1. 实验名称及目的

1.1. 实验名称

MAVLink 模块封装实验

1.2. 实验目的

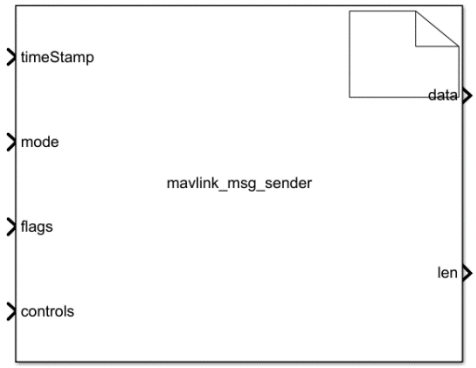
MAVLink （Micro Air Vehicle Link）是一种用于小型无人载具的通信协议，于 2009 年首次发布。该协议广泛应用于地面站（Ground Control Station，GCS）与无人载具（Unmanned vehicles）之间的通信，同时也应用在载具上机载计算机与 Pixhawk 之间的内部通信中，协议以消息库的形式定义了参数传输的规则。MAVLink 协议支持无人固定翼飞行器、无人旋翼飞行器、无人车辆等多种载具。本实验将基于 Simulink 对 MAVLINK_MSG_ID_HIL_ACTUATOR_CONTROLS 消息进行数据发送模块和数据解析模块两部分。

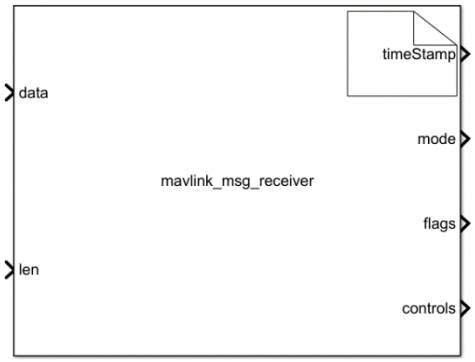
1.3. 关键知识点

无

2. 实验效果

封装完成下图所示的两个模块用于后续实验。





3. 文件目录

例程目录：[安装目录]\RflySimAPIs\6.RflySimExtCtrl\1.BasicExps\e0_ExtAPIUsage\1.MavLinkPackSimulink\

文件夹/文件名称	说明
mavlink	MAVLink 的 C++源代码。
MAVLinkPacking.slx	MAVLink 消息模块封装文件。

4. 运行环境

序号	软件要求	硬件要求

		名称	数量
1	Windows 10 及以上版本	笔记本/台式电脑[①]	1
2	RflySim 工具链		
3	MATLAB 2022b 及以上版本		

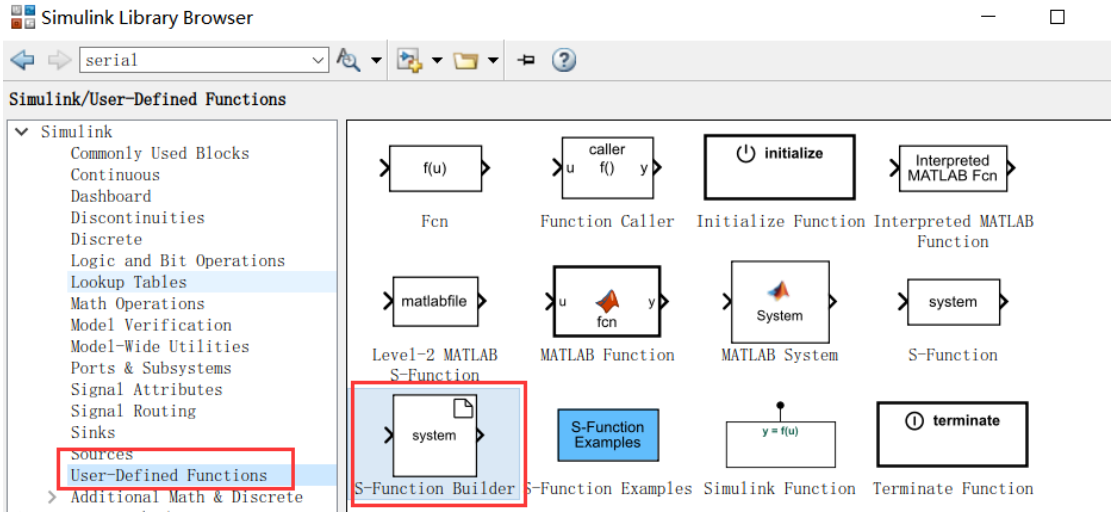
：推荐配置请见：https://rflysim.com/doc/zh/HowToInstall.pdf

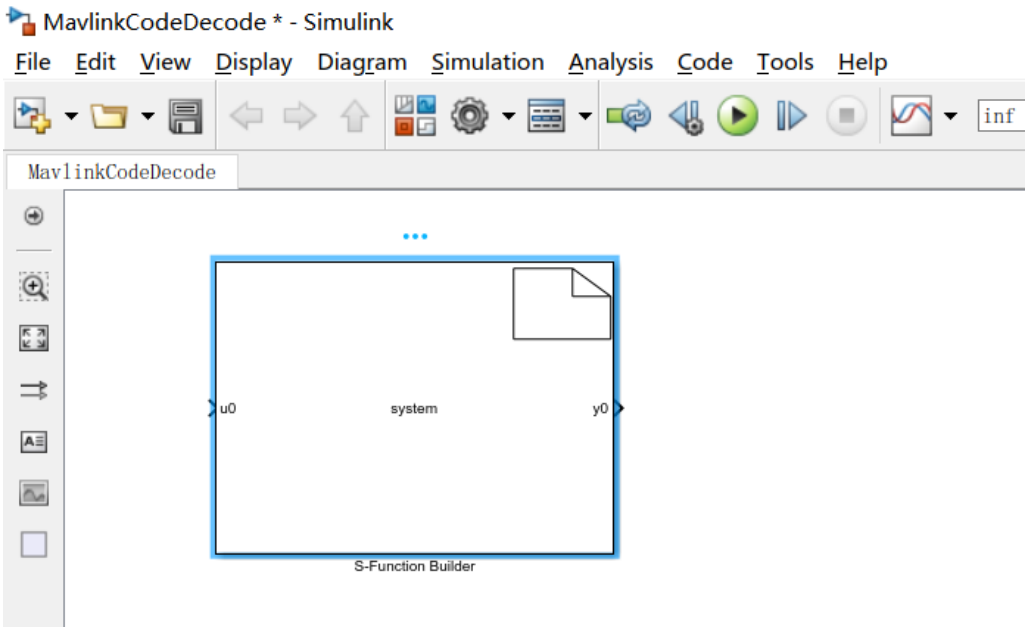
5. 实验步骤

5.1.必做实验：封装 MAVLink 发送端模块

Step 1：空白 Simulink 文件中添加 S-Function Builder 模块

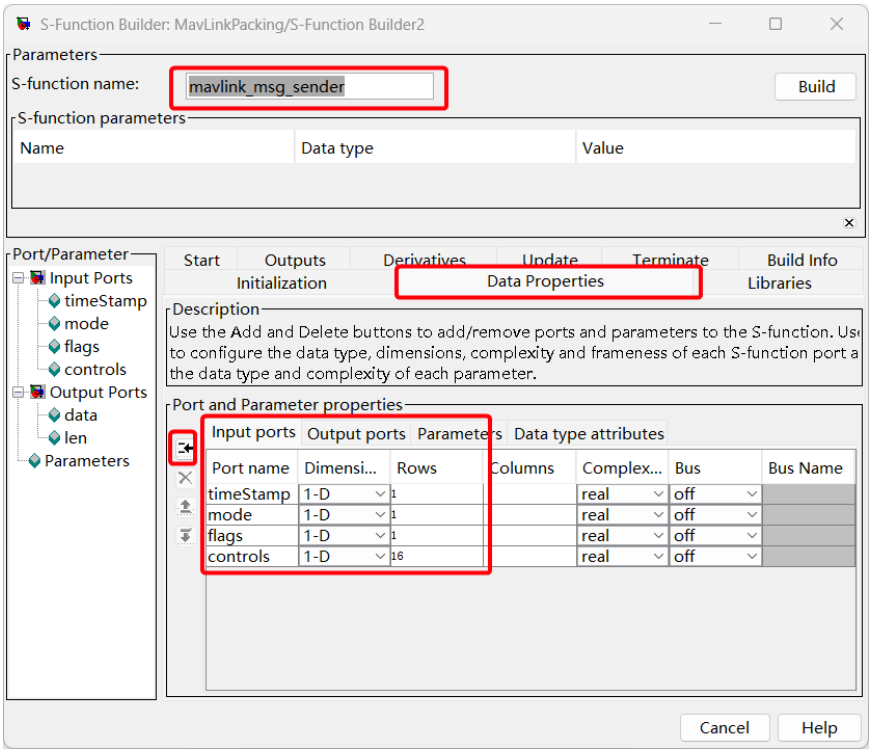
打开 MATLAB 的 Simulink 软件（本次演示使用的为 MATLAB 2017b），新建文件名为 MavLinkPacking（注：可自定义命名），在 Simulink Library Browser 中选择 Simulink-User-Defined Functions，从其中拖一个 S-Function Builder 模块到新建的文件中，如下图所示。



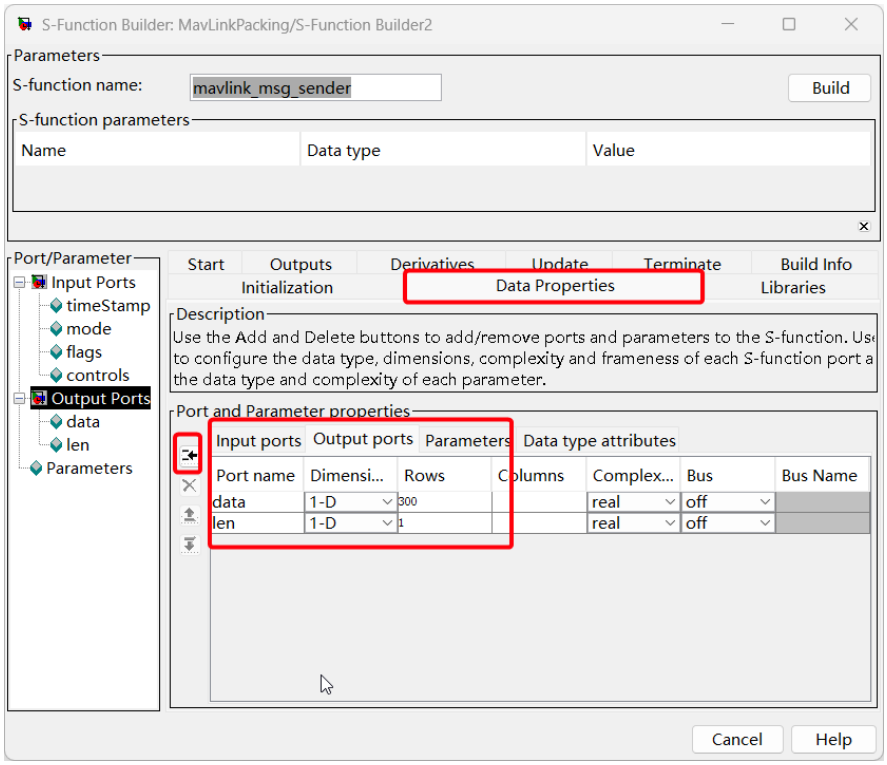


Step 2：S-Function Builder 模块输入输出端口数据名称

双击打开 S-Function Builder 模块，设置模块的名称为：mavlink_msg_sender（注：可自定义命名），选择 Data Properties->Input ports 命令框，在其中新建如下图所示的数据名称。

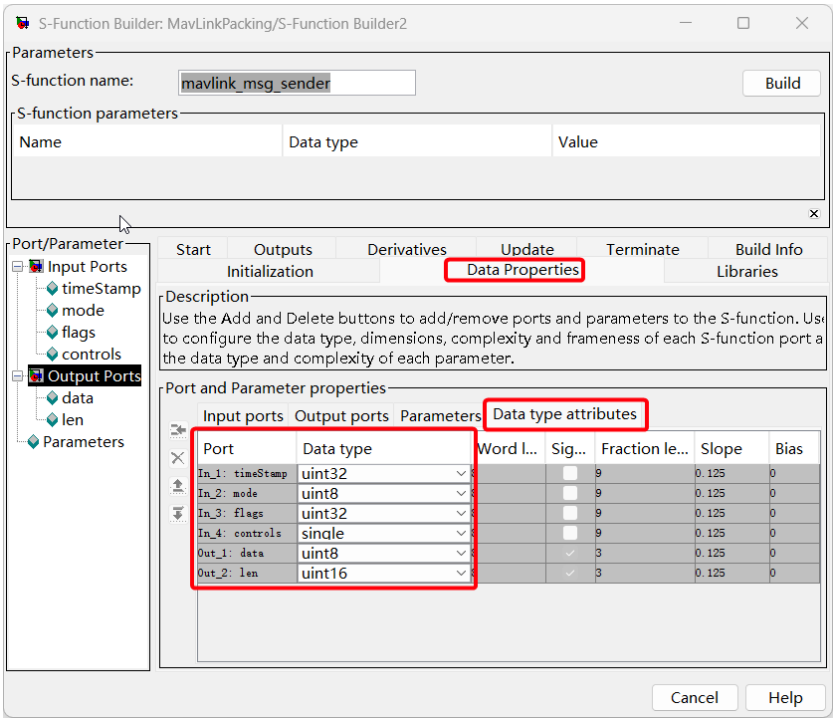


同理，选择 Data Properties->Output ports 命令框，在其中新建如下图所示的数据名称。



Step 3：S-Function Builder 模块输入输出端口数据类型

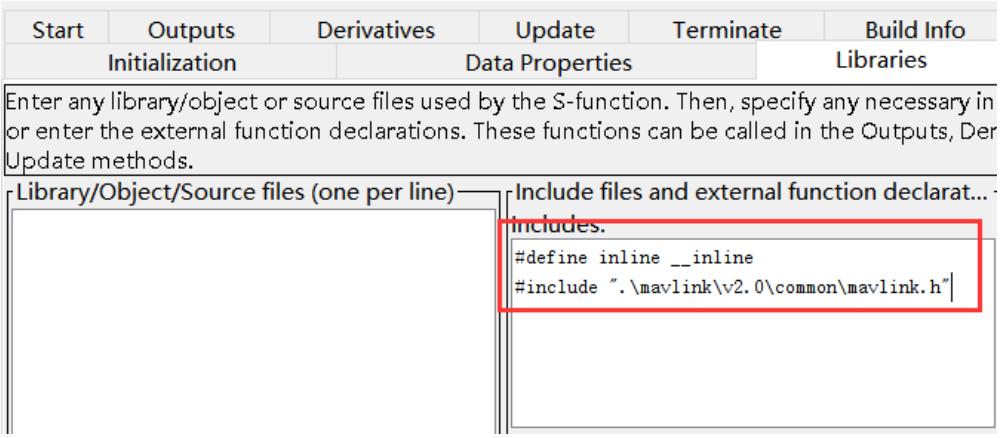
选择 Data Properties->Data Type attributes 命令框，在其中修改上一步所创建的数据，将 Data type 设置为如下图所示，其余保持默认。



Step 4：引入 Mavlink 头文件库

引入 Mavlink 头文件库，在 Libraries->includes 框内加入下列代码

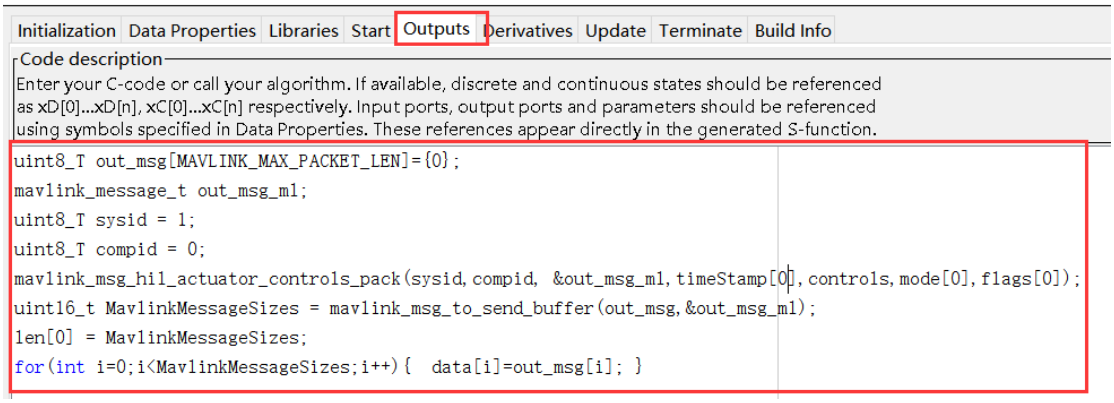
```
#define inline __inline
#include ".\mavlink\v2.0\common\mavlink.h"
```



Step 5: 编写 C/C++代码以获取输入并打包输出

在 Outputs 标签页中，添加获取输入数据，并打包成 Mavlink 消息的 C/C++代码，放到输出口 data 和 len 中。其中 data 是 uint8 的矩阵，len 是数据有效长度。

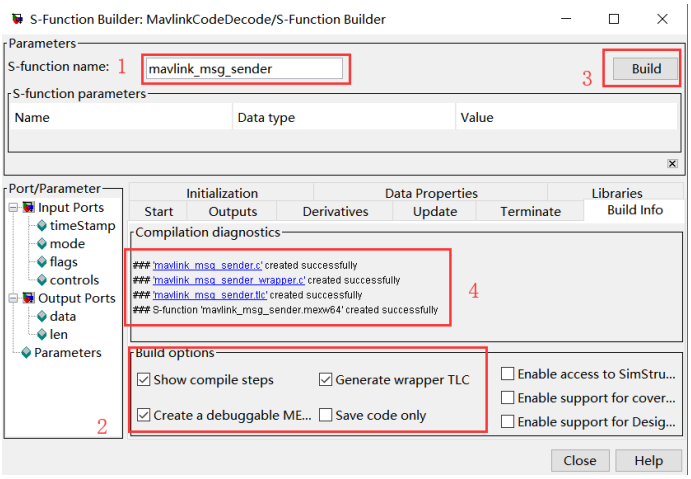
```
uint8_T out_msg[MAVLINK_MAX_PACKET_LEN]={0};
mavlink_message_t out_msg_ml;
uint8_T sysid=1;
uint8_T compid=0;
mavlink_msg_hil_actuator_controls_pack(sysid,compid,&out_msg_ml,timeStamp[0],controls,mod
e[0],flags[0]);
uint16_t MavlinkMessageSizes=mavlink_msg_to_send_buffer(out_msg,&out_msg_ml);
len[0]=MavlinkMessageSizes;
for(int i=0;i<MavlinkMessageSizes;i++){data[i]=out_msg[i];}
```



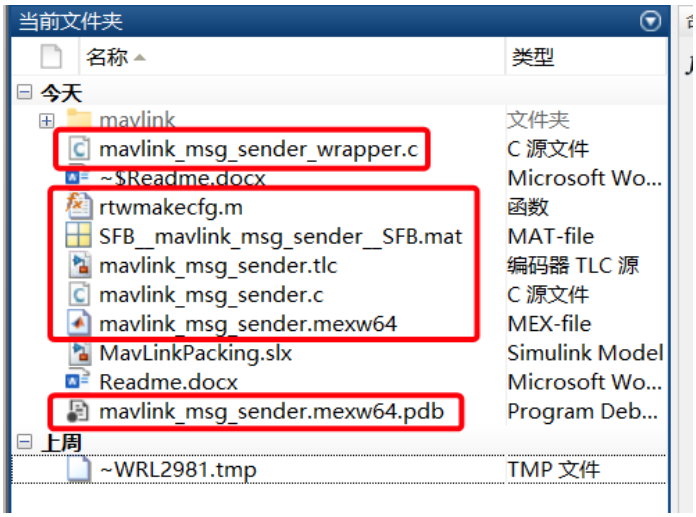
注意：Simulink S-function 的输入输出信号没有标量的概念，所有信号都是向量。因此假设一个输出 len 是一维标量，像“len=***”这种赋值语句是错误的，要用“len[0]=***”的形式。

Step 6: 编译生成 TLC 和 MEX-file

进入 Build Info 标签页，勾选生成 TLC 和 MEX-file 的选项，再点击编译按钮，就可以得到下图所示 Simulink 可调用的文件。



生成的各种文件见下图。

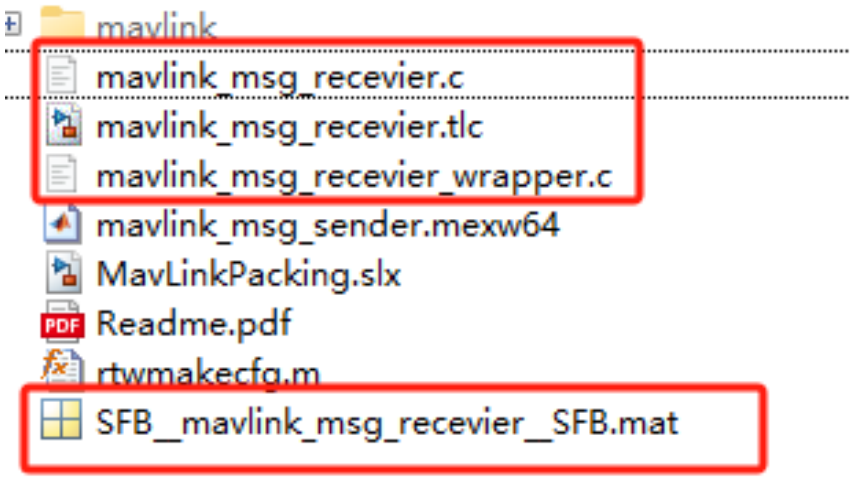


至此，MAVLink 发送端模块封装完成。点击 Close 选择保存即可。

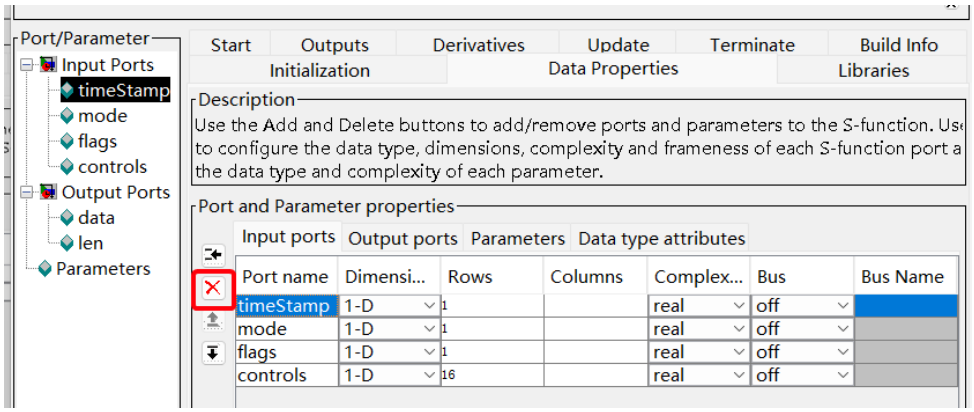
5.2.必做实验：封装 MAVLink 接收端模块

Step 1：新建模块并添加输入输出端口

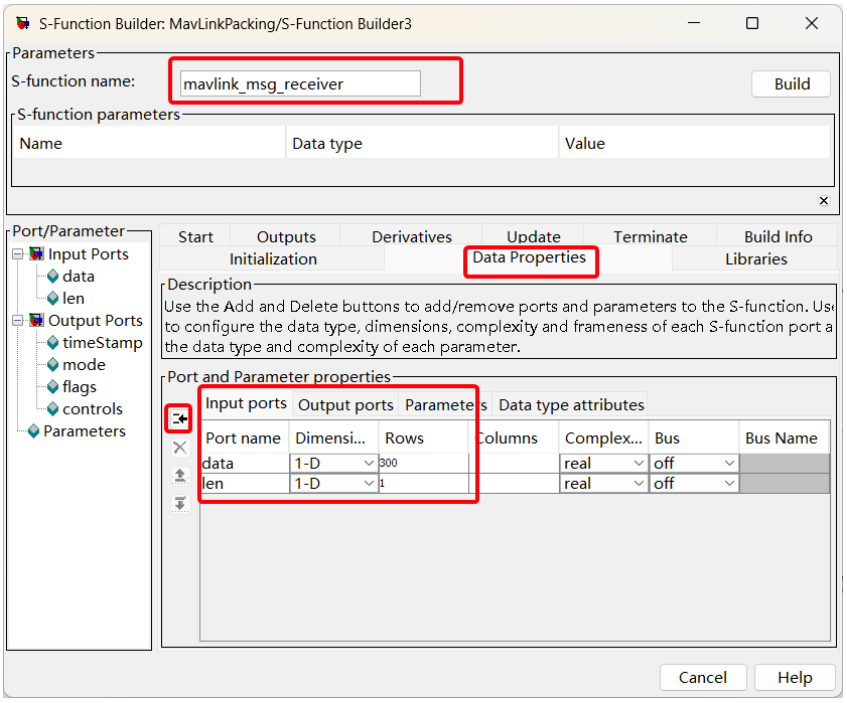
接下来为封装接收(解码)端模块，在进行封装之前，我们需要将前一个封装模块生成的文件进行删除。



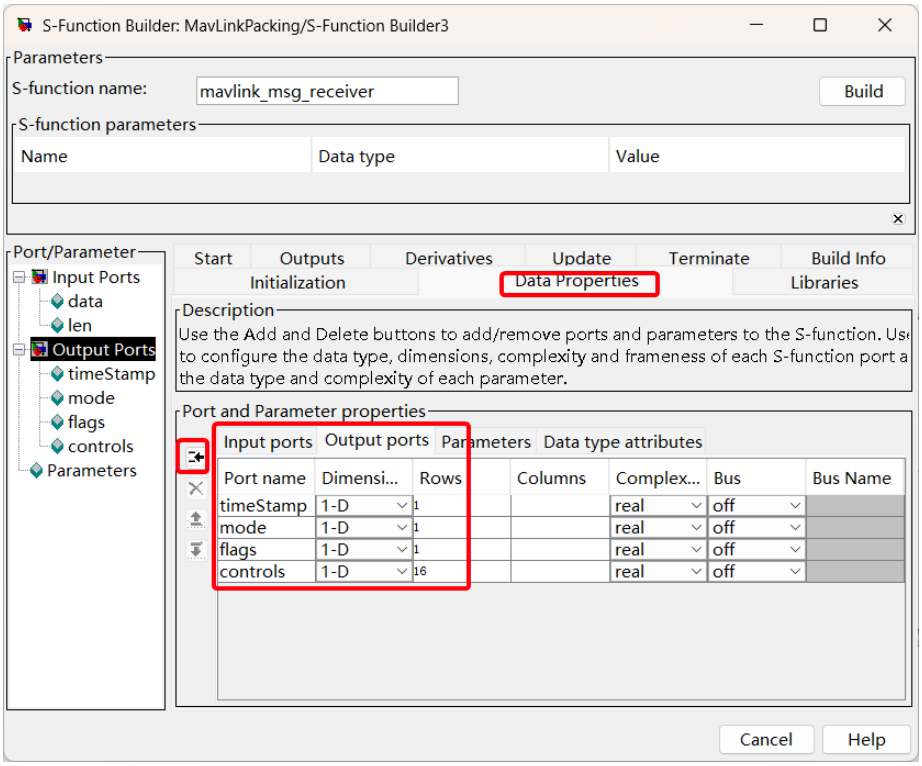
为了方便封装，可复制上述步骤所封装的 mavlink_msg_sender 模块，双击打开复制的 mavlink_msg_sender 模块，将模块的名称修改为：mavlink_msg_receiver（注：可自定义命名），点击左侧的红色×号，删除所有 Inports 和 Output ports 中所有的数据，



选择 Data Properties->Input ports 命令框，在其中新建如下图所示的数据名称。

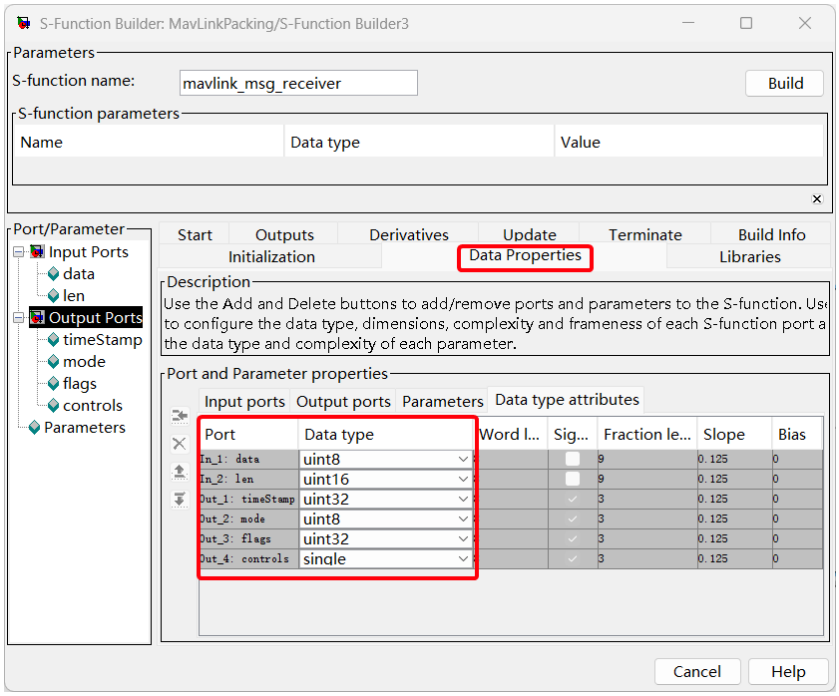


同理，选择 Data Properties->Output ports 命令框，在其中新建如下图所示的数据名称。



Step 2：输入输出端口数据类型

选择 Data Properties->Data Type attributes 命令框，在其中修改上一步所创建的数据，将 Data type 设置为如下图所示，其余保持默认。

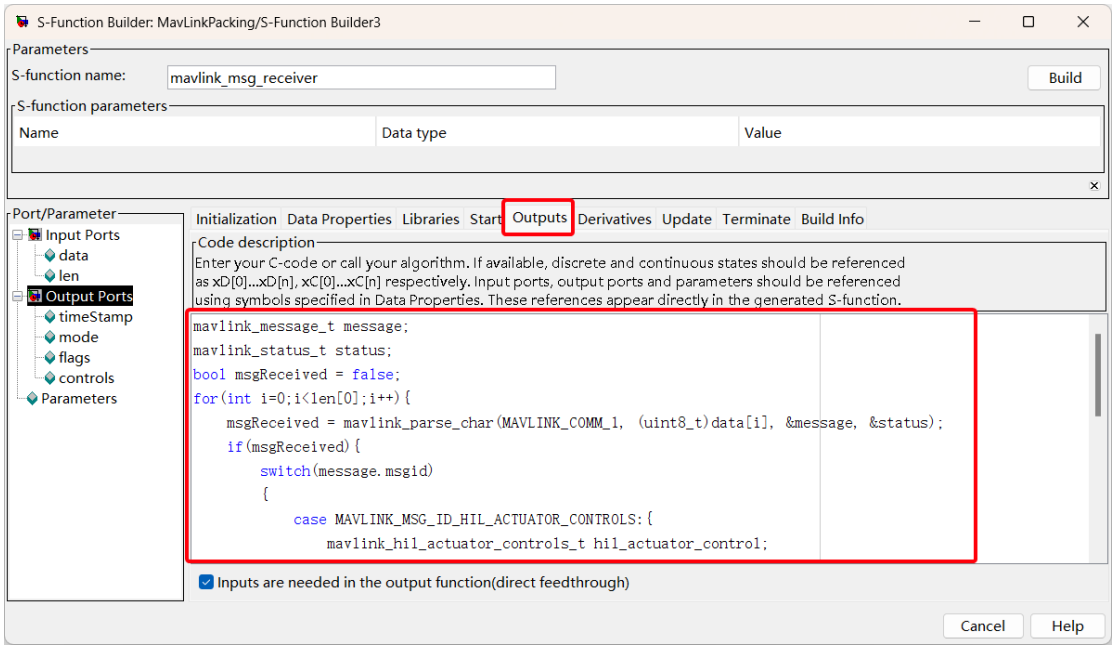


Step 3: 编写 C/C++代码以获取输入并打包输出

在 Outputs 标签页中，设置解码字节流并解析出 Mavlink 消息的代码。同样的道理，在库文件页面，导入 Mavlink 库文件。库文件页面配置与 sender 模块相同。

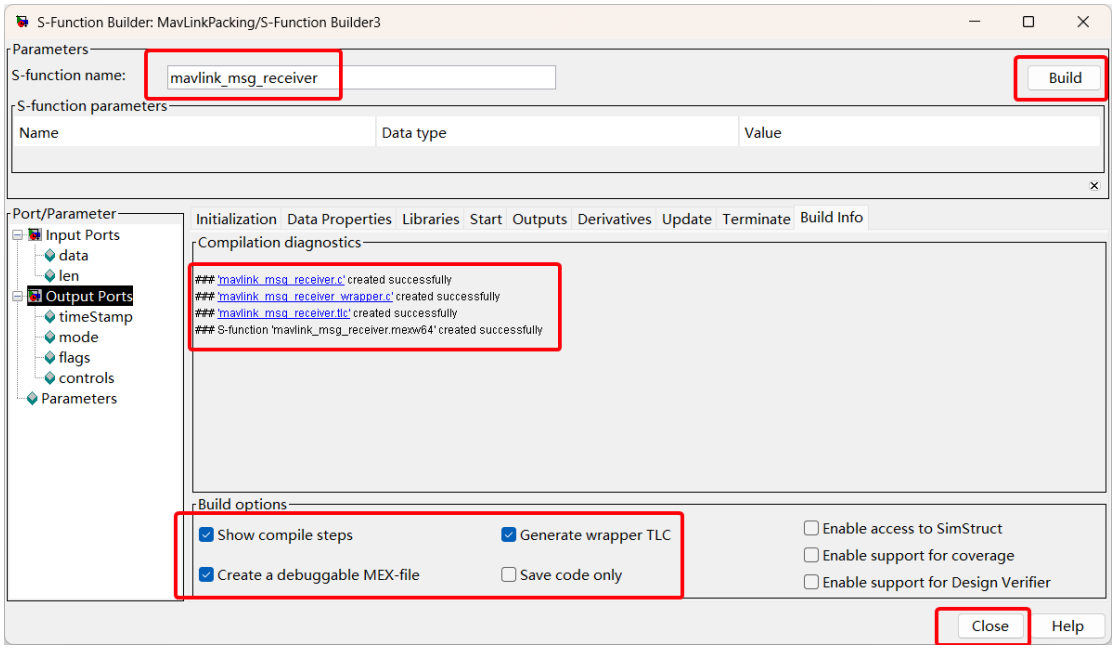
```
    mavlink_message_t message;
    mavlink_status_t status;
    bool msgReceived = false;
    for(int i=0;i<len[0];i++){
        msgReceived = mavlink_parse_char(MAVLINK_COMM_1, (uint8_t)data[i], &message, &statu
s);
        if(msgReceived){
            switch(message.msgid)
            {
                case MAVLINK_MSG_ID_HIL_ACTUATOR_CONTROLS:{
                    mavlink_hil_actuator_controls_t hil_actuator_control;
                    mavlink_msg_hil_actuator_controls_decode(&message, &hil_actuator_contro
l);
                    timeStamp[0] = hil_actuator_control.time_usec;
                    mode[0] = hil_actuator_control.mode;
                    flags[0] = hil_actuator_control.flags;
                    for(int i=0;i<16;i++){
                        controls[i]=hil_actuator_control.controls[i];
                    }
                    break;
                }
                default:{
                    break;
                }
            }
            msgReceived = false;
```

```
        }
    }
```

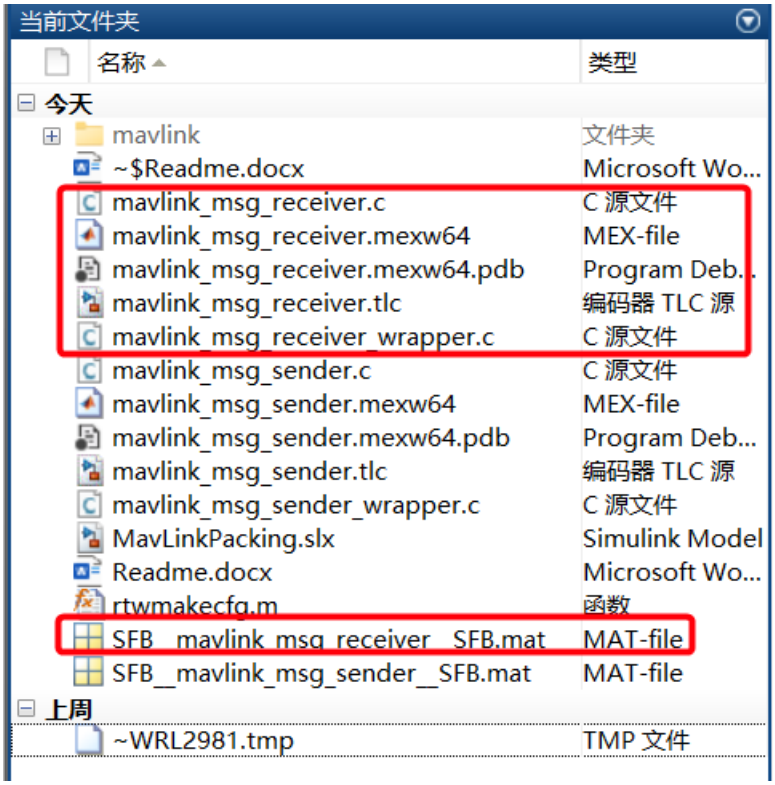


Step 4：编译生成 TLC 和 MEX-file

进入 Build Info 标签页，勾选生成 TLC 和 MEX-file 的选项，再点击编译按钮，就可以得到下图所示 Simulink 可调用的文件。



生成文件如下图所示：



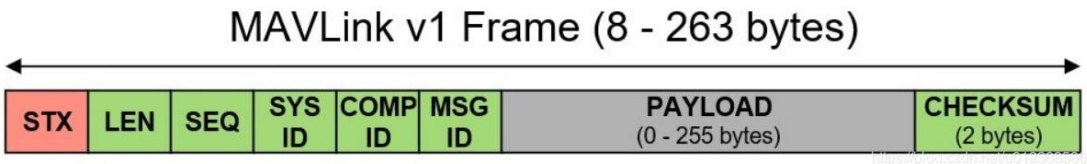
6. 参考资料

[1]. MAVLink 官方使用文档网站：https://mavlink.io/en/messages/common.html

[2]. MAVLink 源码：https://github.com/mavlink/mavlink

[3]. 基于 MAVLink 的 QGroundControl 地面站源码：https://github.com/mavlink/qgroundcontrol

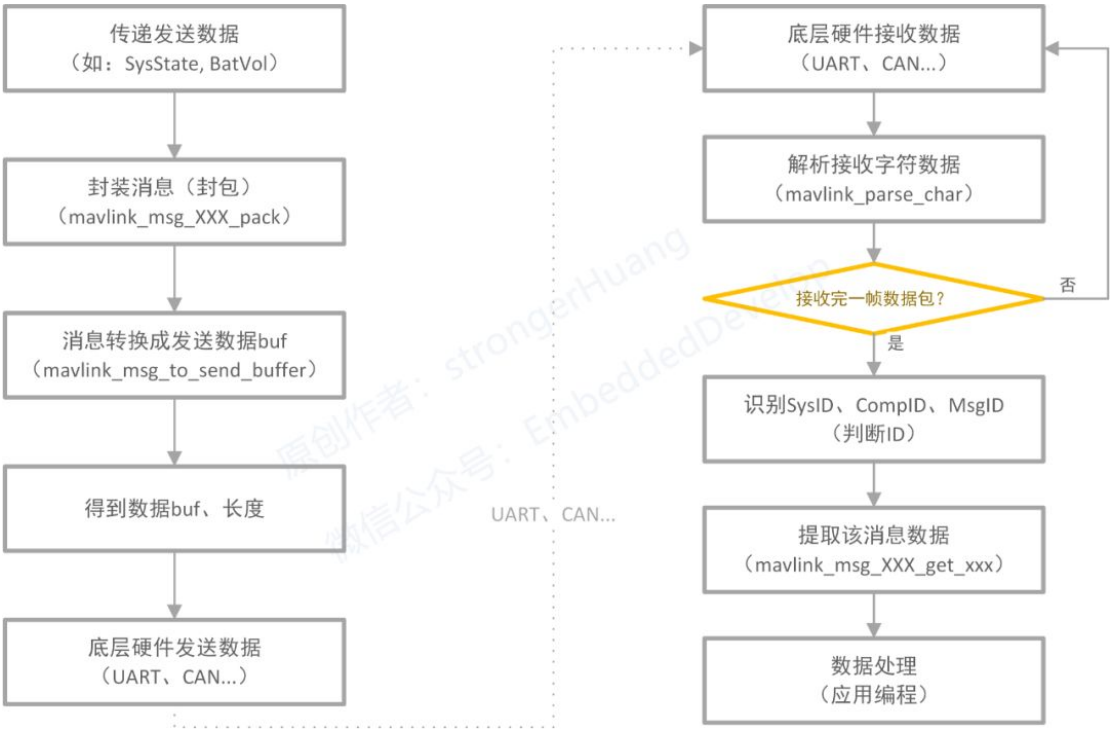
[4]. MAVLink 1 的数据包格式如下：



[5]. MAVLink 2 的数据包如下：

区域	名称	索引	长度	含义	取值
STX	起始标识	0	1	标识新消息的开始，用于消息识别、解析	254
LEN	负载长度	1	1	记录负载信息的长度	N
SEQ	序列码	2	1	消息发送序列码，用于通信可靠性检验	0-255
SYS	系统ID	3	1	发送该消息系统的系统ID	0-255
COMP	组件ID	4	1	发送该消息系统组件的组件ID	0-255
MSG	消息ID	5	1	标识该消息的种类	0-255
PAYLOAD	负载信息	6	1	消息内部负载信息	
CKA	校验位A	N+6	N	CRC校验位	自动
CKB	校验位B	N+7	1	CRC校验位	自动

[6]. 解析原理：**读取：**所有字节流存入 buffer，依次读取 buffer 中的字节数据，遇到 STX 标志位（MAVLink v1 的标志位是 0xFE，v2 的标志位为 0xFD）开始识别一条消息直到消息尾部，如果消息校验正确则将消息发送给处理程序。**发送：**按上一页 PPT 将消息转换成字节流。



[7]. 接收解析源码分析

给定一定长度的字节流 buffer，长度为 length，通过下列脚本解析，每解析出一个 mavlink 数据包就执行 onMavLinkMessage 函数

```
     for(int i = 0 ; i < length ; ++i){
            msgReceived = mavlink_parse_char(MAVLINK_COMM_1, (uint8_t)buffer[i], &message, &status);
            if(msgReceived){
        emit onMavLinkMessage(message);
            }
      }
```

其中：

```
void onMavLinkMessage(mavlink_message_t message);
```

是得到一个 MAVLink 消息包后的处理函数，需要根据这个消息的 ID 来识别当前包的用途（心跳包，GPS 位置，姿态等），并提取出感兴趣的数据。

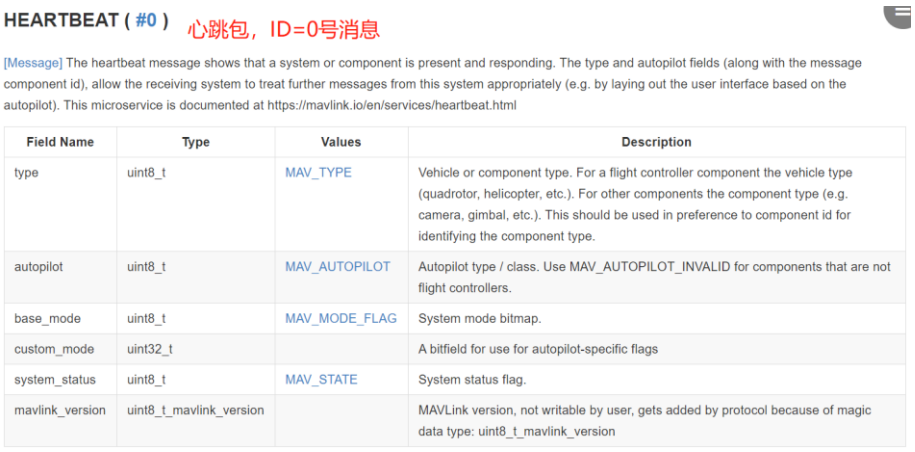
解析函数实现如下，根据 message.msgid 跳到对应的_decode 函数，解码出数据：

```
void onMavLinkMessage(mavlink_message_t message){
    switch (message.msgid){
      case MAVLINK_MSG_ID_GLOBAL_POSITION_INT:{
        mavlink_global_position_int_t gp;
        mavlink_msg_global_position_int_decode(&message, &gp);
        outHilData.time_boot_ms = m_LastReceiveMavMsg;
        outHilData.GpsPos[0]=gp.lat;
        outHilData.GpsPos[1]=gp.lon;
        outHilData.GpsPos[2]=gp.alt;
        outHilData.relative_alt = gp.relative_alt;
        outHilData.GpsVel[0]=gp.vx;
        outHilData.GpsVel[1]=gp.vy;
        outHilData.GpsVel[2]=gp.vz;
        outHilData.hdg = gp.hdg;
        break;
      }
```

[8]. 发送源码解析 — 发送一条 mavlink_hil_actuator_controls 消息：

```
void sendHILCtrlMessage(uint8_t modes, uint64_t flags, float ctrl[])
{
    mavlink_hil_actuator_controls_t hilctrl;
    hilctrl.mode = modes;
    hilctrl.flags = flags;
    for(int i=0;i<16;i++){
        hilctrl.controls[i]=ctrl[i];
    }
    mavlink_message_t mess;
    mavlink_msg_hil_actuator_controls_encode(SystemID, TargetCompID, &mess, &hilctrl);
    char buffer[500];
    memset(buffer,0,500);
    unsigned int length = mavlink_msg_to_send_buffer((uint8_t*)buffer, & mess);
    udp.writeDatagram(buffer,length);//通过 UDP 或者串口将 buffer 发送出去即可
}
```

[9]. MAVLink 消息包的 ID 列表：https://mavlink.io/en/messages/common.html



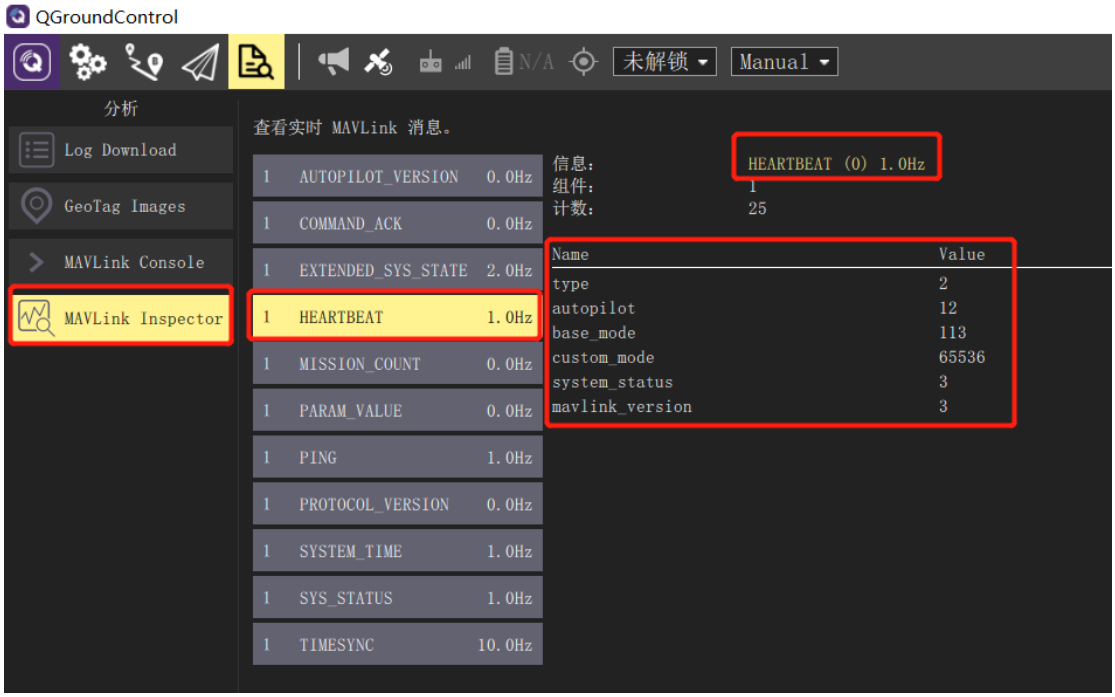
HEARTBEAT (#0) 心跳包，ID=0号消息

[Message] The heartbeat message shows that a system or component is present and responding. The type and autopilot fields (along with the message component id), allow the receiving system to treat further messages from this system appropriately (e.g. by laying out the user interface based on the autopilot). This microservice is documented at https://mavlink.io/en/services/heartbeat.html

Field Name	Type	Values	Description
type	uint8_t	MAV_TYPE	Vehicle or component type. For a flight controller component the vehicle type (quadrotor, helicopter, etc.). For other components the component type (e.g. camera, gimbal, etc.). This should be used in preference to component id for identifying the component type.
autopilot	uint8_t	MAV_AUTOPILOT	Autopilot type / class. Use MAV_AUTOPILOT_INVALID for components that are not flight controllers.
base_mode	uint8_t	MAV_MODE_FLAG	System mode bitmap.
custom_mode	uint32_t		A bitfield for use for autopilot-specific flags
system_status	uint8_t	MAV_STATE	System status flag.
mavlink_version	uint8_t_mavlink_version		MAVLink version, not writable by user, gets added by protocol because of magic data type: uint8_t_mavlink_version

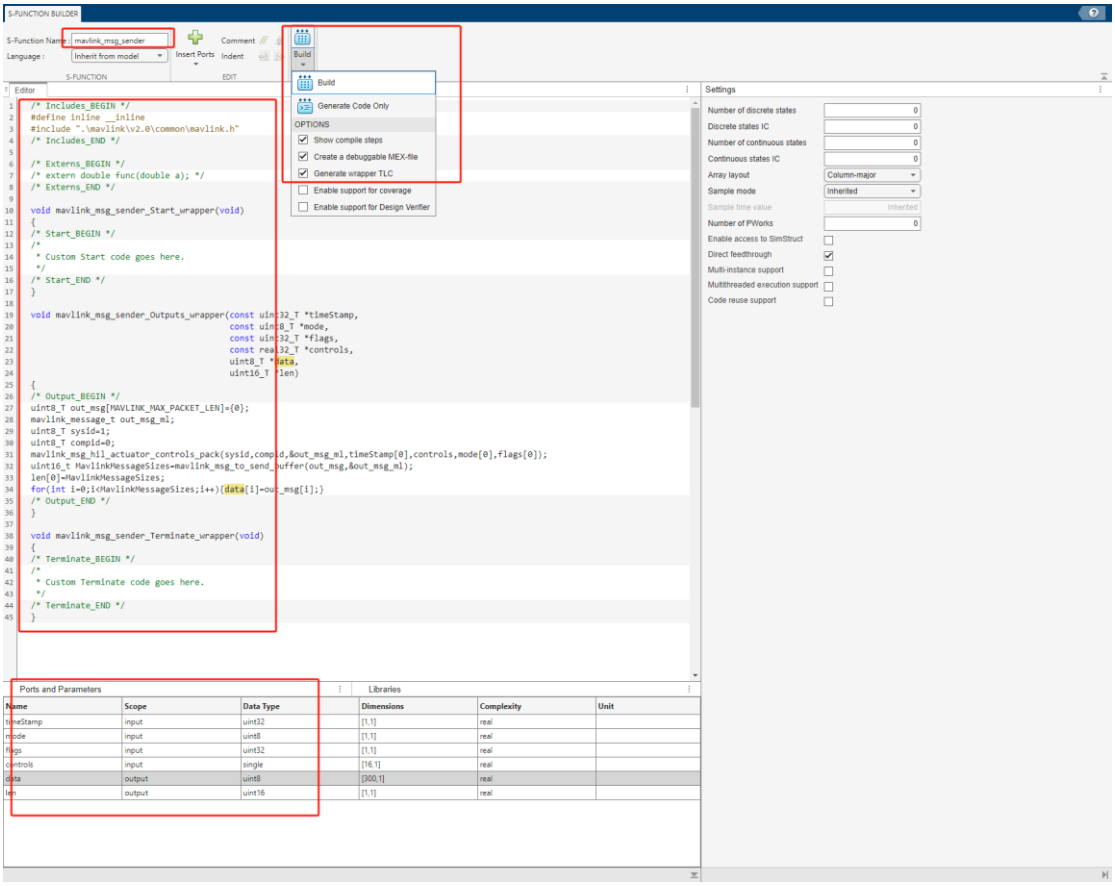
[10]. QGC 地面站查看 MAVLink 消息在 QGC 的 MAVLink Inspector 页面中可以浏览 Pixhaw

k 发送的所有 MAVLink 包，查看各个包的频率以及具体数值。



7. 常见问题

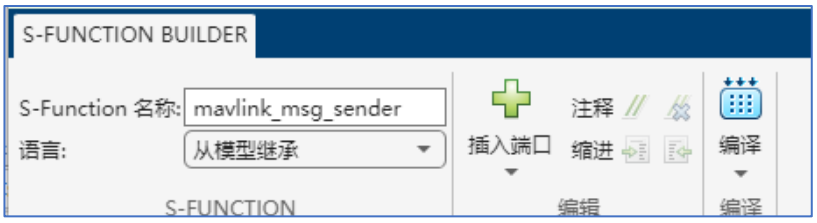
Q1.高版本 MATLAB 的 S-fuction 封装界面



封装 MAVLink 发送端模块

新版本 Step1 如上相同，其他的不同设置如下：

新建模块并添加输入输出端口



端口和参数设置：

端口和参数 | 库

名称	作用域	数据类型	维度	复/实性
timeStamp	input	uint32	[1,1]	real
mode	input	uint8	[1,1]	real
flags	input	uint32	[1,1]	real
controls	input	single	[16,1]	real
data	output	uint8	[300,1]	real
len	output	uint16	[1,1]	real

引入 Mavlink 头文件库

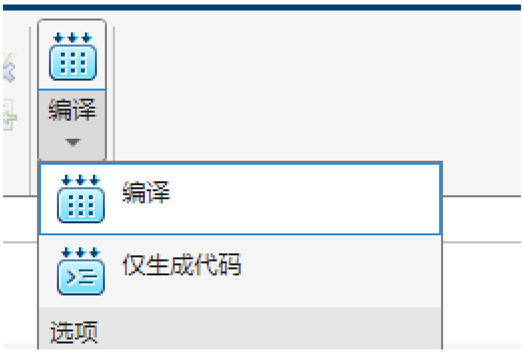
编辑器

```
1  /* Includes_BEGIN */
2  #include <math.h>
3  #define inline __inline
4  #include ".\mavlink\v2.0\common\mavlink.h"
5  /* Includes_END */
```

编写 C/C++代码以获取输入并打包输出

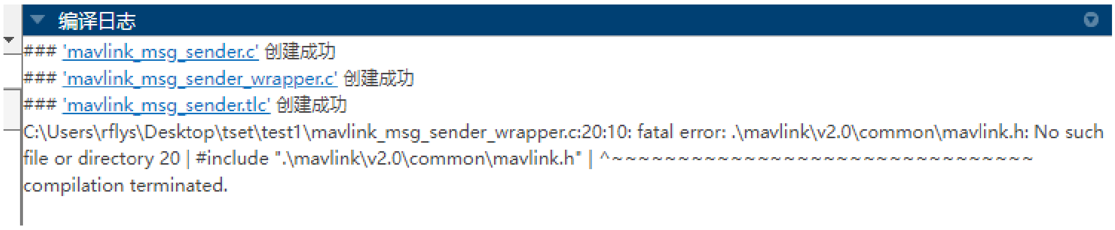
```
{
/* Output_BEGIN */
uint8_T out_msg[MAVLINK_MAX_PACKET_LEN]={0};
mavlink_message_t out_msg_ml;
uint8_T sysid=1;
uint8_T compid=0;
mavlink_msg_hil_actuator_controls_pack(sysid,compid,&out_msg_ml,timeStamp[0],controls,mod
 e[0],flags[0]);
uint16_t MavlinkMessageSizes=mavlink_msg_to_send_buffer(out_msg,&out_msg_ml);
len[0]=MavlinkMessageSizes;
for(int i=0;i<MavlinkMessageSizes;i++){data[i]=out_msg[i];}

 /* 此示例将输出设置为等于输入
 y0[0] = u0[0];
 对于复信号，使用: y0[0].re = u0[0].re;
 y0[0].im = u0[0].im;
 y1[0].re = u1[0].re;
 y1[0].im = u1[0].im;
 */
/* Output_END */
}
```

编译



编译输出文件与上述步骤产生文献相同

若出现如下错误，检查 mavlink 文件夹是否存在于当前目录，若没有，解压该文件夹到正确的目录下


编译日志

'mavlink_msg_sender.c' 创建成功
'mavlink_msg_sender_wrapper.c' 创建成功
'mavlink_msg_sender.tlc' 创建成功
C:\Users\rflys\Desktop\tset\test1\mavlink_msg_sender_wrapper.c:20:10: fatal error: .\mavlink\v2.0\common\mavlink.h: No such file or directory 20 | #include ".\mavlink\v2.0\common\mavlink.h" | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ compilation terminated.

封装 MAVLink 接收端模块

新建模块并添加输入输出端口


端口和参数 库

名称	作用域	数据类型	维度	复/实性
data	input	uint8	[300,1]	real
len	input	uint16	[1,1]	real
mode	output	uint8	[1,1]	real
timeStamp	output	uint32	[1,1]	real
flags	output	uint32	[1,1]	real
controls	output	single	[16,1]	real

引入 Mavlink 头文件库

编辑器

```
1  /* Includes_BEGIN */
2  #include <math.h>
3  #define inline __inline
4  #include ".\mavlink\v2.0\common\mavlink.h"
5  /* Includes_END */
6
```

编写 C/C++代码以获取输入并打包输出

S-FUNCTION 编辑 编译

编辑器

```
10
11  void mavlink_msg_receiver_Start_wrapper(void)
12  {
13  /* Start_BEGIN */
14  mavlink_message_t message;
15  mavlink_status_t status;
16  bool msgReceived = false;
17  for(int i=0;i<len[0];i++){
18      msgReceived = mavlink_parse_char(MAVLINK_COMM_1, (uint8_t)data[i], &message, &status);
19      if(msgReceived){
20          switch(message.msgid)
21          {
22              case MAVLINK_MSG_ID_HIL_ACTUATOR_CONTROLS:{
23                  mavlink_hil_actuator_controls_t hil_actuator_control;
24                  mavlink_msg_hil_actuator_controls_decode(&message, &hil_actuator_control);
25                  timeStamp[0] = hil_actuator_control.time_usec;
26                  mode[0] = hil_actuator_control.mode;
27                  flags[0] = hil_actuator_control.flags;
28                  for(int i=0;i<16;i++){
29                      controls[i]=hil_actuator_control.controls[i];
30                  }
31                  break;
32              }
33              default:{
34                  break;
35              }
36          }
37          msgReceived = false;
38
39          /*
40
41   * 此处显示自定义开始代码。
42   */
```

编译

编译输出文件与上述步骤产生文献相同