于任务通知需要注意以下几点：

① vTaskNotifyGiveFromISR()函数，调用该函数将任务通知值加一。

4、函数 vTaskNotifyGiveFromISR()

此函数为 xTaskNotifyGive()的中断版本，用在中断服务函数中，函数原型如下：

```
void vTaskNotifyGiveFromISR( TaskHandle_t    xTaskHandle,
                             BaseType_t *    pxHigherPriorityTaskWoken );
```

参数：

xTaskToNotify： 任务句柄，指定任务通知是发送给哪个任务的。

pxHigherPriorityTaskWoken: 记退出此函数以后是否进行任务切换，这个变量的值函数会自动设置的，用户不用进行设置，用户只需要提供一个变量来保存这个值就行了。当此值为 pdTRUE 的时候在退出中断服务函数之前一定要进行一次任务切换。

图 3-13　vTaskNotifyGiveFromISR()函数说明

② portYIELD_FROM_ISR()函数，用于中断级的任务切换。在退出串

口中断服务函数之前调用函数 portYIELD_FROM_ISR()进行一次任务调度。

```
#define   portEND_SWITCHING_ISR( xSwitchRequired ) \
          if( xSwitchRequired != pdFALSE )   portYIELD()
#define portYIELD_FROM_ISR( x ) portEND_SWITCHING_ISR( x )
```

图 3-14　portYIELD_FROM_ISR()函数定义

③ ulTaskNotifyTake()函数，该函数用于任务中，通过判断该函数的返回值从而可知是否收到任务通知，若未收到则任务进入堵塞状态，反之则任务正常运行。

1、函数 ulTaskNotifyTake()

此函数为获取任务通知函数，当任务通知用作二值信号量或者计数型信号量的时候可以使用此函数来获取信号量，函数原型如下：

```
uint32_t ulTaskNotifyTake( BaseType_t     xClearCountOnExit,
                           TickType_t     xTicksToWait );
```

参数：

xClearCountOnExit： 参数为 pdFALSE 的话在退出函数 ulTaskNotifyTake()的时候任务通知值减一，类似计数型信号量。当此参数为 pdTRUE 的话在退出函数的时候任务任务通知值清零，类似二值信号量。

xTickToWait: 阻塞时间。

返回值：

任何值　：　任务通知值减少或者清零之前的值。

图 3-15　ulTaskNotifyTake()函数说明

总结：任务通知主要是在中断中发送任务通知，在对应任务中获取任务通知值，若为 1 则任务正常运行，反之则任务处于堵塞状态。

其次在 stm32f4xx_it.c 文件对应串口中断服务函数中调用创建的视觉 USART 中断处理函数，如图 3-10 所示。

```
/**
  * @brief This function handles UART7 global interrupt.
  */
void UART7_IRQHandler(void)
{
  /* USER CODE BEGIN UART7_IRQn 0 */

  /* USER CODE END UART7_IRQn 0 */
  HAL_UART_IRQHandler(&huart7);
  /* USER CODE BEGIN UART7_IRQn 1 */
  Vision_UART_IRQHandler();
  /* USER CODE END UART7_IRQn 1 */
}
```

图 3-16 视觉 USART 中断处理函数的调用

3、数据解码

根据发送的协议及数据格式进行数据解码。

```
/**
  * @brief          视觉数据解码函数
  * @param[in]      buff: 指向数据接收数组的指针
  * @param[out]     Minipc_Rx: 指向存储视觉数据结构体的指针
  * @retval
*/
static void SBUS_To_Vision(volatile const uint8_t *buff, Minipc_Rx_Struct *Minipc_Rx)
{
  if (buff == NULL || Minipc_Rx == NULL)
  {
      return;
  }

  Minipc_Rx->frame_header = buff[0];
  Minipc_Rx->frame_tail   = buff[8];

  if((Minipc_Rx->frame_header == 0xFF) && (Minipc_Rx->frame_tail == 0xFE))
  {
    Minipc_Rx->angle_yaw  = (int16_t)(buff[1] << 8 | buff[2]);
    Minipc_Rx->angle_pit  = (int16_t)(buff[3] << 8 | buff[4]);
    Minipc_Rx->state_flag = buff[5];
    Minipc_Rx->distance   = buff[6]<<8|buff[7];
  }

}
```

图 3-17 视觉数据解码

4、数据发送

目前暂未添加数据发送函数，后续将逐步实际与视觉通信的模式，并增加数据发送函数。

3.3.2 裁判系统数据接收 referee.c

该文件是裁判系统数据接收与解码及发送，包括 USART 初始化、UASRT 中断处理、数据解码、数据发送函数。

1、USART 初始化

参考视觉数据接收部分

2、UASRT 中断处理

参考视觉数据接收部分

3、数据解码

由于裁判系统数据较为复杂，故使用流程图进行分析。请结合代码、裁判系统通信协议及本流程图进行理解与分析。

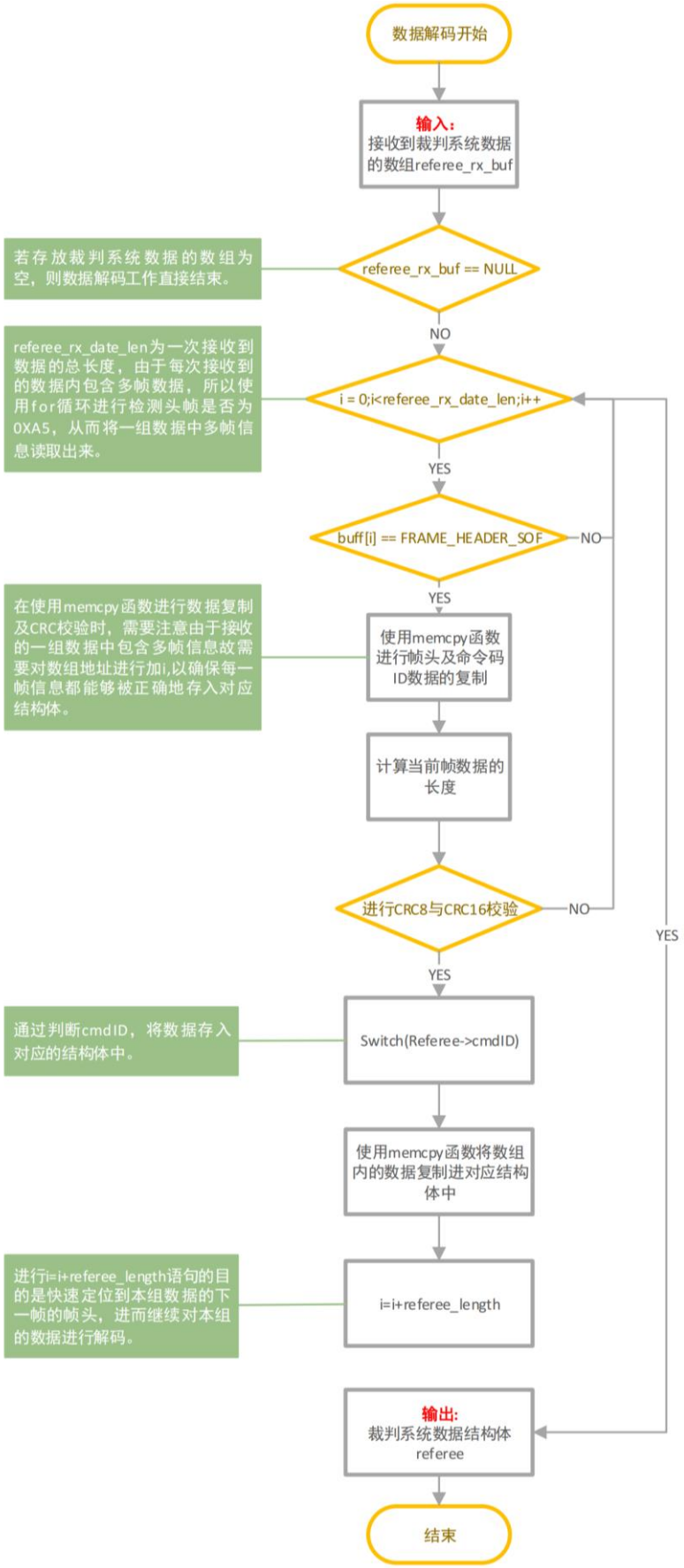


图 3-18 裁判系统数据解码流程图

4、数据发送

主要是通过裁判系统进行机器人与机器人之间的通信，裁判系统与客户端的通信，目前暂未添加后续将逐步完善。

3.3.3 串口打印 usart_printf.c

该文件是使用串口进行数据打印，包括 printf 重定向、虚拟示波器打印相关函数。具体说明请查看该代码注释。

3.3.4 遥控器数据接收 remote_control.c

该文件是遥控器数据接收与解码及发送，包括 USART 初始化、UASRT 中断处理、数据解码、数据发送函数。

遥控器数据接收文件与视觉数据接收文件除数据解码方式不同外其余内容基本一致，因此不再赘述。

3.3.5 PID 无线调参 pid_wireless_debug.c

该文件是 PID 无线调参文件，包括 USART 初始化、USART 接收中断处理、数据解码、数据发送函数。

目前对 PID 无线调参功能进行测试时使用的是 USART2，与串口打印所用串口冲突，后期需要通过宏改为 USART8，并进一步测试。

1、USART 初始化

使用库中的__HAL_UART_ENABLE_IT()宏开启串口接收中断。

2、UASRT 中断处理

详解待续

注意事项：由于未按照库中 HAL_UART_IRQHandler 函数的方式接收数据，而是重新编写了一段接收函数，方便按照自己要求进行数据接收。故需要将 HAL_UART_IRQHandler 函数注释掉。并调用自己写的中断处理函数 PW_IRQHandler()。

3、数据解码

详解待续

4、数据发送

详解待续

此功能文件需要配合南京理工大学开源的上位机软件进行使用。调参软件及数据格式说明文件位于 PID 调参文件夹中。若有兴趣进一步学习可参考开源文件，其位于 PID 调参文件夹的 PID_Regulator-master 压缩包中。

图 3-19　PID 无线调参软件界面

使用方法：根据程序中对不同电机的速度环或位置环设定的不同 PID_ID 号，进而对各电机速度环或位置环 PID 各参数进行设置。

```
typedef enum
{
  PID_CHASSIS_3508MOTOR_SPEED,
  PID_CHASSIS_3508MOTOR_POS,
  PID_TRIGGER_2006MOTOR_SPEED,
  PID_TRIGGER_2006MOTOR_POS,
  PID_YAW_3508MOTOR_ID_SPEED,
  PID_YAW_3508MOTOR_ID_POS,
  PID_PITCH_6020MOTOR_ID_SPEED,
  PID_PITCH_6020MOTOR_ID_POS,

  NUM_OF_PID//所有电机进行PID运算时所需PID结构体的数量

}PID_ID;//电机闭环控制时各电机速度环或位置环PID结构体对应存储的数组编号
```

图 3-20 各电机速度环或位置环的 PID_ID 号

注意事项：在使用 PID 无线调参时，需要将所有电机的 PID 初始化函数注释掉，例如对底盘电机进行无线调参时将底盘任务中的 Motor3508_PID_Init()函数注释掉。所有参数均可通过调参软件进行设定，断电或复位后参数自动清零。

3.4 Motor_Driver Module 文件

3.4.1 定时器 PWM 驱动电机 motor_use_tim.c

该文件为使用 TIM 的 PWM 控制电机的库，包含 PWM 初始化、比较值设定函数。

注意事项：

1、定时器设置

① 选择定时器。首先需要选择具备 PWM 输出的功能的定时器，各定时器功能如图 3-21 所示。

定时器类型	Timer	计数器分辨率	计数器类型	预分频系数	DMA 请求生成	捕获/比较通道	互补输出	最大接口时钟(MHz)	最大定时器时钟(MHz)
高级控制	TIM1 和 TIM8	16 位	递增、递减、递增/递减	1~65536(整数)	有	4	有	90 (APB2)	180
通用	TIM2，TIM5	32 位	递增、递减、递增/递减	1~65536(整数)	有	4	无	45 (APB1)	90/180
	TIM3，TIM4	16 位	递增、递减、递增/递减	1~65536(整数)	有	4	无	45 (APB1)	90/180
	TIM9	16 位	递增	1~65536(整数)	无	2	无	90 (APB2)	180
	TIM10，TIM11	16 位	递增	1~65536(整数)	无	1	无	90 (APB2)	180
	TIM12	16 位	递增	1~65536(整数)	无	2	无	45 (APB1)	90/180
	TIM13，TIM14	16 位	递增	1~65536(整数)	无	1	无	45 (APB1)	90/180
基本	TIM6 和 TIM7	16 位	递增	1~65536(整数)	有	0	无	45 (APB1)	90/180

定时器种类	位数	计数器模式	产生DMA请求	捕获/比较通道	互补输出	特殊应用场景
高级定时器（TIM1,TIM8）	16	向上，向下，向上/下	可以	4	有	带可编程死区的互补输出
通用定时器（TIM2,TIM5）	32	向上，向下，向上/下	可以	4	无	通用。定时计数，PWM输出，输入捕获，输出比较
通用定时器（TIM3,TIM4）	16	向上，向下，向上/下	可以	4	无	通用。定时计数，PWM输出，输入捕获，输出比较
通用定时器（TIM9~TIM14）	16	向上	没有	2	无	通用。定时计数，PWM输出，输入捕获，输出比较
基本定时器(TIM6,TIM7)	16	向上，向下，向上/下	可以	0	无	主要应用于驱动DAC

图 3-21 各定时器功能说明

② 设定定时器计数周期。snail 电机与航模电机类似，一般信号周期为

20ms。

例：对于 TIM5 通用定时器，时钟来源为 APB1,由我们时钟配置可知其时钟为 84MHz，固可设定预分频系数（Prescaler）为 840-1，自动重装载值（Period）为 2000-1；或 Prescaler = 84-1，Period = 20000-1 等。

③ 设定初始比较/PWM 通道的值（Pulse）。建议设置为 0。

2、摩擦轮电机设置

① PWM 行程校准。对于 snail 电机来说占空比一般需要设定在 5%～10%之间。根据 C615 说明书，先将比较值设定为最高即 10%所对应的值（2000*0.1=200），后延时 2s，再将比较值设定为最低 5%对应的值（2000*0.05=100），后延时 1s，即完成初始化。430 电调设置相同。

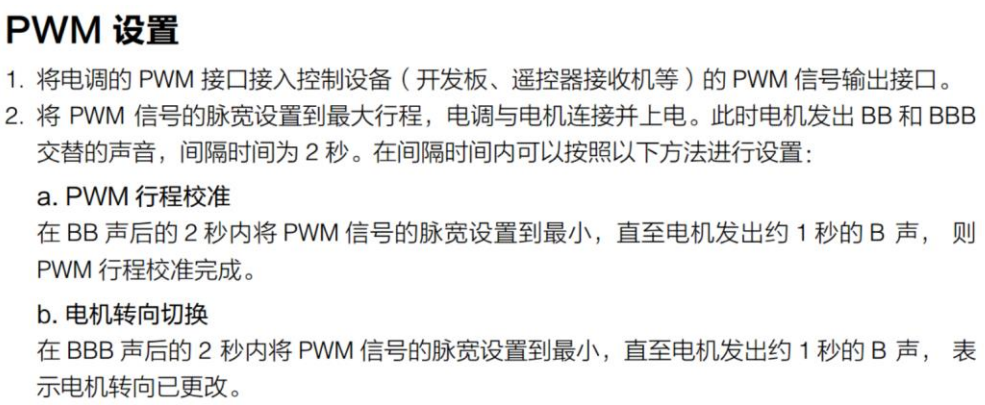


图 3-22　PWM 设置

② 电机转向切换。对于 C615 可以根据上图进行设置转向。首先进入 Debug 模式单步运行，在 Shoot_Firction_Motor(HIGH_SPEED,HIGH_SPEED)之前打断点（第一处断点），在下面 HAL_Delay(2000)语句之前打断点（第二处断点）。点击运行程序，当运行至第一处断点时，电调发出 BB 和 BBB 交替的声音，在电调发出 BBB 声时再次点击运行程序，若电调发出约 1 秒的 B 声，表示电机转向已更改。

③ PWM 行程校准完成后，可在摩擦轮任务中通过斜波函数将比较设定值逐渐增加，最终达到一个目标速度。关于斜波函数具体使用方法及计算方式请自行看程序源码。

④ 上述步骤全部完成后，摩擦轮即开始正常运转，当需要改变转速时，可通过改变 max_value 的值改变比较设定值，从而达到增加或降低转速的效果。当 max_value 设定为 100 即最低占空比对应的比较值时，则电机停止转动。程序参考如图 3-23 所示。

```
//斜坡函数初始化
ramp_init(&shoot_control.fric1_ramp, 0.01f, 120,100);//0.01为斜坡函数的斜率(时间间隔)  120和100为最大、最小值(最大值也为目标值)
ramp_init(&shoot_control.fric2_ramp, 0.01f, 120,100);

for(;;)
{
  Refresh_Task_OffLine_Time(FrictionDriveTask_TOE);//记录任务运行的时间点
  if(rc_ctrl.rc.s1 == 3)
  {
    shoot_control.fric1_ramp.max_value = 100;
    shoot_control.fric2_ramp.max_value = 100;
  }
  else if(rc_ctrl.rc.s1 == 1)
  {
    shoot_control.fric1_ramp.max_value = 125;
    shoot_control.fric2_ramp.max_value = 125;
  }
  else if(rc_ctrl.rc.s1 == 2)
  {
    shoot_control.fric1_ramp.max_value = 130;
    shoot_control.fric2_ramp.max_value = 130;
  }

   ramp_calc(&shoot_control.fric1_ramp, SHOOT_FRIC_PWM_ADD_VALUE);
   ramp_calc(&shoot_control.fric2_ramp, SHOOT_FRIC_PWM_ADD_VALUE);

   Shoot_Firction_Motor(shoot_control.fric1_ramp.out,shoot_control.fric2_ramp.out);
```

图 3-23 摩擦轮驱动程序

3.4.2 CAN 驱动电机 motor_use_can.c

该文件为使用CAN通信控制电机的库，包含各电机的CAN发送函数、接收函数及 CAN 的中断回调函数。

注意事项：

① 若需要增加 CAN 发送或接收函数请于后面进行添加。

② 该程序对所有电机进行了编号，用于 CAN 数据接收时将数据存入对应结构体数组中，使程序更加简洁清晰。如图

```
typedef enum
{
  CHASSIS_3508MOTOR,
  TRIGGER_2006MOTOR,
  GIMBAL_YAW_3508MOTOR,
  GIMBAL_PIT_6020MOTOR,

  NUM_OF_MOTOR      //电机数量
}motor_ID;//电机数据接收结构体对应存储的数组编号
```

图 3-24 电机编号结构体

注：请注意区分电机编号、电机电调 ID 号、电机 PID 编号

电机编号：用于将接收到的 CAN 数据存入对应编号的结构体数组中。

电机电调 ID 号：用于判断接收到的某一帧数据是哪个电机发送的。

电机 PID 编号：用于将各电机速度环或位置环的 PID 数据存入对应编号数组结构体中.

```
typedef enum
{
  CAN_CHASSIS_3508MOTOR_ID = 0x201,
  CAN_TRIGGER_2006MOTOR_ID = 0x203,
  CAN_YAW_3508MOTOR_ID = 0x205,
  CAN_PITCH_6020MOTOR_ID = 0x206,
  CAN_SEND_REMOTEDATE_ID = 0x110,
  CAN_SEND_REFEREEDATE_ID = 0x120,

}CAN_Message_ID;//CAN通信时电机电调对应ID号
```

图 3-25 电机电调 ID 号

```
typedef enum
{
  PID_CHASSIS_3508MOTOR_SPEED,
  PID_CHASSIS_3508MOTOR_POS,
  PID_TRIGGER_2006MOTOR_SPEED,
  PID_TRIGGER_2006MOTOR_POS,
  PID_YAW_3508MOTOR_ID_SPEED,
  PID_YAW_3508MOTOR_ID_POS,
  PID_PITCH_6020MOTOR_ID_SPEED,
  PID_PITCH_6020MOTOR_ID_POS,

  NUM_OF_PID//所有电机进行PID运算时所需PID结构体的数量

}PID_ID;//电机闭环控制时各电机速度环或位置环PID结构体对应存储的数组编号
```

图 3-26 电机 PID 编号

3.4.2 USART 驱动电机 motor_usart_usart.c

该文件适用于特定的舵机驱动。

详解待续

3.5 Application Module 文件

3.5.1 CRC 校验 crc.c

该文件为配合裁判系统数据接收的校验文件，通过 CRC 校验来判断接收到的裁判系统数据是否错误。在 PID 无线调参中同样使用到了 CRC 校验函数。

CRC 校验原理：

1、循环校验码（CRC 码）：

是数据通信领域中最常用的一种差错校验码，其特征是信息字段和校验字段的长度可以任意选定。

2、生成 CRC 码的基本原理：

任意一个由二进制位串组成的代码都可以和一个系数仅为‘0’和‘1’取

值的多项式一一对应。例如：代码 1010111 对应的多项式为 x6+x4+x2+x+1，而多项式为 x5+x3+x2+x+1 对应的代码 101111。

3、CRC 码集选择的原则：

若设码字长度为 N，信息字段为 K 位，校验字段为 R 位(N=K+R)，则对于 CRC 码集中的任一码字，存在且仅存在一个 R 次多项式 g(x)，使得

$$V(x)=A(x)g(x)=x^{R}m(x)+r(x);$$

其中：m(x)为 K 次信息多项式，r(x)为 R-1 次校验多项式，

g(x)称为生成多项式：

$$g(x)=g_0+g_1x+g_2x^2+...+g_{(R-1)}x^{(R-1)}+g_Rx^R$$

发送方通过指定的 g(x)产生 CRC 码字，接收方则通过该 g(x)来验证收到的 CRC 码字。

4、CRC 校验码软件生成方法：

借助于多项式除法，其余数为校验字段。

例如：

信息字段代码为：1011001；对应 $m(x)=x^6+x^4+x^3+1$

假设生成多项式为：$g(x)=x^4+x^3+1$；则对应 g(x)的代码为：11001

$x^4m(x)=x^{10}+x^8+x^7+x^4$ 对应的代码记为：10110010000；

采用多项式除法：得余数为：1010（即校验字段为：1010）

发送方：发出的传输字段为：10110011010　信息字段校验字段

接收方：使用相同的生成码进行校验：接收到的字段/生成码（二进制除法）

如果能够除尽，则正确。

CRC 分析可参考：https://blog.csdn.net/nicholas199109/article/details/8452501

3.5.2 PID 运算 pid.c

该文件为 PID 运算文件，包括 PID 初始化、PID 计算、PID 清零函数。详解待续

注：该文件为修改后版本，将原来的指针函数形式参数初始化修改成直接给参。原方式请见 Module —> Module-Application —> old_pid 文件夹中。

3.5.3 IMU 运算 imu.c

该文件为 IMU 运算文件，包括 IMU 初始化、IMU 数据读取、IMU 数据解算等。详解待续

3.5.4 外接编码器 encoder.c

该文件是使用于读取外接编码器的转动圈数及转动方向。包含编码器所用 TIM 初始化函数、获取 TIM 计数值、外部中断回调函数。

1、编码器

型号：龙邱增量式 Mini512Z 型

输出：步进+方向输出 512 线。

线序：如图 3-26 所示。本工程中使用时只需要连接 VCC、GND、LSB、DIR 即可。Z 相读取的是编码器的绝对零点，若有需求也可以使用。

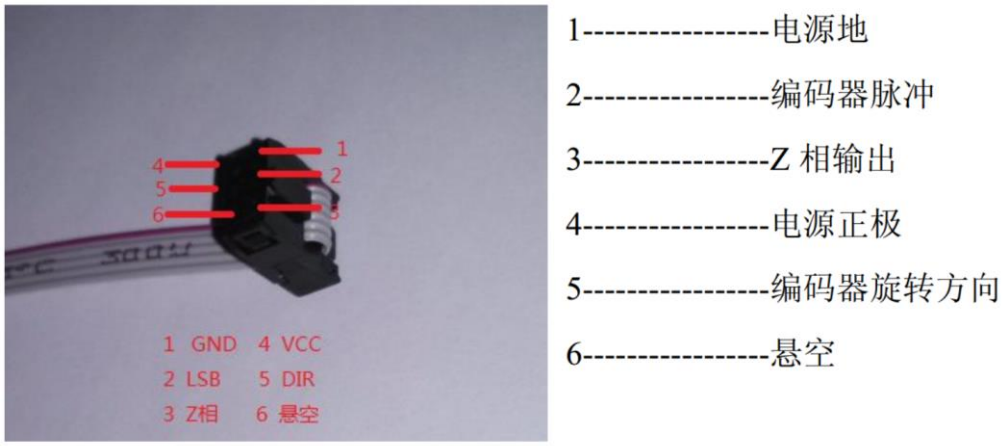


图 3-26 编码器线序图

输出波形：如图 3-27 所示

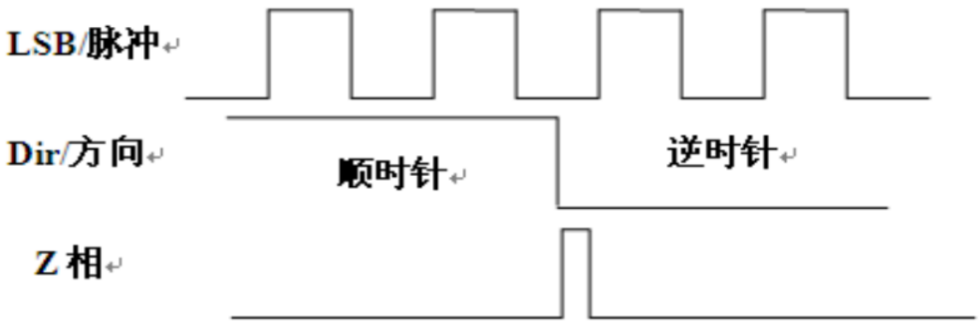


图 3-27 输出波形图

2、脉冲读取方式

脉冲读取思路：通过将 TIM 时钟源设置为外部时钟模式 2 [外部触发输入 (ETR)]来获取脉冲数量，通过外部中断判断旋转方向从而改变 TIM 计数方向。这样可直接通过 TIM 得到编码器旋转圈数。通过 TIMx 控制寄存器 1 (TIMx_CR1)的位 4 [DIR：方向 (Direction)]，可改变计数方式。

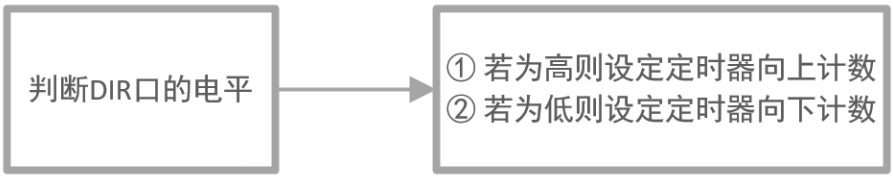


图 3-28 计数方式图

注：不建议使用输入捕获进行捕获脉冲，原因是输入捕获的原理是有上升沿或下降沿信号时将触发中断，并在中断中进行信号的相关处理。该方法确实可以起到计脉冲的作用，但是对于 512 线的编码器就意味着编码器转一圈就会触发 512 次中断，这样频繁进入中断就会在很大程度上影响后台程序的运行。所以不建议使用输入捕获的方式。

3、TIM 设置

在 CubeMX 中将 TIM 时钟源设置为 ETR2（注意使用的 IO 口），同时将 TIM 的自动重装载值设定为 TIM 最大计数值（TIM2 为 32 位的定时器所以设位 0xFFFFFFFF）。

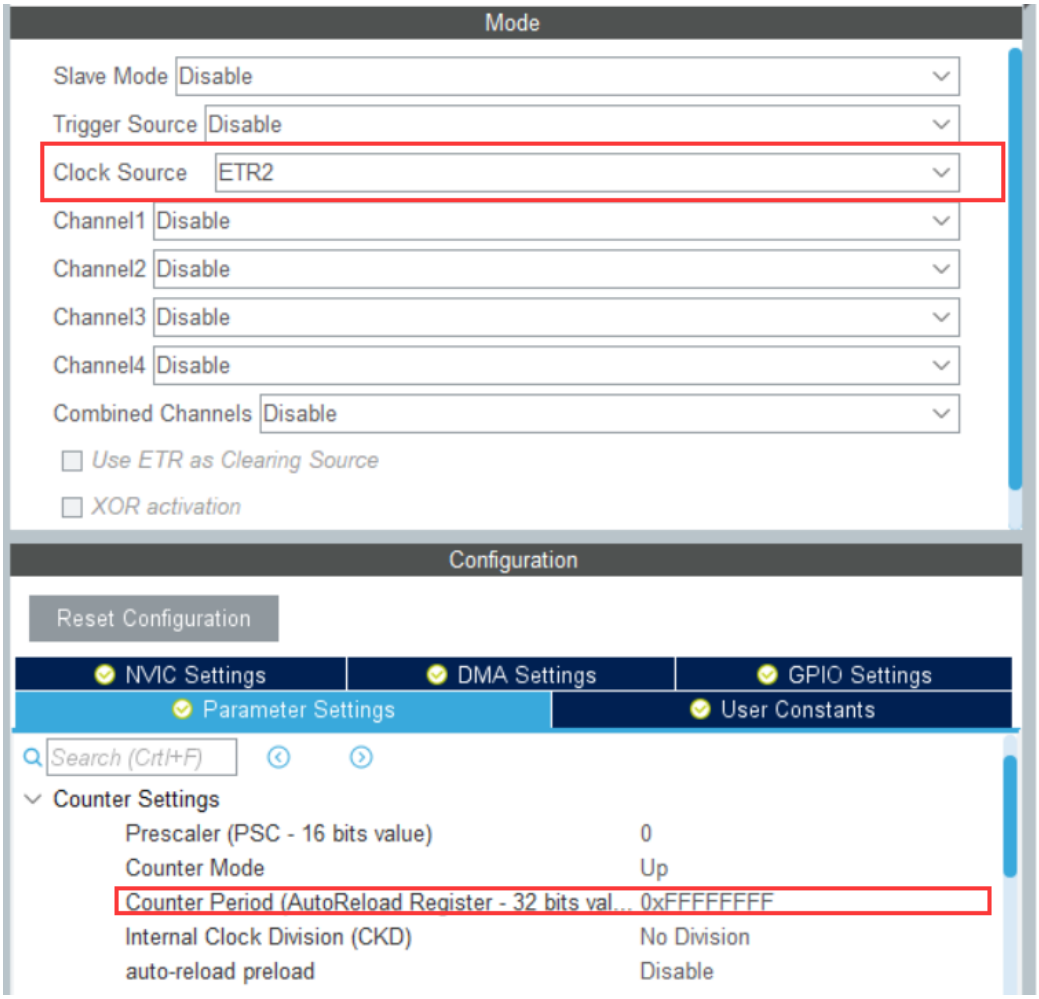


图 3-28 TIM 在 CubeMX 中的设定

3.5.5 断线检测 offline_check.c

该文件为外设及任务的断线检测，主要包括定时器初始化、记录时间等函数。

主要检测流程为，记录每个任务每次运行或每个外设每次接收数据的时间点、记录断线检测程序每次运行的时间点，两者进行相减求出时间差，当该时间差大于某一值时，则判定为断线。

目前检测到断线的操作为：

① 通过 printf 函数进行打印某任务或某外设掉线。

② 每个外设对应一个 LED，当该外设掉线时对应 LED 开始闪烁。

③ 设定一个低优先级的任务，其内容为 GREEN_LED 以周期为 1s 进行闪烁，当其他任务卡死时，该任务则不会继续运行，GREEN_LED 就不会闪烁。

注：该断线检测方式较为简单，可参考官方 C 板整车的例程或 19 年官方开源步兵程序进行优化。

3.5.6 麦轮解算 mecaum.c

给文件为麦轮解算，这里不做详细解释。

注：该麦轮解算方式较为简单，可以参考官方 C 板整车的例程或 19 年官方开源步兵程序进行优化。

3.5.7 底盘功率限制 power_restriction.c

该文件尚未整理完成，详解待续

3.6 Algorithm Module 文件

该文件主要是一些算法库文件的汇总，目前只是调用了 CubeMX 生成的工程中已有的 arm_cortexM4lf_math.lib 库中的 arm_math.h 文件。在使用该文件时需要做如下几步：

① 在工程中添加 arm_cortexM4lf_math.lib 库。该库位于工程文件下的 Drivers->CMSIS->lib->ARM 文件夹中。

图 3-29　arm_cortexM4lf_math.lib 所在位置

② 需要设置全局宏，按照图 3-30 步骤将下面宏复制进去即可。原因自行百度。

USE__DRIVER,STM32F427xx,ARM_MATH_CM4,__CC_ARM,ARM_MATH_MATRIX_CHECK,ARM_MATH_ROUNDING,__FPU_PRESENT=1U,__FPU_USED=1U

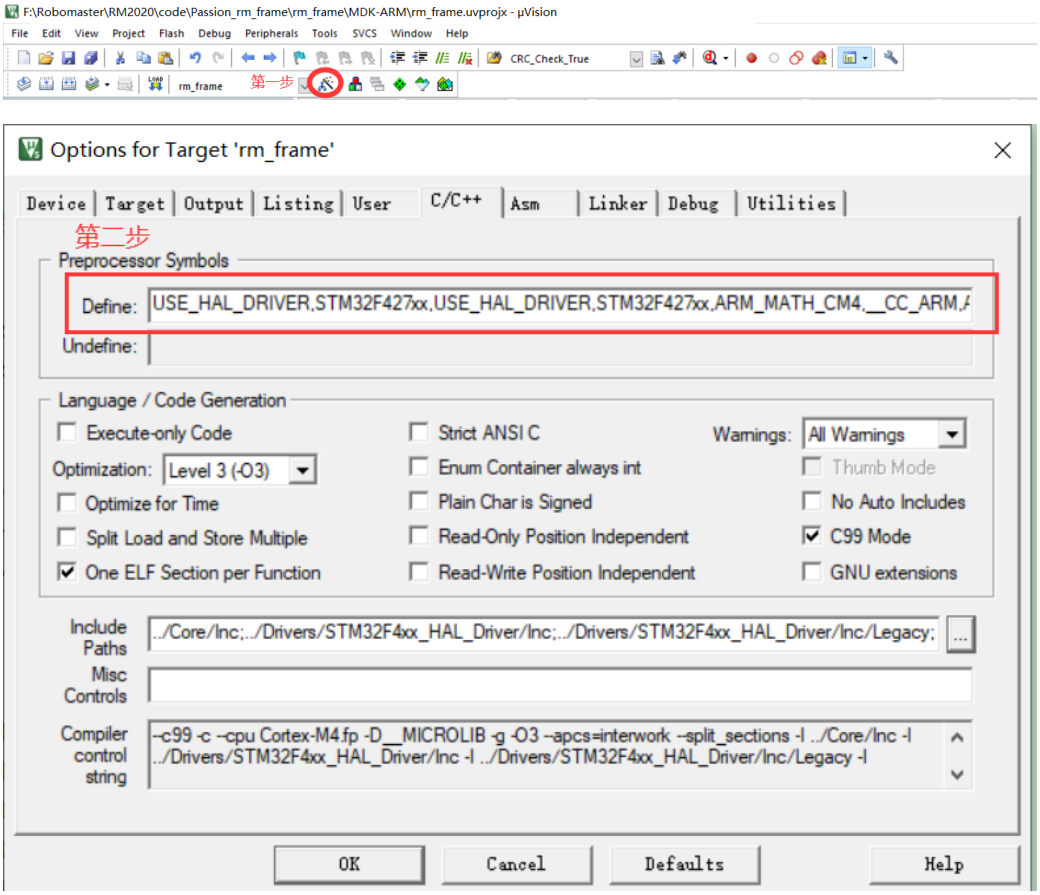


图 3-30　添加全局宏

③ 将 arm_math.h 文件的路径包含进工程。该文件位于工程文件下的 Drivers->CMSIS->DSP->include 文件夹中。

图 3-31　arm_math.h 所在位置

usb_lib.c 文件中是一些常见算法的合集，后续可以将其他常见算法放在该文件中，方便统一管理。

3.7 Boards_BSP 文件

3.7.1 初始化汇总 bsp.c

该文件中主要是 BSP_Init()函数，该函数中包含了所有外设或功能模块的初始化函数，从而可以使在 main.c 中的 main 函数中仅需调用该函数即可完成所有初始化，方便管理。

注：初始化不包含 CubeMX 自动初始化的部分，例如[MX_外设名_Init]与[HAL_外设名_Msplnit]两个函数，前者作用为外设具体配置（外设模式等），后者为外设和 CPU 连接的引脚配置(开时钟，配置 GPIO 引脚的模式、速度，开启中断等)。

3.7.2 CAN 初始化 API 接口 bsp_can.c

该文件为设置 CAN 通信滤波参数、开启 CAN 传输及 CAN 接收中断。

3.7.3 UASRT 初始化 API 接口 bsp_usart.c

该文件主要是封装 UASRT 初始化所需要的 API 接口函数，方便调用。其中 Bsp_UART_Receive_DMA 的主要目的替换 HAL 库中开启 DMA 数据接收的函数 HAL_UART_Receive_DMA。通过对比可发现两个函数的根本区别是 Bsp_UART_Receive_DMA 中去掉了进入各种中断的部分。

```
/* Set the UART DMA transfer complete callback */
huart->hdmarx->XferCpltCallback = UART_DMAReceiveCplt;

/* Set the UART DMA Half transfer complete callback */
huart->hdmarx->XferHalfCpltCallback = UART_DMARxHalfCplt;

/* Set the DMA error callback */
huart->hdmarx->XferErrorCallback = UART_DMAError;

/* Set the DMA abort callback */
huart->hdmarx->XferAbortCallback = NULL;
```

图 3-32　Bsp_UART_Receive_DMA 函数中主要去除部分

这样做的目的在于为串口开启没有中断的 DMA 传输，为了减少中断次数为其他中断空出资源。

3.7.4 自创头文件 myinclude.h

该头文件主要是包含例如<stdio.h>的系统头文件， "can.h"系统初始化头文件，以及其他宏定义等。主要目的方便头文件管理。后续有待优化。

第四章　程序书写规范