

Simulink 内嵌 C 语言

目 录

前 言	1
C Function Block	2
一、创建 C 文件	2
二、配置模型	2
三、构建模型	3
C Caller Block	5
一、创建 C 文件	5
二、配置模型	5
三、构建模型	6
Legacy_Code Tool	7
一、创建 C 文件	7
二、生成嵌 C 的 S 函数和 TLC	7
三、构建模型	7
Stateflow	8
一、创建 C 文件	8
二、模型配置	8
三、构建模型	8
MATLAB Function	10
一、创建 C 文件	10
二、模型配置	10
三、构建模型	10
S Fucntion Builder	12
一、创建 C 文件	12
二、生成嵌 C 的 S 函数和 TLC	12
三、构建模型	13
C Mex S Function	14
一、C Function 嵌的 C 文件	14
1、模型配置	14
2、C Mex S Function	14
3、TLC 编写	15
4、构建模型	16
二、C Caller 嵌的 C 文件	16
1、模型配置	16
2、C Mex S Function	16
3、TLC 编写	18
4、构建模型	18
结 语	20

前 言

基于模型的设计，基于模型的系统工程和数字孪生发展的如火如荼，各种概念层出不穷。很对软件厂商都开发自己的或类似的基于模型的设计软件，MathWorks 旗下的 Simulink、Ansys 旗下的 SCADE 和同元软控的 MWorks 等等，基于模型的设计是一种围绕模型搭建展开的一种项目开发方法。这种方法可必变繁琐的代码编写和调试过程，可极大的提高项目开发效率。

Simulink、Stateflow 和 Embedded Coder 等工具箱提供的基于模型设计的开发给汽车电子开发带来极大的便利，Simulink 可在仿真验证完成后将模型之间生产代码，此举极大的节省的人力和花费但，有时开发者手中有及其稳定和高效的 C 代码，将已有的 C 代码翻译成模型仿真验证将极其费时。另外，Simulink 的代码无法直接生成的驱动底层的 C 代码。

因此，本文提供了七种 Simulink 嵌入 C 代码的方式，给出的例子部分来自于帮助文档，文档中使用的版本是 MATLAB R2021A。为便于复现，将提供提下代码用于创建新的模型。

```
open_system(new_system)
set_param(bdroot, 'SolverType', 'Fixed-step')
set_param(bdroot, 'Solver', 'FixedStepDiscrete')
set_param(bdroot, 'GenCodeOnly', 'on')
set_param(bdroot, 'SystemTargetFile', 'ert.tlc')
set_param(bdroot, 'GenerateReport', 'on')
set_param(bdroot, 'LaunchReport', 'on')
set_param(bdroot, 'SaveTime', 'off')
set_param(bdroot, 'SaveOutput', 'off')
set_param(bdroot, 'SignalLogging', 'off')
set_param(bdroot, 'DSMLogging', 'off')
set_param(bdroot, 'ReturnWorkspaceOutputs', 'off')
set_param(bdroot, 'IncludeHyperlinkInReport', 'on')
set_param(bdroot, 'GenerateTraceInfo', 'on')
set_param(bdroot, 'RTWUseSimCustomCode', 'on')
```

C Function Block

C Function block 位于 Simulink 库的 User-Defined Functions 子目录下，新建模型后将该模块拖到模型文件即可。其它详见帮助文档。

一、创建 C 文件

创建如下内容的头文件，并命名为 data_array.h。

```
/* Define a struct called DataArray */
typedef struct DataArray_tag {
    /* Define a pointer called pData */
    double* pData;
    /* Define the variable length */
    int length;
} DataArray;

/* Function declaration */
double data_sum(DataArray data);
```

创建如下内容的源文件，并命名为 data_array.c。

```
#include "data_array.h"

/* Define a function that takes in a struct */
double data_sum(DataArray data)
{
    /* Define 2 local variables to use in the function */
    double sum = 0.0;
    int i;
    /* Calculate the sum of values */
    for (i = 0; i < data.length; i++) {
        sum = sum + data.pData[i];
    }
    /* Return the result to the block */
    return sum;
}
```

二、配置模型

使用前言给出的代码生成空白模型后，在 Simulink 的 MODELING 面板上单击 Model Setting 按钮打开模型设置（或者使用快捷键 Ctrl+E）。在模型配面板的 Simulation Target->Insert custom C code in generated->Header file 框中输入 #include "data_array.h"、Simulation Target->Additional build information -> Source file 框中输入 data_array.c 并勾选 Code Generation->Custom Code->Use the same custom code setting as Simulation target 框。脚本实现如下：

```
set_param(bdroot, 'SimCustomHeaderCode', '#include "data_array.h"')
set_param(bdroot, 'SimUserSources', 'data_array.c')
set_param(bdroot, 'RTWUseSimCustomCode', 'on')
```

三、构建模型

在 Simulink 模块库中，向已配置好的空白模型加入一个输入两个输出，并将 C Function 模块也加入模型。并双击打开 C Function 模块，在 Output Code 中填入如下：

```
DataArray dataArr;
/* store the length and data coming in from the input port */
dataArr.pData = &data[0];
dataArr.length = length;

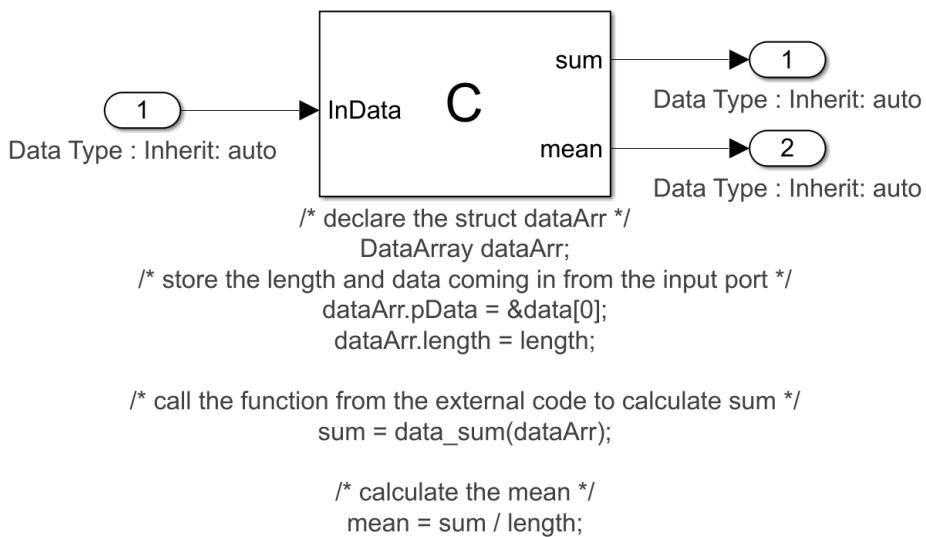
/* call the function from the external code to calculate sum */
sum = data_sum(dataArr);

/* calculate the mean */
mean = sum / length;
```

在 Symbols 中添加四个变量，第一个设为输入，第二个设为 Constant，第三、四个设为输入，具体如下图：

Symbols:						
Name	Scope	Label	Type	Size	Port	
data	Input	InData	double	-1	1	
length	Constant	size(data)	int32	1	-	
mean		mean	double	1	2	
sum	Output	sum	double	1	1	

图中使用颜色框住需要添加后修改。最后将一个输入、两个输出和 C Function 模块相连，结果如下：



最后 Ctrl+B 生成代码即可。生成报告如下：

```

42  /* call the function from the external code to calculate sum */
43  expl_temp.length = 100;
44  expl_temp.pData = dataArr_pData;
45
46  /* Outport: '<Root>/Out1' incorporates:
47   * CFunction: '<Root>/C_Function'
48   */
49  C_Function_Call_C_Y.Out1 = data_sum(expl_temp);
50
51  /* Outport: '<Root>/Out2' incorporates:
52   * CFunction: '<Root>/C_Function'
53   * Outport: '<Root>/Out1'
54   */
55  /* calculate the mean */
56  C_Function_Call_C_Y.Out2 = C_Function_Call_C_Y.Out1 / 100.0;
57 }
58
59 /* Model initialize function */
60 void C_Function_Call_C_initialize(void)
61 {
62     /* (no initialization code required) */
63 }
64
65 /* Model terminate function */
66 void C_Function_Call_C_terminate(void)
67 {
68     /* (no terminate code required) */
69 }
--
```

图中为生成的报告，其中蓝色的框是 c 文件中的函数，红色的框是源文件。
生成代码后使用如下命令验证生成的代码是否可编译链接通过：

```
rtwrebuild(bdroot)
```

C Caller Block

C Caller block 位于 Simulink 库的 User-Defined Functions 子目录下，新建模型后将该模块拖到模型文件即可。其它详见帮助文档。

一、创建 C 文件

创建如下内容的头文件，并命名为 ex_myTypes_LCT.h。

```
#ifndef _MY_TYPES_H_
#define _MY_TYPES_H_

typedef struct {
    double sig1;
    double sig2;
} sigStructType;

typedef struct {
    double param1;
    double param2;
    double param3;
} paramStructType;

void myFcn(sigStructType *in, paramStructType *params,
           sigStructType *out);

#endif
```

创建如下内容的源文件，并命名为 ex_mySrc_LCT.c。

```
#include "ex_myTypes_LCT.h"

void myFcn(sigStructType *in, paramStructType *params,
           sigStructType *out)
{
    out->sig1 = in->sig1 * params->param1;
    out->sig2 = in->sig2 * params->param2 + params->param3;
}
```

二、配置模型

使用前言给出的代码生成空白模型后，在 Simulink 的 MODELING 面板上单击 Model Setting 按钮打开模型设置（或者使用快捷键 Ctrl+E）。在模型配面板的 Simulation Target->Insert custom C code in generated->Header file 框中输入 #include "ex_myTypes_LCT.h"、Simulation Target->Additional build information -> Source file 框中输入 ex_mySrc_LCT.c 并勾选 Code Generation->Custom Code->Use the same custom code setting as Simulation target 框。脚本实现如下：

```
set_param(bdroot, 'SimCustomHeaderCode', '#include "ex_myTypes_LCT.h"')
set_param(bdroot, 'SimUserSources', 'ex_mySrc_LCT.c')
set_param(bdroot, 'RTWUseSimCustomCode', 'on')
```

由于源文件中参数也作为输入，而在 Simulink 中可配置参数，可用此脚本创建：

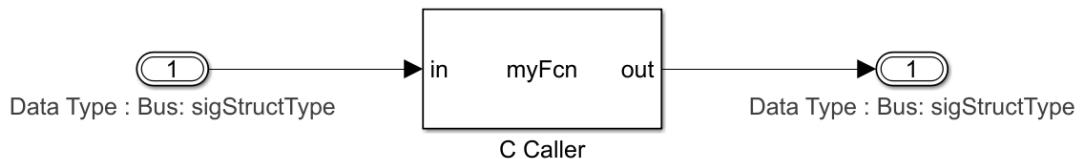
```
Simulink.importExternalCTypes('ex_myTypes_LCT.h');
param = struct;
param.param1 = 15;
param.param2 = 15;
param.param3 = 15;
p = Simulink.Parameter;
p.DataType = 'Bus: paramStructType';
p.Value = param;
```

三、构建模型

在 Simuink 模块库中，向已配置好的空白模型加入一个输入一个输出，并将 C Caller 模块也加入模型。并双击打开输入模块，单击 Signal Attributes，在 Data Type 中输入 Bus: sigStructType，在输出模块做相同的数据类型配置。双击 C Caller 模块，单击右侧更新，之后 Function name 选择 myFcn，展开 Port specification:

Port specification:				
Name	Scope	Label	Type	Size
in	Input	in	Bus: sigStruc...	1
params	Parameter	params1	Bus: paramS...	1
out	Output	out	Bus: sigStruc...	1

如有不一样，需按照上图修改。单击 OK 后再次双击打开，把 p 替换 param1 的编辑框中。整个模型如下：



最后便可仿真和代码生成；

Legacy_Code Tool

Legacy_code 是脚本集成已有 C 代码工具，借助 legacy_code 可自动生成 S Function 和 TLC，非常便捷，其它详见帮助文档。

一、创建 C 文件

创建如下内容的头文件，并命名为 ex_myTypes_LCT.h，见 C Caller block；

创建如下内容的源文件，并命名为 ex_mySrc_LCT.c，见 C Caller block；

二、生成嵌 C 的 S 函数和 TLC

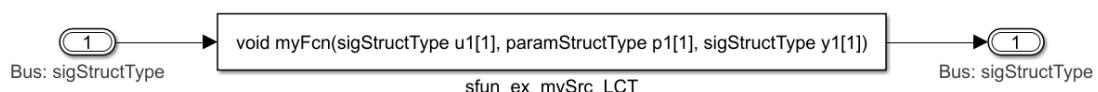
使用如下脚本即可生成 S 函数和 TLC：

```
Simulink.importExternalCTypes('ex_myTypes_LCT.h');
def = legacy_code('initialize');
def.SFunctionName = 'sfun_ex_mySrc_LCT';
def.SourceFiles = {'ex_mySrc_LCT.c'};
def.HeaderFiles = {'ex_myTypes_LCT.h'};
def.OutputFcnSpec = ['void myFcn(sigStructType u1[1],',...
    'paramStructType p1[1], sigStructType y1[1])'];
legacy_code('generate_for_sim',def);
legacy_code('sfcn_tlc_generate',def)
legacy_code('slblock_generate', def);
structParam = struct;
structParam.param1 = 15;
structParam.param2 = 20;
structParam.param3 = 5;
```

运行脚本之后会生成一个新的模型，双击生成的模块，将 P1 编辑框的 0 改成 structParam。

三、构建模型

在 Simuink 模块库中，向已配置好的空白模型加入一个输入一个输出，并将上一节中生成修改模块复制到模型。并双击打开输入模块，单击 Signal Attributes，在 Data Type 中输入 Bus: sigStructType，在输出模块做相同的数据类型配置。连接如下图所示：



最后便可仿真和代码生成；

Stateflow

Stateflow 提供了一种图形语言，包括状态转移图、流程图、状态转移表和真值表，主要用于状态机与流程图对决策逻辑进行建模仿真，支持 C 语言和 M 脚本，Stateflow 嵌 C 时需要将 Stateflow 默认的 M 语言改成 C 语言。其它详见帮助文档。

一、创建 C 文件

创建如下内容的头文件，并命名为 ex_myTypes_LCT.h，见 C Caller block；

创建如下内容的源文件，并命名为 ex_mySrc_LCT.c，见 C Caller block；

二、模型配置

使用前言给出的代码生成空白模型后，在 Simulink 的 MODELING 面板上单击 Model Setting 按钮打开模型设置（或者使用快捷键 Ctrl+E）。在模型配面板的 Simulation Target->Insert custom C code in generated->Header file 框中输入 #include "ex_myTypes_LCT.h"、Simulation Target->Additional build information -> Source file 框中输入 ex_mySrc_LCT.c 并勾选 Code Generation->Custom Code->Use the same custom code setting as Simulation target 框。脚本实现如下：

```
set_param(bdroot, 'SimCustomHeaderCode', '#include "ex_myTypes_LCT.h"')
set_param(bdroot, 'SimUserSources', 'ex_mySrc_LCT.c')
set_param(bdroot, 'RTWUseSimCustomCode', 'on')
```

由于源文件中参数也作为输入，而在 Simulink 中可配置参数，可用此脚本创建：

```
Simulink.importExternalCTypes('ex_myTypes_LCT.h');
param = struct;
param.param1 = 15;
param.param2 = 15;
param.param3 = 15;
```

三、构建模型

在 Simuink 模块库中，向已配置好的空白模型加入一个输入一个输出。并双击打开输入模块，单击 Signal Attributes，在 Data Type 中输入 Bus: sigStructType，并在输出模块做相同的数据类型配置。将 Simulink 模块库的 Stateflow 工具箱中将 Chart 加入到已配置模型，并双击打开 Chart，在 Chart 内部右键单击 properties 打开 Stateflow 配置选项。将 Action Language 设置为 C，使能 Enable C-bit operations。

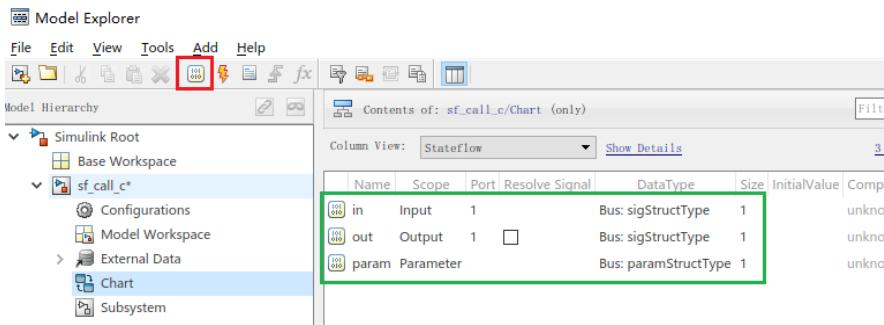
单击 OK 之后 State flow 编辑界面左下角会出现 C 的标识（表示设置成功）。从 Stateflow 左侧的一列按钮中找出默认转移（Default transition）, 单击按钮将默认转移添加到 Chart 中，并单击默认转移将下列语句添加到默认转移线上：

```
{myFcn(&in,&param,&out);}
```

添加完之后 chart 如下图所示：



Chart 还需要在模型资源管理中配置 Chart 的输入和输出，Simulink 顶部 MODELING->DESIGN->Model Explorer (或者使用命令 sfexplr、daexplr) 打开。在模型资源管理器的 Model Hierarchy 面板中选中 Chart。在顶部按钮挑中找到 Add Data 并单击三次添加三个变量。分别命名为 in、out 和 param，其中 Scope 分别设置为 Input、Output 和 Parameter、Size 全部设置为 1、数据类型分别设置为 Bus: sigStructType、Bus: sigStructType 和 Bus: paramStructType。最终模型资源管理器的显示如下图所示：



图中红色的框是 Add Data，绿色的框是设置的输入输出类型。将已加入的输入和输出与 Chart 连接，最终模型布局如下：



最后便可仿真和代码生成；

MATLAB Function

MATLAB Function 是 Simulink 提供调用脚本的模块，可借助该模块用 C 文件，但对指针支持不友好。其它详见帮助文档。

一、创建 C 文件

创建如下内容的头文件，并命名为 doubleIt.h。

```
#ifndef MYFN
#define MYFN
    double doubleIt(double u);
#endif
```

创建如下内容的源文件，并命名为 doubleIt.c。

```
#include "doubleIt.h"
double doubleIt(double u)
{
    return(u*2.0);
}
```

二、模型配置

使用前言给出的代码生成空白模型后，在 Simulink 的 MODELING 面板上单击 Model Setting 按钮打开模型设置（或者使用快捷键 Ctrl+E）。在模型配面板的 Simulation Target->Insert custom C code in generated->Header file 框中输入 #include "doubleIt.h"、Simulation Target->Additional build information -> Source file 框中输入 doubleIt.c 并勾选 Code Generation->Custom Code->Use the same custom code setting as Simulation target 框。脚本实现如下：

```
set_param(bdroot, 'SimCustomHeaderCode', '#include "doubleIt.h"')
set_param(bdroot, 'SimUserSources', 'doubleIt.c')
set_param(bdroot, 'RTWUseSimCustomCode', 'on')
```

三、构建模型

在 Simuink 模块库中，向已配置好的空白模型加入一个输入一个输出，并双击打开输入模块，单击 Signal Attributes，在 Data Type 选择 double，在输出模块做相同的数据类型配置。将 MATLAB Function 添加到模型，双击打开编辑如下：

```
function y = callingDoubleIt(u)
y = 0.0;
y = coder.ceval('doubleIt',u);
```

关闭脚本编辑器，回到 Simulink 编辑区，将输入输出和 MATLAB Function

相连如下图所示：



S Function Builder

S Function builder 位于 Simulink 库的 User-Defined Functions 子目录下，新建模型后将该模块拖到模型文件即可。其它详见帮助文档。

一、创建 C 文件

创建如下内容的头文件，并命名为 ex_myTypes_LCT.h，见 C Caller block；

创建如下内容的源文件，并命名为 ex_mySrc_LCT.c，见 C Caller block；

二、生成嵌 C 的 S 函数和 TLC

在 Simuink 模块库中，将 S Function Builder 加入到脚本生成的空白模型中，双击打开模块的配置界面，

首先，指定生成 S 函数的名字和生成语言类型。S-Function builder 设置为 sf_b (其它也可)、Language 选择 C 语言。

其次，设置 S 函数所需的源文件。单击 Libraryies 框，单击顶部的 Insert Paths 点击 Add Enrty，此时会在 Libraries 中 Tag 列出现 ENTRY，双击右侧将 ex_mySrc_LCT.c 填入即可。

再次，设置 S 函数的输入和输出，单击 Ports And Parameters，单击顶部的 Insert Ports 添加两个输入 (u0 和 u1) 和一个输出 y0，数据分别设置为 Bus:sigStructType、Bus:paramStructType 和 Bus:sigStructType，维度全部设置为 [1 ,1]。

最后调用源文件并生成 S 函数和 TCL，选中 Editor 框，在 Includes-END 上一行输入：

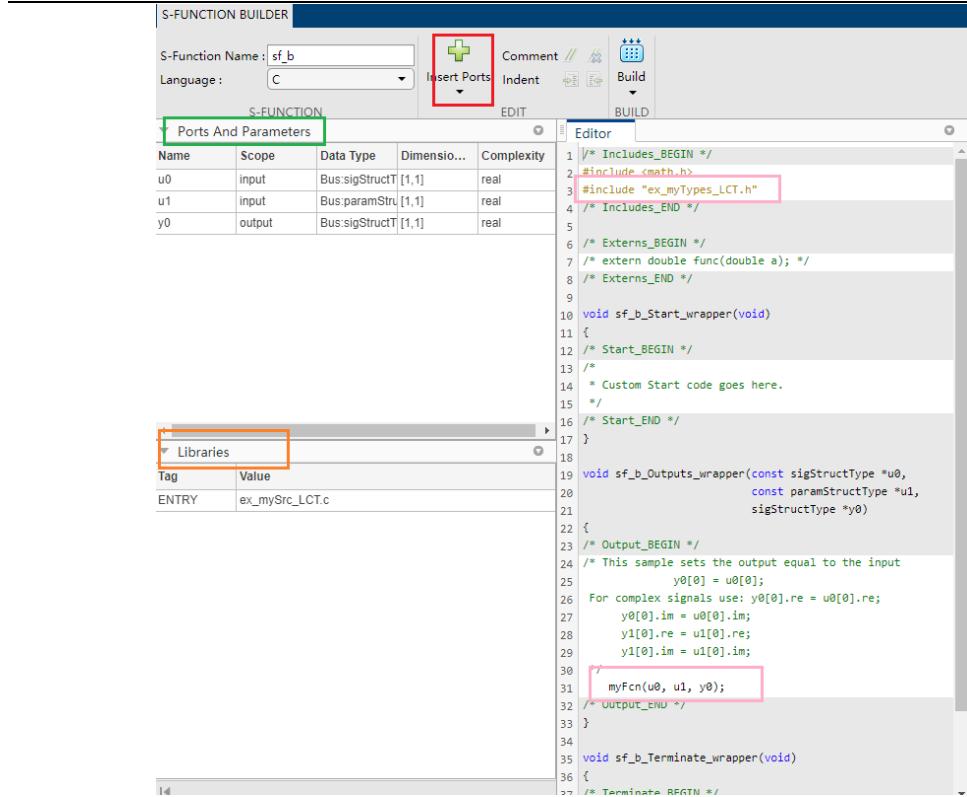
```
#include "ex_myTypes_LCT.h"
```

在 sf_b_Outputs_wrapper 函数中输入：

```
myFcn(u0, u1, y0);
```

点击 S Function builder 顶部 build 下拉按钮确认 Genarete Warpper TLC 已勾选，单击 Build 生成 S Function 和 TLC。

此时 S Function Builder 的界面如下图所示：



The screenshot shows the Simulink S-Function Builder interface. At the top, there's a toolbar with 'Comment //', 'Insert Ports' (highlighted with a red box), 'Indent', 'Build', and 'BUILD'. Below the toolbar is a table titled 'Ports And Parameters' with columns: Name, Scope, Data Type, Dimension..., and Complexity. It lists three ports: u0 (input, Bus: sigStructT [1,1], real), u1 (input, Bus paramStru [1,1], real), and y0 (output, Bus: sigStructT [1,1], real). To the right of the table is a code editor window. The code is written in C and includes sections for header includes, extern declarations, start and terminate functions, and a main loop. A red box highlights the '#include "ex_myTypes_LCT.h"' line. Another red box highlights the 'Libraries' section in the table below the code editor.

```

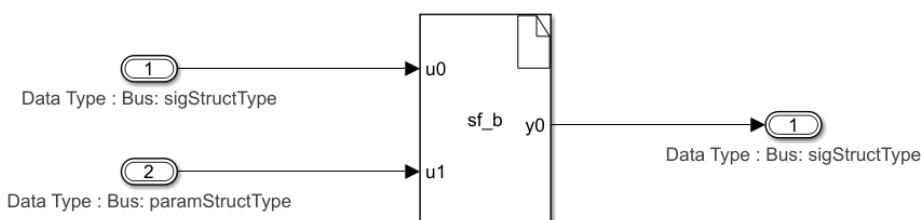
1 /* Includes_BEGIN */
2 #include <math.h>
3 #include "ex_myTypes_LCT.h"
4 /* Includes_END */

5
6 /* Externs_BEGIN */
7 /* extern double func(double a); */
8 /* Externs_END */
9
10 void sf_b_Start_wrapper(void)
11 {
12 /* Start_BEGIN */
13 /*
14 * Custom Start code goes here.
15 */
16 /* Start-END */
17 }
18
19 void sf_b_Outputs_wrapper(const sigStructType *u0,
20                           const paramStructType *u1,
21                           sigStructType *y0)
22 {
23 /* Output-BEGIN */
24 /* This sample sets the output equal to the input
25 y0[0] = u0[0];
26 For complex signals use: y0[0].re = u0[0].re;
27 y0[0].im = u0[0].im;
28 y1[0].re = u1[0].re;
29 y1[0].im = u1[0].im;
30 */
31 myFcn(u0, u1, y0);
32 /* Output-END */
33 }
34
35 void sf_b_Terminate_wrapper(void)
36 {
37 /* Terminate REGTN */
38 }

```

三、构建模型

在 Simuink 模块库中，向已添加 S Function Builder 模型加入两个输入一个输出。并双击打开一个输入模块，单击 Signal Attributes，在 Data Type 中输入 Bus: sigStructType，并在输出模块做相同的数据类型配置，单击另外一个输入，将数据类型设置为 Bus: paramStructType。将输入输出与 S Function Builder 相连如下图所示



由于 S Function Builder 无法将参数配置成 bus，此时参数只能当输入，但 Legacy_code 可以生成 bus 参数的 S 函数。

C Mex S Function

S Function 是 Simulink 提供的自定义模块。Simulink 库虽然类型非常丰富但无法保证满足每一位用户。从级别上 S 函数分为 level 1 和 2，其中 level 1 主要用于仿真，类型单一，输入和输出只有一个，其功能非常有限，但其结构简单。而 level 2 可支持多输入多输出，可通过 TLC 进行定制代码生成。从类型上 S 函数可分为 Matlab 脚本、C、C++和 Fortran。其中 Matlab 写的 S 函数无法输出 Function call，但其结构简单，C Mex S function 功能非常强大，有丰富的可配置选项，例如支持连续、离散的状态方程，可输出 function call，支持多采样率等等。其他的可见官方帮助文档。C Mex S Function 将举两个例子，一个是结构体（C Caller Block），另一个是结构体成员是指针（C Function Block）。

C Mex S Function 的 C 文件均来官方提供的模板，文件名为 sfuntmpl_basic.c，该文件可从 S 函数的例子获取，或者直接去 matlab 安装目录下搜索。

一、C Function 嵌的 C 文件

使用 S 函数去嵌 C Function 调的 C 文件。

1、模型配置

使用前言提供的脚本去创建空白模型文件。之后使用 Ctrl+E 打开模型设置，取消 Code Generation->Custom Code->Use the same custom code setting as Simulation Target 勾选，并在 Code Generation->Custom Code->Additional build information->Source files 中输入 data_array.c。可以用以下代码实现相同的功能：

```
set_param(bdroot, 'RTWUseSimCustomCode','off')
set_param(bdroot, 'CustomSource', 'data_array.c')
```

2、C Mex S Function

官方提供的 C 类型的 S Function 模板中只用到了以下三个函数：初始化函数（mdlInitializeSizes），初始化采样时间（mdlInitializeSampleTimes）和输出函数（mdlOutputs）。对 sfuntmpl_basic.c 的修改如下：

第一，找到 #define S_FUNCTION_NAME sfuntmpl_basic 此行，并修改如下并保存为 Sfunc_Call_c_pointer.c：

```
#define S_FUNCTION_NAME Sfunc_Call_c_pointer
```

第二，找到#include "simstruc.h"此行，并在下一行添加：

```
#include "data_array.h"
```

第三，修改 mdlInitializeSizes 函数，修改如下：

```
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return;
    }
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOperatingPointCompliance(S,
        USE_DEFAULT_OPERATING_POINT);
    ssSetOptions(S, 0);
}
```

第四，修改 mdlInitializeSampleTimes 函数，修改如下：

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

第五，修改 mdlOutputs 函数，修改如下：

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T i;
    DataArray data;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S, 0);
    real_T *y = ssGetOutputPortSignal(S, 0);
    data.length = ssGetOutputPortWidth(S, 0);
    data.pData = uPtrs[0];
    *y = data_sum(data);
}
```

最后编译成可执行文件，编译命令如下：

```
mex Sfunc_Call_c_pointer.c data_array.c
```

编译成功之后即可完成用于 S 函数。

3、TLC 编写

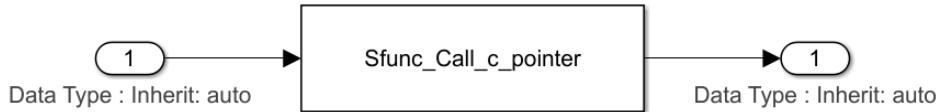
直接给出 TLC 文件如下：

```
%implements Sfunc_Call_c_pointer "C"
%function BlockTypeSetup (block, system) void
    %<LibAddToCommonIncludes("data_array.h")>
%endfunction

%function Outputs (block, system) Output
    %assign u = LibBlockInputSignalAddr(0, "", "", 0)
    %assign y = LibBlockOutputSignal(0, "", "", 0)
    DataArray data;
    data.length = %<LibBlockInputSignalWidth(0)>;
    data.pData = %<u>;
    %<y> = data_sum(data);
%endfunction
```

4、构建模型

在 Simuink 模块库中，向已配置好的空白模型加入一个输入一个输出。将 S-Function 模块加入到仿真模型之中，双击打开模块并在 S-Function name 中填入 Sfunc_Call_c_pointer，最后单击 OK 即可。连接输入、输出和 S-Function，其布局如下：



由于 TLC 的存在，此模型即可仿真也可代码生成。

二、C Caller 嵌的 C 文件

使用 S 函数去嵌 C Caller 调的 C 文件。

1、模型配置

使用前言提供的脚本去创建空白模型文件。之后使用 Ctrl+E 打开模型设置，取消 Code Generation->Custom Code->Use the same custom code setting as Simulation Target 勾选，并在 Code Generation->Custom Code->Additional build information->Source files 中输入 ex_mySrc_LCT.c。可以用以下代码实现相同的功能：

```
set_param(bdroot, 'RTWUseSimCustomCode','off')
set_param(bdroot, 'CustomSource', 'ex_mySrc_LCT.c')
```

2、C Mex S Function

官方提供的 C 类型的 S Function 模板中只用到了以下三个函数：初始化函数（mdlInitializeSizes），初始化采样时间（mdlInitializeSampleTimes）和输出函数（mdlOutputs）。对 sfuntmpl_basic.c 的修改如下：

第一，找到 `#define S_FUNCTION_NAME sfuntmpl_basic` 此行，并修改如下并保存为 Sfunc_Call_c.c：

```
#define S_FUNCTION_NAME Sfunc_Call_c
```

第二，找到`#include "simstruc.h"`此行，并在下一行添加：

```
#include "ex_myTypes_LCT.h"
```

第三，修改 `mdlInitializeSizes` 函数，修改如下：

```
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return;
    }
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 2)) return;
    DTypeId dataTypeIdReg;
    ssRegisterTypeFromNamedObject(S, "paramStructType",
&dataTypeIdReg);
    if (dataTypeIdReg == INVALID_DTYPE_ID) return;
    ssSetInputPortDataType(S, 0, dataTypeIdReg);
    ssSetBusInputAsStruct(S, 0, 1);
    ssSetInputPortComplexSignal(S, 0, COMPLEX_NO);
    ssRegisterTypeFromNamedObject(S, "sigStructType",
&dataTypeIdReg);
    if (dataTypeIdReg == INVALID_DTYPE_ID) return;
    ssSetInputPortDataType(S, 1, dataTypeIdReg);
    ssSetBusInputAsStruct(S, 1, 1);
    ssSetInputPortComplexSignal(S, 1, COMPLEX_NO);
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortWidth(S, 1, 1);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortDataType(S, 0, dataTypeIdReg);
    ssSetBusObjectName(S, 0, (void *)"sigStructType");
    ssSetBusOutputAsStruct(S, 0, 1);
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOperatingPointCompliance(S,
USE_DEFAULT_OPERATING_POINT);
    ssSetOptions(S, 0);
}
```

第四，修改 mdlInitializeSampleTimes 函数，修改如下：

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

第五，修改 mdlOutputs 函数，修改如下：

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    paramStructType *p = (paramStructType*)
    ssGetInputPortSignal(S,0);
    sigStructType *u = (sigStructType*)
    ssGetInputPortSignal(S,1);
    sigStructType      *y = ssGetOutputPortSignal(S,0);
    myFcn(u, p, y);
}
```

最后编译成可执行文件，编译命令如下：

```
mex Sfunc_Call_.c data_array.c
```

编译成功之后即可完成用于 S 函数。

3、TLC 编写

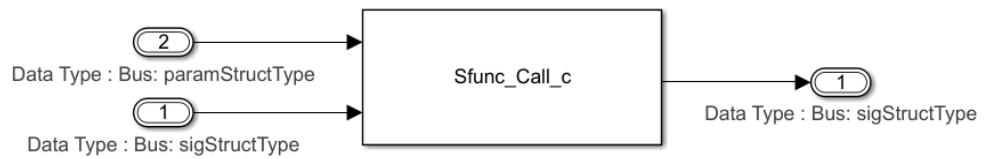
直接给出 TLC 文件如下：

```
%implements Sfunc_Call_c "C"
%function BlockTypeSetup (block, system) void
    %<LibAddToCommonIncludes("ex_myTypes_LCT.h")>
%endfunction
%function Outputs (block, system) Output
    %assign p = LibBlockInputSignalAddr(0, "", "", 0)
    %assign u = LibBlockInputSignalAddr(1, "", "", 0)
    %assign y = LibBlockOutputSignalAddr(0, "", "", 0)
    myFcn(%<u>, %<p>, %<y>);
%endfunction
```

4、构建模型

在 Simuink 模块库中，向已配置好的空白模型加入两个输入一个输出。双击打开一个输入模块，单击 Signal Attributes，在 Data Type 中输入 Bus: sigStructType，并在输出模块做相同的数据类型配置，单击另外一个输入，将数据类型设置为 Bus: paramStructType。将 S-Funciton 模块加入到仿真模型之中，双击打开模块并在 S-Function name 中填入 Sfunc_Call_c，最后单击 OK 即可。连接输入、输出和 S-Function，需要注意的是：S 函数的第一个输入是参数，第二个输入是输入，

不能反接。其布局如下：



结 语

分别使用 C Function、C Caller、Legacy_code、Stateflow、MATLAB Function、S Function Builder 和 C Mex S Function 嵌已有的 C 文件，各种方式有一个共同点都需要在模型设置中添加自定义代码。

其中 C Function 嵌入的结构体带指针，而 C Caller 无法将其嵌入，主要是因为无法自动生成代替指针的 Bus，C Function 比 C Caller 功能更加强大，但 C Caller 比 C Function 使用简单，C Function 需要手写部分代码。Legacy_code 借助简单的几行代码即可实现 S 函数和 TLC 的生成，可支持结构作为参数，非常的强大，但生成的 S 函数和 TLC 复杂臃肿，可能是出于安全性考虑。Stateflow 支持 C 语言，其调用 C 也非常的容易。MATLAB Function 也调用 C 语言非常容易，但非常有限，对函数原型要求较高。S Function Builder 和 Legacy_code 非常的像，都是自动生成 S 函数和 TLC，但不支持参数作为结构体和多维输入输出。最后是 C Mex S Function，它功能非常的强大，自定义非常的高，但需要有一定的 C 语言基础，并且熟悉 S 函数的 API，如果需要生成代码还需要编写 TLC。

以上多提供的多种方法各有优缺点，选择合适的方式嵌已有的 C 文件。