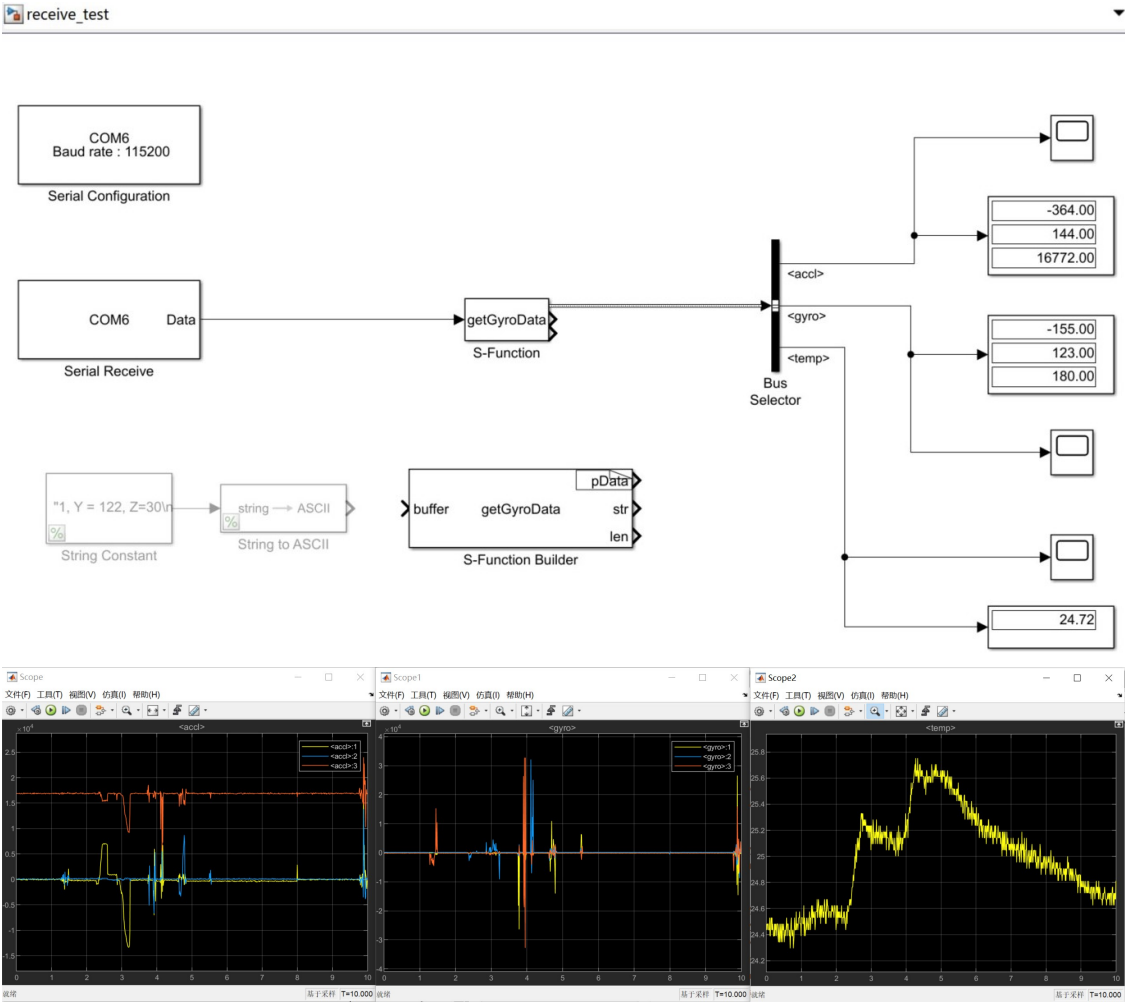


Simulink 中 S-Function 的使用入门

(以串口接收 MPU6050 六轴陀螺仪参数为实例)

1 引言

S-Function 允许使用自定义 C/C++ 函数作为传递函数，具有可移植性。也可以同样利用 MATLAB 函数进行相同的运算，看开发者熟悉程度而定。

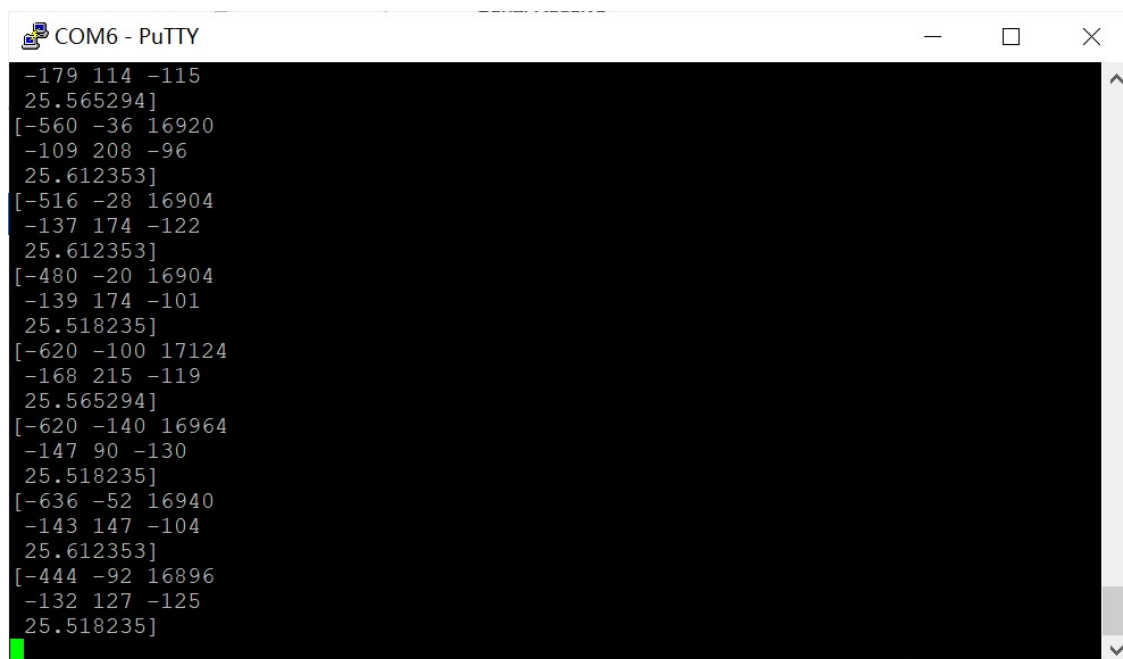


1.1 项目流程

1. 由系统串口接收数据包。
2. 通过 S-Function 自定义函数解析数据包，得到数据集合。
3. 数据集合总线输出，分为各部分数据进行可视化显示。

2 串口接收

首先需要通过 MPU6050 数据手册将其寄存器的各项输出数据通过 UART 读出，并将其处理为我们所需要的 7 项参数（加速度 Acc1. [X,Y,Z]，角加速度 Gyro. [X,Y,Z]，温度 Temp）。但是这并非本文的重点，因此本文假设已经处理好了上述数据，并且以字符打印的形式通过串口稳定输出，（通过 Raspberry Pico PI 预处理）如下：

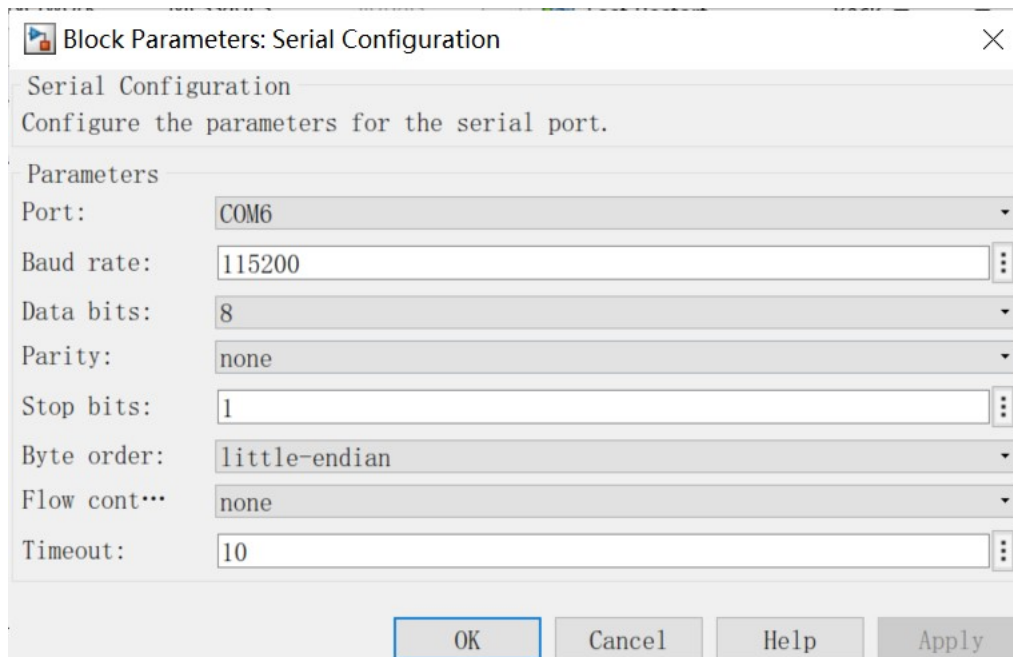


```
COM6 - PuTTY
-179 114 -115
25.565294]
[-560 -36 16920
-109 208 -96
25.612353]
[-516 -28 16904
-137 174 -122
25.612353]
[-480 -20 16904
-139 174 -101
25.518235]
[-620 -100 17124
-168 215 -119
25.565294]
[-620 -140 16964
-147 90 -130
25.518235]
[-636 -52 16940
-143 147 -104
25.612353]
[-444 -92 16896
-132 127 -125
25.518235]
```

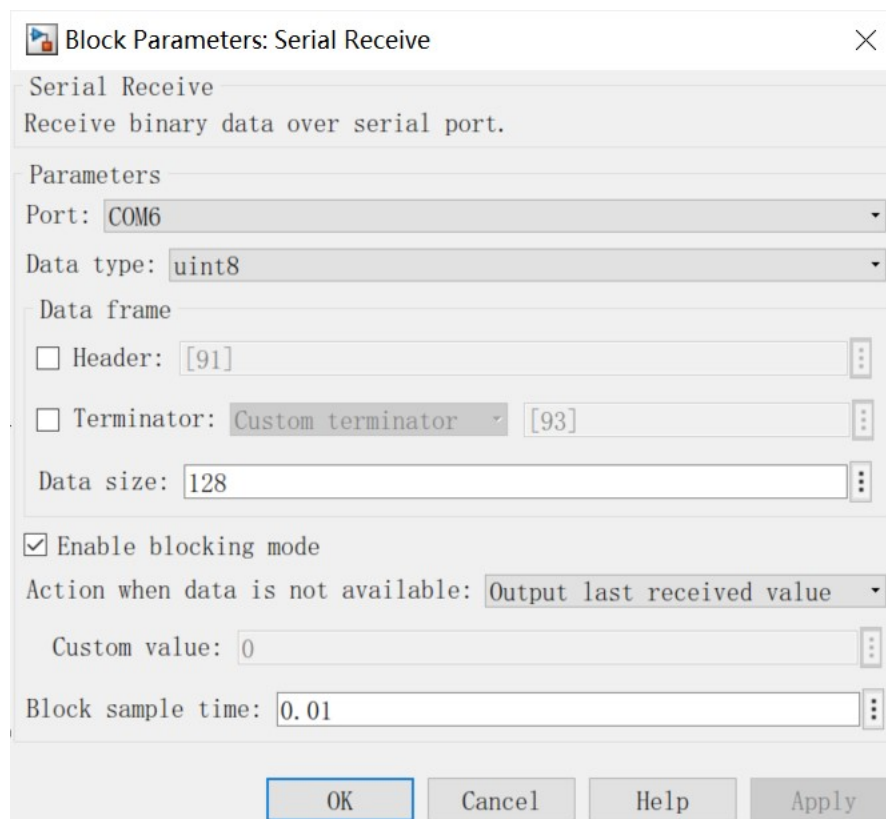
我们需要配置串口接收参数：

COM 口以用户本机分配为准

Serial Configuration 配置如图



Serial Receive 配置如图



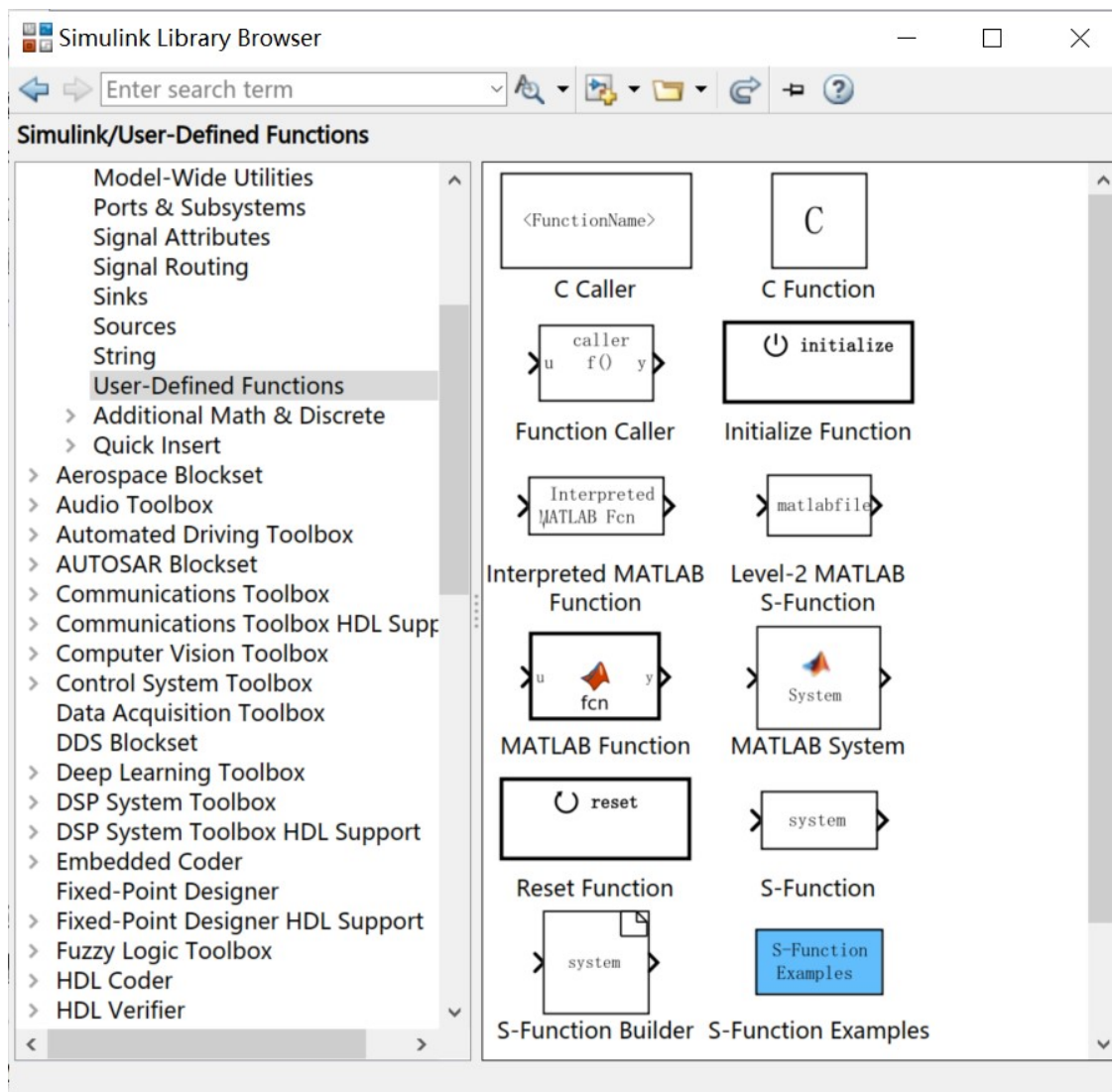
注意，由于是以字符 `char` 接收，因此选择类型为 `uint8`，**Data Size** 设置缓冲区大小，此处设

为 128 为例。同样可以设置数据包头和数据包尾，我们此处不做设置。

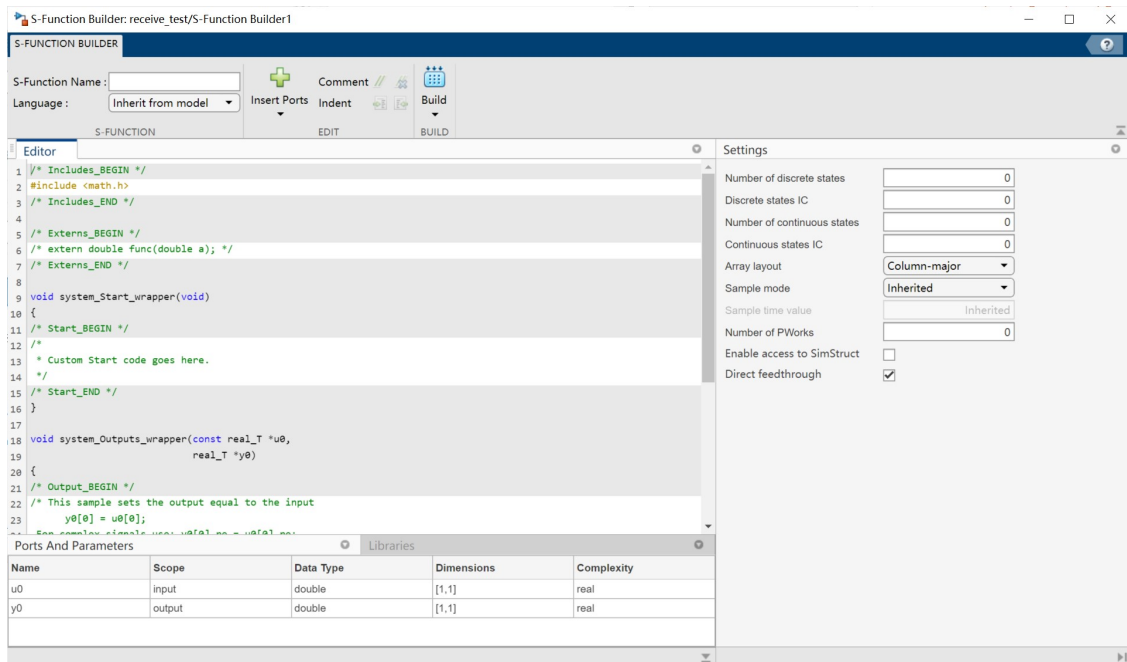
当然，在实际项目中会以更高效率的数据包传输，此处仅为示例作用，因此选用字符接收。

3 S-Function 函数示例

从 Library Browser 中找到 user-define function (用户自定义函数)，在系统仿真中这部分是较为常用的。



选择 S-Function Builder，双击点开面板。



填写函数名，选择语言 **C++**，即可 **Build** 编译。（初次需要按提示安装 **MinGW-w64** 方可编译，且安装目录不能带空格）



注意到，编辑器里面有三个子函数，分别封装 **S-Function** 的起始方程，输出方程和终止方程，在本实例中仅用到输出的封装，其他功能请自行学习浏览官方示例。

```
void xxx_Start_wrapper(void);
void xxx_Outputs_wrapper(const real_T *u0, real_T *y0);
void xxx_Terminate_wrapper(void);
```

注意到，输入输出参数都是以数组指针的形式传参的，带 **const** 表示输入参数，不可更改，不带的即表示输出参数，可更改。函数的目的是按次调用这三个子函数。我们可以先调试好 **C++** 程序

再整合。

在本实例中，输入参数为大小为 **128*1** 的字符数组 (**uint8**)，输出参数为包括三个参数的总线 (**Bus:gyroData**)，以结构体形式调用。

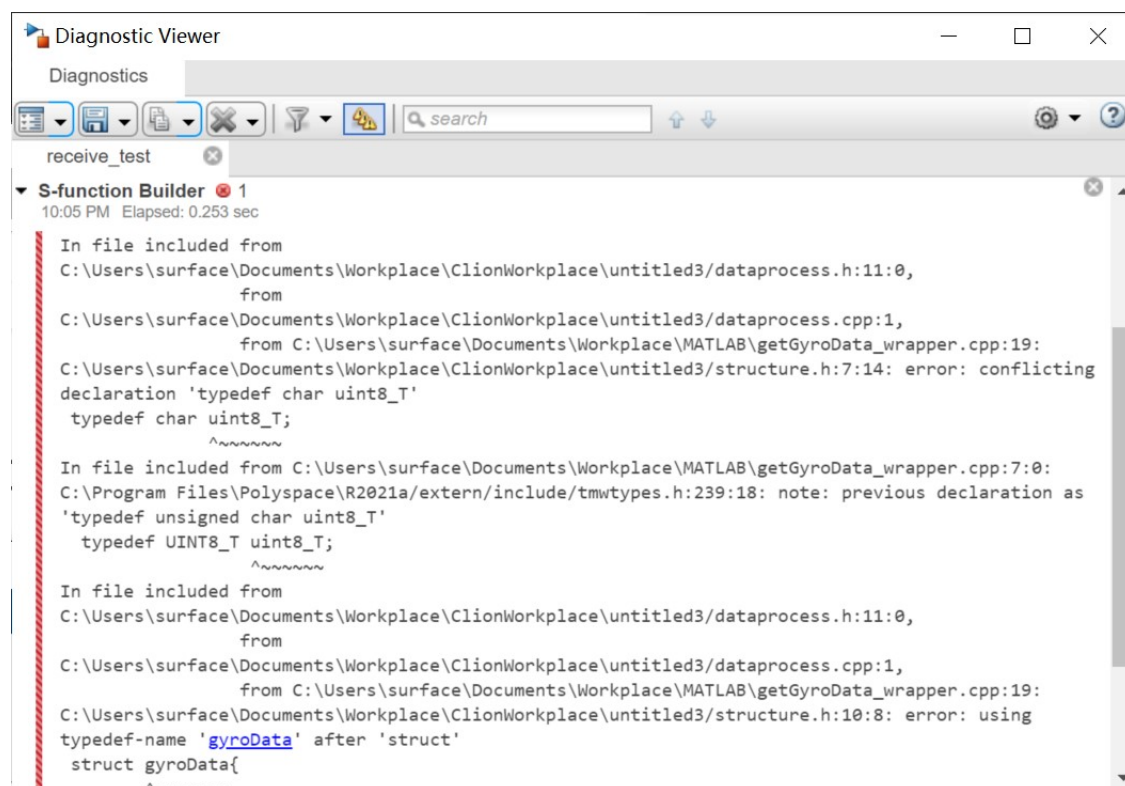
在自己工程文件中定义头文件 (**structure.h**)，用于类型的转换和结构体的定义等，这部分内容是为了便于程序能在本地调试成功，而由于在 **Simulink** 中这部分内容已有定义，因此在 **Simulink** 的 **S-Function Builder** 中编译的时候是需要将其之注释掉的。**structure.h**:

```
#ifndef _STRUCTURE_H
#define _STRUCTURE_H
typedef char uint8_T;
typedef short int16;

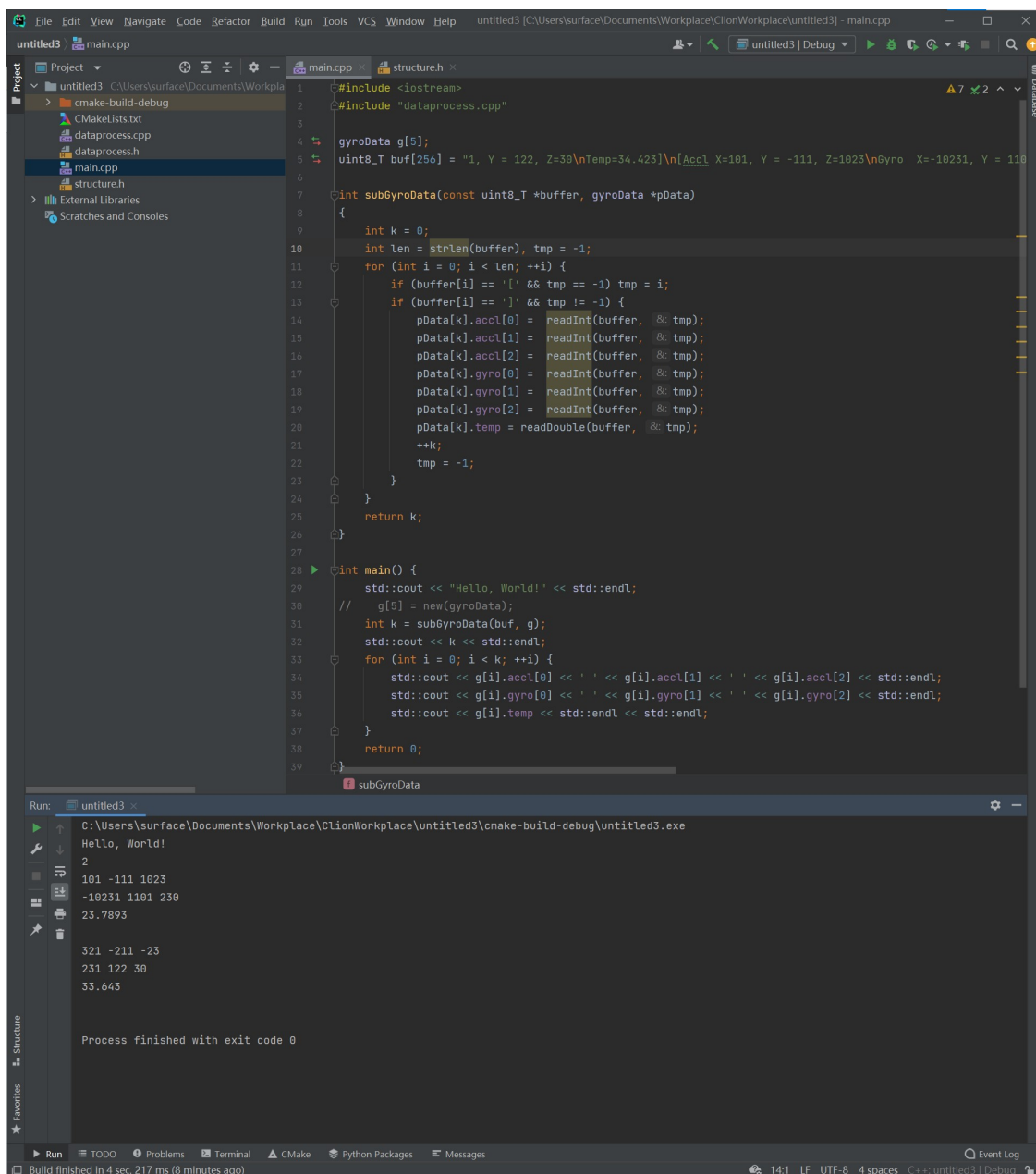
struct gyroData{
    int16_t acc1[3], gyro[3];
    double temp;
};

#endif // _STRUCTURE_H
```

如果不注释掉，在编译时会由于变量类型冲突而编译失败。



在本地 C++ 工程调试成功，即可移植。



```
#include <iostream>
#include "dataprocess.cpp"

gyroData g[5];
uint8_T buf[256] = "1, Y = 122, Z=30\nTemp=34.423\n[Accel X=101, Y = -111, Z=1023\nGyro X=-10231, Y = 1101, Z=30\nTemp=33.643\n";

int subGyroData(const uint8_T *buffer, gyroData *pData)
{
    int k = 0;
    int len = strlen(buffer), tmp = -1;
    for (int i = 0; i < len; ++i) {
        if (buffer[i] == '[' && tmp == -1) tmp = i;
        if (buffer[i] == ']' && tmp != -1) {
            pData[k].accel[0] = readInt(buffer, 8, tmp);
            pData[k].accel[1] = readInt(buffer, 8, tmp);
            pData[k].accel[2] = readInt(buffer, 8, tmp);
            pData[k].gyro[0] = readInt(buffer, 8, tmp);
            pData[k].gyro[1] = readInt(buffer, 8, tmp);
            pData[k].gyro[2] = readInt(buffer, 8, tmp);
            pData[k].temp = readDouble(buffer, 8, tmp);
            ++k;
            tmp = -1;
        }
    }
    return k;
}

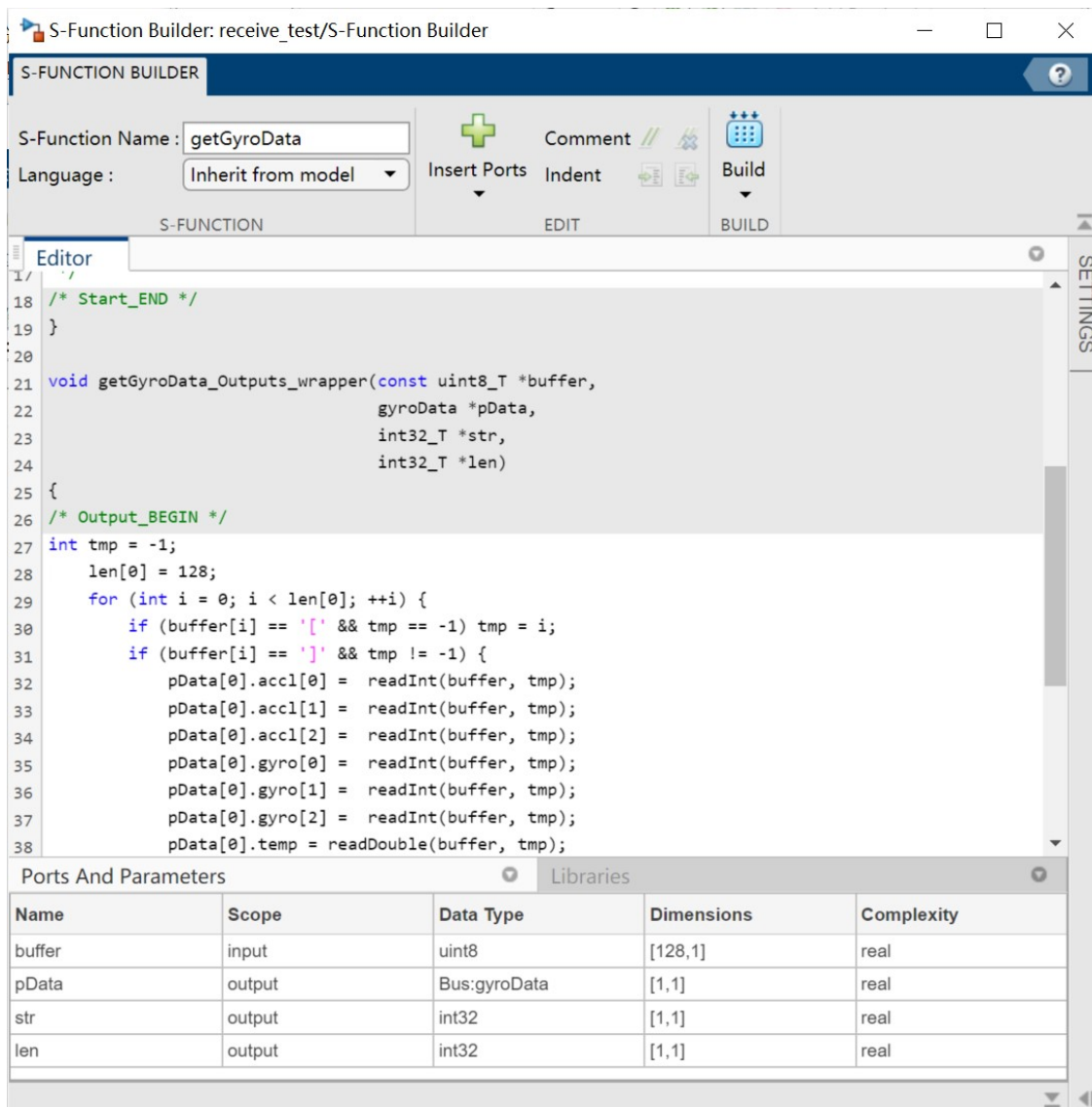
int main() {
    std::cout << "Hello, World!" << std::endl;
    // g[5] = new(gyroData);
    int k = subGyroData(buf, g);
    std::cout << k << std::endl;
    for (int i = 0; i < k; ++i) {
        std::cout << g[i].accel[0] << ' ' << g[i].accel[1] << ' ' << g[i].accel[2] << std::endl;
        std::cout << g[i].gyro[0] << ' ' << g[i].gyro[1] << ' ' << g[i].gyro[2] << std::endl;
        std::cout << g[i].temp << std::endl << std::endl;
    }
    return 0;
}
```

Run: untitled3 x

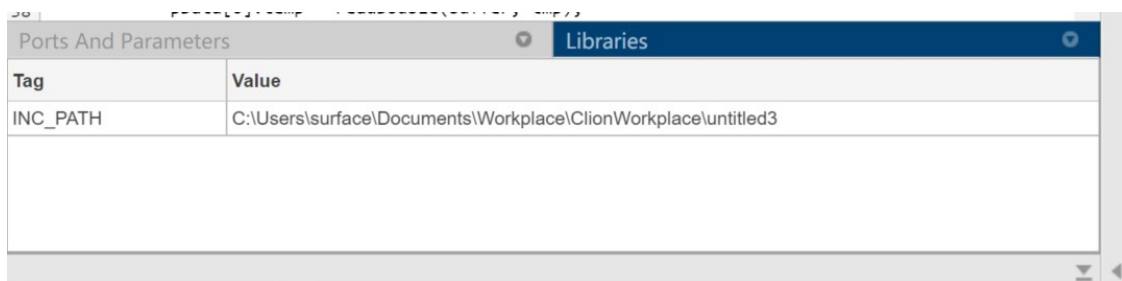
C:\Users\surface\Documents\Workplace\ClionWorkplace\untitled3\cmake-build-debug\untitled3.exe

Hello, World!
2
101 -111 1023
-10231 1101 230
23.7893
321 -211 -23
231 122 30
33.643
Process finished with exit code 0

调试成功后，将其移植至 S-Function Builder 中。注意参数类型、维度都要一一匹配。需要用到的头文件和引入文件也需要在 include 部分添加进来。

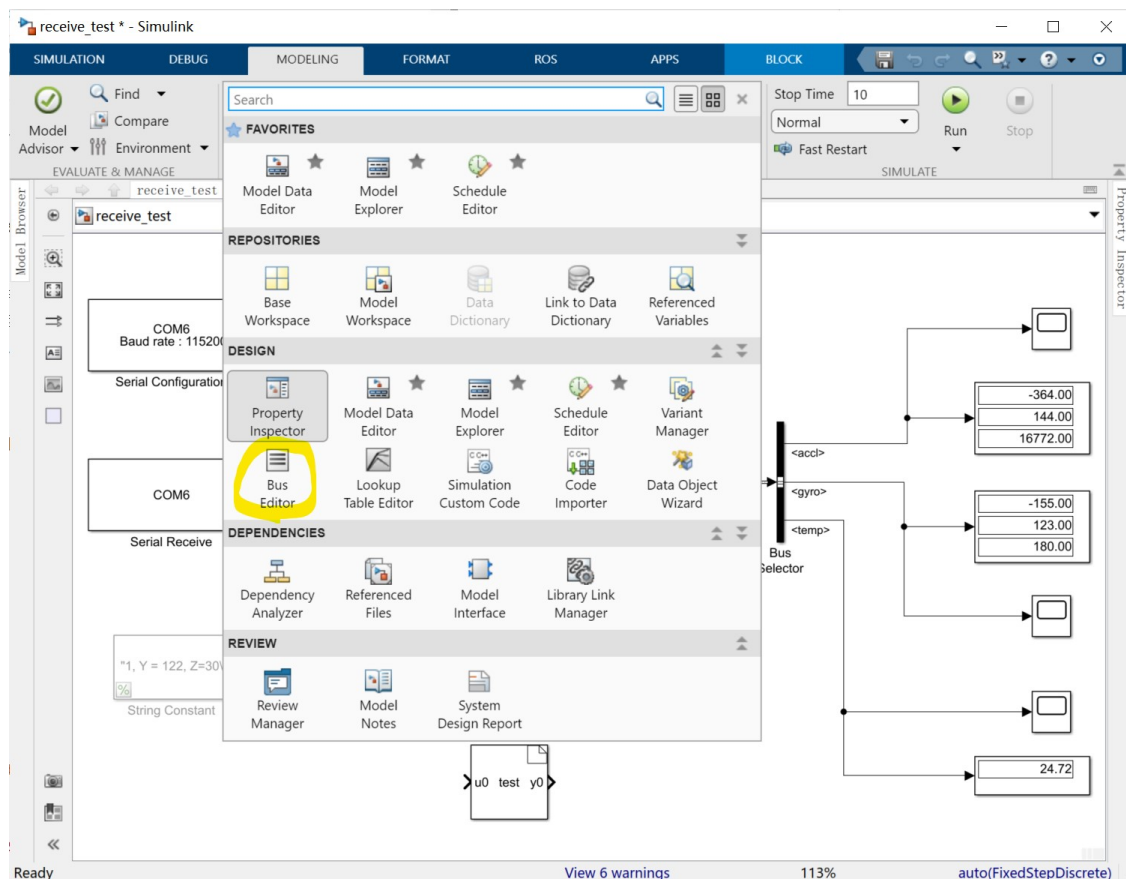


如果 C++ 工程包含其他文件，将工程文件目录添加至 INC_PATH，注意文件目录最好不要含有空格，否则再次打开时可能会产生修正而出错。



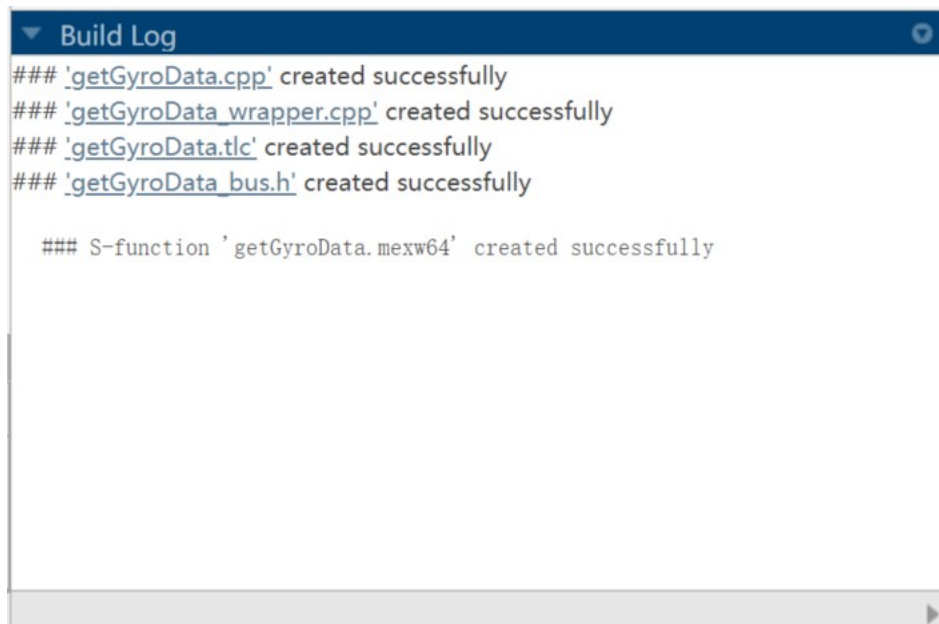
点击编译，提示 ERROR: Bus object for output port 1 gyroData does not exist in workspace，即：总线未被添加至工作区导致编译失败。

手动添加的办法，即打开 **MODELING** 中的 **Bus Editor** 选项进行添加，注意总线名称 **Object Name** 以及元素名称，以及其类型、维度，均设置匹配。当然，也有通过代码区更方便添加的办法请自动查阅相关文档。

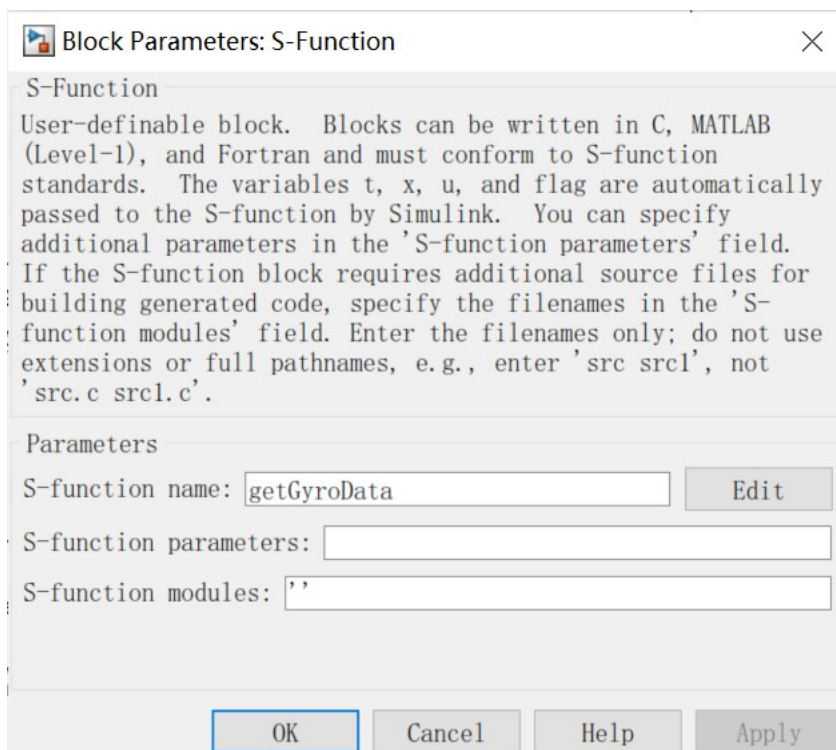


添加后请将其保存为 **.mat** 文件方便日后直接将设置好的总线导入。

如果没有问题，会显示 **S-Function** 编译成功。



这时，我们可以用 **S-Function** 模块直接填写函数名，直接调用，模块会自动产生相应数目的输入输出端口，可供检查。（虽然 **S-Function Builder** 也可直接使用）

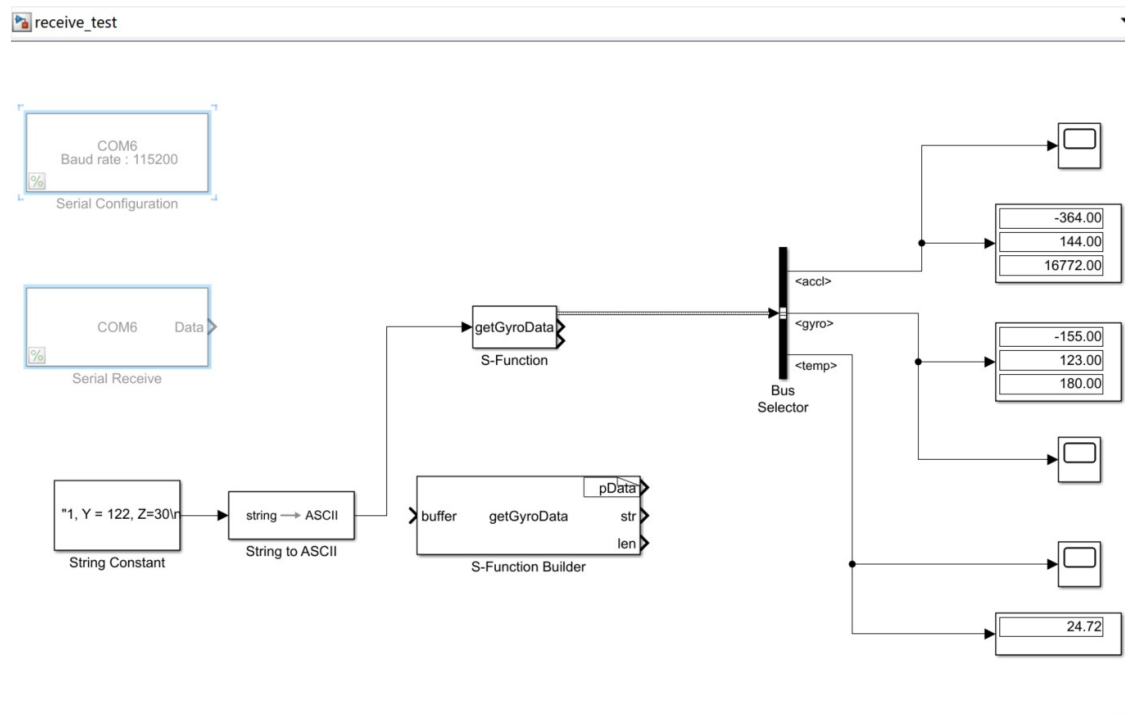


这样，我们的 **S-Function** 就编写完成。

4 数据可视化

我们利用 **Bus Selector** 模块进行对总线拆分，用 **Display** 模块显示文本（数组），用 **Scope** 模块进行示波器图像绘制，这些属于基本操作，就不详加叙述了。

这里提到一个调试的方法，即先使用静态文本（此处用了字符串常量 **String Constant**）进行调试，确认 **S-Function** 功能无误即可接入动态数据进行观察。



5 最后

文章末尾，附上相关的工程文件供参考。

- bus_gyroData.mat gyroData 总线
- getGyroData_ 由 S-Function 生成的函数
- receive_test.slxc 接收端 Simulink 工作区面板
- gyroDataProcess 用于产生 S-Function 的 C++ 文件

请访问我的 Github URL 以下载资源: <https://github.com/Chiron19/Simulink>

感谢阅读！欢迎分享。

