第 5 章

Tracealyzer 集成和配置指南

5.1 Tracealyzer 实验 1 Tracealyzer 介绍

本节是一个安装指南，帮助用户在运行 FreeRTOS 的 STM32F4 微控制器上集成和配置 Tracealyzer。这部分内容能够让项目正常运行起来。

首先，下载和安装 Tracealyzer for FreeRTOS 版本，请访问：

https://percepio.com/tz/freertostrace/和 https://percepio.com/downloads/。

译者注：Tracealyzer 新版本安装程序不划分 RTOS。本书实验作者和译者使用的是 Tracealyzer v4.3.4 版本。

软件安装完成之后，需要执行 4 个关键操作，以确保工具配置成功。

(1) 将 Percepio 跟踪记录器库(Trace Recorder Library)加入项目。

(2) 将 FreeRTOS 与 Tracealyzer 记录器集成。

(3) 配置 CubeMX 项目以满足 Tracealyzer 工具要求。

(4) 修改项目源码来初始化并启动跟踪记录。

可以在下面的网址找到详细资料：

https://percepio.com/docs/FreeRTOS/manual。

如果手册的内容与本书有出入，请以手册的信息为准。此外，对 Tracealyzer 工具升级也可能造成内容冲突。注意：Tracealyzer 实验相关的代码文件及 Tracealyzer 捕获截图文件将发布在 www.hexiaoqing.net 图书栏目，读者可以参考。

5.2 集成跟踪记录器库

5.2.1 将记录器库添加到项目

记录器库的详细资料载于：

https://percepio.com/docs/FreeRTOS/manual/Recorder.html#。

记录器库作为 Tracealyzer 的一部分，存在于独立的文件夹中，如图 5.1 所示，通常安装

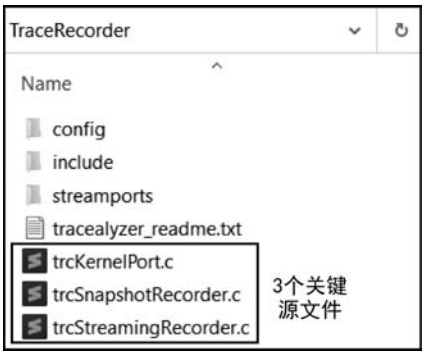

图 5.1 跟踪记录器库的内容

在 C:\Program Files\Percepio\Tracealyzer 4\FreeRTOS。

从图 5.1 中可以看到,记录器库包含 3 个关键源文件,另外 2 个关键的目录是 config 和 include(暂时忽略 streamports 文件夹),其详细内容如图 5.2 所示。

需要将 config、include 目录,以及源文件添加到项目适当的位置,这没有统一的操作方法,取决于个人的选择和项目使用的 IDE。不过,在项目中必须看得到所有的记录库文件。同样地,具体的实现依赖于特定的项目。图 5.3 给出了 Lab2 项目的文件目录结构,该项目使用 Keil μVision IDE。

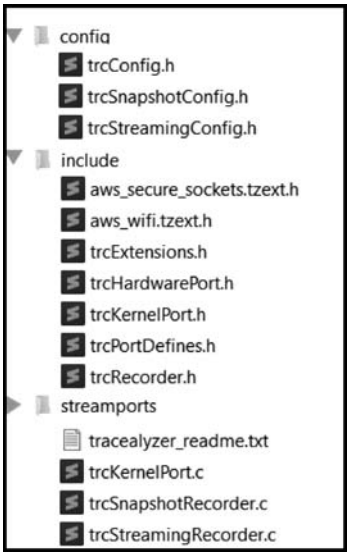

图 5.2 详细的跟踪记录器库内容

图 5.3 示例项目文件结构

5.2.2 为应用配置库文件

如图 5.2 所示，config 目录下有 3 个配置文件。在第一部分实验中，我们会使用快照模式。因此，暂时忽略 trcStreamingConfig.h。对于 trcSnapshotConfig.h 文件，使用默认设置即可，目前不需要修改。但是，需要根据特定的项目修改 trcConfig.h。

请阅读以下文档，了解相关的配置信息：

https://percepio.com/docs/FreeRTOS/manual/Recorder.html#config；

https://percepio.com/docs/FreeRTOS/manual/Recorder.html#tracedetails。

作为一个快速参考，下面列出的项目 1、2 和 3 需要在 trcConfig.h 中更新。

```
项目 1 - 配置前
/*******************************************************************************
 * 包含处理器头文件
 *
 * 这里可能需要包含处理器的头文件，至少对于使用 ARM CMSIS API 的 ARM Cortex-M 的移植
 * 需要，在遇到构建问题时可以尝试，否则，移除下面的#error 行
 ******************************************************************************/
#error "Trace Recorder: Please include your processor's header file here and remove this line."
/*******************************************************************************
项目 1 - 配置后
 ******************************************************************************/
//#error "Trace Recorder: Please include your processor's header file here and remove this line."
#include "stm32f4xx.h"
/*******************************************************************************
```

```
项目 2 - 配置前
/*******************************************************************************
Configuration Macro: TRC_CFG_HARDWARE_PORT
 ******************************************************************************/
#define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_NOT_SET
/*******************************************************************************
项目 2 - 配置后
 ******************************************************************************/
//#define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_NOT_SET
#define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_ARM_Cortex_M
/*******************************************************************************
```

项目 3 配置系统使用 FreeRTOS 版本。你需要确认使用的 FreeRTOS 版本，到 Middlewares(见图 5.4)目录中的 FreeRTOS 下的 include 文件夹，打开 task.h，并找到 MACROS AND DEFINITIONS，确认 FreeRTOS 的版本来配置项目 3。

```
项目 3 - 配置前
/*************************************************************************
 * TRC_CFG_FREERTOS_VERSION
 *
 * 指定使用哪个版本的 FreeRTOS,不用更改,除非使用老版本的 FreeRTOS 的跟踪记录库
 *
 * TRC_FREERTOS_VERSION_7_3 If using FreeRTOS v7.3.x
 * TRC_FREERTOS_VERSION_7_4 If using FreeRTOS v7.4.x
 * TRC_FREERTOS_VERSION_7_5_OR_7_6 If using FreeRTOS v7.5.0 - v7.6.0
 * TRC_FREERTOS_VERSION_8_X If using FreeRTOS v8.X.X
 * TRC_FREERTOS_VERSION_9_0_0 If using FreeRTOS v9.0.0
 * TRC_FREERTOS_VERSION_9_0_1 If using FreeRTOS v9.0.1
 * TRC_FREERTOS_VERSION_9_0_2 If using FreeRTOS v9.0.2
 * TRC_FREERTOS_VERSION_10_0_0 If using FreeRTOS v10.0.0 or later
 *************************************************************************/
#define TRC_CFG_FREERTOS_VERSION TRC_FREERTOS_VERSION_10_0_0
/*************************************************************************
项目 3 - 配置后
 *************************************************************************/
#define TRC_CFG_FREERTOS_VERSION TRC_FREERTOS_VERSION_10_0_1
```

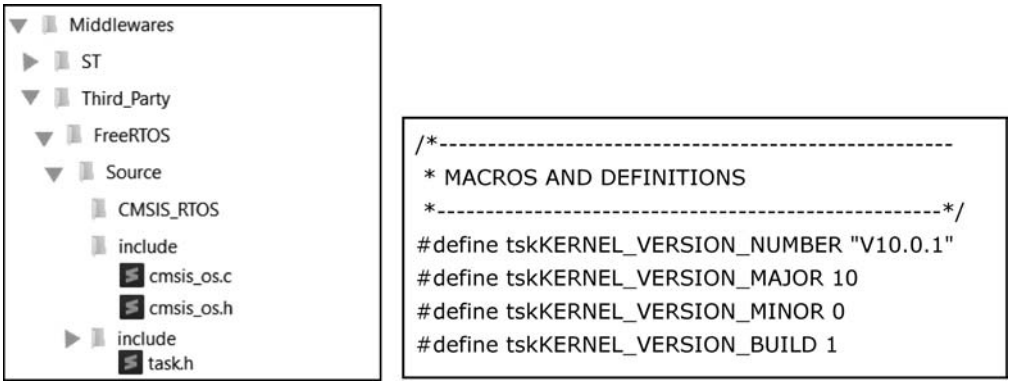

图 5.4 Middlewares 文件(部分)

5.3 在 FreeRTOS 中启用 Tracealyzer 记录器

在 FreeRTOSConfig.h 文件中有一个用于跟踪记录器的主开关。为了确保启用记录器,请检查是否包含以下设置:

```
#define configUSE_TRACE_FACILITY 1
```

注意:如果此设置为 0,则跟踪记录器将完全被禁用并从构建中排除。

检查下面的内容是否已经插入 FreeRTOSConfig.h 文件最后的用户定义部分。

```
/////////////////////////////////
/* 将 Tracealyzer 记录器与 FreeRTOS 集成 */
#if(configUSE_TRACE_FACILITY == 1)
#include "trcRecorder.h"
#endif
/////////////////////////////////
/* USER CODE END Defines */
```

5.4 配置 CubeMX 项目以符合工具需求

在 CubeMX 的 FREERTOS Mode and Configuration 区域的 Config Parameters 面板做以下设置：

(1) 将每个任务的最小堆栈大小设置为 512。

(2) 将 heap 的大小设置为 32000。

(3) 将 USE_TRACE_FACILITY 设置为 Enabled。

5.5 初始化/启动跟踪记录

Tracealyzer 有两种记录模式：快照(Snapshot)和数据流(Streaming)模式。具体信息参见下面的网址：

https://percepio.com/docs/FreeRTOS/manual/Recorder.html#Trace_Recorder_Library_Snapshot_Mode；

https://percepio.com/docs/FreeRTOS/manual/Recorder.html#Trace_Recorder_Library_Streaming_Mode。

5.5.1 快照跟踪模式

本节简要说明如何配置源代码使用快照模式。这个过程很简单，首先初始化系统，然后启动跟踪记录器。

(1) 初始化并开始记录。

在 main 函数中调用 vTraceEnable(TRC_START)。

(2) 初始化记录器，稍后开始跟踪。

首先在 main 函数中调用 vTraceEnable(TRC_INIT)；初始化记录器。然后在代码中需要的位置调用 vTraceEnable(TRC_START)，启动跟踪。

注意：TRC_INIT 调用必须在初始硬件设置之后，在创建任何 RTOS 对象(任务等)之前执行。初始化和跟踪的详细说明可以参考 Tracealyzer 手册中相应的内容，Tracealyzer

for FreeRTOS 链接为 https://percepio.com/docs/FreeRTOS/manual/Recorder.html#vtraceenab。

```
int main(void)
{
  /* 重置所有外设, 初始化 Flash 接口和 Systick */
  HAL_Init();
  /* 配置系统时钟 */
  SystemClock_Config();
  /* 用户代码开始 - 系统初始化 */
  vTraceEnable(TRC_INIT);
  vTraceEnable(TRC_START);
  /* 用户代码结束 - 系统初始化 */
  /* 初始化所有已配置的外设 */
  MX_GPIO_Init();
  /* 创建任务 */
  /* 定义并创建 LedFlashing 任务 */
  osThreadDef(LedFlashingTask, StartLedFlashingTask, osPriorityNormal, 0, 512);
  LedFlashingTaskHandle = osThreadCreate(osThread(LedFlashingTask), NULL);
  /* 启动调度器 */
  osKernelStart();
```

如果在应用测试期间复位开发板,启动 Tracealyzer 记录,将捕获 TraceEnable 调用之后发生的事件。每当你想重新记录时,请执行复位操作。

若在某些情况下希望从任务中选定的点开始记录,推荐的方法是首先初始化记录器,之后再开始记录,如下面的代码片段所示:

```
myBoardInit();
/* 仅初始化,之后再开始记录 */
vTraceEnable(TRC_INIT);
...
/* RTOS 调度器启动 */
vTaskStartScheduler();
...
/* 在任务或者中断中 */
vTraceEnable(TRC_START);
```

5.5.2 流跟踪模式

在源代码中插入如下调用:

```
vTraceEnable(TRC_START_AWAIT_HOST);
```

由此,在运行时(在记录器初始化之后),目标系统在此等待来自主机系统的开始命令。

示例代码片段如下：

```
myBoardInit();
...
/* 启动之后阻塞,等待主机开始命令 */
vTraceEnable(TRC_START_AWAIT_HOST);
...
/* RTOS 调度器启动 */
vTaskStartScheduler();
```

稍后将在第 7 章的 Tracealyzer 实验 7 中详细地讨论这个话题。

5.6 附加检查

1. 跟踪时间控制

在 trcConfig.h 中设置 TRC_CFG_INCLUDE_OSTICK_EVENTS 宏，确定记录多少事件。如果是 1，每当 OS 时钟增加时，都会产生事件。如果是 0，不会生成 OS Tick 事件，允许在相同的 RAM 中记录更长时间的跟踪。默认值是 1，在接下来的大多数实验中，它被设置为 0。

2. 设置快照模式中可以存储的事件数量

在 trcSnapshotConfig.h 中，TRC_CFG_EVENT_BUFFER_SIZE 宏定义事件缓冲区的容量。默认值是 1000(单位为字)，意味着为事件缓冲区分配了 4000 字节。

```
#define TRC_CFG_EVENT_BUFFER_SIZE 1000
```

如果希望记录更长的跟踪，请增大此值(可设置的最大值取决于下载的应用程序代码所使用的内存空间)。