自定义uORB消息实验

1. 实验目的

通过创建一个自定义的uORB消息实现读写功能，以此熟悉并掌握PX4的uORB消息系统。

2. 实验要求

- 软件要求：Windows 10及以上版本；RflySim工具链[1]，MATLAB2022B以上版本，平台安装时的编译命令为：px4_fmu-v6x_default，推荐PX4固件版本为：1.14.3。
- 硬件要求：笔记本/台式电脑1台[2]，遥控器和遥控器接收机；数据线和杜邦线等。

3.实验地址

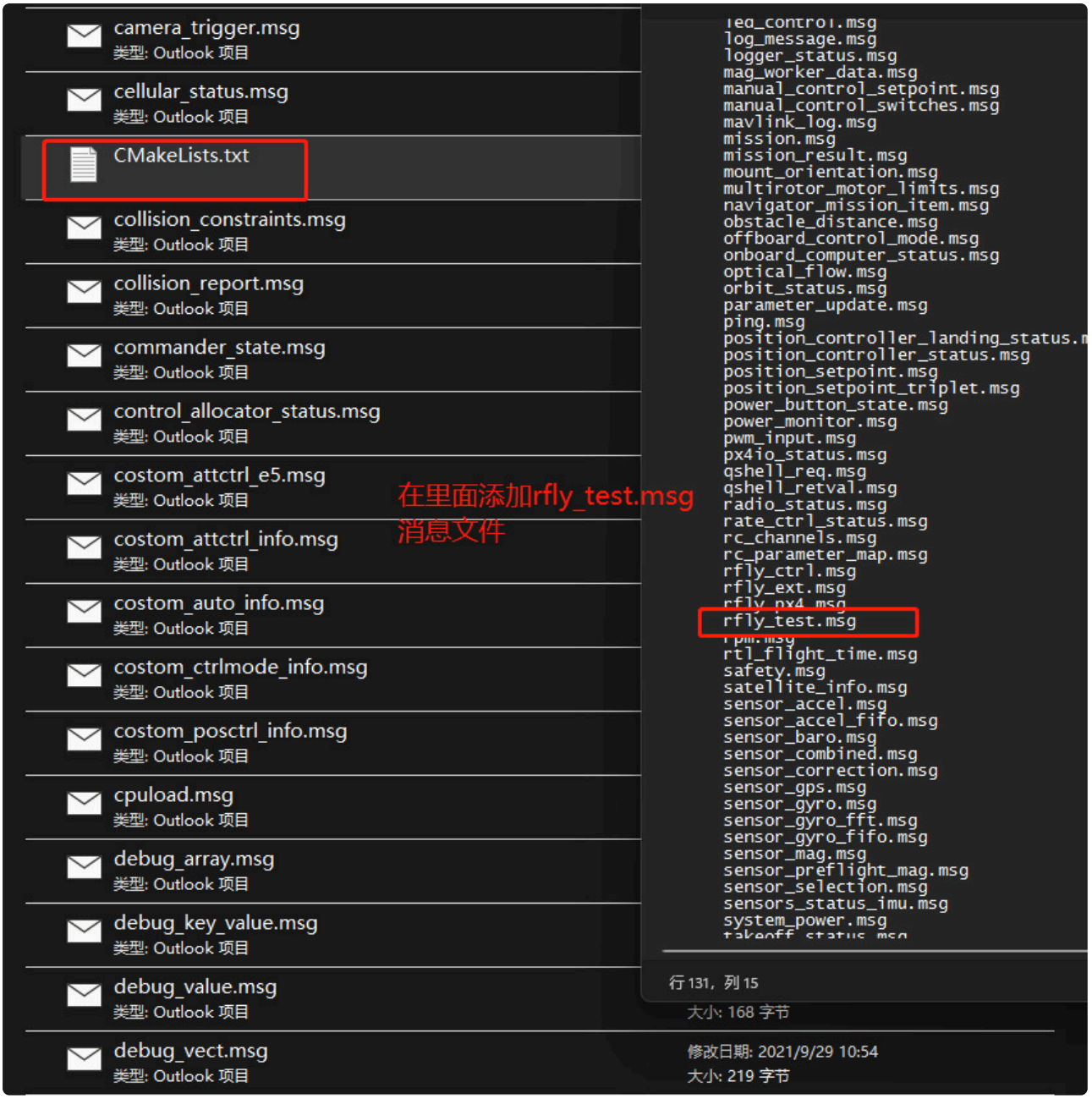
例程目录：[安装目录]\RflySimAPIs\5.RflySimFlyCtrl\0.ApiExps\7.uORB-Create

- PX4uORBMsgGen.m：uORB消息类型生成程序
- rfly_test.msg：uORB消息数据结构体文件
- init_control.m：初始化控制程序
- px4demo_uORB_create.slx：自定义uORB消息定义模型文件

4. 实验内容或步骤

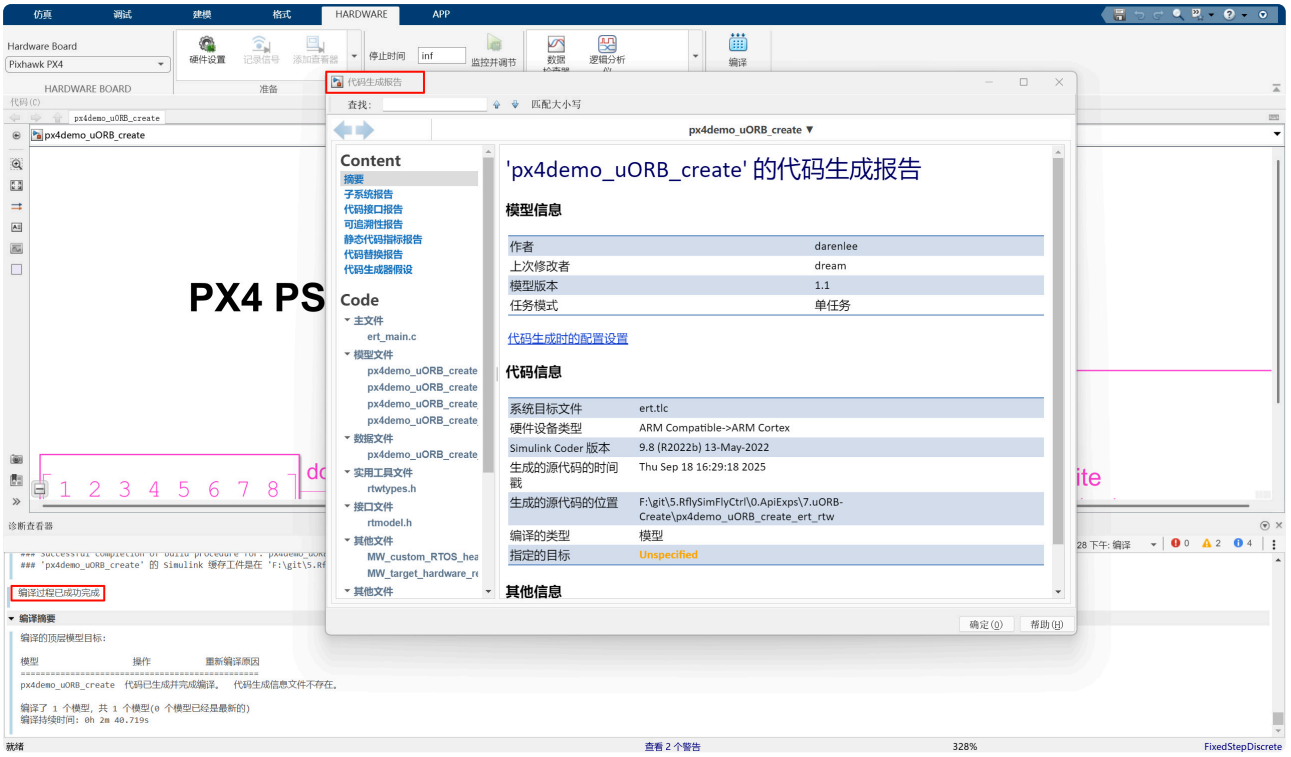
5.1 步骤1：自定义uORB消息实验

打开MATLAB软件，在MATLAB中运行init_control.m文件，即可将自定义的uORB消息加载到飞控固件中。可在“*\PX4PSP\Firmware\msg”中查看到rfly_test.msg文件，同时在本文件夹的CMakeList.txt中也可看到新增的rfly_test.msg消息。如下图所示。

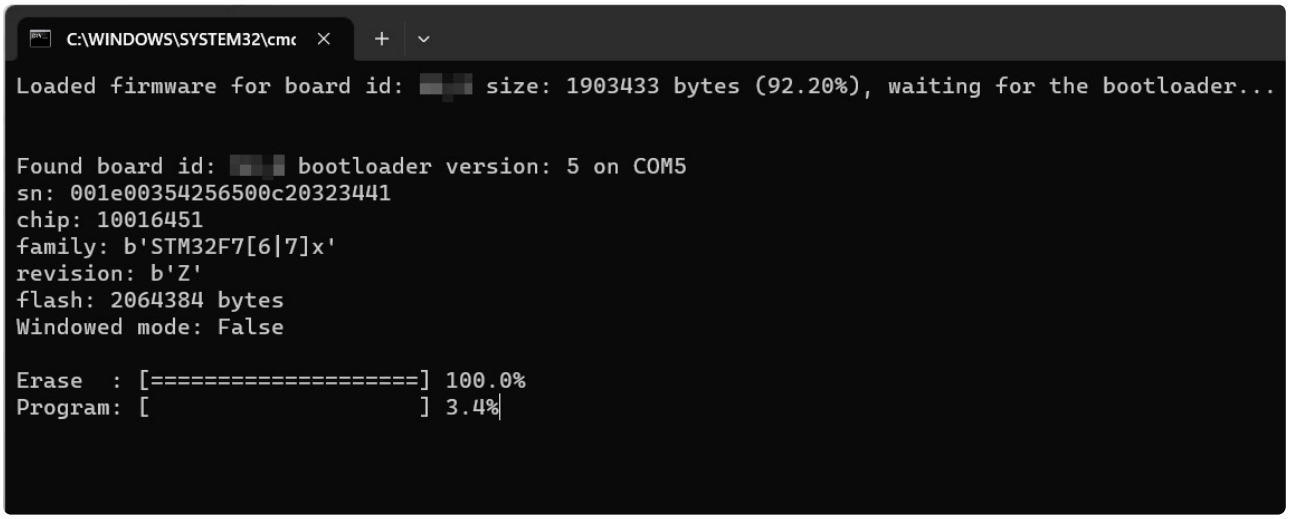


打开MATLAB软件，在MATLAB中打开px4demo_uORB_create.slx文件，在Simulink中，点击编译命令。

在Simulink的下方点击查看诊断，即可弹出诊断对话框，可查看编译过程。在诊断框中弹出编译过程已成功完成，即可表示编译成功，也会弹出代码生成报告。

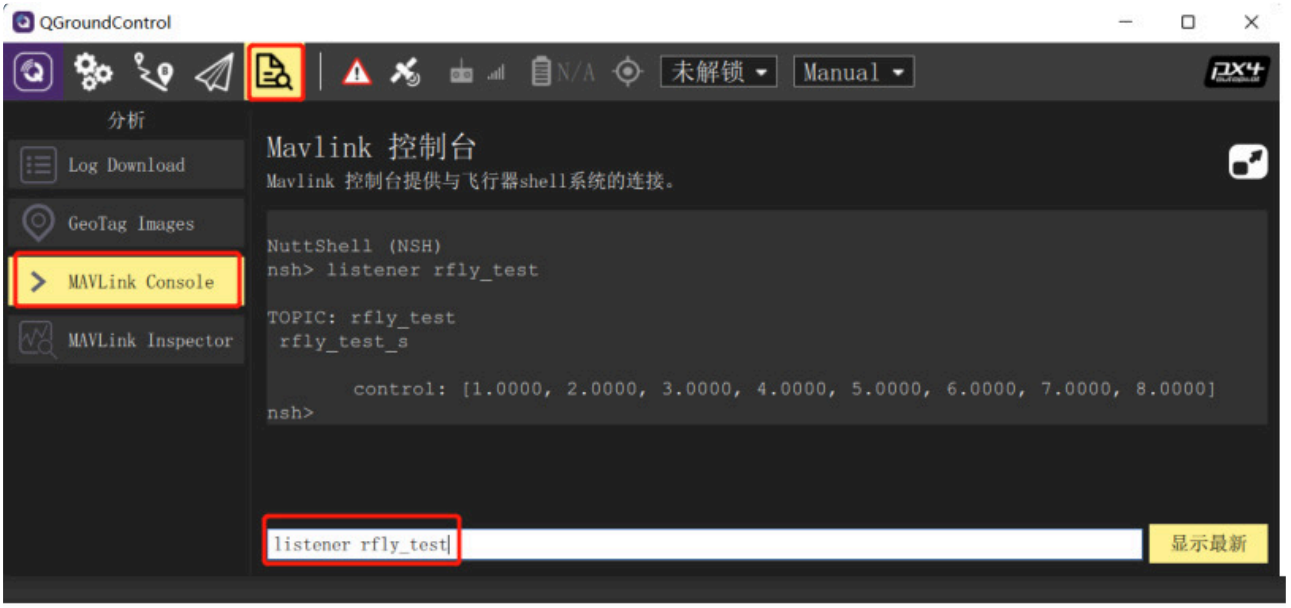


用USB数据线链接飞控与电脑。在MATLAB命令行窗口输入：PX4Upload并运行或点击PX4 PSP：Upload code to Px4FMU，弹出CMD对话框，显示正在上传固件至飞控中，等待上传成功。



打开QGroundControl软件，点击左上角Logo在弹出的对话框中，选中Analyze Tools，在Mavlink控制台中输入：listener rfly_test。

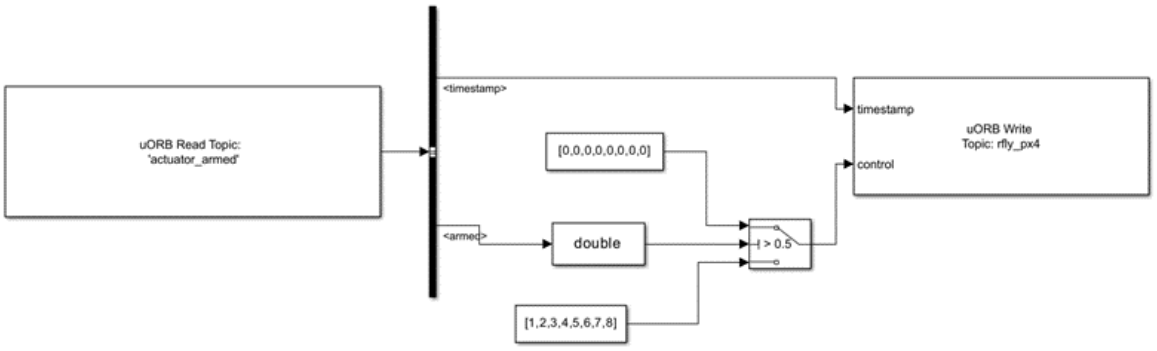
即可得出如下图的结果。



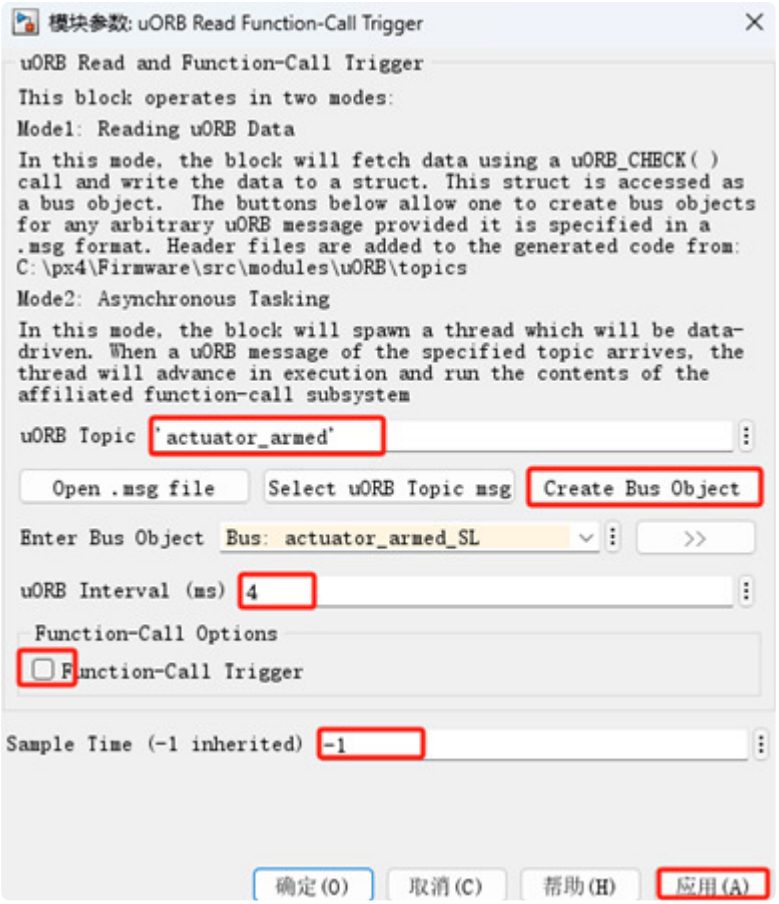
5. 关键知识点

关键知识点1：uORB消息读取实验原理

该模型用uORB Read Function-Call Trigger订阅'actuator_armed'消息，来接收是否解锁，如果未解锁，就输出12345678，反之则输出00000000，然后，数据通过rfly_px4消息发布出去，被其他程序接收。

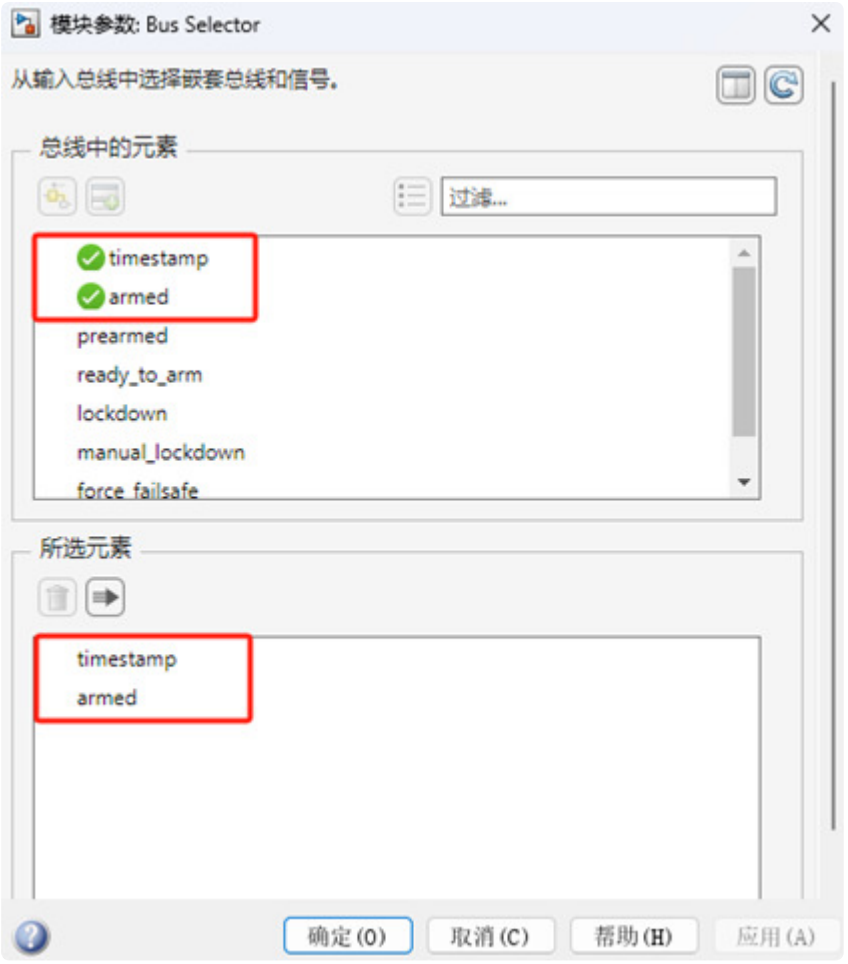


uORB Read Function-Call Trigger模块的关键配置如下：



订阅消息要选中actuator_armed'

1. 点击“Create bus Object”就能创建数据总线
2. uORB interval 时间间隔，这里设置为4ms，也就是250Hz，表示每隔多长时间检查消息更新。实际使用时根据情况设置，通常对于低延迟高频率的消息，可以设置成1ms，也就是1000Hz；这样做会多消耗一些计算量，但是能降低延迟。
3. Function-Call Options为触发函数的功能，本例子不需要使用。默认情况也不需要使用。
4. Sample Time仿真步长，可以设置为-1（使用默认步长，也就是Simulink设置页面的值，或者上一级数据源的步长），这里也可以直接设置为0.01。注意，所有模块的步长与设置页面的基础步长之间，应该成倍数关系。



设置好订阅模块后，就能从Bus Selector中，选择自己感兴趣的消息，进行订阅了。

本例中，选择了timestamp和armed标志位，其中armed表示是否解锁。

具体有哪些uORB消息，以及每个消息的定义，可以参考网站

https://docs.px4.io/main/en/middleware/uorb.html

或者在PX4PSP\Firmware\msg文件夹中自行查看注释。

Switch模块的含义是，如果未解锁，输出12345678，反之则输出00000000。

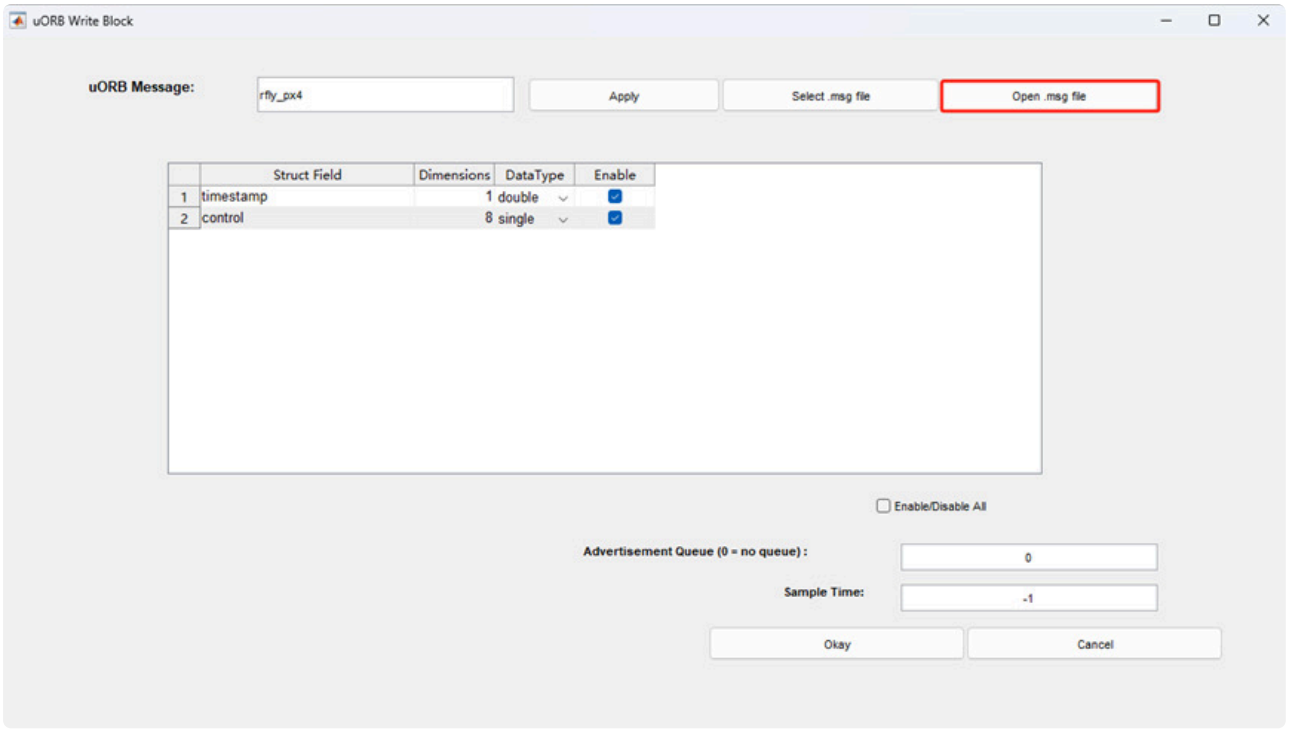
在 uORB Write Advanced1模块中订阅了rfy_px4消息，用来向其他程序发布数据。点击Open .msg file可查阅该uORB消息，该消息包含两个结构体，具体定义如下：

这是一个无符号64位整数，用于表示自系统启动以来经过的时间，单位是微秒（microseconds）。

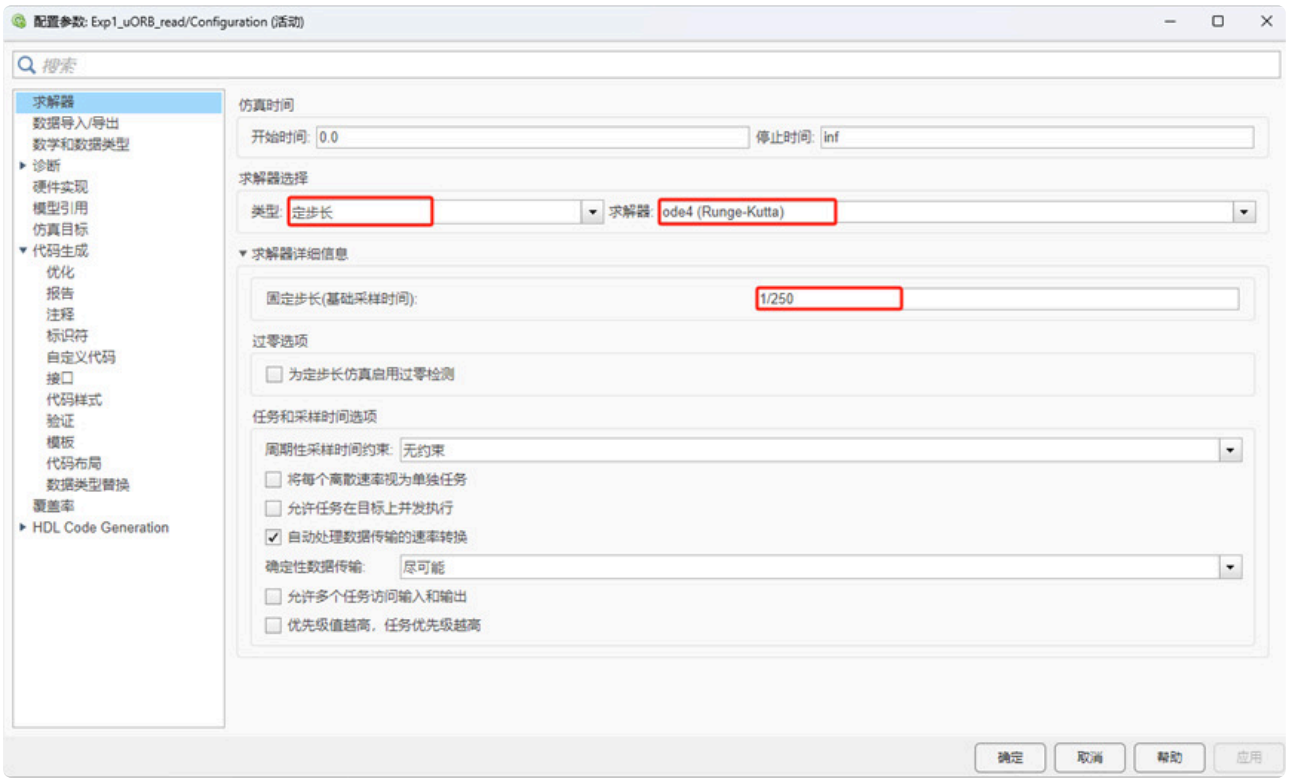
```
uint64 timestamp       # time since system start (microseconds)
```

这是一个包含8个元素的浮点数数组，用于表示8维控制信号。

```
float32[8] control      # 8D control signals
```



在进行自动代码生成前，需要配置仿真环境。

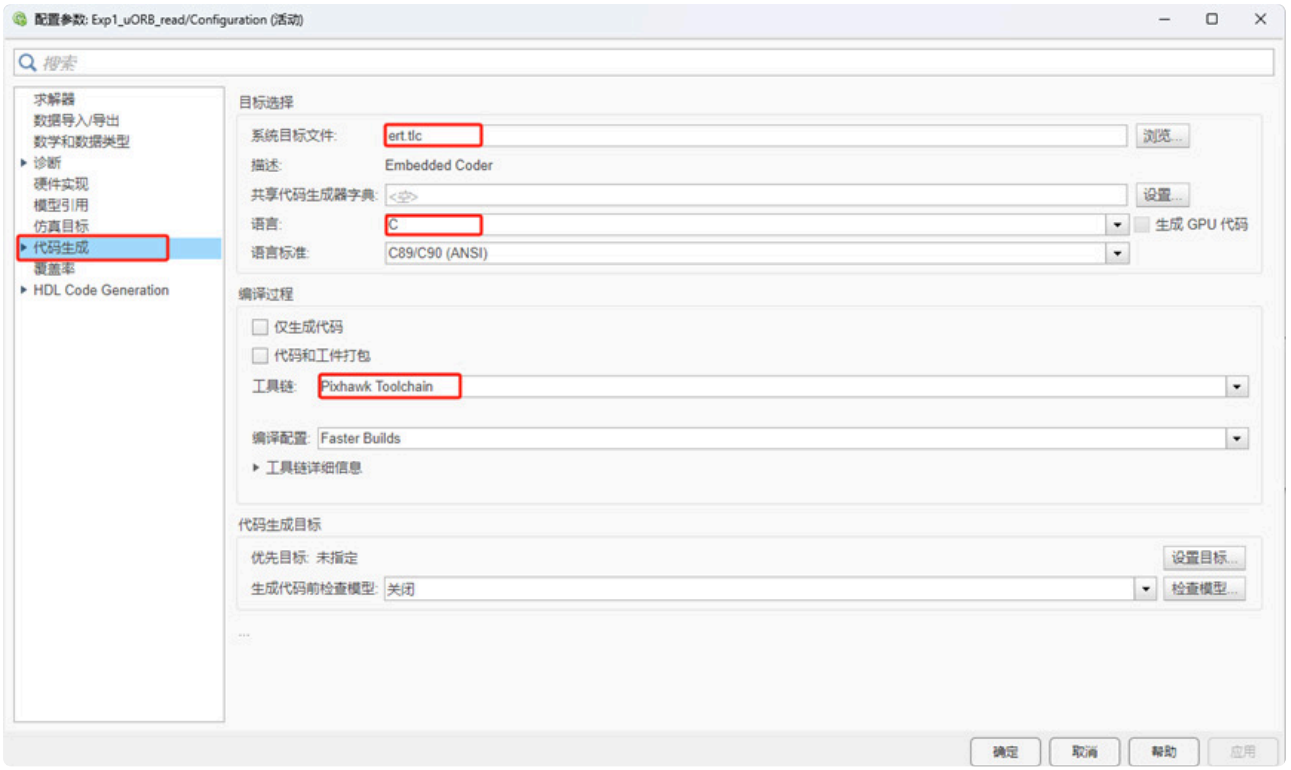


1)必须设置为固定步长。

2)求解器选龙格库塔或其他，阶数越高计算量越大。

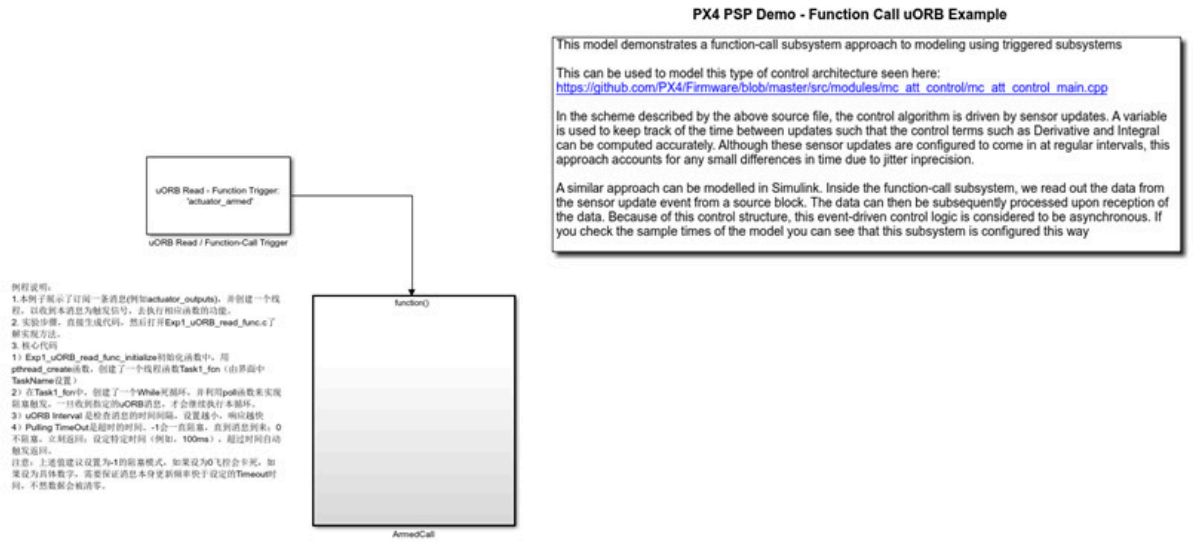
3)仿真步长根据实际情况选择，通常为1/250或1/400，也就是250Hz或400Hz。

4)其他的照页面设置，最重要的确认Toolchain为 Pixhawk Toolchain。



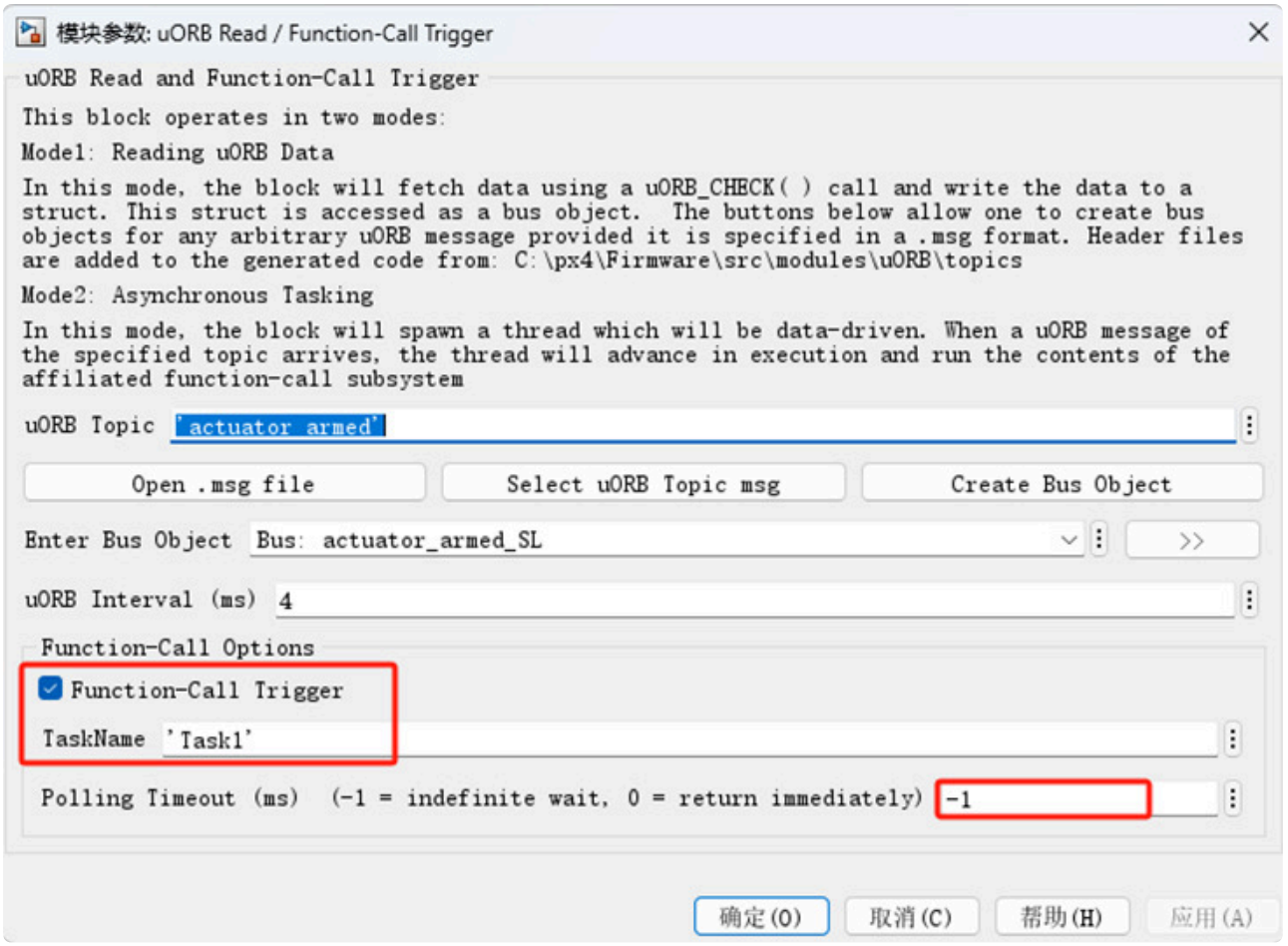
关键知识点2：uORB消息读取触发实验原理

该模型用uORB Read Function-Call Trigger订阅'actuator_armed'消息，来接收是否解锁，如果未解锁，就输出12345678，反之输出00000000，数据通过rfly_px4消息发布出去，被其他程序接收。和“Exp1_uORB_read.slx”实验不同，本例程的数据处理采用了触发方式，即只有actuator_armed更新时，才会去处理数据，并发布rfly_px4消息。



例程实现原理：

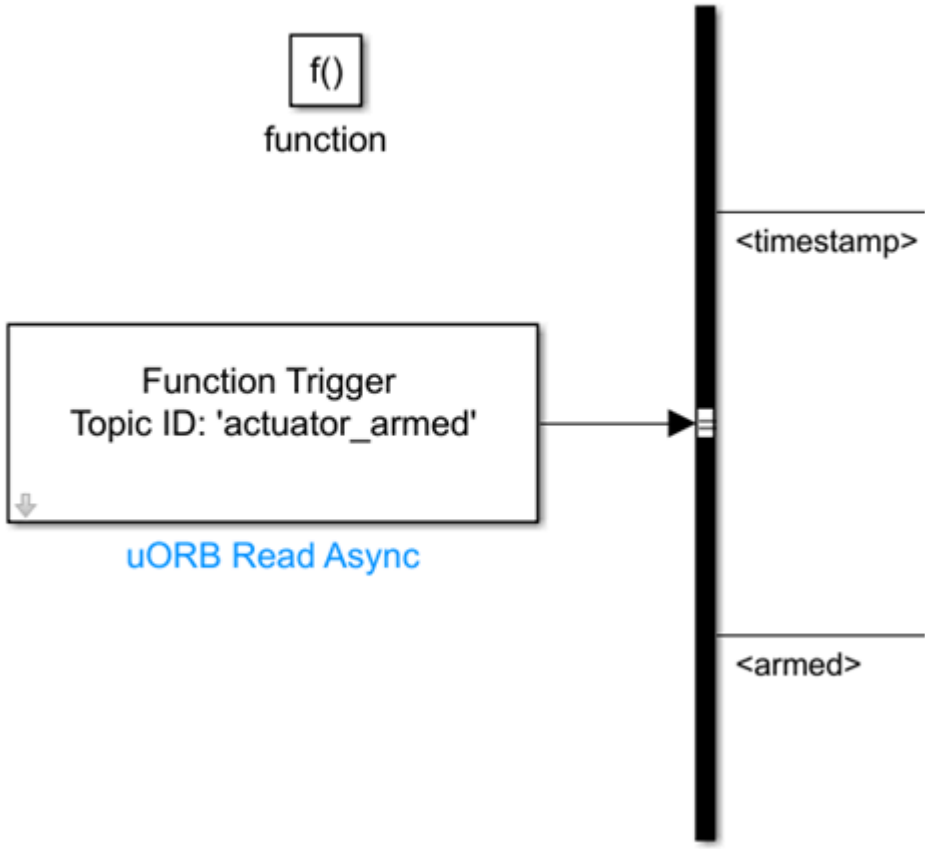
uORB Read Function-Call Trigger的配置页面，要勾选Function-Call Triggle

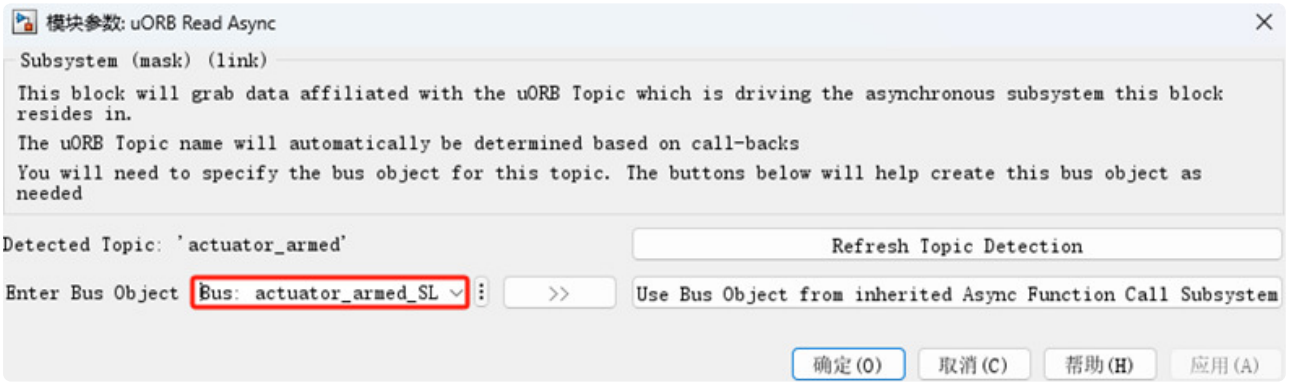


其中Polling Timeout可以设置为-1，或指定值。其中，-1表示会一直阻塞（不会消耗计算性能），直到收到消息，才触发Task1；设置为0，则表示不超时立即返回，即不阻塞；设置为对应毫秒，如果超过指定时间没有收到消息，也会强行触发Task1。注：不能设定为0，会导致飞控以无限快速度执行Task1，导致飞控卡死。

配合function和uORB Read Async模块，来触发并处理actuator_armed数据。

uORB Read Async里面的总线数据要和外层触发模块保持一致，这里是actuator_armed

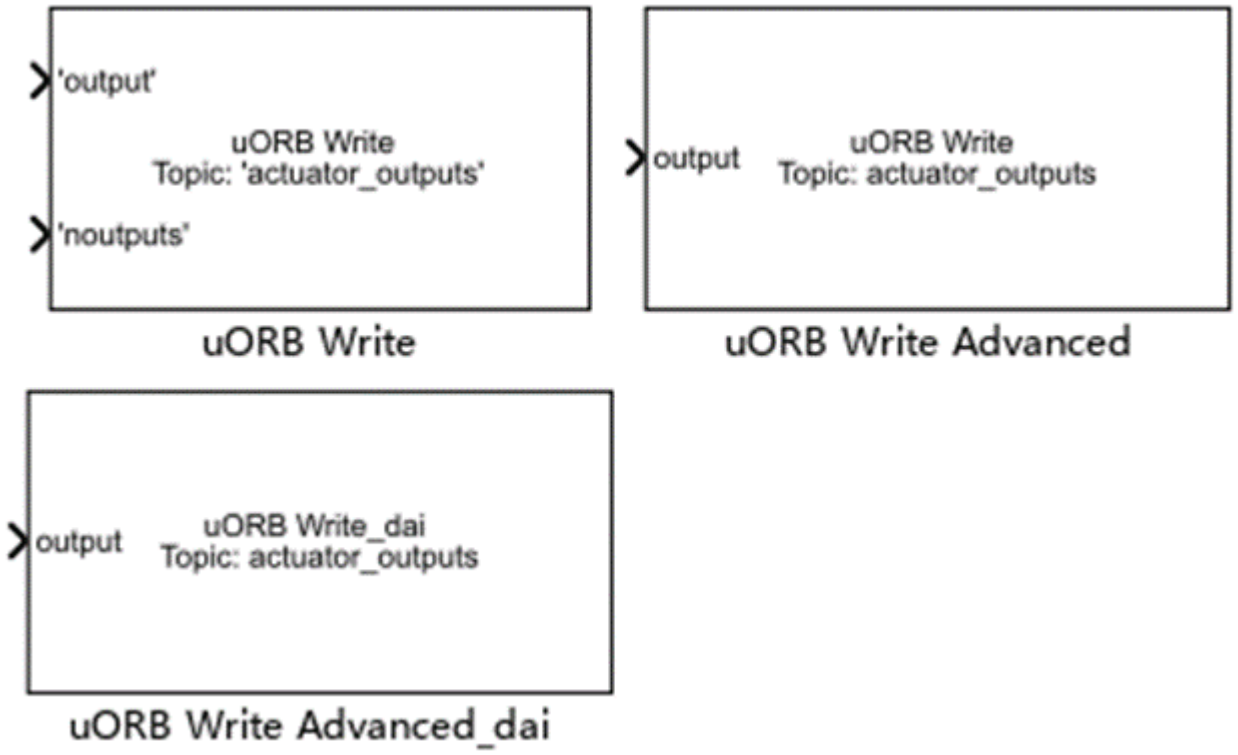




关键知识点3：uORB发送接口使用实验原理

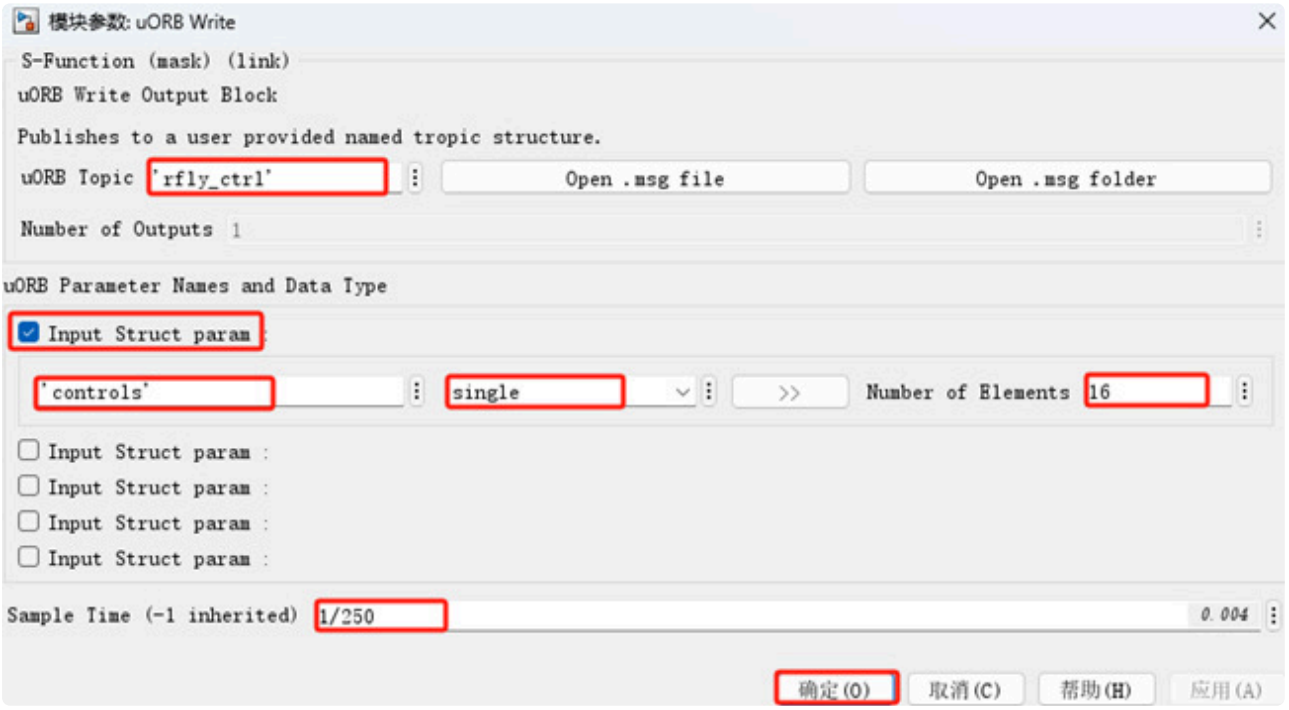
程序内容：监听actuator_armed消息，并通过uORB Write、uORB Write Advanced、uORB Write Advanced_dai三个发送接口，依次发送rfly_ctrl、rfly_px4、rfly_ext三条消息。

在PSP工具库中，三个uORB发送函数如下

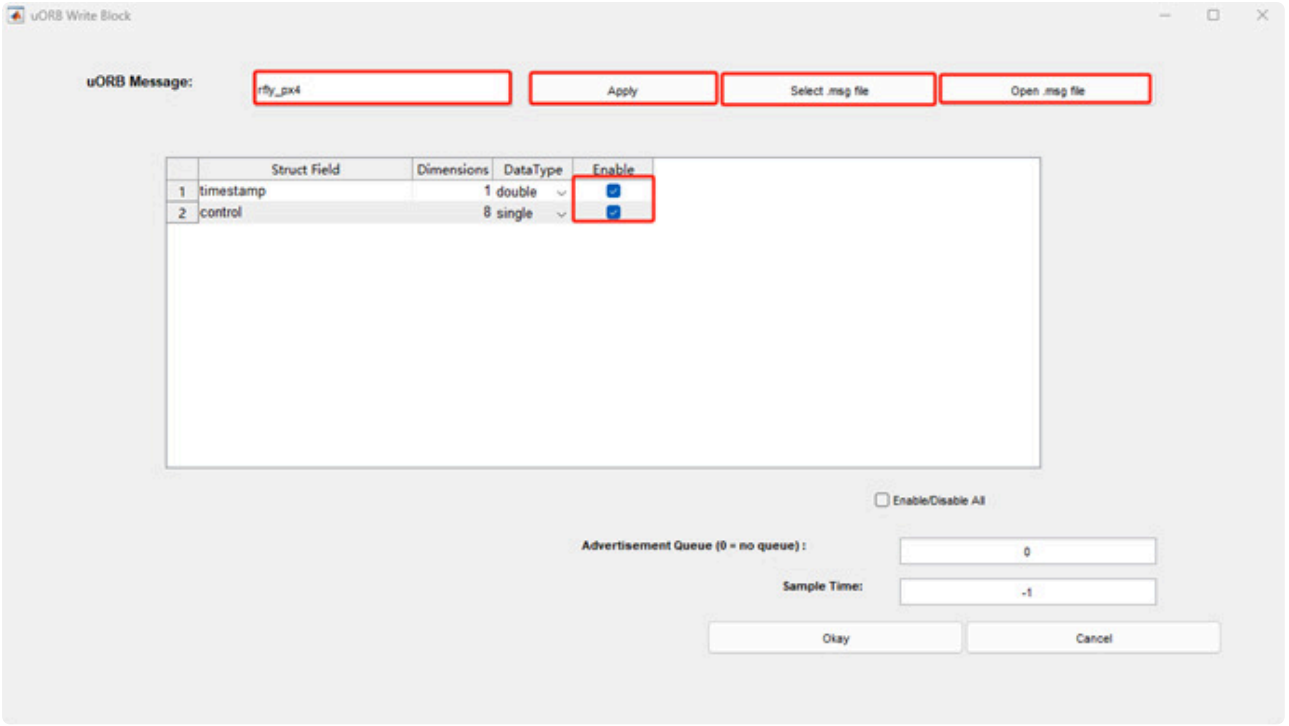


配置方法：

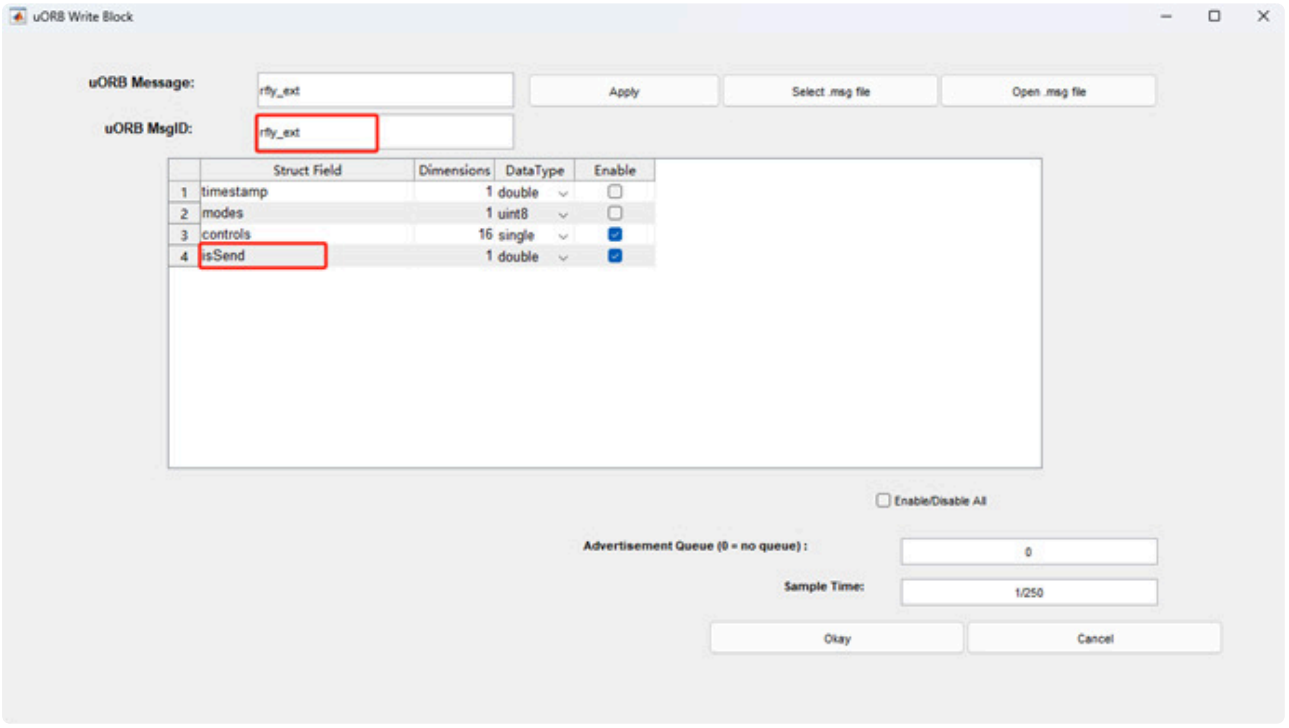
uORB Write 模块需要手动输入消息名，变量名，数据类型，数据维度等，使用较为麻烦



uORB Write Advanced 模块能自动索引uORB消息文件，并可以通过勾选使能想要发送的消息接口，使用较为方便。



uORB Write Advanced_dai 模块支持uORB消息ID和文件名不同，且支持通过isSend接口，控制是否发送消息，适合命令类消息（只需发送一次，而不是持续发送），或者控制发送频率等。



Send1Hz模块，通过时间整除运算，得到1秒钟触发一次的信号，发送给isSend,使得rfly_ext消息的频率被限制在1Hz

Exp3_uORB_write ▸ Send1Hz

```
function y = fcn(t)

% y=1 for 1Hz
tsec = ceil(t*1000);
r = rem(tsec,1000);

y = 0;
if r<0.5
    y = 1;
end
```

关键知识点4：自定义uORB消息实验

打开MATLAB软件，在MATLAB中运行init_control.m文件，即可将自定义的uORB消息加载到飞控固件中。

init_control.m脚本解析：

添加路径 '.\msg'，这个路径应该包含了消息定义的文件。

```
addpath('.\msg')
```

用来生成消息结构体,是一个自定义的脚本，用于从消息定义文件中生成相应的代码。

```
PX4uORBMsgGen
```

msg文件夹中的PX4uORBMsgGen.m脚本的作用是自动化地处理消息文件，并将其集成到PX4 固件中，以便固件能够正确地处理这些消息。

以下是对PX4uORBMsgGen.m脚本的部分解析：

检查固件版本，如果这个变量被设置为false，那么意味着固件版本不低于1.8。

```
isFirmwareLessThan1_8=false;
```

检查是否存在两个.mat文件：CmakeInfo.mat 和 FirmwareVersion.mat。如果这两个文件都存在，它们将被加载到 MATLAB 的工作区中。

```
if exist([PSPPath,'\CmakeInfo.mat'],'file')
        load([PSPPath,'\CmakeInfo.mat']);
        load([PSPPath,'\FirmwareVersion.mat']);
```

检查是否在加载的 Px4PSP_CmakeInfo 结构体变量和 FirmwareVersion 变量中找到了相应的字段。如果找到了，它们将会被提取出来，并且根据 FirmwareVersion 的值来判断固件版本是否小于2.5。如果固件版本小于等于1.8，则会将 isFirmwareLessThan1_8 设置为 true。

```
 if exist('Px4PSP_CmakeInfo','var')
        Px4_Base_Dir=Px4PSP_CmakeInfo.Px4_Base_Dir;
        disp('Found PX4PSP Path.')
 end
 if exist('FirmwareVersion','var')
        if str2num(FirmwareVersion)<2.5 % the Firmware verison <= 1.8
                sFirmwareLessThan1_8=true;
         end
```

输出一些提示信息，指示是否找到了相应的路径和固件版本信息。

```
                disp('Found PX4 Firmware version.')
        end
end
```

然后是一个循环，遍历指定路径下的所有消息文件。在循环中，首先读取消息文件内容，并根据固件版本进行一些处理。例如，如果固件版本较低，则可能会删除或修改一些与新固件版本不兼容的内容。

```
for ii=1:length(AFileList)
          nuttx_str = [AFileList(ii).folder,'\',AFileList(ii).name];
          fid = fopen(nuttx_str);
          strTmp=[];
          while ~feof(fid)
                    str = string(fgetl(fid));
                    if isFirmwareLessThan1_8 && contains(str,'uint64 timestamp')
                              continue;
                    end
                    strTmp=[strTmp;str];
          end
          fclose(fid);

          fiName = AFileList(ii).name(1:end-4);
          if str2num(FirmwareVersion)>7.5
                    if contains(fiName,'_')
                              msg_list = strsplit(fiName,'_');
                              UpStr='';
                              for i=1:length(msg_list)
                                        crStr=msg_list{1,i};
                                        crStr=[upper(crStr(1)),crStr(2:end)];
                                        UpStr=[UpStr,crStr];
                              end
                              fiName=UpStr;
                    end
          else
                    if ~contains(fiName,'_')
                              MsgFileName = regexprep(fiName,'(.)([A-Z][a-
z]+)','$1_$2');
                              MsgFileName = regexprep(MsgFileName,'([a-z0-9])([A-
Z])','$1_$2');
                              fiName=lower(MsgFileName);
                    end
          end
```

将处理后的消息内容写入到 PX4 固件的消息目录中，并更新 CMakeLists.txt 文件以确保消息被正确包含在固件编译中。

```
nuttx_str = [Px4_Base_Dir,'\Firmware\msg\CMakeLists.txt'];
          fid = fopen(nuttx_str);
          strTmp=[];
          isMsgExist=false;
          idx=50;
          i=1;
          while ~feof(fid)
                    str = string(fgetl(fid));
                    if contains(str,fiName)
                              isMsgExist=true;
                              break;
                    end
                    if str2num(FirmwareVersion)>7.5
                              if contains(str,'RcParameterMap.msg')
                                        idx=i;
                              end
                    else
                              if contains(str,'actuator_controls.msg')
                                        idx=i;
                              end
                    end

                    strTmp=[strTmp;str];
                    i=i+1;
          end
          fclose(fid);

          if ~isMsgExist
                    strTmp=[strTmp(1:idx);"  "+string(fiName);strTmp(idx+1:end)];
                    fid = fopen(nuttx_str,'w+');
                    for i=1:length(strTmp)
                              fprintf(fid, '%s\n', strTmp(i,:));
                    end
                    fclose(fid);
          end
          addLoggerList=[addLoggerList;string(msgName)];
end
```

根据固件版本修改日志文件（如 logger.cpp）以确保新的消息能够被正确记录。

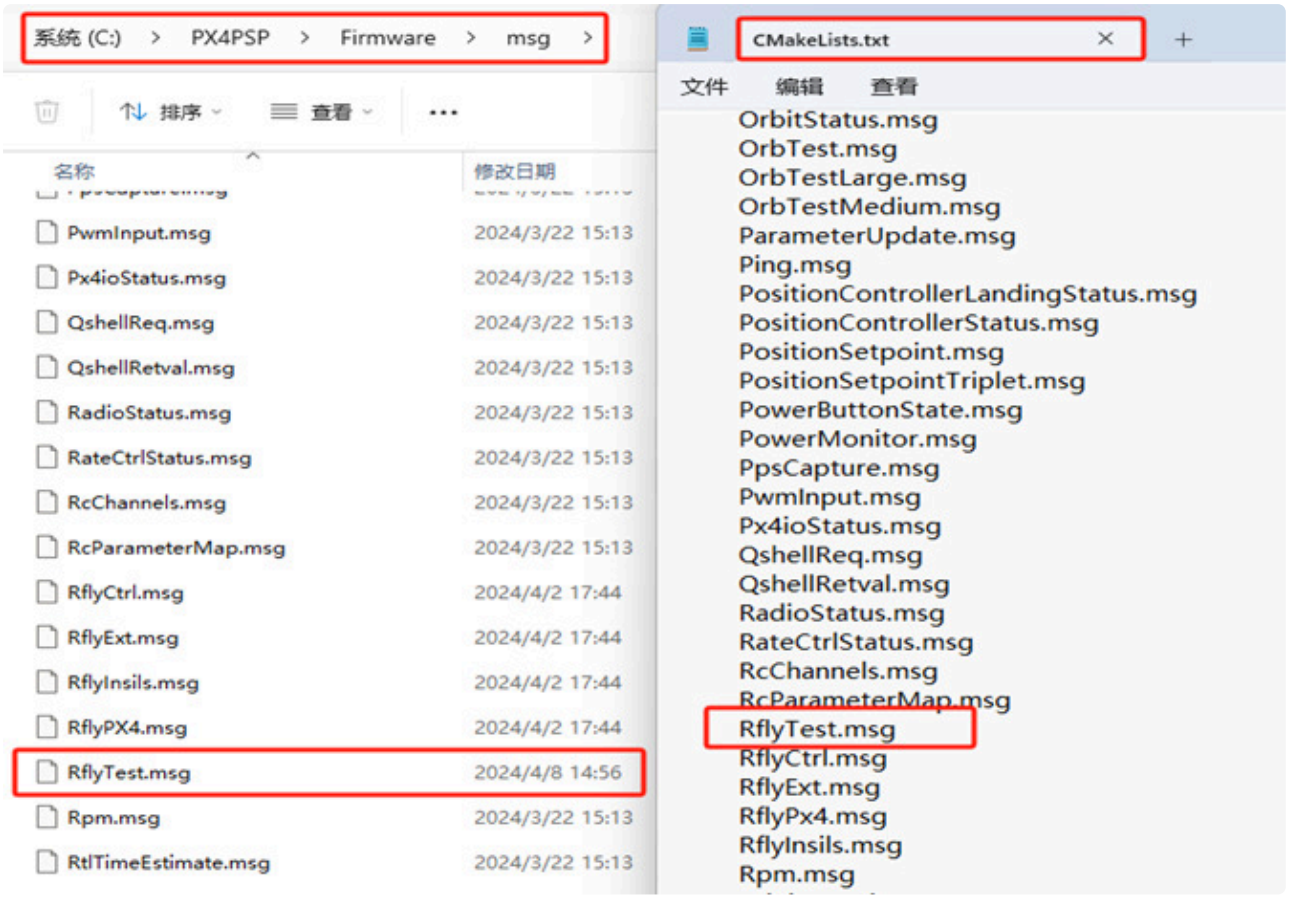
```
LogName = 'logger.cpp';
if str2double(FirmwareVersion) > 4.5
    LogName = 'logged_topics.cpp';
end
nuttx_str = [Px4_Base_Dir, '\Firmware\src\modules\logger\', LogName];

for i = 1:length(addLoggerList)
    logMsgName = ['add_topic("', char(addLoggerList(i)), '", 20);'];

    fid = fopen(nuttx_str);
    strTmp = [];
    j = 0;
    add_default_topics_idx = 0;
    msg_idx = 0;

    while ~feof(fid)
        j = j + 1;
        str = string(fgetl(fid));

        if add_default_topics_idx < 1 && contains(str, '::add_default_topics()')
            add_default_topics_idx = j;
        end

        if msg_idx < 1 && contains(str, logMsgName)
            msg_idx = j;
        end

        strTmp = [strTmp; str];
    end
    fclose(fid);

    if msg_idx < 1 && add_default_topics_idx > 1
        strTmp = [strTmp(1:add_default_topics_idx + 1); "    " + logMsgName;
strTmp(add_default_topics_idx + 2:end)];

        fid = fopen(nuttx_str, 'w+');
        for k = 1:length(strTmp)
            fprintf(fid, '%s\n', strTmp(k, :));
        end
        fclose(fid);
    end
end
```

输出PX4 Firmware uORB modified.表示修改完成

```
disp('PX4 Firmware uORB modified.')
```

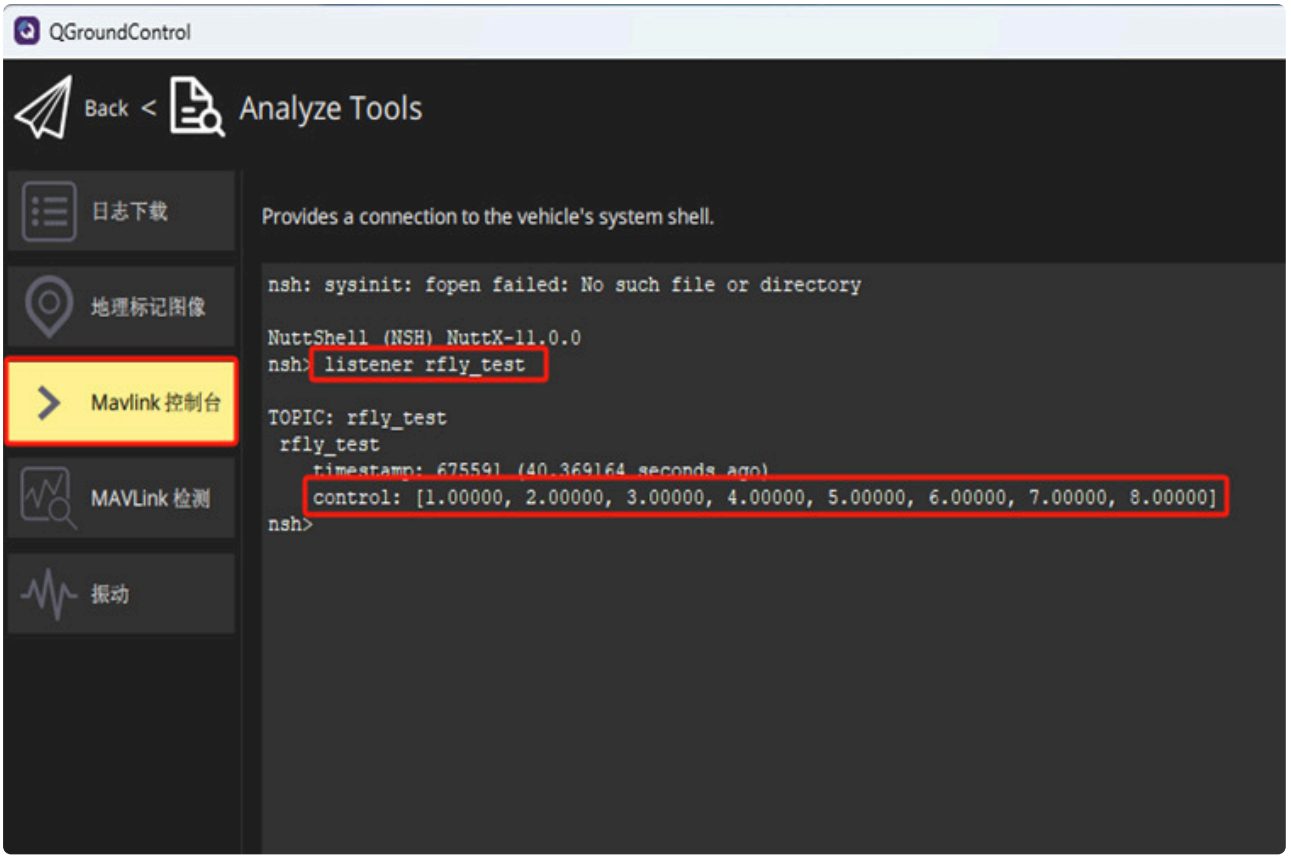
可在“\PX4PSP\Firmware\msg\rfly_test.msg”中查看到rfly_test.msg文件（注意：1.14版本开始是RflyTest.msg命名方式）。同时在本文件夹的CMakeList.txt中也可看到新增的

rfly_test.msg消息。如下图所示。



在进行自动代码生产和固件烧录后，打开QGroundControl软件，点击左上角Logo在弹出的对话框中，选中Analyze Tools，在Mavlink控制台中输入：

```
listener rfly_test
```



可以发现，和本例程我们自定义的uORB消息的输入是一致的。

6.参考资料

1. RflySim官方文档
2. PX4官方文档-uORB
3. PX4源码仓库
4. MathWorks. "Develop Algorithms and Deploy on PX4 Autopilots". MATLAB & Simulink.

7.常见问题

Q1：编译时报错提示uORB数量超过限制?

A1：PX4对uORB消息数量有限制，当添加过多消息时可能会出现类似"enumerator value 256 is outside the range of underlying type"的错误。解决方法包括：检查是否真的需要添加新的uORB消息，考虑复用现有消息；如果确实需要大量消息，可以尝试修改相关代码以支持更多消息类型。

Q2：在QGroundControl中无法看到自定义的uORB消息？

A2：这通常是因为消息没有正确发布或日志配置不正确导致的。确保在代码中正确初始化并发布了uORB消息，并且在使用`listener`命令时使用了正确的消息名称。另外，检查飞控的日志配置，确保自定义消息已被添加到日志记录中。

Q3：如何调试uORB消息是否正常工作？

A3：可以使用多种方法进行调试：

1. 使用`uorb status`命令查看uORB状态
2. 使用`listener <message_name>`实时查看消息内容
3. 使用`uorb top`监控消息更新频率
4. 在代码中添加日志输出确认消息的发布和订阅
5. 使用FlightPlot等地面站软件查看记录的消息日志

---

1. https://rflysim.com/ ↩
2. 推荐配置请见： https://rflysim.com/doc/zh/HowToInstall.pdf ↩