

```
Would you like mex to locate installed compilers [y]/n? y
```

输入 y 并单击回车, 得

```
Select a compiler:  
[1] Lcc - win32 C 2.4.1 in D:\MATLAB\R2013b\sys\lcc
```

```
[0] None
```

```
Compiler: 1
```

```
Please verify your choices:
```

```
Compiler: Lcc - win32 C 2.4.1
```

```
Location: D:\MATLAB\R2013b\sys\lcc
```

此时再输入 y 确认选择的是 Lcc 编译器:

```
Are these correct [y]/n? y
```

```
Trying to update options file: C:\Users\Administrator\AppData\Roaming\MathWorks\MATLAB\R2013b\mexopts.bat
```

```
From template: D:\MATLAB\R2013b\bin\win32\mexopts\lccopts.bat
```

```
Done . . .
```

至此, 编译器配置完毕。则需要在 Command Window 中输入 `mex timestwo.c` 即可完成 c 文件的编译, 生成 `mexw32/mexw64` 文件。建立模型进行仿真, 可观察到输出数据是输入数据的两倍。

### 3. C Mex S 函数的应用

使用 C Mex S 函数编写一个简单的滤波器, 并建立模型对带有噪声的正弦波进行滤波仿真。滤波器的数学模型表述如下:

$$Y(t) = (U(t) - Y(t-1)) \times L_c + Y(t-1)$$

$U(t)$  表示当前采样时刻的输入,  $Y(t)$  表示当前采样时刻输出,  $Y(t-1)$  表示上一个采样时刻的输出值,  $L_c$  表示一阶滤波器的滤波系数。编写 S 函数的 C 代码之前, 需要设计好模块的外观和属性。整个模块采用浮点数据运算, 输入输出均为一个端口, 端口宽度具有相同维数, 此处初始化为动态维数。由表达式可知, 输入端口是具有直接馈入的, 因为输出直接跟输入相关, 且同一时刻进行更新。滤波器系数  $L_c$  作为整个模型的参数, 封装在模块的对话框上(封装方法请参考第 11 章“模块的封装”), 封装后模块图标及参数对话框如图 10.5-15 所示。模块有一个参数需传递到 C Mex S 函数内部去。在 C Mex S 函数中需要通过宏来获取参数值, 由于此参数的变量名为 `g_coef`, 故定义宏为:

```
#define COEF(S) mxGetScalar(ssGetSFnParam(S,0))
```

`ssGetSFnParam` 函数的第 2 个参数为参数的索引号, 从 0 开始, 返回的是指向这个指定参数的指针。`mxGetScalar` 函数则从这个指针指向的对象获取数据。

计算输出时需要使用到上一个采样时刻的输出值, 则在内部设置状态变量 `Dwork` 向量保存这个值, 其维数跟输入/输出一样为动态维数, 初始化值为 0。`Dwork` 变量在 `mdlOutput` 子方法中保存每个时刻的输出值, 以便下个采样时刻计算输出值时使用。由于此系统为离散系统, 故创建一个离散状态变量或者 `Dwork` 变量即可。输入需要在输出中使用, 故输入端口是直接馈入的。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

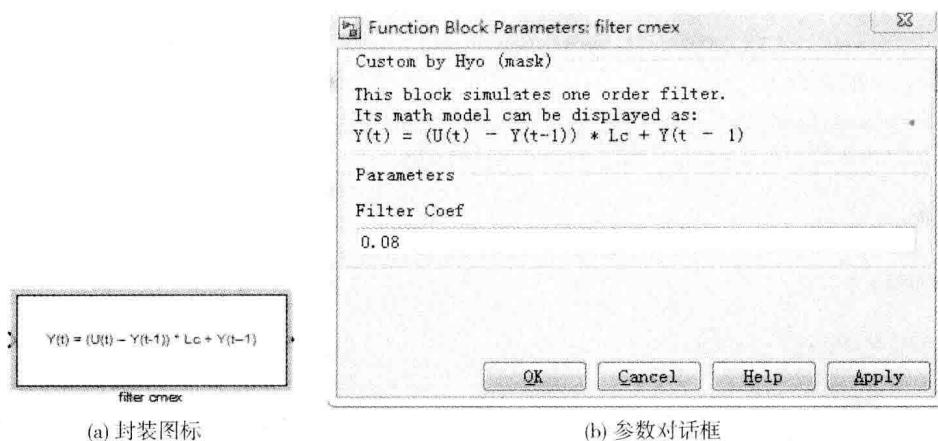


图 10.5-15 滤波器模块封装

根据上述要求,编写一阶滤波器的 C Mex S 函数代码如下:

```
# define S_FUNCTION_NAME sfun_c_filter
# define S_FUNCTION_LEVEL 2

#include "simstruc.h"

#define COEF_IDX 0
#define COEF(S) mxGetScalar(ssGetSFcnParam(S,COEF_IDX))

/* Function: mdlInitializeSizes
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* Abstract:
*   Setup sizes of the various vectors.
*/
static void mdlInitializeSizes(SimStruct * S)
{
    ssSetNumSFcnParams(S, 1);
    if(ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

    if(! ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if(! ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);

    ssSetNumDWork(S, 1);
    ssSetDWorkWidth(S, 0, DYNAMICALLY_SIZED);

    ssSetNumSampleTimes(S, 1);

    /* specify the sim state compliance to be same as a built-in block */
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);
}
```

```

ssSetOptions(S,
    SS_OPTION_WORKS_WITH_CODE_REUSE |
    SS_OPTION_EXCEPTION_FREE_CODE |
    SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}

/* Function: mdlInitializeSampleTimes
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* Abstract:
*   Specify that we inherit our sample time from the driving block.
*/
static void mdlInitializeSampleTimes(SimStruct * S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}

#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* Abstract:
*   Initialize DWork to zero.
*/
static void mdlInitializeConditions(SimStruct * S)
{
    real_T * x = (real_T *) ssGetDWork(S,0);
    x[0] = 0.0;
}
/* Function: mdlOutputs
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* Abstract:
*   y = (u - x[0]) * Lc + x[0]
*/
static void mdlOutputs(SimStruct * S, int_T tid)
{
    int_T i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T y = ssGetOutputPortRealSignal(S,0);
    int_T width = ssGetOutputPortWidth(S,0);
    real_T * x = (real_T *) ssGetDWork(S,0);
    real_T Lc = COEF(S);
    /* calculate the current output according the math equation */
    for(i = 0; i < width; i++)
    {
        y[i] = (*uPtrs[i] - x[i]) * Lc + x[i];
    }
    /* save the current output as the DWork Vector */
    for(i = 0; i < width; i++)
    {
        x[i] = y[i];
    }
}

```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```

    }

}

/* Function: mdlTerminate
=====
* Abstract:
*   No termination needed, but we are required to have this routine.
*/
static void mdlTerminate(SimStruct * S)
{
}

#endif /* MATLAB_MEX_FILE */ /* Is this file being compiled as a MEX - file? */
#include "simulink.c" /* MEX - file interface mechanism */
#ifndef CG_SFUN_H
#include "cg_sfun.h" /* Code generation registration function */
#endif

```

从 Simulink Browser 中拖出一个 S-Function 模块,按照图 10.5-15 封装之后,再将 S 函数名和参数 g\_coef 填写到模块参数对话框中,如图 10.5-16 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

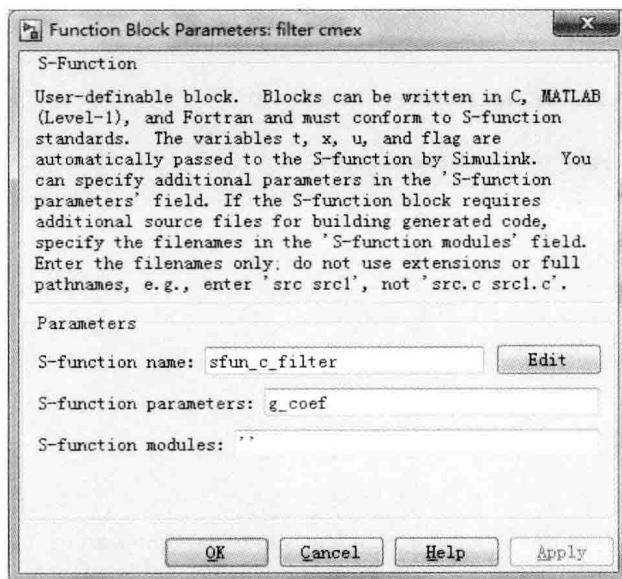


图 10.5-16 filter cmex 的 S 函数对话框

模型的输入采用带有白噪声的正弦波,输入和输出接入 Scope,既显示带有白噪声的正弦波,又显示滤波之后的波形,模型如图 10.5-17 所示。

模型的解算器采用固定步长解算器,discrete 解算方法,其步长设置为 0.01,Simulink 自带模块都采用默认设置,但是各模块的采样时间 Sample time 设置为 -1,继承模型解算器的步长作为采样间隔,filter cmex 模块的 Filter coef 输入为 0.005,仿真长度设为 30 s,运行仿真之后,得到滤波前后波形如图 10.5-18 所示。

简单的一阶滤波器对于混杂了大量白噪声的正弦波起到了相当的滤波效果。

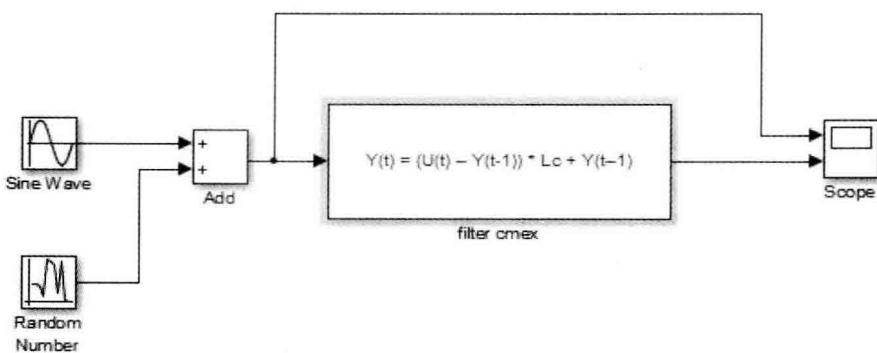


图 10.5-17 白噪声滤波模型

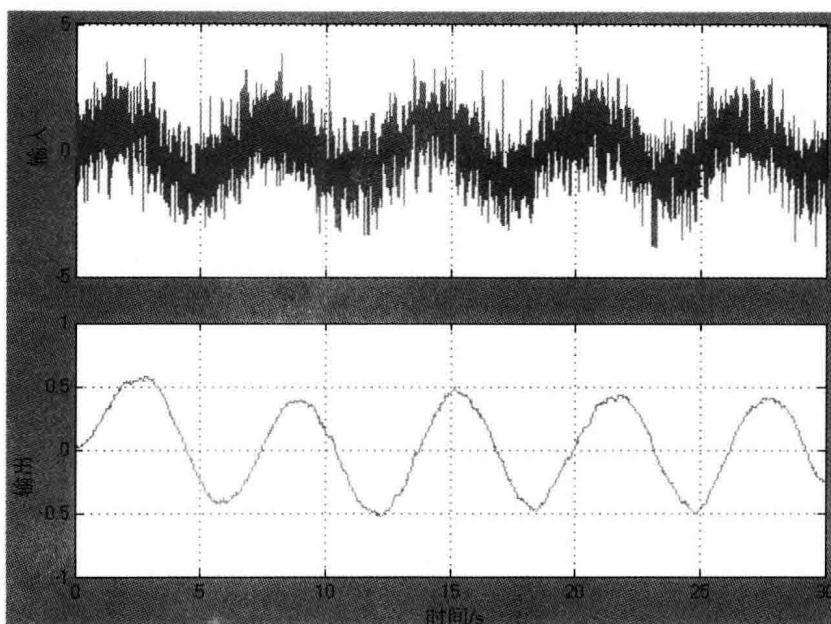


图 10.5-18 滤波前后波形比较

#### 4. C Mex S 函数的自动生成

为了方便不熟悉 C 语言的 Simulink 用户能够方便快捷地通过简单配置得到可以使用的 C Mex S 函数,也为了有大量既有 C 代码用户在转用 Simulink 时,能够方便地将算法成果导入 Simulink 中,让这些成果既能仿真又能够生成代码,Simulink 提供了 2 个工具帮助用户快速生成 C Mex S 函数。一个是根据用户的配置自动生成 C Mex S 函数(同时也可以生成 TLC 文件)的 Simulink 模块,称为 S-Function Builder;另一个是能够将既存的或者用户自定义的 C 代码打包生成内联的 C mex S 函数,并且能够生成嵌入式 C 代码应用于嵌入式目标芯片的工具,名为 Legacy Code Tool。后者在开发硬件驱动模块也可以使用。

所谓内联 S 函数(Inline S-Function)就是指拥有同名的 TLC 文件,支持代码生成的 S 函数。反之,Noninline S-Function 就是指不具有 TLC 文件,不支持代码生成的 S 函数。内联 S 函数有两种内联方式,一种是 Full inlined(完全内联),在 TLC 的 Output 子方法中实现具体的算法,明确给出输入/输出的关系;另一种是 Wapper inlined(包装内联),在 TLC 的 Output

子方法中不是实现具体算法代码,而是规定输入/输出端口变量如何调用已经存在的 C 代码。接下来看下 S-Function Builder 与 Legacy Code Tool 是如何帮助用户生成 S 函数 C 文件的。

### (1) S-Function Builder

S-Function Builder 以模块的形式提供,读者可以从 Simulink Library 中找到它,通过 GUI 配置的方式规定 S 函数初始化属性以及自动生成 C Mex 文件。该模块外观如图 10.5-19 所示。

双击该模块可以打开其 GUI 对话框,如图 10.5-20 所示,包含 3 个部分:

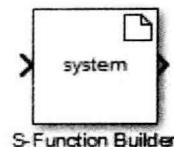


图 10.5-19 S-Function Builder 外观

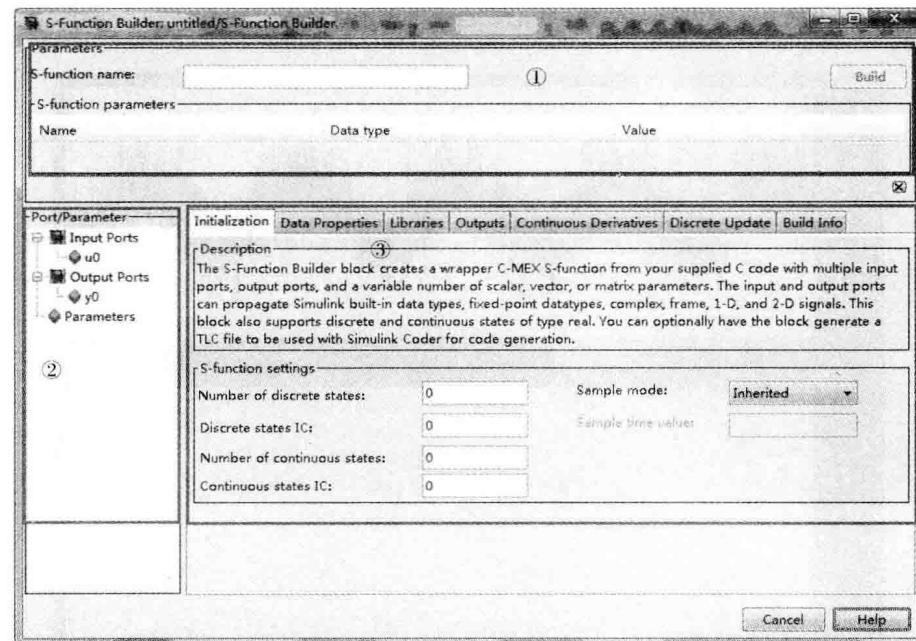


图 10.5-20 S-Function Builder 首页

- ① S 函数名称及参数列表显示部分。
- ② S 函数端口及参数的结构树形图。
- ③ 7 个与 S 函数各个子方法对应的配置页面。

用户可以在①中给定 S 函数的名字,并显示出用户在③的 Data Property 页面下添加的参数属性列表。图 10.5-20 显示了默认时 S-Function Builder 设定的单输入单输出无参数情况下的端口结构树形图。③则与 S 函数的子方法紧密对应,对应关系如表 10.5-15 所列。

表 10.5-15 S-Function Builder 的配置页面与 S 函数子方法的对应关系

| 配置页面            | 对应的 S 函数子方法  |
|-----------------|--|
| Initialization  | mdlInitializeSizes 中状态变量和 mdlInitializeSampleTimes 中采样时间的设定及 mdlInitializeConditions 中状态变量初始值的设定 |
| Data Properties | mdlInitializeSizes 中输入输出端口和参数的个数、数据类型  |
| Libraries       | 无子方法对应,此处填入编译用户代码时所需要的头文件  |

续表 10.5-15

|                       |   |
|-----------------------|---|
| 配置页面                  | 对应的 S 函数子方法   |
| Outputs               | mdlOutputs 子方法  |
| Continous Derivatives | mdlDerivatives 子方法  |
| Discrete Update       | mdlUpdate 子方法   |
| Build Info            | 无子方法对应,用来显示编译信息,并提供给用户功能选项,生成 C 代码时可以选择是否生成 TLC 文件,是否同时编译 C 文件为 mex 文件等 |

Initialization 页面用户配置如图 10.5-21 所示。

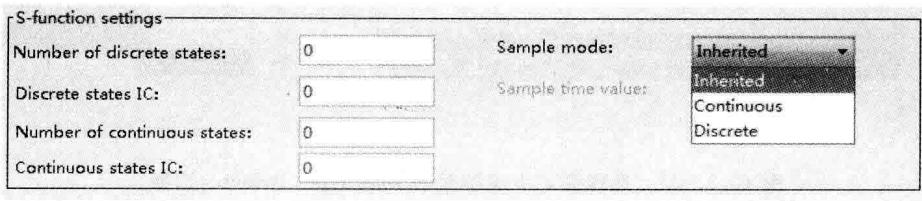


图 10.5-21 S-Function settings 首页

Number of discrete states/Number of continuous states 是指 S 函数中需要设置几个离散/连续状态变量,这些状态变量的初始值通过 Discrete states IC/Continuous states IC 配置。当多个状态变量存在时,可以使用向量形式,如给出 3 个离散/连续状态变量的初始值[0 0 0],元素个数必须与初始化的状态变量(离散或连续)个数相同。右侧的 Sample Mode 提供了 3 种采样时间:

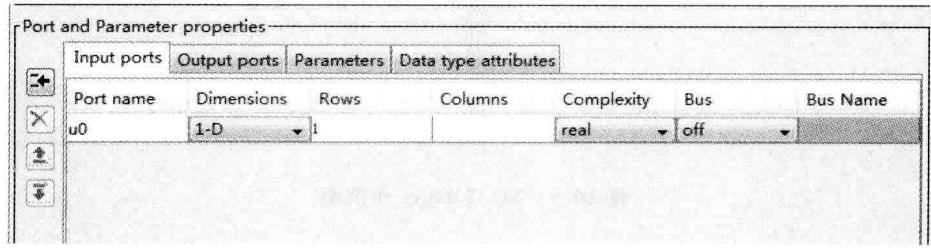
Inherited:S 函数继承输入端口的采样时间。

Continuous:S 函数采用连续采样时间,在每个采样步长更新输出值。

Discrete:S 函数采用离散采样时间,Sample Time Value 设置采样时间间隔。

Sample Time Value 仅当 Sample Mode 选择 Discrete 时才有效,表示 S 函数模块的采样时间。

Data Properties 页面中,将 S 函数相关的数据量和信息分散在 4 个子页面上进行配置,如图 10.5-22 所示。



若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 10.5-22 Port and Parameter properties

图左侧竖栏提供了 4 个按钮,为端口和参数的添加、删除、上移和下移操作按钮。增加一个变量时按第一个按钮进行追加;进行删除或上下移行操作时,首先要先选中操作对象行。

Input ports: 输入端口的名称、维数、数据行列、实虚性和总线类型设置。

Output ports: 输出端口的名称、维数、数据行列、实虚性和总线类型设置。

Parameter: 增加参数、并设置其名称、数据类型和实虚性。

注: 对于上述 3 个子页面, Dimensions 下拉菜单支持 1-D 和 2-D 数据, Rows 中输入数据的行数, 为 -1 时表示行数是动态的。Columns 列仅在 Dimensions 选择为 2-D 时才有效。

Data type attributes: 对输入/输出端口的数据类型进行设定, 包括内建类型和固定点类型。选择固定点类型时, 可对数据类型的详细信息进行配置, 包括数据字长和小数位等, 如图 10.5-23 所示。

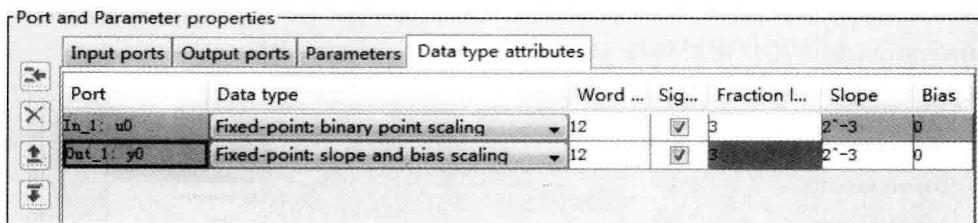


图 10.5-23 选择固定点类型时的 Data type attributes 选择

Library 页面中主要用于添加头文件、外部源文件、用户自定义代码相关文件和函数声明, 如图 10.5-24 所示。

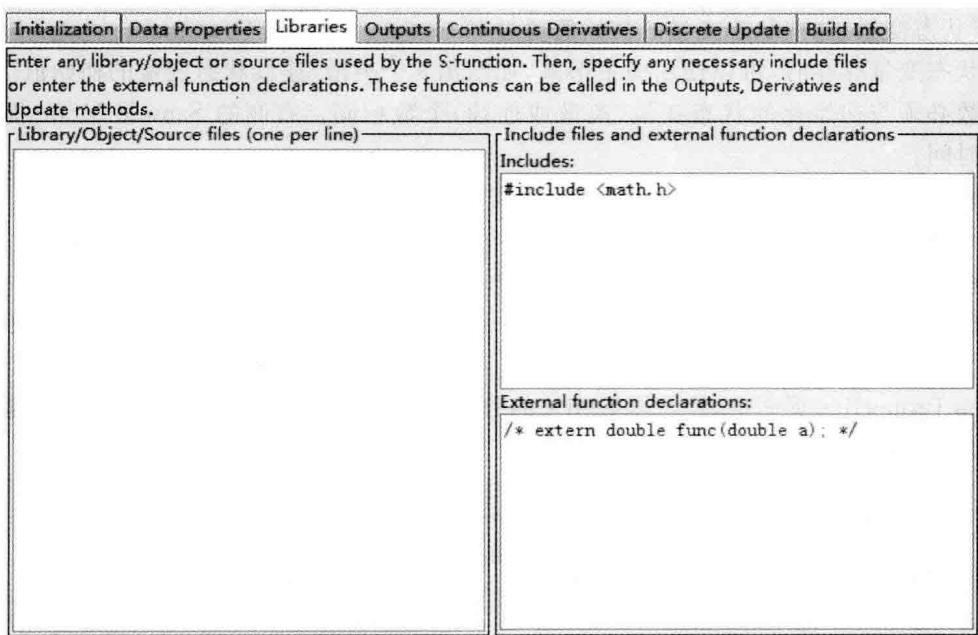


图 10.5-24 Library 子页面

Library/Object/Source files(one per line): 当 S 函数的子方法配置中使用了外部的库、目标文件或源文件时, 需要将库、目标文件和源文件的全路径写在此对话框中。如果这些文件存放在当前路径之下, 只写入文件名即可, 如:

customfunctions.lib、userfunctions.obj 和 freesource.c。

用户还可以在此处通过关键字 LIB\_PATH、INC\_PATH 和 SRC\_PATH 添加搜索路径(每行一个), 在关键字之后给出一个文件名(格式: 关键字 文件路径), S-Function Builder 会

自动选择到关键字后的路径中搜索。多个文件的格式需要分行书写,注意不要使用引号将文件引用起来。合法使用如下:

```
SRC_PATH d:\externalsource
LIB_PATH $ MATLABROOT\customobjs
INC_PATH c:\customfolder
customfunctions.lib
```

关键字 LIB\_PATH 既可以用来追加搜索库文件的路径,也可以用来追加搜索目标文件的路径。

**Includes:** 当用户自定义代码出现在 S-Function Builder 的任何配置中时,所涉及的头文件、函数声明、变量和宏定义等都应该在此使用 #include 语句进行包含。如果包含的头文件是标准 C 语言库,使用尖括号,如 #include <math.h>; 如果包含的头文件是用户自定义代码,使用双引号,如 #include "my\_device.h"。特别地,当被包含的头文件所存放的路径不在当前路径时,需要在 Library/Object/Source files(one per line)对话框中使用 INC\_PATH 来指定这个头文件所在的文件路径。

**External functions Declaration:** 当 S-Function Builder 中需要在状态变量和 Outputs 子方法中调用某些外部函数计算,并且这些计算所调用的函数既不是 Simulink 自带的,也不被 Includes 中列出的头文件所包含时,在此处进行函数声明,如; extern double func(double a);

Outputs 子页面中输入的内容就是 S 函数 Outputs 子方法函数体内容,C Mex S 函数的输入/输出端口数据获部分代码可以省略,直接使用输入/输出端口号和下标索引号即可作为输入/输出变量,并规定它们之间的关系,如图 10.5-25 所示。

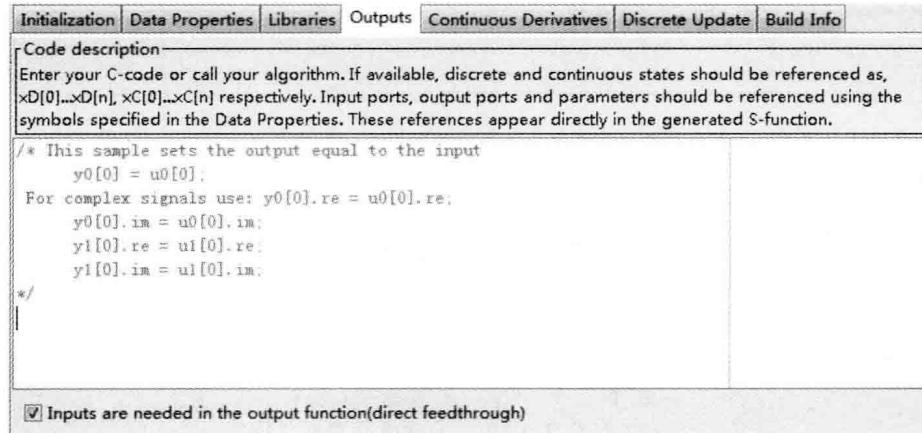


图 10.5-25 Outputs 子页面

输入的直接馈入在此页面左下角的 checkbox 里面进行设置,提示字符串为 Inputs are needed in the output function(direct feedthrough),勾选之后,即可在对话框中使用输入端口变量名。

用户在图 10.5-25 所示对话框中编写的代码将在编译时打包为一个函数 sfun\_Outputs\_wrapper,插入到函数体中,再在 C Mex S 函数的 mdlOutputs 中调用 sfun\_Outputs\_wrapper 函数。sfun 为用户填入的 S-Function Builder 的 S-Function 名。

Continuous Derivatives 子页面的对话框如图 10.5-26 所示,用户可以填入计算连续状态变量的代码,连续状态变量按照维数索引号,可以使用  $xC[0]$ , $xC[1]$  或者  $dx[0]$ , $dx[1]$  表示,这些变量必须是 double 类型。输入/输出端口号和参数必须使用在 Data Properties 中定义过的端口号名

和参数变量名。此对话框中所填入的内容被打包为一个 sfun\_Derivatives\_wrapper 函数,再在 mdlDerivatives 中调用。sfun 为用户填入的 S-Function Builder 的 S-Function name。

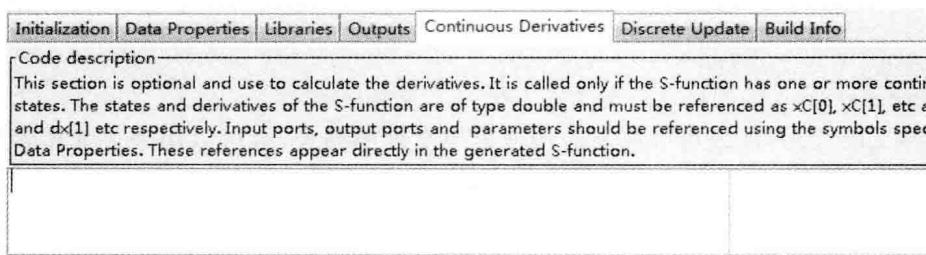


图 10.5-26 Continuous Derivatives 子页面

Discrete Update 子页面如图 10.5-27 所示。用于输入 mdlUpdate 子方法对应的内容,无须使用宏函数获取输入/输出端口、参数和离散状态变量的值。离散状态变量使用 xD[0],xD[1] 等引用即可。输入/输出端口号和参数必须使用在 Data Properties 中定义过的端口名和参数变量名。此对话框中所填写的代码将被打包为一个函数 sfun\_Update\_wrapper,并在 mdlUpdates 子方法中调用。sfun 为用户填入的 S-Function Builder 的 S-Function name。

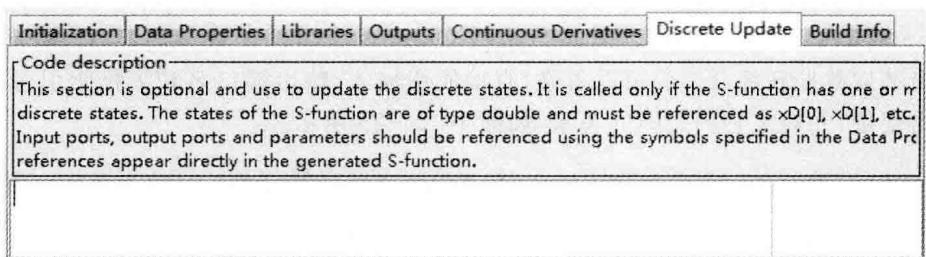


图 10.5-27 Discrete Update 子页面

sfun\_Outputs\_wrapper, sfun\_Derivatives\_wrapper, sfun\_Update\_wrapper 3 个函数的函数声明都是同样类型,返回值为空,参数则包括指向输入/输出端口、参数、状态变量的指针,以及表示输入/输出信号维数和参数维数的变量。此处以参数列表涵盖最全的 sfun\_Outputs\_wrapper 为例,打包后的 wrapper 函数如下:

```
void sfun_Outputs_wrapper (const real_T * u, /* 输入端口指针 */
                           real_T * y, /* 输出端口指针 */
                           const real_T * xD, /* 离散状态变量指针 */
                           const real_T * xC, /* 连续状态变量指针 */
                           const real_T * param0, /* 参数 0 指针 */
                           int_T p_width0 /* 参数 0 的维数 */
                           real_T * param1 /* 指向参数 1 的指针 */
                           int_t p_width1 /* 参数 1 的维数 */
                           int_T y_width, /* 输出信号维数 */
                           int_T u_width) /* 输入信号维数 */

{
    /* Outputs 对话框中填入的代码将会插入到这里 */
}
```

sfun\_Outputs\_wrapper 函数还会被生成到 TLC 文件中去。

Build Info 页面如图 10.5-28 所示,显示编译过程信息,下方的 Build options 则配置编译

生成 mex 文件的选项。

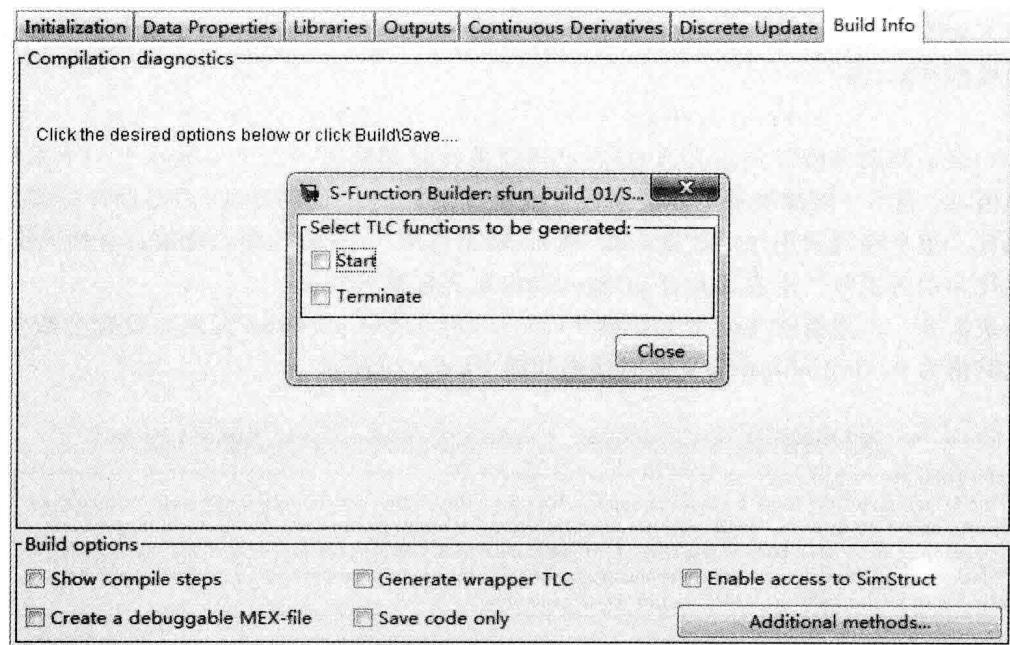


图 10.5-28 Build Info 子页面

Compilation diagnostics 框用于显示编译 C 代码和可执行文件时的编译信息, 编译过程是有错误还是成功生成, 可以在此处根据信息判断。在位于页面下方的 Build options 里面, 提供了 5 个 check-box 选项和一个按钮 Additional methods, 这个按钮按下后弹出 Select TLC functions to be generated 对话框, 提供两个 TLC 方法的选择, 当勾选之后对应的方法会生成到 TLC 代码去。其他 5 个 check-box 的作用如表 10.5-16 所列。

表 10.5-16 S-Function Builder 的 Build Info 页面 options 关系

| Build options                | 作用  |
|------------------------------|---|
| Show compile steps           | 在 Compilation diagnostics 框记录每一个编译步骤信息  |
| Generate wrapper TLC         | 生成 TLC 文件以支持代码生成或加速仿真模式   |
| Enable access to SimStruct   | 使得 Outputs、Continuous Derivatives 和 Discrete Updates 3 个页面的代码可以使用 SimStruct 类提供的宏函数 |
| Create a debuggable MEX-file | 生成 mex 文件时包含调试信息  |
| Save code only               | 只生成 C Mex S 函数代码不生成 MEX 文件  |

Enable access to SimStruct 被勾选之后, 所生成的 wrapper function 参数列表中会自动增加一个 SimStruct \* S 参数, 使得用户可以使用 SimStruct 类中的宏函数。如下面的例子中, 除了指向输出端口的指针、指向参数 a, b 的指针和二者的宽度参数外, 还包括了指向 SimStruct 类对象 S 的指针:

```
void sfun_add_Outputs_wrapper(real_T * y ,
                               const real_T * a, const int_T p_width0,
                               const real_T * b, const int_T p_width1, SimStruct * S)
{
```

若您对此书内容有任何疑问, 可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```
/* Outputs 中输入的代码 */
}
```

接下来采用 S-Function Builder 生成一阶滤波器,其表达式为如图 10.5-15 的 C Mex S 函数的模型所示,即

$$Y(t) = (U(t) - Y(t-1)) \times L_c + Y(t-1)$$

$U(t)$ 表示当前采样时刻输入, $Y(t)$ 表示当前采样时刻输出, $Y(t-1)$ 表示上一个采样时刻的输出值, $L_c$ 表示一阶滤波器的滤波系数。编写 S 函数的 C 代码之前,需要设计好模块的外观和属性。整个模块采用浮点数据运算,输入/输出均为一个端口,端口宽度具有相同维数,此处初始化为动态维数。由表达式可知,输入端口是直接嵌入的。

系统需要一个离散状态变量来存储  $Y(t-1)$ ,可在 Initialization 页面上设置个数为 1,并初始化其值为 0。Initialization 页面对话框如图 10.5-29 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

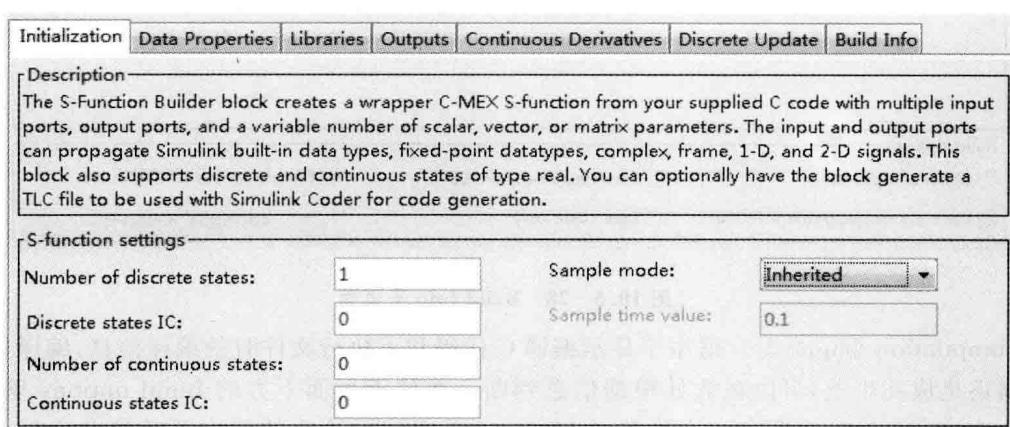


图 10.5-29 离散状态变量的设置

Data Properties 页面上设置输入输出端口属性及参数属性如图 10.5-30 所示。

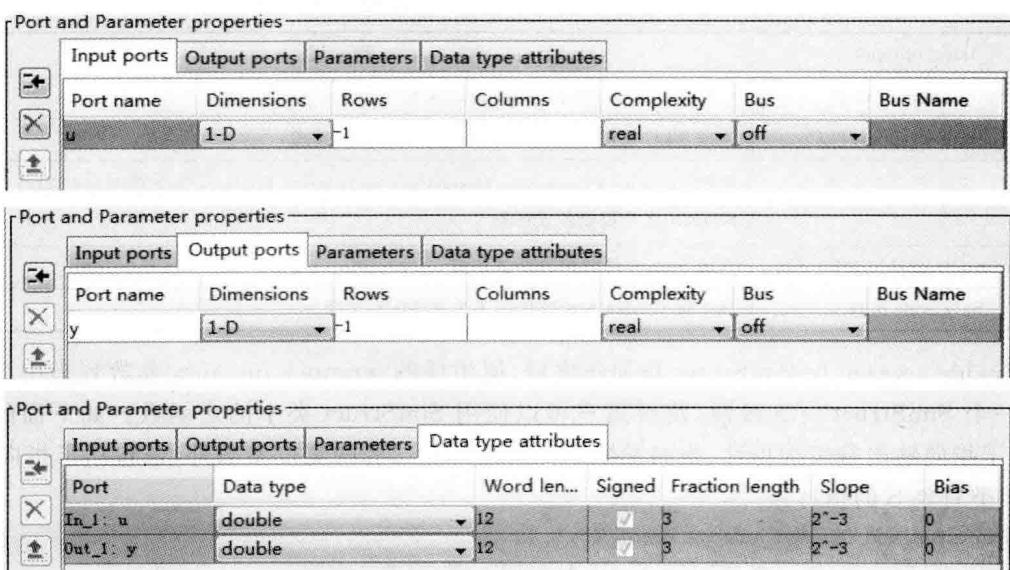


图 10.5-30 输入/输出端口的属性设置

输入/输出端口分别命名为 u、y，维数为一维的，在这里设置 Dimensions 为 1-D，Row 为 -1 表示动态维数，数据类型也均设为 double 类型。

设置一个名为 filter\_coef, double 型实数的参数，如图 10.5-31 所示。

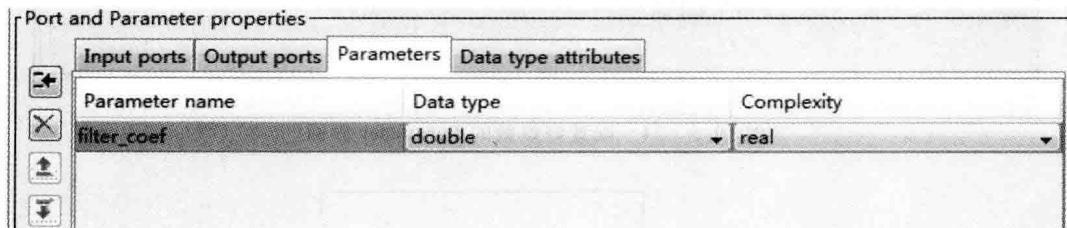


图 10.5-31 设置一个名为 filter\_coef 的参数

在 Outputs 页面输入计算模块输出的代码，上一个时刻的输出值使用离散状态变量  $xD[0]$  存储，如图 10.5-32 所示。

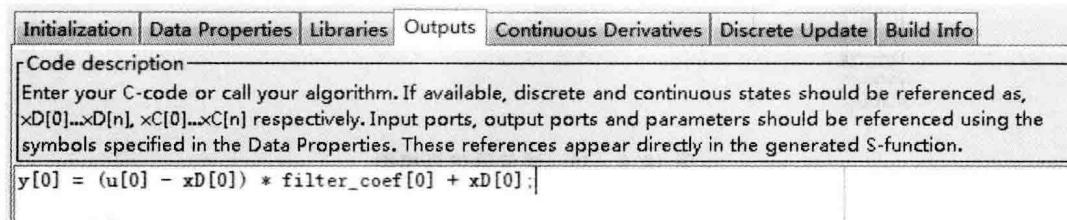


图 10.5-32 Output 页面填入的代码

在每次计算结束后，更新离散状态变量的值，需要在 Discrete Update 页面输入代码，如图 10.5-33 所示。

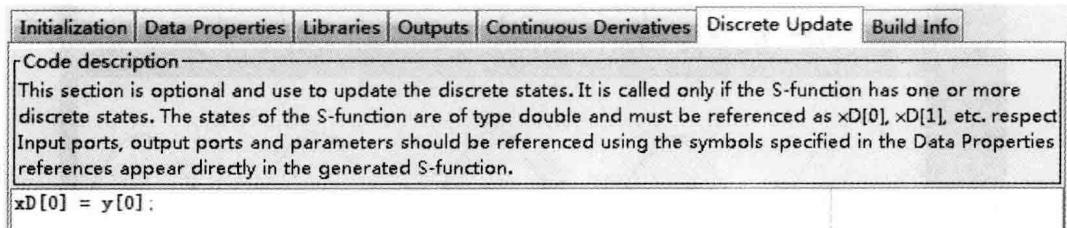


图 10.5-33 Discrete Update 页面填入的代码

参数 filter\_coef 设置为 double 类型，其值设置为 0.005。至此，需要配置的部分都已经配置完毕，可以单击如图 10.5-34 所示右上方的 Build 按钮，生成 C 文件以及 mexw32 文件（使用 32 位机器时）。

编译生成可执行 mexw32 文件后，将带有噪声的正弦波作为模块输入，建立如图 10.5-35 所示模型。滤波前后信号通过 Scope 进行观察，模型的解算器采用固定步长解算器，discrete 解算方法，其步长设置为 0.01，Simulink 自带模块都采用默认设置，采样时间 Sample time 设置为 -1，继承模型的步长作为采样间隔，filter cmex 模块的 filter\_coef 输入为 0.005（可以双击 S-Function Builder 后在 Parameter 界面里进行参数修改，不能够单独封装此模块为子系统并封装 GUI），仿真长度设为 30 s，运行仿真之后，得到滤波前后波形，如图 10.5-36 所示。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

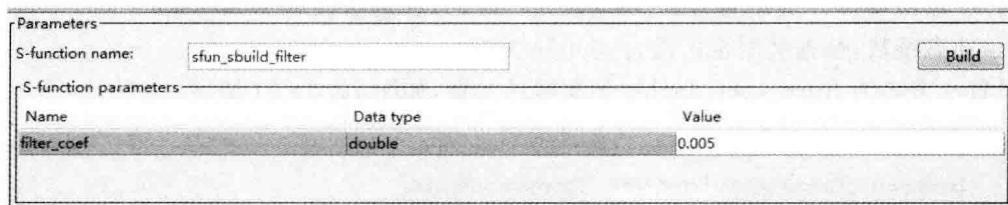


图 10.5-34 配置好参数的 Parameter 界面

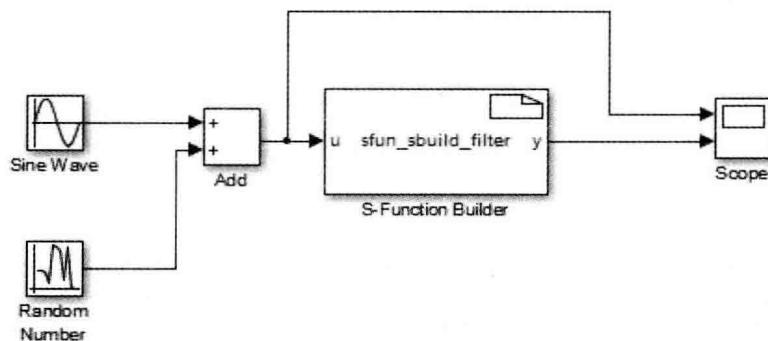


图 10.5-35 滤波器仿真模型

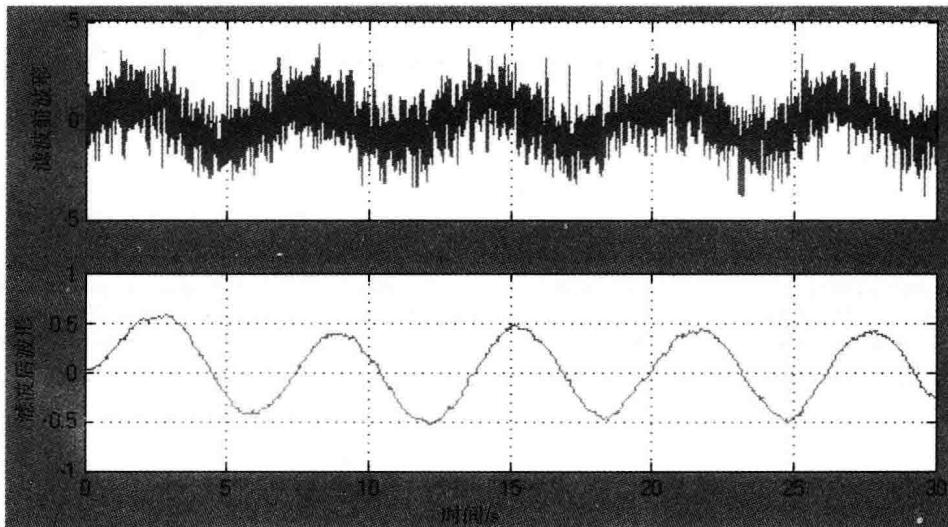


图 10.5-36 滤波器滤波前后波形

252

使用 S-Function Builder 自动生成 C Mex S 函数, 用户可以省去编写调试 C 语言的时间, 可以不用了解如何使用 SimStruct 类提供的宏函数获取输入/输出端口和参数的值等, 对于希望加速及自动生成 S 函数以及 TLC 文件却又不太熟悉 C Mex S 函数的用户来说不失为一个选择。

## (2) Legacy Code Tool

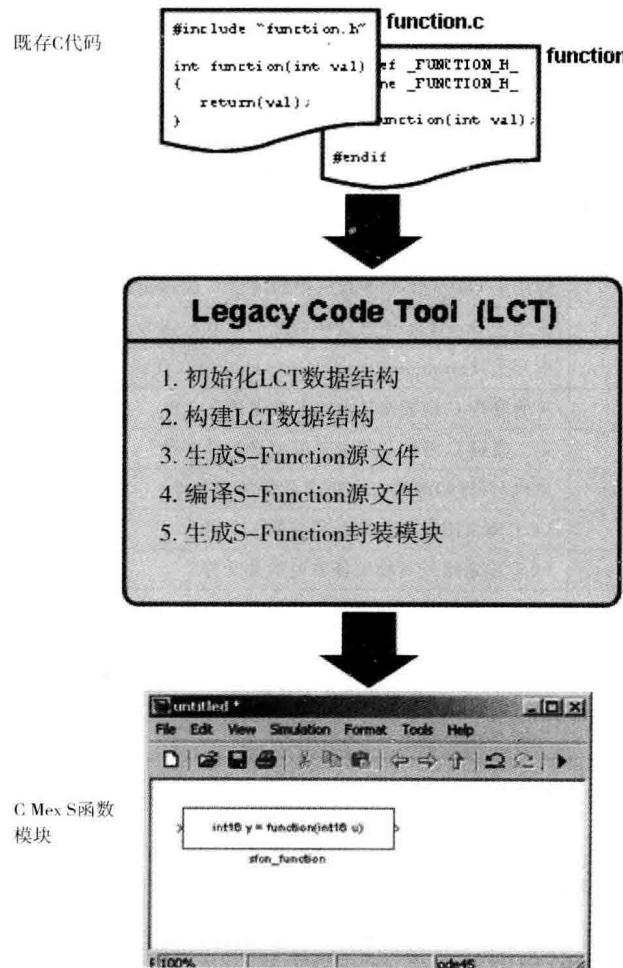
对于很多用户来说, Simulink 库中提供的现成模块不足以满足使用要求, 他们采用传统的软件开发方式, 既存了大量的 C 代码, 其中包括算法或者设备驱动的代码, 还有经过长期大量的实验获取的数据形成的查找表(Lookup Table)。为了加速开发进程, 并将既有的宝贵财

产继承下去,Simulink 提供了另外一个工具——Legacy Code Tool,能够将既存的 C/C++ 代码转换为 Simulink 模型中可以使用的 C Mex S 函数,同时也能生成 TLC 文件。Legacy Code Tool 将用户既存的算法代码插入到 C Mex S 函数的 Outputs 子方法中,用户需提供足够的信息,这些信息包括:为 MATLAB 安装一个 C 编译器,S 函数名,既存算法的函数原型,及为了编译既存 C 文件所需要的其他头文件、源文件及其存放路径。

通过核心命令 `legacy_code` 启动工具和创建对象,将上述信息一一添加到 Legacy Code Tool 的实例对象中去,就能够生成 C Mex S 函数并且可以选择生成 S 函数的模块 TLC 文件以生成 C 代码,应用于嵌入式芯片中。`legacy_code` 命令可以完成以下几件事情:

- ① 根据既有 C 代码初始化 Legacy Code Tool 的数据结构。
- ② 生成可用于仿真的 C mex S 函数。
- ③ 将生成的 S 函数编译链接为动态可执行文件(mex 文件)。
- ④ 生成一个封装起来的模块来调用 S 函数。
- ⑤ Simulink Coder 组件会生成 S 函数的模块级 TLC 文件。

图 10.5-37 从上至下展示了 Legacy Code Tool(以下简称 LCT)的使用流程:



若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

初始化 LCT 数据结构, 使用字符串 'initialize' 作为 legacy\_code 的参数初始化一个 LCT 对象, 其代码如下:

```
lct_spec = legacy_code('initialize')
```

可以通过返回值观察 LCT 对象的数据结构:

```
lct_spec =
    SFunctionName: ''
    InitializeConditionsFcnSpec: ''
    OutputFcnSpec: ''
    StartFcnSpec: ''
    TerminateFcnSpec: ''
    HeaderFiles: {}
    SourceFiles: {}
    HostLibFiles: {}
    TargetLibFiles: {}
    IncPaths: {}
    SrcPaths: {}
    LibPaths: {}
    SampleTime: 'inherited'
    Options: [1x1 struct]
```

LCT 对象的各个属性及其作用如表 10.5-17 所列。

表 10.5-17 LCT 对象的属性列表

| 属性名                         | 作用说明   |
|-----------------------------|--|
| SFunctionName               | 所生成 S 函数的名字  |
| InitializeConditionsFcnSpec | 应用于 InitializeConditions 子方法中的既存 C 代码函数原型                                  |
| OutputFcnSpec               | 应用于 OutputFcn 子方法中的既存 C 代码函数原型   |
| StartFcnSpec                | 应用于 StartFcn 子方法中的既存 C 代码函数原型  |
| TerminateFcnSpec            | 应用于 TerminateFcn 子方法中的既存 C 代码函数原型  |
| HeaderFiles                 | 声明既存 C 函数及其他需要编译的头文件   |
| SourceFiles                 | 定义既存 C 函数及其他需要编译的源文件   |
| HostLibFiles/TargetLibFiles | 主机/目标端编译 C 文件所依赖的库文件   |
| IncPaths                    | LCT 搜索路径寻找编译需要的头文件   |
| SrcPaths                    | LCT 搜索路径寻找编译需要的源文件   |
| LibPaths                    | LCT 搜索路径寻找编译需要的库和目标文件  |
| Sample Time                 | 采样时间有 3 个选项: Inherited(继承源模块的采样时间)、Parameterized(可以调节)、Fixed(用户明确指定固定采样时间) |
| Options                     | 控制 S 函数 Options 的选项  |

对于初始化过的 LCT 对象 lct\_spec, 通过域操作符, 即可访问表中的各个成员并赋值, 如规定将要生成的 S 函数名:

```
lct_spec.SFunctionName = 'sfun_lct';
```

应用于 InitializeConditions/OutputFcnSpec/StartFcnSpec/ TerminateFcnSpec 这 4 个子方法中的既存 C 代码函数原型需要以字符串的方式填入, 包括返回值类型、函数名和输入参数列表 3 个部分, 如:

```
return-spec = function-name(argument-spec)
```

在 return-spec 和 argument-spec 中, 输入变量使用 u, 输出变量使用 y, 参数使用 p 表示。

对于初始化过的 LCT 对象 lct\_spec, 可以有以下命令进行动作执行:

legacy\_code('help')——打开 LCT 工具的详细使用说明的帮助文档。

legacy\_code('sfcn\_cmex\_generate', lct\_spec)——根据 lct\_spec 生成 S 函数源文件。

legacy\_code('compile', lct\_spec)——对生成的 S 函数进行编译链接。

legacy\_code('slblock\_generate', lct\_spec, modelname)——生成一个封装模块调用生成的 S 函数, 并自动将此模块添加到名为 modelname 的模型文件里。

legacy\_code('sfcn\_tlc\_generate', lct\_spec)——生成 S 函数配套的 TLC 文件, 用于加速仿真模式或通过 Simulink 模型生成代码。

legacy\_code('rtwmakecfg\_generate', lct\_spec)——生成 rtwmakecfg.m 文件, 此文件是用于生成适用于当前 lct\_spec 对象的 makefile 的 M 脚本。

**【实例】** 将既有的正弦计算 C 代码使用 Legacy Code Tool 集成到 Simulink 模型中并实现功能仿真。

在传统嵌入式应用开发中, 产品量产化时, 需要考虑降低 MCU 芯片成本, 采用不支持浮点计算的低成本 MCU, 或者即使选用支持浮点计算的 MCU 也会尽量优化算法代码以提高计算效率节省硬件存储空间。正弦计算 sin 本身是一个精度要求较高的浮点运算, 但是应用于低成本 MCU 中必须使用固定点格式计算, 不得不考虑将正弦计算精度降低到一个合理的程度; 另外, 考虑到正弦函数本身是周期性的, 只需要根据 MCU 数据的分辨率提供一个包含一个周期内所有采样点的值的表, 那么输入为任何数值时, 根据周期性将输入转换为一个周期内的输入就可以获取其正弦值。因为查表计算速度相对于通过泰勒级数开展的公式计算正弦值要快的多。一个周期内的正弦波可以划分为 4 个区间, 如图 10.5-38 所示。

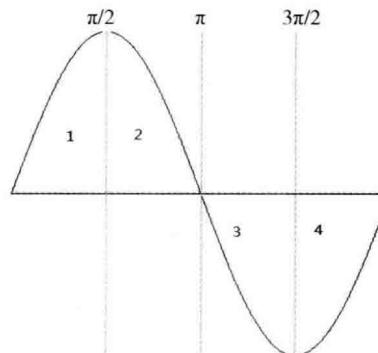


图 10.5-38 一个周期内正弦波的 4 个区间

进一步思考可发现: 不用提供一个完整周期的数据即可获取任何输入的正弦值。一个周期的正弦波可以按照  $[0, \pi/2]$ ,  $[\pi/2, \pi]$ ,  $[\pi, 3\pi/2]$ ,  $[3\pi/2, 2\pi]$  4 个区间分为 4 部分, 称为区间 1, 2, 3, 4。区间 2 与区间 1 左右对称, 区间 3 与区间 2 中心对称, 区间 4 与区间 3 左右对称。那么只需要提供区间 1 内正弦值表即可通过坐标变换和取负值操作得到其余 3 个区间的正弦值。按照这个思想, 可以准备  $[0, \pi/2]$  内的正弦值的十六进制数组变量, 例如:

EmMath.c 文件中的数组 SinTbl。

```
# include "EmMath.h"
```

```
const unsigned short SinTbl[] = {0x0000, //0
0x0019, 0x0032, 0x004B, 0x0064, 0x007D, 0x0096, 0x00AF, 0x00C8, 0x00E2, 0x00FB, //10
0x0114, 0x012D, 0x0146, 0x015F, 0x0178, 0x0191, 0x01AA, 0x01C3, 0x01DC, 0x01F5, //20
0x020E, 0x0227, 0x0240, 0x0258, 0x0271, 0x028A, 0x02A3, 0x02BC, 0x02D4, 0x02ED, //30
0x0306, 0x031F, 0x0337, 0x0350, 0x0368, 0x0381, 0x0399, 0x03B2, 0x03CA, 0x03E3, //40
0x03FB, 0x0413, 0x042C, 0x0444, 0x045C, 0x0474, 0x048C, 0x04A4, 0x04BC, 0x04D4, //50
0x04EC, 0x0504, 0x051C, 0x0534, 0x054C, 0x0563, 0x057B, 0x0593, 0x05AA, 0x05C2, //60}
```

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```

0x05D9, 0x05F0, 0x0608, 0x061F, 0x0636, 0x064D, 0x0664, 0x067B, 0x0692, 0x06A9,//70
0x06C0, 0x06D7, 0x06ED, 0x0704, 0x071B, 0x0731, 0x0747, 0x075E, 0x0774, 0x078A,//80
0x07A0, 0x07B6, 0x07CC, 0x07E2, 0x07F8, 0x080E, 0x0824, 0x0839, 0x084F, 0x0864,//90
0x087A, 0x088F, 0x08A4, 0x08B9, 0x08CE, 0x08E3, 0x08F8, 0x090D, 0x0921, 0x0936,//100
0x094A, 0x095F, 0x0973, 0x0987, 0x099C, 0x09B0, 0x09C4, 0x09D7, 0x09EB, 0x09FF,//110
0x0A12, 0x0A26, 0x0A39, 0x0A4D, 0x0A60, 0x0A73, 0x0A86, 0x0A99, 0x0AAB, 0x0ABE,//120
0x0AD1, 0x0AE3, 0x0AF6, 0x0B08, 0x0B1A, 0x0B2C, 0x0B3E, 0x0B50, 0x0B61, 0x0B73,//130
0x0B85, 0x0B96, 0x0BA7, 0x0BB8, 0x0BC9, 0x0BDA, 0x0BEB, 0x0BFC, 0x0C0C, 0x0C1D,//140
0x0C2D, 0x0C3E, 0x0C4E, 0x0C5E, 0x0C6E, 0x0C7D, 0x0C8D, 0x0C9C, 0x0CAC, 0x0CBB,//150
0x0CCA, 0x0CD9, 0x0CE8, 0x0CF7, 0x0D06, 0x0D14, 0x0D23, 0x0D31, 0x0D3F, 0x0D4D,//160
0x0D5B, 0x0D69, 0x0D76, 0x0D84, 0x0D91, 0x0D9F, 0x0DAC, 0x0DB9, 0x0DC6, 0x0DD2,//170
0x0DDF, 0x0DEB, 0x0DF8, 0x0E04, 0x0E10, 0x0E1C, 0x0E28, 0x0E33, 0x0E3F, 0x0E4A,//180
0x0E55, 0x0E60, 0x0E6B, 0x0E76, 0x0E81, 0x0E8B, 0x0E96, 0x0EA0, 0x0EAA, 0x0EB4,//190
0x0EBE, 0x0EC8, 0x0ED1, 0x0EDB, 0x0EE4, 0x0EED, 0x0EF6, 0x0EFF, 0x0F07, 0x0F10,//200
0x0F18, 0x0F21, 0x0F29, 0x0F31, 0x0F39, 0x0F40, 0x0F48, 0x0F4F, 0x0F56, 0x0F5D,//210
0x0F64, 0x0F6B, 0x0F72, 0x0F78, 0x0F7F, 0x0F85, 0x0F8B, 0x0F91, 0x0F96, 0x0F9C,//220
0x0FA1, 0x0FA7, 0x0FAC, 0x0FB1, 0x0FB6, 0x0FBA, 0x0FBF, 0x0FC3, 0x0FC7, 0x0FCB,//230
0x0FCF, 0x0FD3, 0x0FD7, 0x0FDA, 0x0FDE, 0x0FE1, 0x0FE4, 0x0FE7, 0x0FE9, 0x0FEC,//240
0x0FEE, 0x0FF0, 0x0FF2, 0x0FF4, 0x0FF6, 0x0FF8, 0x0FF9, 0x0FFF, 0x0FFC, 0x0FFD,//250
0x0FFE, 0x0FFE, 0x0FFF, 0x0FFF, 0x0FFF}; //256

```

```

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function name: Em_Sin
description: calculate the value of sin(theta)
input: Angle(0x3FFFF equal one Cycle
output: sine value from the hex table
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
signed long Em_Sin(unsigned long Angle)
{
    unsigned long AngleTemp;
    signed long SineValue;

    AngleTemp = Angle >> 12;
    AngleTemp &= 0x03FF; //0~1024

    if(AngleTemp <= 256)
    {
        SineValue = SinTbl[AngleTemp];
    }
    else if(AngleTemp <= 512)
    {
        AngleTemp = 512 - AngleTemp;
        SineValue = SinTbl[AngleTemp];
    }
    else if(AngleTemp <= 768)
    {
        AngleTemp -= 512;
        SineValue = -SinTbl[AngleTemp];
    }
    else if(AngleTemp <= 1024)
    {
        AngleTemp = 1024 - AngleTemp;

```

```

    SineValue = -SinTbl[AngleTemp];
}

return(SineValue);
}

```

该文件内提供一个 Sintb 数组,存放 $[0, \pi/2]$ 区间内分辨率为 1/256 的数据,通过十六进制整数 $[0x0000, 0x0FFF]$ 表示实际值 0 到 1。该函数将输入的 32 位无符号整型数转换为表格中对应的数据点索引号,按照索引号进行相应变换取得正确的正弦值。那么根据这个资源使用 LCT 将其集成到 Simulink 模型中,生成 C Mex S 函数 C 文件,并进行仿真。

**注:**LCT 生成的 C Mex S 函数入口都被设定为直接馈入。

将 EmMath.c 和 EmMath.h 文件拷贝到工作目录下,省去填写相对路径或绝对路径的麻烦。初始化 S 函数名设为 sfun\_Em\_Math.c,根据 C 函数原型,将 unsigned long 和 signed long 使用 Simulink 内建的数据类型 uint32, int32 数据类型替换,函数的输入参数使用 u1,返回值即输出参数使用 y1 替换,规定 S 函数中原型函数原型为:

```
int32 y1 = Em_Sin(uint32 u1)
```

配置之后,生成名为 Em\_Sin 的 C Mex S 函数,并编译为 mex 文件,自动添加到模型 lct\_model 中去,并将上述需要初始化的信息编写为 M 语言:

```

lct_spec = legacy_code('initialize');
lct_spec.SFunctionName = 'sfun_Em_Math';
lct_spec.HeaderFiles = {'EmMath.h'};
lct_spec.SourceFiles = {'EmMath.c'};
lct_spec.OutputFcnSpec = 'int32 y1 = Em_Sin(uint32 u1)';
legacy_code('sfcn_cmex_generate', lct_spec);
legacy_code('compile', lct_spec);
legacy_code('slblock_generate', lct_spec, 'lct_model');

```

运行上述代码之后,在 Command Window 中显示出如下编译成功的信息;

```

# ## Start Compiling sfun_Em_Math
mex('sfun_Em_Math.c', 'D:\FILES\LCT_try\EmMath.c', '-ID:\FILES\LCT_try')
# ## Finish Compiling sfun_Em_Math
# ## Exit

```

并自动建立一个名为 lct\_model 的模型,将生成的 S 函数封装成模块显示到模型中,如图 10.5-39 所示。

同时在当前文件夹下已经存放了自动生成的 sfun\_Em\_Math.c 和编译之后生成的动态可执行文件 sfun\_Em\_Math.mexw64(使用 64 位 Windows)。

sfun\_Em\_Math.c 在其 mdlOutputs 子方法中对 Em\_Sin 函数进行了调用,如图 10.5-40 所示。

下面通过模型仿真验证生成的模块是否能够实现原本 C 代码的正弦波查表计算功能。源代码设计输入为 $[0, 1023]$ 的 uint32 的整数时能够输出一个周期内的正弦值。在源代码中存在语句:AngleTemp = Angle $\gg$ 12;即将输入量右移 12 位之后才是 $[0, 1023]$ 范围的数据,那么在模型中给定输入时期待的是 0~1023 的向量输入的话,由于 S 函数模块会进行右移 12 位的操作,需要将其放大 $2^{12}$ 倍再输入 S 函数以保持这个过程中输入值匹配。建立模型如图 10.5-41 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

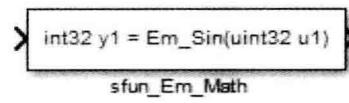


图 10.5-39 LCT 工具生成的 S 函数模块

```

154 /* Function: mdlOutputs =====
155 * Abstract:
156 *   In this function, you compute the outputs of your S-function
157 *   block. Generally outputs are placed in the output vector(s),
158 *   ssGetOutputPortSignal.
159 */
160 static void mdlOutputs(SimStruct *S, int_T tid)
161 {
162     /*
163     * Get access to Parameter/Input/Output/DWork/size information
164     */
165     uint32_T *u1 = (uint32_T *) ssGetInputPortSignal(S, 0);
166     int32_T *y1 = (int32_T *) ssGetOutputPortSignal(S, 0);
167
168     /*
169     * Call the legacy code function
170     */
171     *y1 = Em_Sin(*u1);
172 }

```

若您对此书内容有任何疑问，  
可以凭在线交流卡登录MATLAB中文论坛与作者交流。

图 10.5-40 自动生成 S 函数的 mdlOutputs 子方法

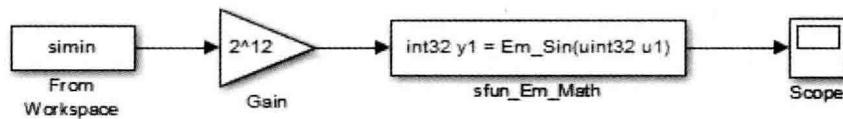


图 10.5-41 正弦查表模块的仿真模型

From Workspace 模块中的信号 simin 使用下述脚本产生 [0,1023] 的时间序列(timeseries)：

```

stop_time = get_param(gcs, 'StopTime');
simin.time = [0 : str2num(stop_time)]'; %采样时间 1 s
simin.signals.values = [0 : length(simin.time) - 1]';
simin.signals.dimensions = [length(simin.time) 1];

```

根据模型的仿真时间长度产生 From Workspace 模块中的数据信号，模型仿真时间设置为 1023 s，在每一秒钟输入当前仿真时间的整数作为正弦波查表函数的输入。运行仿真之后，得到一个完整周期的正弦波图像如图 10.5-42 所示。

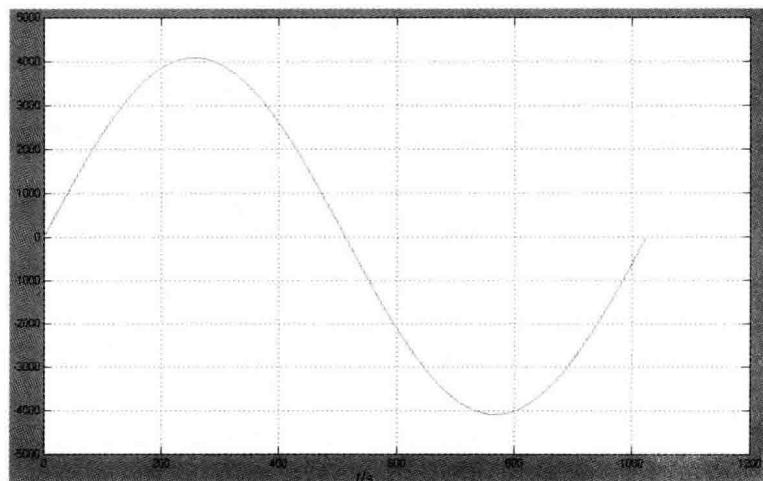


图 10.5-42 正弦查表模块的仿真图像

仿真输出标准的正弦波,其幅值在 0~4095 之间,这是由于 Em\_Sin 函数中输出的是固定点格式数据,在仿真时可以后跟一个 1/4095 的 Gain 模块将其缩小到 0~1 的范围。MCU 中之所以使用 0~4095 表示 0~1 的值是为了能够对小数进行精度为 1/4096 的计算,如果只是用 0~1 表示 0~1,0.1 和 0.03 这样的小数便无法表示,精度太低则无法应用到工程中去。上述模型前的 Gain 负责将真实世界的值转化为 MCU 内部的数值表示,而输出后如果再接一个增益为 1/4096 的 Gain 模块就将 MCU 计算出的值再转换为真实世界的值。固定点 MCU 为了能够使用固定点数表现真实世界的值,通常使用上述方法。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

# 第 11 章

## 模块的封装

**引言:**Simulink 自带了功能强大的模块库,同时也提供给用户自定义模块的 S 函数模块以及各种 S 函数编写方式。当用户编写了自定义的 S 函数之后,可通过封装为这个模块设计显示外观,并为 S 函数所需要的参数添加对应的控件,共同构成模块的参数对话框。另外,当用户使用 Simulink 标准库中的模块搭建子系统后,也可以通过封装为这个子系统追加参数对话框。封装的方法既可以通过手动添加控件设置属性,也可以直接通过编程自动实现。

双击任意一个 Simulink Library Browser 中提供的模块时,将弹出一个对话框,包括模块功能的说明、参数的标签及可编辑控件,其中常见的控件有 Edit、Check-box 和 Popup-menu 等,并且可以根据参数相关性将控件分类,同类参数的控件放置在同一个标签页上,起到美观与简化界面的作用。每个参数都对应模块内部 S 函数的参数(或者内部子模块参数)。在对话框的右下方有 4 个按钮(如图 11.1-1)。其中 Help 按钮被按下时会弹出详细解释这个模块的功能以及每个参数的使用方法的文档。

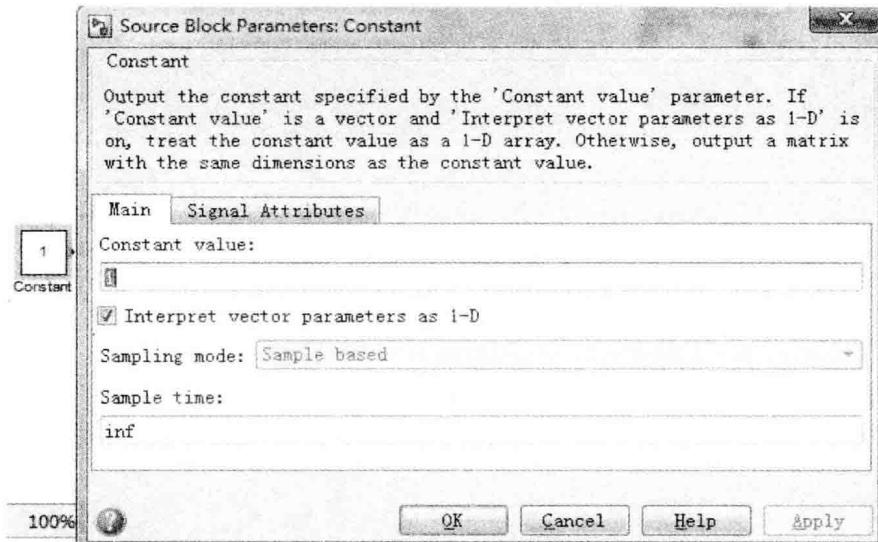


图 11.1-1 常数模块的对话框

封装是用户构建如上所述的这样一个以对话框为接口的交互界面的过程,它将复杂的模块逻辑关系隐蔽起来,封装之后仅提供给用户 GUI 界面填写参数。用户仅需操作对话框界面即可实现定义的功能。

创建封装的方式有多种,可以通过 Mask Editor 创建,也可以通过编写 M 代码定制创建过程,两种方法可以根据应用场景有所取舍。当封装一个比较简单的模块时,可以使用 Mask Editor 创建方法;需要封装一个大型复杂的(如带有数百个参数)模块时,可以采用 M 脚本编写函数自动完成协助封装过程。

## 11.1 Mask Editor 封装模块

对于初学者,初期实践封装时,大多是参数较少的模块,可以采用 Mask Editor 封装方法。使用 Mask Editor 创建可以更细致地了解模块参数对话框的数据结构及其特性,在此基础上使其可以完成 M 代码定制创建过程。Mask Editor 封装的对象有两种,一种是由 Simulink Library 中的模块或由多个模块构成的子系统,另一种则是由 S 函数编写成的模块。前者的每个参数都已经具有变量名和依附的控件,只需要将其链接到新封装的 GUI 控件上即可;后者则需要为每个参数创建变量名和参数控件。

### 11.1.1 封装模块构成的子系统

以简单的数学表达式  $y=a \cdot x^2+b$  为例,其中  $x$  为输入,  $y$  为输出,  $a, b$  为参数。这个数学表达式可由图 11.1-2 所示模型表示出来。

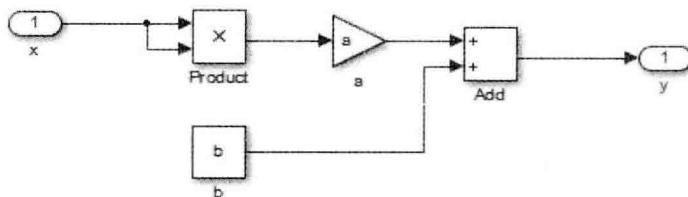


图 11.1-2  $y=a \cdot x^2+b$  模型

其中参数  $a$  为增益模块的增益值,参数  $b$  为 Constant 模块的常数值。封装的结果为用户可用对话框的形式定义这两个参数。选中图 11.1-2 所有模块与信号线后,按下  $Ctrl+G$  即可创建子系统。更改模块大小并移动到合适位置,修改子系统模块名为“ $y=a \cdot x^2+b$ ”,以使封装模块简明、美观,修改后子系统如图 11.1-3 所示。

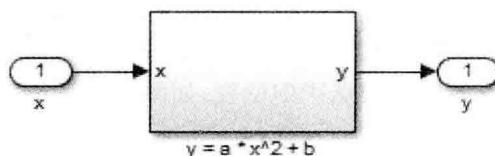


图 11.1-3 创建子系统

右击子系统模块,选择菜单中的 Mask 选项,单击 Create Mask 打开 Mask Editor 对话框,此过程如图 11.1-4 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

**261**

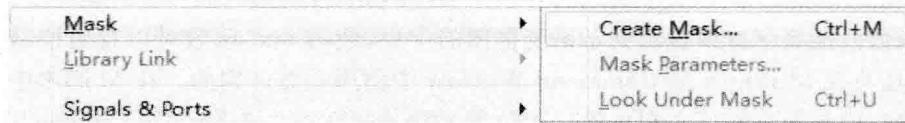


图 11.1-4 右键子系统的菜单

Mask Editor 的界面如图 11.1-5 所示,包括 4 个页面,作用分别是:

Icon & Ports: 编辑子系统的模块外观,如在子系统图标中添加线条、文本和图像等;

Parameters&Dialog：添加或修改模块参数，并为其设计控件类型；  
Initialization：编辑子模块的初始化脚本；  
Documentation：添加子模块的功能介绍及 Help 文档路径。

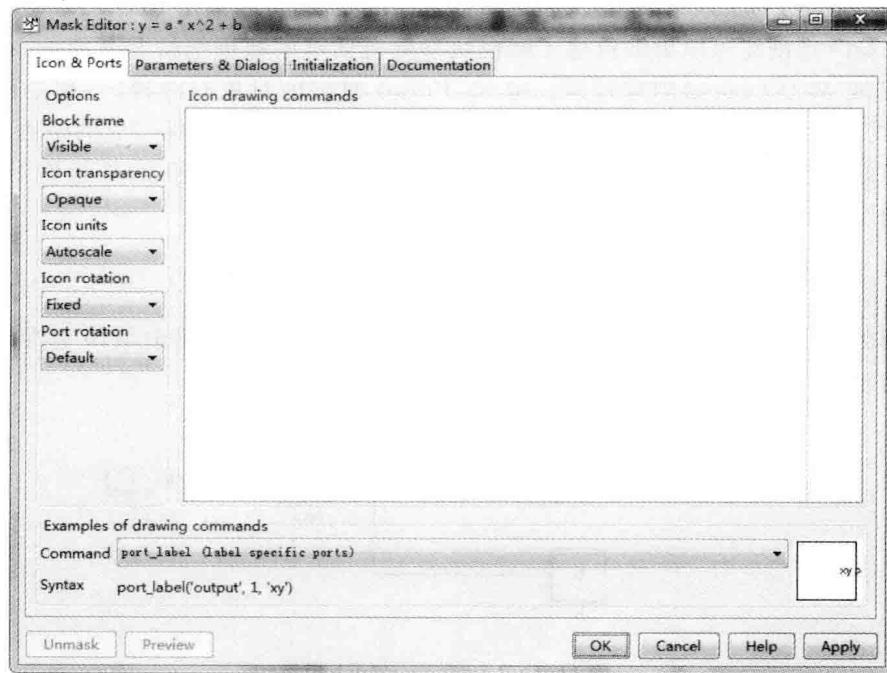


图 11.1-5 Mask Editor 封装编辑器界面

一个完整的封装过程包括定义编辑器内各个参数。如果用户没有特别严格的初始化时刻要求，可以不填写 Initialization 页面的内容。

### 1. Icon & Ports 页面

首先需要在 Icon&Ports 页面的 Icon Drawing Commands 页面中输入 M 脚本，将文字、图像或者绘制线条等图示显示到子系统的图标上去。最常用的函数有 disp、text、image 和 color 等。使用 disp 函数将文本显示在模块居中的位置，如图 11.1-6 所示。

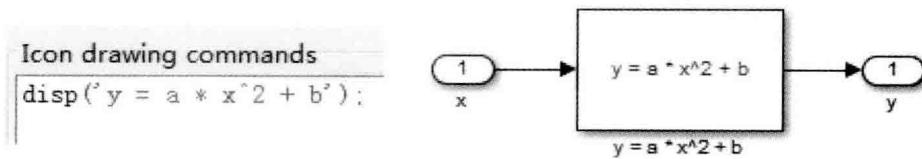


图 11.1-6 使用 disp 显示功能说明

disp 函数只能将字符串显示到子系统框图的中央，使用 text 函数可以自由控制文字显示的位置，但是与在 M Editor 和 Command Window 中使用是有区别的。在 M 脚本中使用 text 是在图形窗口内的坐标平面内根据给定的位置参数显示文本。在 Drawing commands 中使用则受到限制。比如不能设置字体大小，加粗等效果。读者可以理解为，在 M Editor 或 Command Window 中使用的 text 函数与子系统封装的 text 函数是不同的函数，或者说不同种类的函数。在子系统封装 text 时只能使用以下方式：

- ① `text(x, y, 'text')` —— x, y 表示坐标对, 指 text 文本相对原点所显示的位置。  
 ② `text(x, y, 'text', 'horizontalAlignment', 'halign', 'verticalAlignment', 'valign')`  
 —— 设置文本显示的坐标及方位。

③ `text(x, y, 'text', 'texmode', 'on')` —— 开启 tex 文本模式。

此函数产生的对象没有其他属性可以使用, 所以不能使用 `text` 在这里设置字体大小, 加粗等效果。通常采用方式 2 进行文本位置的显示, 如:

```
text(0, 0, 'y = a * x^2 + b', 'horizontalAlignment', 'left', 'verticalAlignment', 'bottom')
```

将表达式显示到模块的左下角, 如图 11.1-7 所示。

`text` 函数的前两个参数为文本起点位置相对于坐标原点的位移坐标, 建议使用归一化坐标, 控制范围为 [0~1]。用户可以通过下拉菜单 Icon Units 选择 Normalized 项来开启归一化坐标单位(见图 11.1-8 粗线框内的部分)。后两个属性 `horizontalAlignment` 与 `verticalAlignment` 是坐标原点的选定。此后, 再输入以下语句:

```
text(0.3, 0.5, 'y = a * x^2 + b', 'horizontalAlignment', 'left', 'verticalAlignment', 'bottom')
```

这样, 文本显示的起点位置相对于坐标原点(左下角), 在横轴右移 0.3 个单位长度, 纵坐标上移 0.5 个单位长度, 如图 11.1-9 所示。

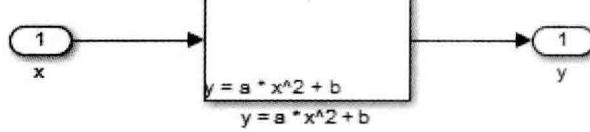


图 11.1-7 使用 `text` 显示字符串到模块左下角

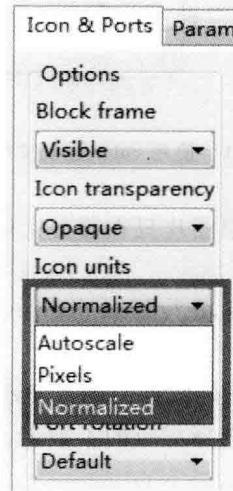


图 11.1-8 使用归一化坐标

在 `text` 函数与 `disp` 函数语句之前使用 `color` 函数, 可以规定文本显示时所用的颜色, 其参数可为 blue、green、red、cyan、magenta、yellow 和 black, 如使用红色:

```
color('red');
```

单击 OK 按钮关闭封装对话框 Mask Editor 之后模块显示如图 11.1-10 所示。

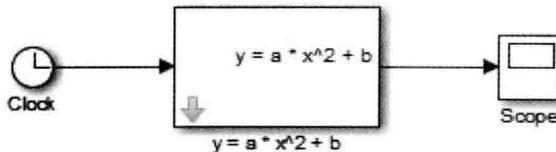


图 11.1-9 使用 `text` 显示字符串到模块右上部分

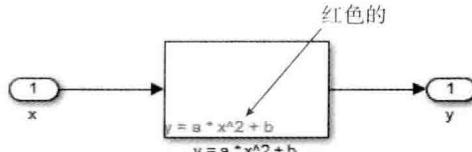


图 11.1-10 使用 `color` 控制文字显示的颜色

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

如果使用多种颜色,可使用多个 color 分段调整文字的颜色。在重新定义颜色前,新添加的文本为最近一个 color 函数所定义的颜色。如:

```
color('red');
text(0,0,'y = a * x^2 + b','horizontalAlignment','left','verticalAlignment','bottom');
disp('Thanks Eflen!');
color('cyan');
text(0.1,0,'hyo','horizontalAlignment','right','verticalAlignment','bottom');
```

编辑完上述代码之后单击 OK 按钮关闭 Mask Editor,则模块显示如图 11.1-11 所示。

模块框图上还可以显示图片。下面举例说明如何使用 image 与 imread 函数,将图片显示到子系统或 S 函数模块框图上。准备一张名为 Mask\_demo01.jpg 的图片,Icon Drawing Command 填入以下语句,获得全伸展的图片显示效果。

```
image(imread('Mask_demo01.jpg'));
```

将上句填入 Icon drawing command 中,单击 OK 按钮关闭 Mask Editor,则模块显示如图 11.1-12 所示。

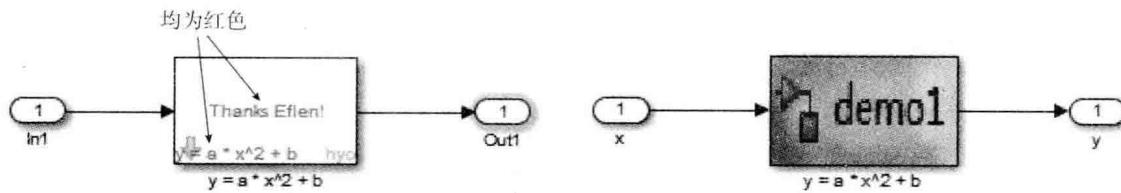


图 11.1-11 使用 color 控制文字显示不同的颜色

图 11.1-12 图片显示到模块

当希望图片只在部分区域显示时,使用 image 的第二个参数来规定显示的位置:

```
image(imread('Mask_demo01.jpg'),'top-left');
```

将上句填入 Icon drawing command 中,单击 OK 按钮关闭 Mask Editor,则模块显示如图 11.1-13 所示。

Image 函数控制图片显示位置的参数有 4 种,如图 11.1-14 所示。用户可以根据需要显示的位置选择对应的参数名。

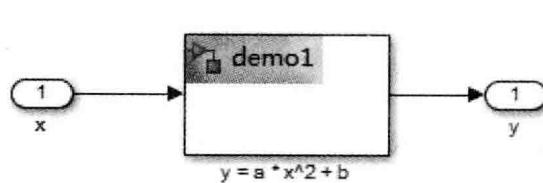


图 11.1-13 图片显示到模块局部

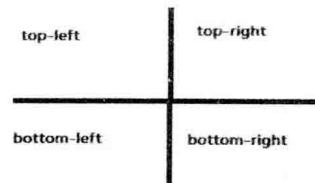


图 11.1-14 图片显示到模块位置的参数名

开发者在封装完 Parameters&Dialog 之后,还可通过 plot 函数绘制图像到所封装模块的框图上,直观地将模型所表示的函数图像表达出来。在 Icon Drawing Commands 输入以下代码获取对话框中封装的参数,并绘制自变量在 [0 10] 范围内的图像:

```
a = str2num(get_param(gcbh,'g_a'));
b = str2num(get_param(gcbh,'g_b'));
t = 0:0.1:10;
plot(t,a*t.^2+b);
```

在参数对话框中输入  $a=1, b=20$ , 单击右下角 OK 按钮, 模块框图如图 11.1-15 所示。

封装子系统的模块图标时, 可使用 port\_label 函数设置输入输出端口的显示名称。该函数使用方式如下:

```
port_label('port_type', port_number, 'label', 'texmode', 'on')
```

port\_type——指定输入或输出端口, 使用 input, output 表示, 当子模型为使能子系统、触发子系统或与流控制相连的 Action 子系统时也可以指定使能端口 Enable、触发端口 trigger 或 Action 端口 Action。

port\_number——当 port\_type 指定的端口类型存在多个端口时, 此参数用来指定其序号, 从 1 开始。

label——在指定的端口显示的文本字样。

texmode, on——两个参数合起来表示 Tex 模式开启与否, 若为 texmode, off 则为关闭 Tex 模式。将以下代码输入 Icon Draw Command 的窗口, 单击 OK, 可得图 11.1-16 所示的图标:

```
port_label('input', 1, 'In\spadesuit', 'texmode', 'on');
port_label('output', 1, 'Out\heartsuit', 'texmode', 'on');
```

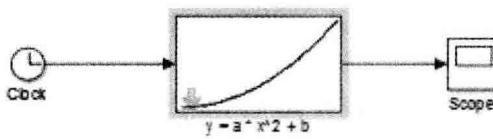


图 11.1-15 使用 plot 绘制数学图像到模块外观

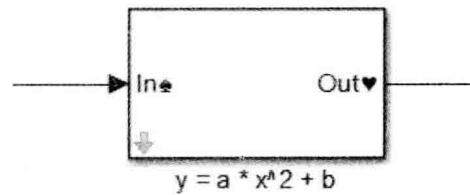


图 11.1-16 自定义输入输出端口的显示符号

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

## 2. Parameters & Dialog 页面

设计完模块的显示外观之后, 需要在图 11.1-17 所示 Parameters & Dialog 页面上定义对话框的控件。

页面左边有一个名为 Controls 的控件选择面板, MaskControl 按照类型不同可以分为 Parameter、Display 和 Action 3 种, 其分别提供的子控件如图 11.1-18 所示。

Parameter: 包括 edit、popupmenu、radiobutton 和 checkbox 等控件, 其具有 Value 属性, 用户可输入或选择参数的值, 并将该值以变量的形式传递到模块内部使用; 用户可以定义控件参数值发生变化时所触发的回调函数。

Display: 包括 Panel、tab、groupbox、text 和 image 5 个控件, 其作用是提示用户信息, 比如显示文字或图片, 或者使用 Panel, tab, group box 进行分组, 控件 tab 和 groupbox 内部还可以再使用 Mask Controls。其中 tab 控件与其他控件有所不同, 不能直接添加到模块对象中, 需要与称作 tabcontainer 的隐藏控件共同使用, 具体编程方法在后面给出。这些控件仅在视觉上起到简明、美观的提示作用, 没有 Value 属性和回调函数。

Action: 包括超链接和按钮 2 种控件, 可通过单击动作触发用户自定义的回调函数, 但是控件本身没有变量名, 也不具备传递值到封装的模型或 S 函数内部的功能。

Dialog Box 框是当前对话框已经包含的控件信息表格, 表格默认显示 3 个属性: type、Prompt 和 Name, 分别表示控件的类型、说明标签和变量名。新建的 Mask Editor 中默认提供

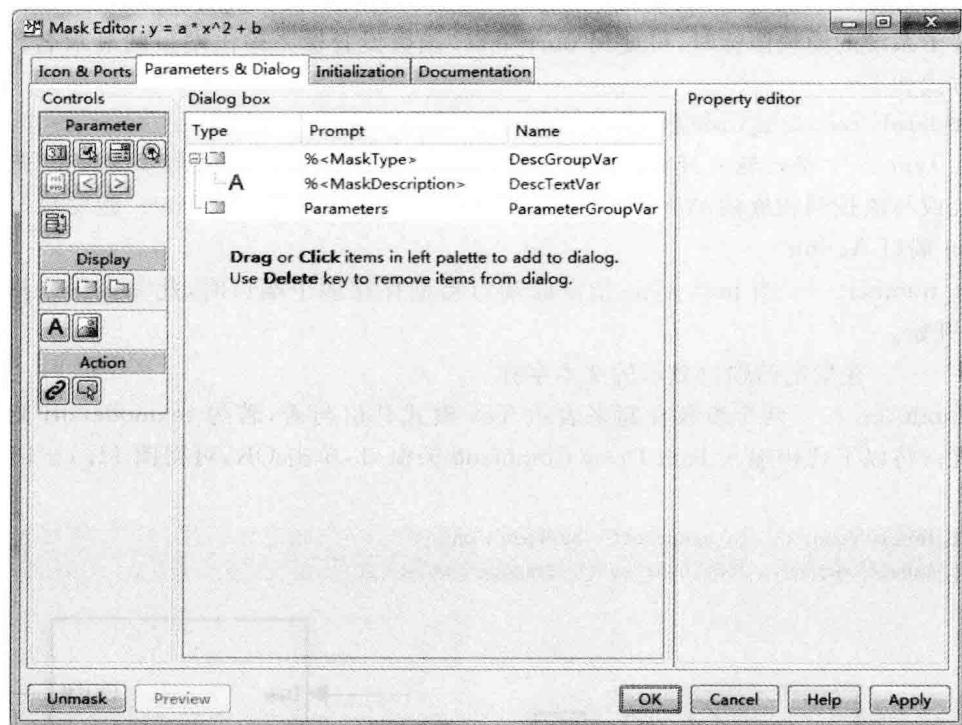


图 11.1-17 Parameters &amp; Dialog 页面

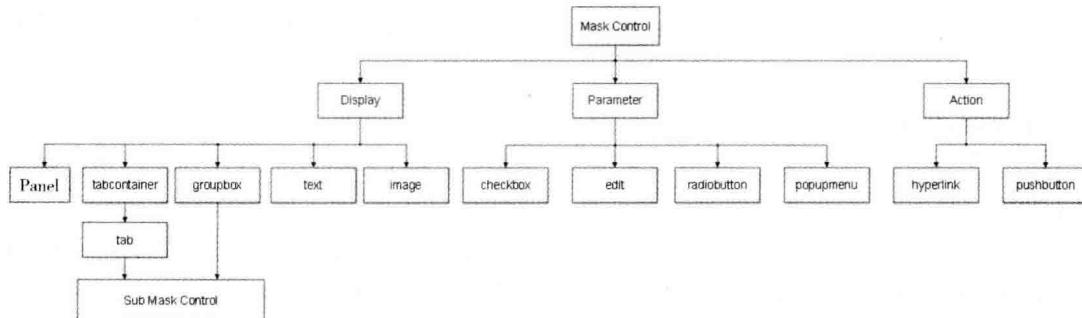


图 11.1-18 Mask Controls 的分类结构

了 3 个 Display 类型的控件: MaskType(Panel)、MaskDescription(Text) 和 Parameters(Panel)。前 2 个是为了从第 4 个页面 Documentation 中提取对应信息显示到对话框中, 第 3 个则是为用户接下来封装的控件提供一个 Panel 组名。

封装时, 用户从 Controls 提供的控件中选择一个, 如 Edit, 该控件类型会自动追加到 Dialog box 中(Parameters&Dialog 对话框如图 11.1-19 所示), Type 栏中显示 #1 表示具有传递参数值到内部模块功能的参数编号, 从 1 开始。Prompt 缺省值为空字符, 用户可以输入简短的文字以说明这个控件的用意。Name 列为该控件的变量名, 可以通过变量名 Name 在被封装的模块或 S 函数内部访问控件的值。

**注:** 此变量名主要表征对话框 GUI 中的参数, 在代码生成时会传递到 S 函数中, rtw 文件中, TLC 文件中, 同一个变量在整个参数传递过程中多次使用, 尤其在 C mex S 函数的 C 文件

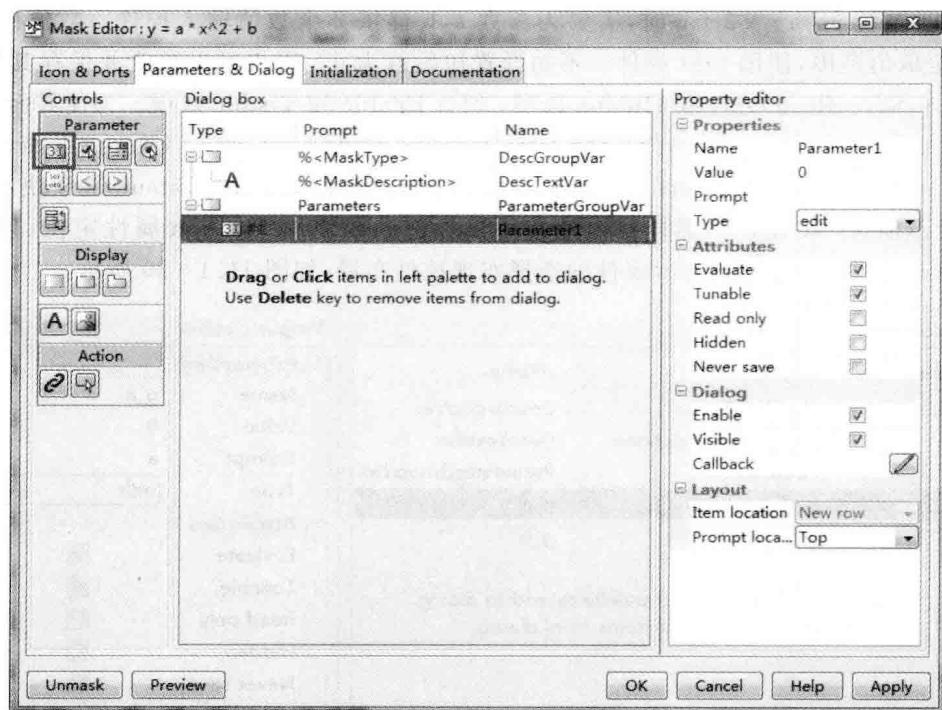


图 11.1-19 添加控件到对话框中

中更是多次出现在各个子方法中。建议用户添加前缀加以区分,以免出现混淆。如在对话框中的变量则建议 g\_ 为开头的变量名,在 C 文件中 c\_,在 TLC 文件中 t\_ 作前缀。

当 Dialog box 中的某一个控件被选中时,会在最右边的 Property editor 中显示其属性列表,常用属性的属性名、所属组别与作用简介如表 11.1-1 所列。

表 11.1-1 Property Editor 属性列表

| 属性名             | 组 别        | 作 用  |
|-----------------|------------|--|
| Name            | Property   | 控件变量名                                      |
| Value           |            | 控件变量的值                                     |
| Prompt          |            | 控件作用提示文字                                   |
| Type            |            | 控件类型选择                                     |
| Evaluate        | Attributes | 控件中输入的内容作为字符串 str,并进行一次 eval(str)操作        |
| Tunable         |            | 仿真过程中控件值是否可变                               |
| Read Only       |            | 只读,不可编辑                                    |
| Hidden          |            | 隐藏控件                                       |
| Never Save      |            | 即使控件值改变也不保存                                |
| Enable          | Dialog     | 仅在 Read Only 不使能时才使用,若不使能则用户不可编辑控件,控件显示为灰色 |
| Visible         |            | 控制控件是否可见,仅当 Hidden 不勾选情况下才可使用              |
| Callback        |            | 编写或调用控件的回调函数                               |
| Item Location   | Layout     | New/Current Row,设定控件放在当前行或者另起一行            |
| Prompt Location |            | Top/Left, 设定说明文字在控件的上方或左方                  |

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

封装子系统  $y=a \cdot x^2+b$  时,需要为参数 a,b 提供 2 个数值输入控件。数值由用户指定,不限定取值范围,使用 Edit 控件。不妨设置初始值为 0。用户可以修改并保存其值,故不勾选 Never Save、Read Only 和 Hidden 选项,勾选 Enable 和 Visible 选项。为使界面紧凑,将 a,b 两个参数放到对话框同一行,a 的 Item location 设为 New row, Prompt location 设为 left;b 的 Item location 设为 current row, Prompt location 设为 Left。Simulink 模块对话框的控件位置不能像 GUIDE 中那样自由地使用鼠标拖动或使用 Position 属性定位,只能通过 Item location 和 Prompt location 属性的选择实现控件布局,如图 11.1-20 所示。

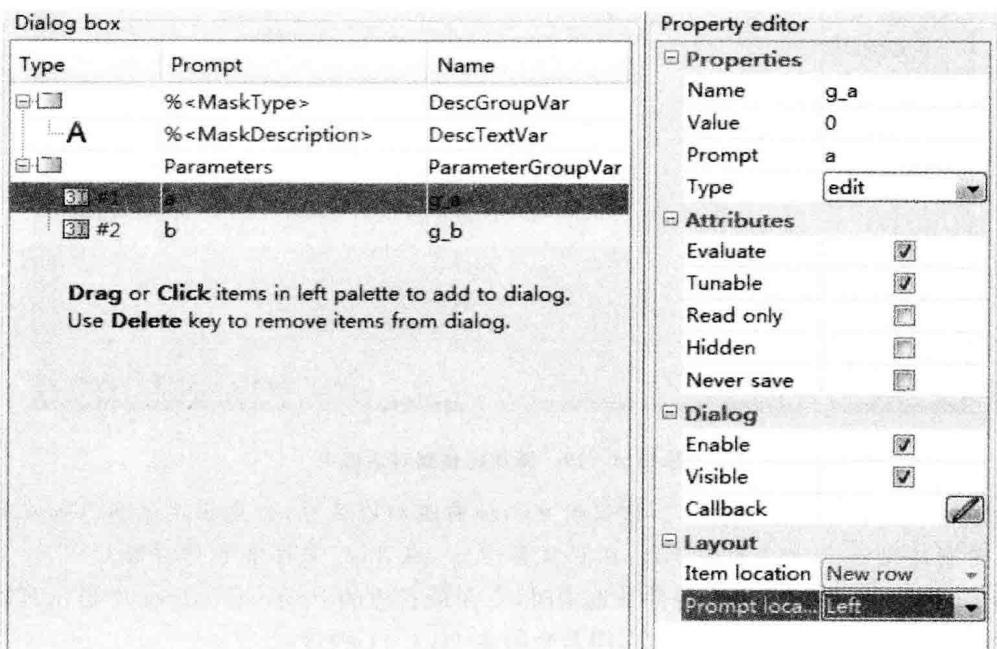


图 11.1-20 为参数 a、b 设计控件类型及属性

按照上述属性设计完之后单击 Mask Editor 右下角的 OK 按钮保存并关闭,再双击  $y=a \cdot x^2+b$  模块打开刚才设计的对话框查看,a 和 b 两个参数的控件都在同一行显示,如图 11.1-21 所示。

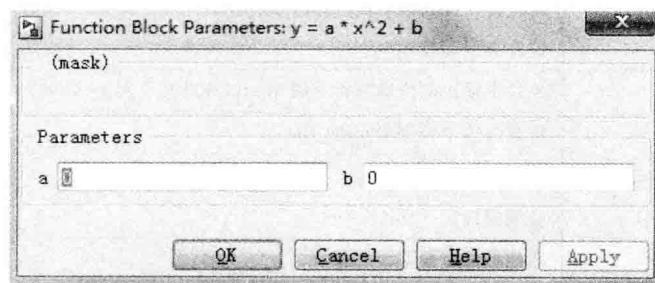


图 11.1-21 封装好的对话框

a,b 的值输入之后,会分别传递给 g\_a, g\_b 这 2 个 Mask 变量。该子系统内部的 Gain 模块和 Constant 模块的参数对话框中的 Gain 和 Constant value 的值分别填入 g\_a,g\_b 即可。使用者只要修改子系统对话框的值即达到了修改模块参数值的效果,如图 11.1-22 所示。

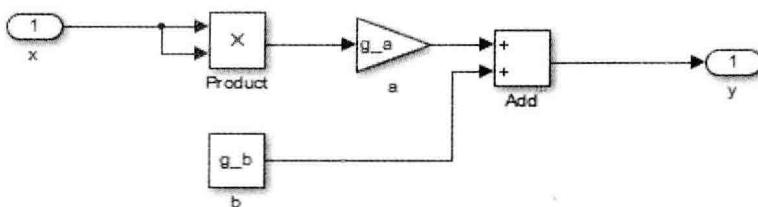


图 11.1-22 修改内部模块的参数名与封装参数名一致

使用者使用 Edit 输入框时存在输入非法字符的可能性,为了规避这种情况,可为 Edit 控件定制回调函数,当用户输入内容后自动进行合法性检查,如果不合要求则按照规定的方式给出提示。如,规定此处 a,b 只能输入数据类型,如果输入字符串,则弹出错误对话框提示用户进行修改。检查功能可以用 check\_num 的函数实现:

```
function check_num(param)
% This function check if the value of param is a number data type.
val = get_param(gcbh, param);
if isletter(val)
    errordlg(['Input value of ', param(end), ' must be a number data - type! ']);
    return;
end
```

这是一个通用的函数,适合 a,b 两个参数使用,首先根据调用时使用的 param 获取参数值,当参数的值是字母时,报出错误框提示用户输入符合要求的内容。在两个 Edit 控件的 Callback Editor 中调用上述函数,如图 11.1-23 所示。

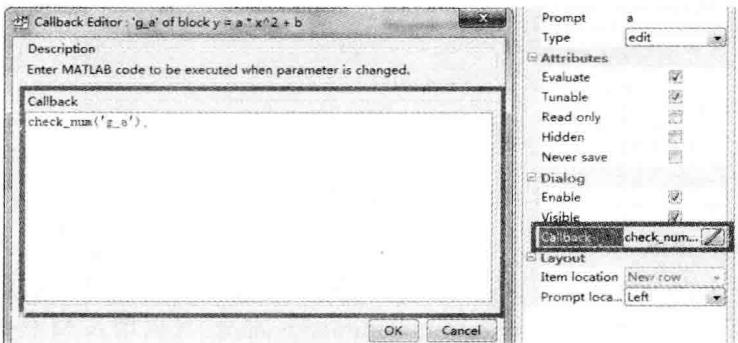


图 11.1-23 参数 a 的回调函数编辑框

通常,可以在图 11.1-23 所示的对话框中直接编写脚本实现上述功能。但是这个框中不具备 M Editor 提供的各种快捷方式,也不支持断点等调试功能,一旦代码中有错误,排查错误时非常不便。所以先在 M Editor 中设计好函数,保存后直接在这里调用这个函数即可。选中参数后,单击右边 Callback 按钮即可打开这个对话框,为 a,b 两个参数分别填入调用的语句即可。

当双击打开  $y=a*x^2+b$  模块时,如果在参数 a 中填入了字母,那么当鼠标焦点移开 a 时,或者按下 OK 按钮或 Apply 按钮时都会触发回调函数,跳出如图 11.1-24 所示的错误对话框。

现在仅仅规定了填入字母的错误提示,如果输入的既不是字母也不是数字会怎么样呢?其实 Simulink 本身就具有一定的字符处理及检测机能。如果用户填入标点符号等非法字符,会自动弹出图 11.1-25 所示错误提示,说明这些字符是不被支持的。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

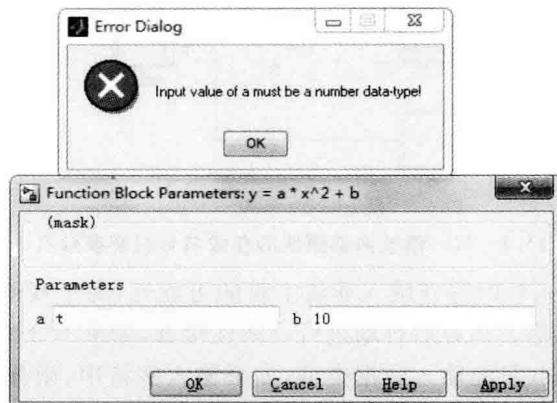


图 11.1-24 参数类型不对应的错误提示

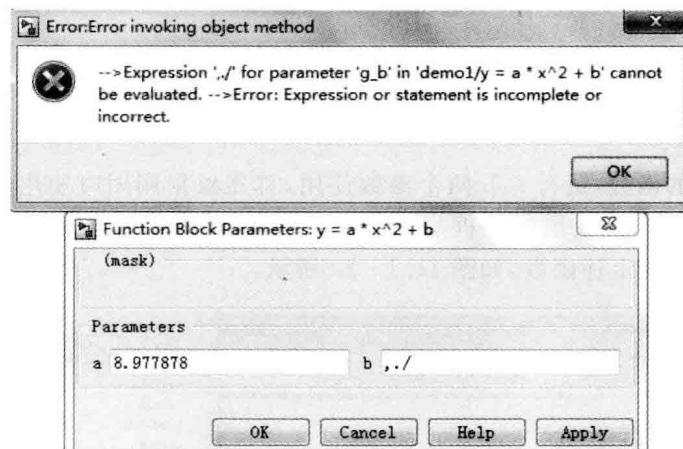


图 11.1-25 Simulink 的错误提示

### 3. Initialization 页面

Initialization 页面主要用于定义模块的初始化命令,已经封装过的参数名将列在左侧,如图 11.1-26 所示。此页面内(Initialization Commands 部分)可以填入 M 代码。

模块的 Initialization Commands 执行的时刻包括以下几个情况:

- ① 在 Icon draw Commands 或 Initialization Commands 里更改封装参数时。
- ② 当 Icon draw Commands 有内容时,翻转或旋转模块。
- ③ 双击打开模块的参数对话框或单击参数对话框上的 Apply/OK/Cancel 按钮关闭对话框时。
- ④ 当使用 set\_param 函数更改模块参数的值时,甚至赋相同值到某参数上时也会触发调用。
- ⑤ 拷贝此模块到同一模型中或其他模型中。
- ⑥ 模型运行仿真时。
- ⑦ 模型编译生成代码时。
- ⑧ 模型工具栏中选择 Update diagram 时(ctrl+D)。

如果需要在上述这些时刻触发某些动作,或者重新刷新绘制模块的外观,可以在这里编写 M 脚本或者像 Callback 那样调用 M 函数。

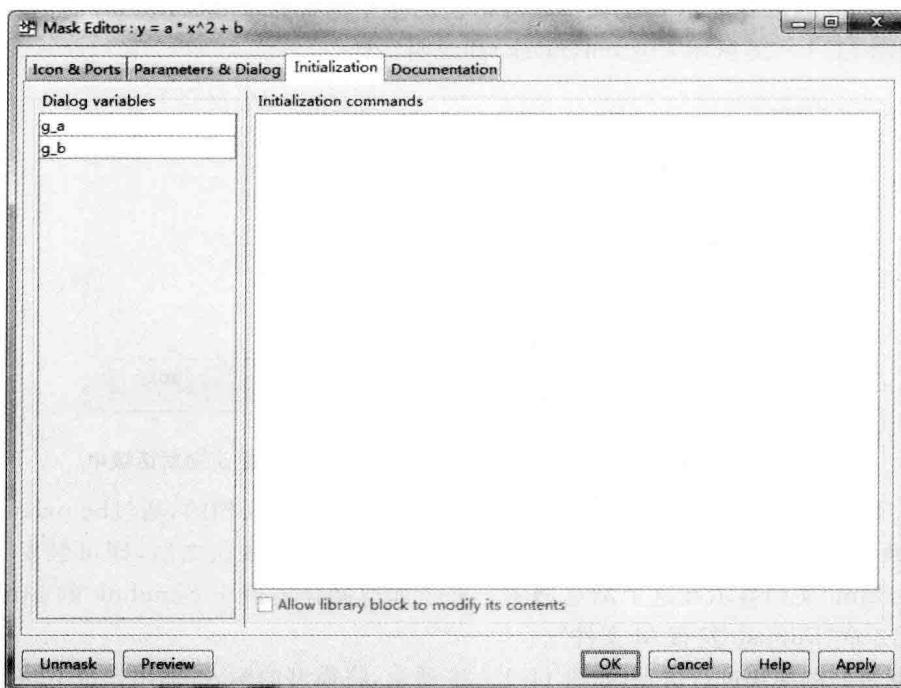


图 11.1-26 初始化命令页面

#### 4. Documentation 页面

Documentation 页面主要用来补充模块的说明信息及 help 按钮的功能,如图 11.1-27 所示。

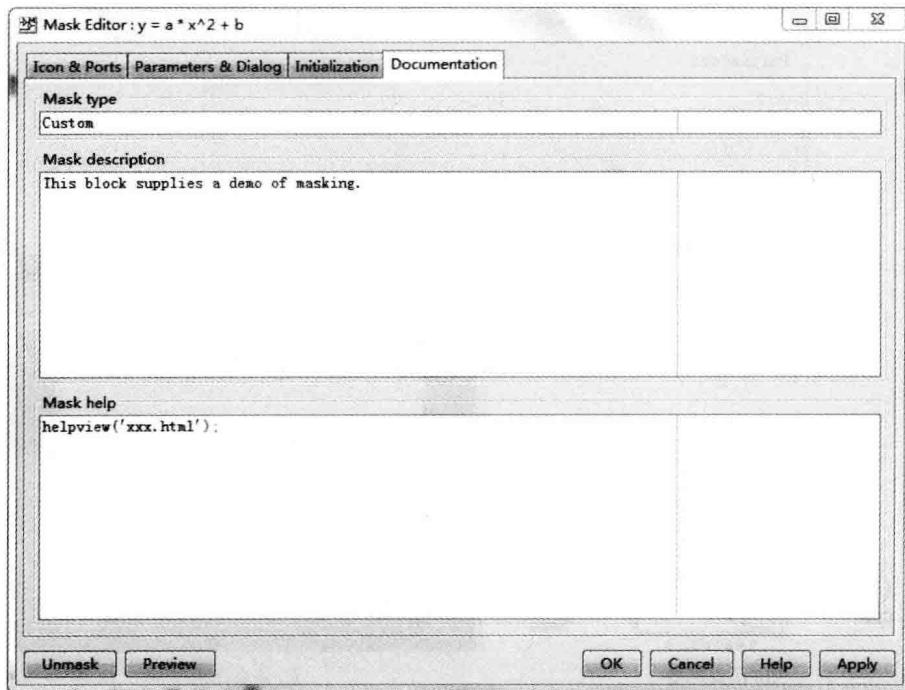


图 11.1-27 Documentation 页面

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

Mask Type 的内容显示到模块参数对话框的左上角, Mask description 的内容则紧接其后显示, 如图 11.1-28 所示 Custom(mask) 和说明字样。

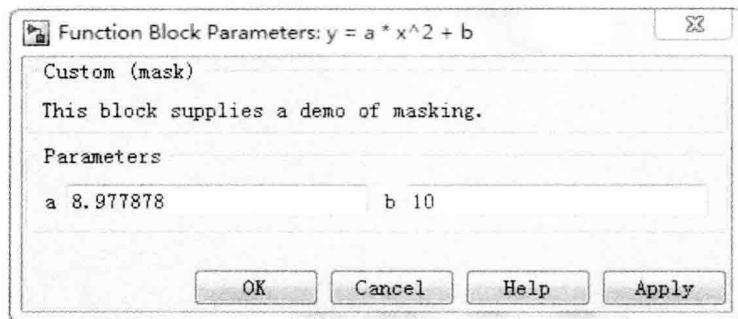
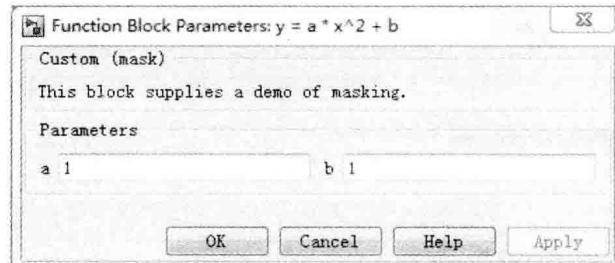


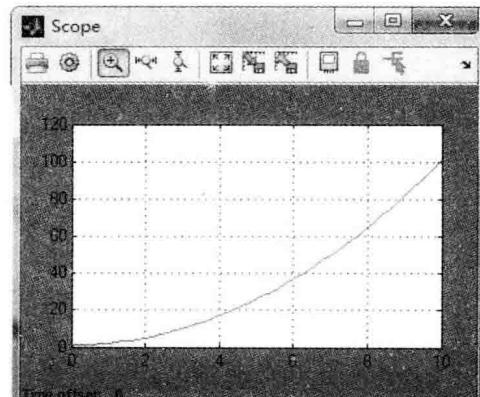
图 11.1-28 封装后 Mask Type 和 Mask Description 显示到对话框中

图 11.1-27 下方的 Mask help 是填写模块所链接的帮助文档的, 通过 helpview 打开 html 类型的帮助文档。当使用者单击图 11.1-27 右下方的 Help 按钮之后, 即可触发 Help 浏览器, 将 xxx.html 文档显示在这个浏览器中。关于如何制作内嵌于 Simulink 的 html 文档, 可以参考第 12 章“Publish 发布 M 文件”。

使用封装好的模块进行仿真, 如图 11.1-29 所示, 将模型的输入/输出换成 Clock 信号源与 Scope 示波器以便观察结果。当 a、b 中都输入 1 时, 仿真时间 20 s 的仿真结果如图 11.1-29(c) 所示。



(a) 参数设置



(b) 仿真模型

(c) 仿真图形

图 11.1-29  $a=1, b=1$  情况下的仿真结果

当  $a = -1$ ,  $b = 1$  时, 如图 11.1-30 所示。

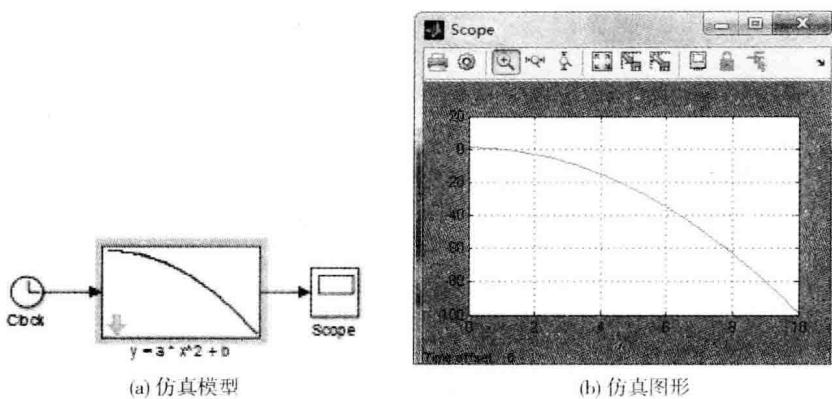


图 11.1-30  $a = -1$ ,  $b = 1$  情况下的仿真结果

当  $a = 0$ ,  $b = 20$  时, 因为  $x^2$  的系数为 0, 仿真图像为一个常数, 如图 11.1-31 所示。

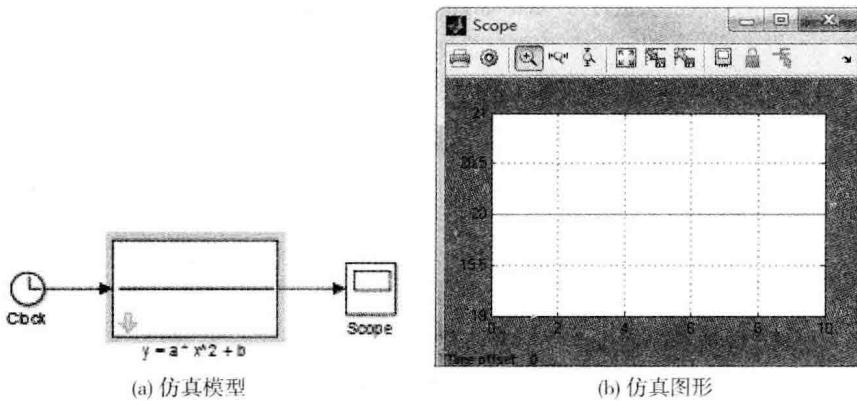


图 11.1-31  $a = 0$ ,  $b = 20$  情况下的仿真结果

## 11.1.2 封装 S 函数编写的模块

封装 S 函数编写的模块跟封装 Simulink 标准模块组成的子系统有相同的过程。以 11.1.1 小节所述同一个数学表达式为例, 使用笔者自制工具 Level 1 M S-Function Generator V1.1 自动生成 S 函数框架后, 直接编写算法构成完整的 Level 1 M S 函数。此工具的 GUI 界面如图 11.1-32 所示。

$y = a * x^2 + b$  这个数学模型有 1 个输入, 1 个输出, 2 个参数  $a$ 、 $b$ , 封装参数变量名仍采用  $g_a$ ,  $g_b$ 。输入输出为直接嵌入关系, 整个模型在每个步长内都在 Output 子方法里计算更新, 因是直接嵌入的, 故不需要状态变量, Update 中也不需要更新各个量的值。

关于此工具的详细功能的介绍, 请参考第 10.5.1 小节。通过工具的生成, 很快能够得到这个 S 函数, 代码如下:

```
function [sys,x0,str,ts,simStateCompliance] = sfun_mask(t,x,u,flag,g_a, g_b)
switch flag,
case 0,
[sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
```

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

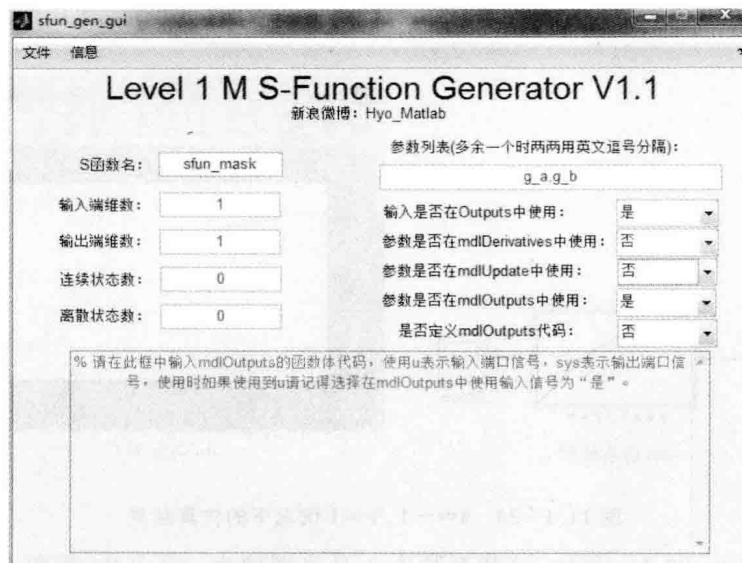


图 11.1-32 S-Function Generator V1.1

```

case 1,
sys = mdlDerivatives(t,x,u);
case 2,
sys = mdlUpdate(t,x,u);
case 3,
sys = mdlOutputs(t,x,u,g_a, g_b);
case 4,
sys = mdlGetTimeOfNextVarHit(t,x,u);
case 9,
sys = mdlTerminate(t,x,u);
otherwise
DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end
function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0,0];
simStateCompliance = 'UnknownSimState';
function sys = mdlDerivatives(t,x,u)
sys = [];
function sys = mdlUpdate(t,x,u)
sys = [];
function sys = mdlOutputs(t,x,u,g_a, g_b)
sys = g_a * u^2 + g_b;

```

```

function sys = mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;
sys = t + sampleTime;
function sys = mdlTerminate(t,x,u)
sys = [];

```

代码保存在文件 sfun\_mask.m 中,从 Simulink Browser 中拖出 S-Function 模块,右击模块,单击 Block Parameter(S 函数)打开图 11.1-33 所示对话框,填入 S 函数名及参数名。

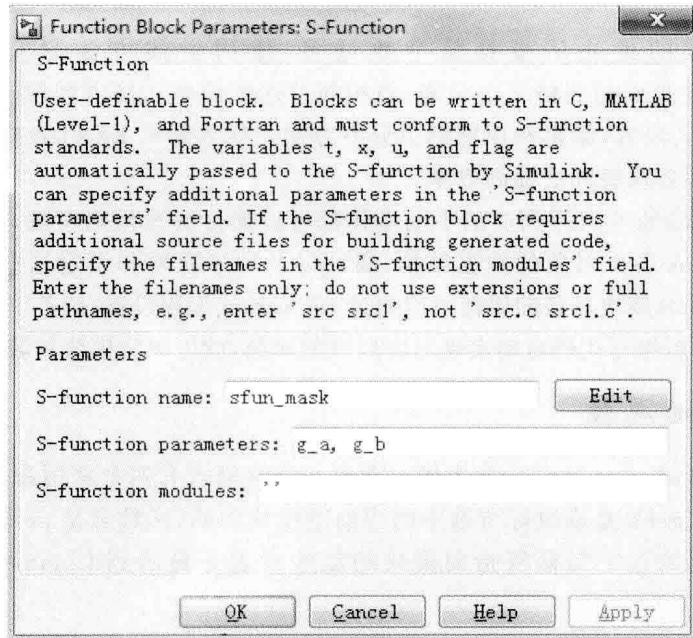


图 11.1-33 S 函数参数对话框

这样,S 函数模块就建立好了,右键此模块选择 Mask,按照 11.1.1 小节所述步骤封装即可。模块外观采用 plot 绘制  $y=a \times x^2+b$  波形而非直接封装图片。双击 S-Function 之后,在弹出的对话框中输入参数 a 和 b 的值 2 和 56,图 11.1-34 所示为封装完成之后的效果图,单击仿真按钮待仿真完成之后,可以观察 S 函数计算出来的图形。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

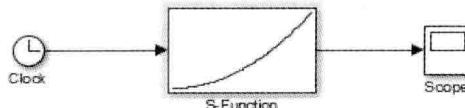
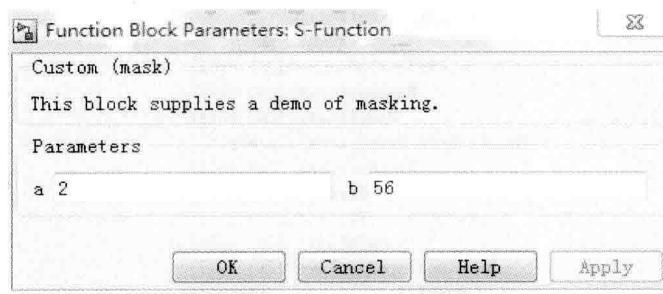


图 11.1-34 封装的 S 函数模块和对话框

参数较少时,可以采用手动方式封装。但当采用 S 函数开发硬件外设驱动模块(Simulink Peripheral Driver blocks)时,需要将带有复杂寄存器的外设参数全部体现在模块的对话框中,数十甚至上百个参数,一一手动封装甚是麻烦,因此需要开发另外的自动化工具协助封装过程,电脑最适合的就是这种重复的劳动了, MATLAB 也不例外。下面看看如何通过编程自动封装模块。

## 11.2 编程自动封装模块

当所需要封装的模块所带参数个数过多,使用上述方法封装耗时耗力,而且 Parameters & Dialog 页面的表格又小又密,当相邻且连续的单元格内容相同时,也必须一个一个地选中单元格输入内容,编辑和识别都十分不方便。如果能像 Excel 表格那样方便就好了。为了使用更加便利,必须想办法提高效率。

MATLAB/Simulink 中几乎所有的手动操作都可以通过编写代码实现,从手动到自动化的过程,需学习 Simulink 各个对象的数据结构、属性以及用于获取和设定这些数据结构、属性的 API 函数等。Simulink 模块具备的属性可以通过 set\_param 编程设置,倘若了解了模块属性中与封装相关的部分,然后编写代码自动实现对应属性的赋值,就可以将模块封装过程自动化。

### 11.2.1 模块的属性

对于一个模块(以  $y = a * x^2 + b$  为例),获取其属性列表有两个常用命令 get(gcbh) 和 inspect(gcbh)。函数 gcbh 是指鼠标所选中的当前的模块句柄,函数名是 get current block handle 的缩写, get() 返回这个句柄所指向模块的属性列表并显示到 Command Window 中,如图 11.2-1 所示。

```

MaskVarAliasString: ''
MaskInitialization: ''
MaskSelfModifiable: 'off'
    MaskDisplay: [1x151 char]
    MaskIconFrame: 'on'
    MaskIconOpaque: 'on'
    MaskIconRotate: 'none'
    MaskPortRotate: 'default'
    MaskIconUnits: 'autoscale'
    MaskValueString: '1|20'
MaskRunInitForIconRedraw: 'off'
    MaskTabNameString: ''
        Mask: 'on'
    MaskCallbacks: {2x1 cell}
        MaskEnables: {2x1 cell}
        MaskNames: {2x1 cell}
    MaskPropertyNameString: 'g_a|g_b'
        MaskPrompts: {2x1 cell}
        MaskStyles: {2x1 cell}
    MaskTunableValues: {2x1 cell}
        MaskValues: {2x1 cell}
MaskToolTipsDisplay: {2x1 cell}
    MaskVisibilities: {2x1 cell}
    MaskVarAliases: {2x1 cell}
    MaskWSVariables: [1x2 struct]
        MaskLabNames: {2x1 cell}
        MaskObject: []
    Ports: [1 1 0 0 0 0 0 0]

```

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 11.2-1 模块的属性显示到 Command Window 中

`inspect()`则将这个句柄所对应的属性显示到 `inspector` 列表中,如图 11.2-2 所示。

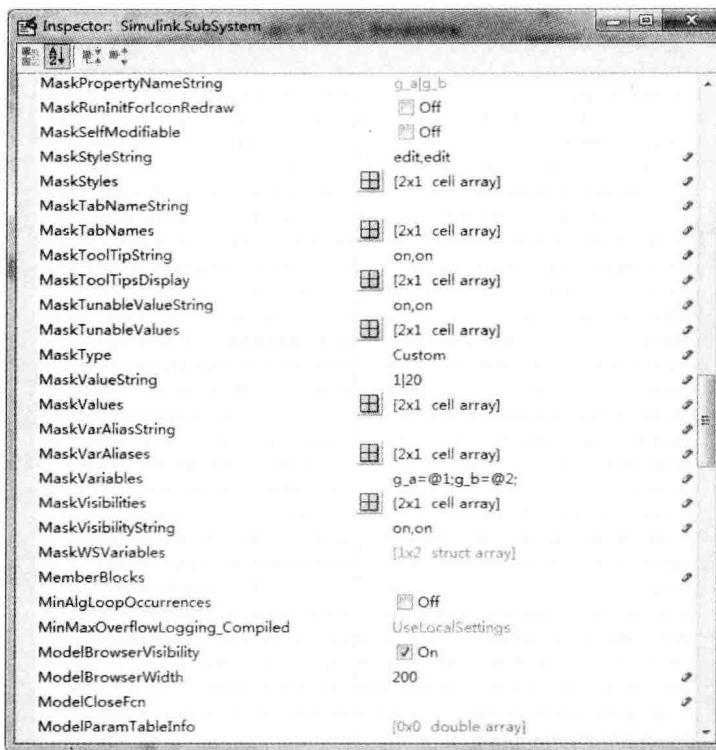


图 11.2-2 模块的属性显示到 `inspector` 列表中

相对于 `get` 返回的结果,`inspect` 所获取的内容更直观更便于了解其内容,如 `MaskValues`,直接单击属性右方  即可观察其内容,从值上可以推测出,表中内容即为参数 a, b 的值,如图 11.2-3 所示。

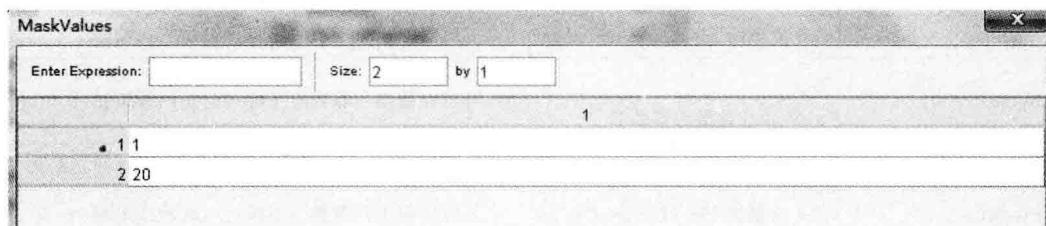


图 11.2-3 Mask Value 的内容

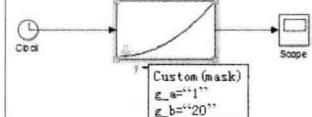
很容易发现,这些以 `Mask` 开头的属性都是跟封装息息相关的,而且属性名本身比较容易辨认,如果不确定可以单击 `Inspector` 中的黄色田字形按钮打开窗口查看内容以辨别其含义。将  $y = a * x^2 + b$  子系统中与封装相关的参数名及对应意义总结如表 11.2-1 所列。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

表 11.2-1 Inspector 属性列表

| 属性名                          | 属性的值                                     | 说明                                |
|------------------------------|--|-----------------------------------|
| <code>MaskType</code>        | 'Custom'                                 | Mask Editor 的 Documentation 页面的参数 |
| <code>MaskDescription</code> | 'This block supplies a demo of masking.' | Mask Editor 的 Documentation 页面的参数 |

续表 11.2-1

| 属性名                    | 属性的值   | 说 明  |
|------------------------|--|--|
| MaskHelp               | 'helpview(''xxx.html'');   | Mask Editor 的 Documentation 页面的参数  |
| MaskPromptString       | 'a  b'   | 参数的说明字符串,不同参数说明字符之间使用“ ”分隔   |
| MaskStyleString        | 'edit,edit'  | 参数的控件类型字符串,两两之间使用“,”分隔   |
| MaskVariables          | 'g_a=@1;g_b=@2;'   | 参数变量名及其编号的字符串,两两之间使用“;”分隔  |
| MaskTunableValueString | 'on,on'  | 表示参数是否在仿真时可调,使用“,”分隔   |
| MaskCallbackString     | 'check_num(''g_a'');   check_num(''g_b'');'  | 表示参数 Callback 调用的内容字符串,参数之间使用“ ”分隔   |
| MaskEnableString       | 'on,on'  | 表示参数是否使能的字符串,参数之间使用“,”分隔   |
| MaskVisibilityString   | 'on,on'  | 表示参数是否可见的字符串,参数之间使用“,”分隔   |
| MaskToolTipString      | 'on,on'  | 表示当鼠标停留在模块外观框范围内时是否在提示框显示参数值,设为 on 时如下图:<br> |
| MaskInitialization     | ''   | Initialization Command 的内容字符串  |
| MaskDisplay            | a = str2num(get_param(gcbh, 'g_a')); b = str2num(get_param(gcbh, 'g_b')); t = 0 : 0.1 : 10; plot(t, a * t.^2 + b); | Icon Draw Commands 的内容字符串  |
| MaskValueString        | '1 20'   | 模块封装参数的当前值,用“ ”分隔  |
| MaskCallbacks          | Cell 类型的 MaskCallbackString  | 可以使用元胞数组下标进行访问或编辑  |
| MaskEnables            | Cell 类型的 MaskEnableString  | 可以使用元胞数组下标进行访问或编辑  |
| MaskNames              | 'g_a'<br>'g_b' Cell 类型的参数名   | 可以使用元胞数组下标进行访问或编辑  |
| MaskPropertyNameString | g_a g_b  | 字符形式,将参数变量名用“ ”分隔并连接起来   |
| MaskPrompts            | Cell 类型的 MaskPromptsString   | 可以使用元胞数组下标进行访问或编辑  |
| MaskStyles             | Cell 类型的 MaskStylesString  | 可以使用元胞数组下标进行访问或编辑  |
| MaskTunableValues      | Cell 类型的 MaskTunableValuesString   | 可以使用元胞数组下标进行访问或编辑  |
| MaskValues             | Cell 类型的 MaskValueString   | 可以使用元胞数组下标进行访问或编辑  |
| MaskToolTipsDisplay    | Cell 类型的 MaskToolTipsString  | 可以使用元胞数组下标进行访问或编辑  |
| MaskVisibilities       | Cell 型的 MaskVisibilitiesString   | 可以使用元胞数组下标进行访问或编辑  |
| MaskTabNames           | Cell 型数组,每个元素为对应参数所在 tab 页面的名字   | 可以使用元胞数组下标进行访问或编辑  |

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

## 11.2.2 使用 set\_param 和 get\_param 封装模块

这两个函数是 Simulink 相关编程中使用频率最高的函数。模型仿真的启动和停止、模块的属性和封装，甚至 Simulink Configuration Parameter 的属性都可以通过 get\_param 获取，并通过 set\_param 对其中可写的参数进行编辑。

get\_param 通常有 2 个输入参数，1 个返回参数：

```
ParameterValue = get_param(Object, ParameterName)
```

Object 表示模块或者模型的句柄或模型模块路径；

ParameterName 表示模块或者模型的属性，如表 11.2-1 所列的属性。

如对于封装后的  $y=a \times x^2+b$  子系统，使用名字获取其 MaskVisibilities 属性的语句：

```
MaskVisibilities = get_param('demo1/y = a * x^2 + b','Maskvisibilities')
```

MaskVisibilities 则成为一个包含两个字符串成员的元胞结构，如果希望改变此属性，则使用 set\_param 函数，格式如下：

```
set_param(object,param,value)
```

object 表示模块或者模型的句柄或带路径的名称；

parameter 表示模块或者模型的属性，如上表中列出的属性；

value 表示希望写入 parameter 属性的值。

parameter 和 value 可以多组出现，即 set\_param 的参数组数不限，对于 MaskVisibilities 来说，需要写入一个具有跟封装参数相同数目的并且以 'on'，'off' 为内容的数组：

```
set_param('demo1/y = a * x^2 + b','Maskvisibilities', {'on','off'})
```

在 Command Windows 里运行之后，双击  $y=a \times x^2+b$  模块可见第 2 个参数 b 已经被隐藏，其可见性设为了 off。S 函数的参数对话框如图 11.2-4 所示。

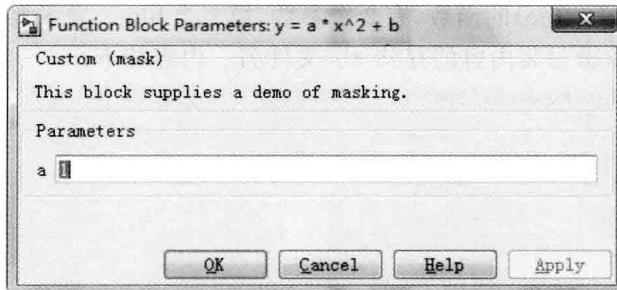


图 11.2-4 设置 MaskVisibilities 属性之后的参数对话框

类似地，MaskStyles、MaskValue、MaskPrompts 和 MaskVariables 等也可以通过 set\_param 进行设置。封装参数较多时，在 Mask Editor 中一个一个书写出来相当费神，并且对整个模块的属性参数进行修改时，每个属性要求的格式不同，有的是字符串，有的要求元胞形式，一个一个排列下来眼都会看晕，直接对参数的各种属性进行编程，其正确性和效率很难保证。所以笔者提倡采用 Excel 表替代 Mask Editor 进行封装，算是一个改进的办法。例如设计如图 11.2-5 所示表格存储各类封装信息：

表格的首列是每个参数的标签名 Prompt，第 2 列是变量名，第 3 列是封装控件类型（edit、popupmenu、checkbox、radiobutton 4 种），接下来的 Popups 列仅对 popup 和 radiobutton 两个类型的控件有效，作为其下拉框内容，两两之间使用“|”分隔。Tab 列是标签页的名字，F

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

| A           | B         | C           | D                                       | E   | F                                |
|-------------|-----------|-------------|---|-----|----------------------------------|
| MaskPrompts | MaskNames | MaskStyles  | Popups                                  | Tab | Callbacks                        |
| 1 prompt1   | g_var1    | edit        |   | t1  |                                  |
| 2 prompt2   | g_var2    | popup       | StringA StringB StringC StringD StringE | t1  |                                  |
| 3 prompt3   | g_var3    | checkbox    |   | t1  |                                  |
| 4 prompt4   | g_var4    | edit        |   | t2  |                                  |
| 5 prompt5   | g_var5    | radiobutton | One Two Three Four Five                 | t2  |                                  |
| 6 prompt6   | g_var6    | checkbox    |   | t2  | disp(get_param(gcbh, 'g_var5')); |

图 11.2-5 直接在 Excel 里面来编辑封装信息

列是每个参数的CallBack 编辑框,可以直接写入参数的回调函数内容,或者写入调用语句。

有了Excel 表格作为编辑工具,当需要封转的参数相邻顺次+1 或者是完全一致的变量名时,直接使用鼠标选中第一个单元格后下拉即可(二者区别就是其中一个需要按住Ctrl 键)。如图 11.2-6 所示。

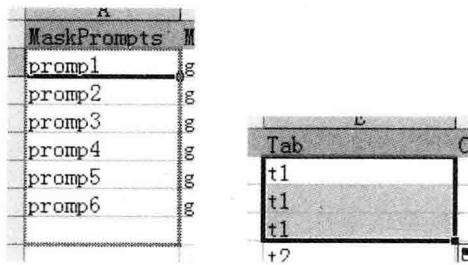


图 11.2-6 Excel 快速编辑内容

使用 Excel 可以方便地改变文字大小、字体颜色和单元格背景色,根据使用用途不同方便地进行标注,很容易观察分类文字内容。借助 Excel 的长处,封装的参数越多越能体现相对于直接使用 Simulink Mask Editor 的优势。

有了表格之后,还需要编写代码将 Excel 表格的内容封装到模块上去,笔者将代码写成名为 MaskTool(block, excelfile) 的函数,有 2 个参数,第 1 个 block 表示要封装的模块对象,第 2 个参数 excelfile 表示编辑封装内容的存储 xls 文件名。内容如下:

```
function masktool(block, excelfile)

% column index definition
promts_index = 1;
names_index = 2;
styles_index = 3;
popups_index = 4;
tabs_index = 5;
callbacks_index = 6;

% get data in excelfile
[num, str] = xlsread(excelfile);

% get mask prompts and saved as cell format
promts = str(2:end,promts_index);

% get mask names and saved as cell format
names = str(2:end,names_index);
% change names to MaskVariables format
len = length(names);
maskvar = '';
```

```

for ii = 1: len
    maskvar = [maskvar, names{ii}, '= @', num2str(ii), ';']; % OK
end

% get mask styles and saved as cell format
styles = str(2:end, styles_index);

% get mask tab names
tabname = str(2:end, tabs_index);

% get callbacks strings
callbackstr = str(2:end, callbacks_index);

% get block's handle
blockhandle = get_param(block, 'Handle');

% get model properties
prop = get(blockhandle);

% mask prompts into block
set_param(blockhandle, 'MaskPrompts', prompts);

% mask names into block (MaskNames or MaskPropertyNameString is a readonly property)
% set_param(blockhandle, 'MaskNames', names);
set_param(blockhandle, 'MaskVariables', maskvar);

% mask styles intoblock
set_param(blockhandle, 'MaskStyles', styles);

% check popup to write strings
style = str(2:end, styles_index);

% get popup strings
popups = str(2:end, popups_index);

% get MaskStyleString
stylestring = get_param(blockhandle, 'MaskStyleString');

% get empty start index from MaskStyleString and its length and its total number
emptystart = regexp(stylestring, '<empty>');
lenempty = length('<empty>');
if ~isempty(emptystart)
    numtorep = length(emptystart);

    % draw popup content from excelfile
    content = cell(numtorep);
    count = 1;
    num = length(style);
    for ii = 1:num
        if strcmp(style{ii}, 'popup')
            content{count} = str(ii + 1, popups_index);
            % content keep format as xxx|xxx|xxx
            count = count + 1;
        end
    end
end

```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```

    end
end
end

% part stylestring to seperate parts
% create cell using() not {}
if ~isempty(emptystart)
    strpart = cell(numtorep + 1);
    strpart{1} = stylestring(1:emptystart(1) - 1);
    for ii = 2: numtorep
        strpart{ii} = stylestring(emptystart(ii - 1) + lenempty:emptystart(ii) - 1);
    end
    strpart{numtorep + 1} = stylestring(emptystart(end) + lenempty: end);

    % insert content into strpart
    maskstring = strpart{1};
    for ii = 1: numtorep
        maskstring = strcat(maskstring, content{ii});
        maskstring = strcat(maskstring, strpart{ii + 1});
    end

    % cell changed to string
    stylestring = char(maskstring);
    % delete char(10)(which is not \n or \r)
    stylestring(findstr(stylestring, char(10))) = ""; % delete char(10) change line token
    % set MaskStyleString into block
    set_param(blockhandle, 'MaskStyleString', stylestring);
end

% set tab names
set_param(blockhandle, 'MaskTabNames', tablename);

% set MaskCallbacks
set_param(blockhandle, 'MaskCallbacks', callbackstr);

% get block name
modelname = get_param(blockhandle, 'Name');
% write Parameters automatically
paramstr = get_param(blockhandle, 'MaskPropertyNameString');
paramstr(findstr(paramstr,'|')) = ',';
set_param(blockhandle, 'Parameters', paramstr);

% write S-Function name automatically
set_param(blockhandle, 'FunctionName', excelfile);

% print OK info
disp(['modelname, ' Mask Over! ']);

```

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

将此函数保存为同名 M 文件，存放在已经添加到 MATLAB Search Path 的文件夹下即可使用。从 Simulink Browser 中拖出一个 S-Function 模块，选中之后在 Command Window 中输入 masktool(gcbh, 'block.xls')；则 Command window 中出现图 11.2-7 所示字样说明封装完毕。

```
>> masktool(gcbh, 'BlkName.xls');
S-Function Mask Over!
```

图 11.2-7 使用 MaskTool 封装完毕

双击 S-Function 模块可以看到,图 11.2-5 所示内容已封装到对话框中,如图 11.2-8 所示。

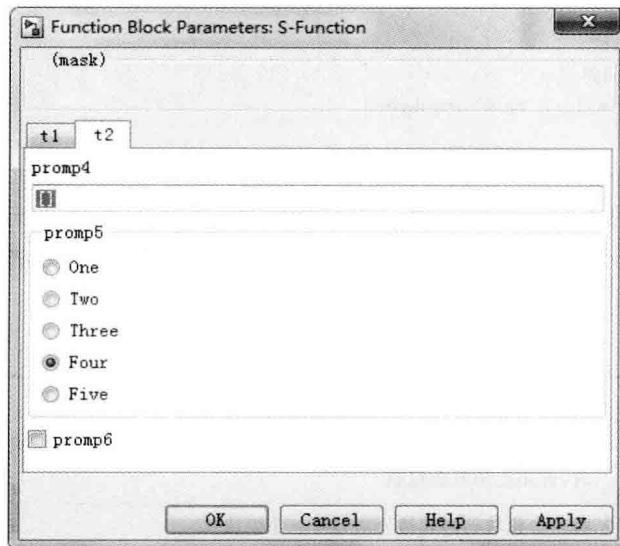


图 11.2-8 自动封装的对话框

使用 masktool 这样自定义的自动化工具进行编辑和封装,比 Simulink Mask Editor 方式有效率易于接受。

### 11.2.3 使用 Simulink. Mask 类封装模块

Simulink 提供了一些参数和函数帮助用户进行模块的封装。如 11.2.2 小节通过 set\_param、get\_param 函数来查询或设置模块参数。但是这两个函数不支持 Unicode 字符,使用受到限制。为了更广泛地支持各种语言环境,用户可以使用类 Simulink. Mask 与 Simulink. MaskParameter 的实例进行编程式控制。

get\_param, set\_param 函数通过模块的参数字符串进行属性值的获取或设定。Simulink. Mask 类通过给模块创建一个封装对象,再通过对对象的子方法来操作其各个属性的。Simulink. Mask 类提供的方法包括以下操作:

- ① 创建、拷贝、删除模块的封装。
- ② 创建、拷贝、删除模块封装的参数。
- ③ 决定封装的对象模块。
- ④ 获取封装时的工作空间变量。

获取一个已经被封装过的模块的封装属性,可以通过 Simulink. Mask 类的 get 子方法:

```
maskObj = Simulink.Mask.get(gcbh)
```

获取的属性全部存入 maskObj 这个变量中,对于  $y=a \cdot x^2 + b$  获取内容如下:

```
Type: 'Custom'
Description: 'This block supplies a demo of masking.'
```

```

Help: 'helpview(''xxx.html'');
Initialization: ''
SelfModifiable: 'off'
Display: [1x151 char]
IconFrame: 'on'
IconOpaque: 'on'
RunInitForIconRedraw: 'off'
IconRotate: 'none'
PortRotate: 'default'
IconUnits: 'autoscale'
Parameters: [1x2 Simulink.MaskParameter]
BaseMask: []

```

可以从属性名看出, Mask Editor 中需要填入的内容全部以属性的方式整理到变量 maskObj 中, 便于编程时的访问和设定管理。之后的操作, 就无须再通过 Simulink. Mask 调用子方法了, 直接使用 maskObj 这个对象变量访问 Simulink. Mask 的子方法即可, Simulink. Mask 的子方法如表 11.2-2 所列。

表 11.2-2 Simulink. Mase 类方法列表

| 子方法名                  | 功 能  |
|-----------------------|--|
| addParameter          | 向封装中增加一个参数, 参数名与值成对输入, 当所封装模块与 Simulink 标准库有关联时, 使用此函数会报错              |
| copy                  | 将一个模块的封装拷贝到另外一个模块  |
| create                | 为未封装的模块创建一个封装对象  |
| delete                | 将模块解封装并删除封装对象变量  |
| get                   | 获取模块的封装并存为一个对象变量   |
| getDialogControl      | 查找封装对象中的控件, 参数为控件的变量名 Name, 返回两个参数分别是被查找对象变量及其父对象变量                    |
| isMaskWithDialog      | 当前对象是否封装了对话框   |
| getOwner              | 获取当前封装所依附的模块   |
| getParameter          | 获取某个参数的属性, 输入参数为这个参数的 Name   |
| getWorkspaceVariables | 获取当前封装所使用的所有参数并将参数名与值作为结构体变量返回   |
| numParameters         | 返回封装中参数的个数   |
| removeAllParameters   | 删除封装中的所有参数, 一般不建议使用。当删除的 Mask 的 Owner 与 Simulink 标准库中存在链接关系的话, 删除是被禁止的 |
| set                   | 设置封装某个(或某些)属性的值, 用法类似 set_param, 属性名与值成对输入                             |
| addDialogControl      | 为父对象添加控件, 父对象可以为模块的封装对象, 也可为 group 和 tab container 等能够包容子空间的控件对象       |

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

Simulink. MaskParameters 类是 Simulink. Mask 的一个子类, 对 Simulink. Mask 类对象获取 Parameter 属性时, 得到的结果就是一个 Simulink. MaskParameters 对象。它仅有一个方法 set, 用来对参数属性进行设置, 功能与 Simulink. Mask 对象的 set 子方法一致, 此处不赘述。使用 Simulink. Mask 类封装基本模块为子系统与 S 函数分别介绍如下:

### 1. 封装 Simulink 子系统

编程封装过程思路上与手动封装相同, 只不过是将手动操作的步骤使用代码的形式表现

并执行。仍然以  $y = a * x^2 + b$  这个数学模型为例, 如图 11.2-9 所示。

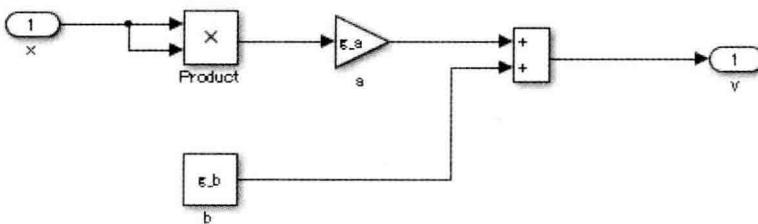


图 11.2-9  $y = a * x^2 + b$  的 Simulink 模型

将所有模块和信号连线选中, 创建子系统, 并使用鼠标单击选中这个子系统, 以便后续操作使用 gcbh 函数直接访问此模块。

首先为子系统建立一个封装对象, `maskObj = Simulink.Mask.create(gcbh)`, 如果 `gcbh` 表示的模块已经封装过了, 则使用 `maskObj = Simulink.Mask.get(gcbh)` 来获取这个封装对象。封装对象的成员包罗了 Mask Editor 上 4 个页面所有的信息。`maskObj` 的所有成员如表 11.2-3 所列。

表 11.2-3 maskObj 的成员列表

| 属性成员名                | 说 明   |
|----------------------|---|
| Type                 | 内容为 Mask Type 的字符串                                  |
| Description          | 内容为 Mask Description 的字符串                           |
| Help                 | 内容为打开 help 文档的代码                                    |
| Initialization       | 内容为初始化命令  |
| SelfModifiable       | 是否使能模块修改自身内容, 如 popup 的下拉菜单内容在参数 Callback 中被改变      |
| Display              | 内容为绘制模块外观的代码  |
| IconFrame            | IconFrame 是否可见                                      |
| IconOpaque           | 模块图标是否透明  |
| RunInitForIconRedraw | 在执行模块外观绘制前是否必须执行初始化命令                               |
| IconRotate           | 模块外观图标是否跟随模块一起旋转                                    |
| IconUnits            | 模块外观图标的单位设置, 共 3 种 'pixel' 'autoscale' 'normalized' |
| Parameters           | 封装参数集合, 类型为 Simulink.MaskParameters                 |
| BaseMask             | 未使用属性   |

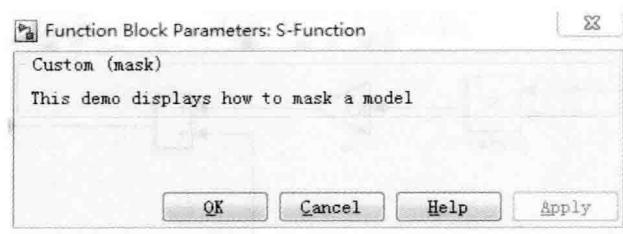
使用 `maskObj` 及点操作符访问其各成员属性, 将 `MaskType`、`MaskDescription` 及模块外观看赋值进去:

```
maskObj.Type = 'Custom';
maskObj.Description = 'This demo displays how to mask a model';
maskObj.Display = 't = 0:0.1:10; y = t.^2 + 1; plot(t,y)';
```

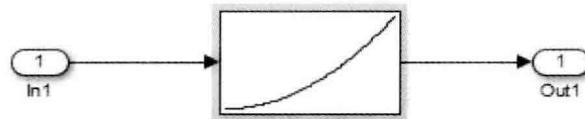
子系统获得了双击可弹出对话框, 以及直接显示其数学原理的模块外观。模块的参数对话框和模块外观如图 11.2-10 所示。

为对话框添加参数, 需使用 `Simulink.Mask` 类的子方法 `addParameter`。添加 2 个参数 `g_a` 和 `g_b`, 控件类型均为 `edit`, 提示性文字显示为 `a` 和 `b`, 且初始值均为 0:

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。



(a) 参数对话框



(b) 模块外观模型

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 11.2-10 封装了 MaskType, MaskDescription 及模块外观的模型

```
g_a = maskObj.addParameter('Type','edit','Name','g_a','Prompt','a','Value','0')
g_b = maskObj.addParameter('Type','edit','Name','g_b','Prompt','b','Value','0')
```

每一个参数的对象变量创建后,可以直接使用变量名和点操作符获取属性并编辑。如将参数 b 的值填写为 100:g\_b.Value='100';或者使用 Simulink. Mask 的子方法 getParameter 获取已经封装的参数:g\_a = maskObj.getParameter('g\_a'),返回值 g\_a 是 Simulink. MaskParameter 类型,可以调用该类型的 set 方法来设置其值:g\_a.set('Value','2')。

那么如果向模块的对话框追加标签页控件,需创建一个 tabcontainer 作为 tab 的容器:

```
tabcontainer = maskObj.addDialogControl('tabcontainer','g_container');
```

addDialogControl 这个函数包括 2 个参数,追加控件的类型名及其变量名。在 tabcontainer 中追加标签页控件也使用这个函数:

```
tab = tabcontainer.addDialogControl('tab','g_tab');
```

将追加的标签页命名为 t1:

```
tab.Prompt = 't1';
```

执行上述代码之后,双击 S-Function 模块打开其参数对话框如图 11.2-11 所示。

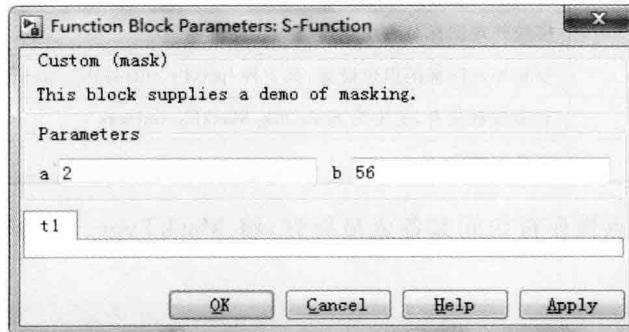


图 11.2-11 追加标签页之后的参数对话框

在追加的标签页中增加 Action 型控件 hyperlink 以给出笔者微博链接:

```
hyperlink = tab.addDialogControl('hyperlink','g_hyperlink');
hyperlink.Prompt = 'Hyo_MATLAB';
hyperlink.Callback = 'web "http://weibo.com/2300570331/" ';
```

创建之后的对话框如图 11.2-12 所示。

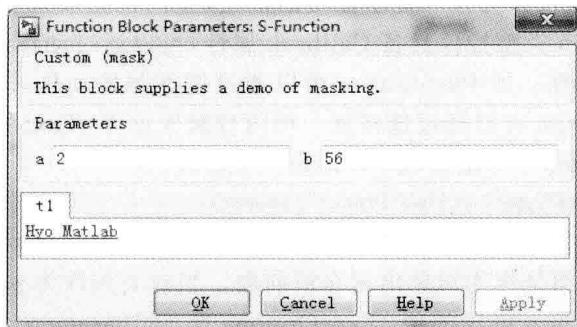


图 11.2-12 增加笔者微博的超链接

一个 tabcontainer 可以容纳多个标签,可追加一个带有按钮的页面,能打开 MATLAB 中文论坛:

```
tab2 = tabcontainer.addDialogControl('tab','g_tab2')
tab2.Prompt = 't2'
pushbutton = tab2.addDialogControl('pushbutton','g_pushbutton')
pushbutton.Callback = 'web "http://www.iloveMATLAB.cn/forum.php"';
pushbutton.Prompt = 'I Love MATLAB!';
```

运行上述代码后,双击模块打开封装对话框,如图 11.2-13 所示。

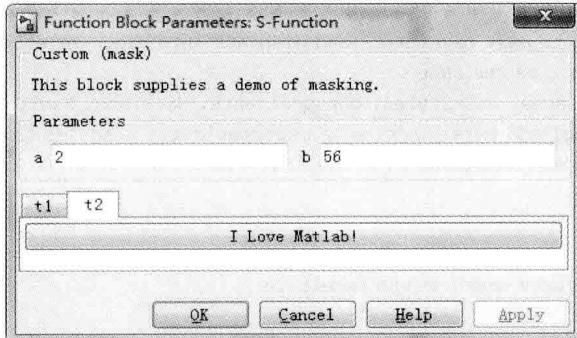


图 11.2-13 追加第 2 个标签页及按钮之后的对话框

单击 t1 的超链接和 t2 的按钮可分别打开笔者新浪微博首页及 MATLAB 中文论坛首页。

## 2. 封装 S 函数模块

11.2.2 小节“使用 set\_param 和 get\_param 封装模块”提出使用 Excel 表格代替 Mask Editor 编辑封装信息,但使用 set\_param 控制模块属性创建自动化封装过程时仅能创建 Parameter 类型的控件(edit、popup、radiobutton 和 checkbox),对于 Display 和 Action 控件无能为力。现在希望通过 Simulink. Mask 类将编写一个功能更加强大的 masktool\_neo.m, 以支持更多控件的创建。2013b 版本新增加的控件都能自动添加到对话框中。

使用工具 Simulink. Mask 类的封装函数 masktool\_neo, 依然使用 Excel 表格取代 Mask Editor, 进行参数的输入及控件的设计。

首先判别被封装的模块是否已经创建了封装, 如果已经封装过了则先解除其封装后再进行封装, 没有封装则需要创建一个空的封装对象, 然后再根据 Excel 内容分别添加信息。

为了支持 Parameter、Display 和 Action 3 种类型的控件, 将 Parameter 类型控件在 Mask

Editor 中编号,其变量名作为“实在”参数,可传递到 S 函数中计算;另外两类控件对应的变量名不传递到 S 函数中,仅仅作为控件显示在对话框上,提供视觉效果及控件的回调函数功能。Parameter 类型与 Action 类型控件具备 Callback 属性,可以通过动作触发某些回调函数,Display 不具有 Callback 属性。如 Pushbutton 可以通过按下触发动作,hyperlink 通过单击显示内容触发动作,radiobutton 可以通过选择某一项内容触发动作,Text 则不具有回调动作。

可使用 addParameter 子方法追加 Parameter 类型控件:

```
maskobj.addParameter('Type', style, 'Prompt', prompt);
```

方法调用时的参数为“属性名+属性值”配对出现,类似 set\_param 的调用方式,只不过对象 maskobj 不作为参数而是作为对象出现在最前面。当封装控件为 popup 或 radiobutton 时,需要在 TypeOptions 中写入可选的值,TypeOptions 是 addParameter 方法的一个参数,其设置值是 Popup 控件的内容,仅支持元胞数组型的字符串,通过字符串处理将 Excel 表格中以“|”分隔的字符串分割为多个 cell 字符串。

使用 addDialogControl 方法可添加其他两种类型的对话框控件:

```
maskobj.addDialogControl(style, names);
```

这个子方法在 MATLAB 中未提供文档说明,即所谓的 undocumented build-in function,通过尝试,总结出以上的调用方式,并加以应用。此方法的参数直接为封装控件类型(group、text、image、pushbutton 和 hyperlink 等)及控件的变量名。

具体代码如下:

```
function masktool_neo(block, excelfile)
%
% This function help to mask parameters and displays into block selected.
% MaskNames should not be the same
% MaskPrompts, MaskNames, MaskStyles, popupcontents, MaskTabs, MaskCallbackStrings
% are written in Excel and this function can automatically transfer them to
% the S-Function block. Block masked by this tool can not be edited in Mask
% Editor.
%
% block - the block need to be masked
% excelfile - parameters saved in the excelfile
%
% author: hyowinner
% history:
% 2014/08/25 - button, image, text, hyperlink, group supported
%
% index definition
promts_index = 1;
names_index = 2;
styles_index = 3;
popups_index = 4;
% tabs_index = 5;
callbacks_index = 6;

% get data in excelfile
[num, str] = xlsread(excelfile);
% get mask prompts and saved as cell format
promts = str(2:end,promts_index);
```

```
% get mask names and saved as cell format
names = str(2:end,names_index);
% change names to MaskVariables format

% get mask styles and saved as cell format
styles = str(2:end,styles_index);

% get mask tab names
% tabname = str(2:end, tabs_index);

% get TypeOptions
typeopts = str(2:end, popups_index);

% get callbacks strings
callbackstr = str(2:end, callbacks_index);

% test if gcbh is masked. If masked , delete the old mask and create a
% blank one.
maskobj = Simulink.Mask.get(block);
if ~isempty(maskobj)
  maskobj.delete;
end
maskobj = Simulink.Mask.create(block);

% parameter list value
p_str = [];

% get the object of groupbox "Parameter"
% par_obj = maskobj.getDialogControl('ParameterGroupVar');

% add controls according to the styles.
len = length(names);
for ii = 1:len
  if ismember(styles{ii}, {'text','group','tab'})
    % addDialogControl without callback
    prop = maskobj.addDialogControl(styles{ii},names{ii});
    prop.Prompt = promts{ii};
  elseif ismember(styles{ii}, {'pushbutton','hyperlink'})
    % addDialogControl for Action controls, they have Callback
    prop = maskobj.addDialogControl(styles{ii},names{ii});
    prop.Prompt = promts{ii};
    prop.Callback = callbackstr{ii};
  elseif ismember(styles{ii}, 'image')
    % addDialogControl without Prompt
    prop = maskobj.addDialogControl(styles{ii},names{ii});
  elseif ismember(styles{ii},{'edit','checkbox'}) % 'dial','spinbox','slider' only supported in
2014a and later not in 2013b
    % addParameter
    p_str = [p_str, names{ii}, ','];
    p = maskobj.addParameter('Type', styles{ii}, 'Prompt', promts{ii}, 'Name', names{ii},
'Callback', callbackstr{ii});
  elseif ismember(styles{ii}, {'popup','radiobutton'})
    % addParameter for parameters that have TypeOptions
```

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

p_str = [p_str, names{ii}, ','];
expression = '\|';
split_str = regexp(typeopts(ii), expression, 'split');
maskobj.addParameter('Type', styles{ii}, 'TypeOptions', split_str, 'Prompt', promts{ii},
'Name', names{ii}, 'Callback', callbackstr{ii});
end
end

% write S-Function name and parameter - list automatically
set_param(block, 'FunctionName', excelfile);
set_param(block, 'Parameters', p_str(1:end - 1));

disp('Current block is masked automatically.');

```

下面再以一个包含各种控件的 Excel 表格为例, 演示使用 masktool\_neo 封装模块的过程。建立的表格如图 11.2-14 所示。

| A           | B         | C           | D                       |
|-------------|-----------|-------------|-------------------------|
| MaskPrompts | MaskNames | MaskStyles  | Popups                  |
| prompt1     | g_var1    | checkbox    |                         |
| prompt2     | g_var2    | edit        |                         |
| prompt3     | g_var3    | text        |                         |
| prompt4     | g_var4    | edit        |                         |
| prompt5     | g_var5    | popup       | One Two Three Four Five |
| prompt6     | g_var6    | hyperlink   |                         |
| prompt7     | g_var7    | radiobutton | 星矢 紫龙 冰河 阿瞬 一辉          |
| prompt8     | g_var8    | text        |                         |
| prompt9     | g_var9    | image       |                         |
| prompt10    | g_var10   | edit        |                         |

图 11.2-14 待封装控件及属性信息的 Excel 表格

从 Simulink Library Browser 中拖出一个空的 S-Function 模块, 选中后在 Command Windows 中输入:

```
masktool_neo(gcbh, 'BlkName')
```

即得到封装好的对话框, 如图 11.2-15 所示。

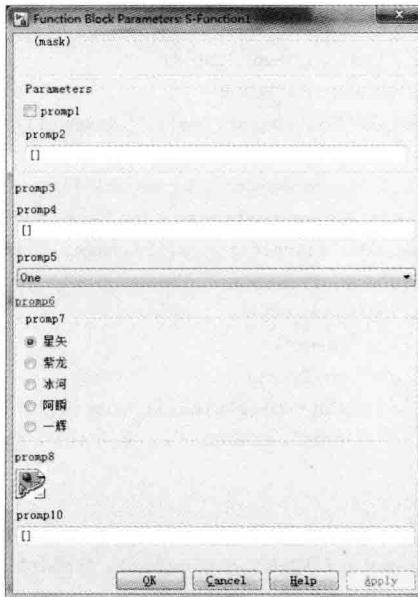


图 11.2-15 masktool\_neo 封装出的对话框

**注意：**通过 Simulink. Mask 类封装的控件如果不赋予其 Name 属性，重新打开 Mask Editor 进行编辑，这些参数的 Prompts 及 Name 属性都会自动命名为默认值出现重名情况，单击 OK 按钮保存封装之后，界面控件 Prompt 就会变得不受控制而导致杂乱无章，所以当读者朋友使用 Simulink. Mask 编写程序时，务必不要忘记给 Parameter 和 Control 的 Name(参数变量名)赋值。

若上例的参数名不赋值，封装出来的对话框变量名如图 11.2-16 所示。

| Type       | Prompt      | Name              |
|------------|-------------|-------------------|
| A          | %<MaskType> | DescGroupVar      |
| Parameters |             | ParameterGroupVar |
| #1         | prompt10    | MaskParam1        |
| #2         | prompt10    | MaskParam2        |
| A          | prompt3     | g_var3            |
| #3         | prompt10    | MaskParam3        |
| o          | prompt6     | g_var6            |
| #4         | prompt10    | MaskParam4        |
| A          | prompt8     | g_var8            |
| (N/A)      |             | g_var9            |
| #5         | prompt10    | MaskParam5        |

图 11.2-16 Name 未赋值的控件的 Prompt 自动设为同一值

读者朋友可以此为参考，根据各种使用场景总结出模式，并定制个性化的自动化封装工具。

### 11.3 使用 GUIDE 封装模块

MATLAB 中文论坛上有些坛友会抱怨说 Simulink 里没有提供读取 Excel 文件的模块，其实有了 S 函数与模块封装的技术做基础，不妨自己定制一个。使用 S-Function 模块结合 Level 1 M S 函数文件可实现能够选择 Excel 文件，并将其中数据按照采样时间顺序输出的功能。在模块的对话框中，将数据通过表格展示，并直接将 Excel 中数据的图像绘制出来。Simulink 自带的 Mask 模块没有提供坐标轴控件，GUIDE 中拥有很多功能强大的控件，可使用 GUIDE 实现模块的对话框，代替 Simulink 自身的封装功能。

首先设计一个界面，实现通过按钮打开 Window 对话框选择 Excel 文件，读取其数据内容，显示到表格中，并绘制图像等功能。结合 Simulink 模块的对话框风格，设计界面(sfun\_xlsread\_gui.fig)如图 11.3-1 所示。

按钮 Open 的回调函数实现以下一些功能：选择 Excel 文件，显示路径，读取数据显示到表格，以及在坐标轴中绘制图像的功能。GUIDE 的按钮可以随意控制显示位置和大小，这一点是 Simulink 模块的 Mask Editor 所无法比拟的。

模块的 S 函数需要一个保存文件路径信息的参数，命名为 g\_file\_path，内容是字符串，在 S 函数中封装为 edit 类型以传递此种类型数据。用户在文件选择对话框中选择文件后得到这个参数，并应用于 S 函数内部为模块的数据显示和绘图功能服务。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

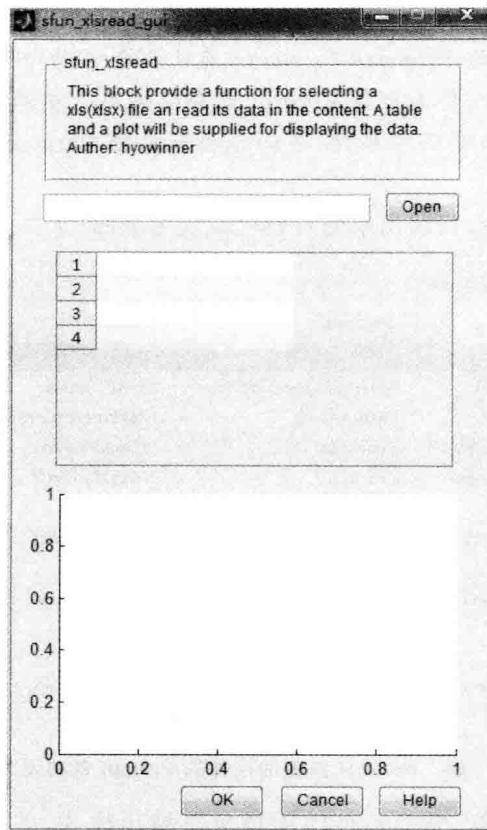


图 11.3-1 Excel 文件读取与显示用对话框

**注意：**不勾选文件路径的 Evaluate 属性，这个属性的意思是将参数内容作为参数调用 eval() 函数。因为参数的内容是一个字符串，不作为变量使用，不需要再进行一次 eval()。Mask Editor 中参数及属性设定如图 11.3-2 所示。

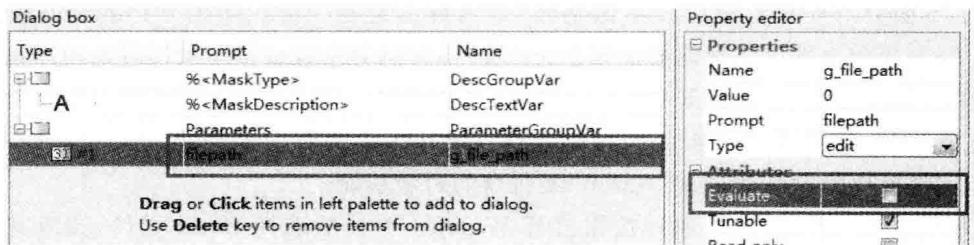


图 11.3-2 封装 edit 类型参数 g\_file\_path

封装之后，右击 S 函数模块的模块参数列表，填入 S 函数名和参数变量名，如图 11.3-3 所示。

为了使用刚创建图 11.3-1 所示对话框替代 Simulink 默认的对话框，可以将对话框作为该 S 函数模块的打开回调函数(OpenFcn)来启动。当双击 S 函数模块时，将启动 sfun\_xlsread-gui 界面作为对话框。右击模块打开 Property 属性对话框，并在 OpenFcn 栏中填入调用代码，如图 11.3-4 所示。

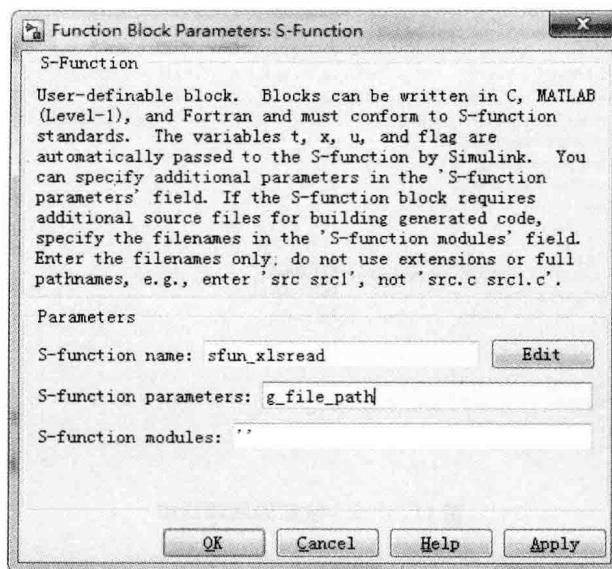


图 11.3-3 S 函数名与参数表

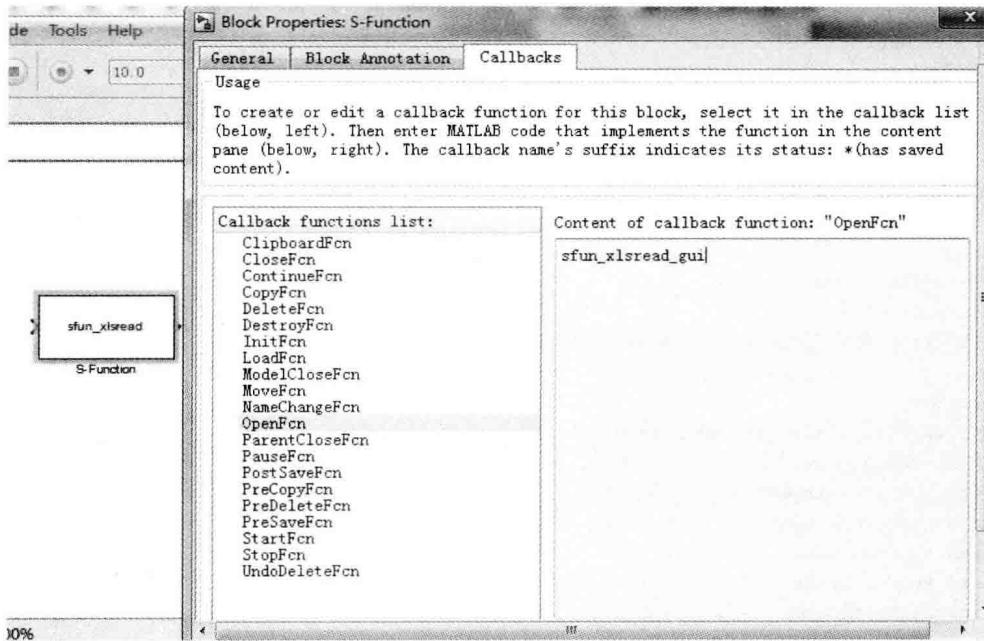


图 11.3-4 S 函数的 OpenFcn

有了回调函数的挂接,又有了 g\_file\_path 这个参数作为数据传递媒介,用户所选择的文件就能够传递到 S 函数中去了。按下图 11.3-1 中的 Open 按钮所执行的代码如图 11.3-5 所示。

S 函数的仿真功能:在每个采样时刻将对话框上所选择的 Excel 文件中的数据依次输出,如果仿真时间比较长,采样点数大于数据数,则最后一个数据点的数据持续输出。实现这个仿真功能的 S 函数采用 Level 1 M S 函数来编写:

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uigetfile(['*.xls','*.xlsx'], 'Select an Excel file');
if isEqual(filename, 0)
    set_param(gcbh, 'g_file_path', '');
    set(handles.edit1, 'string', '');
    disp('User selected Cancel');
    return;
else
    file_path = fullfile(pathname, filename);
    set_param(gcbh, 'g_file_path', file_path);
    set(handles.edit1, 'string', file_path);
end

[data, str] = xlsread(file_path);
set(handles.uitable1, 'data', data);
axes(handles.axes1);
bar(data, 'g');
```

图 11.3-5 按钮的回调函数

```
function [sys,x0,str,ts,simStateCompliance] = sfun_xlsread(t,x,u,flag,g_file_path)
switch flag,
case 0,
[sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes(g_file_path);
case 1,
sys = mdlDerivatives(t,x,u);
case 2,
sys = mdlUpdate(t,x,u);
case 3,
sys = mdlOutputs(t,x,u);
case 4,
sys = mdlGetTimeOfNextVarHit(t,x,u);
case 9,
sys = mdlTerminate(t,x,u);
otherwise
DASStudio.error('Simulink;blocks;unhandledFlag', num2str(flag));
end
function
[sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes(g_file_path)
global data len cnt
[data, str] = xlsread(g_file_path);
len = length(data);
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0,0];
cnt = 0;
simStateCompliance = 'UnknownSimState';

function sys = mdlDerivatives(t,x,u)
```

```

sys = [];
function sys = mdlUpdate(t,x,u)
sys = [];
function sys = mdlOutputs(t,x,u)
global data len cnt
cnt = cnt + 1;
if cnt <= len
    sys = data(cnt);
else
    sys = data(end);
end
function sys = mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;
sys = t + sampleTime;
function sys = mdlTerminate(t,x,u)
sys = [];

```

这个 S 函数可以通过第 10 章“S 函数”中介绍的 Level 1 M S 函数自动生成工具生成,再稍加修改即可。此模块无输入,无状态变量,具有一个输出端口用来输出 Excel 文件的数据。通过 g\_file\_path 读取的数据 data 作为全局变量,在初始化子方法和输出子方法之间传递。另外需要一个全局变量 cnt 进行采样点数的计数,在输出子方法中将对应当前采样点的数据输出。仿真用的模型由 sfun\_xlsread 模块直接输入连接 Scope 模块,运行仿真前双击模块打开参数对话框并选择一个内容为数据数组的 Excel 文件(目前此例仅适用于包含一维数据的 Excel 文件),并运行仿真。可以看到参数对话框中的坐标轴已经预读了 Excel 的数据内容,仿真时将此数组内容按照采样时间逐个输出到 Scope 中显示,如图 11.3-6 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

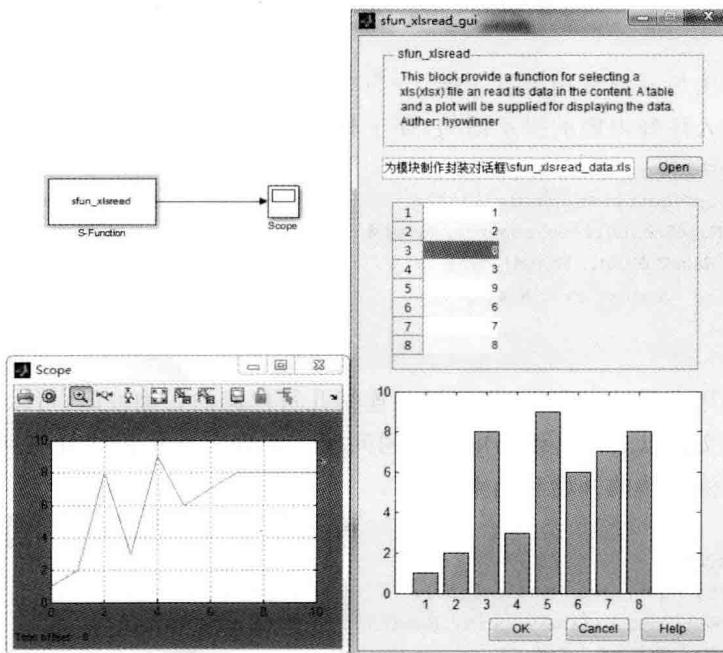


图 11.3-6 sfun\_xlsread 的对话框数据预览及仿真图

通过这个实例,用户可以扩展 Simulink 自定义对话框的功能和形式,借助 GUIDE 上丰富的控件来设计更加具有想象力的自定义 Simulink 模块。

# 第 12 章

## Publish 发布 M 文件

**引言:**在基于模型设计(MBD)广泛应用于汽车电子嵌入式开发的今天,人们也在尝试是否能将该技术推广到家用电器嵌入式控制方面。与传统的嵌入式开发相比,MBD以模型为核心,从算法设计到代码生成甚至工程编译一气呵成,以实现开发流程高度自动化。MATLAB/Simulink给用户提供了一整套可以自定义的自动化开发流程设计工具,这其中也包括了文档的自动生成。MATLAB自带的Help文档,易懂易读,深受MATLAB用户的喜爱。当读者自己开发M函数,自定义Simulink模块的时候,是不是也希望能够提供给使用者同样风格的帮助文档呢。MATLAB的publish函数结合合理的注释书写格式,可以帮你快速做成帮助文档。

### 12.1 M 文件的注释

MATLAB/Simulink工具是MathWorks公司开发的工程软件,在数学计算、快速原型、模型化可执行需求、产品级代码生成方面有很强的支持,已经深入到汽车电子及半导体开发行业中。M语言是MATLAB环境下的编程语言,属于解释性脚本语言,M语言编写的代码保存在.m文件中。

基本的注释方法与其他常用语言一样,分为单行注释和多行注释,其注释符为%。单行注释即直接在%后输入注释内容的提示语句,便于其他软件工程师阅读或维护,如:

```
for ii = 1:sizeLine
    lineProperty = get(line_Handles(ii));           % get the handle of line
    if strcmp(NameToFind,lineProperty.Name) % if line's name is the target
        disp(['NameToFind, ' Found! ']);          % print hint
        set(line_Handles(ii),'Name','');           % delete line's name
    end
end
```

多行注释主要用在调试过程中,当怀疑某连续几行代码的正确性时,可以将其暂时多行注释掉,再度执行M文件以测试是否仍出现相同问题。实现多行代码注释有两种方式,一种是使用%{和}%作为注释块的开始和结束,如下:

```
% {
for ii = 1:sizeLine
    lineProperty = get(line_Handles(ii));           % get the handle of line
    if strcmp(NameToFind,lineProperty.Name) % if line's name is the target
        disp(['NameToFind, ' Found! ']);          % print hint
        set(line_Handles(ii),'Name','');           % delete line's name
    end
end
}%
```

另一种是选中要注释的多行代码,按Ctrl+R,自动实现每一行前增加注释符%的效果:

```
% for ii = 1:sizeLine
%     lineProperty = get(line_Handles(ii));           % get the handle of line
%     if strcmp(NameToFind,lineProperty.Name) % line name is the target
%         disp(['NameToFind, ' Found! ']);            % print hint
%         set(line_Handles(ii),'Name','');             % delete line's name
%     end
% end
```

## 12.2 Cell 模式

MATLAB 的脚本文件中使用注释符还可以启动一种 Cell 模式。Cell 模式以两个连续的注释符为标志。每个`%%`都开启一个新的 cell 块, `%%`后空一格便可以追加注释语句。`%%`标注后所在行会以粗体字显示, 如图 12.2-1 所示。

```
1 %<< 初始化一个IE对象
2 ie = actxserver('internetexplorer.application');
3 % 打开新浪微博登录页面
4 ie.Navigate('http://weibo.com');
5 % 网页设置为可见
6 ie.Visible = true;
7 %% 等待网页加载完毕
8 while ~strcmp(ie.ReadyState, 'READYSTATE_COMPLETE')
9     pause(0.2);
10 end
11 %% 获取登录框输入内容并触发按钮事件
12 % 1 获取账户输入框并填入账户名
13 id_pl_login_form=ie.document.body.firstChild.lastChild.firstChild.nextSibling;
14 username=id_pl_login_form.firstChild.nextSibling.nextSibling.nextSibling.firstChild;
15 username.value='*****';
16 % 2 获取密码输入框并填入密码
17 password=id_pl_login_form.firstChild.nextSibling.nextSibling.nextSibling.firstChild;
18 password.value='*****';
19 % 3 获取登录按钮并触发点击事件 (与老版本不同之处)
20 submit_bt=id_pl_login_form.firstChild.nextSibling.nextSibling.nextSibling.firstChild;
21 submit_bt.click;
```

图 12.2-1 Cell 模式下的代码

图中分为 3 个 Cell 区域, 当前所选中的 Cell 区域以橙色背景显示, 其余都是普通的白色背景。在每一个 Cell 模块内, 仍然可以使用单行注释和多行注释。

Cell 模式的特点在于可以单独执行某一个 Cell 内的代码。例如在调试上述代码时, 需要在 M 文件打开的状态下, 通过脚本与 IE 进行互连传输。如果在调试过程中不小心关闭了 IE 浏览器, 不用将整个代码全部运行一遍, 只需要选中第 1 段(`%%`初始化一个 IE 对象)段的代码, 单击工具栏中 Run Section 图标即可重新打开 IE 并建立 MATLAB 与 IE 的连接, 如图 12.2-2 所示。

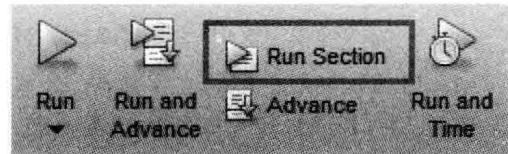


图 12.2-2 Run Tool Bar

Cell 模式的使用, 可以使脚本代码按照功能模块聚类为一个段, 增加了可读性, 而且可以以段为单位独立执行, 增加了调试的灵活性。

## 12.3 注释的 Publish

MATLAB 的注释方式丰富多彩, 以上只是两种常用的技巧。

大家在学习 building-in 函数的时候,经常会使用 doc xxx.m 来查询某函数的具体使用方法,doc 命令会打开 MATLAB 自带的 Help 文档,如图 12.3-1 所示。

### Examples

#### Example 1

##### Step Response Plot of Dynamic System

Plot the step response of the following second-order state-space model.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$y = [1.9691 \quad 6.4493] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
a = [-0.5572 -0.7814; 0.7814 0];
b = [1 -1; 0 2];
c = [1.9691 6.4493];
sys = ss(a, b, c, 0);
step(sys)
```

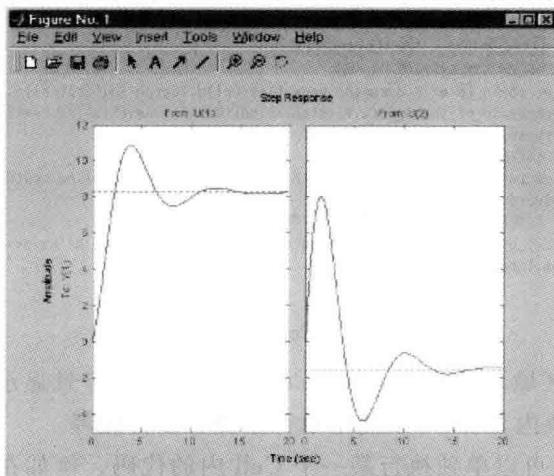


图 12.3-1 MATLAB 自带的 help 文档

MATLAB 自带文档的文字解释、公式说明和图片实例都很简洁明了,可以成为高效文档。资深 MATLAB 用户宁愿去读英文版的 Help 也不会选择简单翻译出来的 MATLAB 教材,更体现了其无法替代的地位。

作为 Simulink 系统建模的软件工程师,为了让工作得以延续,让同事或同伴在后续工作中能够了解你所做的模型和模块的功能,文档制作是必需的技能。制作文档的时候,都希望能够花较少的时间,做出整洁紧凑、条理分明的文档。如果文档制作也像写代码或建模一样游刃有余,那就太棒了。其实 Help 文档就是从代码中产生的。只不过,不是一般的代码,而是纯注释。

注释生成文档源自 MATLAB 的 Publish 功能,可将注释文件生成 .doc、.html 等常用格式文件。当我们按照一定的规则书写的时候,就可以自动生成所期望的漂亮规范的文本。而且,还能让不喜欢写文档的 coder 感受到比直接写 .doc 文档多一点的可操作性和乐趣。

### 12.3.1 正文

文本的正文标准首行必须使用 cell 形式,即`%%`开头。次行之后的文本只需要按照一般的注释格式书写,在`%`后书写正文即可。起始标题使用`%%`开头,正文书写使用单个`%`作为行开头,百分号之后与文字之间需要空一格。如果希望生成的文档中有换行,增加一行单独的`%`行,如图 12.3-2 所示。

```

1    %% Write document with comment
2    % File: sample_make_rtw_hook.m
3    %
4    % Abstract:
5    %
6    % This file is the hook file to do the extra operation for code generation
7    % at diffrent stage
8    %

```

图 12.3-2 用注释编写正文

将上述内容保存为 m 文件,经过 publish 命令之后,得到下面的 html 文档,如图 12.3-3 所示。

#### Write document with comment

File: sample\_make\_rtw\_hook.m

Abstract:

This file is the hook file to do the extra operation for code generation at diffrent stage

图 12.3-3 发布后的正文效果

### 12.3.2 字体控制

通过一些标识符可以实现一些简单的字体控制,如字体加粗,变成斜体等,如图 12.3-4 所示。

```

% _ITALICSTYLE_
% *BOLDSTYLE*
%
```

*ITALICSTYLE*

**BOLDSTYLE**

图 12.3-4 斜体字与粗体字

在需要处理的正文两端增加“\_”符号或者“\*”符号,在发布之后,可以得到斜体字和粗体字显示效果。

### 12.3.3 小标题

技术文档毕竟不是高考作文,只需要主标题和正文。技术文档往往要求简洁明了,思路清晰,按条书写。在`%`注释后增加`*`符号,即可生成点开头的标题;`%`后增加`#`符号,即可生成按

照数字排列的列表标题,单独的%行会打断数字的连续性,如图 12.3-5 所示。

发布后,可以得到如图 12.3-6 所示效果。

```
%  
% # ITEM1  
% # ITEM2  
%  
% * ITEM3  
% * ITEM4  
%
```

图 12.3-5 小标题书写格式

- 1. ITEM1
- 2. ITEM2
- ITEM3
- ITEM4

图 12.3-6 发布后小标题效果

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

### 12.3.4 插入超链接

文档中经常需要引入各种超链接,如网址、音频视频文件和 PPT 文件等,方法如下:

%之后首先写入网址 <http://www.ilovematlab.cn/>,并用<>括住,如此可以将网站地址以超链接方式显示在文档中。如果想通过文字而不是网址本身建立超链接,可以在网址后空一格,并追加希望显示的文字,如图 12.3-7 所示。

```
% Welcome to I Love Matlab, if you want learn more about Matlab/Simulink  
% just visit it.  
%  
% <http://www.ilovematlab.cn/ Matlab中文论坛>  
%
```

图 12.3-7 注释中插入网址

发布后,得到的文档效果如图 12.3-8 所示。单击“MATLAB 中文论坛”字样即可打开论坛如图 12.3-9 所示网站。

Welcome to I Love Matlab, if you want learn more about Matlab/Simulink just visit it.

MATLAB 中文论坛

图 12.3-8 发布后超链接效果

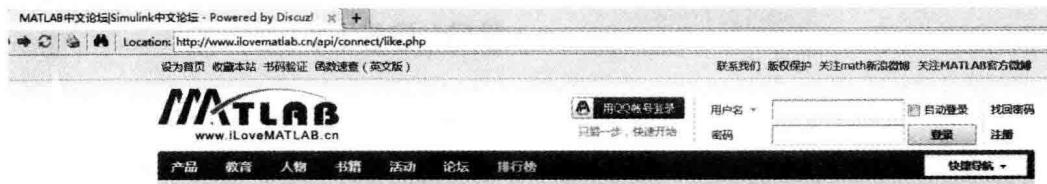


图 12.3-9 通过链接打开的网站

### 12.3.5 插入可执行代码

既然使用的是发布的功能,自然少不了可执行代码的发布。可以通过插入 MATLAB 脚

本命令实现各种控制。

```
% There is one example here.  
%  
% Please press <MATLAB:run('init_Simulation.m') Run Initiation Code> to run the  
% m script demo file.
```

发布后,得到带有链接的文本,单击链接即可执行脚本内容,如图 12.3-10 所示。

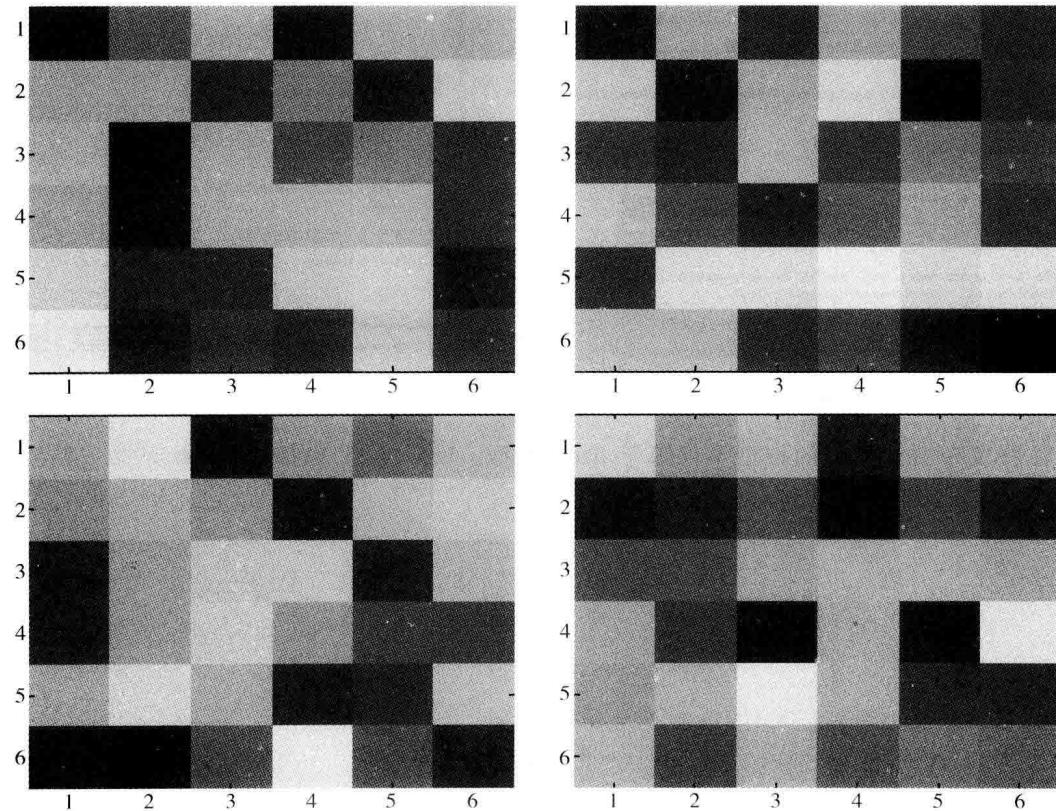
There is one example here.  
  
Please press Run Initiation Code to run the m script demo file.

图 12.3-10 带有可执行链接的效果

除了此种静态插入之外,还可以将代码运行过程中的动态结果插入,并发布出来。

```
for i = 1 : 4  
    imagesc(rand(6,6));  
    snapnow;  
end
```

代码就是通过循环动态产生 4 组  $6 \times 6$  的矩阵,然后作为图像打印出来。在每次的循环里使用 snapnow 命令可以将当前显示的图像发布在 html 文件中,如图 12.3-11 所示。



若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

便得多。而且当发布图像数目过多时,手动操作也是不可实现的。

## 12.4 注释发布功能的应用场景

在半导体市场竞争异常激烈的今天,半导体厂商为了能够使产品更方便更简单地为用户所使用,纷纷出招为生产的芯片增加附加价值,以提高客户接受度。在 MBD 开发已经成为汽车电子、家用电器主流开发手段的今天,半导体产品也将目光投入到 MATLAB/Simulink 平台。在出售芯片时提供该芯片的 Simulink 驱动库,使用户可以直接拖曳模块到自己的算法模型中,即可生成包括驱动在内的嵌入式 C 代码,方便实现快速原型,加速开发与产品上市。其中每个驱动模块的说明文档,就是用上述所讲的注释书写脚本,然后发布出来作为驱动模块的用户手册。双击模块打开参数对话框,单击 Help 按钮启动发布的文档可以查看模块中所有参数的意义及模块使用方法,就像 Simulink 自带的模块一样,如图 12.4-1 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

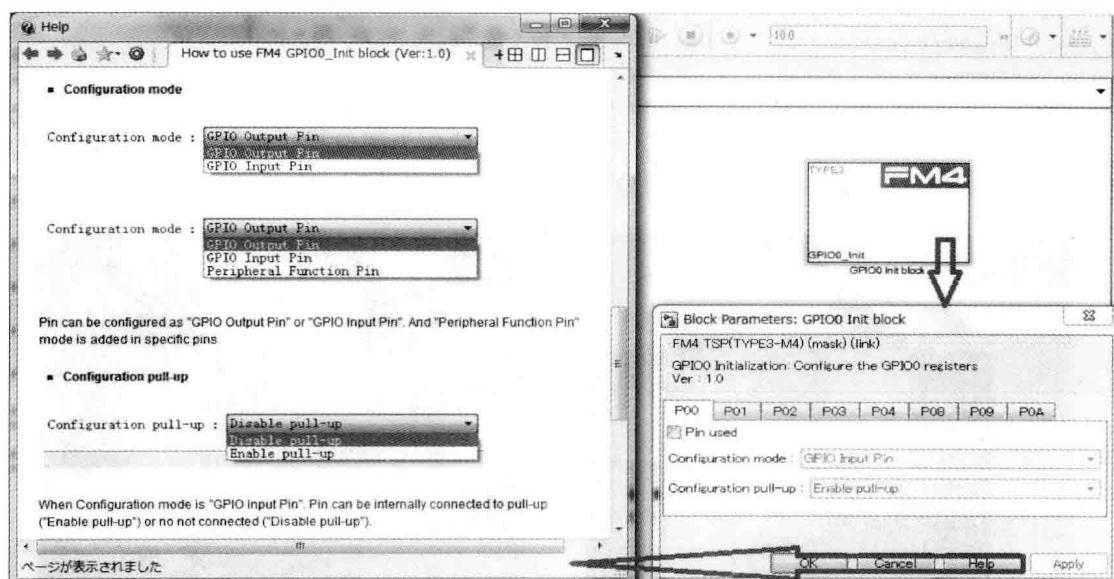


图 12.4-1 注释发布功能在 MCU 芯片驱动库模块中的应用

注释发布功能支持的类型不仅限于 html 类型。Doc、PPT 也都在 publish 发布类型支持范围内,此处不再引申探讨。

# 第 13 章

## Simulink 创建自定义库

**引言:**Simulink 自带了功能强大的模块库,同时也提供给用户自定义模块的 S 函数模块及各种创建子系统的方法。当用户自定义了一类模块之后,可以自定义模块库将同类自定义模块显示到 Simulink Browser 中,作为库模块方便地拖曳到新建模型中建模。

笔者在 MATLAB 中文论坛的简介中(<http://iloveMATLAB.cn/article-29-1.html>)提到过一些通过 S 函数定义的模块,包括天气预报、百度和微博发送等模块,这些模块都是借助互联网在 Simulink 环境下开发的用于个人娱乐的自定义模块。当然,作为研究者或者公司产品的开发者,将一些基本的常用的得到认可的算法、驱动模块添加到 Simulink Library 中,在团队中共享,不失为一种高效的团队工作方式。这些模块能够像 Simulink 自带模块那样通过拖曳即可应用到新建的模型中。本章讲述如何创建自定义的模块库,并显示到 Simulink Library Browser 中去。

建立这样的自定义库需要 3 个条件:

- ① 建立 library 的 mdl 或 slx 文件,将自定义模块添加到文件中保存。
- ② 建立名字为 slblocks 的 M 函数,定义模块库显示到 Simulink Browser 中的规格。
- ③ slblock.m 与 library 模型库文件需要存放到同一个路径下,并将路径添加到 MATLAB 的 Set Path 中(见图 13-1)。

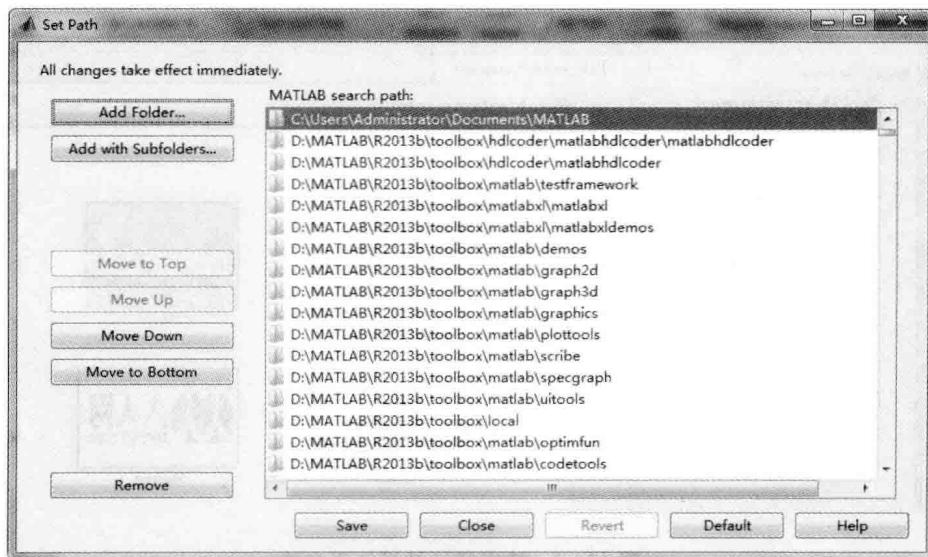


图 13-1 MATLAB 搜索路径存储管理的 Set Path 对话框

所谓 Set Path,是如图 13-1 所示对话框的 MATLAB search path 中存储的路径。当 MATLAB 运行代码或者模型时,从所存储的路径中搜寻代码中使用的函数或模型中的模块,

为了保障运行通畅,需要将自定义的函数或模块的存储路径添加到 MATLAB search path 中。

读者可通过单击左上角的 Add Folder 按钮来将自定义的路径添加到 MATLAB search path 中,或者使用 M 函数 addpath 实现添加过程: addpath('E:\2014 年 MATLAB 中文论坛研讨会')。

在 Simulink Browser 菜单中,选择 File→New→Library 可以新建一个空的模块库文件,如图 13-2 所示。

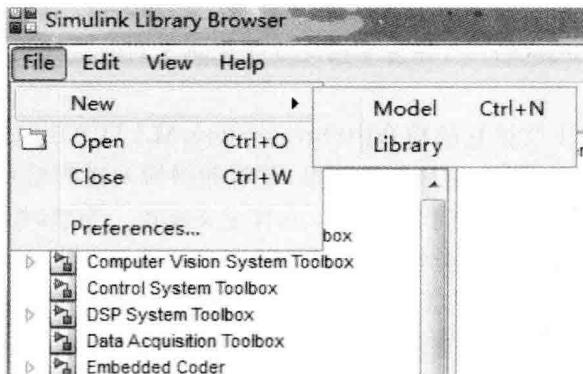


图 13-2 通过菜单新建模块库文件

将希望存入到库中的模块都拖入到库文件中保存。例如图 13-3 所示,将天气预报、新浪和百度等查询、搜索和登录网页的自定义模块追加到库文件 Lib\_entertainment.mdl 中。



图 13-3 拖入模块的模块库文件

Library 与普通 mdl 或 slx 的区别如下:

- ① Library 里的模块不能随意拖动,打开时默认为被锁定的状态。
- ② Library 的工具栏上没有仿真时间和仿真模式的设定。

③ Library 的菜单栏比一般模型文件的菜单栏缺少几个选项: Simulation、Code 和 Tools。如①所述,如果尝试在锁定状态下拖动模块,则会弹出如图 13-4 所示的提示信息。

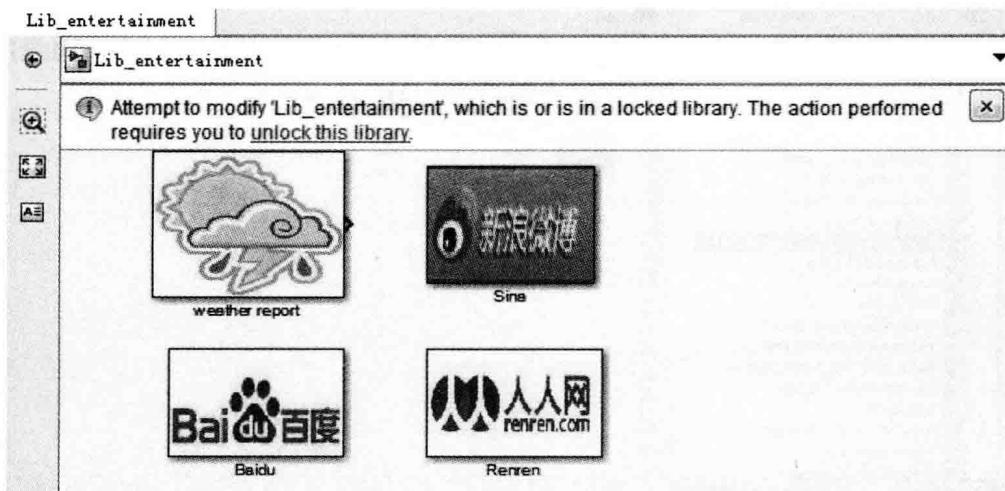


图 13-4 当尝试挪动处于锁定状态的库模块时

将库文件存为 Lib\_entertainment.slx, 并将存储路径添加到 MATLAB search path 中。解锁(Unlock)状态下可将自定义模块拷贝到库中保存。接下来编写 slblocks.m 文件, 这个文件描述了模块库在 Simulink Browser 中的显示方式, 其代码如下:

```
function blkStruct = slblocks
% Information for Simulink Library Browser
Browser(1).Library = 'Lib_entertainment'; % no extension name
Browser(1).Name = 'Entertainment Toolbox @hyowinner';
Browser(1).IsFlat = 1; % no subsystems

blkStruct.Browser = Browser;
```

此函数名必须为 slblocks, 无输入参数, 返回值为 blkStruct, 是一个结构体, 包含了 Browser 等成员。Browser 包含 3 个成员, Library 表示库文件的名字(不带扩展名), Name 表示显示到 Simulink Browser 中标签页和左下角的提示性文字, 一般用来表示该库的主题; IsFlat 则作为一个标志位, 为 1 则表示库文件中的模块不包含子系统。

以上 3 个必要条件都具备了, 打开 Simulink Browser 之后按 F5 键刷新后, 得到如图 13-5 所示效果, 自定义工具箱插入到 Libraries 列表中, 选中时, 右侧会显示出库中所有模块的内容, 左下角也会显示出 slblocks.m 中设置的 Name 属性。

使用这些自定义模块时即可像使用 Simulink 通用模块一样拖曳, 或者右击模块, 在弹出的菜单中选择“add to a new model”。

对于已经封装为库的文件, 如果发现仍然有需要更改的地方, 那么如何再进行编辑呢? 由于模块库打开之后默认处于锁定(Lock)状态无法进行编辑, 因此首先要通过选择菜单栏中的 Diagram→Unlock Library 解锁, 如图 13-6 所示。解锁之后即可进行模块拖动, 增加新模块, 更改既有模块封装等。

新建一个自定义模块库就是这么简单, 读者朋友, 发挥自己的想象力创建更多有趣的模块和库吧。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

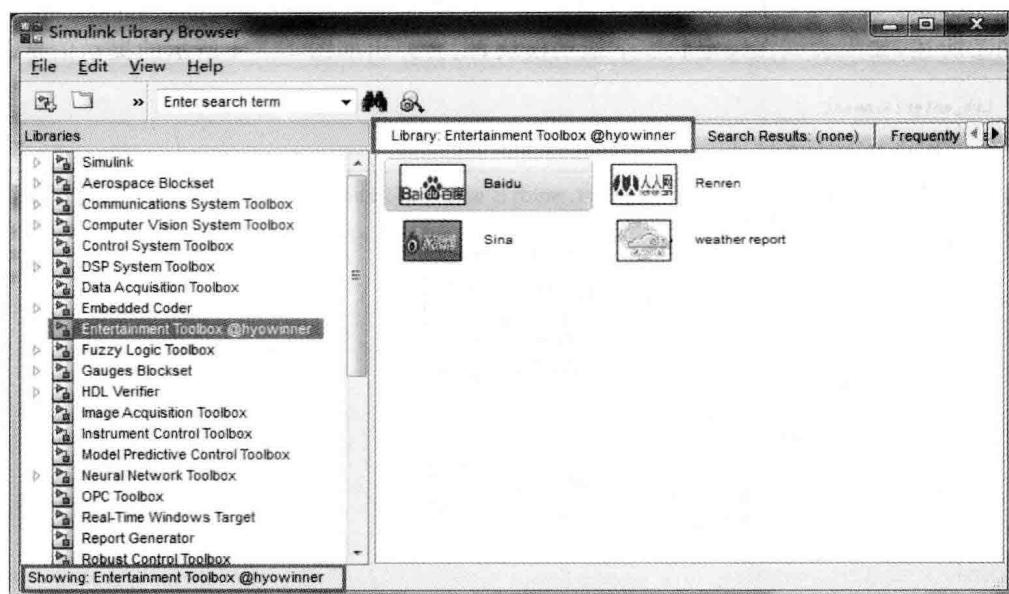


图 13-5 自定义娱乐库添加之后的效果图

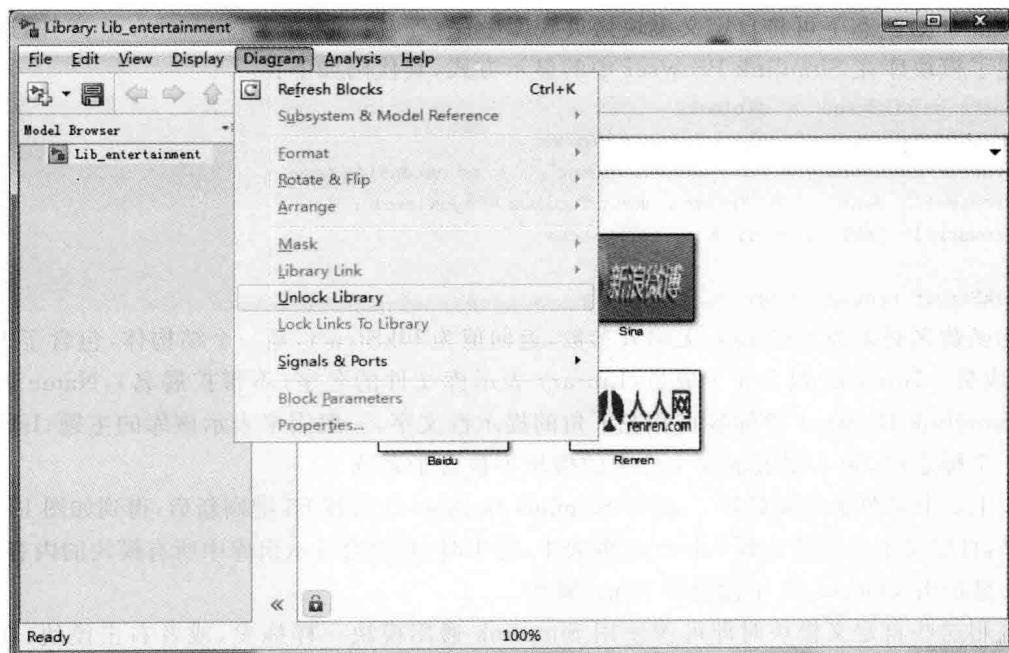


图 13-6 解锁库文件

# 第 14 章

## Simulink 自定义环境

**引言:**Simulink 环境提供了缤纷的仿真、绘图、分析、性能分析和规范检查等功能，菜单栏中分布着各式各样的复杂功能，控制面板上也提供了各式各样的按钮和 check-box 等控件，对于初学者甚至资深用户来说都未必敢说已经熟悉了每一个设置参数的功能。俗话说“过犹不及”，导致的情况就是太多臃肿的功能对于部分用户个人来说是过剩的需求，而一些简单常用的功能反而没有直观地提供给用户。Simulink 通过接口文件 sl\_customization.m 提供了自定义环境的功能，让用户能够根据自身需求和喜好进行自定义。

### 14.1 Simulink 环境自定义功能

sl\_customization.m 函数是 Simulink 提供给用户使用 MATLAB 语言自定义 Simulink 标准人机界面的函数机制。若 sl\_customization.m 存储在 MATLAB 的搜索路径中，则当 Simulink 启动时就会读取此文件的内容进行 Simulink 的人机界面初始化。Simulink 本身就提供了这个函数，用户每次修改之后，必须重启 Simulink，或者使用命令 sl\_refresh\_customizations 使变更起作用。本节主要讲解 Simulink 环境自定义中的菜单栏自定义、GUI 参数对话框控件的属性编辑、Simulink Library Browser 显示效果自定义及 Simulink Hardware Implementation 自定义。

### 14.2 Simulink 工具栏菜单自定义

所谓菜单自定义，并不是将 Simulink 自带的菜单功能进行裁剪或删除，而是在 Simulink 提供的现有菜单的基础上进行菜单项的添加。Simulink Model Editor 中用于添加菜单项的位置有 3 个：顶层菜单的末尾、菜单栏和右键菜单的开始或结尾处。

添加的对象称为项目(item)，为了添加项目，需要以下步骤：

- ① 创建一个定义项目的模式函数(schema function)。
- ② 将这个定义菜单项目的函数注册在 sl\_customization.m 中。
- ③ 为这个菜单项目定义一个触发运行的回调函数。

以增加一个显示当前所选模块属性列表的菜单项目为例，首先创建一个定义此项的模式函数：

```
function schema = get_block_property(callbackInfo)
    schema = sl_action_schema; % 使用 sl_action_schema 函数创建一个对象
    schema.label = 'block property'; % 对这个对象设置各个属性，label 为菜单名
    schema.userdata = 'Hyo Custom';
    schema.callback = @custom_callback; % 选中此菜单时所触发的回调函数
end
```

此函数的回调函数(Callback)命名为 custom\_callback, 其内容为显示当前鼠标所选模块的属性列表, 代码如下:

```
function custom_callback(callbackInfo)
inspect(gcbh);
disp('# ## The property of current block is displayed.');
end
```

上述两个函数可以一同定义在 get\_block\_property.m 中。接着将定义的模式函数注册到 sl\_customization.m 中。函数 addCustomMenuFcn 的作用是将自定义模式函数注册到 Simulink 的菜单栏中。菜单栏的标示能力是由 WidgetId 控制的, 如 'Simulink:ToolsMenu' 就是 Tools 菜单的 WidgetId, 使用下面两句代码可以将所有 Simulink 菜单栏及子层菜单的 WidgetId 显示出来:

```
cm = sl_customization_manager;
cm.showWidgetIdAsToolTip = true
```

在 Command Window 中输入上述两行代码并运行, 可以得到如图 14.2-1 所示效果。

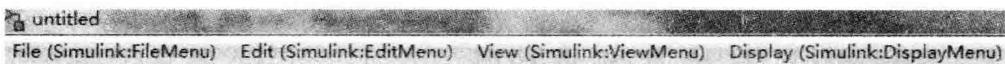


图 14.2-1 显示 WidgetId 的菜单栏

右键菜单的 WidgetId 也同时得以显示。通过 cm.showWidgetIdAsToolTip = false 即可关闭 WidgetId 的显示。了解了 WidgetId, 即可在 sl\_customization.m 中使用 addCustomMenuFcn 方法注册一个自定义菜单项目函数, 如在 Simulink 的 Tools 菜单栏下追加项目:

```
function sl_customization(cm)
% Register custom menu function.
cm.addCustomMenuFcn('Simulink:ToolsMenu', @custom_items);
end
```

在注册的 custom\_items 函数中使用句柄函数方式(亦称匿名函数)指定之前定义过的模式函数到 custom\_items:

```
% Define the custom menu function.
function schemaFcns = custom_items(callbackInfo)
schemaFcns = {@get_block_property};
end
```

这样就把自定义模式函数 get\_block\_property 注册到了 Simulink Model Editor 的 Tools 菜单栏最末尾的一项, 如图 14.2-2 所示。

函数 custom\_items、get\_block\_property 和 custom\_callback 的参数 callbackInfo 并没有被使用, 可使用~表示。添加自定义菜单项目后, 如何使用呢? 模型中任意选中一个模块, 再从 Simulink Editor 菜单中选择添加的项目, 即可弹出所选模块的 inspect 属性列表, 如图 14.2-3 所示。

自定义 Model Editor 的菜单不仅可以是一级菜单, 还可以此为基础继续向下定义多级子菜单。仍在模式函数中实现, 需要 sl\_container\_schema 函数创建一个容器对象, 并将刚才创建的菜单项目作为其成员添加进去:



图 14.2-2 自定义菜单项目

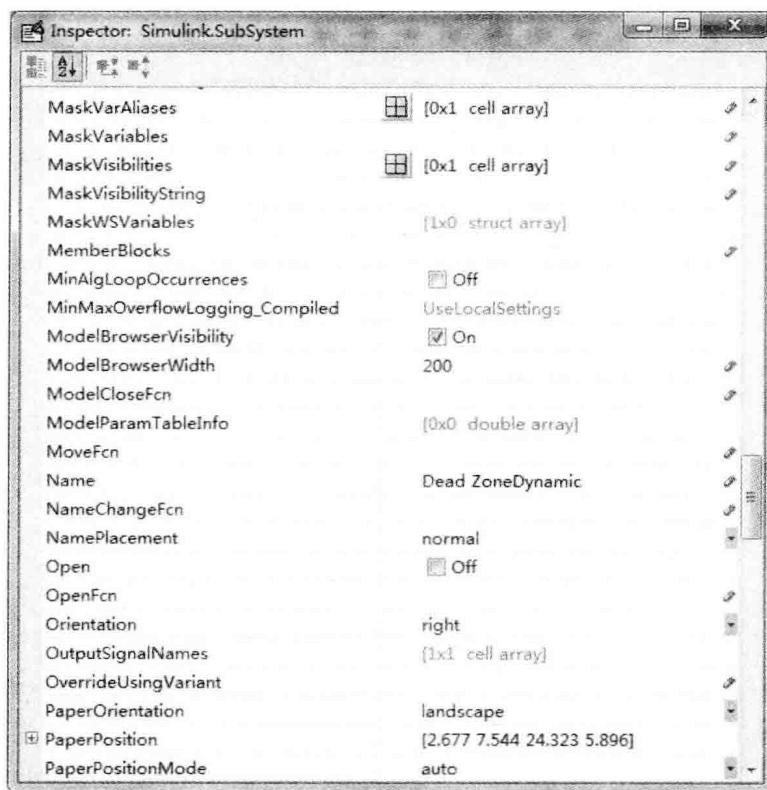


图 14.2-3 所选模块的属性列表

```
% % Define the schema function for multi-layer menu item.
function schema = menu_control(callbackInfo)
schema = sl_container_schema; % 创建一个模式容器
schema.label = 'Hyo Customized'; % 用于显示在菜单栏中的标签名
schema.childrenFcns = {@get_block_property}; % 将子目录项目作为此容器的子成员
end
```

定义菜单项的函数 get\_block\_property 为：

```
function schema = get_block_property(callbackInfo)
schema = sl_action_schema;
schema.label = 'block property';
schema.userdata = 'Hyo Custom';
schema.callback = @custom_callback;
end

function custom_callback(callbackInfo)
inspect(gcbh);
disp('# # # The property of current block is displayed.');
end
```

在 sl\_customization.m 中注册菜单项目的模式函数时将最上层的模式容器对象注册进去即可：

```
function sl_customization(cm)
% % Register custom menu function.
cm.addCustomMenuFcn('Simulink:ToolsMenu', @custom_items);
end
```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```
% % Define the custom menu function.
function schemaFcns = custom_items(callbackInfo)
    schemaFcns = {@menu_control}; % 注册模式容器对象到菜单栏自定义项目中
end
```

使用 sl\_refresh\_customizations 命令刷新 Simulink 环境配置之后,可以在 Model Editor 的菜单栏 Tools 下找到自定义多级菜单,如图 14.2-4 所示。

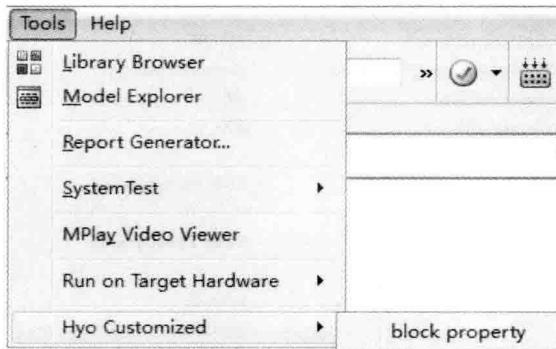


图 14.2-4 自定义多级菜单

自定义菜单项也可以定义到右键菜单中。获取选中模块的属性列表的功能,作为右键菜单提供更加便利,相比于选中模块后去单击工具栏菜单中的选项,省去了先左键单击模块的步骤。实现自定义右键菜单只需要将 sl\_customization.m 中 addCustomMenuFcn 函数的第一个参数设置为 'Simulink:ContextMenu', 其他方法同上。实现效果如图 14.2-5 所示,添加后显示在菜单末尾。

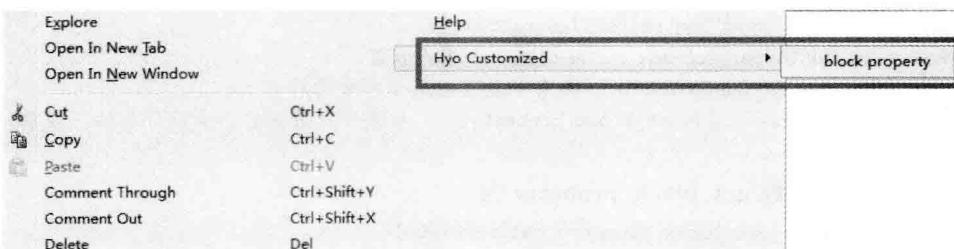


图 14.2-5 自定义右键菜单

### 14.3 Simulink Library Browser 菜单栏自定义

Simulink Library Browser 中各个 Simulink 工具箱的排列顺序是内置的,由优先级和名字 2 个因素决定。优先级数字越小的工具箱排位越靠前;对于同一优先等级下的众多工具箱,则按照字母顺序从 a~z 依次排列。默认情况下 Simulink 库优先度为 -1,其他工具箱优先度为 0。如果使用者希望将某个自己常用的工具箱提到第一位显示,可以更改优先度为小于 -1 的整数即可,这个功能在 sl\_customization.m 中使用 LibraryBrowserCustomizer. applyOrder 方法来实现,如:

```
cm.LibraryBrowserCustomizer.applyOrder(['Embedded Coder', -1.1]);
```

刷新 Simulink 环境后,再度打开 Simulink Library Browser,显示效果如图 14.3-1 所示。

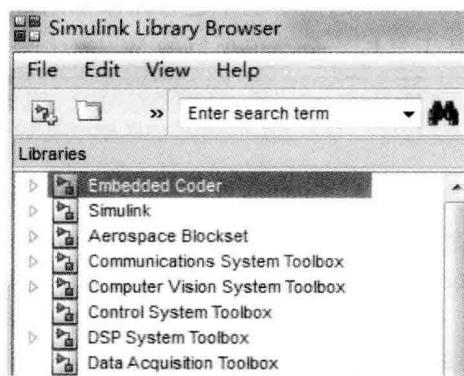


图 14.3-1 自定义工具箱排列顺序

LibraryBrowserCustomizer 是集成了多种自定义 Simulink Library Browser 功能的类, 提供的 applyCoder 方法可以对指定工具箱的排序优先级进行设定; 该类也提供了将指定工具箱隐藏起来不显示, 以及虽然可见但是不可使用的功能, 方法名为 applyFilter, 其格式为:

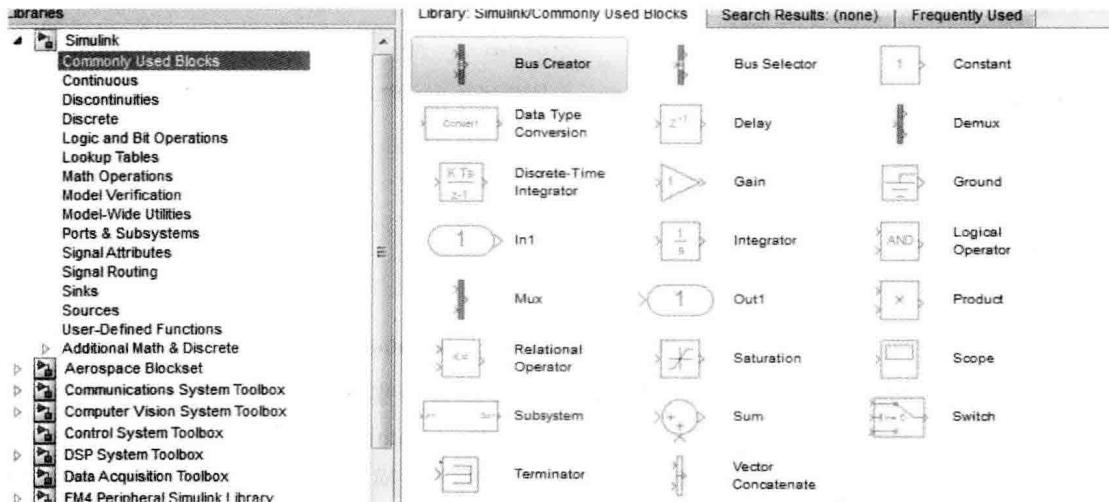
```
cm.LibraryBrowserCustomizer.applyFilter({'PATH1', 'STATE1', ...})
```

PATH 表示操作的目标模块或工具箱, 接受的参数路径可以参考 Simulink Library 中工具箱的树结构层次。STATE 则可以是 visible、invisible、enable 和 disable 等字符串表示可见性和使能与否的设置。

例如将原本排列在首位的 Embedded Coder 工具箱隐藏起来, 并将 Simulink 工具箱设置为不可用, 在 sl\_customization.m 中添加以下代码:

```
cm.LibraryBrowserCustomizer.applyFilter({'Embedded Coder', 'Hidden'});
cm.LibraryBrowserCustomizer.applyFilter({'Simulink', 'Disabled'});
```

刷新 Simulink 环境后, Simulink Library Browser 中的模块库的工具箱再次发生显示变化, Embedded Coder 隐藏了, Simulink 工具箱图表变为灰色, 内部模块全部变为半透明状态, 并且不可拖曳到模型中, 如图 14.3-2 所示。



若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

另外,Simulink Library Browser 的菜单栏中也可以添加用户自定义的菜单,开启了 WidgetId 的显示之后,可以看到其工具栏 4 个菜单项目的 WidgetId 名,如图 14.3-3 所示。

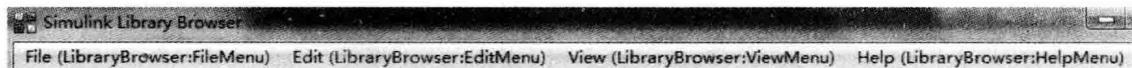


图 14.3-3 Simulink 库菜单的 WidgetId 名

接着将菜单项目模式函数注册到 Library Browser 的 WidgetId 中:

```
cm.addCustomMenuFcn('LibraryBrowser:HelpMenu', @custom_items);
```

其余代码参考 14.2 节。刷新 Simulink 环境后,重新启动 Simulink Library Browser 之后,单击 Help 菜单可以看到新追加的菜单选项,如图 14.3-4 所示。

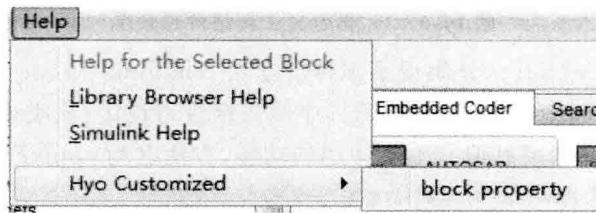


图 14.3-4 添加自定义菜单到 Simulink Library Browser

## 14.4 Simulink 目标硬件自定义

用户可在 Configuration Parameter 对话框中的 Hardware Implementation 子页面选择各个厂家提供的各种目标硬件芯片类型,设置其大小端方式,所支持的各种数据类型的位数,如图 14.4-1 所示。

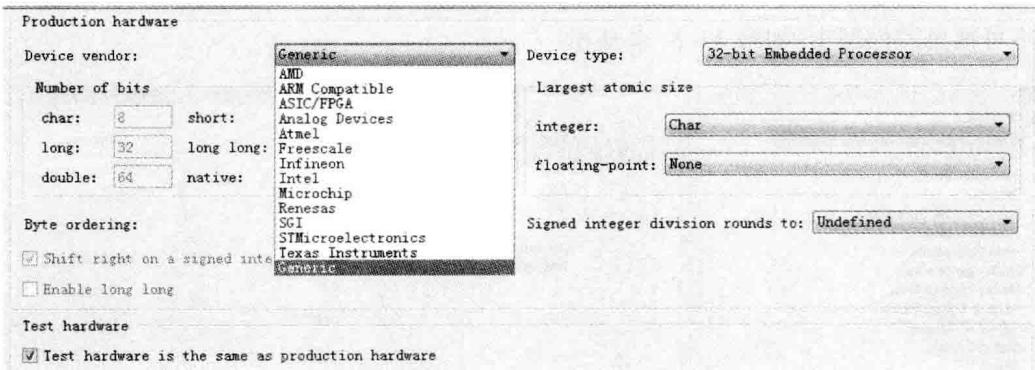


图 14.4-1 默认 Hardware Implementation 页面

这里的硬件实现并非实现硬件内部的运行机理或指令集仿真功能,只是对某种芯片的厂商和型号信息及数据类型格式进行描述。在 sl\_customization.m 中使用 loc\_register\_device 注册一个函数,在这个函数中对目标硬件的属性进行描述即可。描述目标硬件的结构体对象需要使用 RTW.HWDeviceRegistry 创建并配置。需要配置的成员信息如表 14.4-1 所列。

表 14.4-1 目标硬件配置信息

| 配置成员名                | 说 明   |
|----------------------|---|
| Vendor               | 芯片生产商   |
| Type                 | 芯片系列号或类型名   |
| Alias                | 芯片别名,不使用时不用设置,为空  |
| Platform             | Production hardware 以及 Test hardware 两个组别中是否都显示控件清单,二者组别名分别以字符串 'prod', 'test' 表示   |
| setWordSizes         | 设置各种数据类型的位数,按照长度顺序 char ≤ short ≤ int ≤ long。long 以外类型位数必须为 8 的整数倍,最大为 32 位。long 型位数则不能少于 32 位  |
| LargestAtomicInteger | 最大原子整数,生成代码中存在数据拷贝时起作用;若对位数大于此选项设置的数据类型进行整数数据拷贝,则检测数据拷贝完整性体现在代码生成中  |
| LargestAtomicFloat   | 最大原子浮点数,生成代码中存在数据拷贝时起作用;若对位数大于此选项设置的数据类型进行浮点数据拷贝,则检测完整性的代码也一同生成   |
| Endianess            | 字节次序设定,大端 big 或小端 little  |
| IntDivRoundTo        | 有符号整数进行除法时的四舍五入方法选择: 'zero', 'floor' 或 'undefined'  |
| ShiftRightIntArith   | true/false, 设置是否将有符号整数的右移操作作为算数右移   |
| setEnabled           | 选择此目标硬件时哪些 GUI 控件是可选的。表示控件的字符串包括:<br>"BitPerChar" "BitPerShort" "BitPerInt" "BitPerLong" "WordSize" "Endianess"<br>"IntDivRoundTo" "ShiftRightIntArith" "LongLongMode" "BitPerFloat" "BitPerDouble"<br>"BitPerPointer" "BitPerLongLong" "LargestAtomicInteger" "LargestAtomicFloat" |

例如描述一个虚拟目标硬件提供商 Hyo 生产的 SimulinkType 系列芯片,其 M 代码如下:

```

function sl_customization(cm)
cm.registerTargetInfo(@loc_register_device);
end
% loc_register_device registers self defined device into Simulink Parameter
% configuration
function thisprod = loc_register_device
thisprod = RTW.HWDeviceRegistry;
thisprod.Vendor = 'Hyo Ltd.';
thisprod.Type = 'Simulink Type';
thisprod.Alias = {};
thisprod.Platform = {'Prod','Test'};
thisprod.setWordSizes([8 16 16 32 32]);
thisprod.LargestAtomicInteger = 'Char';
thisprod.LargestAtomicFloat = 'Float';
thisprodEndianess = 'Big';
thisprod.IntDivRoundTo = 'floor';
thisprod.ShiftRightIntArith = true;
thisprod.setEnabled({'BitPerPointer'});
end

```

在 Command Window 中输入 sl\_refresh\_customizations, 刷新 Simulink 环境之后, 打开 Hardware Implementation 页面确认, 可以看到上述描述代码已经将一款新的目标硬件添加到

Vendor 下拉框选择之中,选择 Hyo Ltd. 之后 Hardware Implementaion 展示上述配置内容,如图 14.4-2 所示。

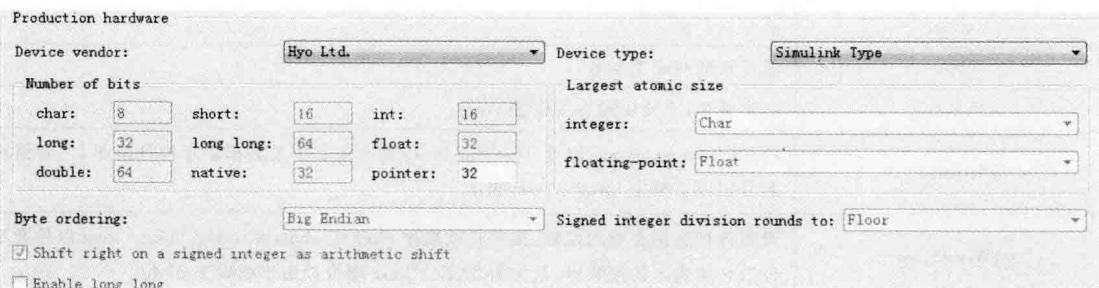


图 14.4-2 追加了自定义硬件描述的 Hardware Implementation 页面

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

## 14.5 Simulink 参数对话框控制

Simulink 环境下的对话框也可以通过属性设置的方法进行一定的自定义设置。执行下面语句之后,将鼠标停留在其上能够显示出 WidgetId 和 dialogId 的控件,都可以由用户指定其可见性和使能与否。

```
cm = sl_customization_manager;
cm.showWidgetIdAsToolTip = true
```

在 Command Window 中运行上述代码之后,将鼠标停留在 Configuration Parameters 对话框的 Type 控件上,可以显示出其 DialogId 和 WidgetId,如图 14.4-3 所示。

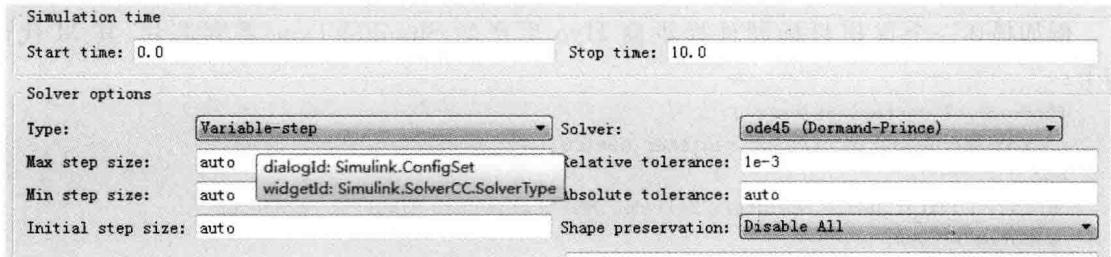


图 14.4-3 Configuration Parameter 控件上显示 WidgetId 名

控制这些属性必须使用 M 语言。实现步骤是首先在 sl\_customization.m 中使用 addDlgPreOpenFcn 方法注册一个回调函数,回调函数绑定在目标控件的父对象上。再在回调函数中编写设置目标控件属性的 M 语句。对一个控件的回调函数需要注册到父对象上的 ConfigSet 中。本质上就是在目标控件的父控件被打开之前调用这个自定义的回调函数实现某控件的使能及可见性设置。这个自定义的回调函数必须带有一个 dialogH 参数,表示目标控件父对象的句柄,通过这个父对象再调用 hideWidgets 函数隐藏目标控件,调用 disableWidgets 方法使能或关闭某控件的使用权。请注意,注册回调函数时使用的是对话框父对象的 DialogId 名,而在回调函数中实现对某个控件操作时使用的是控件本身的 WidgetId 名。如将 Solver 页面下选择解算器变步长还是固定步长的下拉框 SolverType 隐藏,并将仿真开始时间的 Edit 框变为不可编辑状态。

```

function sl_customization(cm)
% % Register custom dialog pre-opening function.
cm.addDlgPreOpenFcn('Simulink.ConfigSet', @disable_solver_type);
end
function disable_solver_type(dialogH)
dialogH.hideWidgets({'Simulink.SolverCC.SolverType'});
dialogH.disableWidgets({'Simulink.SolverCC.StartTime'});
end

```

刷新 Simulink 环境后重新启动 Configuration Parameter 对话框打开 Solver 页面, 可以看到 Start time 编辑框变为灰色的不可编辑状态, SolverType 的选项也整个消失, 如图 14.4-4 所示。

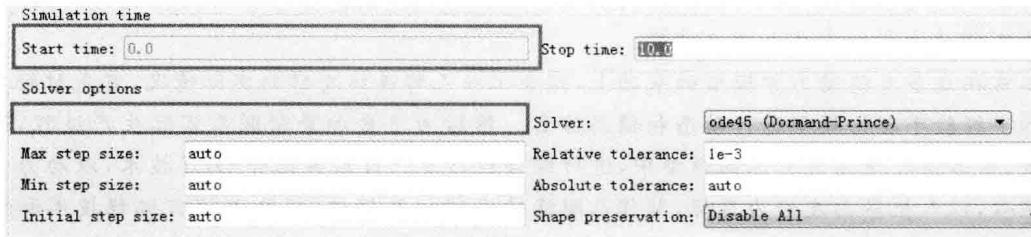


图 14.4-4 Configuration Parameter 被改变属性的控件

上述的控件属性控制只是对启动状态(初始状态)的设置, 并不一定持续整个使用过程。这是因为 Configuration Parameter 上的控件之间存在互相约束的关系(Simulink 内建 Callback), 一旦触发了这些内置的约束关系的回调函数, 就可能打破用户自定义的初始值属性约束。例如 Code Generation 页面下的 Build 页面, 即使将其设置为 Disable, 但是启动后用户将 Generate Code Only 的 check-box 再勾掉之后, Build 就变为可以单击的状态了。

读者也可以根据自己的需求定义个性化的控件设置。如当用户打开一个系统目标文件设置为 ert.tlc 的模型时, 自动设置代码生成报告选项为勾选状态并将其设置为不可编辑状态, 以保证生成代码时肯定可以启动代码报告:

```

function sl_customization(cm)
% register pre-open callback function
cm.addDlgPreOpenFcn('Simulink.ConfigSet', @ert_tlc_callback)
end

function ert_tlc_callback(dialogH)
systargetfile = dialogH.getSource.getParam('SystemTargetFile');
if strcmp(systargetfile, 'ert.tlc')
    dialogH.getSource.setParam('GenerateReport', 'on')
    dialogH.disableWidgets({'Simulink.RTWCC.CodeGenReport'});
end

```

在 sl\_customization.m 中设置对话框中各个控件的状态只能在 Configuration Parameter 启动的时候调用, 不会因对话框中每个控件的值变化而即时动作。即时动作要求被触发的控件存在回调函数, 读者可以对控件编写回调函数以实现自定义即时动作, 如自定义系统目标文件时可以定义系统目标文件的选择回调函数 rtwgenselect, SelectCallback, 以及自定义 Configuration Parameter 时可以定义单击 OK 或 Apply 按钮时的应用回调函数 rtwgenselect, PostApplyCallback。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

# 第 15 章

## Simulink 在流程工业中的仿真应用

**引言:**与石油能源相比,乙醇不仅是理想的替代品,而且是十分重要的清洁能源,广泛地用于食品、化学、医药、燃料、国防等行业。随着发酵技术的发展及工业生产要求的提高,人们提出了许多发酵动力学模型来研究发酵过程和分析发酵过程,但是对于工业化生产的贡献不高。计算机技术的发展为进一步研究和提高乙醇工业制取效率提供了可靠的硬件支持,使得进一步发展发酵动力学仿真技术成为可能。

本章在众多发酵动力学模型的基础上,结合工业乙醇连续发酵的实际情况,首先对经典的流程加半经验半理论模型进行动态和稳态仿真。然后为了更加紧密联系实际生产过程,逐步将温度、气体排放等因素考虑到模型中,进行综合性仿真。最后再结合 GUI 技术,以动力学模型为核心,以人机界面友好为目标,制作乙醇连续发酵仿真软件,将脚本语言编程技术和模块化仿真技术结合起来,建立起工业乙醇发酵的仿真平台,并结合 Simulink 模型的代码生成技术来加快仿真速度,提高平台的应用效率。

### 15.1 工业乙醇生产与计算机仿真

乙醇作为可再生清洁能源不仅可以替代四乙基铅作汽油的防爆剂,还可以制造汽油醇。这一巨大的潜在需求促使人们去寻找提高乙醇工业生产率的途径,使人们着手于发酵工程的研究。微生物学、发酵研究的发展,使微生物反应过程的种类和规模不断地扩大,其应用也深入到多个工业领域,然而由于反应涉及活细胞的参与,菌体生长及产物生成等机理复杂多变,目前尚难为人们了解和把握,更难以进行统一的描述。人们一般通过实验的方法寻求微生物的生长规律,通过数值分析和拟合了解发酵过程的规律。这样的做法周期性长,需要消耗的资源多,且无法在短时间内对工业流程中出现的问题提出及时的应对方案。

计算机仿真技术作为分析和研究系统运行行为、揭示系统动态过程和运动规律的一种重要手段和方法,随着系统科学的研究的深入,以及控制理论、计算技术、计算机科学与技术的发展而形成一门新兴的学科。近年来,随着信息处理技术的突飞猛进,仿真技术得到迅速发展。“仿真是一种基于模型的活动”,涉及多学科、多领域的知识和经验。成功进行仿真研究的关键是有机、协调地组织实施仿真全生命周期的各类活动。这里的“各类活动”,就是“系统建模(Modeling)”、“仿真建模(Simulation Test)”、“仿真实验(Real World Test)”,联系这些活动的要素是“系统(System)”、“模型(Model)”、“计算机(Computer)”。其中,系统是研究的对象,模型是系统的抽象,仿真是通过对模型的实验达到跟真实系统对比并接近于研究其特性的目的。三者关系如图 15.1-1 所示。

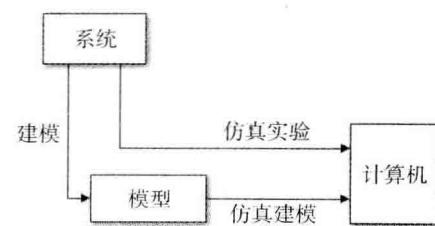


图 15.1-1 系统、模型及计算机的关系

## 15.2 工业乙醇发酵流程

以某化工原料有限公司的乙醇连续发酵系统为背景,其生产系统简图如图 15.2-1 所示。

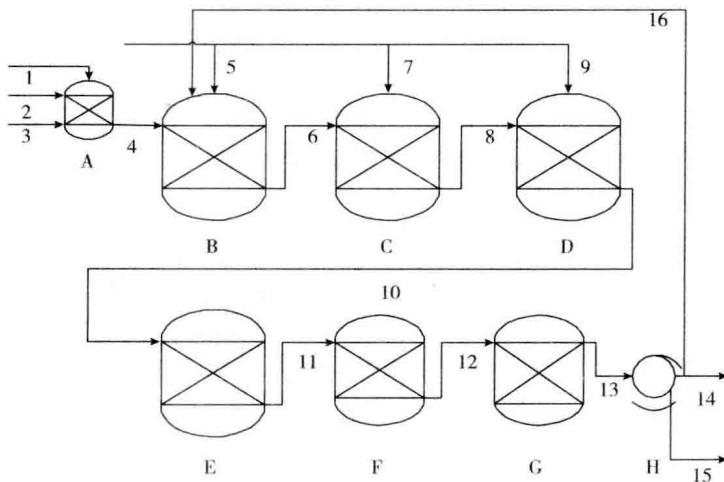


图 15.2-1 发酵流程图

整个发酵流程的主要设备包括 1 个预发酵罐 A、6 个发酵罐(B~G)、离心分离机(H)等。涉及的进出料流股一共有 16 个。发酵设备与流股之间的关系可以用简单的流程图表示出来,图 15.1-1 是整个发酵过程的流程图,其中,A 为预发酵罐( $165\text{ m}^3$ ),进料 1、2、3 分别为干法糖液、营养盐、酵母;B 为第一级发酵罐( $600\text{ m}^3$ ),进料 4 为 A 的出料,进料 5 为干法糖液,16 为循环回来的酵母;C 为第二级发酵罐( $600\text{ m}^3$ ),进料 6 为 B 的出料,进料 7 为干法糖液;D 为第三级发酵罐( $600\text{ m}^3$ ),进料 8 为 C 的出料,进料 9 为干法糖液;E 为第四级发酵罐( $600\text{ m}^3$ ),进料 10 为 D 的出料;F 为第五级发酵罐( $500\text{ m}^3$ ),进料 11 为 E 的出料;G 为第六级发酵罐( $500\text{ m}^3$ ),进料 12 为 F 的出料;H 为离心分离机,进料 13 为 G 的出料,出料 15 为产品,含  $11\% \sim 12\% (\text{V}/\text{V})$  的乙醇,送入蒸馏装置分离,出料 14 为废酵母,其中一部分循环回去。发酵温度要求控制在  $31 \sim 33^\circ\text{C}$ ,前 3 个发酵罐需要补充液料,发酵过程中补充菌种。

## 15.3 乙醇发酵动力学方程

在介绍了工业生产乙醇的流程和工艺之后,本文进一步深入研究乙醇发酵动力学知识,在前人研究总结的半经验半理论模型的基础上进行计算机仿真(基于时间序列的系统动态仿真),在得到较好的仿真结果之后,结合实际生产环境和要求,将实际影响因素添加到动力学模型中,从而建立更加完善的仿真模型,将仿真结果与工业标准相对比。经过检验之后,将已经建立的动力学模型作为基础,着手乙醇连续发酵软件的开发,以友好的界面、简单的操作和可靠有效的仿真能力为目标,设计并制作出乙醇连续发酵的实验用软件,为获得实验数据,研究发酵特性,以及为寻找发酵过程最优控制等后续研究提供基础。人们从经典的 Monod 方程出发,提出了一些发酵动力学模型,如:

$$\mu = \mu_{\max} \frac{s}{K_s + s} \frac{K_p}{K_{p\ell} + p} (1 - x/K_{x\ell}) \quad (15.3-1)$$

$$\mu = \mu_{\max} \exp(-K_p p) \quad (15.3-2)$$

$$\mu = \mu_{\max} \frac{s}{K_s + s + s^2/K_s} \frac{K_p}{K_p + p + p^2/K_p} \quad (15.3-3)$$

等,用于描述发酵从开始到停滞阶段的行为。各项抑制常数的引入使模型更符合实际,但是这些模型的模拟都是半理论半经验的。对不同的发酵过程,方程形式及抑制常数的选取不一样。 $x$  表示酵母菌体的浓度, $g/L$ ;  $p$  表示乙醇的浓度, $g/L$ ;  $s$  表示葡萄糖的浓度, $g/L$ ;  $\mu$  表示菌体比生长率(单位时间单位体积浓度变化率),  $h^{-1}$ ;  $\mu_{\max}$  为最大菌体比生长速率,  $h^{-1}$ ;  $K_s$  等以  $K$  开头的参数为底物抑制常数。

为了建立乙醇连续发酵的动力学方程,首先要明确在静止情况下(没有补充液和流进流出)各种物料之间的关系:①葡萄糖液在酵母的作用下分解产生乙醇;②酵母细胞本身还要吸收消化葡萄糖来促进自身的生长。三者之间的关系可以用三个微分方程来描述(虽然可以用比死亡率表示菌体的衰败,但是发酵过程中比生长率远远大于比死亡率,所以比死亡率可以忽略不计):

酵母菌体:

$$\frac{dx}{dt} = \mu x \quad (15.3-4)$$

乙醇:

$$\frac{dp}{dt} = vx \quad (15.3-5)$$

葡萄糖:

$$\frac{ds}{dt} = -(\alpha + \beta)x \quad (15.3-6)$$

$v$  表示乙醇比生产率,  $\alpha$  表示由菌体消化掉葡萄糖的消耗率,  $\beta$  表示分解产生乙醇的葡萄糖的消耗率。然后引入得率的概念:

$Y_{x/s}$  菌体相对于葡萄糖的得率,即葡萄糖消耗后产生菌体的比率。

$Y_{p/s}$  乙醇相对于葡萄糖的得率,即葡萄糖分解产生乙醇的比率。

从得率的意义可知: $Y_{x/s} = \mu/\alpha$ ,  $Y_{p/s} = v/\beta$ ,代入葡萄糖的方程(15.3-6),可得:

$$\frac{ds}{dt} = -\left(\frac{\mu}{Y_{x/s}} + \frac{v}{Y_{p/s}}\right)x \quad (15.3-7)$$

模型描述了静止发酵反应中浓度的变化情况,实际工业生产往往是连续发酵,发酵流程不仅创造了酵母生长的合适环境,也提供了发酵稳定的条件,提高了发酵能力和生产效率。生产连续化,设备利用率高,一般采取的是多罐串联式连续发酵(本文对 6 个串联罐进行仿真)。

为了适应工业生产流程,上述模型也应该显现出流动性。

首先考虑理想情况,稳定发酵情况下,由单个发酵罐子中物料守恒可得出两个恒等关系:

① 对每个发酵罐而言,发酵液流入量=流出量;

② 对每种物质而言,其质量变化=流入/流出质量变化+反应生成/消耗质量变化。

$F_{in}$  和  $F_{out}$  分别表示流入量、流出量,  $V$  表示发酵罐体积,  $x_0$  是流入发酵罐的物质浓度(此处是菌体浓度):

$$F_{in} = F_{out} \quad (15.3-8)$$

$$V \frac{dx}{dt} = F_{in}x_0 + \mu x V - F_{out}V \quad (15.3-9)$$

式中仅以酵母菌为例,其余类似。

因为  $F_{in} = F_{out}$ ,令二者均为  $F$ ,代入式(15.3-9)并整理得:

$$\frac{dx}{dt} = \mu x + \frac{F}{V}(x_0 - x) \quad (15.3-10)$$

式中,  $\frac{F}{V}$  称为稀释率,表示为  $D$ ,由此可将单个罐子中的连续发酵动力学模型表示为:

$$\frac{dx}{dt} = \mu x + D(x_0 - x) \quad (15.3-11)$$

$$\frac{dp}{dt} = ux + D(p_0 - p) \quad (15.3-12)$$

$$\frac{ds}{dt} = -\left(\frac{\mu}{Y_{x/s}} + \frac{v}{Y_{p/s}}\right)x + D(p_0 - p) \quad (15.3-13)$$

因为实际生产过程中基质浓度较高,需要考虑底物和产物对细胞生长的抑制作用。 $\mu, v$  表示为:

$$\mu = \mu_{max} \frac{S}{K_s + S + S^2/K_{sl}} \frac{K_p}{K_p + p + p^2/K_{pl}} \quad (15.3-14)$$

$$v = v_{max} \frac{S}{K_{sp} + S + S^2/K_{spl}} \frac{K_{pp}}{K_{pp} + p + p^2/K_{ppt}} \quad (15.3-15)$$

上述动力学方程组中各参数采用吕欣得出的数据如表 15.3-1 所列。

表 15.3-1 各抑制参数

| 符 号         | 取 值     | 单 位                              | 说 明                            |
|-------------|---------|----------------------------------|--------------------------------|
| $Y_{x/s}$   | 0.160 4 | $\text{kg} \cdot \text{kg}^{-1}$ | 菌体相对于葡萄糖的得率                    |
| $Y_{p/s}$   | 0.498 6 | $\text{kg} \cdot \text{kg}^{-1}$ | 乙醇相对于葡萄糖的得率                    |
| $\mu_{max}$ | 0.113 2 | $\text{h}^{-1}$                  | 最大比生长率                         |
| $v_{max}$   | 0.998 2 | $\text{h}^{-1}$                  | 最大乙醇比生产率                       |
| $K_s$       | 101.276 | $\text{kg} \cdot \text{m}^{-3}$  | Monod 常数(菌体饱和时底物浓度)            |
| $K_{sp}$    | 9.916   | $\text{kg} \cdot \text{m}^{-3}$  | 基于底物的乙醇饱和常数(乙醇饱和时底物的浓度)        |
| $K_p$       | 28.779  | $\text{kg} \cdot \text{m}^{-3}$  | 乙醇饱和常数(菌体饱和时乙醇的浓度)             |
| $K_{pp}$    | 660.54  | $\text{kg} \cdot \text{m}^{-3}$  | 基于产物的乙醇饱和常数(乙醇饱和时的浓度)          |
| $K_{sl}$    | 106.500 | $\text{kg} \cdot \text{m}^{-3}$  | 底物抑制常数(抑制作用使细胞停止生长时底物的浓度)      |
| $K_{spl}$   | 296.540 | $\text{kg} \cdot \text{m}^{-3}$  | 基于底物的乙醇抑制常数(抑制作用使乙醇停止生产时的底物浓度) |
| $K_{pl}$    | 5.968   | $\text{kg} \cdot \text{m}^{-3}$  | 乙醇抑制常数(抑制作用使细胞停止生长时乙醇的浓度)      |
| $K_{ppt}$   | 16.658  | $\text{kg} \cdot \text{m}^{-3}$  | 基于产物乙醇的乙醇抑制常数(抑制作用使乙醇停产时乙醇的浓度) |

乙醇连续发酵的实际情况是存在多个入料口,且参数各不相同。具体流程如图 15.3-1 所示。

表 15.3-2 所列为流程中典型工况下的各股进料输入情况。整个发酵流程包括 1 个预发酵罐,6 个发酵罐。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

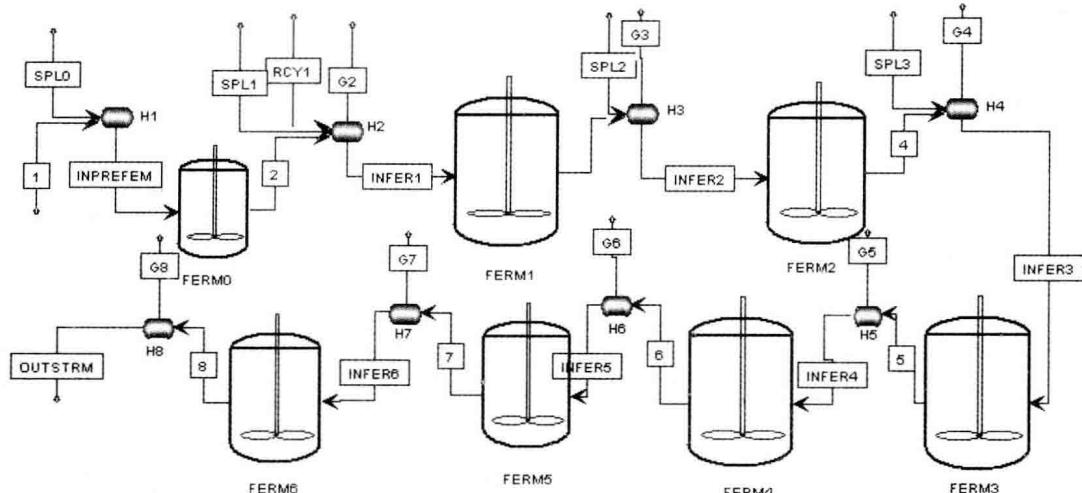


图 15.3-1 发酵过程模型流程图

表 15.3-2 模型进料组分表

| 名称   | 组成      | 浓度/ $\text{kg} \cdot \text{m}^{-3}$ | 流量/ $\text{m}^3 \cdot \text{h}^{-1}$ | 温度/°C | 压力/MPa |
|------|---------|-------------------------------------|--------------------------------------|-------|--------|
| 1    | Water   | 852.21                              | 0.886                                | 30    | 0.1    |
|      | Glucose | 1.07                                |                                      |       |        |
|      | Ethanol | 45.12                               |                                      |       |        |
|      | Yeast   | 37.47                               |                                      |       |        |
| SPL0 | Water   | 850.02                              | 7.02                                 | 30    | 0.1    |
|      | Glucose | 217.00                              |                                      |       |        |
| SPL1 | Water   | 850.02                              | 73.0                                 | 32    | 0.1    |
|      | Glucose | 217.00                              |                                      |       |        |
| SPL2 | Water   | 850.02                              | 22.28                                | 32    | 0.1    |
|      | Glucose | 217.00                              |                                      |       |        |
| SPL3 | Water   | 850.02                              | 11.14                                | 32    | 0.1    |
|      | Glucose | 217.00                              |                                      |       |        |
| RCY1 | Water   | 752.92                              | 7.62                                 | 32    | 0.1    |
|      | Glucose | 0.50                                |                                      |       |        |
|      | Ethanol | 100.00                              |                                      |       |        |
|      | Yeast   | 147.00                              |                                      |       |        |

预发酵罐体积小,主要用来缓冲各种物料。主发酵罐在整个流程中的作用是向系统中补充足够的具有活性的酵母。预发酵罐出料的变化情况对产品最终乙醇的含量影响并不大,此处不细致讨论。

## 15.4 发酵动力学方程组的 MATLAB 求解

乙醇连续发酵的仿真过程主要是对上述非结构型动力学方程的求解，并通过图像表达出来。鉴于 MATLAB 强大的数值计算能力和图形绘制能力，先使用 M 语言对乙醇连续发酵过程进行仿真。

从反应机理来说是葡萄糖在酵母作用下转化为乙醇，酵母通过吸收葡萄糖生长。从数学本质上来说，是对单罐连续发酵动力学模型求数值解。该问题涉及常微分方程系统 ODE 的求解。MATLAB 使用龙格-库塔-芬尔格(Runge-Kutta-Fehlberg)等方法求解 ODE 问题。相关的函数有 ode45、ode23、ode113、ode15s、ode23s、ode23t、ode23tb 等。格式为：

```
[T,Y] = solver(@odefun,tspan,y0,options,p1,p2,...)
```

参数说明：solver 为命令 ode45、ode23、ode113、ode15s、ode23s、ode23t、ode23tb 之一。odefun 为显式常微分方程  $y' = f(t, y)$  或包含混合矩阵的方程  $M(t, y) \times y' = f(t, y)$ 。命令 ode23 只能求解常数混合矩阵的问题；命令 ode23t 与 ode15s 可以求解奇异矩阵的问题。tspan 积分区间(即求解区间)的向量  $tspan = [t_0, t_f]$ 。要获得问题在其他指定时间点  $t_0, t_1, t_2, \dots$  上的解，则令  $tspan = [t_0, t_1, t_2, \dots, t_f]$ (要求是单调的)。y0 为包含初始条件的向量。options 为用命令 odeset 设置的可选积分参数。p1, p2, ... 等是传递给函数 odefun 的可选参数。

```
[T,Y] = ode(@rigid,[tspan],[init],options);
```

@是微分方程子函数的句柄符号，给 ode 函数提供一个地址入口；tspan 是求解的时间区间，比如[0 12]；init 是微分方程的初始值，比如[0 1 0]；options 是具体的误差等控制选项，此处使用默认值。T 是对时间区间根据步长分割的时刻点，Y 是对应时刻点的各分量的数值。

此处选用 ode45 函数来求解，综合整个动力学模型，对单罐的描述为函数 func(t, n, V)，其中：t：时刻(h)；n：各物料输入浓度( $\text{kg} \cdot \text{m}^{-3}$ )；V：发酵罐的体积( $\text{m}^3$ )；Vy：流入该发酵罐的料液流量( $\text{kg}/\text{h}$ )。

```
u = umax * n(3)/(Ks + n(3) + n(3)^2/Ksi) * Kp/(Kp + n(2) + n(2)^2/Kpi);
v = vmax * n(3)/(Kps + n(3) + n(3)^2/Kpsi) * Kpp/(Kpp + n(2) + n(2)^2/Kppi);
D = Vy/V;
f(1) = u * n(1) + D * (x0 - n(1));
f(2) = v * n(1) + D * (p0 - n(2));
f(3) = -(u/Yxs + v/Yps) * n(1) + D * (s0 - n(3));
```

程序中的各个参数基本采用了其本身表征符号，常数采用表 15.3-2 的数值，f(1)~f(3) 代表 3 种物料的微分量，通过叠代求得。n(1)~n(3) 代表但不完全同于原本的  $x, p, s$ ，主要是为了发挥 MATLAB 强大的向量计算能力，以便于在调用该子函数时输入简单明了，操作方便。调用该子函数进行求解的语句是：

```
[T, F] = ode45(@func,[0 0.5 time0],[x0 p0 s0]);
```

用 MATLAB 自带的 ode45 函数求解很方便，但是同时也出现一个问题，ode45 是变步长的，在震荡剧烈的地方取点密，步长小，在比较平缓的地方取点稀疏，步长大。实际生产过程控制很难做到随时改变数据采样时间。

龙格-库塔(Runge-Kutta)方法是一种在工程上应用广泛的高精度单步算法。由于此算法精度高，采取措施对误差进行抑制，所以其实现原理也较复杂。

利用这样的原理，经过复杂的数学推导(见参考文献[12])，可以得出截断误差为  $O(h^5)$

的四阶龙格-库塔公式为：

$$\begin{aligned} K_1 &= f(X_n, Y_n); \\ K_2 &= f(X_n + h/2, Y_n + (h/2) \times K_1); \\ K_3 &= f(X_n + h/2, Y_n + (h/2) \times K_2); \\ K_4 &= f(X_n + h, Y_n + h \times K_3); \\ Y_{n+1} &= Y_n + h \times (K_1 + 2K_2 + 2K_3 + K_4) \times (1/6); \end{aligned}$$

为了更好更准确地把握时间关系,应自己在理解龙格-库塔原理的基础上,编写定步长的龙格-库塔函数。

对微分方程组,可以想象成多个微分方程并行进行求解,时间、步长都是共同的,首先把预定的初始值赋值给每个微分方程,然后每迭代一步,对多个微分方程共同求解。在不使用 MATLAB 提供的 odefun 情况下编写定步长的龙格-库塔函数求解常微分方程组的程序如下:

```
function [x,y] = runge_kutta(ufunc,y0,h,a,b,Vg) % 参数表顺序依次是微分方程组的函数名称,初始
% 值向量,步长,时间起点,时间终点,发酵罐体积
n = floor((b-a)/h); % 求步数
x(1) = a; % 时间起点
y( :, 1) = y0; % 赋初值,可以是向量,但是要注意维数
for ii = 1 : n
    x(ii+1) = x(ii) + h;
    k1 = ufunc(x(ii),y( :, ii),Vg); % Vg 可以根据 ufunc 参数做取舍
    k2 = ufunc(x(ii) + h/2,y( :, ii) + h * k1/2,Vg);
    k3 = ufunc(x(ii) + h/2,y( :, ii) + h * k2/2,Vg);
    k4 = ufunc(x(ii) + h,y( :, ii) + h * k3,Vg);
    y( :, ii+1) = y( :, ii) + h * (k1 + 2 * k2 + 2 * k3 + k4)/6; % 按照龙格-库塔方法进行数值求解
end
```

调用的子函数及其调用语句:

```
function dy = test_fun(x,y)
dy = zeros(3,1); % 初始化列向量
dy(1) = y(2) * y(3);
dy(2) = -y(1) + y(3);
dy(3) = -0.51 * y(1) * y(2);
```

对该微分方程组用 ode45 和自编的龙格-库塔函数进行比较,调用如下:

```
[T,F] = ode45(@test_fun,[0 15],[1 1 3]);
subplot(121)
plot(T,F) % MATLAB 自带的 ode45 函数效果
title('ode45 函数效果')
[T1,F1] = runge_kutta(@test_fun,[1 1 3],0.25,0,15); % 测试时改变 test_fun 的函数维数,别忘
% 记改变初始值的维数,Vg 此处舍去
subplot(122);
plot(T1,F1); % 自编的龙格-库塔函数效果
title('自编的显式龙格-库塔函数');
```

运行结果如图 15.4-1 所示。

由图可知,在相同时间范围内,以同样的初始值开始计算,得到的图形基本相同,说明了该龙格-库塔方法计算程序的可靠有效性。由于自编龙格-库塔函数是定步长的,所以在尖锐的区域,其准确性不如变步长的 ode45 函数,却如实地反应了实际传感器获得数据的方式:等间隔时间的采样。而且由单罐仿真波形图 15.4-1 可知,在图中并没有突变的地方,所以自编的龙格-库塔函数是完全符合要求的。

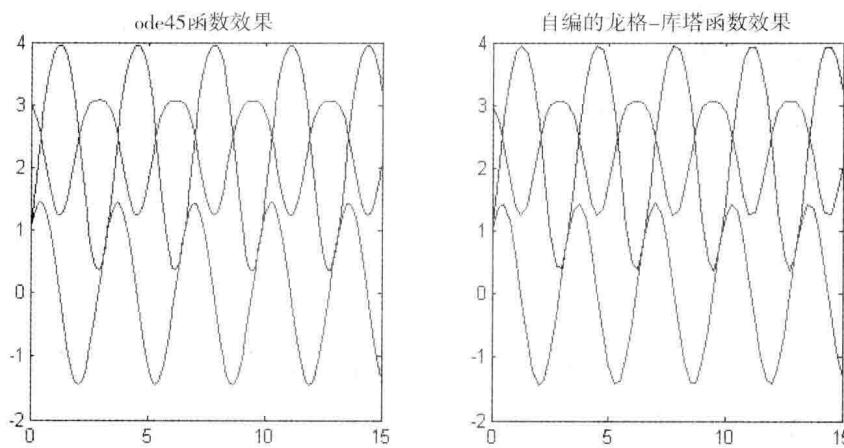


图 15.4-1 自编龙格-库塔函数与 ode45 的效果对比

如此一来,根据 `odefun` 或自编的龙格-库塔算法都可以求解单罐发酵动力学方程组,将求解出来的结果最后一个时刻的值作为下一次求解的初始值,反复求解 7 次,就可以仿真一个预发酵罐和 6 个发酵罐串联的发酵效果了,此处不再赘述 M 代码的实现。

## 15.5 发酵动力学方程组的 Simulink 求解

图 15.2-1 所示系统由一个预发酵罐和 6 个发酵罐串联构成,虽然大小和初始物料浓度等因素不同,但反应原理是一样的,即先仿真出一个发酵罐的动态反应情况是前提。对于一个发酵罐的仿真,其本质上就是在给定数据条件下求单罐连续发酵动力学模型的数值解。Simulink 模型的 Configuration Parameter 中提供的求解方法有许多,包括变步长解算方法和固定步长解算方法,如 Dormand-Prince 法、Runge-Kutta 法、Bogacki-Shampine 法、Adams 法和 Euler 法等。本文采用比较常用的 4 阶 Runge-Kutta 法来求解,或者采用模块仿真,将所要求解的微分方程组搭建成模型,通过循环迭代逼近求解其数值解。

尽管 Simulink 进行系统动态仿真很方便,但是用 Simulink 仿真发酵动力学微分方程组的问题比较棘手。许多资料都是关于计算机仿真技术和控制系统仿真的,有不少都讲到了关于微分方程的 Simulink 求解,但都是线性的,即使非线性的例子也只是因为三角函数产生的非线性,发酵动力学方程中分母上存在变量,是比较难解决的问题。总结一下通常的 3 种解法:

- ① 按照方程搭建模型,对于本课题来说这是最复杂的方法,但是任何方程都适用;
- ② 用 S 函数,但需要求解状态方程,但非构造型微分方程组不容易求出其状态方程;
- ③ 用传输函数,对时域方程进行拉普拉斯变换,该方程组无法进行,即使用 MATLAB 求也得不到解析解。
- ④ 向量法,用 Fcn 模块表示每一个微分方程中的一维输入,三个 Fcn 模块并联,所以,本质上同 `ode` 函数是一样的思想。Fcns 模块只允许用向量  $u$  表示的输入,如果是 3 个输入量,用  $u(1)、u(2)、u(3)$  表示。仿真时需要给定进料的初始浓度  $x_0, P_0, S_0$ ,直接把这几个符号变量写到 Fcn 的向量表达式中会报错,使用全局变量又显得繁冗。所以通过 mux 将  $x_0, P_0, S_0$  合并作为 Fcn 模块的输入,以  $u(4) \sim u(6)$  来表示。通过积分器构成迭代环,求解出每个采样时刻的微分方程组数值解。单罐模型如图 15.5-1 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

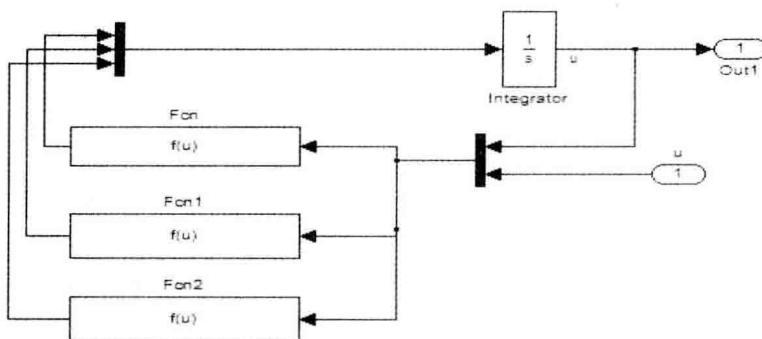


图 15.5-1 单个发酵罐模型

Fcn 模块中编写菌体、葡萄糖和乙醇 3 种物料的浓度微分方程迭代表达式, 表达方式都是  $du = \text{expression}(u)$  的形式, 只需要将  $\text{expression}(u)$  填写到 Fcn 模块中, 菌体的表达式如图 15.5-2 所示。

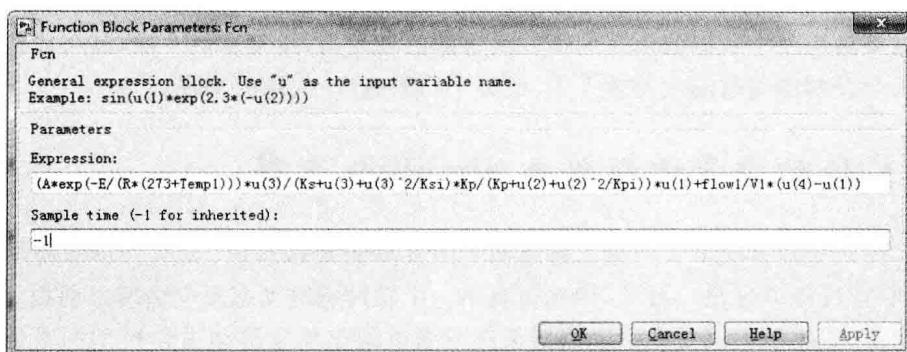


图 15.5-2 Fcn 模块中菌体浓度的迭代表达式

表达式中的各个常数变量均在模型的 InitFcn 中进行定义, 每次运行模型仿真时会自动运行并赋值到 Base Workspace 中。

把图 15.5-1 的模型封装为子系统, 使用 Constant 模块给定各个物理的初始浓度, 进行针对单罐模型的仿真, 图形如图 15.5-3 所示。

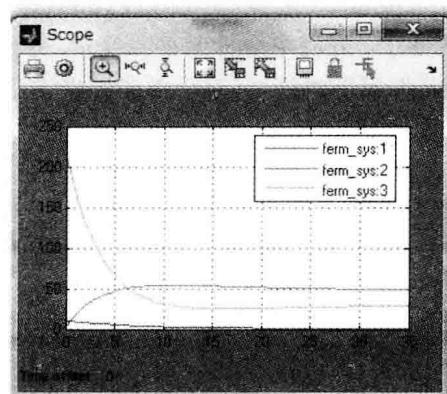


图 15.5-3 Simulink 单罐仿真曲线

可以看出该模型具有良好的动态特性和稳定性,3个物理量的浓度最终会稳定在某个值并保持平衡,符合发酵反应的机理。

## 15.6 乙醇连续发酵流程的 Simulink 仿真

将一个预发酵罐和6个主发酵罐串联起来,并且把各个进料添加口也加入到系统中,设各个物质在进入发酵罐时瞬间混合均匀,为此,本文在仿真系统中的发酵罐前增加一个混合器(mixer)来完成进料时混合浓度计算的功能。系统模块化仿真图如图 15.6-1 所示。

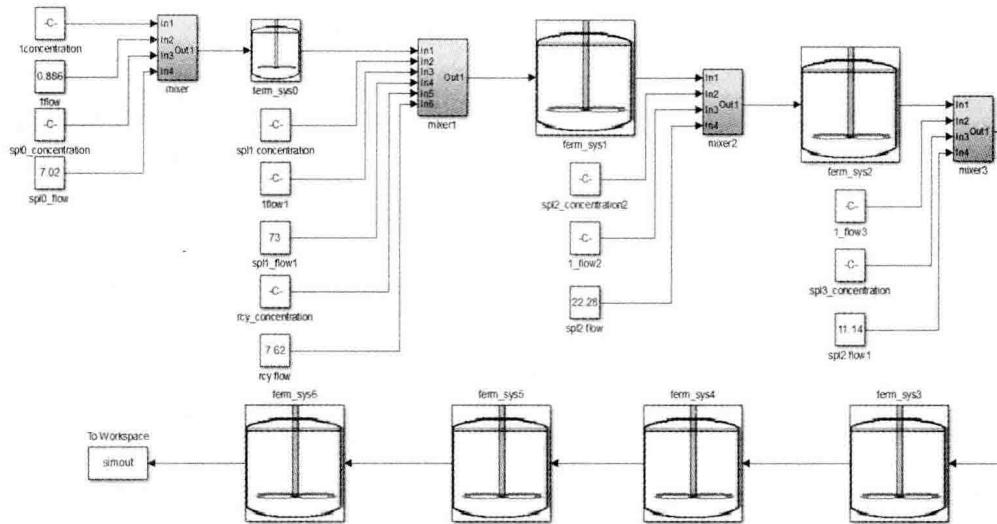


图 15.6-1 乙醇连续发酵系统基本模型

根据实际工业乙醇发酵流程时间,设置该系统仿真时间 32 s 代表 32 h 的运行过程,采用固定点解算器,步长为 0.1 s 表示 0.1 h 采样一次,运行结果(7 个发酵罐的物质浓度曲线)如图 15.6-2 所示。

从仿真图观察,在预发酵罐和发酵罐 1 中糖类的分解并不彻底,乙醇产量也不够高。随着流程推进到最后一个发酵罐 6 中,糖类已经基本分解完毕,乙醇浓度也达到  $100 \text{ kg/m}^3$  以上。

在上述模型基础上,将温度因素对发酵过程的影响也考虑进来,用 Arrhenius 方程描述温度对菌体生长的影响,温度对产物乙醇的影响表现在酶的活性及反应速率上,而不会直接影响比生产率,故不予考虑。菌体的比生长率随温度变化的表达式为:

$$\mu_t = Ae^{-E/[R(273+T)]} \quad (15.6-1)$$

式中: $\mu_t$ ——温度影响下的菌体比生长率;

$A$ ——生长常数  $8.46 \times 10^{14}$ ;

$E$ ——生长活化能  $9.23 \times 10^4 \text{ J}$ ;

$R$ ——气体常数 [ $R=8.28 \text{ J}/(\text{mol} \cdot \text{K})$ ];

$T$ ——摄氏温度。

将  $\mu_t$  的计算式代入单罐发酵动力学方程组(15.3-3)中,代替常量  $\mu_{\max}$  得到新的菌体生长率表达式:

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

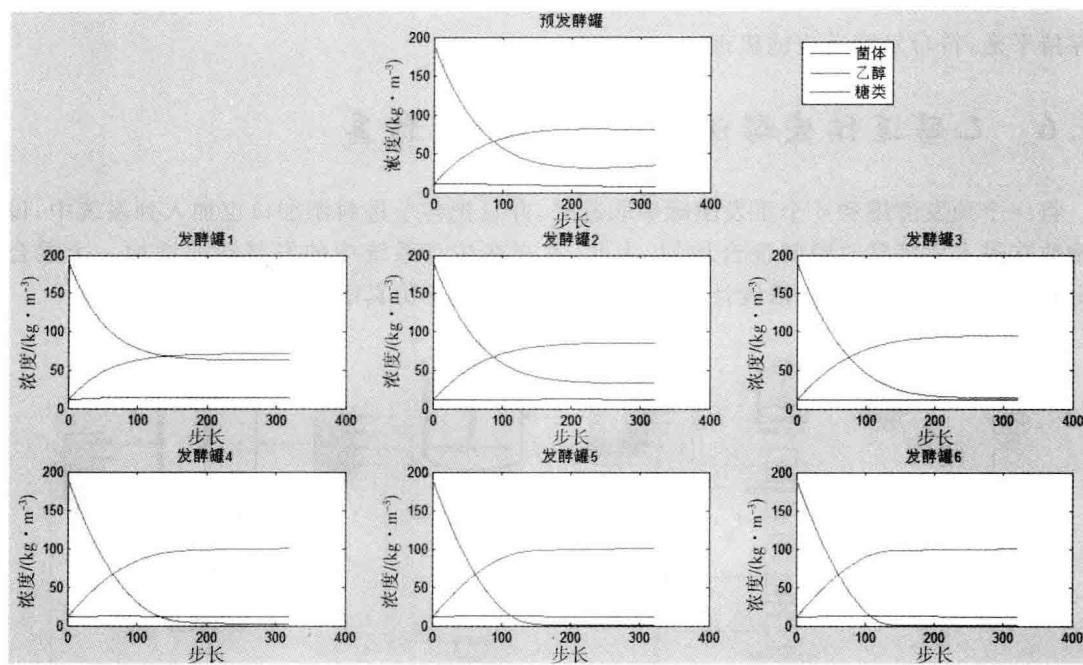


图 15.6-2 乙醇连续发酵系统基本模型仿真图

$$\mu = Ae^{-E/[R(273+T)]} \frac{s}{K_s + s + s^2/K_s} \frac{K_p}{K_p + p + p^2/K_{p\ell}} \quad (15.6-2)$$

将其应用到 Simulink 连续发酵动力学方程中, 仿真时每个罐子都多了一个参数——反应温度。

由于实际工业生产所购买的原料浓度与规定的浓度也存在些微可以容忍的浓度误差, 这个误差作为浓度波动反映到仿真模型中。任何实际工业生产过程总会伴随不可控的扰动出现, 使用随机数 random Number 模块, 在各个入料环节增加高斯白噪声模拟生产过程中的扰动, 扰动模块如图 15.6-3 所示。

将这个扰动模块作为一个乘法因子跟各个入口点的物料浓度相乘之后, 扰动模型如图 15.6-4 所示。

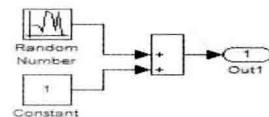


图 15.6-3 扰动发生模块

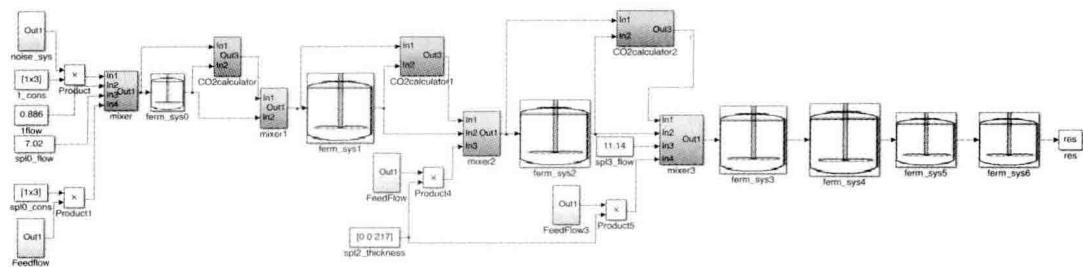


图 15.6-4 扰动模型

最终添加了温度因素和浓度波动的模型仿真图如图 15.6-5 所示。

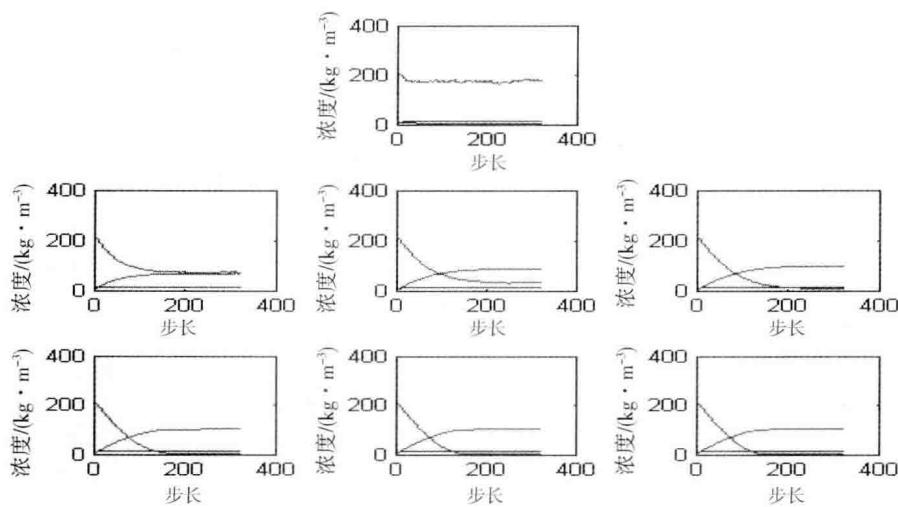


图 15.6-5 扰动模型仿真图

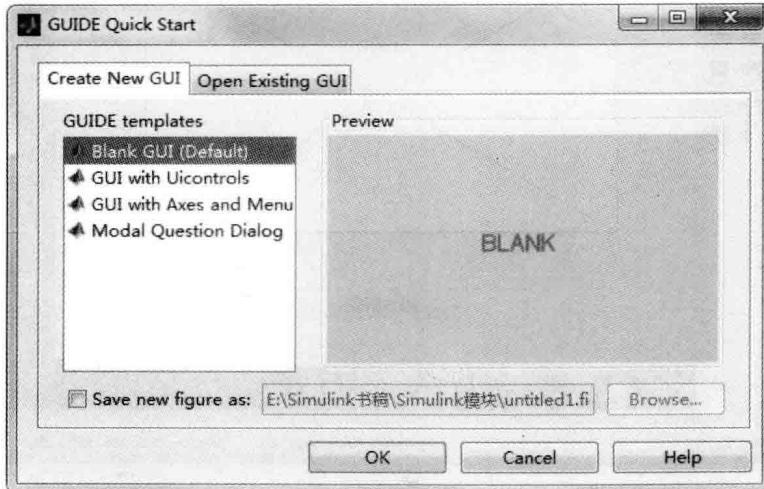
可以看出,输入扰动的存在在预发酵罐中得到较明显的体现。经历多次混合后扰动逐渐消失,说明了该模型的稳定性,也较好地体现了真实的发酵过程。

## 15.7 乙醇连续发酵的仿真软件设计

### 15.7.1 GUIDE 介绍

GUI(Graphical User Interface)是图形用户界面的意思,像很多高级编程语言一样,MATLAB 也有图形用户界面开发环境,随着计算机技术的飞速发展,人与计算机的通信方式也发生了很大的变化,从原来的命令行通信方式(例如很早的 DOS 系统)变化到了现在的图形界面下的交互方式,现在绝大多数的应用程序都是在图形化用户界面下运行的。

① 新建一个 GUI 文件:在命令窗口 Command Window 输入 GUIDE,启动界面如图 15.7-1 所示。



若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 15.7-1 GUIDE 启动界面

② 选择 Blank GUI(Default), 单击 OK 按钮, 弹出如图 15.7-2 所示图像。

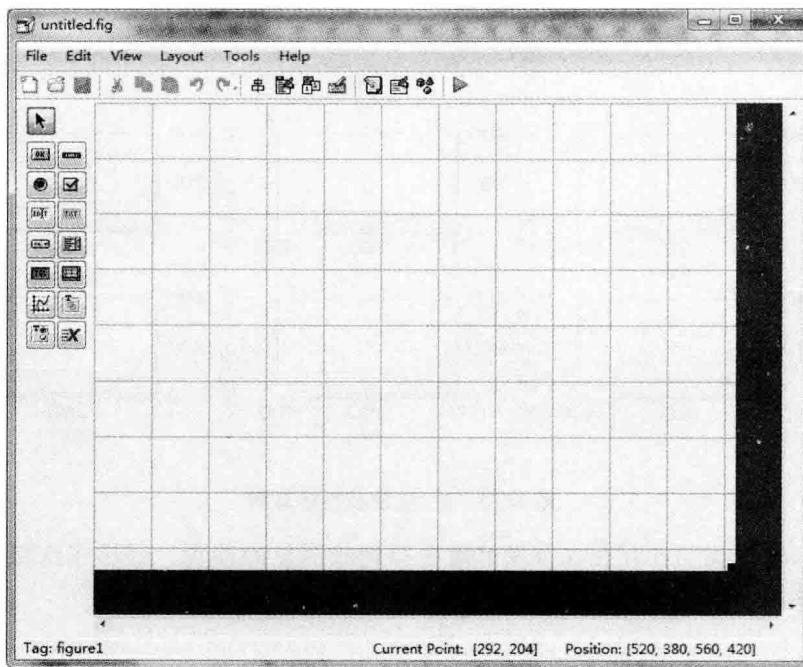


图 15.7-2 GUI 设计版面

③ 窗口的左边有许多控件, 类似 VB, 可以将控件拖到窗口中。比如做一个按钮控制图片显示的界面, 拖曳 axes 及 pushbutton 控件到界面中, 使布局如图 15.7-3 所示。

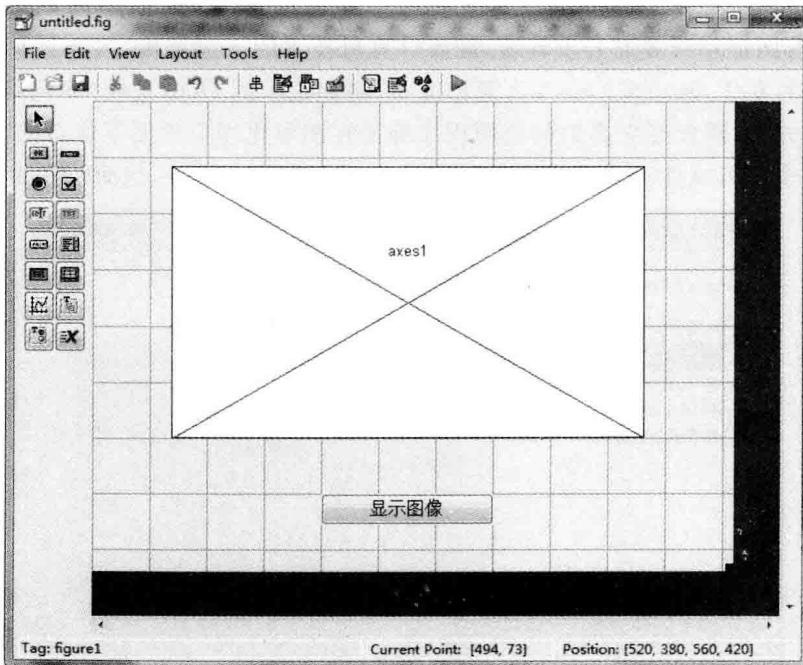


图 15.7-3 拖动控件的实例

按钮的显示字符和字体大小可双击按钮，在弹出的对话框中修改。

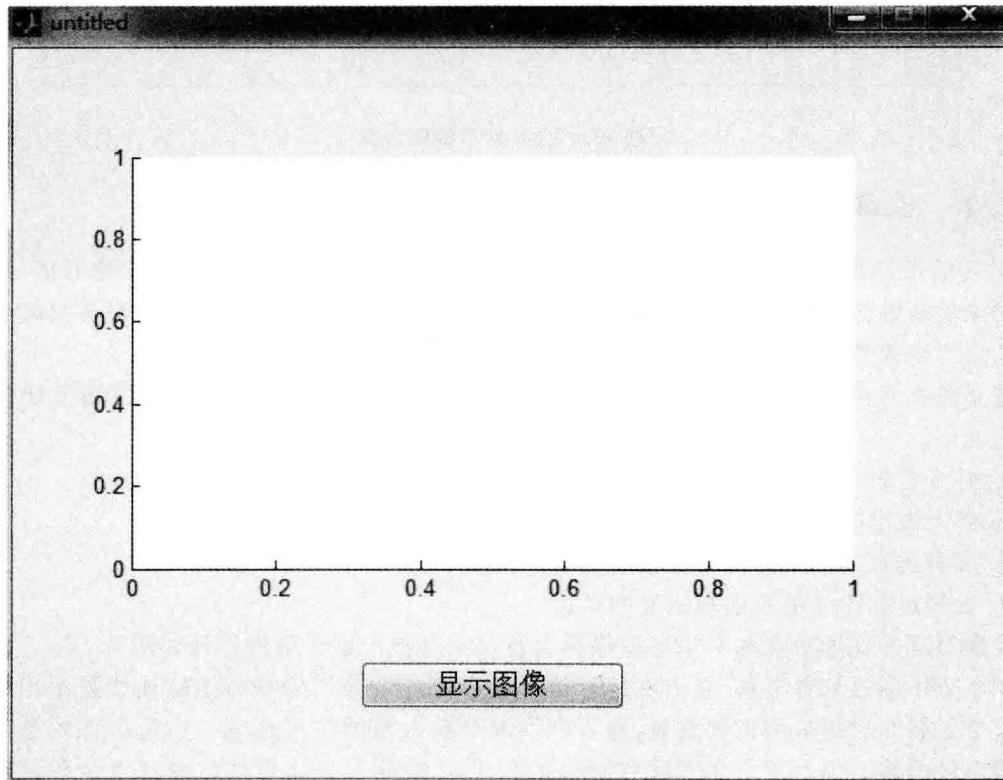
④ 对按钮控件功能的编程：

右击按钮，选择 View callbacks→Callbacks 回调函数，弹出界面的 M 文件，光标定位在该按钮的子函数开始处，写好单击该按钮时触发的语句如图 15.7-4 所示。

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
imshow('flower.jpg');
```

图 15.7-4 按钮的回调函数代码

⑤ 保存文件，单击 Fig 文件窗口上的三角或者运行其 M 文件以执行该程序，弹出如图 15.7-5 所示 GUI 界面。Fig 被运行后如图 15.7-4 所示。



若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 15.7-5 运行设计的界面

axes 呈现出空白的状态，单击已经编辑好回调函数的“显示图像”按钮，则可以将图像显示到 Fig 上，如图 15.7-6 所示。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。



图 15.7-6 按下按钮之后

### 15.7.2 乙醇连续发酵程序仿真软件界面

GUI 的设计是为了给实验室环境提供一种可靠的数据获取手段和决策安排方法,综合了之前讨论的所有仿真因素在内。在给定各个环节具体数据的情况下,能够得到与实际生产相贴近的生产结果数值。

鉴于以上考虑,将用户能调节的参数、需要观察的仿真结果(数据或图像)都设计到 GUI 界面上:

- ① 各个进料口的流量在合理的范围内可以调节;
- ② 各个发酵罐的发酵温度可以调节;
- ③ 发酵过程的曲线图的可视化;
- ④ 发酵过程的详细数据和结果的显示;

发酵过程的详细数据和结果可以保存为自定义文档。设计软件界面如图 15.7-7 所示。

图上方中间是软件名称,左方在运行仿真后会把各个罐子的物料浓度曲线显示出来。右边最上方是各个进料口浓度的设置,输入框下面是输入值的许可范围。右边中部是各个发酵罐发酵温度的输入值。下方是发酵数据的显示,可以控制只显示最终浓度或者全程详细浓度的显示。将显示的数据通过保存按钮保存为 xls 文件,用于科学研究或者多次仿真实验的对比等。已保存的文件也可以通过清除按钮删除掉。右下角的按钮是启动仿真或者退出该界面的功能,右边显示的是主要发酵的时间长度。

具体步骤如下:

- ① 标题,将左边 text 标签拉动到界面窗口,双击后在属性窗口中修改 string 和 font size 属性即可。

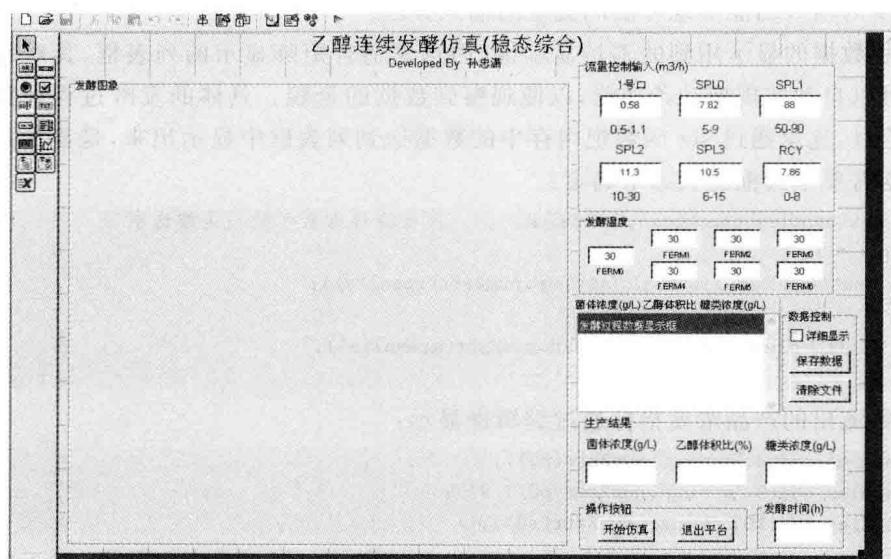


图 15.7-7 稳态仿真软件界面

② 流程上各个进料口流量值的输入,标签的处理方式同①。要特别注意数值格式的转换问题,在界面上显示的是字符串,而不数据类型,所以要用 str2num 函数进行数据类型转换。另外将数据从输入框取到内存的函数是 get,此处要取出 6 个数据,取出数据在仿真开始之前进行,需写在开始仿真按钮的回调函数中:

```
lowy = str2num(get(handles.edit1,'string')); % 预发酵罐的流速
flow0 = str2num(get(handles.edit2,'string')); % SPL0 的流速
flow1 = str2num(get(handles.edit3,'string')); % SPL1 的流速
flow2 = str2num(get(handles.edit4,'string')); % SPL2 的流速
flow3 = str2num(get(handles.edit5,'string')); % SPL3 的流速
flowr = str2num(get(handles.edit6,'string')); % RCY 的流速
```

用户有时给定的输入数据不在合理范围内,进行仿真将得到荒谬的结果数据,为了避免这种情况发生,当用户填入非法数据时应,弹出错误对话框,提醒用户修改不合理的数据,代码如下:

```
if flow3<6|flow3>15
    errordlg('请输入符合实际的 SPL3 流量 ','Error');
end
if flow2<10|flow2>30
    errordlg('请输入符合实际的 SPL2 流量 ','Error');
end
if flow1<50|flow1>90
    errordlg('请输入符合实际的 SPL1 流量 ','Error');
end
if flow0<5|flow0>9
    errordlg('请输入符合实际的 SPL0 流量 ','Error');
end
if flowy<0.5|flowy>1.1
    errordlg('请输入符合实际的 1 号口流量 ','Error');
end
if flowr<0|flowr>8
    errordlg('请输入符合实际的 RCY 流量 ','Error');
end
```

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

③ 温度的输入功能实现方法同流量的输入方法。

④ 过程数据的显示用到的不是显示框,而选用适合矩阵显示的列表框,其对超出显示范围的数据可以自动实现拖动条功能,以便观察到数据的全貌。具体的发酵过程计算在前面几章已经讨论过,这里通过 set 函数把内存中的数据送到列表框中显示出来,是显示具体数据还是最终数据需要单选框的状态来确定:

```
flag = get(handles.checkbox1,'value'); % 检测详细显示是不是被选中了
if flag == 0
    set(handles.listbox1,'string',num2str(results));
else
    set(handles.listbox1,'string',num2str(dresults));
end
```

⑤ 最终流出的产品浓度指标通过编辑框显示:

```
set(handles.edit8,'string',num2str(x0));
set(handles.edit9,'string',num2str(p0/7.9));
set(handles.edit10,'string',num2str(s0));
```

⑥ 数据保存功能:将列表框中的数据写到 xls 文件中,便于观察和使用,对于数据的导入也是十分方便的。

```
data = get(handles.listbox1,'string');
xlswrite('dataT.xls',str2num(data));
msgbox('已经保存为 dataT.xls 文件');
```

删除文件的同时也给出消息提醒:

```
delete('dataT.xls');
msgbox('已经清除 dataT.xls 文件');
```

⑦ 发酵过程曲线的绘制,需要先选中一个坐标轴,再调用 plot 函数绘制计算出来的各物料浓度值。主体程序的核心仍然是动力学微分方程组的求解,增加的部分是 GUI 数据的获取与显示。写入完整的回调函数后运行,输入参数,进行仿真,GUI 界面如图 15.7-8 所示。

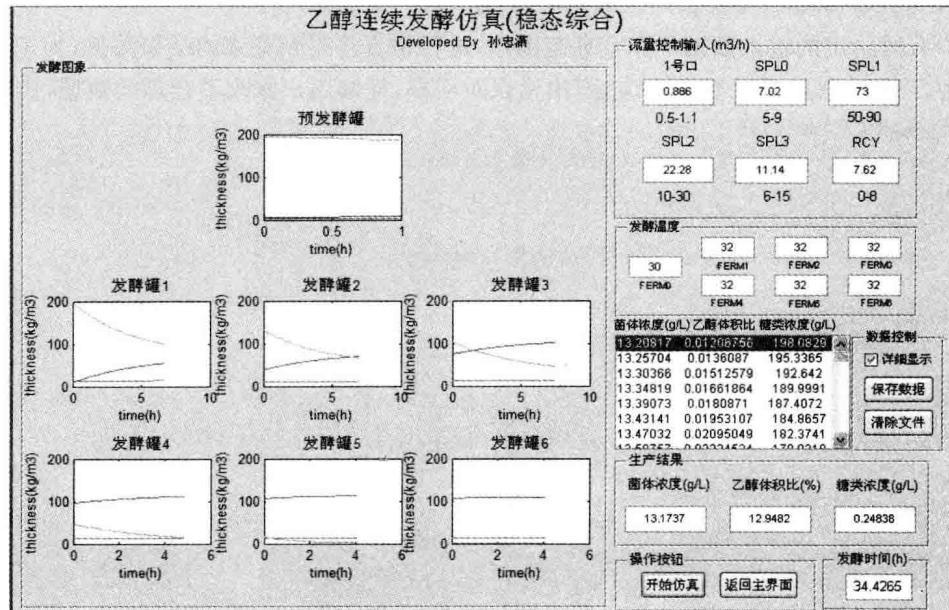


图 15.7-8 稳态仿真软件运行界面

### 15.7.3 Simulink 动态仿真控制器制作

动态仿真基于 Simulink 模型的。为了将模块化仿真功能综合到仿真软件中去,需要设计一个简单的控制器来控制模型的打开、运行和关闭动作。鉴于 MATLAB/Simulink 及 GUI 之间强大的协调工作能力,该功能可如此设计:

① 数据在 GUI 工作空间和 Simulink 工作空间的传递可以使用 evalin 和 assignin 函数实现,也可以使用全局变量 global 声明实现,此处笔者通过制定模型使用的工作空间为 GUI 回调函数工作空间实现二者数据在同一工作空间存储。

② GUI 控件控制 Simulink 模型的动作可使用 open\_system、close\_system 和 set\_param 等函数操作 Simulink 模型及其属性。

考虑上述因素之后,通过控件拖曳,布局如图 15.7-9 所示 GUI 界面设计。

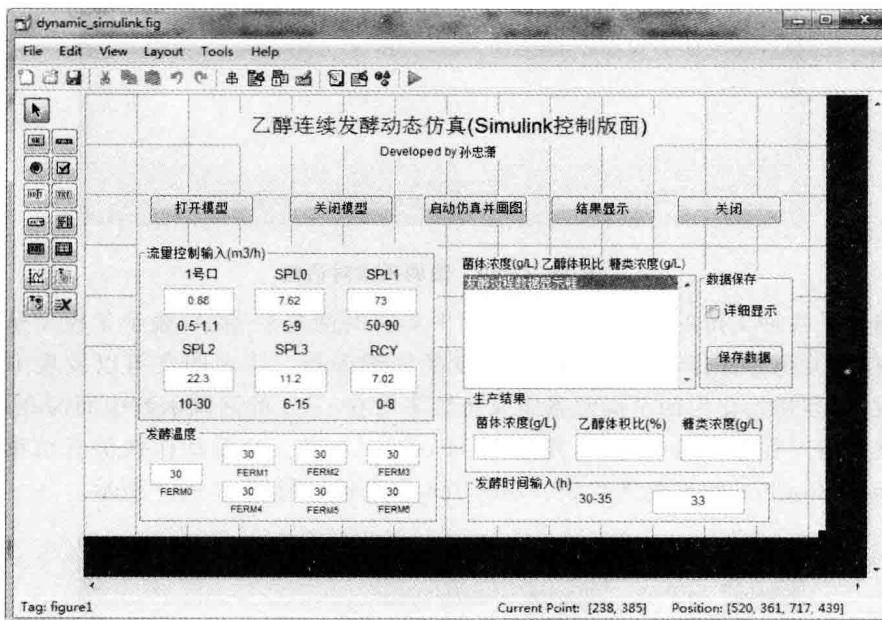


图 15.7-9 Simulink 控制界面

首先设计和控制版面,根据需要修改各个控件的属性和名称,对 3 个控制按钮分别编写其回调函数就可以了。

打开模型的回调函数:

```
open_system('integrated_model');
```

关闭模型的回调函数:

```
close_system('integrated_model');
```

**注意:** 此处 open\_system 函数的参数是需要打开或关闭的模型文件名,不需要后缀。

启动仿真绘图回调函数,运行 mdl 文件之前,需要将动态仿真系统中的各个全局常量参数初始化,以及获取用户从 GUI 界面输入的参数值之后,再启动模型进行仿真。从 Simulink 模型获取仿真结果数据,再在回调函数中绘图将数据结果可视化。GUI 与 Simulink 模型的衔接在于二者之间数据的传递。这种做法虽然比较典型,但却未发挥 Simulink 模型文件环境下提供的各种回调函数机制。此处模型仿真结果通过 to workspace 模块传递到 Base work-

space,再使用 M 代码编写绘图程序绘制仿真结果图像。模型仿真操作过程中的回调函数编辑框可以通过选中 File→Module properties 后打开,选中 Callbacks,界面如图 15.7-10 所示。

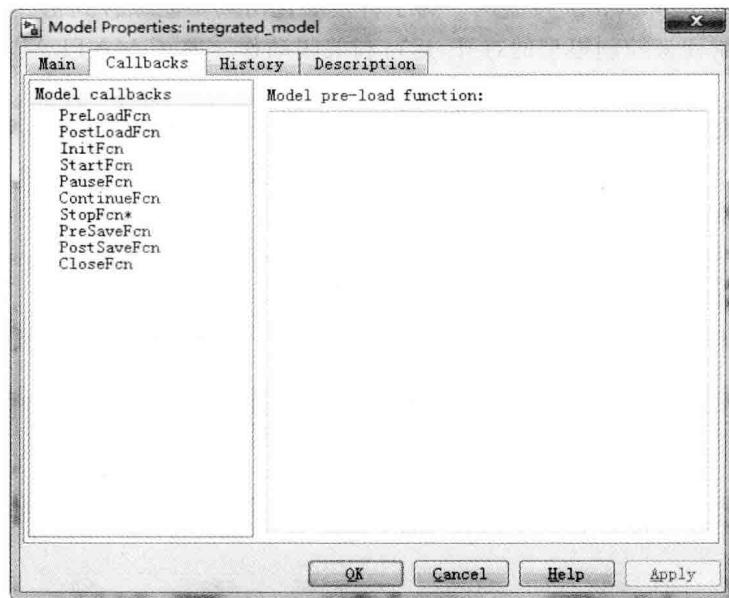


图 15.7-10 回调函数对话框

左边列出了各种支持的回调函数,从上到下具有时间先后顺序,表示了模型从被打开、运行仿真、保存和关闭等操作动作执行时所触发的回调函数。读者朋友可以发现 InitFcn 表示初始化函数,模型初始化用的全局常量定义可以不写在 GUI 的回调函数中而写在这里。右边的对话框就是编写程序代码的地方,用法同 M-editor 一样。绘图动作在仿真结束之后执行,使用回调函数 StopFcn 编写绘图代码。StopFcn 的代码如图 15.7-11 所示。

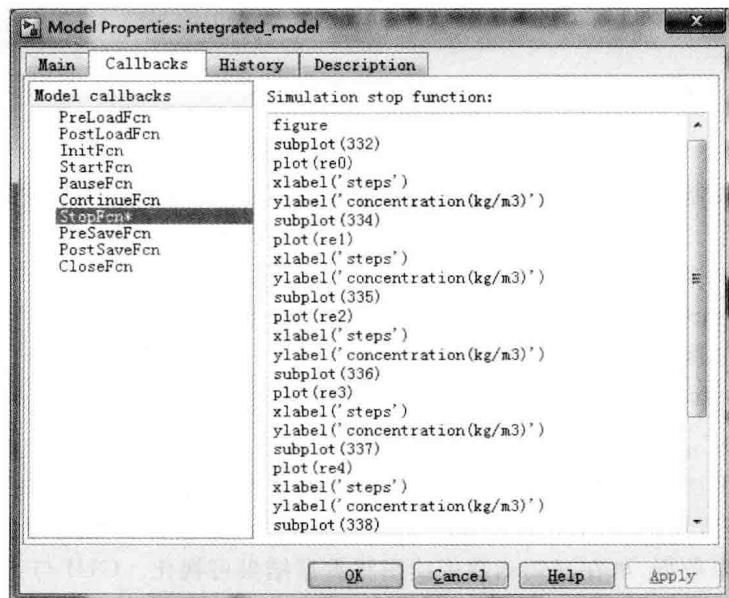


图 15.7-11 绘图函数

如果每个罐子的仿真结果都连接 Scope，则运行之后弹出窗口太多，窗口密集，界面不友好，而且逐一关闭也是一件麻烦事。如果通过并行方法接到一个总的 Scope 上，又因图像线条过多难以区分各发酵罐的情况；使用 To Workspace 模块，Save format 选择 Array，仿真结果数据会保存到当前模型运行所处的 Workspace 中（即 GUI 的回调函数工作空间），以矩阵形式存在，仿真结束时调用 StopFcn 回调函数绘制图像，这样用户使用时只要在 GUI 界面上打开模型，单击运行就可以获得结果图像了。GUI 界面及仿真后图像如图 15.7-12 所示。

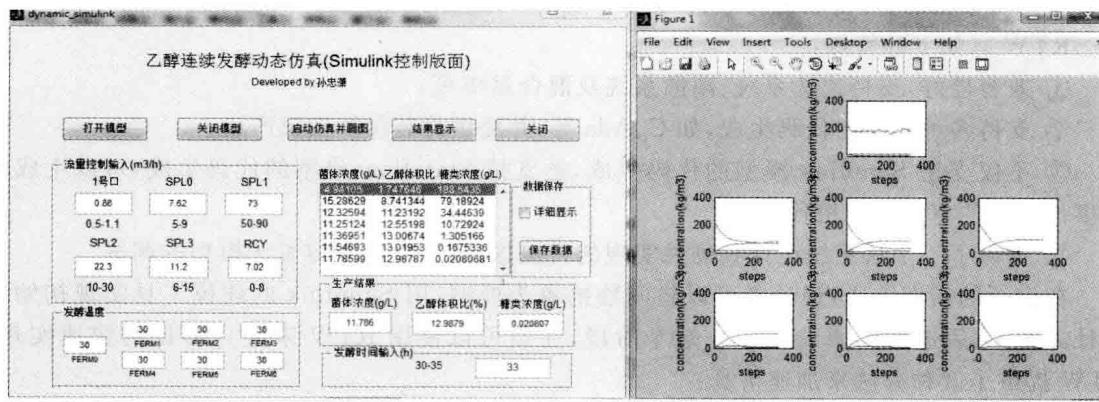


图 15.7-12 设置好回调函数后运行模型的结果

在 GUI 界面输入的数据被内存获取之后并不送入 Base Workspace，但是 Simulink 仿真需要的数据默认都是从 Base Workspace 获取，这是一个矛盾。

Simulink 中有个变量为 f1，编辑框 Tag 属性为 edit1，相应的代码为 f1 = str2num(get(handles.edit1,'string'))，其中，str2num 是将字符串转变为数值的函数。

启动 Simulink 模型仿真之前需要写上 options = simset('SrcWorkspace','current')。

设置 Simulink 模型中的各个变量的搜索位置是在当前工作空间——GUI 的 Callback 工作空间上，通过语句

```
sim('model_name',[0 sim_time],options);
```

启动仿真。如果不设置模型运行的工作空间，则模型默认到 Base Workspace 里寻找变量，会导致仿真出错。运行仿真后的数据传递，一般情况下将 Simulink 模型仿真所得到的数据通过 To Workspace 输出到 Base Workspace 中，再从 Base Workspace 载入 GUI 的回调函数 workspace 里。需要做如下处理：首先将 Simulink 模型中期待输出的数据用 To Workspace 模块连接，再通过语句

```
assignin('base','re0',re0);
```

才能将其内容保存到 Base Workspace 中，再从 Base Workspace 中读取到 GUI 回调函数 workspace 中，通过语句

```
re0 = evalin('base','re0');
```

实现。由于之前通过

```
options = simset('SrcWorkspace','current');
```

的设置，Simulink 模型已经不再默认从 Base Workspace 获取数据，而是直接从“仿真并绘图”按钮的回调函数 workspace 里获取和存储数据，由于工作空间已经统一，故不需要再进行 as-

signin 和 evalin 的操作。

### 15.7.4 基于代码生成的模型仿真加速

MBD 之所以能够在 MATLAB/Simulink 平台上广泛应用,是因为该软件提供了一个实时工作站 Real Time Workshop 与 Real Time Workshop-EC(MATLAB 2013b 中分别称之为 Simulink Coder 与 Embedded Coder),将上述实施工作站统称 RTW。RTW 作为 MATLAB 中极其重要的工具箱,给使用者提供了一个基于 Simulink 的代码生成环境,能将 Simulink 模型转换为可移植的、可定制的高效代码。

RTW 有如下的特点:

- ① 兼容性好,支持连续系统、离散系统及混合系统等;
- ② 支持多种语言的代码生成,如 C、Ada 等,并提供封装保护知识产权;
- ③ 不仅支持 Simulink 模型的代码生成,还支持 Stateflow 模型的代码生成,可以生成基于事件驱动的有限状态机代码;
- ④ 提供了一个从系统设计到硬件实现的直接途径,同时还兼容基于模型的调试。

在设计的初期阶段,设计者可以将问题抽象为模型,用 Simulink 的建模工具实现初始的设计方案,然后进行仿真验证。在这个阶段,完全可以用 RTW 来进行验证的快速实现。RTW 提供了 3 种方法来加速仿真:

- ① Simulink 加速器目标(Simulink Accelerator Target):相对标准的仿真过程可以提高 2~8 倍的速度,并且支持定步长和变步长解算器;
- ② 快速仿真目标(Rapid Simulation Target):相对于标准的仿真过程提高 5~20 倍速度,适合处理批量参数的仿真;
- ③ S 函数目标(S-Function Target):效果与 Simulink 加速器相似,编译为动态链接库后,加入到模块中,作为一个 S-Function Block 使用。

S 函数的可执行文件 mex 文件是一种动态链接对象,类似 Windows 的 DLL 文件。主要用于扩展 Simulink 的建模环境。通过 S 函数开发,用户可将自定义模块算法加入到 Simulink 环境中。S 函数具有很强的灵活性,既能够实现复杂的算法或方程组,也能够实现底层的设备驱动程序。RTW 支持 S 函数的使用,可将 S 函数代码直接内嵌到所生成的代码中。内嵌 S 函数(RTW 提供的目标语言编辑器 TLC 支持该功能)可以有效地减少内存使用量和函数调用的开销。

快速仿真目标包含了一些目标文件,用于模型所在机器非实时运行。在使用快速仿真目标时,用户通过 RTW 生成快速的用于单机的程序进行仿真,这种仿真程序还支持批量的参数调节,更高效的是能够直接从 MATLAB 的数据文件.mat 文件中读取仿真数据,而无须对模型重新编译。

S 函数目标可将模型转化为 S 函数组件,S 函数在 Simulink 中的地位极其重要,适用于各种其他模型,这种 S 函数虽然需要手动编写,但是在代码重用上能够发挥很大作用。在共享过程中,可以通过 S 函数把模块封装起来,隐藏细节,是一种保护开发者知识产权的方式。

第 1 种加速目标器与 S 函数目标相似,即从模型中生成 S 函数;不同之处在于:加速器目标所生成的 S 函数只是在后台执行。加速器目标能加快仿真速度,同时保留所有的仿真功能,如改变参数、信号可视化等。

## 1. RTW 工作过程

模型设计好后,RTW 可以开始对模型进行解析,主要包括:仿真的解算器和参数、信号维度和系统采样时间、模型中各个模块的先后计算顺序、计算 S 函数使用的工作向量的大小。

模型文件 model.mdl 编译之后,生成中间文件 model.rtw,它是一种信息文本,记录了模型中所有的属性和参数信息,如图 15.7-13 所示。

```

1 CompiledModel {
2     Name          "assay"
3     OrigName      "assay"
4     Version       "8.1 (R2011b) 08-Jul-2011"
5     SimulinkVersion "7.8"
6     ModelVersion   "1.229"
7     GeneratedOn    "Tue Dec 18 21:14:04 2012"
8     HasSimStructVars 0
9     HasCodeVariants 0
10    PreserveExternInFcnDecls 1
11    ExprFolding    1
12    TargetStyle     "StandAloneTarget"
13    NumDataStoresGlobalDSM 0
14    RightClickBuild 0
15    ModelReferenceTargetType "NONE"
16    AllowModelRefFcnCallInputs 1
17    AllowNoArgFcnInReusedFcn 0
18    StandaloneSSSupported 0
19    StandaloneSubsystemTesting 0
20    PadderActive    0
21    DWorkAndBlockIOWorkCombined 0
22    PrmModelName     SLDataModelName(assay)
23    TrigSSSplitOutUpd 1

```

图 15.7-13 模型对应的 rtw 文件

接着,由目标语言编译器根据 TLC 文件把 rtw 文件转化成目标代码。目标语言编译器(Target Language Compiler)是一种解释性编程语言,可以将 rtw 文件按照编写好的 TLC 程序转化为目标代码。TLC 程序由一个或者多个 TLC 文件组成,具体如何根据 RTW 生成代码的细节,则在 TLC 文件中进行规定。

然后,RTW 为模型生成自定义联编文件 model.mk,作用在于指导联编程序对模型产生的主程序、库文件和用户自定义的模块进行编译和链接。RTW 提供了一些系统模板联编文件(System Template Makefile),即 system.tmf,用以生成 model.mk。系统模板联编文件允许用户指定编译器、编译选项,以及可执行文件生成时所附带的信息。Model.mk 的生成过程是:

- ① 复制 system.tmf 中的内容。
- ② 针对描述模型配置的标识符进行扩展。

基于用户的不同需求,可以选择不同的系统联编文件进行编译。用户也可以自己修改模板,以适应自定义开发系统。系统联编文件如图 15.7-14 所示。

联编文件选定之后需要选定编译器对模型进行编译。默认情况下 32 位 MATLAB 的编译器是 LCC。默认下的 LCC 不支持嵌入式软件中常用的位域操作。读者可以使用 mex-setup 指令搜索机器上已安装的其他编译器,选择使用。笔者加速仿真使用 VC6.0 的编译器,如图 15.7-15 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

----- Tokens expanded by make_rtw -----
#
# The following tokens, when wrapped with "|>" and "<|" are expanded by the
# build procedure.
#
# MODEL_NAME           - Name of the Simulink block diagram
# MODEL_MODULES        - Any additional generated source modules
# MAKEFILE_NAME        - Name of makefile created from template makefile <model>.mk
# MATLAB_ROOT          - Path to where MATLAB is installed.
# MATLAB_BIN           - Path to MATLAB executable.
# S_FUNCTIONS          - List of S-functions.
# S_FUNCTIONS_LIB      - List of S-functions libraries to link.
# SOLVER               - Solver source file name
# NUMST                - Number of sample times
# TID01EQ              - yes (1) or no (0): Are sampling rates of continuous task | |
#                           (tid=0) and 1st discrete task equal.
# NCSTATES             - Number of continuous states
# BUILDDARGS            - Options passed in at the command line.
# MULTITASKING          - yes (1) or no (0): Is solver mode multitasking
# EXT_MODE              - yes (1) or no (0): Build for external mode
# EXTMODE_TRANSPORT     - Index of transport mechanism (e.g. tcpip, serial) for extmode
# EXTMODE_STATIC         - yes (1) or no (0): Use static instead of dynamic mem alloc.
# EXTMODE_STATIC_SIZE   - Size of static memory allocation buffer.

```

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 15.7-14 tmf 文件中被扩展的标识符

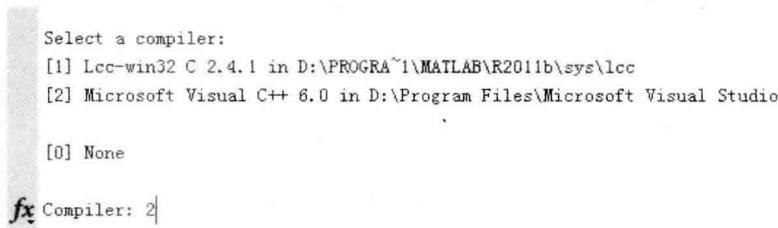


图 15.7-15 选择 VC6.0 编译器进行编译

在选定 VC 编译器的情况下,如果用户希望在编译时生成一个 exe 文件,应该选择名为 xxxx\_vc.tmf 的模板连编文件,包括 4 种:ert\_vc.tmf、grt\_malloc\_vc.tmf、grt\_vc.tmf 和 rsim\_vc.tmf。

如果用户希望在 VC 工程下以 C 语言形式对模型进行加速仿真,则应使用的联编文件包括:ert\_msvc.tmf、grt\_malloc\_msvc.tmf 和 grt\_msvc.tmf。

此处选择 grt\_msvc.tmf,使模型产生通用实时 C 代码,供快速仿真使用。用于生成代码的模型采用连续乙醇发酵的动态仿真模型,名字为 assay.mdl。模型如图 15.7-16 所示。

## 2. 模型参数配置与代码生成

为了使模型能够生成通用实时 C 代码,需要进行一些配置。打开 Real-Time WorkShop 选项卡配置基本的仿真参数如表 15.7-1 所列。

当用户希望仿真能达到实时效果时,可通过模型生成代码,并在嵌入式系统或实时计算系统中运行生成的代码。选择固定步长解算器仿真模型,其原因在于实时计算系统以固定的采样速率运行,若采用变步长将与现实生产中采样频率固定的事实有悖,并且在原本资源就有限的嵌入式目标芯片中实现变步长算法也是不合理的。

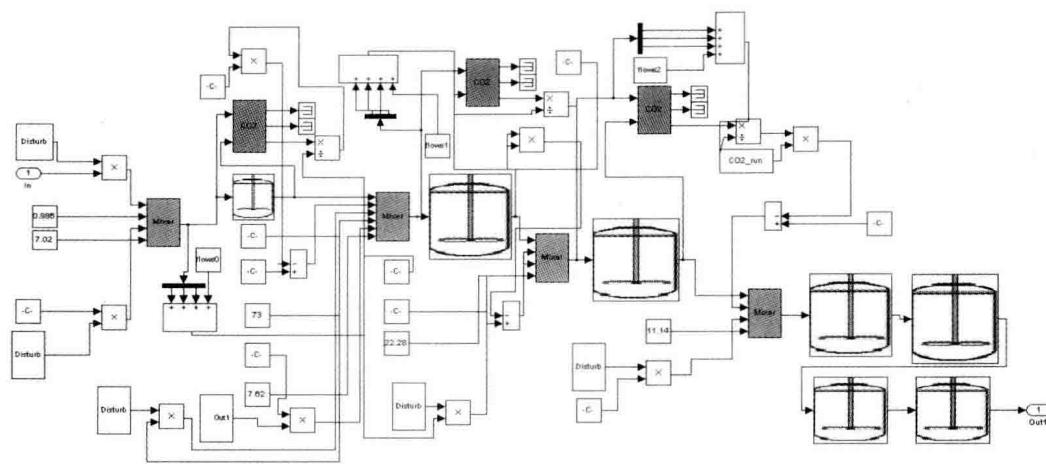


图 15.7-16 乙醇连续发酵仿真模型

表 15.7-1 模型的配置列表

| 选项卡                       | 参 数                             | 值            |
|---------------------------|---------------------------------|--------------|
| [Solver]                  | [Type]                          | Fixed-step   |
|                           | [Solver]                        | Discrete     |
| [Hardware Implementation] | [Device vendor]                 | Generic      |
|                           | [Device type]                   | Unspecified  |
| [Real-Time Workshop]      | [System target file]            | grt.tlc      |
|                           | [language]                      | C            |
|                           | [Create code generation report] | on           |
|                           | [Generate makefile]             | grt_msvc.tmf |
|                           | [Generate code only]            | off          |
|                           | [Include comments]              | on           |
|                           | [Retain .rtw file]              | on           |

配置模型 Configuration Parameter 参数后，在 mex-setup 中设置 VC6.0 的编译器，在 Command Window 中输入 rtwbuild(gcs) 启动模型编译以及代码生成；当显示 # ## Successful completion of build procedure for model: assay 时说明编译成功结束，生成的代码报告自动弹出，如图 15.7-17 所示。

生成的文件及函数包括：

- ① assay.c：自动生成的乙醇发酵算法代码。
- ② grt\_main.c：用于仿真的 main() 函数。
- ③ assay.h：输出 Simulink 的数据符号、模型数据结构体、模型入口函数。
- ④ assay\_private.h：输入 Simulink 的数据符号，RTW 专属的细节。
- ⑤ assay\_types.h：模型结构体 rtModel\_model 和参数结构体的声明。
- ⑥ rtwtypes.h:simstruc\_types.h 和数据类型定义，宏定义及枚举类型定义等。
- ⑦ 本例虽然定义了很多用于初始化的常数，生成代码时都存储在文件 assay\_data.c 中。

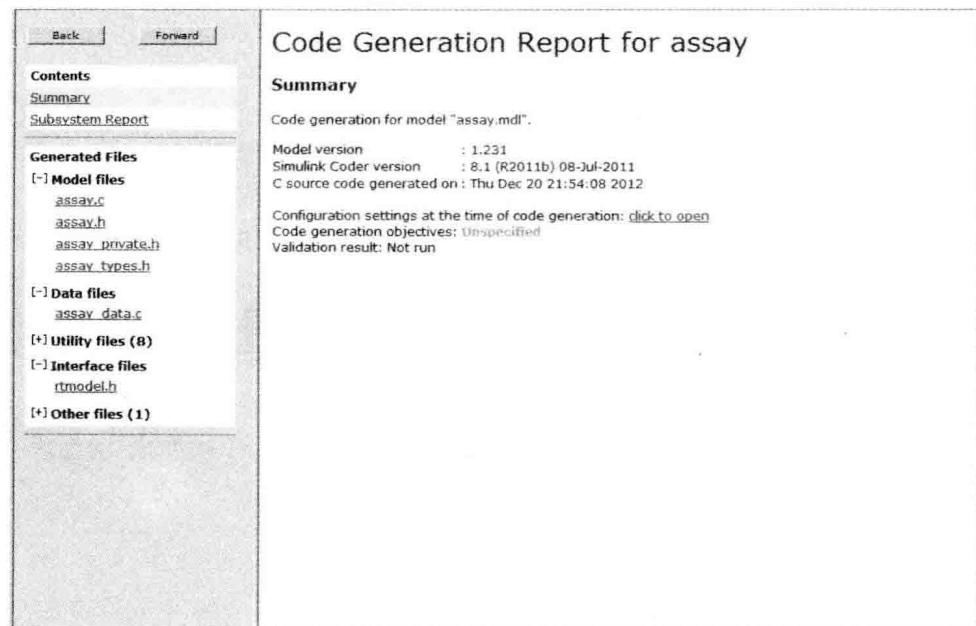


图 15.7-17 Real-Time Workshop HTML Report

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

⑧ assay\_initialize(), 包含于 assay.c 文件中, 用于初始化模型。

⑨ rt\_OneStep(), 包含于 grt\_main.c 文件中, 是计算的核心, 每一个步长被调用一次, 其内部调用 assay\_step() 函数用以完成模型在每个步长内的计算过程。

⑩ assay\_update(), 包含于 assay.c 文件中, 由 rt\_OneStep() 调用。在该函数中描述了具体模型的输入输出的关系, 并更新相应结构体成员的值。

在生成的 C 代码中, RTW 使用的变量名来自于模块名、属性名和信号名。用于保存系统信息的变量被分类存储到结构体中。在未用 Simulink. Parameter/Signal 对象定义数据结构的情况下进行代码生成时, Simulink 会默认将输入/输出定义为结构体变量。有多个输入/输出的情况下, 每个输入/输出就是结构体变量的一个元素。RTW 默认使用表 15.7-2 所列的表示方法。

表 15.7-2 模型中不同类型对象生成代码的结构体标志

| 结构体名       | 表示方法          | 结构体名    | 表示方法    |
|------------|---------------|---------|---------|
| 实时模型结构体    | RtModel_model | 外部输入结构体 | Model_U |
| 模块 I/O 结构体 | Model_B       | 外部输出结构体 | Model_Y |
| 参数结构体      | Model_P       |         |         |

### 3. VC 环境下的快速仿真

RTW 编译生成代码时, 在工作目录下产生了一个名为 assay\_grt\_rtw 的文件夹, 其中 assay.mak 文件用 VC6.0 环境打开, 提示时确定即可转换为 VC 的工程文件, 并包含上述的 C 文件。生成的工程文件列表如图 15.7-18 所示。

如此建立的工程可以直接通过编译链接运行, 运行的过程就是模型的仿真过程。为了显示出编译为 C 代码后编译速度加快, 可以在代码中添加代码, 显示模型运行的时间, 并与 MATLAB 环境下的时间进行对比。两个环境下的执行时间如图 15.7-19 所示, 这是经过多次运算实验的平均仿真速度。

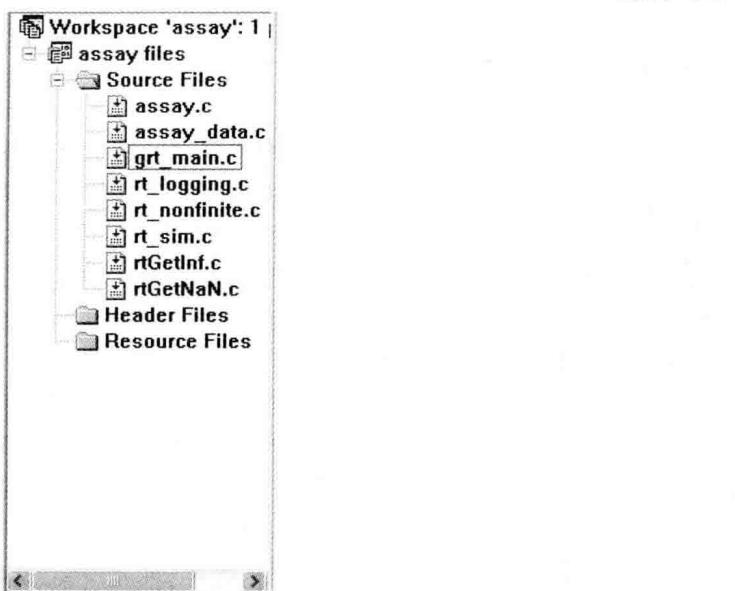


图 15.7-18 VC 环境下模型生成的工程文件列表

```
** Starting the model **
Simulation Done!
Average Time Used is:202ms
continue command...
```

(a) MATLAB

```
** starting the model **
Simulation Done!
Average Time Used is: 15ms
Press any key to continue...
```

(b) VC6.0

图 15.7-19 仿真时间对比图

可见,经过 RTW 编译后的代码运行仿真速度相比模型仿真速度提高了十余倍。特别是 MATLAB 首次执行模型仿真时,需要从硬盘读取.mat 文件,载入大量的初始化数据,需要耗时高达 7 s,而通过 RTW 编译后,初始化过程被极大地提速。

#### 4. 仿真参数的更改与结果输出

仿真的目的就是依据可靠的模型进行不同工况下的模拟,以优化生成流程或控制参数。如果要修改连续发酵仿真程序的仿真时间,可以在 grt\_main.c 文件中的 assay\_initialize 中找到 rtmSetTFinal 函数,其第二个参数就是程序定义的仿真时间,修改其值即可;又如改变流入各个发酵罐中原料流量的大小,可以在 assay\_data.c 文件中的 Parameters\_assay assay\_P 结构体里修改,各个数据后面已经自动生成了注释,可以方便地理解每个数据对应模型的物理量;仿真结束后为了查看结果数据,例如产品乙醇的浓度,可以添加代码到主程序的末尾处,如 MdlTerminate() 后面,添加 C 代码:

```
printf("Ethanol Concentration: %f\n", assay_Y.Out1[2]);
```

运行后可得如图 15.7-20 所示结果。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```
** starting the model **
Ethanol Concentration:72.565636
Press any key to continue...
```

图 15.7-20 VC6.0 运行仿真结果

## 15.8 总结

Simulink 模块化的图形展示及操作界面能使人比较容易地了解建模对象系统,不像代码那么抽象,难以掌握全貌;而且其提供了动态仿真的强大功能,为动态仿真研究提供了很大的研究发展空间。本章以乙醇发酵流程为例,先搭建发酵罐、混合器等子模块系统,再逐一连接起来构成整个流程系统,最后逐渐添加各种影响因素,并进行动态仿真,得到了理想的仿真效果。通过 GUI 的结合,将相对复杂的模型隐藏于后台,使用户通过简单对话框操作就可以实现仿真;最终结合代码自动生成技术将模型转换为 C 代码仿真,加速了仿真时间。

基于模型的开发技术不仅在嵌入式硬件开发中起到降低开发风险、提高效率、缩减成本和开发周期的作用,而且在加速模型仿真上也能起到显著的效果;不仅能够完全将模型的算法完整生成代码,根据不同的联编模板文件生成支持不同编译环境的工程文件,而且直接打开工程文件无需做任何修改即可运行。经过实际工程项目的实践,证明了 RTW 在快速仿真方面的良好效果,特别是针对大型耗时的仿真模型,将节约更多的时间,提高生成效率,增加收益。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

# 高 级 篇



# 第 16 章

## Simulink 基于模型设计的工业应用概述

**引言:**Simulink 是 MATLAB 中重要的组件之一,它提供一个动态系统建模、仿真和综合分析的集成环境。目前,汽车电子控制软件的开发以及各芯片制造商开发的针对半导体 MCU 芯片的目标支持工具箱(TSP)等都是以 Simulink 为核心工具开发的。那么世界范围内还有哪些公司也在使用 Simulink 进行工业应用与开发工作?本章结合 Simulink 的应用场合展开论述。

### 16.1 Simulink 用途概述

Simulink 是 MATLAB 中的一种集可视化仿真建模、代码自动生成及综合分析为一体的开发及设计工具,是一种基于 MATLAB 的框图设计环境,被广泛应用于线性系统、非线性系统、控制系统及数字信号处理、图像与视频处理、嵌入式设计及辅助,以及各个科研领域的建模和仿真中。Simulink 可以用连续采样时间、离散采样时间或两种混合的采样时间进行建模,既支持单一速率系统也支持多速率系统,也就是系统中的不同子模块具有不同的采样速率。为了创建动态系统模型,Simulink 提供了一个建立模型方块图的图形用户接口(GUI),这个创建过程只需单击和拖动鼠标操作就能完成,提供了一种更快捷、直接明了的方式,模型建立正确的情况下,用户可以迅速看到系统的仿真结果。Simulink 提供了丰富的模块库供开发者使用,用户可通过 M、C 等语言自定义新的模块并创建模块库添加到 Simulink 库中去,方便以后使用。Simulink Library Browser 图像如图 16.1-1 所示。

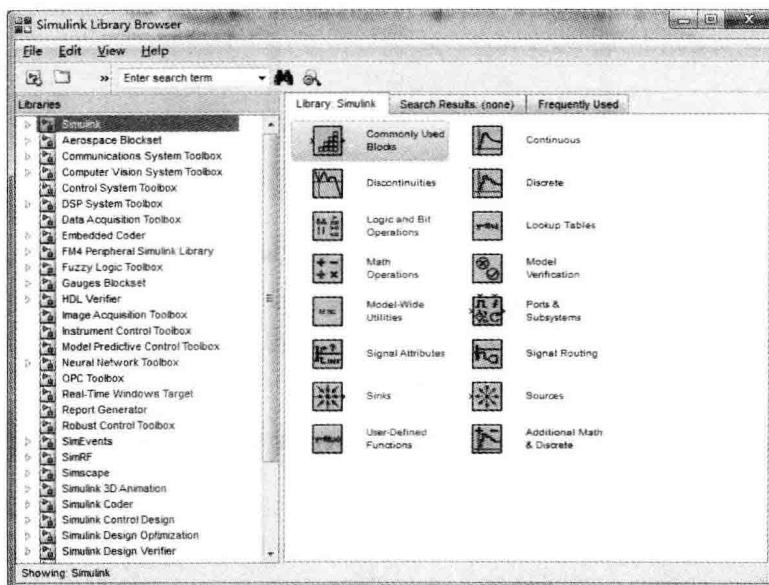


图 16.1-1 Simulink 模块库

在基于模型设计(Model Based Design, MBD)广泛应用于汽车电子嵌入式开发的今天,该技术也逐步推广到各种嵌入式控制方面。与传统的嵌入式开发相比,MBD以模型为核心,从算法设计到代码生成甚至工程编译一气呵成,以实现开发流程高度自动化。驱动层代码通过赋予寄存器不同的值实现不同工作方式,其结构相对固定,也遵循结构化设计,将其制作成Simulink环境下可以使用的模型,用户无须手写代码,只要在模型上设置寄存器工作方式即可自动生成可直接使用的嵌入式C代码,不仅加速了开发过程,而且除去了手写代码引入的不具合,并且能够提供接口连接算法模型,让嵌入式工程师省下手写驱动代码的时间,更专注于算法设计,真正做到高效开发。下面一起来探讨一下Simulink在世界范围内的应用情况。

## 16.2 Simulink 的工业应用

基于模型设计是Simulink为核心开发工具在世界范围内应用最广泛的开发流程,特别是汽车控制系统研发方面。宝马、奥迪、戴姆勒-奔驰甚至特斯拉电动车也都采用Simulink工具进行开发,将最先进的研究成果、市场需求、控制系统的算法、嵌入式软硬件等因素全部集成到一个图形化的可执行设计规约中来,不仅可进行早期仿真验证和过程自动化,还可以支持在各个层级不断验证(SIL、PIL、HIL)。应用于嵌入式开发的基于模型设计流程的结构图如图16.2-1所示。

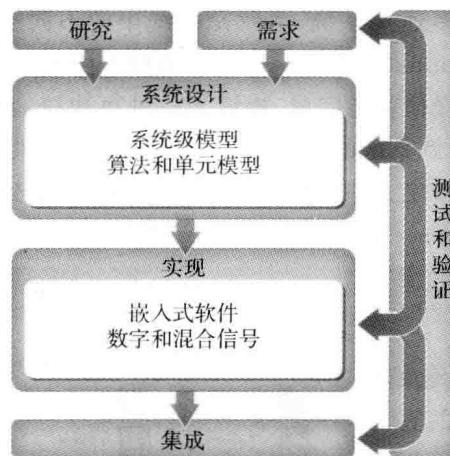


图 16.2-1 常用的基于模型设计流程

2014年5月24日,笔者应邀参加在南京东南大学举行的MATLAB中文论坛研讨会,并且与MathWorks的多位专家进行了交流,从他们那里得到了很多Simulink在世界范围内应用的消息。

### 16.2.1 Airbus 使用基于模型的设计为A380开发出燃油管理系统

Airbus(空中客车公司,<http://baike.baidu.com/view/302841.htm>)是业界领先的民机制造商。公司以客户为中心的理念、商业知识、技术领先地位和制造效率使其跻身行业前沿。2005年,空中客车公司的营业额超过了200亿欧元,目前已牢固地掌握了全球一半的民用飞机订单。

Airbus 使用 MATLAB、Simulink 进行基于模型的设计,对 A380 的燃油管理系统进行建模和控制逻辑仿真,并且因为所有部门使用同一平台做设计开发,加强了各个协作部门之间的沟通和交流效率,整体上加速控制器的开发周期。A380 如图 16.2-2 所示。

在开发过程中,通用算法通过建立标准模块库得到了很好的重用,系统集成时充分发挥了模型参考的优势,方便地将各个部门负责的子功能模块引用到系统模型里。在没有增加额外员工的情况下提前数月完成了燃油管理系统的设计。Airbus 的 Christopher Slack 提到“Simulink 模型使我们可以提前验证需求并向我们的供应商传达功能规范,以便按照 ARP 4754 工业标准补充书面需求。我们可以重复使用这些模型来创建桌面仿真器,试运行 HIL 测试平台,在我们的虚拟集成平台上运行以及向客户展示系统功能。”

### 16.2.2 马自达加快开发下一代应用创驰蓝天技术(SKYACTIV TECHNOLOGY)的发动机

马自达是大家都熟知的品牌,他们也在为了使自己的产品能够符合各项严苛的国际排放标准,不断创新,优化产品质量,提高效率。在这样的前提下,顺应时代要求,优化创驰蓝天发动机的开发过程中,也采用了 Simulink 工具盒基于模型设计的开发流程,使用 Simulink 和 Model-Based Calibration Toolbox(基于模型的标定工具箱)加速生成和开发最佳标定设置、ECU 可嵌入式模型,以及用于 HIL 仿真的发动机模型。

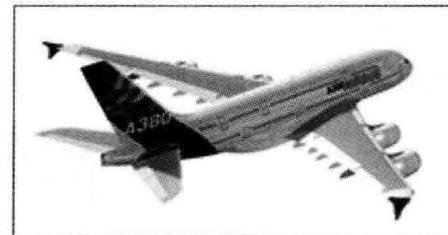
通过基于模型的标定工具箱可以极大地降低发动机标定工作的工作量,并可以有效地降低模型的复杂度,使得开发工作高效进行,并且模型有很高的计算精度。

马自达的 Shingo Harada 说到:“最初,我们的嵌入式最大汽缸压力模型有 38 个参数。通过使用 Model-Based Calibration Toolbox,将参数减少到 20 个,从而降低了 CPU 上的负载。同样,该工具箱还使得我们能够将排气温度模型的参数从大约 40 个减少到 20 个,而且保持同等级别的精确性。”马自达创驰蓝天柴油发动机如图 16.2-3 所示。

### 16.2.3 特斯拉电动跑车 Roadster

特斯拉汽车公司(Tesla Motors),一家生产和销售电动汽车及零件的公司,只制造纯电动车,成立于 2003 年,总部设在美国加州的硅谷地带。他们在 Simulink 的帮助下,以有限资金的投入开发了一款电动运动跑车,也就是现在的 Roadster。

特斯拉使用 Simulink 基于模型设计工具对电动跑车建模、仿真、评估和性能分析,并评估设计权衡,不仅得到了较好的系统仿真模型,并且省去了建造实际系统的步骤,通过仿真测试



空客 A380, 世界最大的商用飞机

图 16.2-2 A380 机型图

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

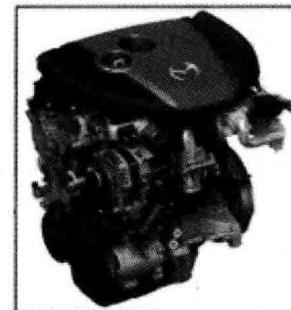


图 16.2-3 马自达创驰蓝天柴油发动机

了数百个动力总成的配置。

由于此款电动车完全通过 Simulink 设计开发,进行了多领域模型的创建并整合,其中电池建模技术得到了很好的推动和发展。特斯拉 Roadster 如图 16.2-4 所示。



图 16.2-4 特斯拉 Roadster 电动跑车

“We couldn’t have built this car without MATLAB/Simulink tools. It would have taken resources that our new automotive startup company simply did not have. We will continue to rely on MATLAB and Simulink to help us make informed design decisions for the next generation of Tesla vehicles.”(如果不使用 MATLAB/Simulink,我们不可能造出这辆车。若非如此,它所需要消耗的人力和时间资源必定超出我们这个刚刚起步的汽车公司的承受能力。我们将继续依赖 MATLAB/Simulink 来进行新一代特斯拉汽车的设计及决策。)Chris Gadda 与 Andrew Simpson 两位博士谈到。

#### 16.2.4 罗斯胡尔曼理工学院使用 Simulink 和 SimDriveline 设计混合动力汽车动力总成系统

罗斯胡尔曼理工学院是一所位于印地安纳州的私立大学,拥有 2100 名本科学生和 100 多名硕士研究生。这个规模是一个典型的文理学院规模,而学校的教学也具有文理学院的气息,例如较高的师生比例(1 : 13)让教授可以关注到每个学生,学校强调动手实践型的教育。同时学校非常关注学生的职业方向,该校的很多毕业生成为了美国的优秀工程师、公司创始人或者企业高管。

MATLAB/Simulink 也成为他们师生所钟爱的设计工具,他们建模和仿真汽车的控制系统及混合动力汽车的动力总成,如图 16.2-5 所示,并获得现实世界的工程经验。不仅让学生获得了很好的锻炼机会,增加了实际工程经验,做到了基于模型设计,基于项目学习,并且使开发动力总成系统的时间降低了 80%。



图 16.2-5 罗斯胡尔曼理工学院师生

## 16.2.5 三星(英国)利用 Simulink 开发出 4G 无线系统

除了汽车领域,Simulink 在通信领域也有很大的市场和发展。在国内,华为、中兴等通信巨头也在广泛采用 MATLAB/Simulink 进行产品研发。

由于与通信相关的开发几乎所有研究人员都使用 MATLAB 和 Simulink,因此很容易共享工作和维护应用。另外,Simulink 还协助三星(英国)与欧盟项目中的其他三星办公室及合作伙伴进行交流,在 Simulink 下直接无缝整合合作伙伴或集团内其他成员的成果。只要事先指定好标准化规范,便能够快速设计通信系统,同时提升协作和重复使用性。

## 16.3 总 结

Simulink 在工业界的使用可见一斑,欧美及日本正在以很快的速度普及和发展,国内尚在一个踌躇期,在盛开期到达之前如何将此技术尽快化为自身的技术能力,并积极投身到基于模型设计的工业应用中去是当前应尽力思考之事。

若您对此书内容有任何疑问,可以凭在线交流卡登录MATLAB中文论坛与作者交流。

# 第 17 章

## Simulink 代码生成技术详解

**引言:**Simulink 自带了种类繁多功能强大的模块库,连接模块的信号线负责将数据从源模块传递到中间模块,经过重重计算,最终流到接收器模块中。在基于模型设计的新型开发流程下,Simulink 建立的模型从早期验证、代码自动生成到后期 SIL、PIL、HIL 等提供了全流程的快速开发工具链和品质保障措施,不仅通过仿真可以进行早期设计的验证,还可以生成 C/C++、PLC、VHDL 等代码直接应用于 PC、MCU、DSP、FPGA 等平台。在嵌入式软件开发中发挥着重要的作用,本章以 Simulink 模型生成嵌入式 C 代码为主体详细分析代码生成的原理及应用技巧。

### 17.1 基于模型的设计

基于模型设计是一种流程,较之传统软件开发流程而言,使开发者能够更快捷、更高效地进行开发。适用范围包括汽车电子信号处理、控制系统、通信行业和半导体行业。图 17.1-1 所示系统的 V 字模型就是整个开发流程的整体描述,Simulink 模型贯穿需求、设计、编码与测试。

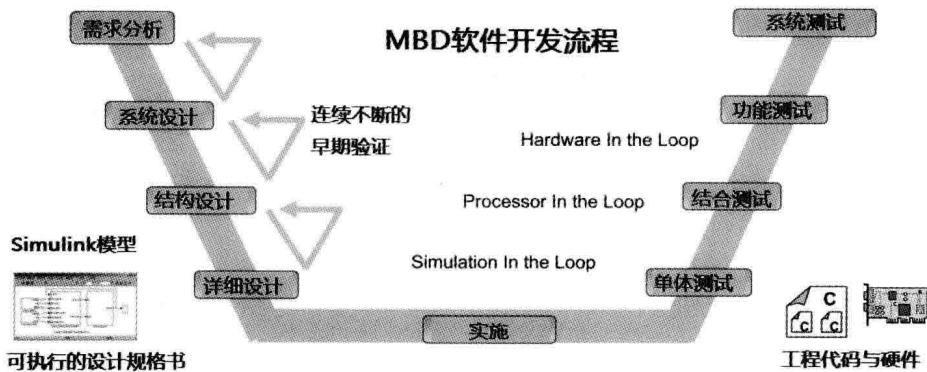


图 17.1-1 基于模型设计开发流程

模型本身就是一个可执行的规格书,开发者修改优化模型就是对设计的修缮,修缮之时立刻可以进行设计的验证,无须到编码实现之后通过测试再进行系统验证。这是由于仿真功能在早期设计时期就可以进行设计验证,并且在实施阶段,特别是对于嵌入式软件,代码由模型生成,模型与需求挂接,在仿真中验证设计,一经验证即可直接生成嵌入式代码,并且下游开发环境可以通过 MATLAB/Simulink 进行集成,并提供代码优化功能和连续不断的测试功能,产出的代码符合产品级要求。早期验证消除了在测试中发现 bug 并回归修正及测试的风险,自动代码生成消除了手写代码引入的 bug,在持续的测试下 MIL、SIL、PIL、HIL 将代码在 PC、MCU、非实时与实时环境下进行验证与测试,以保证工程的可靠性和实时性。MBD 开发流程可大大提高嵌入式开发效率。

使用基于模型设计流程开发软件有以下的优势：

- ① 在整个项目开发过程中使用同一的设计环境。
- ② 可以直接将需求与设计链接起来，易于对比变更点，降低设计遗漏的可能性。
- ③ 将测试集成到设计中以持续验证并纠正错误。
- ④ 通过多域仿真优化算法。
- ⑤ 自动产生嵌入式软件代码。
- ⑥ 开发标准模块库便于重用。
- ⑦ 自动生成文档。
- ⑧ 支持针对硬件目标的设计重用。

下面以一个滤波器为例，介绍基于模型的嵌入式软件设计过程。

### 17.1.1 需求文档

对于开发者来说，项目开始于需求，需编写需求文档，如图 17.1-2 所示为简化的需求文档。

Requirements for a fast filter.

1. Two input 

  - One for signal input and the other for filter coefficient.

2. One output 

  - The output for signals filtered by the filter.

3. Title 

  - There should be a title in the model.

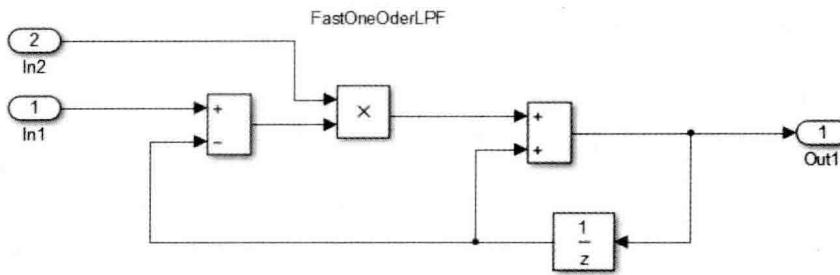
4. Algorithm 

  - The calculation equation should be displayed in the model.

图 17.1-2 滤波器的需求文档

### 17.1.2 根据需求进行设计

作为简单的 Demo，设计作为整体的一个阶段说明，此处不细分为系统设计、功能设计或详细设计。根据一阶快速滤波器的计算公式可以建立如图 17.1-3 所示模型。



$$\text{Output}[i] = (\text{Input}[i] - \text{Output}[i-1]) * \text{Lpfk} + \text{Output}[i-1]$$

图 17.1-3 根据需求设计出系统模型

### 17.1.3 需求与设计的挂接

通常需求文档使用微软 Word 或 IBM 的 Doors 等软件管理,本小节以 Word 为例说明需求与设计的挂接。MATLAB 为了能将设计规格书与模型关联起来,通过 rmi setup 注册 Active-X controls 后 Simulink Model 菜单栏增加了需求追踪功能,通过这个功能追加 Simulink 链接到需求设计规格书的 WORD 文档中,在每一项需求设计条目后都会出现 Simulink 的小图标,能够从需求文档链接到模型;可通过链接检查需求变更是否及时反映到所设计的模型中,以保持设计文档和模型的一致性,如图 17.1-4 所示可以从需求文档链接到模型中对应模块。

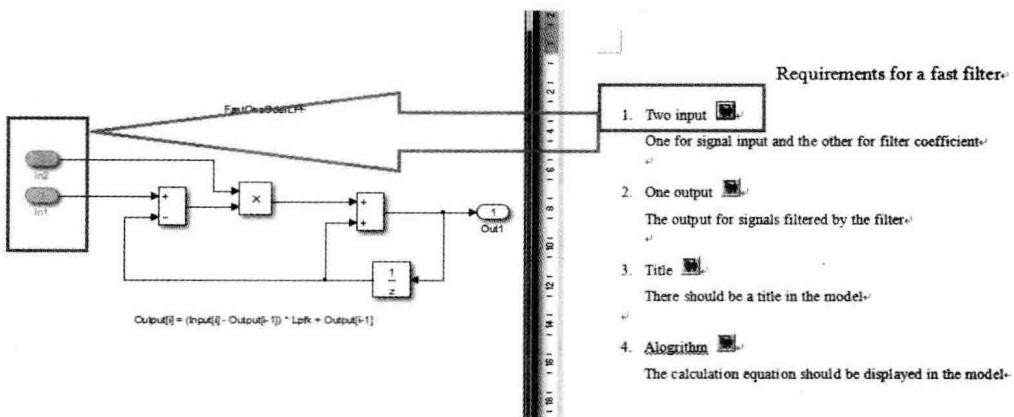


图 17.1-4 滤波器的设计规格书到模型的链接

### 17.1.4 模型的仿真

经过多次设计与需求的对照及修正后,模型设计得以确定,如图 17.1-5 所示。对这个确定的模型给定合适的输入输出进行仿真,以验证模型计算的正确性。此模型输入包括滤波系数和需要滤波的信号(带噪音的正弦波)。仿真时,选择合适的解算算法至关重要,不同的应用场合使用的解算方法不尽相同。以嵌入式软件开发为目的的模型,使用固定步长解算器。

仿真结果如图,下方坐标轴为待滤波信号——混杂了噪音的正弦波信号,上方坐标轴为滤波后的信号。带有噪音的正弦波滤波后得到了圆滑的正弦信号,如图 17.1-6 所示。

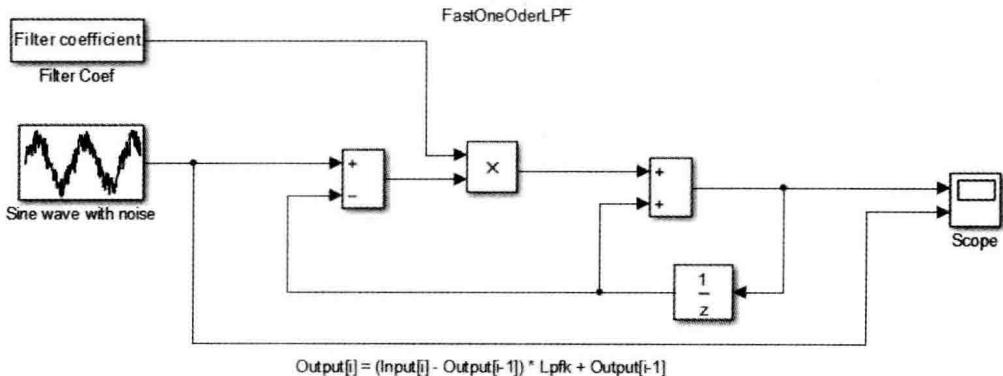


图 17.1-5 滤波器的仿真模型

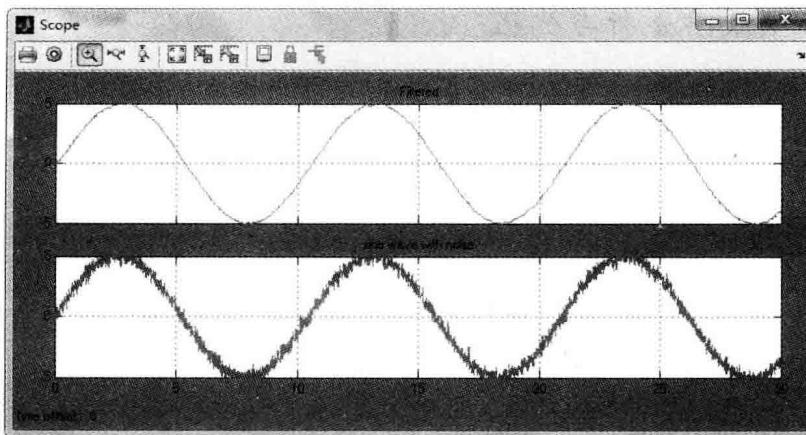


图 17.1-6 滤波器的仿真模型波形

### 17.1.5 模型的性能分析及修正

使用模型生成代码,特别是在产品级代码生成时,所用模型必须按照特定规则进行审查。在工业自动化生产、汽车电子开发过程中,随着模型复杂度的提高,人工检查模型属性、配置及对于 MAAB 标准的遵守度逐渐成为负担,使用 Simulink 工具栏中的 Model Advisor 工具可自动进行标准及模型参数配置和属性的检查,并产生检查报告。检查的内容:

- ① 是否会导致系统仿真出错。
- ② 是否会导致生成的代码无效(Simulink Coder& Embedded Coder)。
- ③ 生成的代码是否符合安全标准(Simulink Coder& Embedded Coder& Simulink V&V)。

自动化的模型检查高效方便,模型检查规则可以使用 Model Advisor 提供的检查方案,也可以进行检查规则的自定义。启动检查并完成之后会给出如图 17.1-7 所示检查报告,设计者可参考检测报告中的建议进行模型修正。

Model Advisor Report - filter\_demo.slx

Simulink version: 8.2      Model version: 1.19  
System: filter\_demo      Current run: 30-Apr-2014 16:15:45

| Run Summary                      |    | Pass                  |   | Fail                  |   | Warning               |  | Not Run               |   | Total |
|----------------------------------|----|-----------------------|---|-----------------------|---|-----------------------|--|-----------------------|---|-------|
| <input checked="" type="radio"/> | 12 | <input type="radio"/> | 0 | <input type="radio"/> | 1 | <input type="radio"/> |  | <input type="radio"/> | 1 | 14    |

---

**Check and update mask image display commands with unnecessary imread() function calls**

Identify masks using an image display commands with unnecessary calls to imread(). Since 2013a, a performance and memory optimization is available for mask images specified via image path instead of RGB triple matrix.

**Passed**  
No masked block found with unnecessary imread() calls in image display commands.

---

**Check and update model to use toolchain approach to build generated code**

**Warning**  
Model is not configured to use the toolchain approach.

**Recommended Action**  
Model cannot be automatically upgraded to use the toolchain approach.

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 17.1-7 模型的检查报告

### 17.1.6 模型效率分析与优化

使用 Profiler Report 了解模型每个环节的时间消耗和各种子方法调用次数。如图 17.1-8 所示报告中提示模型执行时间、调用函数清单和每个函数调用的详细效率分析信息。让设计者迅速知道运行过程中最耗时的位置,有针对性的寻找对策,提高模型执行效率。

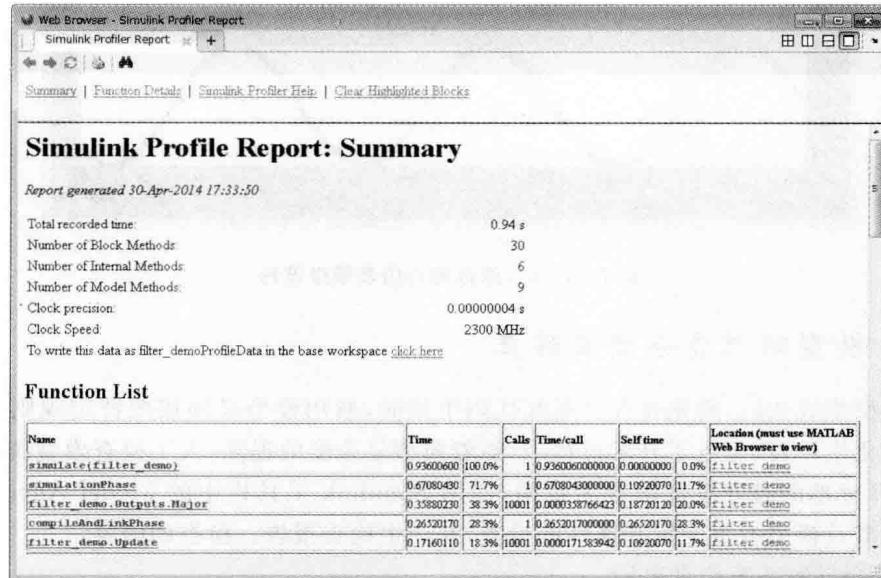


图 17.1-8 Simulink Profile Report

### 17.1.7 模型的代码生成

Simulink 的 Simulink Coder 工具箱提供了将模型转换为可优化的嵌入式 C 代码的功能。将模型的信号源和信号接收部分模块替换为输入/输出端口,系统文件使用 ert.tlc,负责统筹调用代码生成的整个过程,将模块转化为 C 代码,只不过这时的 C 代码并非专门面向某种嵌入式芯片的,是未经过任何优化的通用性嵌入式代码,可读性不强。滤波器模型及其在 ert.tlc 作用下生成的代码如图 17.1-9 所示。

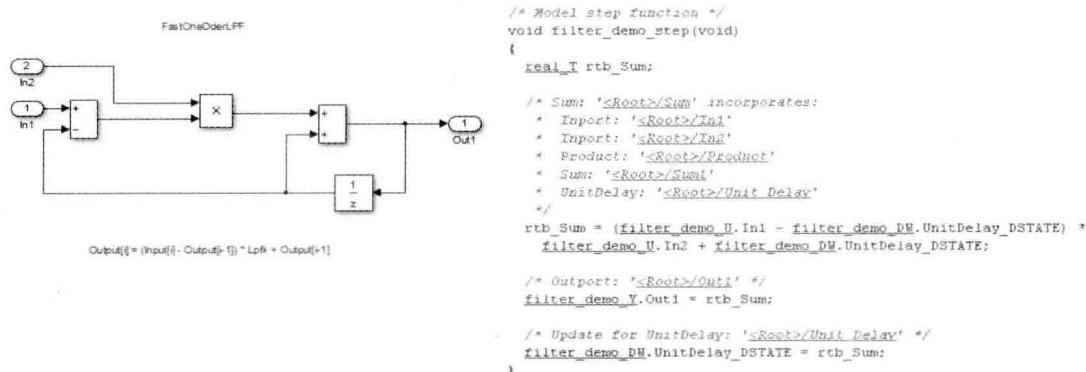


图 17.1-9 ert.tlc 控制下生成的 C 代码

### 17.1.8 模型生成代码的优化

Simulink 的 Embedded Coder 工具箱提供了两方面的优化方式:一是通过对信号线的存储类型进行设置改变代码生成时的变量生成方式;二是通过结构化的思想将算法模型构成一个原子子系统,并将其代码生成在指定文件的指定函数内,便于移植或重用。模型生成代码的优化配置及生成的代码如图 17.1-10 所示。

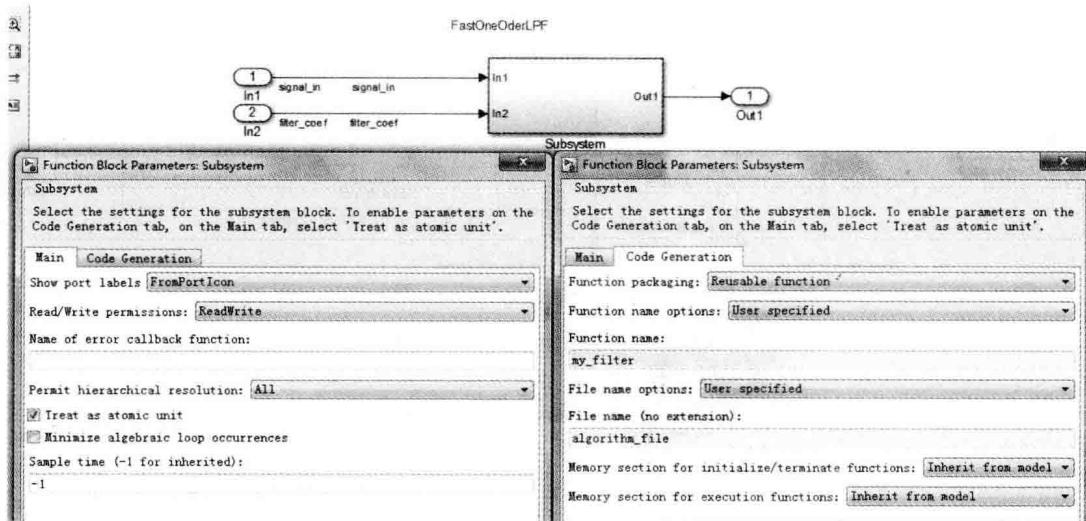


图 17.1-10 提高可读性并将算法代码单独存储

生成的代码中表示信号的变量不再是杂乱无章的结构体变量,而是以模型信号线命名的变量,优化后的代码生成如图 17.1-11 所示。

```

16 #include "algorithm_file.h"
17
18 /* Include model header file for global data */
19 #include "filter_demo_P23.h"
20 #include "filter_demo_P23_private.h"
21
22 /* Output and update for atomic system: '<Root>/Subsystem' */
23 void my_filter(real_T rtu_signal_in, real_T rtu_filter_coef)
24 {
25     /* UnitDelay: '<S1>/Unit Delay' */
26     signal_out_old = signal_out;
27
28     /* Sum: '<S1>/Sum' incorporates:
29      * Product: '<S1>/Product'
30      * Sum: '<S1>/Sum1'
31      */
32     signal_out = (rtu_signal_in - signal_out_old) * rtu_filter_coef +
33     signal_out_old;
34 }
35
36 /*
37  * File trailer for generated code.
38 */

```

图 17.1-11 优化后的滤波器算法代码以函数形式单独生成在一个文件中

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

### 17.1.9 代码的有效性验证

模型在 Simulink 环境下仿真的确得到了滤波效果,那么该模型生成的嵌入式 C 代码是否拥有同样的功能和效果呢?对于初次接触基于模型设计方法的读者必定心存疑问。对于算法模型或控制逻辑模型等应用层模型生成代码的有效性,需要实际硬件环境下验证才行。Processor In the Loop Simulation(PILS, 处理器在环仿真)便提供了验证算法有效性的方法。PIL 仿真环境的构成示意图如图 17.1-12 所示。

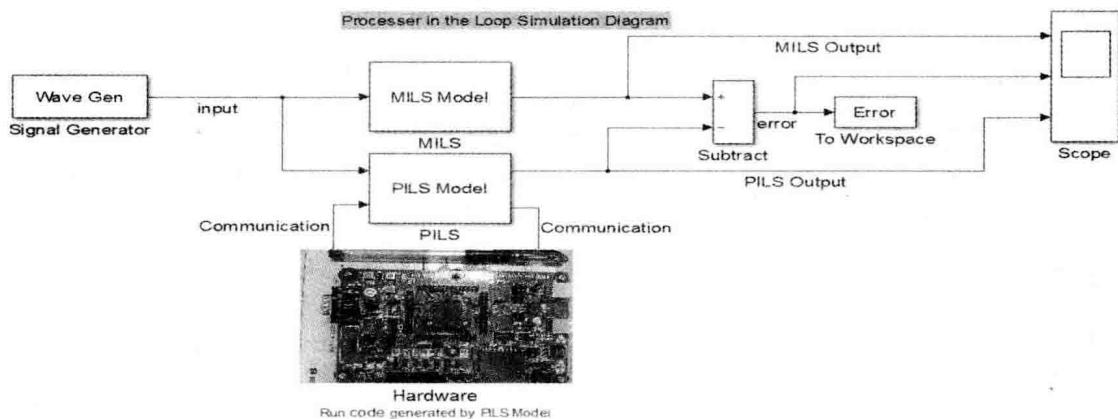


图 17.1-12 PILS 环境构成

PILS 需要 Simulink 模型与目标硬件协同工作。滤波器算法模型以不同的模式存在于 Simulink 模型中(Normal 仿真模式及 PIL 仿真模式),普通仿真模式自不必言,PIL 仿真模式则是将 PIL 模型生成嵌入式 C 代码之后编译为目标文件下载到目标硬件中去,硬件与 PC 通过串口通信方式(或 USB 通信或以太网通信)连接,建立 Simulink 与硬件开发板上 MCU 之间的通信通路。Simulink 的信号源模块提供信号输入,经过串口传递给目标硬件,经过 MCU 计算完之后再通过串口传回 Simulink 模型,接着信号源模块再输入下一个采样点,如此反复。跟 Normal 模式下 MIL 模型的仿真的结果进行比较,比较二者在相同输入相同参数,同步仿真下计算的输出是否相同。PILS 是为了验证算法设计在硬件上执行的正确性,以及算法实机执行时间的测试手段。

### 17.1.10 其他验证方法

在 PILS 阶段之前,MBD 开发方式还提供一种非实时的模型与其生成的代码的等效性验证,Software In the Loop Simulation(SILS, 软件在环仿真)。在 PILS 之后,MBD 提供 Hardware In the Loop(HILS, 硬件在环仿真)的复杂嵌入式系统的测试方法。HILS 提供一个平台,能够将各种复杂的被控对象以数学的表示方法作为动态系统追加到测试环境中,它能够很好地仿真被控对象,这些被仿真出来的被控对象通过传感器等设备作为接口,将控制系统 MCU 与被控对象的 HIL 平台连接起来,进行实时仿真。

在很多情况下,最有效的测试方法就是将控制系统跟实际的被控对象连接测试。但是,考虑到成本和安全性等因素(大型电网或整车运行的直接实际运行存在不安全因素),能够提供安全的测试环境和高效的测试质量的 HIL 成为嵌入式控制系统测试过程中最为人们所青睐

的方法。通过 HIL 进行大型复杂嵌入式系统设计具有的优点包括：节省成本，避免实际设备损坏；验证系统的实时效果；加快开发进程；降低安全隐患，保证安全性。

## 17.2 Simulink 代码生成流程及技巧

Simulink 能够为嵌入式开发提供基于模型设计支持的关键在于框图式建模的设计方法、仿真功能及代码自动生成技术。在前面的各个章节通过 Simulink 的构成元素、运行机制及仿真方法的理解和掌握，使读者朋友们能够进入本章内容的学习——Simulink 代码自动生成技术。

Simulink 代码自动生成支持多种语言，C/C++、PLC 和 VHDL 等，分别由不同的工具箱支持其代码生成功能。本书主要围绕嵌入式 C 代码的生成技术进行讲解。在 Simulink 环境中，嵌入式代码生成是由 Embedded Coder、Simulink Coder 为主导，MATLAB Coder 辅助其进行生成代码优化。主要应用目标是嵌入式 MCU，片上快速原型开发板，以及应用于民生电子、工业领域的 MCU 微处理器等。Embedded Coder 负责生成可重用、结构紧凑且执行快速的实时 C 代码，同时能使 MATLAB Coder 和 Simulink Coder 这两个工具箱的选项和高级优化功能来控制生成代码的文件结构、函数形式和数据存储等。笔者并不希望读者刻意地去记住哪些代码生成功能由 Simulink Coder 负责，哪些由 Embedded Coder 负责，对于开发过程来说这不是最重要的，所以此后直接记为 Simulink 提供的代码生成功能，淡化二者功能上的区别，意在让读者尽快掌握 Simulink 的核心使用方法。在真正开始代码生成技术讲解之前，了解一下利用 Simulink 的这些代码生成优化功能能够带来的优势包括：

- ① 提高生成代码的执行效率；
- ② 方便地将既有的手写 C 代码（自定义数据类型、函数及既有源文件等）整合到模型中或生成代码中共同构建工程；
- ③ 快捷地校正控制系统的参数以直接应用于工业领域；
- ④ 针对特定的嵌入式目标硬件开发自定义的自动化工具链，工具链包括应用层与驱动层代码生成、代码自动 Merge、工程自动生成及目标文件的自动生成、自动下载和启动等。

从软件产业标准上讲，Embedded Coder 生成的代码支持 MISRA-C、AUTOSAR 和 ASAP2 软件标准；从工业标准上讲，Embedded Coder 生成的代码支持 ISO2626、IEC 61508 及 DO-178 标准，这些标准主要面向汽车电子、航空航天研发及工业自动化领域等。基于模型设计的成功案例，相信读者在第 16 章“Simulink 基于模型设计的工业应用概述”领略过了。

另外，Embedded Coder 能够提供代码追踪报告、代码接口的文档描述及软件自动验证技术等，本书以 Simulink 环境为依托，主要涉及的范围有：① 针对目标硬件的代码生成及优化；② 存储类型、数据类型及数据对象和数据词典；③ 单速率及多速率建模及代码生成；④ 包括 MIL/PIL 在内的代码验证，包括需求与模型和代码与模型的双向追踪功能；⑤ 整合目标嵌入式硬件及其使用的单个或多个 IDE 集成编译环境到 Simulink 自定义工具链中，进行基于目标硬件支持包 TSP(Target Support Package)的控制应用开发。

阅读此章的读者最好已经阅读 Simulink 的建模仿真和 S 函数等的基础知识，如果再加上一点 MCU 嵌入式基本知识及 C 语言基础，那么本章内容学习起来会有事半功倍的效果。本书旨在教会读者如何使用 Simulink 软件进行建模、仿真及代码生成的方法，理解采样时间、编译顺序、代码组织方式，读者应在熟练掌握此软件的基础上可以自主进行嵌入式应用领域的研究和开发。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

### 17.2.1 代码生成时的模型配置方法

在模型仿真的章节中已经说明过 Configuration Parameters 的配制方法,与之相比,嵌入式代码生成用的模型配置要复杂得多。先以图 17.2-1 所示简单的模型为例,说明生成代码的配置方法。

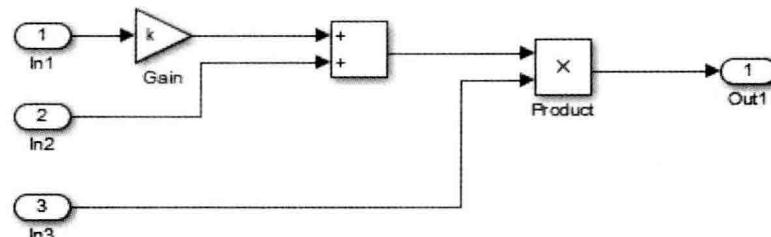


图 17.2-1 代码生成用示例模型

Configuration Parameter 中集中管理着模型的代码生成方法、格式等约束条件。为了生成嵌入式代码,至少需要配置 3 部分:模型的解算器 solver,模型的系统目标文件(如 ert.tlc 或其他自定义的嵌入式系统目标文件),硬件实现规定(Hardware Implementation)。

按下 Ctrl+E 打开模型的 Configuration Parameter 对话框,solver 页面如图 17.2-2 所示,解算器类型必须选择固定点解算器。固定点 solver 中提供了多种算法,此模型由于没有连续状态,可以选择 discrete 方法。步长默认 auto,在简单的通用嵌入式代码生成过程中此参数没有实际作用,可以采用默认或设置 0.01 s。而在针对目标芯片定制的代码生成过程中,硬件驱动工具箱往往会将步长 step size 作为其外设或内核中定时器的中断周期,使得生成的算法代码在硬件芯片中以同样的时间间隔执行。并且由于解算器步长为整个模型提供了一个基础采样频率,故被称为基采样率(base-rate)。

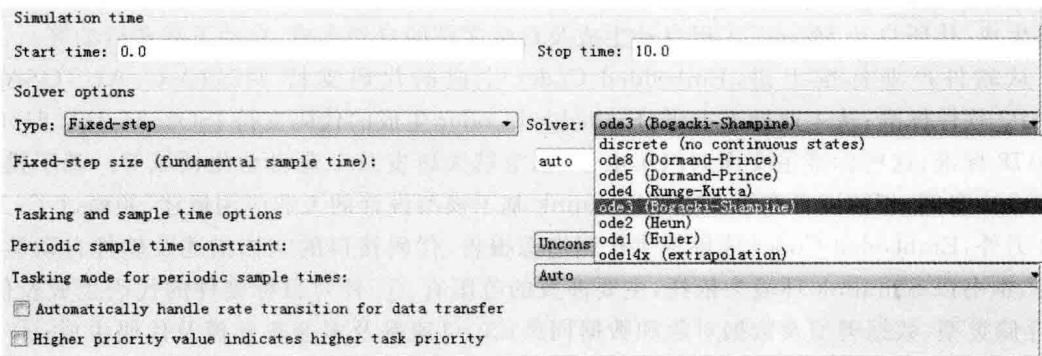


图 17.2-2 代码生成解算器配置

基采样率是当模型中存在多个不同采样频率的子系统时,最快的采样频率也不能高于 solver 步长的倒数,而第二快的采样频率以及更慢的采样频率,其采样周期都应该是 solver 步长(也就是基采样频率的倒数)的整数倍。采样频率与步长是倒数关系,为了方便,此后均以采样周期与 solver 步长的关系来说明。如果设置为非整数倍,在 solver 选为固定步长类型时,会报出采样时间错误,错误信息如:

Invalid setting for fixed-step size(xxxx) in model 'xxxxx'. All sample times in your model must be an integer multiple of the fixed-step size.

当模型中使用参数变量,如 Gain 模块的增益值,在生成代码时,如果希望使用该参数的值直接展开到代码中,就需要设置参数内联选项,如图 17.2-3 所示框中选项。

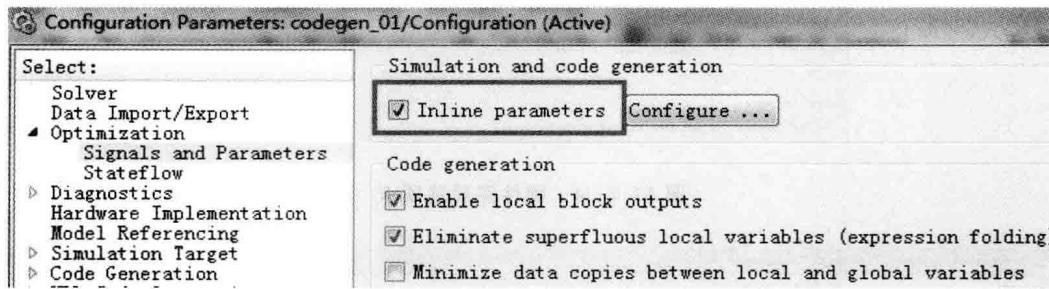


图 17.2-3 参数内联选项

Inline parameters 选项决定是否将参数内联到代码中去。勾选此选项后,代码生成时模型的参数将以常数方式直接生成到代码逻辑中,不再以一个参数变量的形式生成。当模型中的参数需要作为实时可调的参数生成到代码中时,不勾选 Inline parameter,参数将作为变量生成;如果不希望实时调整参数,可以选择节省存储空间的方式,勾选 Inline parameter,将参数以数值常数的形式生成到代码中。举例说明当模型中参数写为常数时,Gain 模块中增益值为 5,选项 Inline Parameter 勾选与否的区别如表 17.2-1 所列。

表 17.2-1 Inline parameters 勾选与否时的常数变量声明及使用

| Inline parameters | 生成代码实例  |
|-------------------|---|
| on                | codegen_03_Y.Out1 = 5.0 * codegen_03_U.In1;   |
| off               | P_codegen_03_T codegen_03_P = {<br>5.0<br>};<br><br>struct P_codegen_03_T_ {<br>real_T Gain_Gain;<br>};<br><br>typedef struct P_codegen_03_T_ P_codegen_03_T;<br><br>codegen_03_Y.Out1 = codegen_03_P.Gain_Gain * codegen_03_U.In1; |

Hardware Implementation 选项是规定目标硬件规格的选项。在这个选项卡中可以配置芯片的厂商和类型,设置芯片的字长、字节顺序等。学习使用通用嵌入式芯片为目标的代码生成流程及原理,选择 32 位嵌入式处理器作为芯片类型,如图 17.2-4 所示框中部分。

另外一个关键的设置选项是控制整个代码生成过程的系统目标文件 System Target File,ext.tlc 文件是 Embedded Coder 提供的能够生成专门用于嵌入式系统 C 代码的系统目标文件。在 Code Generation 页面中,单击图 17.2-5 所示右上角 Browse 按钮可以弹出对话框以选择系统目标文件。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

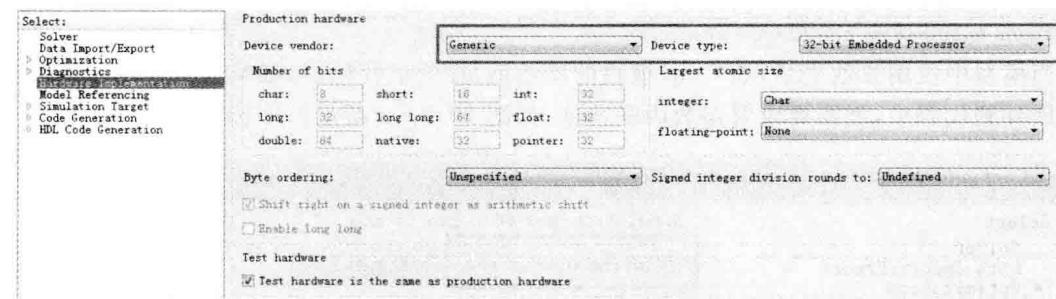


图 17.2-4 硬件实现选项卡

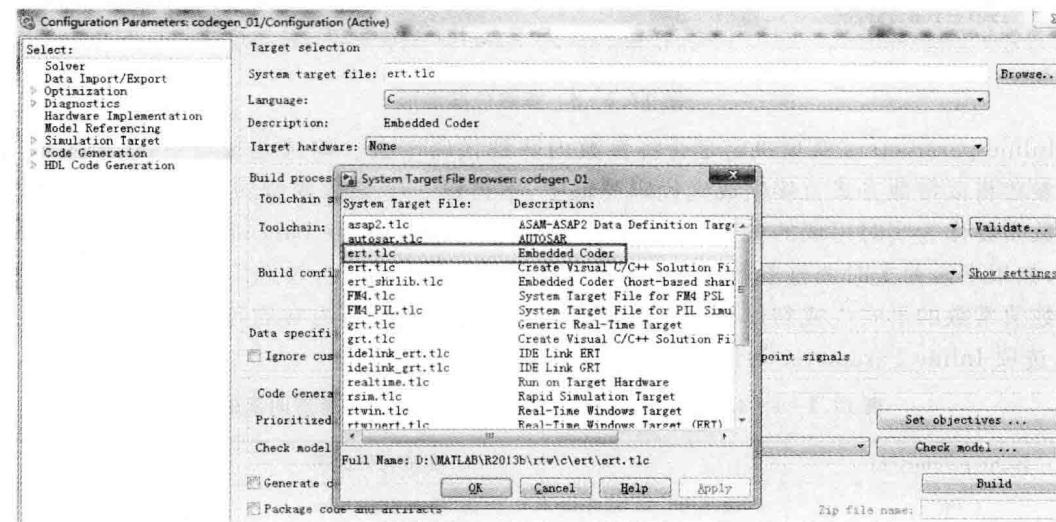


图 17.2-5 系统目标文件选择

在选择框中选择 ert.tlc 之后,Code Generation 标签页下面的子标签也会发生变化,提供更多的功能选项标签,如图 17.2-6 所示,方框内为新增子标签。

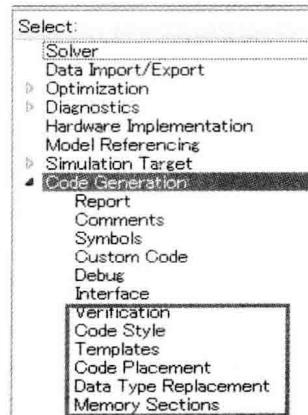


图 17.2-6 ert.tlc 模式下 Code Generation 的子标签

Report 子标签能够打开设置关于生成代码报告的页面,可以选择是否创建 HTML 格式

的代码生成报告，并通过勾选框选择是否在模型编译结束后自动打开。其对话框页面如图 17.2-7 所示。

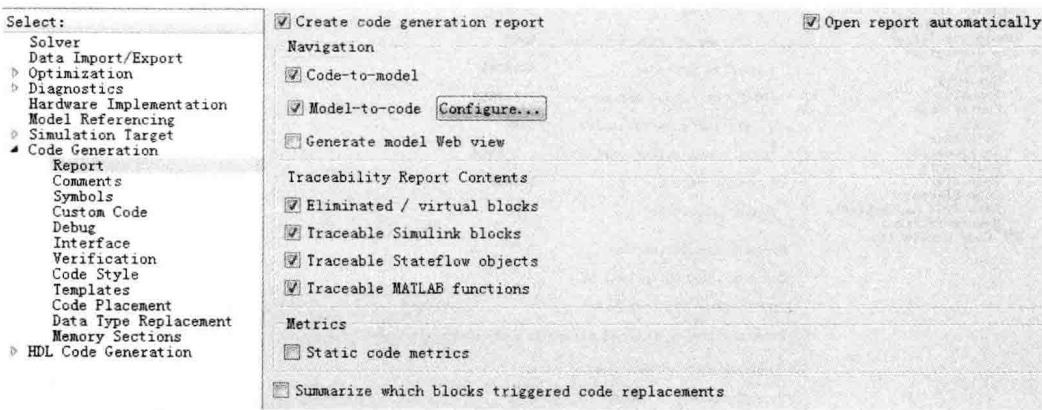


图 17.2-7 ert.tlc 模式下 Code Generation 的 Report 子标签

随后的 Navigation 参数组可以选择是否在代码报告中追加模型与代码的双向追踪。在 Traceability Report Contents 中可以设置追踪的内容。Metrics 组的 Static code metrics 选择框，勾选时将会在代码生成报告中包含静态代码的参数指标。

推荐读者勾选 Create Code Generation Report 及 Open report automatically 两个选项，模型生成代码完毕后会自动弹出报告列表，而不需要到文件夹中逐一将源文件手动查找并打开。

Comments 子标签中包含对生成代码中注释内容的配置，其对话框如图 17.2-8 所示。

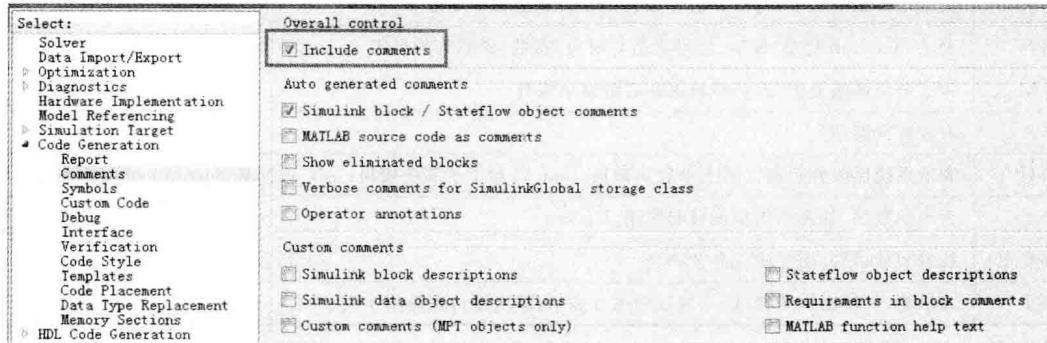


图 17.2-8 ert.tlc 模式下 Code Generation 的 Comment 子标签

Include comments 选项的勾选决定是否在生成代码中添加 Simulink 自带的注释。启动此选项后，Auto Generated comments 组及 Custom comments 组的选项便被使能，读者可以根据需要选择希望生成的注释内容。

推荐读者启动 Include comments 选项并勾选 Simulink block/Stateflow object comments 选项以生成注释，注释中带有可以从代码跳转到对应模型的超链接，方便读者追溯模块与代码的对应关系。

Symbol 子标签页面用于设置 ert.tlc 一族系统目标文件控制下的代码生成不变定义规则，如图 17.2-9 所示。这些符号包括数据变量和数据类型定义、常量宏、子系统方法、模块的输出变量、局部临时变量及命名的最长字符数等。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

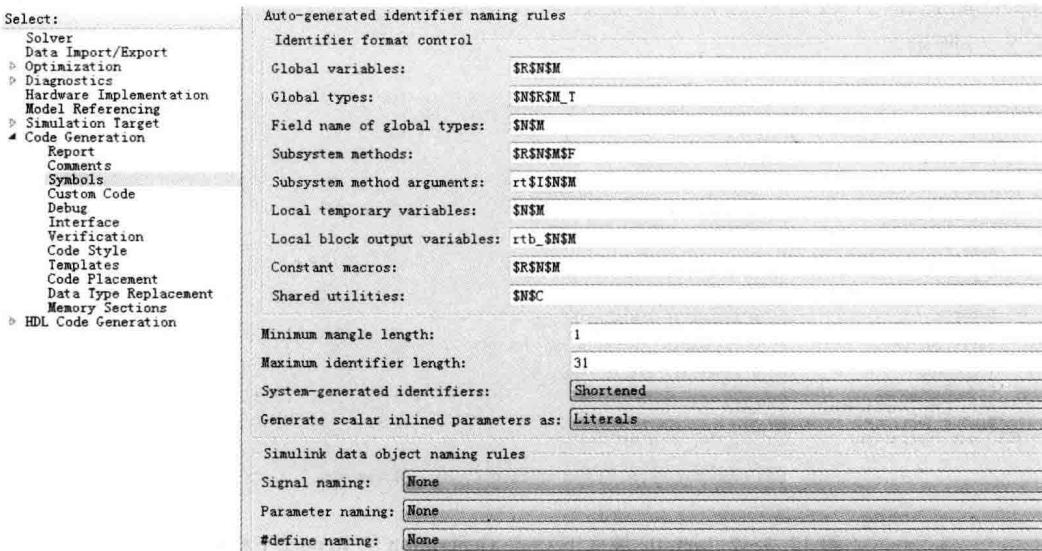


图 17.2-9 ert.tlc 模式下 Code Generation 的 Symbol 子标签

Identifier format control 参数数组里默认使用标示符 \$R\$N\$M\$T, 是 Embedded Coder 内部使用的标示符, 具体意义如表 17.2-2 所列。

表 17.2-2 控制代码生成的标示符

| 标示符  | 作用说明  |
|------|---|
| \$ R | 表示根模型的名字, 将 C 语言不支持的字符替换为下划线                                    |
| \$ N | 表示 Simulink 对象: 模块、信号或信号对象、参数、状态等的名字                            |
| \$ M | 为了避免命名冲突, 必要时追加在后缀以示区分  |
| \$ A | 表示数据类型  |
| \$ H | 表示系统层级标示符。对于根层次模块, root_;; 对于子系统模块:, sN_,, N 是 Simulink 分配的系统编号 |
| \$ F | 表示函数名, 如表示更新函数时使用_Update  |
| \$ C | 校验和标示符, 用于防止命名冲突  |
| \$ I | 表示输入/输出标示符, 输入端口使用 u 表示, 输出端口则使用 y 表示                           |

通过上表各种标示符的不同组合, 即可规定生成代码中各部分(变量、常量、函数名、结构体及对象)的名称的生成规则。

Simulink 提供的这些标示符生成的变量名虽然可读性不强, 但是不会引起代码编译错误。推荐用户使用默认设置, 不要为了提高生成代码可读性轻易进行修改, 以免引入错误造成不必要的麻烦。提高代码可读性的优化有更好更安全的方法, 在后面章节说明。

Custom Code 子标签页面主要用于添加用户自定义的或者编译模型时必需的源文件、头文件、文件夹或者库文件等, 其页面如图 17.2-10 所示。

Debug 子标签页面提供了关于编译过程和 TLC 过程的选项, 如图 17.2-11 所示。

Verbose build 的勾选可以将编译过程信息显示在 Command Window 中。Retain .rtw file 则能够保留编译模型生成的 rtw 文件。TLC process 组参数能够启动 TLC 文件的 profile 功能和调试功能, 使得开发者能够对 TLC 语言文件进行断点、单步调试等动作, 以及了解

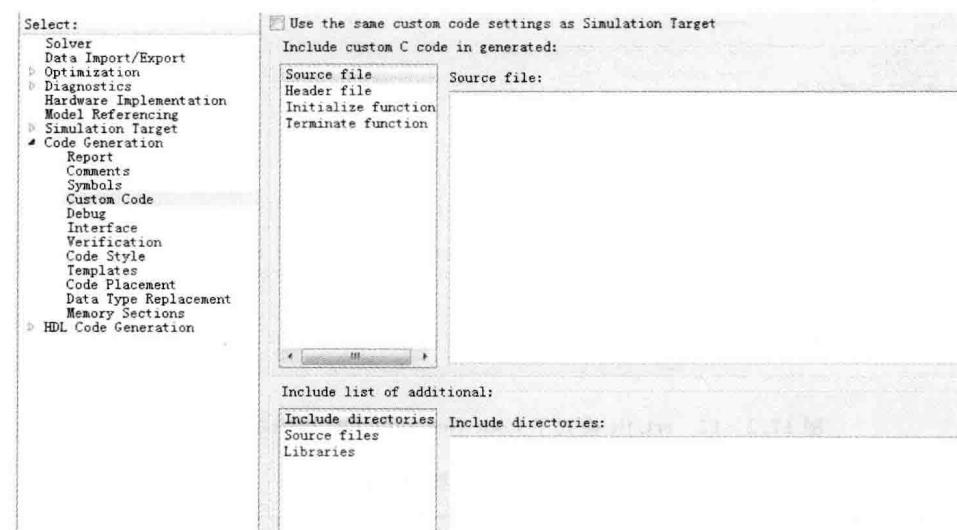


图 17.2-10 ert.tlc 模式下 Code Generation 的 Custom Code 子标签

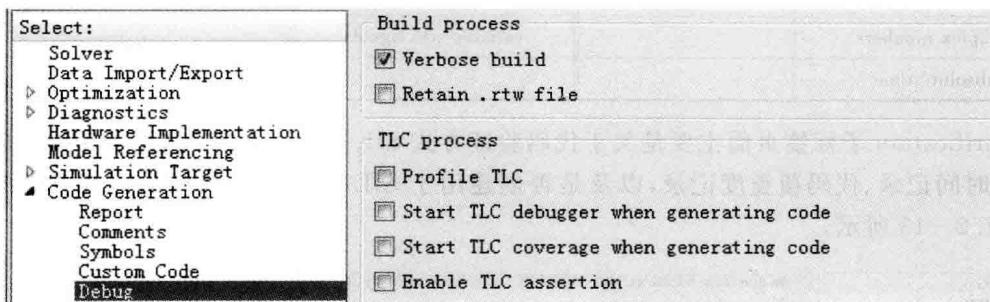


图 17.2-11 ert.tlc 模式下 Code Generation 的 Debug 子标签

TLC 语句的覆盖度等情况。

.rtw 文件是代码生成过程中从 Simulink 模型得到的中间文件, 它记录了模型相关的所有需要被 TLC 文件使用的信息。TLC 结合这些信息及 tlc 文件内容生成代码。TLC 的原理及编写方法请参考第 18 章“TLC 语言”章节。对于初学代码生成的读者朋友, 此处的参数设置推荐使用默认选项。

Interface 子标签页面中包含 3 组参数: Software Environment、Code Interface、Data Exchange, 其对话框如图 17.2-12 所示。

Software Environment 组的参数中提供 CPL(Code Placement Library)的选择, CPL 中定义一个表, 根据表格将 Simulink 模块与所对应目标语言的数学函数及操作函数库挂接, 以便从模型生成代码。Embedded Coder 提供默认的 CPL。Support 参数组由 7 个选择框构成, 每个选择框代表一种嵌入式编码器对代码生成的支持功能, 其中一些功能是需要 Simulink 提供的头文件来支持才能编译为目标文件的, 这些头文件一部分存储在路径为 MATLABroot\simulink\include 的文件夹中, 一部分是在模型生成代码过程中自动生成的(rt\_开头的头文件)。具体参考表 17.2-3。

Code Interface 与 Data Exchange 参数组用来配置生成代码的接口及数据记录的方式, 如无特殊要求建议使用默认配置。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

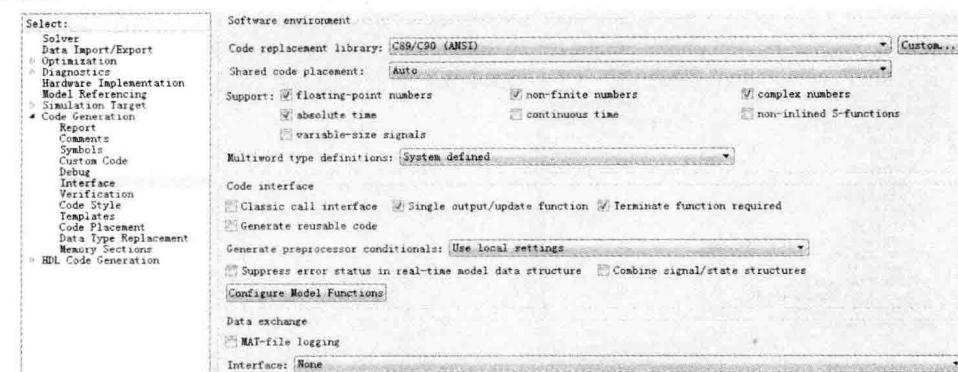


图 17.2-12 ert.tlc 模式下 Code Generation 的 Interface 子标签

表 17.2-3 Support 参数对应的头文件

| 选择框条目                  | 所需头文件          | 选择框条目                  | 所需头文件           |
|------------------------|----------------|------------------------|-----------------|
| floating-point numbers | rtw_solver.h   | continuous time        | rt_continuous.h |
| non-finite numbers     | rt_nonfinite.h | non-inlined S-Function | fixedpoint.h    |
| complex numbers        | —              | variable-size signals  | —               |
| absolute time          | —              |                        |                 |

Verification 子标签页面主要是关于代码验证方式 SIL 与 PIL 的配置,是否使能代码中函数执行时间记录、代码覆盖度记录,以及是否创建用于 SIL 和 PIL 的模块等。其页面对话框如图 17.2-13 所示。

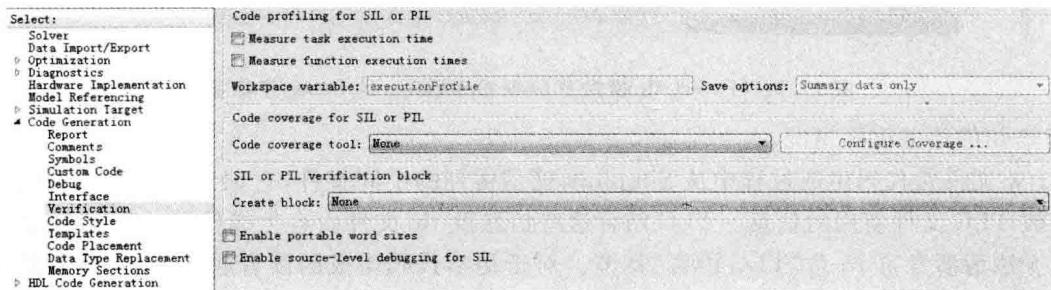


图 17.2-13 ert.tlc 模式下 Code Generation 的 Verification 子标签

Code Style 子标签页面提供了一些关于生成代码风格的选择框选项,如 if else 分支的完整性确保,if else 与 switch case 语句的选用,生成括号的频度,是否保留函数声明中 extern 关键字等。读者根据期待的代码风格选用。

Template 子标签页面内嵌入式编码器提供了一组默认的代码生成模版,如图 17.2-14 所示。ert\_code\_template.cgt 中使用 TLC 变量方式规定了文件生成的顺序及添加模型信息注释的位置。模型生成的源文件、头文件及全局数据存储和外部方法声明文件的生成可以使用统一模板;File customization template 则提供给用户自定义代码生成过程的文件输入,读者根据应用场景需要可以调用读取/写入目标硬件的 TLC 文件、自定义函数或注释,甚至根据模型的单速率/多速率不同分别调用不同的主函数模板等。在读者学习完第 18 章内容之后,可以详细阅读 ert.tlc 提供的 example\_file\_process.tlc。

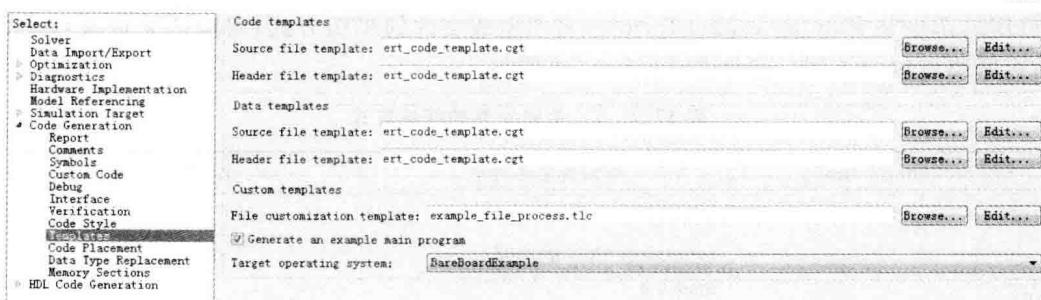


图 17.2-14 ert.tlc 模式下 Code Generation 的 Template 子标签

ert\_code\_template.cgt 中主要规定了代码段的顺序, section 包含了源文件从注释到变量再到函数体各种分段, 具体如图 17.2-15 所示, 顺序从上到下, 依次为: 文件说明的注释(File Banner)、头文件包含(Includes)、宏定义(Defines)、数据结构类型的定义(Types)和枚举类型的定义(Enum)、各种变量的定义(Definitions), 以及函数体的声明(Declarations)和函数体定义(Functions)。读者可以在相邻的段中插入自定义内容, 但是不要打乱既存段的相对顺序。

Generate an example main program 提供是否生成一个示例主函数的选项。这个示例主函数名为 ert\_main.c, 包含一个 main() 函数和一个调度器代码。主函数调用模型初始化函数 model\_initialize() 初始化模型需要的数据, 以及复位异常状态标志; 调度器代码仅提供一个模型每个采样时间点应该执行的函数模版 rt\_OneStep(), 此函数应该绑定到目标硬件的定时器中断上作为其中断服务函数(ISR), 执行周期应该与模型基频(即固定点解算器的步长)一致, 内部应该调用模型单步函数 model\_step()。另外, rt\_OneStep() 内部应该对其调用进行溢出检测。

Code Placement 子标签提供的选项将影响生成代码的文件组织方式和数据存储方式及头文件包含的分隔符选择等, 其页面如图 17.2-16 所示。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

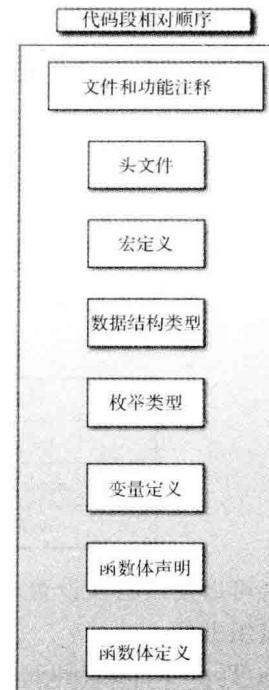


图 17.2-15 代码段的组成及顺序

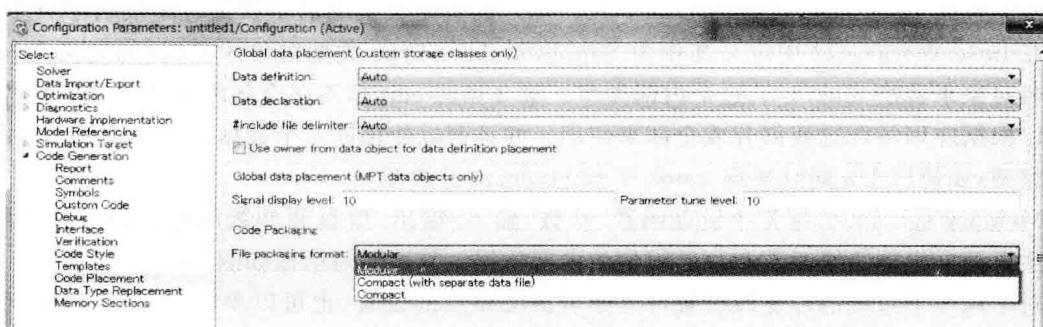


图 17.2-16 ert.tlc 模式下 Code Generation 的 Code Placement 子标签

常用的选项是 File packaging format, 表示生成文件的组织方式, 对应的生成文件个数不同, 内容紧凑程度也不同。具体如表 17.2-4 所列。

表 17.2-4 生成文件的组织方式

| File packaging format            | 生成的文件列表  | 省去的文件类表  |
|----------------------------------|--|--|
| Modular                          | model.c<br>subsystem files(optional)<br>model.h<br>model_types.h<br>model_private.h<br>model_data.c(conditional) | —  |
| Compact(with separate data file) | model.c<br>model.h<br>model_data.c(conditional)  | model_private.h<br>model_types.h                 |
| Compact                          | model.c<br>model.h   | model_data.c<br>model_private.h<br>model_types.h |

省去的只是文件个数, 其内容被合并到了其他文件中, 内容的转移如表 17.2-5 所列。

表 17.2-5 省去的文件内容转移列表

| 省去的文件           | 省去文件的内容转移目标       |
|-----------------|-------------------|
| model_data.c    | model.c           |
| model_private.h | model.c 和 model.h |
| model_types.h   | model.h           |

读者可以使用默认设置, 如果希望减少生成代码列表中文件的个数, 可以考虑使用 Compact 的组织方式。

Data Type Replacement 子标签默认情况下仅提供一个选择框选项, 勾选之后则弹出 3 列数据类型类表, 分别是 Simulink Name、Code Generation Name 和 Replacement。勾选 Replace data 选项之后界面如图 17.2-17 所示。

前两列按照数据类型的对应关系给出了每种数据类型在 Simulink 和嵌入式编码器生成代码中的类型名, 第 3 列则供用户设置, 填入自定义的类型名之后, 生成代码时将使用自定义的类型名替换 Code Generation Name。用户填入的自定义类型名不仅是一个别名字符串, 还必须在 Base Workspace 中定义其作为 Simulink AliasType 类型对象才可以, 如定义 U16 数据别名对象来替换 uint16\_T 这个内部类型。第 3 列的 edit 框不必全部填入自定义类型名, 读者可以根据应用场合选择部分或全部来使用。并且可以使用同一个数据类型名替代多个内建数据类型, 如使用 U8 同时替换 uint8\_T 和 boolean\_T 类型。

Memory section 子标签中设置函数、常数、输入/输出、数据和参数的存储段。存储段的设置主要面向模型等级函数和顶层模型的内部数据。其页面对话框如图 17.2-18 所示。

对于原子子系统的存储段设置可以继承顶层模型的设置, 也可以单独定义。当原子子系统 Code Generation 子页面上的 Function packaging 被选择为 Nonreusable function 或者 Reusable function 时, 可以单独对函数或内部数据进行存储段的设置, 如图 17.2-19 所示。

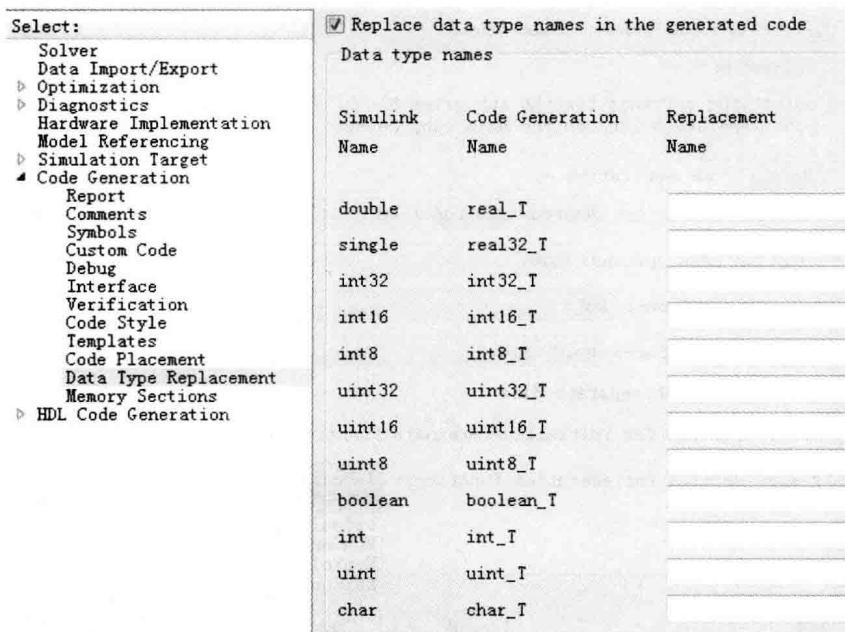


图 17.2-17 ert.tlc 模式下 Code Generation 的 Data Type Replacement 子标签

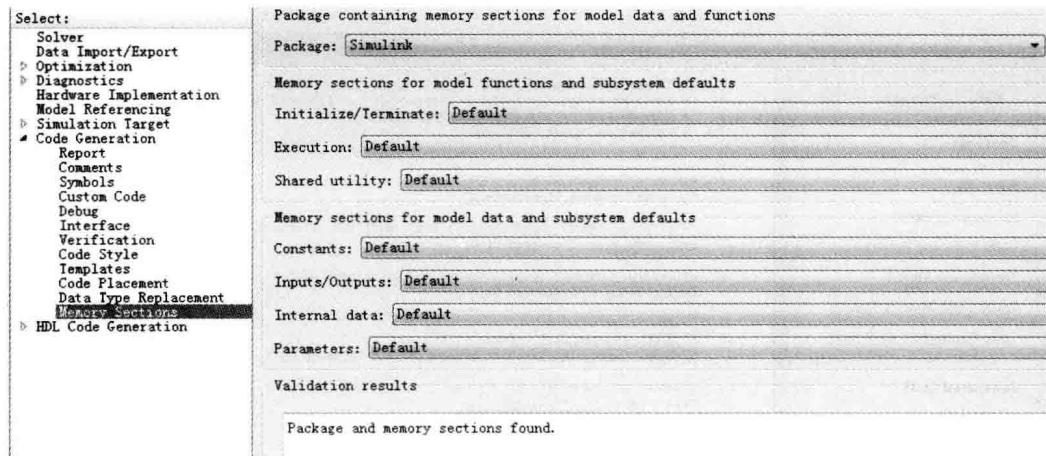


图 17.2-18 ert.tlc 模式下 Code Generation 的 Memory Section 子标签

如无须生成 pragma 等存储段指定命令则此子标签页面下使用默认设置即可。

Code Generation 标签页下提供的子标签页功能说明均已完成。

图 17.2-1 所示模型配置 system target file 为 ert.tlc 并勾选同一面上的 Generate Code Only(只生成代码不对生成的代码进行编译链接), 将 solver 选择为 fixed solver, 步长 auto, 并且选择 Hardware Implementation 为 Generic → 32-bit Embedded Processor, 按下 Ctrl + B 或者在 Command Window 中输入 rtwbuild(gcs), 则启动模型编译, 模型左下角从 ready 显示为 building, 片刻后弹出 Code Generation Report 界面如图 17.2-20 所示。

与模型名相同的.c 若模型配置无误, 则文件中包含 model\_step() 函数, 这里的代码表示模型所搭建的逻辑:

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

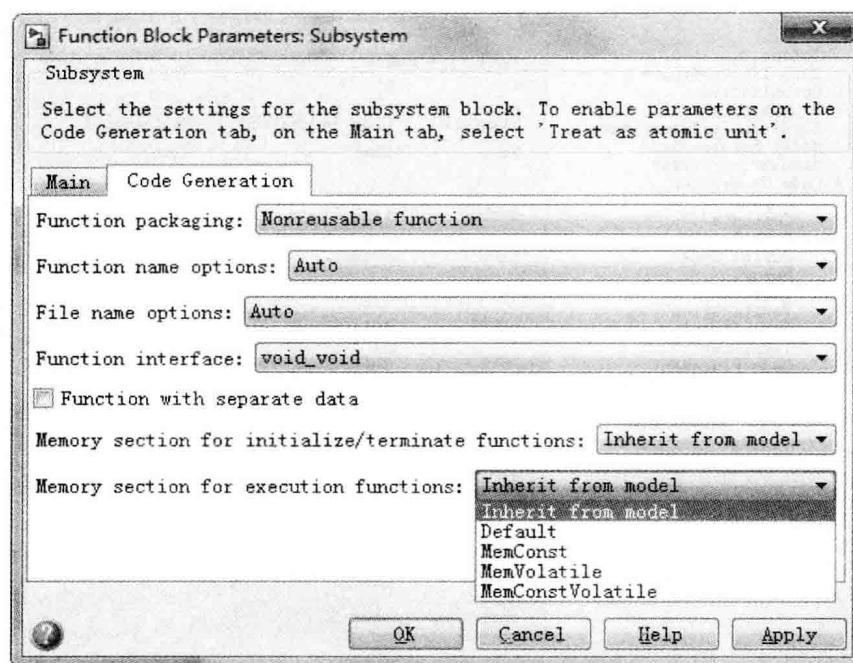


图 17.2-19 ert.tlc 模式下原子子系统的 Memory Section 设置菜单

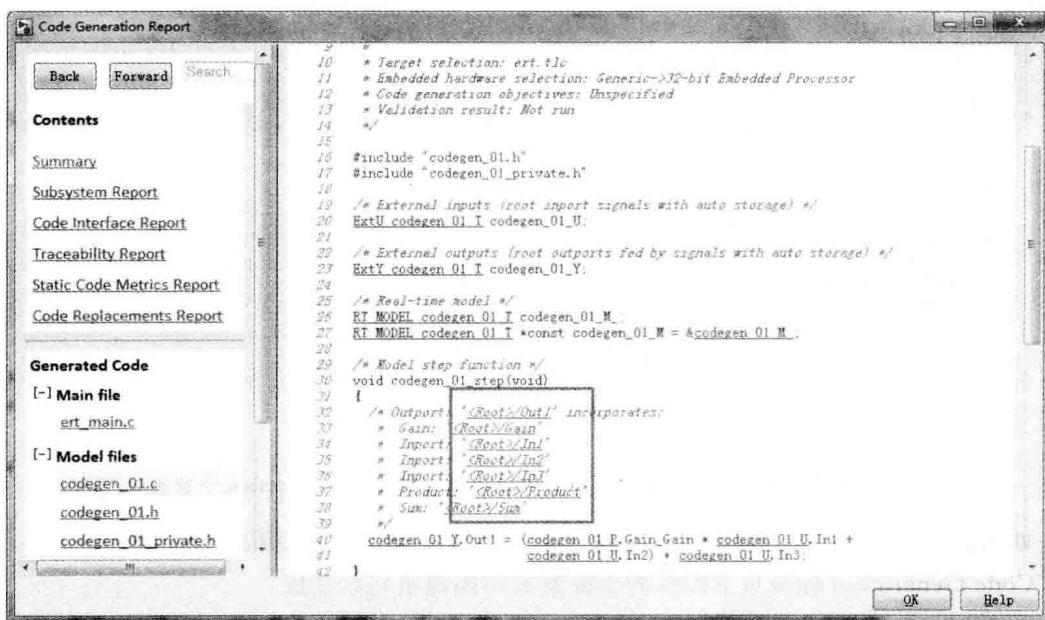


图 17.2-20 代码生成报告

```
codegen_01_Y.Out1 = (codegen_01_P.Gain_Gain * codegen_01_U.In1 +
codegen_01_U.In2) * codegen_01_U.In3;
```

未经优化的代码可读性较差,但是从四则运算关系中及结构体的成员名上可以看出每一个变量所代表的意义。

生成的代码还提供了 Code to Model 追踪功能,单击图 17.2-20 方框中的超链接,可以直

接跳转到模型中对应的模块,该模块或子系统将会以蓝色显示。这个追踪功能提示用户模型与代码的对应关系。

在开发自定义目标芯片的 TSP 时,往往在 Code Generation 下追加上述标签页以外的自定义标签页面,通过 GUI 控件提供目标芯片的选择下拉框、编译链接命令的输入框、目标芯片对应的集成继承编译环境(IDE)的选择和启动配置,以及工程文件设置重用等自定义功能。接下来介绍代码生成的流程。

## 17.2.2 代码生成的流程

模型生成代码的顺序之前也已经提及,首先通过 rtwbuild 命令将编译模型为 rtw 文件,Simulink Coder 中的目标语言编译器(Target Language Compiler)将 rtw 文件转换为一系列的源文件,这个过程中 TLC 所使用的文件包括 3 类:系统目标文件(ert.tlc, grt.tlc 等)、模块的目标文件(如与 S 函数配套的 TLC 文件)和支持代码生成的 TLC 函数库等文件。

模型的源代码全部生成之后,可以使用 Simulink 提供的模版自动生成 makefile 来编译链接得到目标文件,也可以将生成的源代码加入到目标芯片所使用的编译集成环境 IDE 的工程项目中去,使用 IDE 编译链接,最终通过仿真器下载到目标硬件中进行实机运行。代码生成的流程如图 17.2-21 所示。

图中展示了一个最简单的典型代码生成过程及后续实机执行所需要的步骤,从源代码之后编译链接下载的部分虽然不属于代码生成的范畴,却是嵌入式开发必经过程。本书将在此流程基础上进行嵌入式开发的技术解析,理解原理之后再去学习方法,二者兼有之后再通过实际工程项目参悟代码生成的道理。

整个代码生成过程还可以进行更加细致的划分,划分出更多的阶段供开发者自定义开发。在模型编译开始之后,整个过程分为 6 个不同阶段进行,如图 17.2-22 所示。

这 6 个阶段分别是 Entry、Before TLC、After TLC、Before Make、After Make 和 Exit。在保持顺序执行的前提下,每个阶段都可以由用户追加一些自定义行为,进行模型内部约束关系的检查,或者是链接外部第三方开发工具等。在 Entry 阶段,可以对自定义目标进行预配置、参数配置和检验等操作。Before TLC 是生成代码之前的阶段,可以将需要用到编译信息存储到一个结构体中管理,或者将编译需要使用的头文件、源文件和库文件等进行路径定位和添加。After TLC 是代码生成结束之后的阶段,可以将生成代码文件夹需要的文件拷贝到目标芯片的集成编译环境工程文件夹内以备使用;在 Before Make 阶段可以对代码进行分析、验证和模型或模块的用户设置检测等;After Make 则可以通过 MATLAB 控制集成编译环境自动生成工程文件,更新其中生成的代码列表,并自动编译链接及下载到目标硬件中去;Exit 阶段则可以将整个过程是否顺利执行的信息进行显示或临时变量清除等操作。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

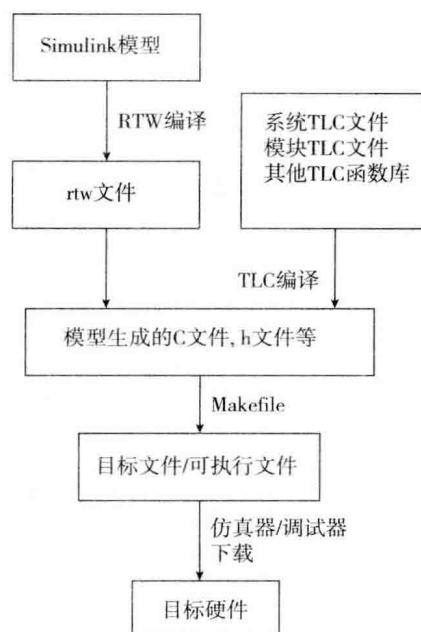


图 17.2-21 代码生成流程

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

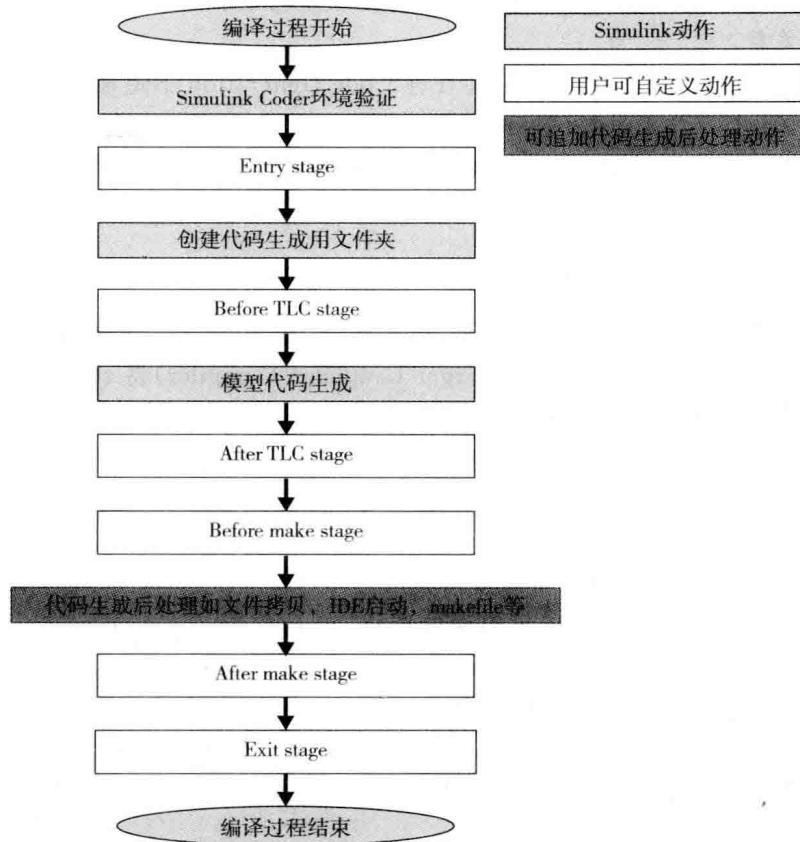


图 17.2-22 代码生成的 6 个阶段(英文为阶段)

Make 阶段是可选的(视是否在 MATLAB 里编译目标文件而定),可以在代码生成流程内进行 make 的动作,也可以不做定义,整个过程执行完毕之后再调用自定义 M 文件来实现或使用 IDE 编译。而 TLC 过程是不可获取不能裁剪的。存储模型所有信息的中间文件. rtw 及目标语言编译器 TLC 都是代码生成技术中十分重要的概念。

### 1. rtw 文件

rtw 文件作为模型编译器的输入文件和编译过程的中间产物,记录了模型创建信息和编译信息、名字与版本号、配置参数集、输入输出、参数等所有信息。可以通过图 17.2-11 的 Retain. rtw file 选项在编译过程中保留这个中间文件。图 17.2-1 所示模型编译出来的 rtw 文件存放在 model\_ert\_rtw 文件夹里,名为 model. rtw,使用文本编辑软件可以打开观察其内容,如图 17.2-23 所示。

rtw 文件内容特别长,不逐一说明每个条目的意义。rtw 文件的构成元素是 record,称为记录,其格式如下:

```
recordName { itemName itemValue }
```

rtw 文件的这一条条记录是按照层次划分的,最上层是 CompiledModel,具有全局的访问范围。向内逐层分别为 System、block。每条记录都由一个统领关键字和一对{}构成,{}内部称为统领关键字的域,访问域中成员与结构体访问方法一致,使用“.”操作符访问。记录之中可以再嵌套记录,正如 CompiledModel 中嵌套 subsystem, subsystem 中嵌套 block 一样。以

```

1 CompiledModel {
2     Name          "codegen_01"
3     OrigName      "codegen_01"
4     Version       "8.5 (R2013b) 08-Aug-2013"
5     SimulinkVersion "8.2"
6     ModelVersion   "1.6"
7     GeneratedOn    "Mon Dec 01 21:22:57 2014"
8     HasSimStructVars 0
9     HasCodeVariants 0
10    PreserveExternInFcnDecls 1
11    ExprFolding    1
12    TargetStyle     "StandAloneTarget"
13    NumDataStoresGlobalDSM 0
14    RightClickBuild 0
15    ModelReferenceTargetType "NONE"
16    PILSimTargetUnification 1
17    IsExportFcnDiagram no
18    AllowNoArgFcnInReusedFcn 0
19    StandaloneSSSupported 1
20    StandaloneSubsystemTesting 0
21    PadderActive     0
22    DWorkAndBlockIOCombined 0
23    PrmModelName     SLDataModelName(codegen_01)
24    IrigSSSplitOutUpd 1
25    UniqueFromFiles  []
26    UniqueToFiles    []
27    ErrorXMLMode    0
28    BlockDiagramType model
29    GlobalScope {
30        tIIDType        "int_1"

```

图 17.2-23 rtw 文件内容片断

下面是简化的 rtw 文件框架：

```

CompiledModel {
    Name "modelname"           -- 每一行记录都是“参数 值”成对的格式
    ...
    Subsystem {
        ...
        Block {
            Type "S-Function"
            ...
            Name "Hyo - Custom"
            ...
            Parameter {
                Name "P1"
                Value Matrix(1,2) [[1, 2];]
            }
            ...
            Block {           -- 同一子系统的下一个模块记录
                DataInputPort {
                    ...
                }
                DataOutputPort{
                    ...
                }
            }
        }
    }
}

```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```
ParamSettings {  
    ...  
}  
Parameter {  
    ...  
}  
}  
...  
Subsystem {  
    -- 下一个子系统的记录  
}
```

要访问此 rtw 文件中的模型名称可以使用:CompiledModel. Name;

当模型中存在多个子系统, 子系统中存在多个模块时, 可以通过下标逐个访问。如访问模型的首个子系统中首个模块的参数:CompiledModel. Subsystem[0]. Block[0]. Parameter . Value;

当层次结构过多时, 层层访问需要很长的语句, 为了方便, 可以使用%with 限定一下访问范围, 就可以直接访问被限定范围内的域成员了, 如访问首个模块的名字:

```
% with CompiledModel.System[0].Block[0]  
% assign blockName = Name  
% endwith
```

请注意, 以上这些访问 rtw 文件的语句都是 TLC 语句, 并非在 Command Window 中直接输入的 M 脚本。访问 RTW 的 TLC 语句可以写在 TLC 文件中或者当启动 TLC Debugger (见图 17.2-11) 之后, 编译模型的过程会停止在 TLC 执行阶段的开始, 这时在 Command Window 中使用 print 命令打印出 RTW 中域成员的值, 如图 17.2-24 所示。

```
ILC-DEBUG> print CompiledModel.System[0].Block[0].Parameter.Value  
[4.0E+0]
```

图 17.2-24 TLC Debugger 中打印 rtw 域成员的值

如果希望向 rtw 文件中插入信息应该如何做呢? RTW 是众多条记录的集合体记录, 可以嵌套。追加信息进来时也要遵循记录的格式, 使用 addtorecord 命令, 如添加模型作者信息到 RTW 的 CompiledModel 中:

```
% addtorecord CompiledModel Author{Name "Hyo"}
```

为了在模型编译过程中运行此句, 可以将该语句添加到图 17.2-14 的 File customization template 选项的 example\_file\_process.tlc 中。example\_file\_process.tlc 的内容为图 17.2-25 所示。

然后在启动 TLC Debugger 后, 使用 break 语句将断点设在新追加的语句之前的位置, 如图 17.2-26 所示。

设置断点之后, 输入 step 或 next 单步执行这条追加记录的语句。此句被执行之后才能够访问其值, 使用 print CompiledModel. Author. Name 可以将 Author. Name 的值取出来, 打印语句如图 17.2-27 所示。

如上所述, RTW 将模型的各种信息集于一身, 供 TLC 访问和调用, 为代码生成搭建了桥梁。

## 2. TLC 文件简介

TLC 文件即 TLC 编译器所编译的目标文件, 包括两个等级: 系统目标文件(如 ert.tlc)和模块目标文件(如 S 函数的内联 TLC 文件)。两者作用如下:

```

1 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2 %% $RCSfile: example_file_process.tlc,v $
3 %% $Revision: 1.1.6.5 $
4 %% $Date: 2010/09/13 16:20:21 $
5 %%
6 %% Abstract:
7 %%   Example Embedded Coder custom file processing template.
8 %%
9 %%   Note: This file can contain any valid TLC code, which Embedded Coder
10 %%  executes just prior to writing the generated source files to disk.
11 %%  Using this template "hook" file, you are able to augment the generated
12 %%  source code and create additional files.
13 %%
14 %% Copyright 1994-2010 The MathWorks, Inc.
15 %%
16 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
17 %selectfile NULL_FILE
18 %% Insert Hyo customized code
19 %addtorecord CompiledModel Author{Name "Hyo"}
20 %% Uncomment this TLC line to execute the example

```

图 17.2-25 追加自定义语句到模版文件

```

-dc switch
00012: %selectfile NULL_FILE
TLC-DEBUG> break example_file_process.tlc:19

```

图 17.2-26 在新追加的语句处打上断点

```

Breakpoint 1
00019: %addtorecord CompiledModel Author{Name "Hyo"}
TLC-DEBUG> next

00026: %if EXISTS("ERTCustomFileTest") && ERTCustomFileTest == TLC_TRUE
TLC-DEBUG> print CompiledModel.Author.Name

Hyo|

```

图 17.2-27 TLC 调试器中打印 RTW 新追加的成员值

- (1) 系统目标文件: 规定生成代码的全局结构, 以匹配所支持的目标芯片群及目标语言。
- (2) 模块目标文件: 规定 Simulink 模块生成代码时的代码实现, 一般与模块的 S 函数同名, 存储在同一文件夹, 并从 S 函数中获取参数信息以生成目标代码。

在 TLC 运行阶段, 首先运行的就是系统目标文件, 如 ert.tlc。系统目标文件是 TLC 运行的起点, 其他 TLC 文件(包含在系统文件内的 TLC 文件, 或主函数生成 TLC 和模块 TLC)会被其调用, 整个执行过程按照 TLC 命令行逐一执行。TLC 文件执行过程中, 会读取、增加或修改 rtw 文件中的记录信息(记录名与记录值)。被称为中间产物的 rtw 文件, 编译阶段是模型编译的产物, 代码生成阶段是 TLC 的输入, 与其他 TLC 文件一道受系统目标文件的统领。

Simulink Coder 和 Embedded Coder 提供一些系统目标文件供用户选择, 如图 17.2-28 所示。

系统目标文件对话框会将所有存在于 MATLAB 搜索路径上的系统级 TLC 文件显示出来, 如果有自定义的系统 TLC 也将显示在这里。嵌入式代码生成时的模型配制方法内容重点围绕 ert.tlc 展开(Embedded Coder 提供)。常用的目标文件有用于生成通用实时目标的 grt.

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

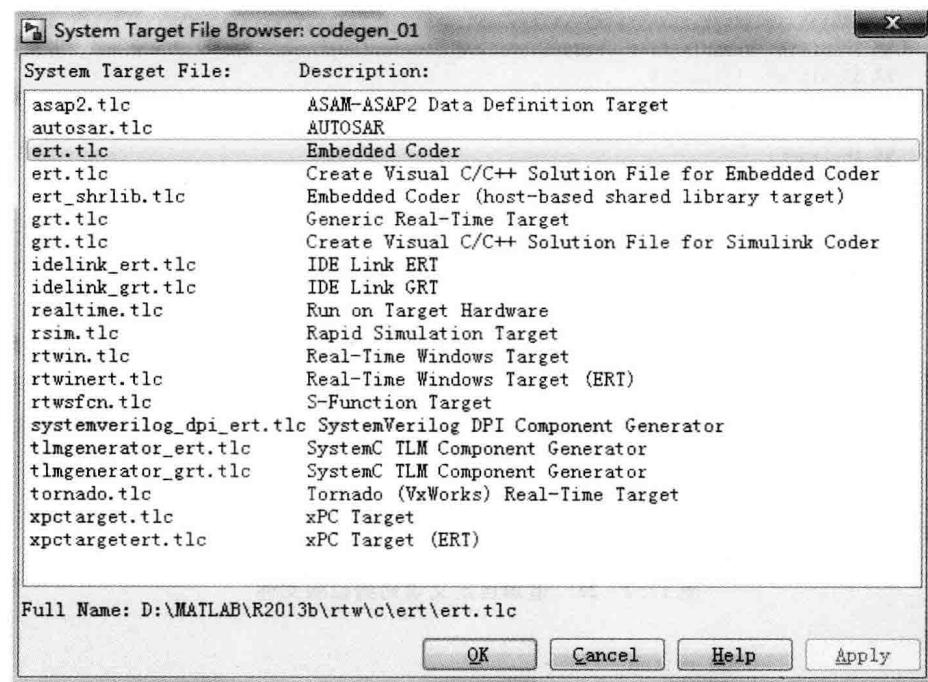


图 17.2-28 Simulink 提供的系统目标文件

tlc(Simulink Coder 提供),第 15 章中介绍的工业乙醇发酵模型使用它生成 VC6.0 下进行仿真的工程文件;rsim.tlc 可以用作快速仿真;autosar.tlc 支持模型生成符号 AUTOSAR 标准的代码;idelink\_ert.tlc 用于连接 CCS 软件作为代码生成后编译链接下载的工具链的系统目标文件,支持的目标芯片为 TI(德州仪器)的 DSP 芯片等。

每一个支持代码生成的 Simulink 模块都具有一个模块目标文件,后缀也是.tlc,名字与对应的 S 函数相同,且存储在相同的路径下。二者既合作又分工,S 函数(如果是 C Mex S 函数则是其可执行文件 mexw32/mexw64)负责仿真时执行,模块目标 tlc 文件负责编译模型时生成代码。模块 tlc 文件其内部组成类似 S 函数,由多个执行时间有前后之分的多态方法构成,这些方法被系统目标文件调用。

关于 TLC 的详细讲解请参考第 18 章“TLC 语言”。

### 17.2.3 代码生成方法与技巧

Simulink 模型生成代码的流程和基本配置方法都了解之后,可以进一步学习 Simulink 生成代码的原理,并掌握生成代码的模型配置方法,熟悉 Simulink 数据对象为基础的编程技巧,生成可以满足各种应用场合需求的代码。首先来看一下模块是如何生成代码的,生成的代码包括哪些组成部分。

**注意:**如果读者朋友使用 32 位 MATLAB 进行代码生成,编译模型并生成代码时可能经常会遇到 Out of Memory 的错误,这时即使输入 clear all; bdclose all; 等命令也无法让 MATLAB/Simulink 模型继续完成代码生成工作。建议重启 MATLAB。如果日常工作中严重影响使用,请考虑更换 64 位 MATLAB。

## 1. Simulink 模块代码生成的结构

在前面的章节已经明确了模块是构成 Simulink 模型的最小单元,那么模型生成的代码肯定也源自这些模块,模块的构成包括输入/输出端口、参数和状态量等,模块的这些构成部分分别如何生成代码呢?

如图 17.2-29 所示。可明确将一个模块分为 7 个部分,除去表示接地的 rtGround 之外,其余 6 个部分都明确了生成代码时所使用的变量生成格式,都对构成模块的各个对象的生成变量进行了结构体的约束,并且对结构体的变量名前缀进行了规定,详见表 17.2-6 所列。

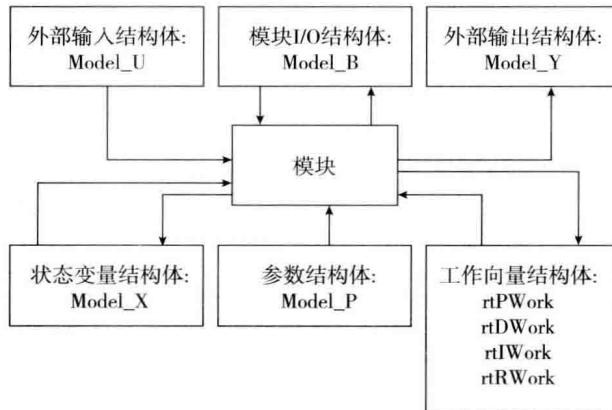


图 17.2-29 模块的组成要素对应的代码生成

表 17.2-6 模块构成要素生成结构体名

| 模块的构成要素 | 生成结构体的名字                           |
|---------|------------------------------------|
| 外部输入    | Model_U                            |
| 模块 IO   | Model_B                            |
| 外部输出    | Model_Y                            |
| 工作向量    | rtRWork, rtDWork, rtIWork, rtPWork |
| 模块参数    | Model_P                            |
| 模块状态变量  | Model_X                            |

结构体名中 Model 表示当前模型名。以表 17.2-1 所列代码为例,模型的输入端口 In1 生成代码为 codegen\_03\_U.In1,输出端口 Out1 的结构体成员变量为 codegen\_03\_Y.Out1,而 codegen\_03\_P.Gain\_Gain 则为 Gain 模块的结构体中一个成员。可以看出,ert.tlc 这个系统目标文件引领下生成的代码遵循表 17.2-6 所列命名规则,并且每个结构体中将“模块名”作为输入/输出端口结构体的成员变量,使用“模块名\_参数名”作为参数结构体变量的成员变量名。有了这个规则,即使代码不做可读性优化,相信读者们也有信心能够读懂它们了。

## 2. Simulink 信号线在不同存储类型下的代码生成

信号线生成代码是如何控制的呢?右击模型中的某根信号线,在弹出的菜单中选择 Properties,可以打开信号线的属性对话框,其第 2 个标签页是关于信号线代码生成的配置页面,在信号线不命名的情况下,代码生成功能选项是 disable 状态,如图 17.2-30 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

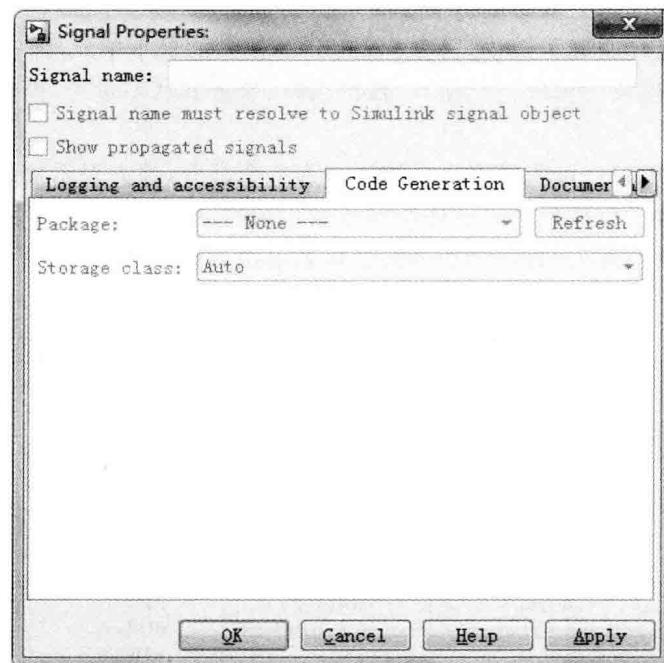


图 17.2-30 信号线属性对话框

当在 Signal name 中输入变量名之后, Package 和 Storage class 两个参数被使能, 可以选择。Package 是 Simulink 或用户自定义的包, 其中包含了一系列的存储类型(Custom Storage Class), 规定了代码生成的各种方式, 单击右侧 Refresh 按钮可以将搜索路径用户自定义的包导入。Simulink 提供 None、Simulink 和 mpt 3 种选择。即使选择 Package 为 None, Storage class 选项中也有可以使用的存储类型, 默认值为 Auto, 如图 17.2-31 所示。

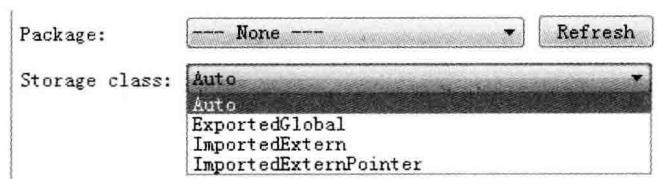


图 17.2-31 默认的存储类型

仍以图 17.2-1 所示模型为例, 将加法模块的输出线命名为 line, 模型如图 17.2-32 所示, 并将其生成代码, 说明信号线在不同存储类型下生成代码的区别。

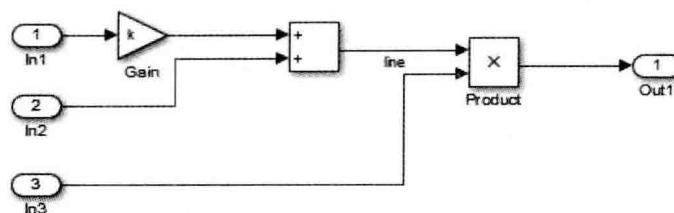


图 17.2-32 信号线 line 所在模型

在不选择 Package 时不同存储类型(Storage Class)生成代码的样例如表 17.2-7 所列。

表 17.2-7 不选择 Package 时不同存储类型下信号线生成代码比较

| 信号线存储类型                         | 生成代码声明方式   | 生成代码   |
|---------------------------------|--|--|
| Auto(signal storage reuse: on)  | —  | codegen_01_Y.Out1 = (codegen_01_P.Gain_Gain * codegen_01_U.In1 + codegen_01_U.In2) * codegen_01_U.In3;   |
| Auto(signal storage reuse: off) | In model.h:<br>typedef struct {<br>real_T Gain;<br>real_T line;<br>} B_codegen_01_T; | codegen_01_B.Gain = codegen_01_P.Gain_Gain * codegen_01_U.In1;<br>codegen_01_B.line = codegen_01_B.Gain + codegen_01_U.In2;<br>codegen_01_Y.Out1 = codegen_01_B.line * codegen_01_U.In3; |
| ExportedGlobal                  | In model.h:<br>extern real_T line;<br>In model.c:<br>real_T line;                    | line = codegen_01_P.Gain_Gain * codegen_01_U.In1 + codegen_01_U.In2;<br>codegen_01_Y.Out1 = line * codegen_01_U.In3;   |
| ImportedExtern                  | In model_private.h:<br>extern real_T line;   | line = codegen_01_P.Gain_Gain * codegen_01_U.In1 + codegen_01_U.In2;<br>codegen_01_Y.Out1 = line * codegen_01_U.In3;   |
| ImportedExternPointer           | In model_private.h:<br>extern real_T *line;  | *line = codegen_01_P.asg_Gain * codegen_01_U.In1 + codegen_01_U.In2;<br>codegen_01_Y.Out1 = *line * codegen_01_U.In3;  |

**注意：**Configuration Parameter 中 Optimization→Signals and Parameters 中 Signal storage reuse 的选择与否对表 17.2-7 中 Auto 存储类型有影响，勾选此项时，生成代码会为了节省存储用量而不为中间信号生成中间变量；反之，不勾选时，Simulink 为每一个模块的输出信号生成一个中间变量。在模型规模很大的时候，不勾选 Signal storage reuse 的选项将会增加很多存储用量。建议用户勾选此项。

### 3. Simulink 模块代码生成的接口

默认时 ert.tlc 作用下生成的模型接入点函数通常有 2 个：model\_initialize() 和 model\_step()，生成代码中函数声明如图 17.2-33 所示。model\_initialize() 将在 model\_step() 函数运行之前调用过一次初始化之后对模型进行初始化，model\_step 函数将在每个 rt\_OneStep() 中被周期性调用（需要绑定到目标硬件定时器上实现）。

```
/* Model step function */
void untitled_step(void)

/* Model initialize function */
void untitled_initialize(void)
```

图 17.2-33 模型函数接口

其实 model\_step() 函数内部包含 2 个子函数：model\_update() 和 model\_output()，分别用于计算模型中的离散状态变量及模型的输出。是合并还是分别生成，可在 Configuration Pa-

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

parameter 的 Code Generation 页面下的 Interface 子页面中的 Code Interface 参数组中通过对 Single output/update function 参数设置。合并及分别生成的代码如图 17.2-34 所示。

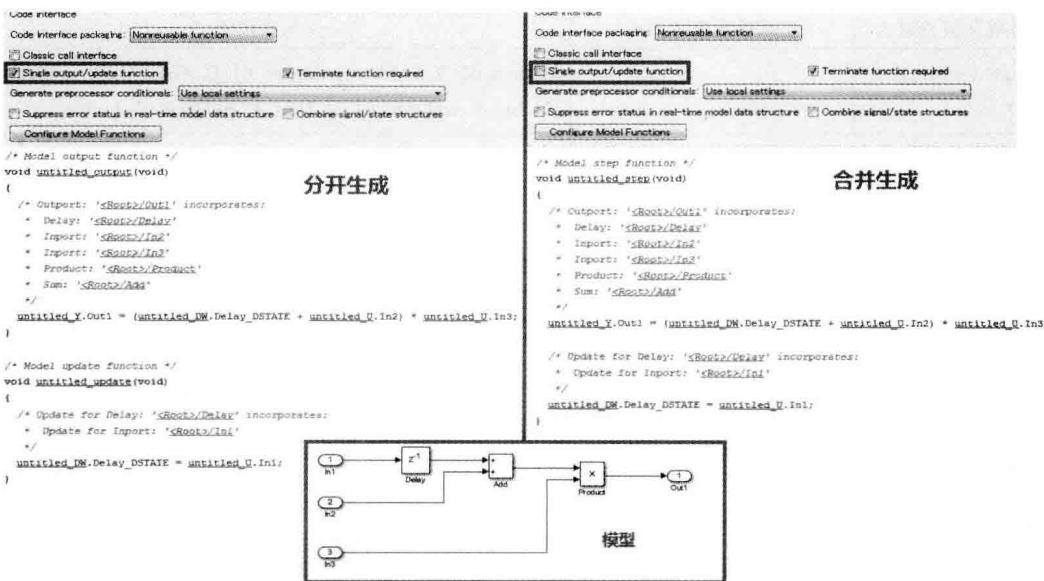


图 17.2-34 update/output 与 step 函数的选择

通过 Terminate function required 选项的勾选,可以为模型生成 model\_terminate() 函数,如图 17.2-35 所示。

```
/* Model terminate function */
void untitled_terminate(void)
{
    /* (no terminate code required) */
}
```

图 17.2-35 Terminate 函数的生成

Classic Call interface 的开启与否则决定模型生成代码的接口风格。默认不勾选,生成代码的层次较少,执行效率略高;若勾选此选项,model\_step 函数将被强行拆分为 update 和 output 函数,再封装一层 MdlOutputs() 和 MdlUpdate()。整个模型生成的代码将按照 C mex S 函数的子方法相同的方式生成代码,并在对应的子方法下调用当前模型生成函数。ert\_main.c 中将生成非常复杂的处理机制。这种旧方法的代码风格如图 17.2-36 所示。

默认情况下,ert.tlc 生成的 model\_initialize() 和 model\_step() 函数都是空参数列表,这样的代码不便于重用。如果开发者希望能够生成可重用的函数,方便在代码中多处调用,从代码紧凑简约上来讲是一个可优化之处。

预处理条件命令的生成由选项 Generate preprocessor conditionals 控制,是针对 Variant Model 的,提供 3 个选项 Use local Settings、Enable all 和 Disable all,对于不使用此模块的模型来说设置哪一个选项生成代码上都没有区别。

Suppress error status in real-time model data structure 选项勾选时将会忽略实时模型数据结构 rtModel 的错误状态域,即获取错误的宏函数不再被生成到 Model.h 文件中,这样能够减少存储量。并且 model\_terminate() 函数的执行也会从主函数中取消。

```

void MdlOutputs(int_T tid)
{
    untitled_output();
    UNUSED_PARAMETER(tid);
}

void MdlUpdate(int_T tid)
{
    untitled_update();
    UNUSED_PARAMETER(tid);
}

void MdlInitializeSizes(void)
{
}

void MdlInitializeSampleTimes(void)
{
}

void MdlInitialize(void)
{
}

void MdlStart(void)
{
    untitled_initialize();
}

void MdlTerminate(void)
{
    untitled_terminate();
}

```

```

int _T main(int _T argc, const char *argv[])
{
    untitled_rtModel *S;
    real_T finaltime = -2.0;
    int_T oldStyle_argc;
    const char_T *oldStyle_argv[5];

    /*****
     * Parse arguments
     *****/
    if ((argc > 1) && (argv[1][0] != '-')) {
        /* old style */
        if (argc > 3) {
            displayUsage();
            return(EXIT_FAILURE);
        }

        oldStyle_argc = 1;
        oldStyle_argv[0] = argv[0];
        if (argc >= 2) {
            oldStyle_argc = 3;
            oldStyle_argv[1] = "-tf";
            oldStyle_argv[2] = argv[1];
        }

        if (argc == 3) {
            oldStyle_argc = 5;
            oldStyle_argv[3] = "-port";
            oldStyle_argv[4] = argv[2];
        }
    }

    argc = oldStyle_argc;
    argv = oldStyle_argv;
}

```

图 17.2-36 旧风格生成的代码接口

rt\_main.c 中关于 rtModel 的错误状态获取以及 model\_terminate() 的调用代码在此选项的不勾选与勾选两种情况下生成的代码是有区别的, 不勾选时生成获取实时模型数据结构的接口代码, 如图 17.2-37 所示。

```

/* Macros for accessing real-time model data structure */
#ifndef rtmGetErrorStatus
#define rtmGetErrorStatus ((rtm)->errorStatus)
#endif

#ifndef rtmSetErrorStatus
#define rtmSetErrorStatus(rtm, val) ((rtm)->errorStatus = (val))
#endif

```

图 17.2-37 获取实时模型数据结构错误状态的接口

上述宏函数的创建仅在不勾选 Suppress error status in real-time model data structure 时才会生成, 并且在主函数中被调用。实时错误检测代码如图 17.2-38 所示。

勾选 Suppress error status in real-time model data structure 之后再度生成代码则出现了下面的注释和预处理命令, 如图 17.2-39 所示。

模型的信号与状态, 二者相辅相成, 信号作为模型输入/输出传递和连接的桥梁, 状态则为模型不同时刻的数据缓存提供支持。默认情况下它们会分别生成到各自的结构体中, 如图 17.2-40 所示。

如果勾选了 Combine signal/state structures 选项, 那么生成代码时模型中的信号变量及状态变量将会真正结合在一个结构体变量中, 如图 17.2-41 所示。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

while (rtmGetErrorStatus(codegen_01_M) == (NULL)) {
    /* Perform other application tasks here */
}

/* Disable rt_OneStep() here */

/* Terminate model */
codegen_01_terminate();
return 0;
}

```

图 17.2-38 判断实时模型数据结构错误状态的代码

```

/* The option 'Suppress error status in real-time model data structure'
 * is selected, therefore the following code does not need to execute.
 */
#endif
#if 0

/* Disable rt_OneStep() here */

/* Terminate model */
codegen_01_terminate();

#endif

return 0;
}

```

图 17.2-39 忽略实时模型数据并抑制终止函数的调用

```

/* Block signals (auto storage) */
typedef struct {
    real_T Gain;                                /* '<Root>/Gain' */
    real_T line;                               /* '<Root>/Sum' */
    real_T Delay;                             /* '<Root>/Delay' */
} B_codegen_01_T;

/* Block states (auto storage) for system '<Root>' */
typedef struct {
    real_T Delay_DSTATE;                      /* '<Root>/Delay' */
} DW_codegen_01_T;

```

图 17.2-40 默认情况下信号变量与状态变量分别存储

```

/* Block signals and states (auto storage) for system '<Root>' */
typedef struct {
    real_T Gain;                                /* '<Root>/Gain' */
    real_T line;                               /* '<Root>/Sum' */
    real_T Delay;                            /* '<Root>/Delay' */
    real_T Delay_DSTATE;                      /* '<Root>/Delay' */
} DW_codegen_01_T;

```

图 17.2-41 信号与状态变量生成在一个结构体中

上述的种种开关设置,是关于预处理条件命令、数据变量结合方式、实时模型数据结构相关功能的使能与否,并不真正改变模型或子系统生成函数时的接口。Configuration Model functions 按钮则真正意义上提供了一个自定义模型入口函数和子系统函数接口的方式。这个按钮仅在系统目标文件为 `ert.tlc` 及以 `ert.tlc` 为基础系统目标的文件中存在。

### (1) Simulink 模型入口函数接口自定义

单击 Configuration Model functions 按钮,会弹出一个对话框,如图 17.2-42 所示。

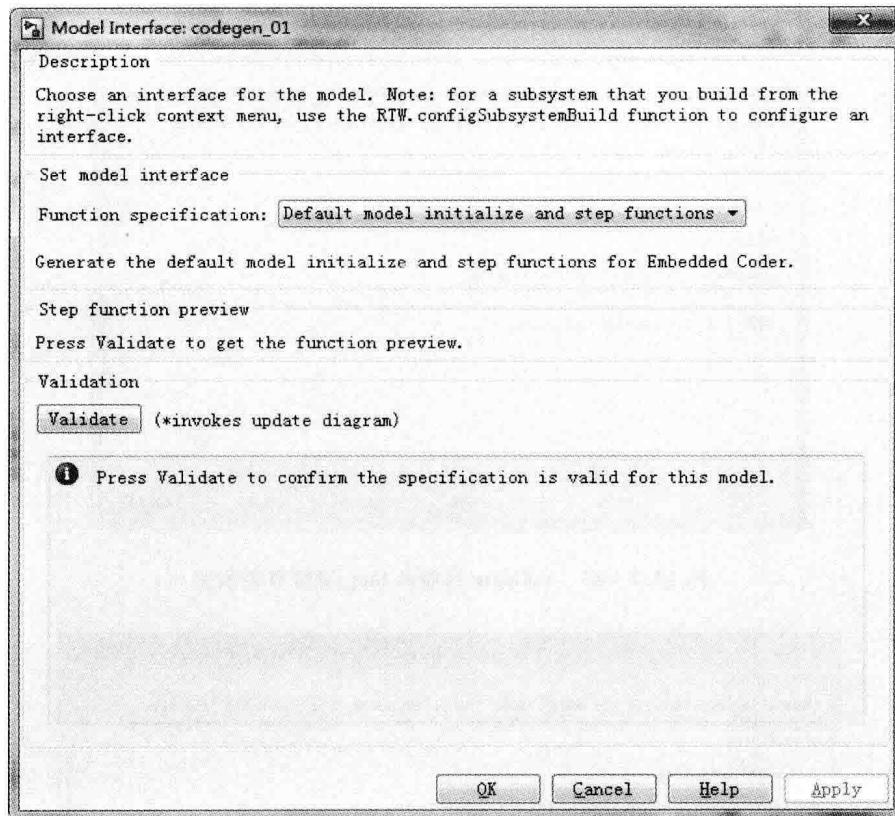


图 17.2-42 Configuration Parameter Function

默认时,此对话框提供 2 个控件,Function specification 的下拉框和 Validate 按钮。Function specification 下拉框有 2 个选项:Default model initialize and step functions 和 Model specific C prototypes。当选择 Default model initialize and step funtions 选项时,模型生成的 model\_initialize() 和 model\_step() 采用默认的函数原型,单击 Validate 按钮可以对模型进行验证,验证 OK 之后将会把 model\_step 函数的原型作为预览显示在 Step function preview 栏目下,如图 17.2-43 所示。

在默认函数原型时,用户是无法对其进行自定义的。

当 Function specification 选择 Model specific C prototypes 时,对话框将发生变化(如图 17.2-44 所示),Validate 按钮处变为 Get Default Configuration 按钮,Validate 则出现到 Step function arguments 表格的左下方。Get Default Configuration 按钮能够获取自定义默认参数列表;模型初始化函数及 step 函数名则直接以 Edit 框的形式提供给用户,可以重命名;模型的输入/输出,则按照次序以表格的方式提供给用户以设定其接口元素,并最终成为 step function 的函数原型组成部分。

输入/输出端口作为函数的参数罗列在 Step function arguments 的表格里,包括 6 列,其中后 3 列参数可配置,如表 17.2-8 所列。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

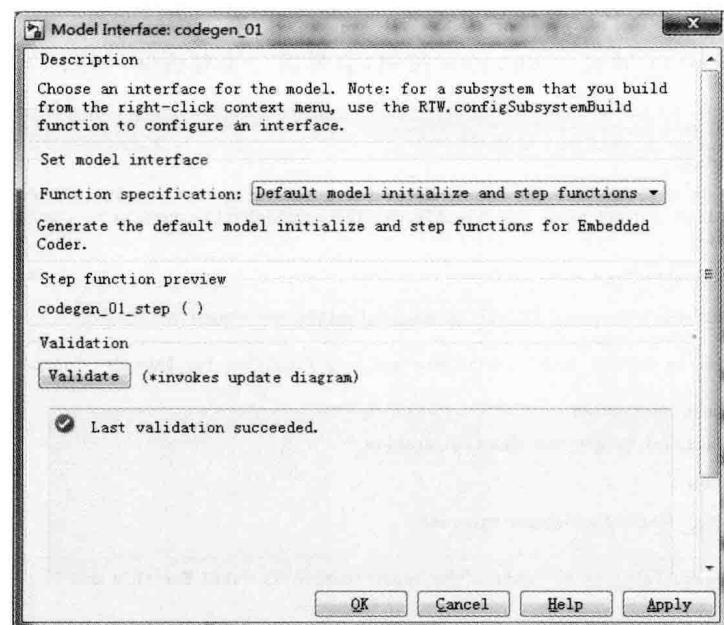


图 17.2-43 Validate 后显示 Step 函数原型预览

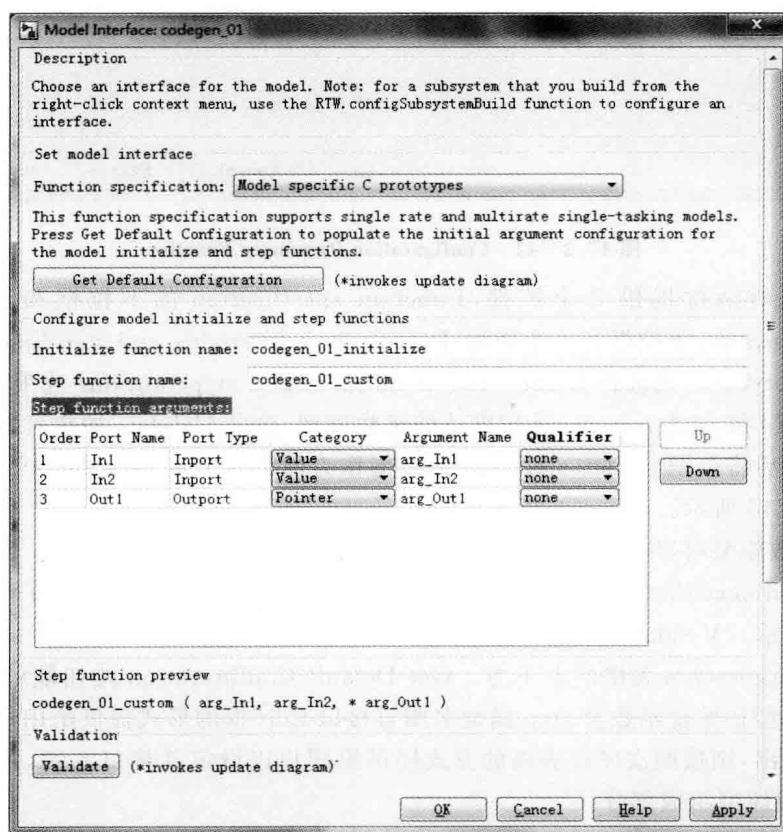


图 17.2-44 自定义 C 函数原型

表 17.2-8 Step Function 参数

| 列名            | 说明  |
|---------------|---|
| Order         | 参数顺序号,通过右侧 Up/Down 按钮可以调整参数顺序               |
| Port Name     | 端口模块名,只读                                    |
| Port Type     | 端口类型(输入或输出端口),只读                            |
| Category      | 参数传递方式选择(Value 值传递,Pointer 通过指针传递地址)        |
| Argument Name | 参数变量名                                       |
| Qualifier     | 变量限定符(const, const *, const * const 或 none) |

表格中配置的选项反映到 step function preview 的显示框中。如将 step function name 改为 test\_function, 输入参数 In1 的值传递类型设置 Pointer, 输入 In2 的限定符设置为 const, 输出的参数 Out1 传递类型选为 Pointer。则 Step function preview 处自动显示出更新后的函数原型预览, 限定符不显示在其中。单击 Validate 来验证设置的有效性。验证成功之后, 左下角显示绿色勾号, 如图 17.2-45 所示。

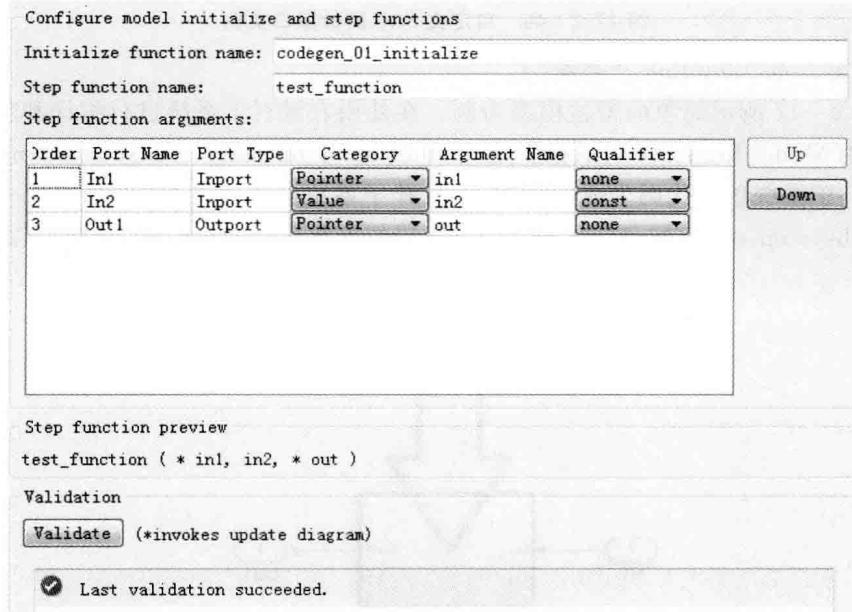


图 17.2-45 自定义函数接口实例

单击 OK 按钮之后, 再启动代码生成, 观察代码生成的实际原型, 函数原型与设定一致, 标示符也包含在其中。输入/输出端口参数则直接使用 Step function arguments 中的 Argument Name, 不再使用模块生成的端口结构体变量(codegen\_01\_U.In1 等)表示。如此, Step 函数的接口实现了自定义, 如图 17.2-46 所示。

## (2) Simulink 模型子系统函数接口自定义

为了简化模型结构, 在建立模型时往往将实现同一个功能的众多模块组建立为子系统。仅仅外表以子系统方式存在, 内部执行时间依旧保持原模块设定的子系统称为虚拟子系统, 将整个子系统设置为原子子系统之后, 每个模块的执行时间都一致化了, 子系统的边框也加粗显

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

/* Model step function */
void test_function(real_T *in1, const real_T in2, real_T *out)
{
    /* Gain: '<Root>/Gain' incorporates:
     *   Import: '<Root>/In1'
     */
    codegen_01_DW.Gain = 5.0 * *in1;

    /* Sum: '<Root>/Sum' incorporates:
     *   Import: '<Root>/In2'
     */
    codegen_01_DW.line = codegen_01_DW.Gain + in2;

    /* Delay: '<Root>/Delay' */
    codegen_01_DW.Delay = codegen_01_DW.Delay_DSTATE;

    /* Outport: '<Root>/Out1' incorporates:
     *   Product: '<Root>/Product'
     */
    *out = codegen_01_DW.line * codegen_01_DW.Delay;

    /* Update for Delay: '<Root>/Delay' */
    codegen_01_DW.Delay_DSTATE = *out;
}

```

图 17.2-46 自定义 Step 函数接口实例

示(详细参考第 4 章“Simulink 子系统”)。

以图 17.2-47 所示简单的增益模型为例。在使用右键对子系统进行编译和代码生成时,同样可以使用 Model Interface 对话框自定义 subsystem\_initialize() 和 subsystem\_step() 两个函数的原型。子系统单独代码生成的方式为:打开子系统的右键菜单,选择 C/C++ Code→Build this subsystem,如图 17.2-48 所示。弹出设置参数属性的对话框,如图 17.2-49 所示。

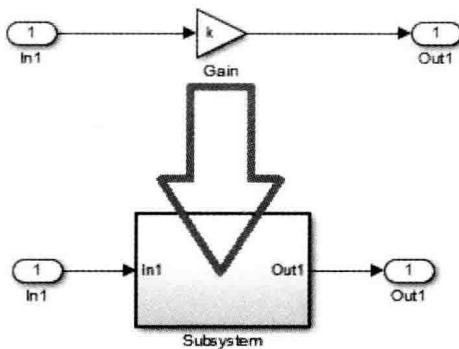


图 17.2-47 增益子系统

单击 Build 按钮即可启动子系统的代码生成,成功编译后也会弹出代码的报告。为了自定义子系统的入口函数,需要使用命令启动 Model Interface 对话框:

```
RTW.configSubsystemBuild(gcbh)
```

使用这个命令的前提是模型系统目标文件设置 ert.tlc,运行上述代码之后,弹出如图 17.2-50 所示对话框。

这个对话框与 Configuration Model functions 按钮弹出的对话框基本相同,只不过作用对象不同,Configuration Model functions 按钮针对整个模型,而此处仅针对子系统的代码生成,Description 处也有说明。Function Specifications 里包括 Default Model initialize and step

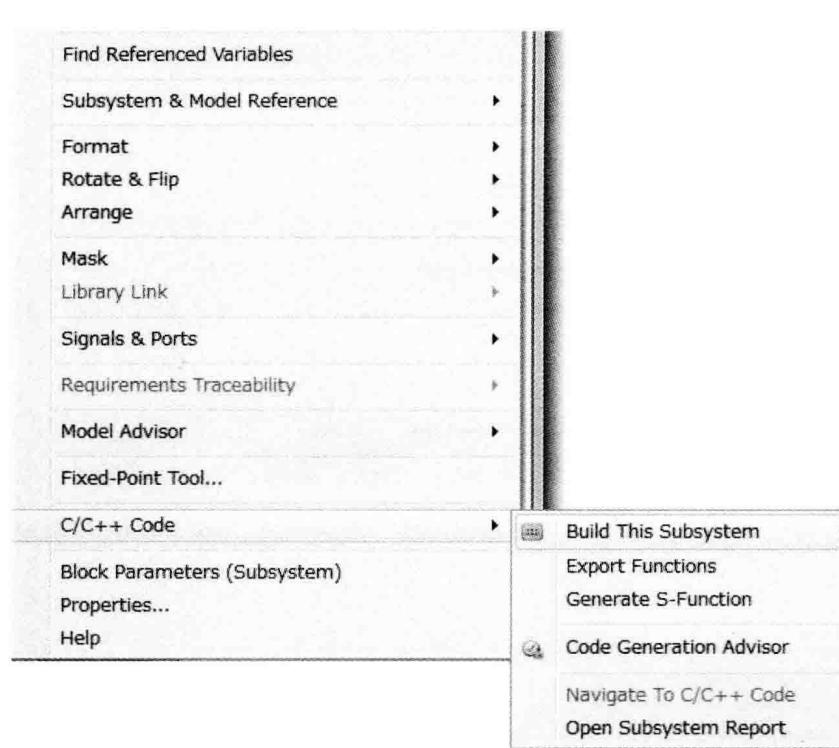


图 17.2-48 子系统右键编译生成代码菜单

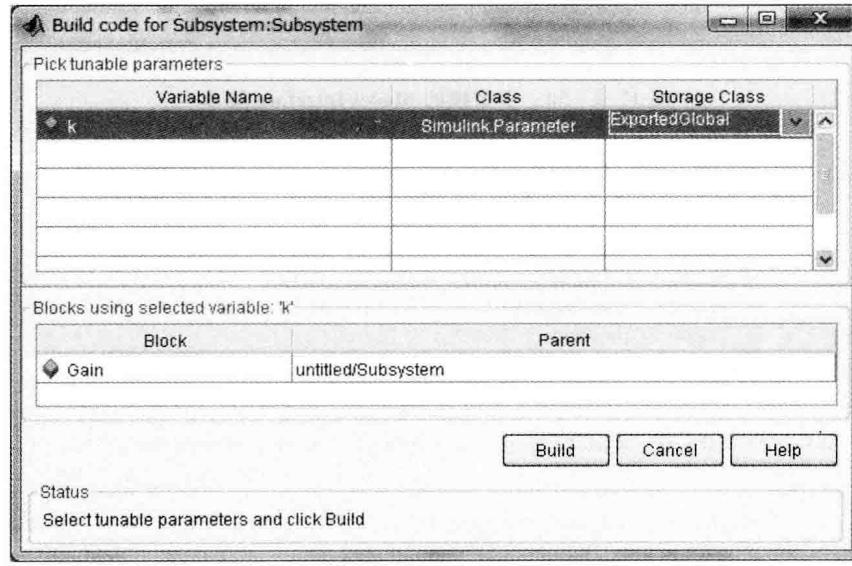


图 17.2-49 参数及变量设置菜单

function 和用户自定义的 Model specific C prototypes。在上述界面中设置完参数列表和函数名之后, 使用右键菜单编译生成代码如图 17.2-51 所示。

子系统在单独生成代码与其在整个模型里生成代码时所使用的配置是不同的, 图 17.2-46 所示模型(此时模型名为 codegen\_02.slx)使用组合键 Ctrl+b 进行代码生成时, 上述 Model Inter-

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

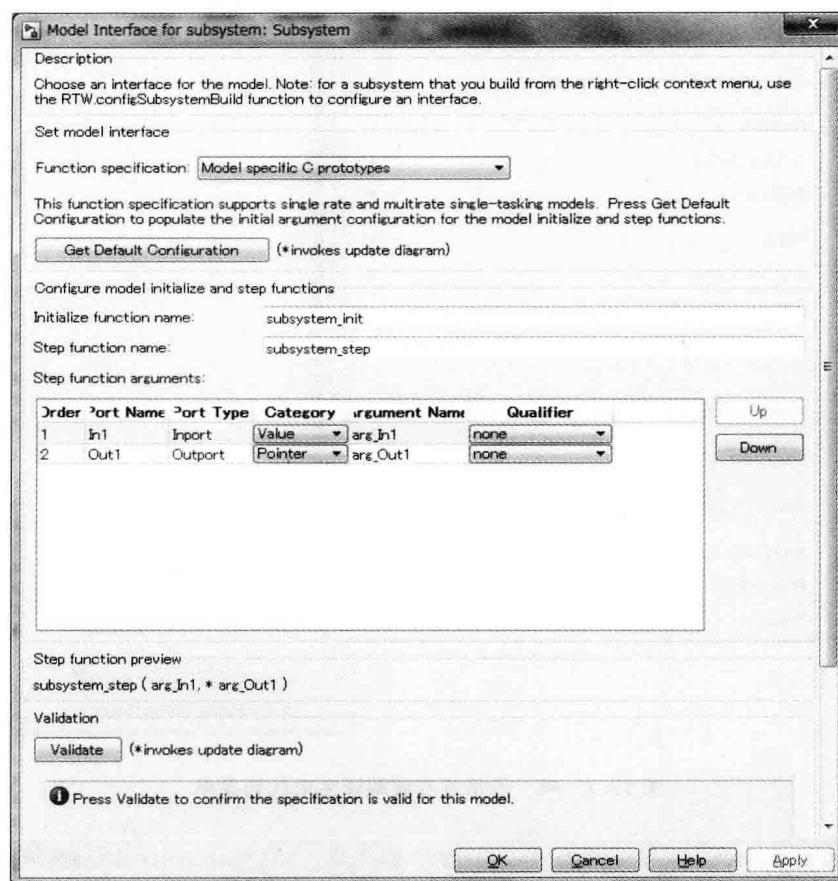


图 17.2-50 子系统的 Model Interface 菜单

```

/* Model step function */
void subsystem_step(real_T arg_In1, real_T *arg_Out1)
{
    /* Outputs for Atomic SubSystem: '<Root>/Subsystem' */
    /* Outport: '<Root>/Out1' incorporates:
       * Gain: '<S1>/Gain'
       * Inport: '<Root>/In1'
     */
    *arg_Out1 = k * arg_In1;

    /* End of Outputs for SubSystem: '<Root>/Subsystem' */
}

/* Model initialize function */
void subsystem_init(void)
{
    /* Registration code */

    /* initialize error status */
    rtmSetErrorStatus(Subsystem0_M, (NULL));
}

```

图 17.2-51 自定义子系统的函数实现

face 对话框的设定便完全不起作用。rtwbuild 作用之后生成的代码如图 17.2-52 所示。

```

/* Model step function */
void codegen_02_step(void)
{
    /* Outputs for Atomic SubSystem: 'Root/SubSystem' */
    /* Outport: 'Root/Out1' incorporates:
     * Gain: 'S1/Gain'
     * Inport: 'Root/In1'
     */
    codegen_02_Y.Out1 = k * codegen_02_U.In1;

    /* End of Outputs for SubSystem: 'Root/SubSystem' */
}

/* Model initialize function */
void codegen_02_initialize(void)
{
    /* Registration code */

    /* initialize error status */
    rtmSetErrorStatus(codegen_02_M, (NULL));

    /* external inputs */
    codegen_02_U.In1 = 0.0;

    /* external outputs */
    codegen_02_Y.Out1 = 0.0;
}

```

图 17.2-52 原子子系统在模型中的代码生成

如何在整个模型编译生成代码的时候自定义子系统的函数接口呢？右击子系统之后选择 Block Parameters(Subsystem) 属性，弹出的对话框中包含两个页面，Main 和 Code Generation。Code Generation 页面就是模型生成代码时子系统的代码生成方式的设置之处，如图 17.2-53 所示。

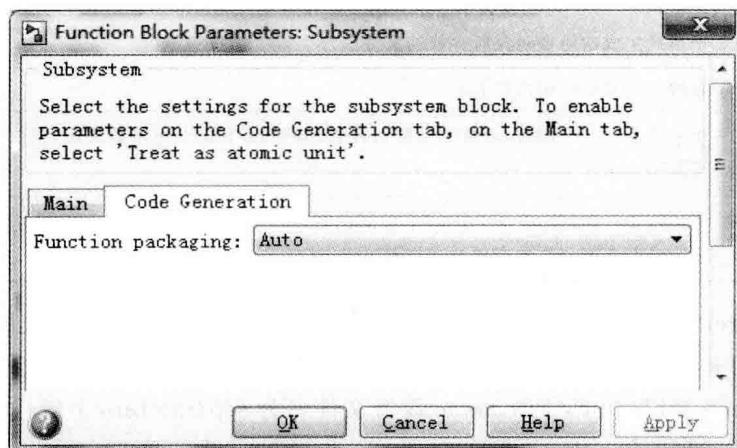


图 17.2-53 代码生成设置页面

此对话框默认仅一个参数 Function packaging，承载它的是 popup 控件，默认值为 Auto，

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

另外的选项为：Inline、Nonreusable Function、Reusable Function。当选择后两个选项时，对话框又会多出两个参数 Function Name options 和 File name options，选择 ert.tlc 之后还会出现 Memory Section 相关参数。设为 Nonreusable Function 时还可以设置 Function interface 及函数是否生成单独的数据。图 17.2-54 所示为选择 Reusable Function 时的 Function Block Parameters 对话框。

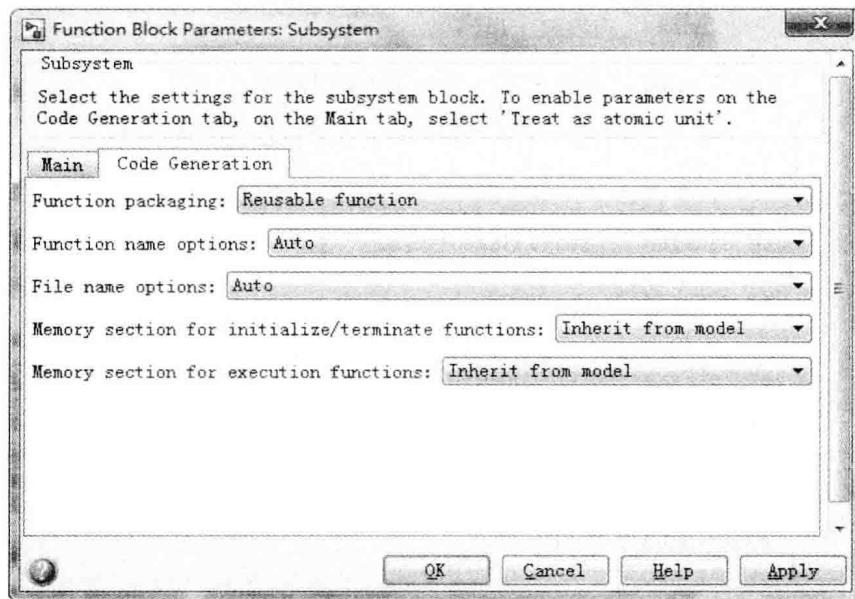


图 17.2-54 原子子系统生成函数的设置

Function packaging 的 4 个选项各自的功能如表 17.2-9 所列。

表 17.2-9 Function packaging 的选项

| 选 项                  | 功 能  |
|----------------------|--|
| Auto                 | Simulink Coder 基于模型中子系统的个数和类型自动选取最优的代码生成格式。此时不能控制子系统生成的函数名及所存放的文件    |
| Inline               | 直接将子系统代码内联展开   |
| Nonreusable function | 子系统生成一个函数，函数的参数列表由 Function Interface 参数决定，函数名及函数所存放的文件都可以指定         |
| Reusable function    | 当模型中存在同一个子系统的多个实例时，Simulink Coder 为其生成带有可重用参数列表的函数。函数名及函数所存放的文件都可以指定 |

当选择 Nonreusable function 时，控件会更新，以图 17.2-55 所示形式展示出来。

Function name options 和 File name options 都设置为 User specified。Function name 中设置函数名为 un reusable\_fun，File name 设置文件名为 separatefun，生成代码时，子系统将会生成到函数 un reusable\_fun 中，并保存在 separate\_fun.c 中。Function interface 选择 void\_void 表示生成的函数参数列表的输入输出都为 void。而 Function with separate data 的勾选则会将子系统内的参数单独生成到一个数据文件中。将上述设置的模型编译之后，代码生成如图 17.2-56～图 17.2-58 所示。

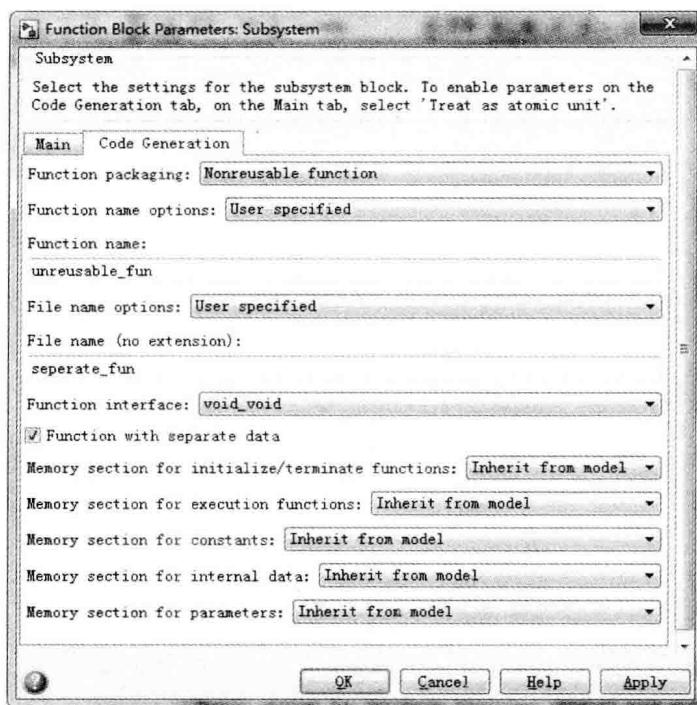


图 17.2-55 Reusable function 的设置

File: seperate\_fun.c

```

1  /*
2   * File: seperate_fun.c
3   *
4   * Code generated for Simulink model 'codegen_02'.
5   *
6   * Model version           : 1.2
7   * Simulink Coder version : 8.5 (R2013b) 08-Aug-2014
8   * C/C++ source code generated on : Sat Dec 20 00:40:08 2014
9   *
10  * Target selection: ext.tlc
11  * Embedded hardware selection: 32-bit Generic
12  * Code generation objectives: Unspecified
13  * Validation result: Not run
14  */
15
16 #include "seperate_fun.h"
17
18 /* Include model header file for global data */
19 #include "codegen_02.h"
20 #include "codegen_02_private.h"
21
22 /* Output and update for static system: '(Root)/Subsystem' */
23 void unreusable_fun(void)
24 {
25     /* Outport: '(Root)/Out1' incorporates:
26      * Gain: '(S1)/Gain'
27      * Import: '(Root)/Int'
28      */
29     codegen_02_Y.Out1 = unreusable_fun.P.Gain_Gain * codegen_02_U.In1;
30 }
31
32 /*
33  * File trailer for generated code.
34  *
35  * [EOF]
36  */
37

```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

**File: seperate\_fun\_data.c**

```

1  /*
2  * File: seperate_fun_data.c
3  *
4  * Code generated for Simulink model 'codegen_02'.
5  *
6  * Model version           : 1.2
7  * Simulink Coder version : 8.5 (R2013b) 08-Aug-2013
8  * C/C++ source code generated on : Sat Dec 20 00:40:08 2014
9  *
10 * Target selection: ert.tlc
11 * Embedded hardware selection: 32-bit Generic
12 * Code generation objectives: Unspecified
13 * Validation result: Not run
14 */
15
16 #include "codegen_02.h"
17 #include "codegen_02_private.h"
18
19 P_unreusable_fun T unreusable_fun_P = {
20     5.0
21     /* Expression: k
22      * Referenced by: <S1>/Gain
23      */
24 };
25 /*
26 * File trailer for generated code.
27 *
28 * [EOF]
29 */
30

```

图 17.2-57 子系统中的参数生成在单独数据文件中

**[–] Model files**

[codegen\\_02.c](#)  
[codegen\\_02.h](#)  
[codegen\\_02\\_private.h](#)  
[codegen\\_02\\_types.h](#)

```

29 /* Model step function */
30 void codegen_02_step(void)
31 {
32     /* Outputs for Atomic SubSystem: '<Root>/Subsystem' */
33     unreusable_fun();
34
35     /* End of Outputs for SubSystem: '<Root>/Subsystem' */
36 }
37

```

图 17.2-58 model\_step 函数中调用这个子系统函数

生成的函数输入输出列表均为空,单独保存在 C 文件中,并且 gain 模块的参数也单独保存在 C 文件 model\_data.c 中。

当模型中同样的子系统存在多个实例时,为了使生成的代码简洁,可以将子系统的每个实例的 Function packaging 均设置为 reusable,如图 17.2-59 中两个子系统内部模型完全相同。

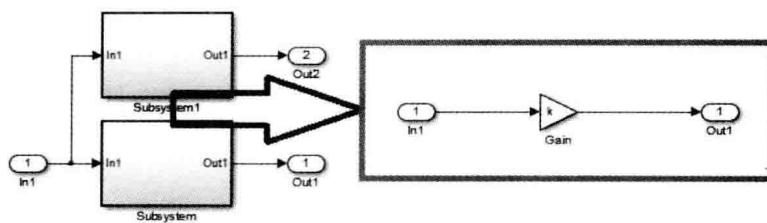


图 17.2-59 模型中同一子系统存在多个实例

Code Generation 的设置如图 17.2-60 所示。

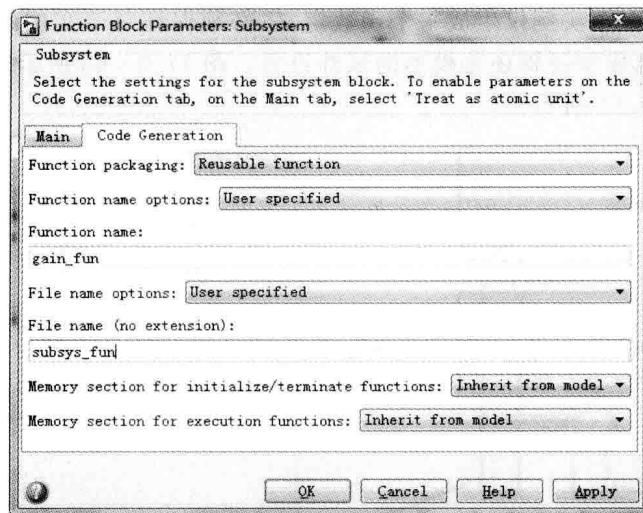


图 17.2-60 子系统设置生成 reusable 函数

生成的代码将这个子系统的函数单独保存在 subsys\_fun.c 中, subsys\_fun.h 中为其头文件, 如图 17.2-61 所示。

```

[-] Model files
  codegen_02.c
  codegen_02.h
  codegen_02_private.h
  codegen_02_types.h

[-] Subsystem files
  subsys_fun.c
  subsys_fun.h

```

```

22  /*
23   * Output and update for atomic system:
24   *   <Root>/Subsystem
25   *   <Root>/Subsystem1
26   */
27 void gain_fun(real_T rtu_In1, B_gain_fun_I *localB, P_gain_fun_I *localP)
28 {
29   /* Gain: 'CS1/Gain' */
30   localB->Gain = localP->Gain_Gain * rtu_In1;
31 }
32 /*
33   * File trailer for generated code.
34   *
35   * [EOF]
36 */
37

```

图 17.2-61 子系统生成的可重用函数

在模型的 model\_step 函数中根据子系统的每个实例(subsystem 和 subsystem1)进行函数调用, 如图 17.2-62 所示。

```

/* Model step function */
void codegen_02_step(void)
{
  /* Outputs for Atomic SubSystem: '<Root>/Subsystem' */
  /* Inport: '<Root>/In1' */
  gain_fun(codegen_02_U.In1, &codegen_02_B.Subsystem, (P_gain_fun_I *)
    &codegen_02_P.Subsystem);

  /* End of Outputs for SubSystem: '<Root>/Subsystem' */

  /* Outport: '<Root>/Out1' */
  codegen_02_Y.Out1 = codegen_02_B.Subsystem.Gain;

  /* Outputs for Atomic SubSystem: '<Root>/Subsystem1' */
  /* Inport: '<Root>/In1' */
  gain_fun(codegen_02_U.In1, &codegen_02_B.Subsystem1, (P_gain_fun_I *)
    &codegen_02_P.Subsystem1);

  /* End of Outputs for SubSystem: '<Root>/Subsystem1' */

  /* Outport: '<Root>/Out2' */
  codegen_02_Y.Out2 = codegen_02_B.Subsystem1.Gain;
}

```

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

为了使子系统的多个实例仅生成一个函数定义,必须保证每个实例的子系统设置都是完全相同的,无论是内部模型逻辑还是模型的属性设置。图 17.2-63 所示模型中子系统生成单独的函数,如图 17.2-64 所示。

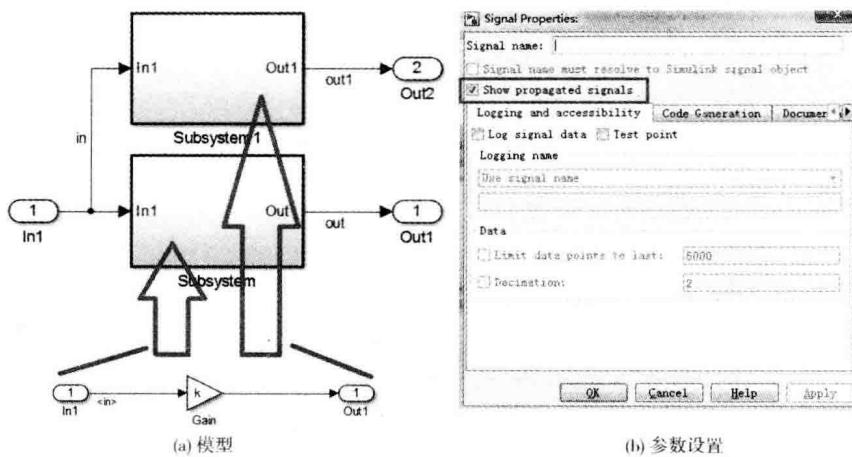


图 17.2-63 Reuse function 子系统中信号线设置

```
void gain_fun(real_T rtu_In1, B_gain_fun_I *localB)
{
    /* Gain: <SI>/Gain */
    localB->Gain = k * rtu_In1;
}
```

图 17.2-64 Reuse function 的多个实例生成的函数

结合信号线的存储类型设置,可以将代码再进一步简化。如图 17.2-64 所示,将所有的上层信号线都命名并将存储类型设置为 ExportedGlobal,参数 k 也设置为 ExportedGlobal 存储类型并开启 Inline parameter 选项。子系统内承接上层系统数据传递的信号线设为继承方式,则不需重新给继承的信号线命名或设置存储类型,并且能够使子系统的两个实例具有相同的输入变量。

模型的 model\_step 函数中为每个子系统的实例生成一个调用上述函数的代码,经过存储类型的优化之后,代码变得简洁,可读性提高。生成的代码如图 17.2-65 所示。

```
/* Model step function */
void codegen_02_step(void)
{
    /* Outputs for Atomic SubSystem: 'Root/Subsystem' */
    /* Import: <Root>/In1 */
    gain_fun(in, &codegen_02_B.Subsystem);

    /* End of Outputs for SubSystem: 'Root/Subsystem' */

    /* SignalConversion: 'Root/ImpSignal ConversionAtSubsystemOutput1' */
    out = codegen_02_B.Subsystem.Gain;

    /* Outputs for Atomic SubSystem: 'Root/Subsystem1' */
    /* Import: <Root>/In1 */
    gain_fun(in, &codegen_02_B.Subsystem1);

    /* End of Outputs for SubSystem: 'Root/Subsystem1' */

    /* SignalConversion: 'Root/ImpSignal ConversionAtSubsystemOutput1' */
    out1 = codegen_02_B.Subsystem1.Gain;
}
```

图 17.2-65 Reuse function 的信号线设置

#### 4. Simulink 自定义存储类型与信号/参数数据对象

Simulink 模型中的数据可以分为 2 种：信号与参数。信号是信号线中传递的数据，随着模型的采样时刻不断变化，通常都是计算得出的量；参数是模型中各种常数，如 Gain 的增益，Look-up table 中的表格数据，既可以是算法的参数也可以是通过实验得到的数据，在模型的运行过程中通常不发生变化。在 MCU 中，信号存储在 RAM 中，参数存储在 ROM 中。Simulink 中使用数据对象表示和管理这 2 种数据，并通过数据对象的存储类型控制这 2 种数据生成代码时的表现方式。

信号线属性对话框的 Code Generation 页面下可以配置信号线的存储类型（Storage Class），如图 17.2-66 所示。此参数上方的 Package 选项，默认为 None 以外，还有 Simulink 和 mpt 2 个选项。所谓 Package 就是定义了一族存储类型 Storage Class 的超类。

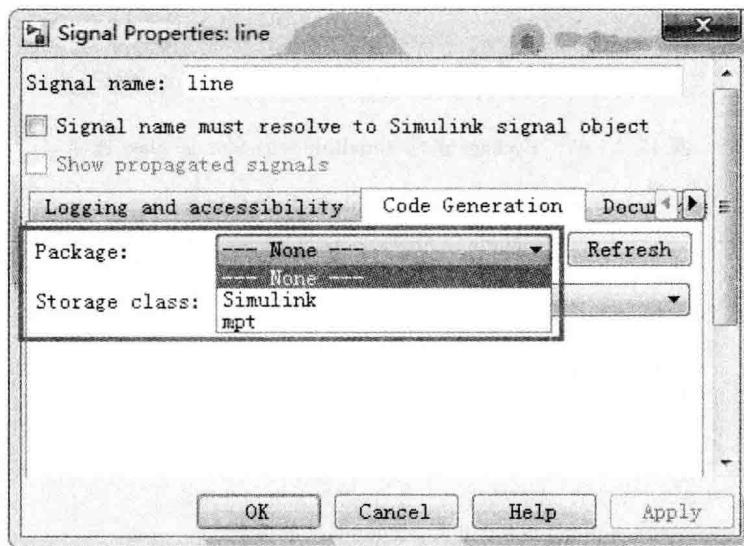


图 17.2-66 Package 选项

图 17.2-66 中的 Storage Class 为 Package 选择—None—时的可选存储类型，仅包括 Simulink 的内建类型：Auto、ExportedGlobal、ImportedExtern 和 ImportedExternPointer。这 4 种类型称为非自定义存储类型(Non-custom Storage Class)。本章图 17.2-31 已经提及在这 4 种存储类型的作用下可以控制生成代码中引用数据的方式，但是控制方式比较局限。如果选择 Simulink 或 mpt 这两个 package，Storage class 的选择有更多的选项，上述 4 种存储类型以外的称为自定义存储类型(Custom Storage class)，简称 CSC，Embedded Coder 根据自定义存储类型的设定表现到生成代码中。非自定义存储类型影响 grt.tlc 和 ert.tlc 作用下模型的代码生成，CSC 只影响 ert.tlc 作为系统目标文件的嵌入式代码生成。Simulink 包里的存储类型选择如图 17.2-67 所示。

##### (1) Simulink 包 CSC 与数据对象

Simulink 包里提供的后面带有(Custom)的存储类型，尽管是 Simulink 内建的，但是也被 Simulink 称为 CSC。具体支持对象和使用方式如表 17.2-10 所列。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

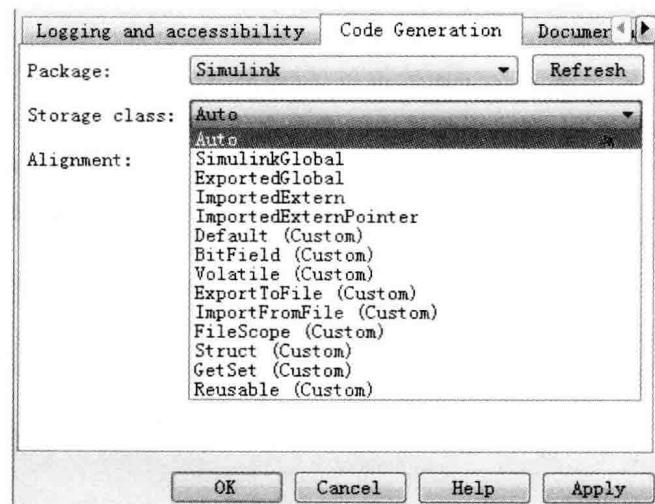


图 17.2-67 Package 选择 Simulink 时的 Storage class 选项

表 17.2-10 Simulink 包中 CSC 列表

| CSC 名            | 功 能  | 是否可用于信号 | 是否可用于参数 |
|------------------|--|---------|---------|
| Default          | 数据对象的属性 CoderInfo. CustomStorageClass 的默认设置, 直接使用模型中变量或参数名到代码中                   | 是       | 是       |
| BitField         | 声明支持布尔类型的位域结构体类型   | 是       | 是       |
| Volatile         | 生成 Voatile 类型标示符变量   | 是       | 是       |
| ExportToFile     | 生成一个头文件, 用户指定其名称, 内部包含全局变量声明, model.h 中会生成包含此头文件的 #include 命令                    | 是       | 是       |
| ImportedFromFile | 生成头文件包含命令, include 包含全局变量声明的头文件, 所包含的文件名由用户指定, 在 model_private.h 中生成 #include 命令 | 是       | 是       |
| FileScope        | 为变量声明时生成静态后缀以限定访问范围在当前文件   | 是       | 是       |
| Struct           | 将信号或变量生成结构体结构  | 是       | 是       |
| GetSet           | 跟 Data Store Memory Block 联合使用支持特殊函数调用, 如对某地址的读取与写入接口函数                          | 是       | 是       |
| Reusable         | 当一对 IO 信号设置为 Reusable 并且有相同信号名时, 代码生成器允许生成可重用的代码                                 | 是       | 否       |

下面通过一些实例说明 CSC 作用下 Embedded Coder 生成代码中数据类型的声明和接口方式。信号线生成结构体变量, 模型中使用 Bus Creator 模块, 绑定工作空间的 bus object 对象的方式, 相比之下, 直接使用 Struct 存储类型更简洁方便。图 17.2-68(a)所示模型 2 个输入和 1 个输出, 都使用 Simulink 包下的 Struct 存储类型, 输入端口的 StructName 结构体命名为 input, 输出结构体命名为 output。其中 In1 的属性对话框填写如图 17.2-68(b) 所示。

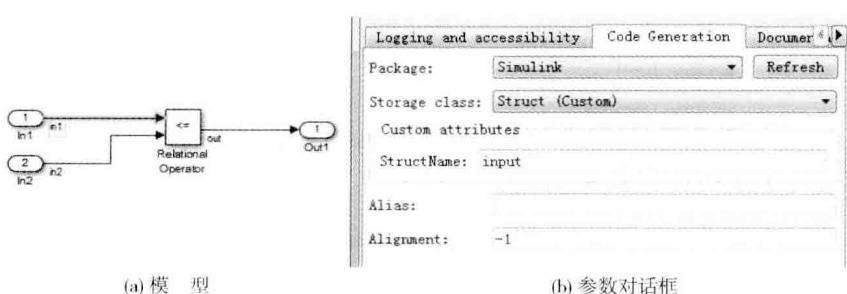


图 17.2-68 Struct 存储类型

如此配置生成的代码中，在 model\_types.h 中分别定义输入/输出口的结构体如图 17.2-69 所示。

```
/* Type definition for custom storage class: Struct */
typedef struct input_tag {
    real_T in1;
    real_T in2;
} input_type;

typedef struct output_tag {
    boolean_T out;
} output_type;
```

图 17.2-69 输入/输出口的结构体定义

在 model.c 中声明结构体变量并在 model\_step 函数中使用，结构体变量名为模型信号线属性中的 StructName，而其成员——信号线变量名为信号线上的标签名，如图 17.2-70 所示。

```
/* Definition for custom storage class: Struct */
input_type input;
output_type output;

/* Real-time model */
RT_MODEL_codegen_04_T codegen_04_M;
RT_MODEL_codegen_04_T *const codegen_04_M = &codegen_04_M;

/* Model step function */
void codegen_04_step(void)
{
    /* RelationalOperator: '<Root>/Relational Operator' incorporates:
     *   Import: '<Root>/In1'
     *   Import: '<Root>/In2'
     */
    output.out = (input.in1 <= input.in2);
}
```

图 17.2-70 结构体变量的声明及使用

直接将所有不带 extern 限定符信号设置为 Default(custom) 存储类型则可以方便地实现类似存储类型为 ExportedGlobal 的效果，生成的代码如图 17.2-71 所示。

位域(BitField)是嵌入式 C 代码中常用的数据结构。对于有些变量，它不需要占据一个完整的字节，如果单独给它分配一个字节太过浪费，特别对于嵌入式 MCU 芯片来说，资源紧张必须使用紧凑的存储空间排布。位域可以将一个完整的字节划分为多个不同的二进制位区

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

/* Definition for custom storage class: Default */
real_T in1;
real_T in2;
boolean_T out;

/* Real-time model */
RT_MODEL_codegen_04_T codegen_04_M;
RT_MODEL_codegen_04_T *const codegen_04_M = &codegen_04_M;

/* Model step function */
void codegen_04_step(void)
{
    /* RelationalOperator: '<=' Relational Operator' incorporates:
     *   * Import: '<='
     *   * Import: '<='
     */
    out = (in1 <= in2);
}

/* Model initialize function */
void codegen_04_initialize(void)
{
    /* Registration code */

    /* initialize error status */
    rtmSetErrorStatus(codegen_04_M, (NULL));

    /* block I/O */

    /* custom signals */
    out = FALSE;

    /* external inputs */
    in1 = 0.0;
    in2 = 0.0;
}

```

图 17.2-71 Default(Custom)存储类型生成的代码

间,每个二进制位域存放一个 0 或一个 1,这也是 MCU 的驱动库里寄存器地址映射时常用的方法。在 Simulink 包的 BitFiled 存储类型下,可以实现单个位域结构体的生成。需要将模型的输入/输出数据类型更改为布尔型,如图 17.2-72 所示。

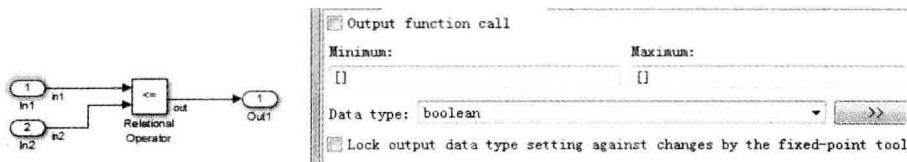


图 17.2-72 输入/输出设置为布尔型

BitField 存储类型下设置 Custom attributes 中 StructName 为 bit\_struct, 如图 17.2-73 所示。

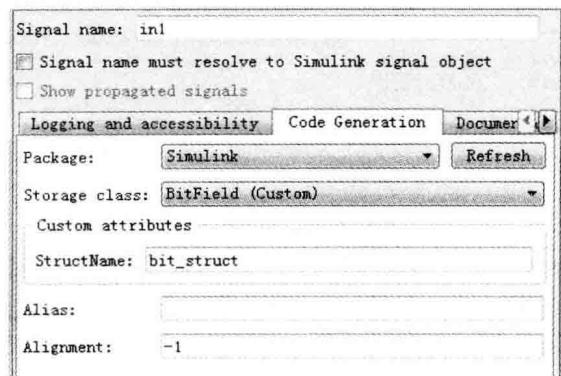


图 17.2-73 BitField 存储类型设置

在生成代码的 model\_types.h 中可以查看位域结构体数据结构的定义如图 17.2-74 所示。

```
/* Type definition for custom storage class: BitField */
typedef struct bit_struct_tag {
    uint_T in1 : 1;
    uint_T in2 : 1;
    uint_T out : 1;
} bit_struct_type;
```

图 17.2-74 位域结构体定义

model.c 中声明上述结构体的变量，并根据算法模块的数据结构类型进行数据类型转换，如图 17.2-75 所示。

```
/* Definition for custom storage class: BitField */
bit_struct_type bit_struct;

/* Model step function */
void codegen_04_step(void)
{
    /* RelationalOperator: 'Root>/Relational Operator' incorporates:
     * Import: 'Root>/In1'
     * Import: 'Root>/In2'
     */
    bit_struct.out = ((int32_T)bit_struct.in1 <= (int32_T)bit_struct.in2);
}
```

图 17.2-75 位域结构体变量的声明及使用

上面的例子中是通过右击信号线弹出的对话框进行存储类型配置的。另外一种配置方法是通过 M 语言在 Base Workspace 里创建数据对象，设置数据对象的存储类型等属性，再将数据对象绑定到信号线上以生成代码。以 Simulink 包为例，可以通过 Simulink. Signal 创建信号数据对象，通过 Simulink. Parameter 设置参数数据对象。使用语句：

```
dataobj = Simulink.Signal
```

即可创建一个信号数据对象，其变量存储在 Base Workspace 中，双击这个变量可以启动数据对象的属性对话框，如图 17.2-76 所示。

Signal 是 Simulink 类的子类，使用它创建一个实例就是信号数据对象，其属性详见表 17.2-11。

表 17.2-11 Simulink. Signal 数据对象属性列表

| 属性名             | 功能说明   |
|-----------------|--|
| Data Type       | 信号的数据类型，可以是 Simulink 内建类型也可以是通过 Simulink. NumericType 创建的自定义数据类型 |
| Complexity      | 信号为实数或虚数   |
| Dimensions      | 信号的维数，-1 表示继承，[m, n] 表示矩阵  |
| Dimensions Mode | Auto：表示维数或者可变或者固定<br>Fixed：表示维数固定<br>Variable：表示维数是可变的           |
| Sample Time     | 采样时间，即信号每隔多久计算更新一次，默认值为 -1                                       |
| Minimum         | 信号值的下限，默认 [] 不设置   |
| Maximum         | 信号值的上限，默认 [] 不设置   |

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

续表 17.2-11

| 属性名           | 功能说明   |
|---------------|--|
| Sample Mode   | 采样模式,auto/Sample based/Frame based之一   |
| Initial Value | 仿真开始前的初始值  |
| Units         | 信号所代表物理量的单位,字符串形式  |
| Storage Class | 生成代码所使用的 CSC,M 语句中使用 obj.CoderInfo.CustomStorageClass 来访问或设置   |
| Alias         | 信号别名,如果设置,则生成代码时以此作为信号名而忽略其本身的名字   |
| Alignment     | 对齐方式设置,每个信号存储的起始地址间隔多少字节。填入-1 则表示将字节对齐交给 Simulink Coder 去处理。填入其他整数需要满足一个条件,即填入的数字要是 2 的整次幂,并且不能超过 128。仿真时此属性不起作用,仅用于代码生成 |
| Description   | 用于添加描述数据对象的文字  |

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

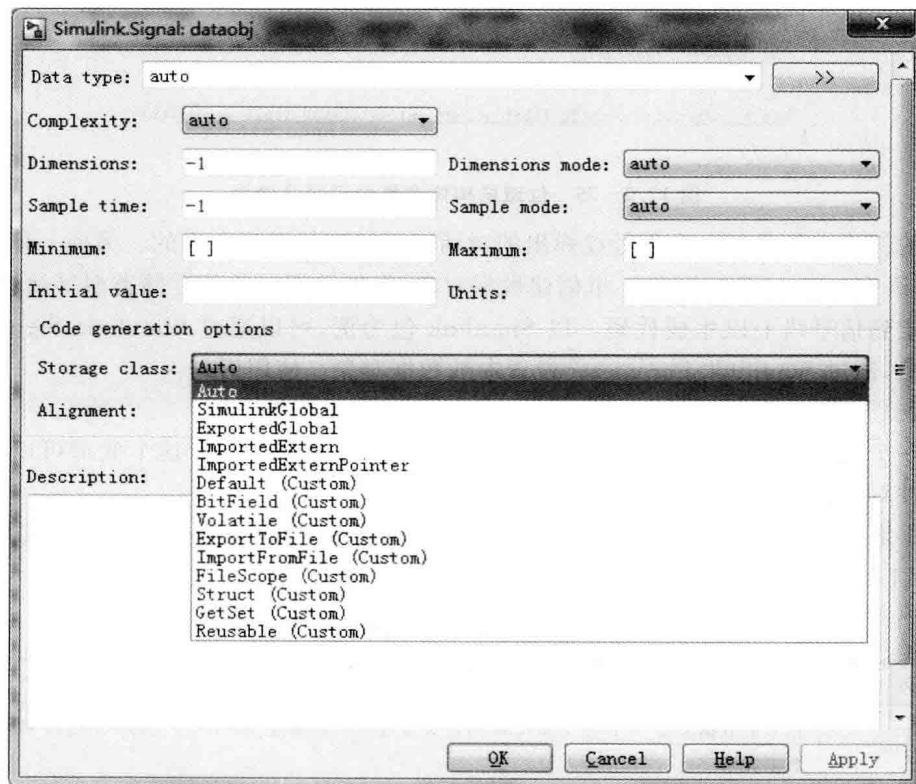


图 17.2-76 Simulink. Signal 对象属性对话框

通过 `k = Simulink.Parameter` 可以创建一个参数的数据对象。在 Workspace 中双击变量 `k` 可打开其属性对话框,其属性与信号数据对象的属性基本相同,唯一不同的就是其 `Value` 在整个模型执行过程中不变,而 `Signal` 的值是根据模型 Sample Time 更新的。由于这个差异,Parameter 对象中 `Value` 的值设置后一直保持不需要更新,且不需要设置采样模式和采样时间。数据对象 `k` 的属性对话框如图 17.2-77 所示。

可通过 M 脚本定义数据对象,对于图 17.2-72 中模型信号线的设置,可通过以下语句在 Base Workspace 里定义出具有同样效果的信号数据对象:

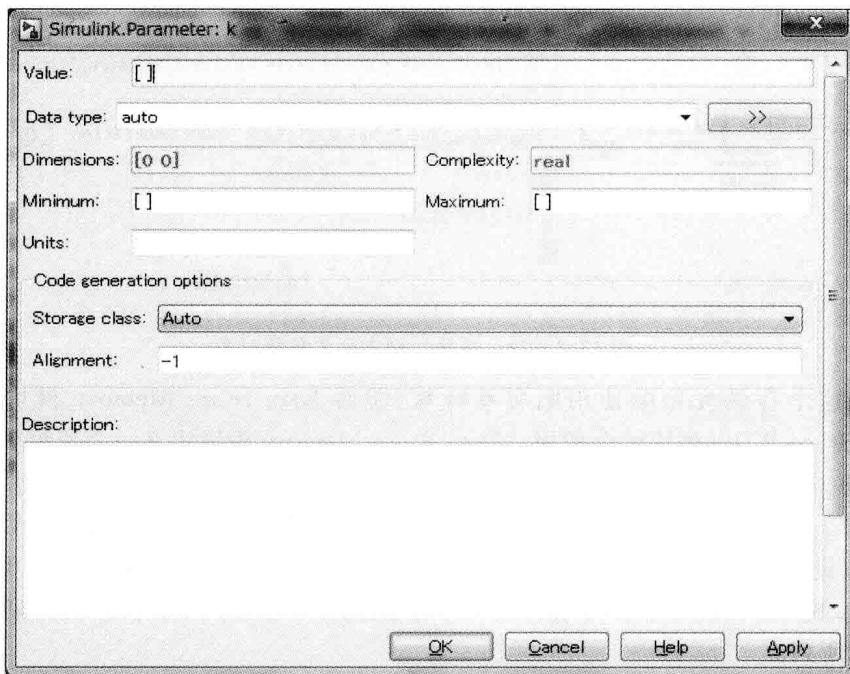


图 17.2-77 Simulink. Parameter 对象属性对话框

```

in1 = Simulink.Signal;
in1.InitialValue = '0';
in1.StorageClass = 'Custom';
in1.CoderInfo.CustomStorageClass = 'Struct';
in1.CoderInfo.CustomAttributes.StructName = 'input';

in2 = Simulink.Signal;
in2.InitialValue = '0';
in2.StorageClass = 'Custom';
in2.CoderInfo.CustomStorageClass = 'Struct';
in2.CoderInfo.CustomAttributes.StructName = 'input';

out = Simulink.Signal;
out.InitialValue = '0';
out.StorageClass = 'Custom';
out.CoderInfo.CustomStorageClass = 'Struct';
out.CoderInfo.CustomAttributes.StructName = 'output';

```

再在信号线属性对话框中勾选 Signal name must resolve to Simulink Signal object 以实现数据对象与信号线的绑定, 绑定了数据对象的信号上有一个蓝色右向三叉标记, 如图 17.2-78(a) 所示模型。

对这个选项的勾选也可以通过 M 语句来实现。首先获得模型中的信号线的句柄, 再对每条线的 MustResolveToSignalObject 属性设置为 1:

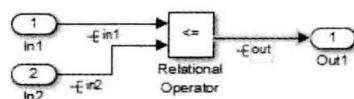
```

line_h = find_system(gcs, 'findall', 'on', 'Type', 'line');
set(line_h, 'MustResolveToSignalObject', 1);

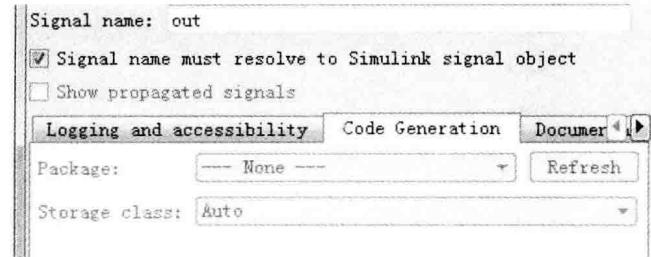
```

取消绑定通过 set(line\_h, 'MustResolveToSignalObject', 0); 实现。此处 line\_h 虽然是一个向量, 但是 set 函数可以支持向量参数输入, 将其作为一个参数整体使用。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。



(a) 模型



(b) 属性设置

图 17.2-78 信号线绑定同名数据对象

GetSet 这个存储类型的使用则有些特殊,需要 Data Store Memory 模块、Data Store Read/Data Store Write 模块配合使用。Data Store Memory 模块定义一个变量名指向一片存储区域,在同一层模型或其子层模型的 Data Store Read/Write 模块都可以对这块存储区域进行读/写操作。Data Store Memory 中的信号名与 Base Workspace 中定义的信号数据对象绑定之后,可以设置生成代码中读取/写入此信号的接口函数。当数据对象的存储类型被设置为 GetSet 之后,其 CustomAttribute 属性的三个子属性用于设置生成代码时的函数接口,详见表 17.2-12 所列。

表 17.2-12 GetSet 存储类型的接口设置属性

| 属性名         | 功 能  |
|-------------|--|
| GetFunction | 读取存储地址的函数名                                 |
| SetFunction | 写入存储地址的函数名                                 |
| HeaderFile  | 可选项,设置需要包含的头文件全名,该头文件中应该声明上述 2 个参数中配置的函数原型 |

使用 M 语言定义一个信号数据对象 A,将其存储类型设为 GetSet,并设置其 GetFunction 和 SetFunction:

```
A = Simulink.Signal;
A.CoderInfo.StorageClass = 'Custom';
A.CoderInfo.CustomStorageClass = 'GetSet';
A.CoderInfoCustomAttributes.GetFunction = 'DataRead';
A.CoderInfoCustomAttributes.SetFunction = 'DataWrite';
```

这里设置的读/写函数名只作为函数在生成的代码中被调用,其功能由用户自己提供,其声明函数原型的头文件设置到 CustomAttributes. HeaderFile 中。建立模型时必须在 Data Store Memory 内填写信号名,再通过图 17.2-79 所示选择框 check box 勾选此信号与同名的数据对象绑定。

模型中 in 与 out 信号使用 Simulink 包的 Default 存储类型,该模型与其生成的 model\_step 函数如图 17.2-80 所示。

## (2) mpt 包 CSC 与数据对象

mpt 包是 Simulink 软件提供的另一个内建类,包括信号与参数 2 种数据对象,分别使用 mpt. Signal 和 mpt. Parameter 创建,属性的配置方式同 Simulink 包的数据对象。它提供比 Simulink 包更多的 CSC 类型,如 Global 和 StructVolatile。选择 Global 存储类型时,out 的属性对话框如图 17.2-81 所示。

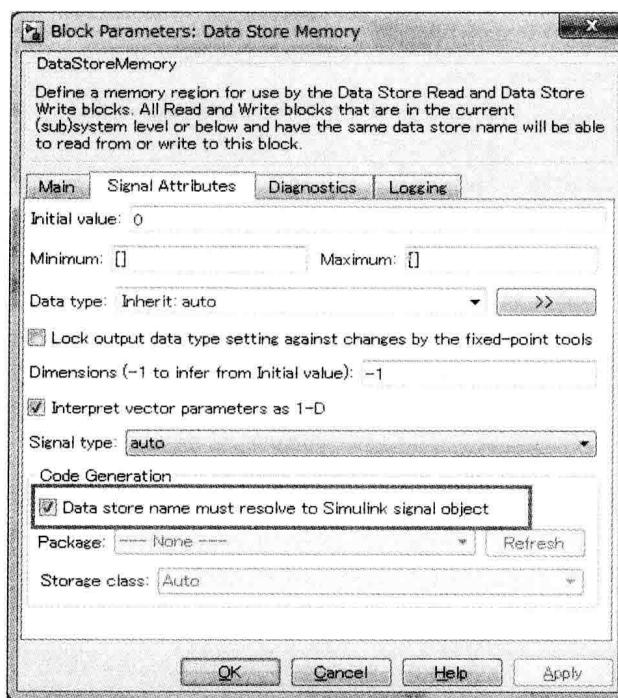


图 17.2-79 Data Store Memory 模块的 Signal Attributes 页面下设置数据对象绑定

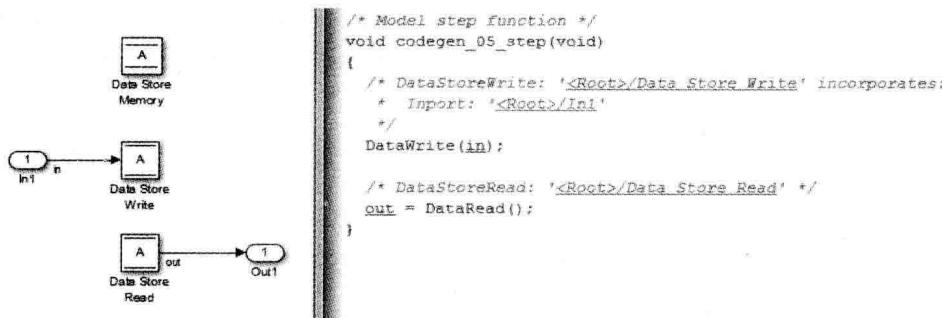


图 17.2-80 Data Store 系列模块建模及代码生成

Global 存储类型是将信号或参数数据对象作为全局变量声明及定义在生成代码中, 是 mpt 包的默认 CSC 选项。HeaderFile 和 DefinitionFile 两个属性即为声明和定义全局变量的文件名。全局变量声明时以 extern 作为限定符。MemorySection 中提供了数据对象生成代码时的限定符, 如表 17.2-13 所列。

表 17.2-13 MemorySection 列表

| 选 项              | 功 能  |
|------------------|--|
| Default          | 声明数据对象生成的对象时不生成类型限定符或 pragma 命令              |
| MemConst         | 声明数据对象生成的对象时生成 Const 限定符, 仅用于参数数据对象          |
| MemVolatile      | 声明数据对象生成的对象时生成 Volatile 限定符                  |
| MemConstVolatile | 声明数据对象生成的对象时生成 Const Volatile 限定符, 仅用于参数数据对象 |

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

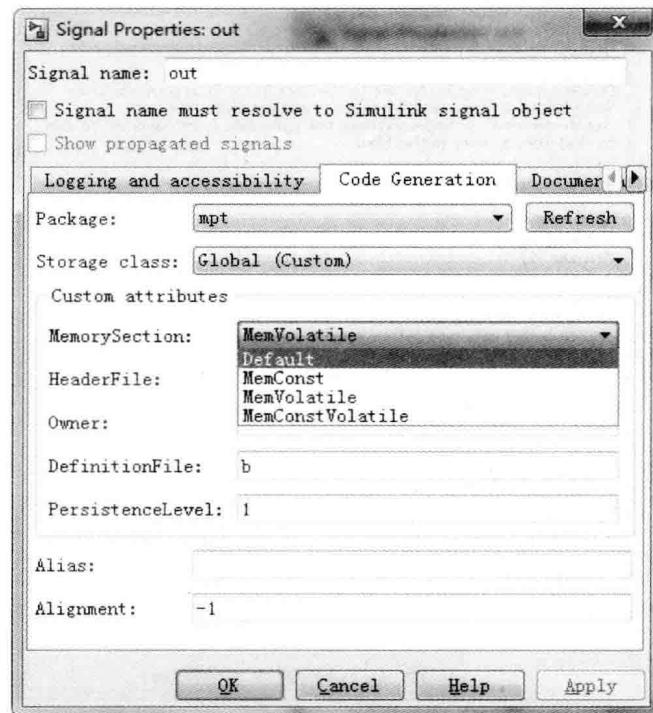


图 17.2-81 选择 mpt 的 Global 存储类型时 out 信号的属性对话框

例如,一个简单的增益模型(由一个 in,一个 gain 和一个 out 直连构成),输入输出信号分别为 in 和 out, Gain 模块的增益变量为 k。3 个变量都使用 mpt 包的数据对象进行定义并绑定到模型中:

```

in = mpt.Signal;
in.InitialValue = '0';
in.StorageClass = 'Custom';
in.CoderInfo.CustomStorageClass = 'Global';
in.CoderInfo.CustomAttributes.MemorySection = 'MemVolatile';

out = mpt.Signal;
out.InitialValue = '0';
out.StorageClass = 'Custom';
out.CoderInfo.CustomStorageClass = 'Global';
out.CoderInfo.CustomAttributes.MemorySection = 'Default';

k = mpt.Parameter;
k.Value = 6;
k.StorageClass = 'Custom';
k.CoderInfo.CustomStorageClass = 'Global';
k.CoderInfo.CustomAttributes.MemorySection = 'MemConstVolatile';

line_h = find_system(gcs, 'findall', 'on', 'Type', 'line');
set(line_h, 'MustResolveToSignalObject', 1);
set_param(gcs, 'InlineParams', 'on');

```

运行上述代码之后的模型启动代码如图 17.2-82 所示。生成的代码紧凑简明。

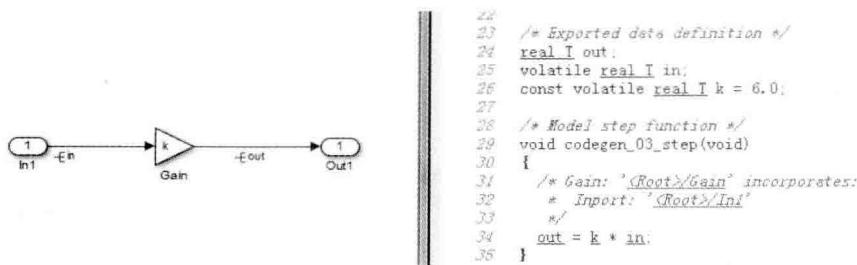


图 17.2-82 Global CSC 作用下的信号和参数代码生成

VolatileStruct 存储类型将在生成的结构体变量前面增加 Volatile 限定符。对于上述增益模型,通过 M 语言创建数据对象绑定到信号线及参数中去:

```

in = mpt.Signal;
in.CoderInfo.StorageClass = 'Custom';
in.CoderInfo.CustomStorageClass = 'StructVolatile';
in.CoderInfo.CustomAttributes.StructName = 'VolaStruc';

out = mpt.Signal;
out.CoderInfo.StorageClass = 'Custom';
out.CoderInfo.CustomStorageClass = 'StructVolatile';
out.CoderInfo.CustomAttributes.StructName = 'VolaStruc';

k = mpt.Parameter;
k.CoderInfo.StorageClass = 'Custom';
k.CoderInfo.CustomStorageClass = 'StructVolatile';
k.CoderInfo.CustomAttributes.StructName = 'VolaStruc';

line_h = find_system(gcs, 'findall','on', 'type','line');
set(line_h, 'MustResolveToSignalObject', 1);
set_param(gcs, 'InlineParams', 'on');

```

再对模型进行编译和代码生成,从生成的代码中可以看到定义结构体变量带有 Volatile 限定符,变量名即为上述代码中 StructName 设置的名称,如图 17.2-83 所示。

数据对象的创建和设定既可以通过信号线属性对话框、Data Store Memory 参数对话框或 M 代码,也可以使用 Model Explorer 创建和管理。启动 Model Explorer 有 3 种方式:

- ① 在 Command Window 中输入 sfxexpr or slexpr 命令。
- ② 在模型中单击工具栏的  按钮。
- ③ 在选中模型后使用组合键 Ctrl+H 启动。

Model Explorer 为 Simulink 模块、信号、工作空间的变量和数据对象提供了一个集中管理和编辑的场所,可以方便地新建一个对象,修改编辑其参数,界面如图 17.2-84 所示。在变量列表中选中一个变量或数据对象,其详细属性通过 GUI 显示到右侧属性栏中。

- ① 通过按钮新建一个变量、信号数据对象或参数数据对象。
- ② Model Hierarchy: 模型层次结构图,从 Simulink Root 开始到 Base Workspace 及各个模型的层次结构列表。选择某一项目,③显示所选项目的子对象或自变量,④显示所选对象的默认属性说明。
- ③ Contents, 显示 Model Hierarchy 中所选项目的内容,即其子成员——普通变量或数据

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

/* Type definition for custom storage class: StructVolatile */
typedef struct VolaStruc_tag {
    real_T in;
    real_T out;
    real_T k;
} VolaStruc_type;

/* Definition for custom storage class: StructVolatile */
volatile VolaStruc_type VolaStruc = {
    /* in */
    0.0,
    /* out */
    0.0,
    /* k */
    0.0
};

/* Model step function */
void untitled_step(void)
{
    /* Gain: '<Root>/Gain' incorporates:
     * Import: '<Root>/In1'
     */
    VolaStruc.out = VolaStruc.k * VolaStruc.in;
}

```

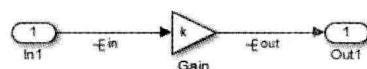


图 17.2-83 StructVolatile CSC 作用下的信号和参数代码生成

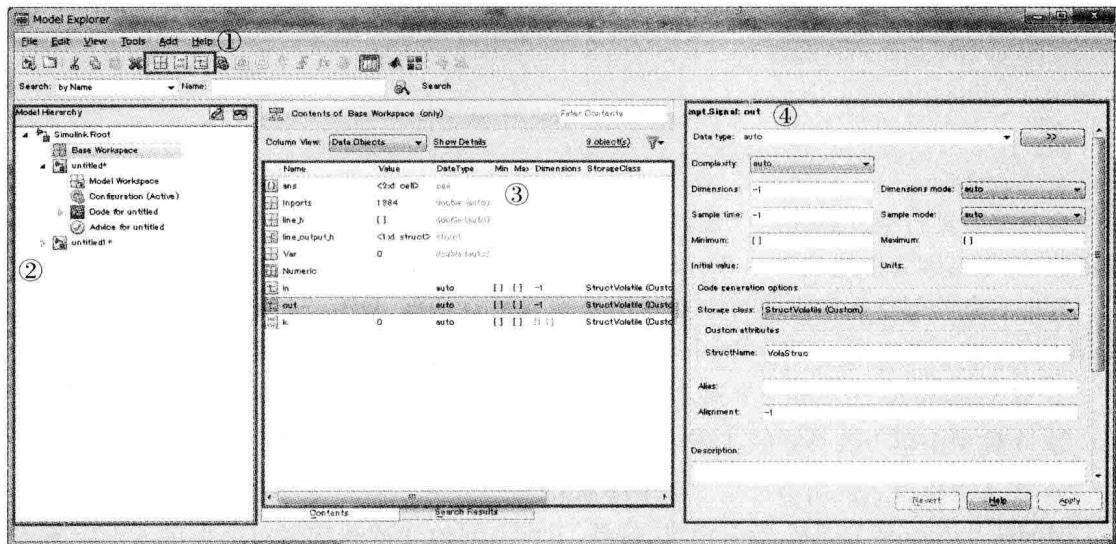


图 17.2-84 Model Explorer 界面

对象或配置页面对象。

④ 将 Contents 中选中项目的属性或内容列表通过内嵌 GUI 对话框的形式提供出来。对于数据对象而言，其 GUI 格局与在工作空间里双击变量弹出的 GUI 基本相同；对于配置项目而言，与独立的 Configuration Parameters 对话框也基本相同；对于代码生成之后的 HTML 报告则直接内嵌到 Model Explorer 的右侧。

下面使用 Model Explorer 来创建增益模型所需要的数据对象：

在 Command Window 中输入 `sflexpr` 按下回车则自动打开 Model Explorer 界面，选中 Model Hierarchy 的 Base Workspace 之后，单击菜单栏中的 Add Simulink Parameter 按钮 [101]，则 Contents 列表中出现了一个名为 Param 的参数，选中其后右侧属性栏显示出 Simulink. Parameter 的属性对话框，如图 17.2-85 所示。将对应属性设置进去即可。

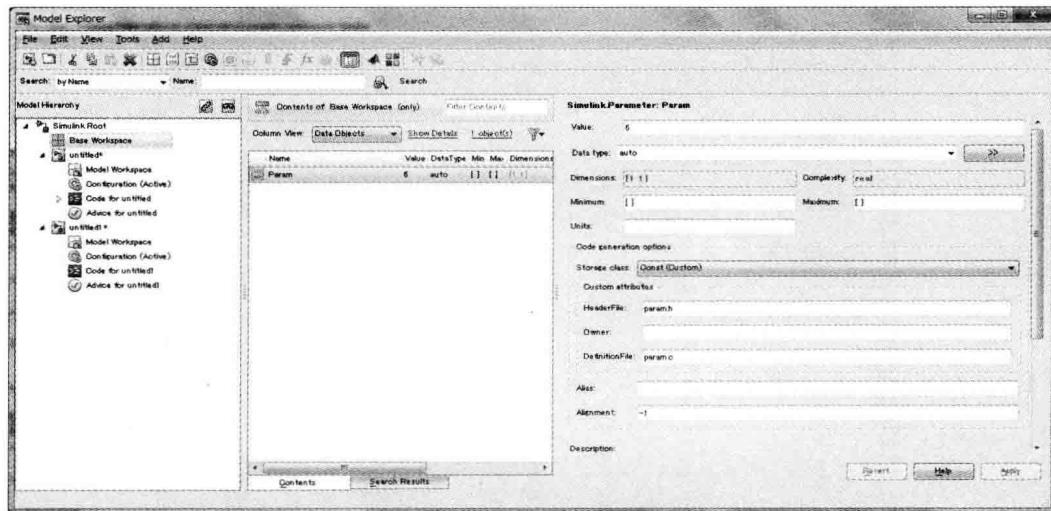


图 17.2-85 Model Explorer 创建数据对象实例

设置好参数对象的属性后，打开 MATLAB 主界面，可以看到工作空间里已经存在这个参数对象了，双击 Workspace 中的变量会弹出其属性对话框，如图 17.2-86 所示。

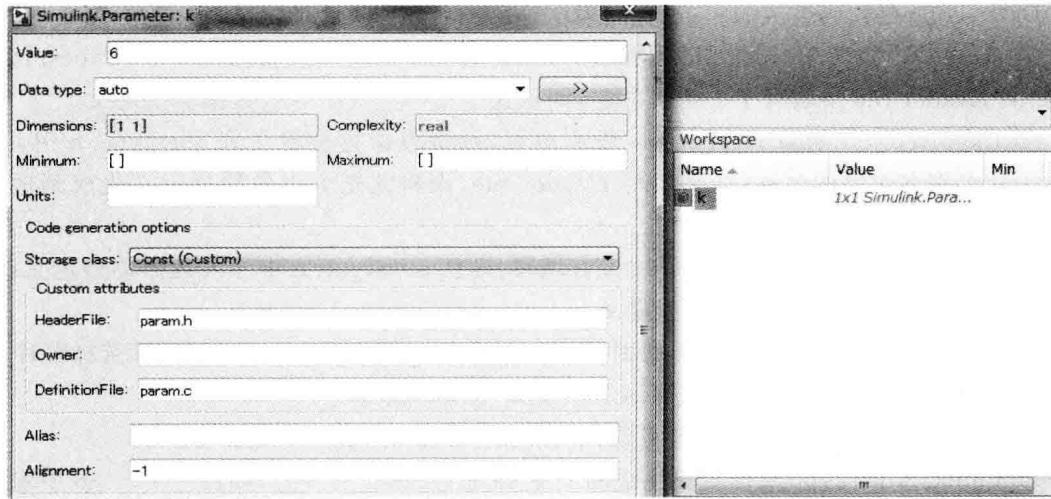


图 17.2-86 Model Explorer 创建的参数对象及属性

## 5. Simulink 其他数据对象

除了 Simulink. Signal 与 Simulink. Parameter 这两个数据对象以外，Simulink 类还有 AliasType、NumericType 和 Bus 等子类。其中 Simulink. Bus 和 Simulink. BusElement 用来配合 Bus Creator 模块创建和管理 Bus 变量对象及其成员。在 2.2.14 小节中已经有过讲解。Sim-

ulink. AliasType 是为一个数据类型定义别名的类; Simulink. NumericType 提供了定义一个数据类型的功能。

首先来看 Simulink. AliasType 的功能以及代码生成时的作用和使用方法。

```
custom_float = Simulink.AliasType;
```

上述语句是在 Base Workspace 中创建一个别名类型对象, 双击 Workspace 中 custom\_float 变量可以弹出别名类型对象的属性对话框, 如图 17.2-87 所示。

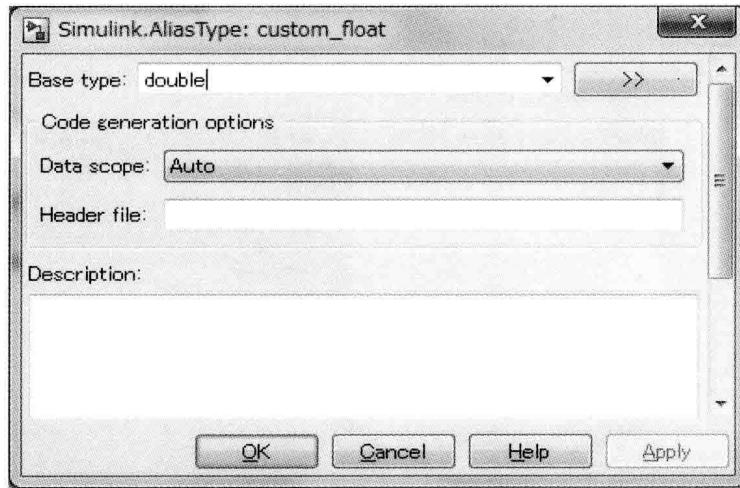


图 17.2-87 Simulink. AliasType 属性对话框

Base type 是别名对象的基础类型, 本质上是 Base type 所表示数据类型的一个别名。Base type 中可以选择 Simulink 所有内建数据类型, 默认是 double。

Data scope 是代码生成的相关属性, 表示这个别名数据类型的定义所存放的位置:

① Auto——如果 Header file 没有填写, 则将数据类型定义生成到 model\_types. h 文件中; 如果 Header file 中填写了一个文件名, 则从这个文件生成中导入数据类型定义。

② Exported——如果 Header file 没有填写, 则将数据类型定义到 aliastype. h 中, aliastype 为这个别名对象的名称; 如果填写 Header file, 则将此别名对象数据定义生成到这个文件中。

③ Imported——如果 Header file 没有填写, 则从 aliastype. h 中导入数据类型定义; 如果填写, 则从指定文件中导入数据类型定义。

Header file 中填写定义数据类型别名的头文件名(带不带有. h 均可), 不管选择哪种 Data Scope, 都会在 model\_types 中生成 #include 这个头文件的命令。

Description 中可以添加此别名对象相关的文本信息, 对模型生成没有影响。

Simulink. AliasType 数据对象生成的 C 代码为 typedef 命令行, 如图 17.2-88 中定义的信号数据对象都将其数据类型设置为 custom\_float, custom\_float 为 double 数据类型的别名, 设置生成代码 Data scope 为 Exported, 定义这些数据对象及绑定到模型的 M 代码为:

```
custom_float = Simulink.AliasType;
custom_float.HeaderFile = 'custom';
custom_float.DataScope = 'Exported';

in = mpt.Signal;
```

```

in.CoderInfo.StorageClass = 'Custom';
in.CoderInfo.CustomStorageClass = 'StructVolatile';
in.CoderInfo.CustomAttributes.StructName = 'VolaStruc';
in.DataType = 'custom_float';

out = mpt.Signal;
out.CoderInfo.StorageClass = 'Custom';
out.CoderInfo.CustomStorageClass = 'StructVolatile';
out.CoderInfo.CustomAttributes.StructName = 'VolaStruc';
out.DataType = 'custom_float';

A = Simulink.Signal;
A.CoderInfo.StorageClass = 'Custom';
A.CoderInfo.CustomStorageClass = 'GetSet';
A.CoderInfo.CustomAttributes.GetFunction = 'DataRead';
A.CoderInfo.CustomAttributes.SetFunction = 'DataWrite';

line_h = find_system(gcs, 'findall','on', 'type','line');
set(line_h, 'MustResolveToSignalObject', 1);
set_param(gcs, 'InlineParams', 'on');

```

启动代码生成之后，在 model\_types 中生成输入/输出的结构体，成员变量数据类型均以 custom\_float 取缔 real\_T 数据类型，定义此数据类型别名的 typedef 语句生成到 custom.h 中，如图 17.2-88 所示。

```

/* Type definition for custom storage class: StructVolatile */
typedef struct VolaStruc_tag {
    custom_float in;                                model_types.h
    custom_float out;
} VolaStruc_type;

#ifndef RTW_HEADER_custom_h_
#define RTW_HEADER_custom_h_
#include "rtwtypes.h"                               custom.h

typedef real_T custom_float;
typedef creal_T ccustom_float;

#endif                                         /* RTW_HEADER_custom_h_ */

```

图 17.2-88 Simulink. AliasType 生成的代码

模型在 `ert.tlc` 下的 Configuration Parameters 中，Code Generation 的子标签 Datatype Replacement 中开启数据类型替换之后，即可使用数据类型别名定义功能。将 Simulink. AliasType 定义过的数据类型别名填入到 Replacement Name 列，模型生成代码时不再使用默认数据类型名，而生成自定义的数据类型别名。Datatype Replacement 页面对话框如图 17.2-89 所示。

这样做的好处是方便代码移植。当模型生成的代码用在不同字长的平台上时，如果不使用数据类型别名的话，必须修改代码中大量变量名声明和定义时的数据类型限定符，因为有些数据类型在不同平台上代表的长度是不同的，如 `int`；反之，如果使用数据类型别名对象的话，不管数据别名内部引用的是什么类型，生成的代码中声明或定义变量都是以这个别名为限定符，移植时只要重新设定一下它所引用的数据类型，也就是 `base type` 即可，方便简洁。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

Replace data type names in the generated code

Data type names

| Simulink Name | Code Generation Name | Replacement Name |
|---------------|----------------------|------------------|
| double        | real_T               | FLOAT64          |
| single        | real32_T             | FLOAT32          |
| int32         | int32_T              | INT32            |
| int16         | int16_T              | INT16            |
| int8          | int8_T               | INT8             |
| uint32        | uint32_T             | UINT32           |
| uint16        | uint16_T             | UINT16           |
| uint8         | uint8_T              | UINT8            |
| boolean       | boolean_T            | BOOL             |
| int           | int_T                | INT32            |
| uint          | uint_T               | UINT32           |
| char          | char_T               | CHAR             |

图 17.2-89 Simulink 代码生成数据类型名替换例

那么用 M 脚本创建这些数据类型别名对象时,若仍然像之前的例子那样一个一个顺次使用 M 脚本创建,均是重复的代码编写,枯燥乏味。那么可以采取适当的手段来优化代码,使代码简短精湛:

```
replaced_type = {'FLOAT64','FLOAT32','INT32','INT16','INT8','UINT32','UINT16','UINT8','BOOL',
'INT32','UINT32','CHAR'};
base_type = {'double','single','int32','int16','int8','uint32','uint16','uint8','boolean',
'int32','uint32','uint8'};
type_num = length(replaced_type);
for ii = 1: type_num
    eval(['replaced_type{ii},' = Simulink.AliasType;']);
    eval(['replaced_type{ii}.BaseType = base_type{ii};']);
    assignin('base', replaced_type{ii}, eval(replaced_type{ii}));
end
```

执行上述代码之后,可以在 Base Workspace 中查看这些定义好的数据类型别名对象,如图 17.2-90 所示。

Workspace

| Name    | Value                  |
|---------|------------------------|
| BOOL    | 1x1 Simulink.AliasType |
| CHAR    | 1x1 Simulink.AliasType |
| FLOAT32 | 1x1 Simulink.AliasType |
| FLOAT64 | 1x1 Simulink.AliasType |
| INT16   | 1x1 Simulink.AliasType |
| INT32   | 1x1 Simulink.AliasType |
| INT8    | 1x1 Simulink.AliasType |
| UINT16  | 1x1 Simulink.AliasType |
| UINT32  | 1x1 Simulink.AliasType |
| UINT8   | 1x1 Simulink.AliasType |

图 17.2-90 通过循环生成的数据类型别名对象

8 行左右的代码就定义了 12 个数据类型别名对象, 比起一个一个定义的代码要简短很多。并且当数据类型别名对象存在于 Base Workspace 中时, 可以直接在模型的信号或参数的 Data type 中使用。在信号的属性对话框中 Signal Attribute 页面下单击 Data Type 下拉按钮, 可见自定义的数据类型别名对象可在 Data Type 中使用, 如图 17.2-91 所示。

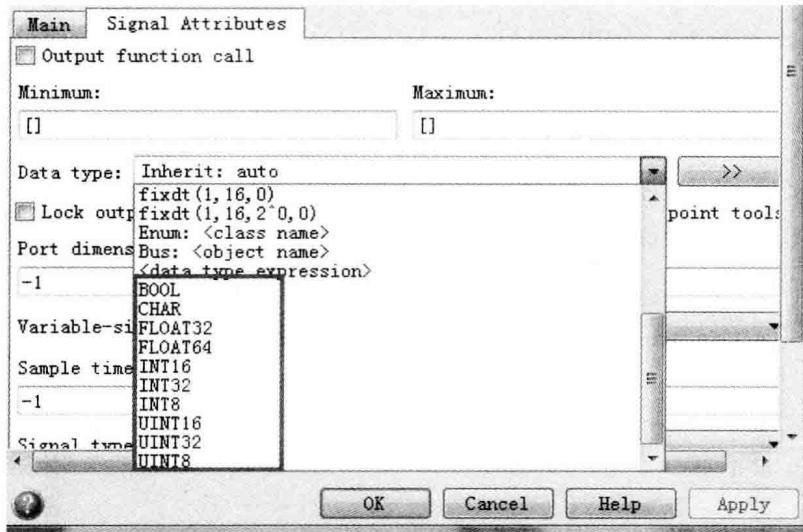


图 17.2-91 自定义的数据类型别名对象可以在 Data type 中使用

开启了 Replacement 功能的这些数据类型别名定义在 DataScope 设为 Auto 时会生成到 rtwtypes.h 中去; 若 DataScope 为 Imported, 则需要用户提供一个定义了这些别名的头文件。不允许设置 DataScope 为 Exported。

Simulink. NumericType 数据对象比 Simulink. AliasType 功能更多一些, 不仅可以创建一个数据别名对象继承既有数据类型, 也可以创建一个全新的自定义数据类型。下面使用 M 语言创建一个 Simulink. NumericType 对象, 定义一个新的数据类型 fixdt32\_8 作为 uint32 类型的别名, 规定它以二进制固定点类型作为数据类型基础模型, 并配置为 32 位无符号整数, 其中低 8 位表示小数部分。生成代码时将此数据类型作为导出类型生成到 custom.h 中。

```
fixdt32_8 = Simulink.NumericType;
fixdt32_8.DataTypeMode = 'Fixed-point; binary point scaling';
fixdt32_8.Signedness = 'Unsigned';
fixdt32_8.WordLength = 32;
fixdt32_8.FractionLength = 8;
fixdt32_8.IsAlias = 1;
fixdt32_8.DataScope = 'Exported';
fixdt32_8.HeaderFile = 'custom.h';
fixdt32_8.Description = 'This data type is created by Simulink.NumericType.';
```

运行上述代码后, 可以在 Base Workspace 中看到变量 fixdt32\_8, 双击打开属性对话框如图 17.2-92 所示。

将这个自定义的数据类型应用到模型中去, 模型中 Data Memory 信号对象的定义及 in 和 out 信号数据对象的定义请参考前面的例子。模型及自定义数据类型生成的代码如图 17.2-93 所示。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

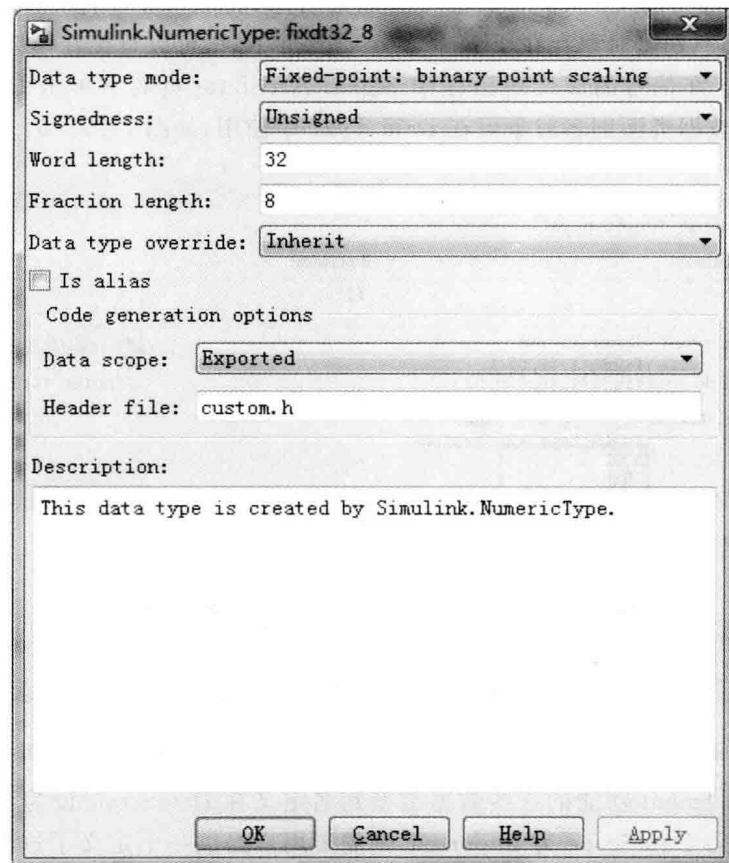


图 17.2-92 Simulink. NumericType 对象属性对话框

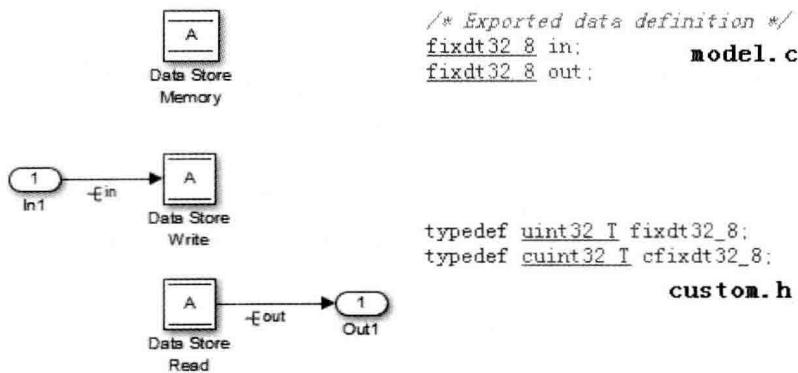


图 17.2-93 Simulink. NumericType 对象在模型中生成的代码

请注意，上述例中是将 Simulink. NumericType 数据对象作为数据类型别名使用的，此时 Is Alias 是选中状态，如果 IsAlias 属性设置为 0，那么生成代码中则直接使用该数据类型的 data type mode 属性引用的类型，本例中设置的无符号 32 位字长的类型，即 uint32\_T 类型。生成代码时不再生成 custom.h 文件。

## 6. 为 Simulink 模型创建数据词典

当模型所包含的 Simulink 包和 mpt 包的数据对象个数较少时, 使用 M 语言或属性对话框及 Model Explorer 来创建数据对象都可以。当一个模型结构复杂, 包含成百上千个信号和参数时, 如果仍然一个一个手动编写代码或配置属性对话框, 工作量之大效率之低下让人难以忍受。对于这种信号量和参数量都比较大的模型, 其数据对象的创建和存储可以通过数据词典(Data Dictionary, 简称 DD)管理。DD 伴随模型存在, 模型作为系统的骨架, DD 作为系统的血脉使得模型能够进行仿真和代码生成。DD 的优势还在于能够帮助在模型的设计者和模型的使用者之间建立一个渠道, 让双方在对设计规格的理解上能够更容易达成一致。

数据词典可以说是 Simulink 的一个数据库, 包含 Signal、Parameter 及其他描述模型行为的数据对象。每个数据对象作为一条条目, 每个条目中包括数据对象的各个属性(名称、最大值、最小值、数据类型、存储类型、单位、分辨率等)。为了模型仿真而设计的输入数据及模型仿真得到的输出数据不属于数据词典的范畴。

根据数据词典中项目条数的多少可以使用 MATLAB 的 Base Workspace、Excel 或其他数据库软件等介质作为建立数据词典的基盘。本书通过 Excel 文档介绍一个简单数据词典的建立和导入方式。

数据词典的内容包括信号量和参数量, 可根据信号/参数数据对象的属性编制 DD 的表头, 内容包括变量名、变量规模(行数、列数)、变量描述、最小值/最大值、初始值、单位、数据类型及 CSC 等。例如一种 Excel 版 DD 如图 17.2-94 所示。

|    | A     | B   | C      | D                       | E     | F       | G             | H    | I        | J                     |
|----|-------|-----|--------|-------------------------|-------|---------|---------------|------|----------|-----------------------|
| 1  | Name  | Row | Column | Description             | Min   | Max     | Initial value | Unit | DataType | CSC                   |
| 2  | Sig1  | 1   |        | 1 This is a demo signal | 0     | 10      | 0 s           |      | double   | ExportedGlobal        |
| 3  | Sig2  | 1   |        | 1 This is a demo signal | -5    | 20      | 1 m           |      | double   | SimulinkGlobal        |
| 4  | Sig3  | 3   |        | 5 This is a demo signal | 5     | 10      | 8 kg          |      | double   | Auto                  |
| 5  | Sig4  | 2   |        | 1 This is a demo signal | -100  | 100     | 0 mg          |      | int8     | ImportedExtern        |
| 6  | Sig5  | 1   |        | 1 This is a demo signal | 0     | 90      | 60 kPa        |      | uint8    | SimulinkGlobal        |
| 7  | Sig6  | 2   |        | 2 This is a demo signal | 0     | 1       | 0             |      | boolean  | ExportedGlobal        |
| 8  | Sig7  | 3   |        | 4 This is a demo signal | 0.1   | 0.6     | 0.5 g         |      | double   | ImportedExternPointer |
| 9  | Sig8  | 5   |        | 2 This is a demo signal | -1000 | 1000    | 0 km          |      | int16    | ExportedGlobal        |
| 10 | Sig9  | 10  |        | 2 This is a demo signal | 0     | 100000  | 0 N*m/s       |      | uint32   | SimulinkGlobal        |
| 11 | Sig10 | 5   |        | 3 This is a demo signal | 0     | 6000000 | 0 kW          |      | uint32   | ImportedExternPointer |

图 17.2-94 数据词典信号量样例

DD 中可以将信号量与参数量分别保存在不同的 Sheet 中, 便于区分和管理。参数量的属性与信号量基本一致, 只不过没有初始值这个属性, 取而代之的是常数值, 如图 17.2-95 所示。

|    | A       | B   | C      | D                          | E     | F       | G       | H    | I        | J                     |
|----|---------|-----|--------|----------------------------|-------|---------|---------|------|----------|-----------------------|
| 1  | Name    | Row | Column | Description                | Min   | Max     | Value   | Unit | DataType | CSC                   |
| 2  | Param1  | 1   |        | 1 This is a demo parameter | 0     | 10      | 0 s     |      | double   | ExportedGlobal        |
| 3  | Param2  | 1   |        | 1 This is a demo parameter | 0     | 1       | 0       |      | boolean  | SimulinkGlobal        |
| 4  | Param3  | 3   |        | 5 This is a demo parameter | 0     | 2000    | 50 kg   |      | double   | Auto                  |
| 5  | Param4  | 2   |        | 1 This is a demo parameter | -100  | 100     | 0 mg    |      | int8     | ImportedExtern        |
| 6  | Param5  | 1   |        | 1 This is a demo parameter | 0     | 90      | 60 kPa  |      | uint8    | SimulinkGlobal        |
| 7  | Param6  | 2   |        | 2 This is a demo parameter | 5     | 10      | 8 m     |      | uint8    | Auto                  |
| 8  | Param7  | 3   |        | 4 This is a demo parameter | 0.1   | 0.6     | 0.5 g   |      | double   | ImportedExternPointer |
| 9  | Param8  | 5   |        | 2 This is a demo parameter | -1000 | 1000    | 0 km    |      | int16    | ExportedGlobal        |
| 10 | Param9  | 10  |        | 2 This is a demo parameter | 0     | 100000  | 0 N*m/s |      | uint32   | ImportedExternPointer |
| 11 | Param10 | 5   |        | 3 This is a demo parameter | -1    | 6000000 | 0 kW    |      | int32    | ExportedGlobal        |

图 17.2-95 数据词典参数量样例

DD 的属性列数可以根据用户的应用场景和实际需要进行增减或合并, 如针对不支持浮点运算的 MCU 时可以增加分辨率属性; 从数据对象种类上细分的话, 甚至可以将 Lookup Table 作为单独一个 sheet 管理在 DD 之中, 因为它既常用又是参数对象的容器, 每一个坐标

轴和查找表都是由参数对象来描述的。

具备了 DD 和模型之后,如何让二者联合起来工作呢? M 语言可以进行 DD 的解析,批处理创建数据对象存放于 Base Workspace 之中并绑定到模型上去。

```
% read dd
[sig_data, sig_str] = xlsread('DataDic.xls',1);
sig_name = sig_str(2:end, 1);
sig_dimensions = sig_data(:, 1:2);
sig_description = sig_str(2:end,4);
sig_min = sig_data(:, 4);
sig_max = sig_data(:, 5);
sig_initval = sig_data(:, 6);
sig_units = sig_str(2:end, 8);
sig_datatypes = sig_str(2:end, 9);
sig_csc = sig_str(2:end, 10);

% get num of signals
sig_num = size(sig_data, 1);

% create signal objects
for ii = 1: sig_num
    eval(['sig_name{ii}', '= Simulink.Signal;']);
    eval(['sig_name{ii}', '.Dimensions = [', num2str(sig_dimensions(ii,:)),'];']);
    eval(['sig_name{ii}', '.Description = "', sig_description{ii}, '"']);
    eval(['sig_name{ii}', '.Min = ', num2str(sig_min(ii)),']);
    eval(['sig_name{ii}', '.Max = ', num2str(sig_max(ii)),']);
    eval(['sig_name{ii}', '.InitialValue = "', num2str(sig_initval(ii)), '"']);
    eval(['sig_name{ii}', '.DocUnits = "', sig_units{ii}, '"']);
    eval(['sig_name{ii}', '.DataType = "', sig_datatypes{ii}, '"']);
    eval(['sig_name{ii}', '.CoderInfo.StorageClass = "', sig_csc{ii}, '"']);
end
line_h = find_system(gcs, 'findall', 'on', 'Type', 'line');
set(line_h, 'MustResolveToObject', 1);
```

运行上述代码之后,DD 中所有的信息都被自动导入 Base Workspace 并创建为数据对象,模型中如果有同名的信号线,将会自动绑定数据对象,如图 17.2-96 所示。

DD 中的参数也同样可完成自动导入。

### 7. Simulink 中的实时任务调度及代码生成

应用于嵌入式代码生成的 Simulink 模型在ert.tlc 系统目标文件的作用下,特别是在没有 OS 的应用场合下,如何能够满足嵌入式 MCU 执行任务的实时性和周期性? 这就要靠ert.tlc 下生成的rt\_OneStep()函数与MCU的定时器中断联合作用来实现了。

rt\_OneStep()函数是ert.main.c文件中实现的一个函数,它是将模型中的算法逻辑执行一个采样步长计算的函数,model\_step()函数被其调用。为了适应嵌入式系统或应用的特点,同时兼顾 Simulink 模型中多个子系统所描述的任务可以以不同的周期来执行,rt\_OneStep()

| Workspace |                     |
|-----------|---------------------|
| Name      | Value               |
| Sig1      | 1x1 Simulink.Signal |
| Sig10     | 1x1 Simulink.Signal |
| Sig2      | 1x1 Simulink.Signal |
| Sig3      | 1x1 Simulink.Signal |
| Sig4      | 1x1 Simulink.Signal |
| Sig5      | 1x1 Simulink.Signal |
| Sig6      | 1x1 Simulink.Signal |
| Sig7      | 1x1 Simulink.Signal |
| Sig8      | 1x1 Simulink.Signal |
| Sig9      | 1x1 Simulink.Signal |

图 17.2-96 DD 被自动转换为数据对象

函数将根据模型本身子系统执行速率的不同采取不同的代码生成策略。

17.2.1 小节“代码生成时的模型配置方法”中提到,为了生成嵌入式 C 代码的模型,其固定点解算器的步长 step0 作为系统的基采样率,每个子系统或模块的采样时间必须是 step0 的正整数倍。这时由于作为运算核心驱动的 solver 都采样不到的时间点,是无法进行解算的。当模型中所有模块都按照同一个采样时间设置时,称为单频率系统,当存在多个不同的采样时间时,称为多频率系统。在单频率系统中,只需要 rt\_OneStep() 函数作为 MCU 内核或外设中任意定时器的中断服务函数即可周期执行,为了与模型仿真的运行周期一致,需要对解算器的步长进行换算,根据 MCU 晶振频率和所使用的定时器的分频系统共同计算出计数值,以使这个定时器的中断周期与模型解算器的步长一致起来。在多频率系统中,上述操作也是必要的,但是更为复杂的是模型中的每一个频率需判断:当基频的采样时间到来之时,是否应该调用采样时间为基频倍数的各个子频的函数。

除了频率的个数之外,一个模型中还可以存在任务个数的区别,分为单任务系统与多任务系统。所谓任务,在模型中由不同的子系统来表现。Simulink 中存在完成不同功能的子系统的模型称为多任务系统,仅存在一个子系统即单任务系统。

当执行速率与任务个数结合起来考虑时,Simulink 模型可以划分为 3 种类型:单速率单任务系统、多速率单任务系统及多速率多任务系统。之所以没有单速率多任务类型是因为 Simulink 的单速率解算器不支持多任务,如表 17.2-14 所列。

表 17.2-14 Simulink 对速率与任务的支持情况

| 模 式 | 单速率 | 多速率 |
|-----|-----|-----|
| 单任务 | 支持  | 支持  |
| 多任务 | 不支持 | 支持  |

### (1) 单速率单任务系统调度代码生成

首先来看单速率单任务的调度代码,这种情况下 rt\_OneStep() 将在每一个采样时刻(同时也是定时器定时周期)调用 model\_step() 函数一次。model\_step() 内包含了模型中唯一的任务所生成的代码。rt\_OneStep() 的伪代码如下:

```
rt_OneStep()
{
    // disable interrupt
    //Check for interrupt overrun or other error
    if(OverrunFlag) {
        return;
    }
    OverrunFlag = TRUE;
    //Enable "rt_OneStep"(timer) interrupt
    model_step() -- Time step combines output,logging,update
    OverrunFlag = FALSE;
}
```

当 rt\_OneStep() 被 MCU 的定时器中断调用时,已进入此函数,首先要关闭中断,并检查定时器是否存在溢出或其他错误情况。如果存在错误,则立即返回;如一切正常,则说明溢出标志为假,在内部将其设定为真并开启定时器中断,然后再调用子系统生成的任务代码 model\_step()。当 model\_step() 正确正常地执行完毕之后,再次将溢出标志清空。在开启了中断之

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

后,执行 model\_step()函数的过程中,如果此中断再次产生,则被 rt\_OneStep()函数视为溢出错误,因为此段时间内溢出标志始终为真。如果子系统的内部模块数量过多,生成的 model\_step()代码的执行时间超过了 MCU 定时器的定时间隔,就有可能造成实际 MCU 运行时的周期与仿真周期不一致。这时,需要读者根据情况修改模型或者修改 rt\_OneStep 中断溢出检测的策略。

## (2) 多速率多任务系统调度代码生成

在多速率多任务系统中,生成的代码采用一种按优先级分类的可抢占式的机制。模型中假设包括了 NumTask 个任务,模型的固定解算器步长相同周期的任务为基速率,其他任务的执行周期都是基速率的整数倍,称为子速率。基速率的子系统生成的代码为 model\_step0,其余子速率的任务按照执行周期的快慢生成的代码为 model\_step1~model\_stepNumTask-1,统称为 model\_stepN,N = 0~NumTask-1. 当模型中具备 4 个不同的速率时,每个速率的任务模型生成代码在 model.c 中声明为:

```
void my_model_step0(void);
void my_model_step1(void);
void my_model_step2(void);
void my_model_step3(void);
```

在 NumTask 个任务中,基速率的任务是周期最短,执行频率最高的一个,同时也具有最高的优先级;其他子任务则按照执行周期从短到长依次具有逐渐降低的优先度。即,Task NumTask-1 周期时间最长,具有最低的优先级。

多速率多任务模型中生成的 rt\_OneStep()函数的伪代码如下:

```
rt_OneStep()
{
    static BOOL OverrunFlags[NumTask] ; /* Model has NumTask rates */
    static BOOL eventFlags[NumTask];
    static INT32 taskCounter[NumTask];
    // Disable "rt_OneStep" interrupt
    //Check for base - rate interrupt overrun
    //Enable "rt_OneStep" interrupt
    //Determine which rates need to run this time step
    model_Step0()    // run base - rate time step code
    For N = 1:NumTasks - 1    // iterate over sub - rate tasks
        If(sub - rate task N is scheduled)
            Check for sub - rate interrupt overrun
            model_StepN()// run sub - rate time step code
        EndIf
    EndFor
}
```

rt\_OneStep()中拥有同单速率单任务系统相同的定时器溢出检测,为每一个任务都设置一个溢出标志。除此之外,在每个 MCU 定时器中断时刻,对每个子速率系统是否需要被执行进行检测,这些标志位存储在 eventFlags 数组中,数组中每个成员表示一个任务的执行标志,只有 eventFlags[n] 为真时,才会在当前 rt\_OneStep() 中执行 model\_stepn。是否将 eventFlags 中的数组元素置位,取决于数组 taskCounter,其元素表示各个任务的计数值。taskCounter 在每次 rt\_OneStep() 被调用时对各个子速率对应的计数值 +1,当达到该子速率周期/基速率周期值时 taskCounter 被清零,接着 eventFlags 的对应位变为 1,从而形成了 rt\_OneStep() 调

用对应的 model\_stepn() 函数的条件。

以一个双速率双任务的模型为例说明此种方式下任务调度的代码生成及原理, 图 17.2-97 所示为一个多速率模型。

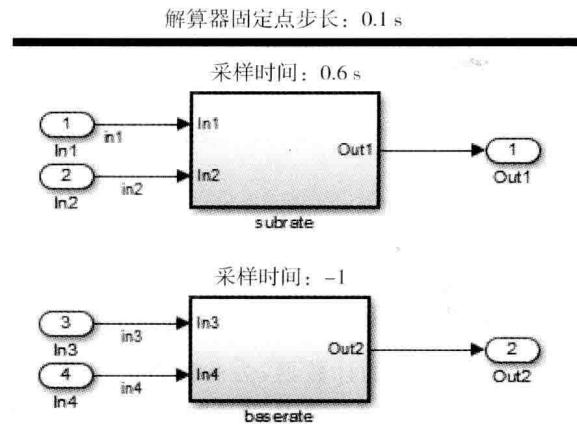


图 17.2-97 双速率双任务模型

图中模型解算器步长为 0.1 s, 两个原子子系统的采样时间分别设置为 0.6 s 和 -1。后者继承模型解算器采样时间 0.1 s。故下面的子系统是基速率任务, 上面的子系统是子速率任务, 执行周期为基速率的 6 倍。ert.tlc 作用下生成的 rt\_OneStep() 函数内容: 检测定时器中断溢出标志位数组 OverrunFlags、子系统对应的代码是否执行标志位数组 eventFlags、基频任务计数器数组 taskCounter, 其生成的代码如图 17.2-98 所示。

```
void rt_OneStep(void)
{
    static BOOL OverrunFlags[2] = { 0, 0 };

    static BOOL eventFlags[2] = { 0, 0 }; /* Model has 2 rates */

    static INT32 taskCounter[2] = { 0, 0 };
}
```

图 17.2-98 标志位及计数器数组定义

这些数组都定义为静态类型, 只需要在首次进入 rt\_OneStep() 时赋初值, 之后在逻辑中更新其值并保存。基速率任务在 rt\_OneStep() 中每次都会执行, 不需要对其进行计数或判断。OverrunFlags[0] 是否为 TRUE 标志着基速率是否出现溢出, 即是否正在执行中, 检测代码如图 17.2-99 所示, 直接判断是否有中断溢出之类的错误情况。

```
/* Check base rate for overrun */
if (OverrunFlags[0]) {
    rtmSetErrorStatus(codegen_04_M, "Overrun");
    return;
}

OverrunFlags[0] = TRUE;
```

图 17.2-99 检测基速率是否有溢出

对于子速率, 并非每次 rt\_OneStep() 内都会执行其对应代码, 需要先判断是否执行, 再进行中断溢出等错误检测。如果没有溢出, 将溢出检测标志位设为 TRUE, 表示接下来要进行 model\_

step0()函数的调用,调用结束之后再将 OverrunFlags[0]赋值为 False。代码如图 17.2-100 所示。

```

if (taskCounter[1] == 0) {
    if (eventFlags[1]) {
        OverrunFlags[0] = FALSE;
        OverrunFlags[1] = TRUE;

        /* Sampling too fast */
        rtmSetErrorStatus(codegen_04_M, "Overrun");
        return;
    }

    eventFlags[1] = TRUE;
}

```

图 17.2-100 检测子速率是否有溢出

taskCounter[1]及模型子速率对应的计数器,为 0 时有 2 种情况,一个是模型在刚运行时初始值为 0,此时 eventFlags[1]也是 0;另一个是当计数值达到 6 时清零,这时 eventFlags[1]为 TRUE。如果是前者,模型刚刚初始化,子速率任务需要执行但尚未执行,将 eventFlags[1]置 1。对于后者,说明子速率对应代码正在执行中且尚未执行完毕,此时再次进行检测,必然是溢出。基频任务计数器达到子速率/基速率执行周期的值时清零的代码如图 17.2-101 所示。

```

taskCounter[1]++;
if (taskCounter[1] == 6) {
    taskCounter[1] = 0;
}

```

图 17.2-101 子速率的计数操作

对于子速率来说,其计数值 taskCounter[1]每运行 rt\_OneStep()一次增加 1,当达到 6 时(子速率执行周期除以基速率执行周期的倍数)说明当前这次中断应调用子速率的代码了,故将对应计数值清零。基速率代码函数每次都被调用,如图 17.2-102 所示。

```

/* Step the model for base rate */
codegen_04_step0();

/* Get model outputs here */

/* Indicate task for base rate complete */
OverrunFlags[0] = FALSE;

```

图 17.2-102 基速率的代码被调用

基速率对应的函数 model\_Step0 被调用,并在调用结束之后设置溢出标志位为假。接下来处理子速率的任务函数调用。由于优先级的存在,当子速率的代码仍在执行中时出现中断溢出,直接返回,那么比它优先级低的子速率代码由于出现在其后,便没有机会执行,其调度就被无视了。检测溢出的代码如图 17.2-103 所示。

```

/* If task 1 is running, don't run any lower priority task */
if (OverrunFlags[1]) {
    return;
}

```

图 17.2-103 溢出返回则屏蔽了低优先级的调度

rt\_OneStep() 函数的最后就是对子速率的判断, eventFlags[1] 为真, 就将子速率的中断溢出标志置位, 然后调用 model\_step1() 函数, 执行完毕之后再清除中断溢出标志位及子速率执行标志位。调用子速率函数的代码如图 17.2-104 所示。

```

/* Step the model for substrate */
if (eventFlags[1]) {
    OverrunFlags[1] = TRUE;

    /* Set model inputs associated with substrates here */

    /* Step the model for substrate 1 */
    codegen_04_step1();

    /* Get model outputs here */

    /* Indicate task complete for substrate */
    OverrunFlags[1] = FALSE;
    eventFlags[1] = FALSE;
}

```

图 17.2-104 调度子速率任务

如果模型中不同任务的不同速率数目达到 3 个以上(即存在 2 个以上的子速率), 根据 eventFlags 判断是否执行子速率的代码则以 switch case 语句方式生成, 如在上例模型中增加 0.8 s 的原子子系统, 整个模型中则具有 3 个不同速率, 其中子速率 2 个, 生成的代码中判断是否执行子速率函数的代码如图 17.2-105 所示。

```

/* Step the model for any substrate */
for (i = 1; i < 3; i++) {
    /* If task "i" is running, don't run any lower priority task */
    if (OverrunFlags[i]) {
        return;
    }

    if (eventFlags[i]) {
        OverrunFlags[i] = TRUE;

        /* Set model inputs associated with substrates here */

        /* Step the model for substrate "i" */
        switch (i) {
            case 1 :
                codegen_04_step1();

                /* Get model outputs here */
                break;

            case 2 :
                codegen_04_step2();

                /* Get model outputs here */
                break;

            default :
                break;
        }

        /* Indicate task complete for sample time "i" */
        OverrunFlags[i] = FALSE;
        eventFlags[i] = FALSE;
    }
}

```

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

图 17.2-105 调度多个子速率任务的代码

### (3) 多速率单任务系统调度代码生成

多速率单任务系统实际上就是指系统具有多个不同的采样速率,每个采样速率都是模型解算器步长的整数倍。由于任务数目是单个,所以不用对 model\_step() 函数进行编号,生成的 rt\_OneStep() 函数仅一个基速率的 model\_step() 函数在多个速率下被调用即可。

那么如何构建一个多速率单任务的模型呢?原子子系统是一个非虚拟子系统,不能具有多个采样时间,必须是单速率的。但是连接原子子系统的外部输入端口 in 模块可以具有不同的采样速率,如图 17.2-106 中虚线框起的部分。

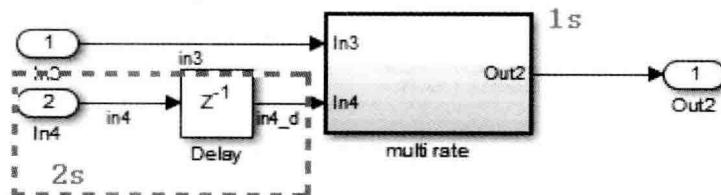


图 17.2-106 多速率单任务模型

其中 In3 的采样时间是 1 s,而 In4 的采样时间是 2 s,二者的采样时间参数设置如图 17.2-107 所示。

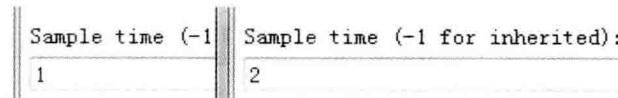


图 17.2-107 输入端口的采样时间

为了在模型中省去速率转换模块,需要在模型的 Configuration Parameter 里设置模型的周期采样时间约束,包括任务模式和采样时间。任务模式设置为 SingleTasking,采样时间属性设置为 [[1,0,0];[2,0,1];],每个采样时间由一个三元向量表示,格式为 [period, offset, priority]。前两个参数表示离散采样时间的周期和偏移量,最后一个参数是子速率的优先级级别,不同速率之间通过优先级区分,数字小的优先级高,设置如图 17.2-108 所示。

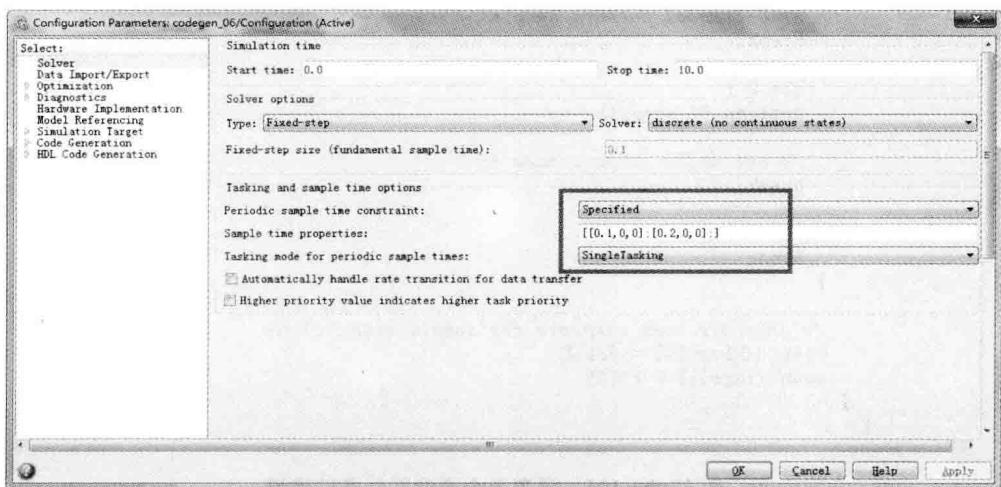


图 17.2-108 设置周期采样时间约束

模型生成的代码中 `rt_OneStep()` 函数的代码同单速率单任务代码相似, 如图 17.2-109 所示。

```

void rt_OneStep(void)
{
    static BOOL OverrunFlag = FALSE;

    /* Disable interrupts here */

    /* Check for overrun */
    if (OverrunFlag) {
        rtmSetErrorStatus(codegen_06_M, "Overrun");
        return;
    }

    OverrunFlag = TRUE;

    /* Save FPU context here (if necessary) */
    /* Re-enable timer or interrupt here */
    /* Set model inputs here */

    /* Step the model for base rate */
    codegen_06_step();

    /* Get model outputs here */

    /* Indicate task complete */
    OverrunFlag = FALSE;

    /* Disable interrupts here */
    /* Restore FPU context here (if necessary) */
    /* Enable interrupts here */
}

```

图 17.2-109 多速率单任务的 `rt_OneStep` 代码

区别在于 `model_step()` 函数的内部, 不同速率的调度代码在内部生成。`model.h` 中会定义一个调度计数器的数据结构如图 17.2-110 所示。

```

struct tag_RTM_codegen_06_T {
    const CHAR * volatile errorStatus;

    /*
     * Timing:
     * The following substructure contains information regarding
     * the timing information for the model.
     */
    struct {
        struct {
            UINT8 TID[2];
        } TaskCounters;
    } Timing;
};

```

图 17.2-110 多速率单任务的 TID 计数数据结构代码

代码通过 `Timing`. `TaskCounter`. `TID` 数组对多个子速率分别进行计数, 当计数达到阈值后被清零并执行对应的子速率代码。这里 `model_step` 函数中仅仅对 `TaskCounters` 中的 `TID` 进行判断, 当其值为 0 时则执行相应的代码, 如图 17.2-111 所示。

由于此模型只有 2 个速率, 基速率每次执行 `rt_OneStep` 时都应该执行, 故不用计时及判断, 即不使用 `TID[0]`。`TID[1]` 对采样周期为 2 s 的子速率进行计数。`model_multirate()` 函数是原子子系统生成的函数, 每 1 s 都进行计算, 而 `Delay` 模型与 `In4` 相关, 继承了其采样时间, 是 2 s 计算一次, 所以 `delay` 到 `in4_d`, `in4` 到 `delay` 的数值传递都是 2 s 更新一次。通过 `TID`

```

void codegen_06_step(void)
{
    if (codegen_06_M->Timing.TaskCounters.TID[1] == 0) {
        /* Delay: '<Root>/Delay' */
        codegen_06_E.ind_d = codegen_06_DW.delay; run per 2s
    }

    /* Outputs for Atomic SubSystem: '<Root>/multirate' */

    /* Inport: '<Root>/In3' */ run per 1s
    codegen_06_multirate(codegen_06_U.in3, codegen_06_E.ind_d);

    /* End of Outputs for SubSystem: '<Root>/multirate' */
    if (codegen_06_M->Timing.TaskCounters.TID[1] == 0) {
        /* Update for Delay: '<Root>/Delay' incorporates:
         * Update for Inport: '<Root>/In4'
         */
        codegen_06_DW.delay = codegen_06_U.in4;
        run per 2s
    }

    rate_scheduler();
}

```

图 17.2-111 model\_step 函数代码

[1]是否被清 0 来判断是否执行相应的子速率代码。TID 数组的计数在 rate\_scheduler() 函数中实现,如图 17.2-112 所示。

```

static void rate_scheduler(void)
{
    /* Compute which subrates run during the next base time step. Subrates
     * are an integer multiple of the base rate counter. Therefore, the subtask
     * counter is reset when it reaches its limit (zero means run).
     */
    (codegen_06_M->Timing.TaskCounters.TID[1])++;
    if ((codegen_06_M->Timing.TaskCounters.TID[1]) > 1) /* Sample time: [2, 05,
        codegen_06_M->Timing.TaskCounters.TID[1] = 0;
    }
}

```

图 17.2-112 rate\_scheduler 函数代码

rate\_scheduler() 函数作为调度控制器,每次 rt\_OneStep 中都被调用,并对 TaskCounter.TID[1]增加 1。TaskCounter.TID[1]初值为 0,当其值大于阈值 1(阈值:该采样周期为基速率采样周期的倍数减一时,就将其 TID 位清零。model\_step 根据 TID 是否为 0 来决定其对应的代码是否在当前这次 rt\_OneStep() 的调用中执行。

Simulink 调度器代码分析如上所述。总结下来,就是依靠基频计数判断子频率是否执行、执行未完成时不可再重入,以保证代码执行顺序/时序跟模型的顺序/时序一致起来。

# 第 18 章

## TLC 语言

**引言:**Simulink 模型在 Simulink Coder 和 Embedded Coder 的支持下可以生成嵌入式 C 代码,应用于 MCU、DSP 等芯片。模型生成代码需要靠系统目标文件与模块目标文件的支持。这两个等级的目标文件都是由 TLC(Target Language Compiler, 目标语言编译器)进行语言转换的 TLC 文件。TLC 语言作为代码生成流程中的重要环节,拥有目标语言转换的能力。本章通过原理和实例说明其语法、组织结构方法、所支持的数据类型、访问范围、编写方法和执行及调试方法。并介绍如何使用 TLC 进行文件流控制,如何访问和添加 rtw 文件中的记录,以及如何为 S 函数提供代码生成用的模块级 TLC 文件等。

TLC 语言如同其名 Target Language Compiler 是一种为转换为目标语言而存在的解释性语言,其目的就是将模型编译出来的 rtw 文件转换为目标代码(C/C++等)。与 M 语言类似,既可以写成脚本文件,也能够作为函数存在,都是解释性语言,更相似的是它们都提供具有强大功能的内建函数库。在“代码生成流程”一节中已经介绍了其访问 rtw 文件或修改其内容的方法,在之后的 17.2.2 节内容中将更加频繁地使用其更加丰富多彩的功能,在代码生成及工具链定制过程中起到至关重要的作用,所以将 TLC 语言作为单独的章节,充分讲解其语法、编写方法和调试方法等。

### 18.1 TLC 的作用

TLC 语言最根本的作用就是将模型编译出的 rtw 文件转化为支持某种平台或硬件的代码。Simulink 提供的模块中之所以有一些支持代码生成,也是因为 MathWorks 已经为这些模块编写好了 TLC 文件,用户已经潜移默化地使用它们建模并生成代码。但是在以 MATLAB 为工具进行基于模型设计开发的今天,很多用户不满足于 Simulink 提供的模块,希望既能够定制自己的模块(如算法模块、硬件驱动模块),又能够定制一些自动化工具链将开发流程自动进行,将这两件事结合,起到事半功倍的效果。在这种情况下,TLC 逐渐显示出其强大作用:

- ① 支持模型针对通用或特定目标硬件的代码生成功能。
- ② 为 S 函数模块提供代码生成功能,可以让用户自己增加支持代码生成的模块。
- ③ 在代码生成过程中,生成不依赖 S 函数模块的自定义过程代码。

Simulink 中提供了很多既有 TLC 文件,如果擅自修改,可能导致 Simulink Coder 功能性错误,所以 MathWorks 提倡用户尽量不要涉及 TLC。可是话说回来,在需求多样化的现在,如果能熟练掌握 TLC 的运行机制和编写方法,不仅不会伤害 Simulink Coder 的功能,还能够巧妙地利用 TLC 语言实现更多的自动化代码生成功能。下面跟着笔者一起看 TLC 是怎样炼成的。

### 18.2 TLC 的语法

TLC 是一种以单个%打头的关键字为命令,空格之后跟参数的脚本语言,自身包含了流

控制语法、内建函数、关键字和常用命令,支持脚本和函数两种编程方式。其语言风格与 M 语言很接近。

## 18.2.1 基本语法

TLC 语言的基本语法包括以下 2 种:

(1) [text | %<expression>]\*

text 表示字符串,将原原本本地展开到输出流中。在%<>之中的是 TLC 变量,通过%<>作用将变量的执行结果显示到输出流中。

(2) %keyword [argument1, argument2, ...]

%Keyword 表示 TLC 语言中的命令符,[argument1, argument2, ...]中则表示这个命令符所操作的参数。如:

```
% assign Str = "Hello World"
```

%assign 表示其后的语句是对变量赋值,上句对变量 Str 赋值“Hello World”字符串。%assign 指令既可以创建一个新变量,也可以改变既存变量的值,如对上述变量更改其所表示的字符串值。

```
% assign Str = "I Love MATLAB"
```

又如%warning,%error 和%trace 命令,可以将其后所跟变量或字符串的内容输出。如:

```
% warning Simulink User
```

将上句保存在文本文件中,命名为 text.tlc,保存在 MATLAB 的当前路径下或搜索路径中,在 Command Window 中输入 tlc text.tlc 即可运行此 TLC 文件。执行结果为:

```
tlc text.tlc
Warning: Simulink User
```

tlc xxxx.tlc 是运行脚本 TLC 文件的方法,脚本 TLC 文件是指可以直接运行的 TLC 文件。

%error 命令和%warning 命令使用方法基本相同,只是在输出内容前自动增加 error,并非 warning 的字样,并且会给出内置 TLC 报错信息。如图 18.2-1 所示。

```
Error: Type is Vector.
Main program:
==> [00] text.tlc:<NONE>(10)

Error using tlc_new
Error: Errors occurred - aborting

Error in tlc (line 85)
    tlc_new(varargin{:});
```

图 18.2-1 %error 命令的报错

如果使用%trace 命令代替%warning 命令显示信息,在执行 TLC 文件时需要在最后增加-v 或-vl 才能够将%trace 的命令给显示出来。追加与不追加-v 的区别如图 18.2-2 所示。

为命令输入方便,之后的信息输出都使用%warning 来实现。

TLC 语言有两个内建宏 TLC\_TRUE=1 和 TLC\_FALSE = 0,在 TLC 语言的编写中会经常用到。

```
>> tlc text.tlc -v
Warning: File: text.tlc Line: 1 Column: 7
%trace directive: Simulink User
>> tlc text.tlc -v1
Warning: File: text.tlc Line: 1 Column: 7
%trace directive: Simulink User
>> tlc text.tlc
>>
```

图 18.2-2 trace 命令是否生效

## 18.2.2 常用指令

在掌握了基本语法之后,可了解常用指令及其应用,常用指令掌握之后基本上能够了解TLC语言全貌。

### 1. 注释

TLC语句可以通过双百分号进行单行注释, % % comment:

```
% % trace Simulink User
```

可以通过/% comment %/来进行块注释,块中内容可以是单行或多行:

```
/%
% include "test.tlc"
% assign Str = "Hello World"
%
% assign Str = "I Love MATLAB"
% trace Simulink User
```

### 2. 变量内容扩展

所谓变量扩展即将 TLC 变量通过%<>操作符将其内容扩展到输出流中。如两个数值类型作加法并将其输出:

```
% assign input1 = 3
% assign input2 = 5
% warning %<input1> + %<input2> = %<input1 + input2>
```

上述脚本文件执行后%<>中的变量及变量的计算结果都将自动扩展为结果值:

```
Warning: 3 + 5 = 8
```

**注意:** %<>不能嵌套使用,否则会报错:Expansion directives %<> cannot be nested。

### 3. 条件分支

TLC语言中条件分支的格式如下:

```
% if expression
% elseif expression
% else
% endif
```

与 M 语言很相近的是每个 if else 一定要使用 end 来结束。expression 的执行结果一定要是一个整数。例如,判断一个变量是否为 1,如果不为 1 则输出警告信息:

```
% if ISEQUAL(var, 1)
    % warning everything is OK.
% else
    % warning var should be 1 but now there's something wrong.
% endif
```

若您对此书内容有任何疑问,可以凭在线交流卡登录MATLAB中文论坛与作者交流。

每个%if 仅对应一个%else,如果出现多个%else 则会报错。ISEQUAL(expr1, expr2)是TLC语言的一个内建函数,用来判断变量是否相等。expr1 和 expr2 两个参数可以是数值型数据,也可以是字符串(string)或者记录(record)。并且在ISEQUAL()函数中不需要使用%<>,通过变量名即扩展为变量的值。

#### 4. 开关分支

TLC 语言中的 switch case 语句使用如下的格式:

```
% switch expression
% case expression
% break
% default
% break
% endswitch
```

expression 必须是一个能够使用==操作符来比较是否相等的数据类型。每个%case 分句后如果不跟%break 语句的话,%switch 语句将直接执行下一个%case 之后的语句。开关语句使用如下:

```
% assign data = [1,2,3,4,5]
% switch TYPE(data)
% case "Number"
% warning Type is Number.
% break
% case "String"
% warning Type is String.
% break
% case "Vector"
% warning Type is Vector.
% break
% case "Matrix"
% warning Type is Matrix.
% break
% endswitch
```

运行上述 tlc 文件的执行结果是:

```
Warning: Type is Vector.
```

如果将上述的%break 全部删除之后,再度运行则显示:

```
Warning: Type is Vector.
```

```
Warning: Type is Matrix.
```

说明从第 3 个 case 语句开启之后,由于没有%break 的存在,第 4 个%case 语句不做判断直接执行第 4 个%case 之后的语句。建议读者养成好习惯,为每个%case 语句配上一个%break 语句。

#### 5. 循环

相对于 M 语言中的 for 循环,TLC 语言提供了多种循环控制方式,包括%foreach, %roll 及%for 3 种常用方式。三者既有相似之处也有区别。

##### (1) %foreach

%foreach 命令实现循环的格式如下:

```
% foreach loopIdx = iterNum
xxxxx
% endforeach
```

上述语句将 loopIdx 作为循环体句柄变量控制循环进行,从 0 开始,每次增加 1,一直循环到 iterNum-1 为止。每个%foreach 需要使用%endforeach 来终止。例如使用循环打印一个数组中的值:

```
% assign data = [1,2,3,4,5]
% foreach idx = 5
    % warning data[ %<idx> ] = %<data[ idx]>
% endforeach
```

运行结果:

```
Warning: data[0] = 1
Warning: data[1] = 2
Warning: data[2] = 3
Warning: data[3] = 4
Warning: data[4] = 5
```

在循环体中可以使用%continue 终止当前循环,从下一个循环在开始;或者使用%break 直接跳出循环。如在循环体中输出 warning 信息之前加入条件判断,如果当前循环 idx 变量为 1 则直接开始下一次循环:

```
% assign data = [1,2,3,4,5]
% foreach idx = 5
    % if ISEQUAL(idx,1)
        % continue
    % endif
    % warning data[ %<idx> ] = %<data[ idx]>
% endforeach
```

执行结果为:

```
Warning: data[0] = 1
Warning: data[2] = 3
Warning: data[3] = 4
Warning: data[4] = 5
```

将上述代码中的%continue 换为%break,执行结果为:

```
Warning: data[0] = 1
```

在循环变量变为 1 之后,遇到%break 则立刻跳出循环体。

## (2) %roll

TLC 中提供%roll 这种循环方式主要是为 Simulink 模块端口信号相关代码生成定义时使用的。当模块的输入/输出信号维数是多维时,需要通过%roll 对信号的每一维进行循环,使其生成的代码同在 Simulink 环境中进行仿真时具有相同的维数。其格式如下:

```
% roll
% endroll
```

比如  $y=2 \times u$  这样一个表示输出为输入乘以 2 的模块,为了使其支持代码功能,需要为其编写模块 TLC 文件,描述输入/输出端口的关系时需要使用%roll 命令:

```
/* %<Type> Block: %<Name> */
% assign rollVars = ["U", "Y"]
% roll sigIdx = RollRegions, lcv = RollThreshold, block, ...
"Roller", rollVars
% assign y = LibBlockOutputSignal(0, "", lcv, sigIdx)
% assign u = LibBlockInputSignal(0, "", lcv, sigIdx)
% <y> = %<u> * 2;
% endroll
```

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

对这个看起来有点复杂的循环,逐一分析其要素。首先%assign rollVars = ["U", "Y", "P"] 定义 rollVars 变量存储循环体所涉及的模块要素,U 表示输入端口,Y 表示输出端口,在带有参数的模块中,P 表示模块的参数。rollVars 将被传递给 Roller,设置模块输入/输出或参数中每一维 roll 的结构体。

%roll sigIdx = RollRegions 这句话则定义了循环体的循环变量 sigIdx,RollRegions 是自动计算出来的模块输入/输出或参数的维数向量,如 20 维输入信号的 RollRegions 为[0 : 19],sigIdx 按照这个向量逐一循环。

lcv = RollThreshold,lcv 是循环控制变量(loop control variable),从 RollThreshold 获取值,表示当信号维数小于此数值时不生成 for 循环语句,而逐条生成语句。只有信号或参数的维数等于或大于此数值时才生成 for 循环。TLC 全局变量 RollThreshold 的值可以通过 Configuration Parameter 中 Signals and Parameters 页面的 Loop unrolling threshold 参数来设置,默认值为 5。

```
% assign y = LibBlockOutputSignal(0, "", lcv, sigIdx)
% assign u = LibBlockInputSignal(0, "", lcv, sigIdx)
```

LibBlockInputSignal(portIdx, ucv, lcv, sigIdx)函数包含 4 个参数,portIdx 表示模块输入端口的索引号(对于使能或触发端口,可以使用字符串 enable 和 trigger 获取),ucv 通常为空,lcv 和 sigIdx 同前面定义。

上面这两句是使用 TLC 库函数获取模块的输入/输出和参数中第 sigIdx 维所对应的变量,再通过%<y> = %<u> \* %<k>;进行代码生成的变量展开。

在循环体的结尾,使用%endroll 作为终止符号。使用上述 TLC 代码所描述的模块建立的模型,输入/输出信号为 5 维时,代码生成如图 18.2-3 所示。

当%roll 命令使用 TLC 文件中自定义向量的循环时,需要在生成的代码中使用指针指向向量(假设使用 DWork 存储)的地址:

```
datatype * buf = (datatype * )%<LibBlockDWork(name,"","",0)
% roll sigIdx = RollRegions, lcv = RollThreshold, block, ...
"Roller", rollVars
* buf ++ = Customization Code;
%endroll
```

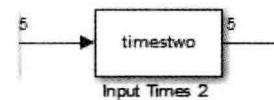
如果自定义循环变量所遍历的维数,可以使用以下方式定义 sigIdx 的循环数据向量:

```
% roll sigIdx = [ 1 2 3 : 5, 6, 7 : 10 ], ....
%endroll
```

### (3) %for

%for 的使用方法与%foreach 基本相同,并且加入了对于是否执行 body 以外部分进行 roll 的判断。其格式如下:

```
% for ident1 = const - exp1, const - exp2, ident2 = const - exp3
    Code section 1
    % body
    Code section 2
    % endbody
    Code section 3
    % endfor
```



```
for (i1=0; i1 < 5; i1++) {
    y0[i1] = u0[i1] * 2.0;
}
```

图 18.2-3 维数为 5 时生成循环代码

当 const-exp2 为非零值时,则 Code section1/2/3 所有代码就会执行一次,并且 ident2 会接收 const-exp3 的值;而 const-exp2 为 0 时则仅执行 Code section 2 段,其他段不执行,并且对其以 ident1 为循环变量进行循环,ident2 为空值。如下例:

```
% for idx = 3, 1, str = "yes"
% warning OK?
% body
    % warning Answer is %<str> .
% endbody
% warning Over
% endfor
```

执行结果为:

```
Warning: OK?
Warning: Answer is yes .
Warning: Over
```

若将%for 第二个参数从 1 改为 0,则执行此脚本的结果为:

```
Warning: Answer is .
Warning: Answer is .
Warning: Answer is .
```

可见,str 为空,并且%for 与%body 之间,%endbody 与%endfor 之间的代码都被忽略,且%body 与%endbody 之间的代码被循环了 3 次。

## 6. 文件流

TLC 语言使用%openfile 创建文件流缓存或打开一个文件,%selectfile 选中或激活一个存在的文件流缓存或文件,%closefile 关闭一个文件流缓存或文件。格式如下:

```
%openfile streamId = "filename.ext" mode {open for writing}
%selectfile streamId {select an open file}
%closefile streamId {close an open file}
```

%openfile 在打开文件时,第 2 个参数 mode,可以是 a 或者 w,表示以“追加”或“重写”的方式创建文件或缓存块。默认以重写的方式写入 streamId 变量或其表示的文件中。当所填写的文件名不存在时,则使用 streamID 为名建立一个缓存块 buffer 进行后续操作。在%openfile 跟%closefile 之间的内容将被写入到缓存块 buffer 中作为变量保存起来,并可以使用%<buffer>将其展开到生成代码中去。这个 buffer 就称为“流”。

```
%openfile buffer
text to be placed in the 'buffer' variable.
%closefile buffer
```

如使用文件流控制方式创建一个名为 my\_flow\_control.txt 的文档:

```
%openfile buffer = "my_flow_control.txt"
This is the first time flow control is used.
%closefile buffer
```

将上述代码存储为 xxxx.tlc 文件,再在 Command window 中使用 tlc xxxx.tlc 命令即可执行,则在当前文件夹下生成了一个 my\_flow\_control.txt 文件,内容就是流缓存 buffer 的内容。

StreamID 所表示的参数有 2 个内建流变量:NULL\_FILE 和 STDOUT,分别表示无输出和使用终端输出文件流内容。%selectfile 同 STDDOUT 联合使用时,也可输出字符串内容到 MATLAB 的 Command Window 中。如:

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```
% selectfile STDOUT
text to be placed in the 'buffer' variable.
```

运行后将上一行的语句显示到 Command Window 中。

## 7. 记录(record)

TLC 记录相当于 M 语言或 C 语言的结构体类型,但形式不同,是构成 rtw 文件的基本元素,格式为:record\_item{ Name Value}。Name 是字符串格式,按照字母顺序进行排列;Value 既可以是字符串也可以是数据类型,这个数据可以是 scalar、向量或矩阵类型。

### (1) 创建一个新记录

使用%createrecord 命令创建一个新记录,其格式为:

```
%createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
```

NEW\_RECORD 为新创建的记录名,{ }内是其所属的子记录。同一层次的子记录之间使用“;”隔开,子记录再创建嵌套子记录时使用 record\_item{ Name Value}方式。可以通过最上层记录名访问其子记录内容:

```
% assign var = NEW_RECORD.foo
% assign var2 = NEW_RECORD.SUB_RECORD.foo
```

也可以创建具有全局访问权限的记录(:表示全局变量标识符):

```
%createrecord::NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
```

### (2) 追加记录

%addtorecord 命令可以向已经存在的记录中追加新的子记录,其格式为:

```
%addtorecord OLD_RECORD NEW_FIELD_NAME NEW_FIELD_VAL
```

%addtorecord 命令后跟 3 个参数,第 1 个参数为追加记录的对象,第 2 个和第 3 个参数表示子记录的名字和值。使用方式如:

```
%createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
%addtorecord NEW_RECORD str "I love Simulink"
%warning %<NEW_RECORD>
```

运行上述代码之后得到的结果为:

```
Warning: { SUB_RECORD { foo 2 } ; foo 1 ; str "I love Simulink" }
```

追加的结果在同一层次中按照首字母顺序排列,相邻两个子记录之间使用“;”分隔。

### (3) 合并记录

对于既存的两个记录,可以进行内容的合并。%mergerecord 命令提供了合并记录的功能。其格式为:

```
%mergerecord OLD_RECORD NEW_RECORD
```

合并之后的内容保存在第 1 个参数 OLD\_RECORD 记录中。如:

```
%createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
%createrecord NEW_RECORD2 {str "I love Simulink"}
%mergerecord NEW_RECORD NEW_RECORD2
%warning NEW_RECORD = %<NEW_RECORD> NEW_RECORD2 = %<NEW_RECORD2>
```

运行之后,合并的结果存在 NEW\_RECORD 中,而 NEW\_RECORD2 的内容不改变。运行上述代码的输出:

```
Warning: NEW_RECORD = { SUB_RECORD { foo 2 } ; foo 1 ; str "I love Simulink" } NEW_RECORD2 = {
str "I love Simulink" }
```

当新旧两个记录中同样层次下存在相同名的记录时,则保留这项记录各自的值不合并,

如：

```
% createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
% createrecord NEW_RECORD2 {foo 12}
% mergerecord NEW_RECORD NEW_RECORD2
% warning NEW_RECORD = %<NEW_RECORD> NEW_RECORD2 = %<NEW_RECORD2>
```

运行上述代码之后生成的结果为：

```
Warning: NEW_RECORD = { SUB_RECORD { foo 2 } ; foo 1 } NEW_RECORD2 = { foo 12 }
```

#### (4) 拷贝记录

记录也可以使用%copyrecord 进行拷贝，格式为：

```
% copyrecord NEW_RECORD OLD_RECORD
```

OLD\_RECORD 是一个既存的记录，NEW\_RECORD 则是 OLD\_RECORD 的一份拷贝。

例如：

```
% createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
% copyrecord NEW_RECORD2 NEW_RECORD
% warning NEW_RECORD = %<NEW_RECORD> NEW_RECORD2 = %<NEW_RECORD2>
```

运行上述代码后，可以看出输出结果中 NEW\_RECORD2 的内容与 NEW\_RECORD 是完全一样的：

```
Warning: NEW_RECORD = { SUB_RECORD { foo 2 } ; foo 1 } NEW_RECORD2 = { SUB_RECORD { foo 2 } ; foo 1 }
```

#### (5) 删除记录

当需要对记录中的域或整个记录删除时，使用%undef 命令。格式为：

```
% undef var
```

var 表示一个 TLC 变量、一个记录名或一个记录中的域的名字。

```
% createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
% copyrecord NEW_RECORD2 NEW_RECORD
% undef NEW_RECORD
```

再次访问 NEW\_RECORD 会报错。如果要删除记录中的一个域成员，可以借助%with 进行范围指定，在此范围内进行变量的删除，如删除 NEW\_RECORD 的 foo 1 这个子记录：

```
% createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
% with NEW_RECORD
% undef foo
% endwith
% warning NEW_RECORD = %<NEW_RECORD>
```

运行之后的结果为 NEW\_RECORD 仅剩下 SUB\_RECORD 的值：

```
Warning: NEW_RECORD = { SUB_RECORD { foo 2 } }
```

直接使用“.”操作符删除记录的域会造成语法错误：

```
% createrecord NEW_RECORD { foo 1 ; SUB_RECORD {foo 2} }
% undef NEW_RECORD.foo
```

执行上述代码会报出错误，如图 18.2-4 所示。

### 8. 变量清除

在基本语法部分已经说明了 TLC 变量的定义方法，%assign 命令。相对地，如何像 MATLAB 中的 clear 一样方便地删除 TLC 语言文件中既存的变量呢？%under 命令即可删除 TLC 变量。如以下的语句中，在 foreach 循环之前将定义的数组变量 data 删除再进行输出：

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```
Error: File: record_text.tlc Line: 2 Column: 18
syntax error
Error using tlc_new
Error: Errors occurred - aborting
```

```
Error in tlc (line 85)
    tlc_new(varargin{:});
```

图 18.2-4 %undef 直接删除记录的域时的报错

```
% assign data = [1,2,3,4,5]
% undef data
% foreach idx = 5
    % warning data[ %<idx>] = %<data[idx]>
% endforeach
```

执行之后报错：

```
Error: File: for_each.tlc Line: 4 Column: 31
Undefined identifier data
```

## 9. 语句换行连接

当 TLC 语句过长,全部写在一行里看起来会比较吃力时,可以考虑换行编写,使用换行连接符将上下行的语句连接起来表示完整的操作意义。TLC 换行连接符包括两种:C 语言的换行连接符“\”和 MATLAB 的换行连接符“...”。经常使用在%roll 命令这种后面所跟参数众多的语句中:

```
% roll sigIdx = RollRegions, lcv = RollThreshold, block, ...
"Roller", rollVars
% roll Idx = RollRegions, lcv = RollThreshold, block, \
    "Roller", rollVars
```

上面两种情况都等效于%roll Idx = RollRegions, lcv = RollThreshold, block, "Roller", rollVars 不换行的情况。

**注:**在字符串中使用换行连接符是非法的。

## 10. 访问范围

使用%assign str = “I Love Simulink”建立的 string 型变量 str 是局部变量,如果定义在 TLC 脚本里,只有此脚本里的语句可以访问;如果定义是在函数里,仅此函数能访问该变量;特别地,如果变量定义在 for 循环等语句块内部,其访问范围也只在这个语句块内部。看一个例子,下面的几行里面,foreach 循环之外已经定义了 idx\_i 这个变量,但在 foreach 循环体内也使用它作为循环变量:

```
% assign idx_i = "original string"
% assign data = [[1, "good"]; [3, 4.5F]]
% foreach idx_i = 2
    % foreach idx_j = 2
        % warning The element of data[ %<idx_i>][ %<idx_j>] is ...
        %<data[idx_i][idx_j]>
    % endforeach
% endforeach
% warning The original is %<idx_i>
```

上述代码的执行结果为:

```
Warning: The element of data[0][0] is 1
Warning: The element of data[0][1] is good
Warning: The element of data[1][0] is 3
Warning: The element of data[1][1] is 4.5E + 0F
Warning: The original is original string
```

可以看出,在循环体内外虽然使用同样的变量,但其值却不一样,这就是作用域的作用。

如果希望一个变量可以在任意处被访问并且内容都是同一个值,应将其定义为全局变量。

```
% assign ::str = "I Love Simulink"
```

“::”即为 TLC 的全局变量标示符。上述例子中由于变量 idx\_i 既作为字符串又作为向量下标,强行使用全局变量的话会报错。

%with/%endwith 命令可以锁定记录数据类型的范围,方便其内部 TLC 语句以子记录的名字方式直接访问其值,无须从最上层记录逐层域访问。例如在 rec 这个记录中包含 3 个域 field1、field2、field3,为了访问其值通常需要使用 rec.field1,使用%with 限定范围之后,在%with/%endwith 内部的 TLC 代码可以直接使用子记录名 field1 来访问其值。如:

```
%createrecord rec {field1 1 field2 2 field3 3}
%with rec
  % warning filed1 = %<field1>
%endwith
%warning rec.field1 = %<rec.field1>
```

在层次众多的记录文件中,这种方法能体现出更大的方便之处。

## 11. 输入文件控制

输入文件控制包括 2 种方式:%include string 和%addincludepath string。

%include string 将在搜索路径下寻找名为 string 的文件,并在%include 语句出现的地方将此文件内容内联展开。

%addincludepath string 中 string 是一个绝对路径或相对路径,%addincludepath 将这个路径添加到 TLC 的搜索路径中,以便出现%include 语句时搜索其包含的文件。比如:

```
%addincludepath "C:\\folder1\\folder2" % %添加一个绝对路径
%addincludepath ".\\folder2" % %添加一个相对路径
```

TLC 搜索路径依照如下顺序:

① 当前路径。

② 所有的%addincludepath 添加的路径,对于多个%addincludepath,依照从下到上搜索的顺序。

③ 命令行中通过-I 命令添加的路径。

%include 功能类似于 C 语言的 #include,%addincludepath 功能类似于 MATLAB 的 addpath 命令。

## 12. 输出格式控制

TLC 中使用%realformat 命令控制输出实变量所显示的格式。如使用 16 位精度的指数显示:

```
%realformat "EXPONENTIAL"
```

或者无精度损失和最小字符数格式(为 S 函数提供生成代码功能的模块级 TLC 文件就使用此种格式):

```
%realformat "CONCISE"
```

例如 real32 类型的数据 4.5F,在两种输出格式下的输出分别如表 18.2-1 所列。

表 18.2-1 不同格式下 real32 显示方式

| EXPONENTIAL 格式 | CONCISE 格式 |
|----------------|------------|
| 4.5E+0F        | 4.5F       |

### 13. 指定模块生成代码的语言种类

Simulink 中支持代码生成的模块的 TLC 文件中都需要指定该模块生成代码的语种, %language 就是指定模块 TLC 文件的命令。如果自定义支持嵌入式代码生成的模块及其 TLC 文件,那么在 TLC 文件中就可以选择使用%language “C”指定语言类型为 C 语言。对于 Simulink 模块的 TLC 文件,其中必须包含%implements 指令,其格式为:

```
% implements "Block-Type" "Language"
```

Block-Type 是指该模块的 S 函数名, TLC 文件也是此名字。Language 表示其生成的目标代码的语种类型。对于自定义的为了生成嵌入式 C 语言的 MCU 芯片驱动中断控制器模块,其 TLC 文件开头处必须包含这样一个命令:

```
% implements Interrupt "C"
```

紧接着此句的是%function 的函数定义,实现 S 函数代码生成的具体功能。

另外,%generatefile 提供一个匹配关系,将 Simulink 模块与 TLC 文件联系起来。其格式为:

```
% generatefile "Type" "blockwise.tlc"
```

Type 为 rtw 文件中模块的记录中 Type 参数的值,也即模块的 blocktype 属性,如:%generatefile "Sin" "sin\_wave.tlc"。

### 14. 断言

%assert 命令为 TLC 提供断言功能,其格式为:

```
% assert expression
```

当 expression 结果为 TLC\_FALSE 时,TLC 将进行堆栈追踪。为了开启 TLC 的断言功能,必须在 Configuration Parameter 选项卡的 Code Generation→Debug 页面勾选 Enable TLC assertion 选项,如图 18.2-5 所示。默认此选项是关闭的。

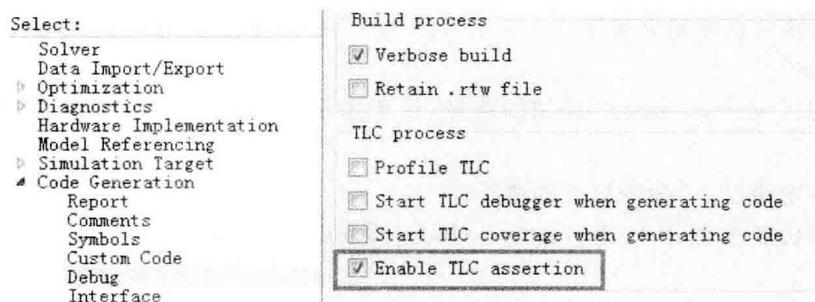


图 18.2-5 开启 TLC 断言功能

### 15. 函数

TLC 语言支持脚本与函数定义两种方式。使用%function 为开头定义函数原型,以%endfunction 为终止符结束函数体。其格式如下:

```
% function name(optional - arguments) void
% return
% endfunction
```

name 表示函数名字, optional-arguments 表示函数的参数列表。其后可以跟 void 表示函数没有返回值;也可以跟 Output 表示函数有返回值。当函数设置为 Output 时函数体中需要 %return 命令返回变量作为函数的输出。例如:

```
% function LibGetMathConstant(ConstName, ioTypeId) void

% assign constInfo = SLibGetMathConstantInfo(ConstName, ioTypeId)
% if ! ISEMPTRY(constInfo)
%   return constInfo.Expr
% else
%   return ""
% endif

% endfunction
```

### 18.2.3 变量类型

TLC 语言使用的变量类型与 MATLAB 变量所使用内建类型有所不同。在 TLC 语言中不仅是数据的类型,甚至数据的组织方式都被作为一个单独的类型,如 Matrix、Vector、Range 类型等。具体请参考表 18.2-2 所列。

表 18.2-2 TLC 变量类型表

| 数值类型名(简写)             | 表现形式举例                                    | 说 明   |
|-----------------------|---|---|
| Boolean(B)            | 0 == 0                                    | 返回值为 TLC_TRUE(真)或 TLC_FALSE(假),由逻辑比较或其他逻辑操作返回   |
| Number(N)             | 100                                       | 整型数   |
| Real(D)s              | 3.14159                                   | 浮点数   |
| Real32(F)s            | 3.14159F                                  | 32 位浮点数   |
| Complex(C)            | 1.0 + 2.0i                                | 64 位双精度浮点复数   |
| Complex32(C32)        | 1.0F + 2.0Fi                              | 32 位单精度浮点复数   |
| Gaussian(G)           | 1+2i                                      | 32 位整型复数  |
| Unsigned(U)           | 100U                                      | 32 位无符号整数   |
| Unsigned Gaussian(UG) | 1U + 2Ui                                  | 32 位无符号整型复数   |
| String                | "I Love MATLAB"                           | 包含在双引号中的字符串   |
| Identifier            | abc                                       | 此类型仅出现在 rtw 的记录中,不可直接在 TLC 表达式中使用。如果希望比较其内容,可以直接与 String 类型比较,Identifier 将自动转换为 String 类型 |
| File                  | %openfile out= "xx.c"<br>%openfile buffer | 使用 %openfile 打开的字符串缓存或者文件   |
| Function              | %function my_func ...                     | TLC 的函数类型,需要使用 %endfunction 结束  |
| Range                 | [1 : 10]                                  | 表示一组整数序列,如 RollRegions 使用在 roll 循环体中表示循环变量遍历的值  |
| Vector                | [1, 2.0F, "good"]                         | 向量类型,每个元素的类型可以是 TLC 的内建类型,但是不能是 Vector 或 Matrix   |

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

| 数值类型名(简写) | 表现形式举例                               | 说 明  |
|-----------|--------------------------------------|--|
| Matrix    | %assign a = [[1, "good"]; [3, 4.5F]] | 矩阵类型,矩阵中每个元素类型可以不同,但是不能为 Vector 或 Matrix。每行各个元素使用“,”分开,每列使用“;”分开 |
| Scope     | System { ... }                       | 范围类型,如 rtw 中的 CompiledModel、block 记录的范围                          |
| Subsystem | <sub>                                | 子系统标示符,作为语句扩展的内容   |
| Special   | FILE_EXISTS                          | 特殊内建类型如 FILE_EXISTS  |

其中, Boolean、Real、Real32、Complex、Complex32、Gaussian、Number、Unsigned 和 Unsigned Gaussian 类型又称为数值(Numeric)类型。Boolean、Number 和 Unsigned 3 种数据类型被统称为整数类型。对于 Matrix 或 Vector 类型,内部所包含的元素数据类型可以不相同,在 TLC 代码中可以使用两层 for 循环将元素一一输出:

```
% assign data = [[1, "good"]; [3, 4.5F]]
% foreach idx_i = 2
    % foreach idx_j = 2
        % warning The element of data[ %<idx_i>][ %<idx_j>] is ...
    >
    % endforeach
% endforeach
```

执行之后,结果为:

```
Warning: The element of data[0][0] is 1
Warning: The element of data[0][1] is good
Warning: The element of data[1][0] is 3
Warning: The element of data[1][1] is 4.5E + 0F
```

### ★ 数据类型提升

当 TLC 对不同类型进行操作或运算时,会对结果进行数据类型的提升。通常提升之后使用表示范围较广的类型存储输出值。提升之后的数据类型如表 18.2-3 所列。

表 18.2-3 不同类型运算后经过提升的输出数据类型

|     | B   | N   | U   | F   | D   | G   | UG  | C32 | C   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| B   | B   | N   | U   | F   | D   | G   | UG  | C32 | C   |
| N   | N   | N   | U   | F   | D   | G   | UG  | C32 | C   |
| U   | U   | U   | U   | F   | D   | UG  | UG  | C32 | C   |
| F   | F   | F   | F   | F   | D   | C32 | C32 | C32 | C   |
| D   | D   | D   | D   | D   | D   | C   | C   | C   | C   |
| G   | G   | G   | UG  | C32 | C   | G   | UG  | C32 | C   |
| UG  | UG  | UG  | UG  | C32 | C   | UG  | UG  | C32 | C   |
| C32 |
| C   | C   | C   | C   | C   | C   | C   | C   | C   | C   |

## 18.2.4 操作符和表达式

TLC 语言使用的操作符号和表达式如表 18.2-4 所列。

表 18.2-4 TLC 操作符号及表达式表

| 操作符和表达式                            | 作用说明  |
|------------------------------------|---|
| <code>::variablename</code>        | 全局变量,当出现在函数中时,告诉函数该变量的访问权限是全局的,不使用函数的局部访问权限   |
| <code>expr[expr]</code>            | 数组或矩阵的下标索引访问方式,expr 必须是 0~(N-1),N 为数组长度或矩阵某一维的长度  |
| <code>func([expr],expr,...)</code> | 函数调用,func 为函数名,expr 为函数的参数列表  |
| <code>expr.expr</code>             | 域访问符,第 1 个 expr 为记录类型,第 2 个 expr 为其内部的一个参数名   |
| <code>(expr)</code>                | 括号,括号内的表达式优先度高  |
| <code>! expr</code>                | 逻辑取反。expr 必须是 Boolean 或数值类型,返回值为 TLC_TRUE 或 TLC_FALSE   |
| <code>-expr</code>                 | 一元操作符,取负。expr 必须为数值类型   |
| <code>+expr</code>                 | 一元操作符,正号,expr 必须为数值类型。一般表达式默认为+,不写出   |
| <code>~expr</code>                 | 按位取反。expr 必须是整数   |
| <code>expr * expr</code>           | 乘法运算符。expr 必须是数值类型  |
| <code>expr / expr</code>           | 除法运算符。expr 必须是数值类型  |
| <code>expr + expr</code>           | + 运算符,可以用于 scalar、vector、matrix 和 record 等多种数据类型,且作用不同<br>用于 scalar 数值时表示两个数相加,此时 expr 必须是数值类型<br>用于 String 类型时,将两个字符串拼接起来返回<br>当首个 expr 是 Vector,第 2 个 expr 是数值类型时,则第二个数追加到 Vector 中去<br>当首个 expr 是 Matrix,第 2 个 expr 是 Vector 时,如果 Vector 与 Matrix 的列数相同,则追加到 Matrix 中作为新的一行元素<br>如果第 1 个 expr 是记录类型,则第 2 个 expr 作为一个参数追加到这个记录类型中去,其值为第 2 个 expr 的当前值 |
| <code>expr - expr</code>           | 减法运算符。expr 必须是数值类型  |
| <code>expr % expr</code>           | 求余数运算符。expr 必须是整数类型   |
| <code>expr &lt;&lt; expr2</code>   | 左移操作符,将 expr 按照二进制位向左移动 expr2 位   |
| <code>expr &gt;&gt; expr2</code>   | 右移操作符,将 expr 按照二进制位向右移动 expr2 位。>>不能直接在%<>中被识别,需要使用\进行转义,%<expr>>expr2>   |
| <code>expr &lt; expr2</code>       | 比较 expr 是否小于 expr2。expr,expr2 都必须是数值类型  |
| <code>expr &gt; expr2</code>       | 比较 expr 是否大于 expr2。expr,expr2 都必须是数值类型。%<>中使用时需要进行转义%<expr>>expr2>  |
| <code>expr &lt;= expr2</code>      | 比较 expr 是否小于等于 expr2。expr,expr2 都必须是数值类型  |
| <code>expr &gt;= expr2</code>      | 比较 expr 是否大于等于 expr2。expr,expr2 都必须是数值类型。<>中使用时需要进行转义%<expr>>expr2>   |
| <code>expr == expr2</code>         | 比较 expr 是否等于 expr2  |
| <code>expr != expr2</code>         | 比较 expr 是否不等于 expr2   |
| <code>expr &amp; expr2</code>      | 将两个操作数进行二进制按位“与”操作,expr 和 expr2 必须是整数类型   |
| <code>expr   expr2</code>          | 将两个操作数进行二进制按位“或”操作,expr 和 expr2 必须是整数类型   |
| <code>expr ^ expr2</code>          | 将两个操作数进行二进制按位“异或”操作,expr 和 expr2 必须是整数类型  |

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

续表 18.2-4

| 操作符和表达式           | 作用说明  |
|-------------------|---|
| expr && expr2     | 将两个操作数进行按逻辑“与”操作,结果返回 TLC_TRUE 或 TLC_FALSE。expr 和 expr2 可以是数值类型或 Boolean 类型 |
| expr    expr2     | 将两个操作数进行按逻辑“或”操作,结果返回 TLC_TRUE 或 TLC_FALSE。expr 和 expr2 可以是数值类型或 Boolean 类型 |
| expr? expr1:expr2 | 当 expr 值为 TLC_TRUE 时,返回 expr1;否则返回 expr2                                    |
| expr1, expr2      | 逗号分隔符,返回后面的变量 expr2。多个逗号分开时返回最后一个变量   |

**注意:**当在“%<>”中使用“>”,“>=”和“>>”之类带有“>”符号的操作符时,不能直接使用,需要前面加一个\以进行转义。

若您对此书内容有任何疑问,可以凭在线交流卡登录MATLAB中文论坛与作者交流。

## 18.2.5 TLC 内建函数

TLC 语言提供了一些内建函数,虽然没有 M 语言的内建函数那样丰富,但是也十分实用,方便了代码生成时必需的操作。TLC 提供的内建函数全部都使用大写字母书写,本节将其中经常用到的内建函数通过讲解和举例进行说明。

### 1. 数据类型转换

CAST 函数是 TLC 语言中负责数据类型转换的重要函数,其格式为:

```
CAST("DataType", variablename)
```

第 1 个参数表示第 2 个参数转换的目标类型名,如"Number","Real32","String"等。第 2 个参数则是进行转换的操作数。如:

```
% assign data = [[1, "good"]; [3, 4.5F]]
% assign cast = CAST("Real32", %<data[0][0]>
% warning %<cast>
```

运行上述代码,1 将被转换为 1.0F。

### 2. 变量的存在性

通常为了避免语法错误,在访问记录的某个参数前需要确定它是否真正存在。这时就需要 EXISTS 函数,其格式为:EXISTS(Var)。Var 是一个变量名或一个记录的参数;返回值为 Boolean 型。例如,在打印记录的某个成员之前先用%if 和 EXISTS 联用确定其存在性:

```
% createrecord record {foo 1 subrec{foo 2}}
% if EXISTS(record.foo)
    % warning record has a parameter which value is %<record.foo>.
% endif
```

另外,对于文件的存在性判断需要使用 FILE\_EXISTS(expr) 函数,其中 expr 为表示文件名的字符串。判断一个文件是否存在与当前目录的代码为:

```
% if FILE_EXISTS("text.tlc")
    % warning text.tlc is on the path.
% endif
```

### 3. 记录的域操作

%ISFIELD、%GETFIELD、%SETFIELD 及%REMOVEFIELD 函数都是与记录的域相关的内建函数。通过%ISFIELD 判断某个字符串表示的参数名是否为记录的域;通过%SETFIELD 和%GETFIELD 设置/获取记录中参数的值。其格式为:

```
% ISFIELD(record, "fieldname")
% GETFIELD(record, "fieldname")
% SETFIELD(record, "fieldname", value)
% REMOVEFIELD(record, "fieldname")
```

四者联用,可以先查询记录中是否存在某个参数,如果存在则获取其值,然后再改变其值,再次获取并打印该参数值,最后删除此参数并判断它是否还存在与记录之中。代码如下:

```
% createrecord record {foo 1 subrec{foo 2}}
% if ISFIELD(record, "foo")
% warning record has a parameter foo which value is %<GETFIELD(record, "foo")>.
% <SETFIELD(record, "foo", 12)>
% warning parameter foo value is %<GETFIELD(record, "foo")> now.
% <REMOVEFIELD(record, "foo")>
% warning parameter foo of record now is existed? Ans: %<ISFIELD(record, "foo")>
% endif
```

上述代码运行之后得到以下的结果:

```
Warning: record has a parameter foo which value is 1.
Warning: parameter foo value is 12 now.
Warning: parameter foo of record now is existed? Ans: 0
```

除上述 4 个函数之外,还有%FIELDNAMES 函数可以查询记录中包含的内一层的记录名。对于上述代码创建的 record,此函数将返回作为参数的记录中次层的记录名:

```
% createrecord record {foo 1 subrec{foo 2}}
% warning %<FIELDNAMES(record)>
```

运行上述代码后得到的结果为:

```
Warning: [foo, subrec]
```

#### 4. 相等判断

%ISEQUAL 函数可以判断 2 个变量是否相等。其格式为:

```
ISEQUAL(expr1, expr2)
```

当 expr1,expr2 都为数值类型时,即使不是同一个数据类型,只要表达式运算的值大小相同即返回 TLC\_TRUE,否则返回 TLC\_FALSE。如果 expr1 和 expr2 不是数据类型,二者的变量类型和内容必须完全一样时才返回 TLC\_TURE。例如下面的代码返回 1 1 0:

```
% warning %<ISEQUAL(TLC_TRUE,1)> %<ISEQUAL(3, 3.0F)> %<ISEQUAL("Str", "St")>
```

#### 5. 空值判断

判断一个变量是否为空的 TLC 函数为%ISEMPTY,它可以判断的数据类型为字符串、向量 Vector、矩阵 Matrix 和记录 Record 等。如定义一个名为 MM 的记录,然后再将其唯一的参数 Name 删除,这时再判断其是否为空,返回 TLC\_TRUE(1)。

```
% createrecord MM {Name "Hyo"}
% with MM
% undef Name
% endwith
% warning %<ISEMPTY(MM)>
```

返回值为 1。同样的,直接对一个空矩阵或向量判断也返回 1,如:

```
% warning %<ISEMPTY([])>
```

#### 6. 判断变量类型

%TYPE 函数可以返回一个变量的类型,格式为:TYPE(expr)。例如判断下列变量的变量类型:

```
% warning %<TYPE("Str")> %<TYPE([1,2;3,5])> %<TYPE(10.3F + 6.71Fi)>
```

运行上述代码后返回值为：

```
Warning: String Matrix Complex32
```

## 7. 判断空格

判断空格与判断空值不同,当一个变量内仅仅包含空格,如\n, \t, \r 等时返回 1,否则返回零。WHITE\_SPACE 函数就是判断参数是否为全空格的函数。其格式为:

```
WHITE_SPACE(expr)
```

如将各种空格表达方式组成一个字符串再使用函数判断,结果返回 1:

```
% warning %<WHITE_SPACE(" \t \n \r \r")>
```

## 8. 调用 matlab 函数

TLC 虽然本身不具有 MATLAB 那么强大的函数库,但却可以方便地调用 MATLAB 的内建函数。调用有两种方式,当调用没有返回值的 MATLAB 函数时使用%matlab 命令,调用有返回值的函数时则使用%FEVAL 命令,对于有多个返回值的函数,TLC 只能接收首个返回值。

%matlab 是少见的以小写字母方式使用的 TLC 命令。其格式为

```
% matlab fun(agr)
```

如调用 MATLAB 的 disp 函数:

```
% matlab disp("TLC calls MATLAB function")
```

运行上述代码后,Command Windows 中显示结果:TLC calls MATLAB function。

%FEVAL 函数的使用格式为:

```
% assign result = FEVAL(MATLAB - function - name, rhs1, rhs2, ... rhs3, ... );
```

FEVAL 函数的首个参数为 MATLAB 函数名,使用双引号括起来。其后参数为这个 MATLAB 函数的参数列表。返回值只能接收 MATLAB 函数的首个返回值,且其数据类型自动转换为 TLC 内建数据类型。例如使用 FEVAL 调用 MATLAB 的正弦函数 sin 来计算一个输入向量的点对应的正弦值,并显示返回结果的值和 TLC 数据类型:

```
% assign data = FEVAL("sin", [0 : 9])
```

```
% matlab disp(data)
```

```
% warning The data type of variable "data" is %<TYPE(data)>. And the element data type is %<TYPE(data[0])>.
```

运行上述代码之后,得到以下结果:

```
0 0.8415 0.9093 0.1411 -0.7568 -0.9589 -0.2794 0.6570 0.9894 0.4121
```

```
Warning: The data type of variable "data" is Vector. And the element data type is Real.
```

又如使用 FELAL 调用 MATLAB 中的正则表达式函数 regexp 进行字符串中的字串位置查找:

```
% realformat "CONCISE"
```

```
% assign pos = FEVAL("regexp", "I love Simulink", "Simu")
```

```
% assign pos = CAST("Number", pos)
```

```
% warning The index of Simu is %<pos>
```

运行上述代码之后,显示信息为:

```
Warning: The index of Simu is 8
```

## 18.2.6 TLC 命令行

在 Command Window 中调用 TLC 编译器运行 TLC 脚本文件的命令行是 TLC 命令,它

负责 TLC 编译器的调用,格式为: tlc[switch1 expr1 switch2 expr2 ...] filename.tlc。之前的例子中已经多次使用到了 tlc filename, tlc 这样的命令,此处关于[switch1 expr1 switch2 expr2 ...]的开关命令展开讲解。这些命令可以有多个,顺序随意,switchn 表示开关符号,expr1 表示开关的参数。如果同一个开关符号出现多次,那么最后一个被认为是有效的。TLC 开关命令如表 18.2-5 所列。

表 18.2-5 TLC 开关命令表

| 开关             | 意义   |
|----------------|--|
| -r filename    | 读取数据文件载入到 TLC 中,如 rtw 文件   |
| -v[number]     | 设置输出信息的详细级别为 number,不设置时默认为级别 1  |
| -Ipath         | 增加特定文件夹路径到 TLC 搜索路径中   |
| -Opath         | 指定输出文件的存放路径。输出文件包括%openfile 和%closefile 生成的流文件或者日志文件   |
| -m[number]     | 指定最大的报错数为 number,不设置时,默认为报出前 5 个错误。如果只写-m 不写 number,则认为 number 为 1   |
| -x0            | 仅解析 tlc 文件而不执行   |
| -lint          | 进行一些简单的性能检查  |
| -p[number]     | 设置 TLC 每执行 number 个操作输出一个  |
| -d[a c f n o]  | 启动 TLC 的 debug 模式<br>-da 使 TLC 执行%assert 命令,但是当使用 rtwbuild() 或 Ctrl+B 启动模型代码生成时,忽略 -da 命令,因为这时断言是否有效是根据 Enable TLC assertion 这个选项定的<br>-dc 启动 TLC 命令行调试器<br>-df filename 将启动 TLC 调试器并运行名为 filename 的 tlc 调试脚本文件。所谓调试脚本文件,就是包含调试命令的文本文件。TLC 仅在当前路径下搜索有效脚本文件<br>-dn 将为所执行的 tlc 文件产生行覆盖度日志,告知哪一行被执行了,哪一行没有执行。例如:<br><pre>1: % if 0  0: % assign at = "gsd"  0: % endif - do 则停止调试行为</pre> |
| -dr            | 检查是否存在环形记录文件,这种文件彼此互相引用,会造成内存泄漏  |
| -a[ident]=expr | 为一个变量 ident 设置一个初始值 expr。与%assign 命令功能相同   |
| -shadow[0 1]   | 设置是否开启遮蔽警告功能,当记录中的参数覆盖了一个局部变量时:<br>-shadow0 关闭警告<br>-shadow1 开启警告  |

例如,使用命令行运行一个名为 test.tlc 的 tlc 脚本读取一个 rtw 文件中模型的名字,并打印输出信息。假设这个 rtw 文件已经创建完毕。这个 test.tlc 文件仅有一行内容:

```
% warning CompiledModel.Name = %<GETFIELD(CompiledModel, "Name")>
```

在 Command Window 中输入命令:

```
tlc test.tlc -r untitled.rtw -v
```

输出信息为:

```
Warning: File: test.tlc Line: 1 Column: 9
% warning directive: CompiledModel.Name = untitled
```

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

### 18.2.7 TLC 调试方法

TLC 语言的调试不像 M 语言那样方便,直接单击行号即可设置断点,单击 run 后程序运行到断点的行号即可停下,可进行 step、step in、step out 和 continue 等操作,并且可以在工作空间查看当前语句所在的控件中各个变量的值。

TLC 语言的调试由 TLC 调试器掌控,必须开启之后才能够在运行模型的代码生成时进入到调试模式。TLC 调试器的开关在 Configuration Parameter 对话框中 Code Generation 的 Debug 子标签中。Debug 页面对话框如图 18.2-6 所示。

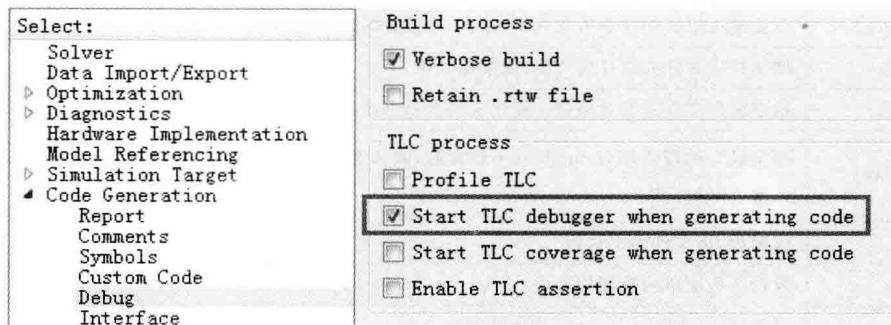


图 18.2-6 TLC 调试器开关

对一个配置好生成代码(固定点解算器,系统目标文件设置为 ert.tlc)的模型,开启 TLC 调试器选项。按下  $ctrl + B$  启动代码生成编译过程,Command Window 上动态显示编译过程信息,显示如图 18.2-7 所示信息时便进入了 TLC 调试模式。

```
-dc switch
00001: %% SYSTLC: Embedded Coder TMF: ert_default_tmf MAKE: make_rtw %
TLC-DEBUG>
```

图 18.2-7 TLC 调试器启动后进入调试模式

这时,可以使用命令行进行 tlc 文件的调试,使用 whos 查看当前访问范围中存在的变量及变量的 TLC 数据类型。在 TLC-DEBUG 命令行输入 whos 之后按回车显示如图 18.2-8 所示信息。

其中能观察到具有全局访问权限 CompiledModel 记录名。如果希望查看某个变量的值,可使用 print 调试命令将其显示到 Command Windows 中。如查看变量 LaunchReport 的值使用图 18.2-9 所示代码。

在 print 命令后也可以调用 TLC 内建函数,如输入 print TYPE(LaunchReport) 再按下回车可以得到其数据类型 Number 作为返回值。如果希望观察此后的代码情况,可以输入 list 命令,将返回当前程序之后的 10 行代码信息如图 18.2-10 所示。

list 命令后面可以跟两个非负整数,使用逗号隔开,如 list 10,20,意义是显示当前 TLC 文件的第 10~20 行的代码。使用 next 命令或简易命令 n 和 step 命令可进行单步执行,next 是同一层次的单步执行,step 则是 step in 执行。使用 continue 或 cont 可以使 TLC 调试器全速执行,只有遇到断点和错误才会停下。断点设置的方法也是通过 TLC 调试器命令 break 或简易命令 b 进行,并制定断点所在的文件及行数,文件全名及行数之间使用“:”分隔,如 break

```
TLC-DEBUG> whos
Variables within: <GLOBAL>
CombineOutputUpdateFcns      Number
CompiledModel                 Scope
ERICustomFileBanners         Number
ERICustomFileTemplate        String
ExtMode                       Number
ExtModeStaticAlloc            Number
ExtModeStaticAllocSize        Number
ExtModeTesting                Number
ExtModeTransport              Number
FoldNonRolledExpr            Number
ForceBlockIOInitOptimize     Number
ForceParamTrailComments      Number
GenCodeOnly                   Number
GenFloatMathFcnCalls         String
GenerateASAP2                 Number
GenerateComments              Number
GenerateErtSFunction         Number
GenerateFullHeader            Number
GenerateReport                Number
GenerateSampleERTMain         Number
GenerateTraceInfo             Number
INT16MAX                      Number
INT16MIN                      Number
INT32MAX                      Number
INT32MIN                      Number
```

图 18.2-8 使用 whos 命令查看当前访问范围下的变量及其类型

```
TLC-DEBUG> print LaunchReport
1
```

图 18.2-9 使用 print 命令查看变量值

```
TLC-DEBUG> list
00012: %selectfile NULL_FILE
00013:
00014: %assign CodeFormat = "Embedded-C"
00015:
00016: %assign TargetType = "RT"
00017: %assign Language = "C"
00018: %if !EXISTS("AutoBuildProcedure")
00019:   %assign AutoBuildProcedure = !GenerateSampleERTMain
00020: %endif
00021:
```

图 18.2-10 使用 list 命令查看代码

ert.tlc:14 将断点设置在 ert.tlc 的第 14 行,再输入 cont 命令全速运行即会停止在离开开始运行的语句最近的一处断点处,停止到断点后显示信息如图 18.2-11 所示。

如果 break 命令后面的数值所对应的行没有代码(全部为空或全部为注释),那么 TLC 调试器将会把断点自动转移到其后最近的有效代码行,并给出 warning 提醒,如图 18.2-12 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```
TLC-DEBUG> cont

Breakpoint 1
00014: %assign CodeFormat = "Embedded-C"
```

图 18.2-11 使用 cont 运行到断点

```
TLC-DEBUG> b ert.tlc:20
Warning: File: Debugger Command Line Line: 1 Column: 1
Breakpoint number 3 was set on the next valid line (24)
```

图 18.2-12 断点自动转移到有效行

每次断点设置后,运行到断点处停下来时,断点序号会被显示出来,这个序号作为断点的标示符,可以用 clear 命令清除(clear 1),或者模型编译生成代码全过程执行完毕时,断点也会自动清除。clear all 可以清除已经设置的所有断点。

也可以使用 Disable 关闭断点,再次使用时用 Enable 使能。

在调试模式下,可以省略%直接使用 TLC 命令 assign 进行操作,如图 18.2-13 所示。

```
TLC-DEBUG> assign str = "Let me try TLC"
TLC-DEBUG> print str
Let me try TLC
```

图 18.2-13 在调试模式下定义新变量

其他 TLC 命令不能在调试模式下使用。调试结束后,可以输入 cont 运行代码生成流程,或者输入 quit 命令退出调试模式。其他的调试命令还有:condition、up、down、finish、ignore、loadstate、savestate、stop、thread、threads 和 where 等。可以使用 help command 查询命令的帮助,如图 18.2-14 所示。

```
TLC-DEBUG> help disable
disable [<breakpoint number>] - Disable a breakpoint
Disable the specified breakpoint. If no breakpoint is specified, DISABLE disables the last created b

TLC-DEBUG> help finish
finish - Break after completing the current function
Continues execution from where it is stopped, and re-enters the debugger after
the current function has exited, or some other reason to enter the debugger
(e.g., a breakpoint or error) is encountered, whichever comes first.
```

图 18.2-14 使用 help 查询调试命令的格式和用法

## 18.2.8 TLC 文件的覆盖度

软件测试中的单元测试重点是测试事件的覆盖度,100%的覆盖度测试是保证软件质量品质的基础。代码生成时,启动覆盖度检测功能如图 18.2-15 所示。

选择 Start TLC coverage when generating code 项后,启动代码生成过程中,TLC 编译器会为每个被执行的 TLC 文件生成一个 log 文件,存放在 model\_ert\_rtw 文件夹中,该文件夹下的文件列表如图 18.2-16 所示。

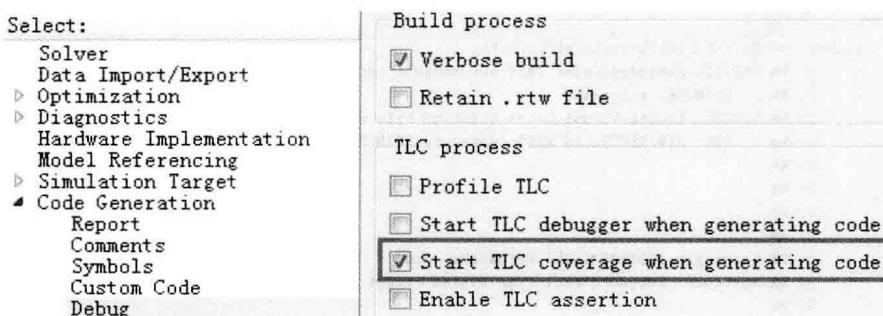


图 18.2-15 启动覆盖度检测

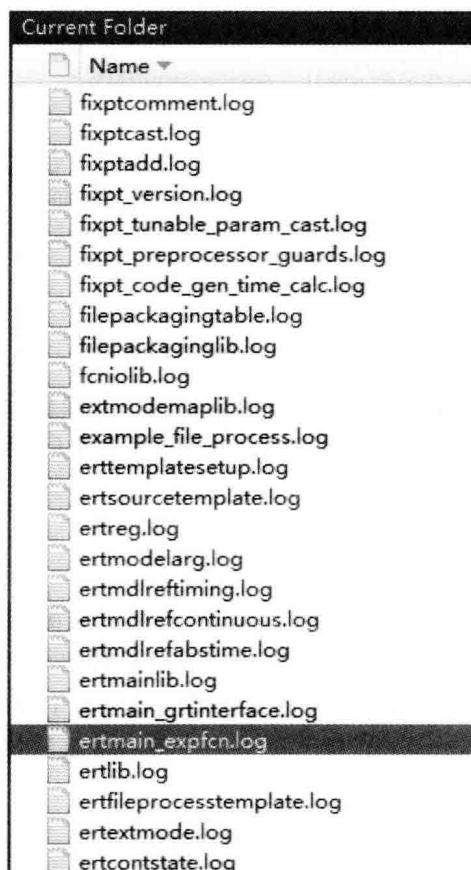


图 18.2-16 覆盖度 log 文件

这些 log 文件对 TLC 的每行语句在代码生成过程中的执行次数做统计, 0 表示没有执行过, 1 表示执行 1 次。ert.tlc 的 log 文件如图 18.2-17 所示。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

**注意：**TLC 编译器将下列命令认为是不执行的语句, 其执行次数都是 0, 也不产生时间消耗:

```
% filescope      % else      % endif
% endforeach    % endfor    % endroll
% endwith       % body      % endbody
% endfunction   % endswitch % default
Comment: % % or / % text % /
```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```

ert.log  +
Source: D:\MAILAB\R2013b\rtw\c\ert\ert.tlc
0: %% SYSILC: Embedded Coder TMF: ert_default_tmf MAKE: make_rtw \
0: %% EXIMODE: ext_comm
0: %% SYSILC: Create Visual C/C++ Solution File for Embedded Coder\
0: %% TMF: RIW_MSVCBuild MAKE: make_rtw EXIMODE: ext_comm
0: %%
0: %%
0: %%
0: %%
0: %%
0: %%
0: %%
0: %%
0: %% Copyright 1994-2011 The MathWorks, Inc.
0: %% Abstract: Embedded real-time system target file.
0: %%
1: %selectfile NULL_FILE
1:
1: %assign CodeFormat = "Embedded-C"
1:
1: %assign TargetType = "RI"
1: %assign Language = "C"
1: %if !EXISTIS("AutoBuildProcedure")
1: %assign AutoBuildProcedure = !GenerateSampleERIMain
0: %endif
1:
0: %% The model_SetEventsForThisBaseRate function is not required for the
0: %% VxWorks environment, i.e., when using an operating system.
1: %assign SuppressSetEventsForThisBaseRateFcn = (TargetOS == "VxWorksExample")
1: %if !EXISTIS("InlineSetEventsForThisBaseRateFcn")
1: %assign InlineSetEventsForThisBaseRateFcn = ILC_TRUE
0: %endif
1: %if !EXISTIS("SuppressMultiTaskScheduler")

```

图 18.2-17 ert.tlc 的 log 文件

根据 log 文件, 开发者能够很快发现哪些分支语句没有被执行过, 并根据这个分支语句的判断条件进行新的测试事件的设置, 重新执行并分析覆盖度, 不断改进, 使 tlc 文件编写得更加可靠高效。

## 18.2.9 TLC Profiler

TLC 代码的执行时间取决于 TLC 脚本、宏、函数和内建函数这些构成 TLC 代码的元素的执行时间。TLC Profiler 可以在执行过程中收集 TLC 代码各个元素的执行时间, 并汇总到 HTML 的报告中, 让开发者更容易分析并找到代码生成过程中最花费时间的代码。

勾选 Configuration Parameter 的 Debug 页面中的对应选项, 如图 18.2-18 所示, 可启动 TLC Profiler。

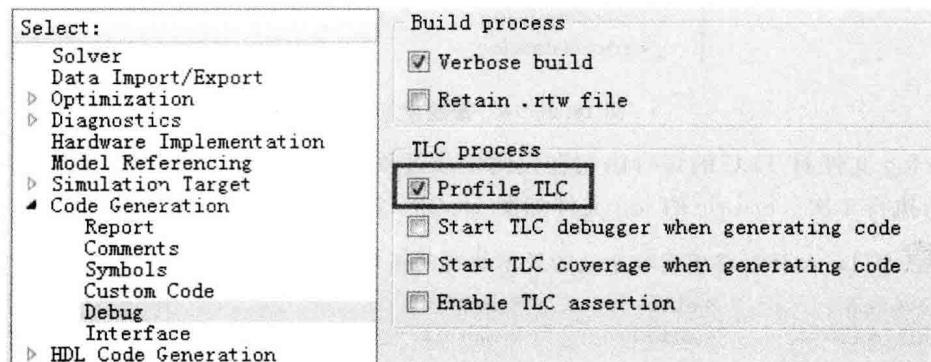


图 18.2-18 勾选 Profile TLC 选项

勾选 Profile TLC 选项后,在代码生成的最后阶段将启动 Profile 功能,分析过程显示如图 18.2-19 所示进度条。进度条满之后,将生成一个 HTML 报告,存放在当前目录的子目录 model\_ert\_rtw 文件夹下。报告首页主要包括两个方面:TLC profile 报告摘要和函数列表。报告摘要中详细列出了所监测的 TLC 代码中包括的脚本数目、函数数目和时钟频率等信息;函数列表中则列出 TLC 函数,按字母顺序排列,对于调用到的 TLC 函数,列出其自身执行时间(self-time,不包括调用其他函数的时间)、所调用子函数的执行时间及两个时间的百分比和每个 TLC 函数的被调用次数等信息。TLC Profile Report 如图 18.2-20 所示。

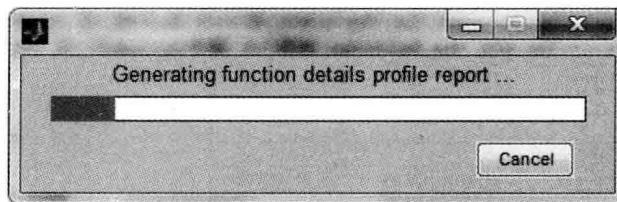


图 18.2-19 勾选 Profile TLC 选项后的进度条

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

| Name                             | Time       | Calls  | Time/call        |
|----------------------------------|------------|--------|------------------|
| <a href="#">codegenentry.tlc</a> | 4.49282880 | 100.0% | 1 4.492828800000 |
| <a href="#">ert.tlc</a>          | 4.49282880 | 100.0% | 1 4.492828800000 |
| <a href="#">commonsetup.tlc</a>  | 2.66761710 | 59.4%  | 1 2.667617100000 |

图 18.2-20 Profile Report

单击函数列表中的某个函数(蓝色函数名自身即为超链接),将打开其详细 profile 信息,除了调用次数和执行时间外,还会给出该函数的父函数和子函数列表,如图 18.2-21 所示。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

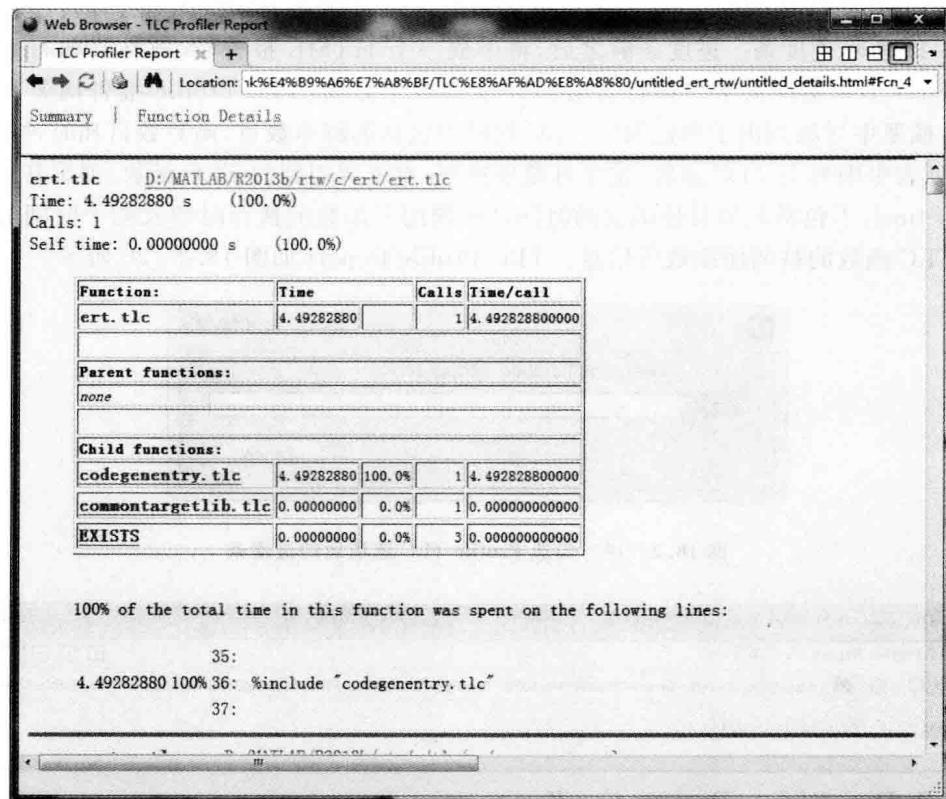


图 18.2-21 单击某个 TLC 文件后链接到文件细则

在图的表格下方,给出文字信息告知用户最耗时的函数。用户可以知道开发 TLC 代码(如支持代码生成的自定义 S 函数模块)的过程中哪些函数是提高运行速率的关键对象。例如,尽量少地使用 TLC 函数 EXISTS,它对于一个变量或域的存在性的确认是十分耗时的;再如,尽可能多地内联简单的 TLC 函数,对于少行数(特别是单行)TLC 代码,将其直接内联在程序中要比函数化省时得多,当然要对可读性在内的各种因素综合考虑、判断后决定 TLC 程序编写方案。

## 18.3 为 S 函数编写 TLC 文件

Simulink 中提供的模块虽然种类繁多,功能强大,但有时也不能完全满足用户在各个行业应用中的需求。S 函数是直接被 Simulink 引擎驱动的自定义接口,用户可自定义模块用于仿真模型,并使用 TLC 语言编写 S 函数的模块目标文件(又称模块 TLC 文件),以支持其生成代码的功能。

### 18.3.1 支持代码生成的 S 函数

第 10 章“S 函数”,已经明确说明了只有 C MEX S 函数和 Level 2 M S 函数才支持代码生成功能,并且这个功能要求 S 函数有配套的 TLC 文件。这个 TLC 文件就是代码生成流程中所需的模块级 TLC 目标文件。为了使 S 函数支持代码生成,在其编写模块级 TLC 文件之

前,先了解一下 S 函数应该如何做好数据中转工作,如何提供数据接口,以便让 TLC 能够接收 S 函数的参数。本节以使用普遍的 C Mex S 函数为例。

带有参数的 S 函数首先从 GUI 上获取用户的输入作为参数,通过 C 语言的宏将 GUI 控件上的值读入 S 函数,再通过 S 函数的子方法 mdlRTW 将参数值写入到模型的 rtw 文件中,使得 TLC 文件能够获取这些参数的值,最终展开到生成代码中合适的位置去。代码生成流程及参数数据的流向如图 18.3-1 所示。

### 1. C Mex S 函数获取 GUI 参数的宏

GUI 制作的参数对话框作为 S 函数的最前端能够提供给使用者方便简洁的操作方式。常用的 Simulink Mask 控件包括 Edit、popup、radiobutton、check-box 和 pushbutton 等(可以参考第 11 章“模块的封装”)。不同控件所容纳的数据类型不同,Edit 中可以输入数值也可以输入字符串,从 popup/radiobutton 中既可以提取表示所选项目的编号也可以提取所选项目的字符串内容。读者朋友可以根据使用场景不同,有选择地提取数据类型。在 C Mex S 函数中,首先定义能够直接或间接获取 GUI 控件的宏,主要介绍以下几种方式:

(1) 获取 Edit 中的数值

```
#define PARAM(S)(mxGetScalar(ssGetSFcnParam(S,g_param)))
```

g\_param 为 Edit 控件的参数在 GUI 控件中的索引号,首先通过 ssGetSFcnParam 宏函数获取指向 Edit 控件中参数的指针,再使用 mxGetScalar 宏函数获取指针指向地址的数值。

(2) 获取 Edit 中的数组

```
#define PARAM(S) mxGetNumberOfElements(ssGetSFcnParam(S, g_param))
```

在使用 ssGetSFcnParam 宏函数获取 Edit 控件的指针之后,使用 mxGetNumberOfElements 获取此数组的数据长度,以传递相同长度的数据到 rtw 文件。

(3) 在 Edit 中获取字符串

```
#define Param(S)(ssGetSFcnParam(S,g_param))
```

仅在宏定义中获取指向参数的指针,在 S 函数的 mdlRTW 函数中再进行字符串传递。

(4) 获取 Popup/radiobutton 所选项目的字符串

```
#define Param(S)(mxArrayToString(ssGetSFcnParam(S,g_param)))
```

在使用 ssGetSFcnParam 获取控件的指针后,使用 mxArrayToString 宏函数获取此参数内容的字符串。注意此时 g\_param 所代表的控件的 Evaluate 属性不要勾选,否则有可能引起 MATLAB 崩溃。

(5) 获取 popup/radiobutton 所选项目的索引号

与(1)相同。

(6) 获取 Check-box 的值

与(1)相同。

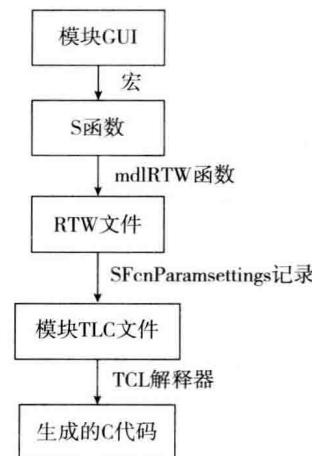


图 18.3-1 带有参数的 S 函数数据流程图

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

## 2. C Mex S 函数的 mdlRTW 函数

mdlRTW 函数是专为支持代码生成的 S 函数设计的, 不支持仅用于仿真的 C Mex S 函数, 其功能传递 S 函数的参数数据到 rtw 文件(这些参数必须被设置为 not tunable)。它必须被包含在以下这个预处理语句中:

```
# if defined(MATLAB_MEX_FILE)
# endif
```

则函数定义体呈现为:

```
# if defined(MATLAB_MEX_FILE)
# define MDL_RTW
static void mdlRTW(SimStruct * S)
{}
```

函数内实现两个作用: 一是通过定义的宏获取 GUI 上的数据, 另一个是将这些输入写入 rtw 文件中。将 S 函数中通过宏获取的参数数据写入 rtw 文件使用函数 ssWriteRTWParamSettings, 其格式为:

```
int_T ssWriteRTWParamSettings(SimStruct * S, int_T nParamSettings, int_T paramType,
const char_T * settingName, ...)
```

其参数列表包括:

SimStruct \* S——SimStruct 结构体指针, 指向当前模块;  
int\_T nParamSettings——要写入 rtw 文件中参数的个数;  
int\_T paramType——参数的数据类型;  
const char\_T \* settingName——参数名, 当存在多个参数时, paramType 和 settingName 按顺序成对书写, 对数与该 S 函数的参数个数相同。

...——针对不同类型的参数, 其后需要跟着一些不同的设置参数, 如写入 RTW 后以什么类型存储等信息设置, 多个参数可以通过一个 ssWriteRTWParamSettings 函数实现。

此处举一些不同数据类型参数传入到 rtw 中的实例(仅将 paramType、settingName 及之后的设置参数列出, GET\_PARAM 表示获取 GUI 参数数据的宏)。

① 将一个整数以 uint8 类型写入到 rtw 文件:

```
uint8_T c_int = GET_PARAM(S);
SSWRITE_VALUE_DTYPE_NUM,"r_int",&c_int,DTINFO(SS_UINT8, COMPLEX_NO),...
```

上述代码将 S 函数内的 c\_int 变量的值作为无符号 8 位整数写入到 rtw 文件中, 变量名为 r\_int。SSWRITE\_VALUE\_DTYPE\_NUM 表示写入的是一个复数形式, 此处仅使用了实数部分, 虚数部分未使用。此方式同样适用于其他数据类型(double, int6 等)的写入操作。

② 将一个字符串写入到 rtw 文件:

```
char_T * c_str = (char_T *)malloc(mxGetNumberOfElements(GET_PARAM(S)) + 1);
boolean_T flag = mxGetString(GET_PARAM(S), c_str, mxGetNumberOfElements(GET_PARAM(S))
+ 1);
SSWRITE_VALUE_STR,"r_str ",c_str,...
```

上述代码首先开辟新的存储空间, 通过宏函数获取用户输入 GUI 的字符串拷贝到新开辟的空间中, 并用指针 c\_str 指向此字符串数组。最后一句代码片断将 c\_str 这个字符串写入到 rtw 文件中, 变量名为 r\_str。SSWRITE\_VALUE\_STR 则表示以字符串类型写入 rtw 文件。

③ 将一个数组(数组元素类型以 int8 为例)写入到 rtw 文件:

```
int8_T * c_vec = mxGetData(GET_PARAM(S));
SSWRITE_VALUE_VECT,"r_vec",c_vec,mxGetNumberOfElements(GET_PARAM(S)),...
```

上述代码中首先使用 `mxGetData` 函数结合获取参数的宏函数返回一个指向 Edit 框中数组的指针,再将这个指针指向的数组元素以向量形式写入 rtw 文件,变量名保存为 `r_vec`。

以上 3 种写入 RTW 的方式可用于将 Edit、Popup、check-box 和 radiobutton 等带有变量,能够传递数据的 GUI 控件上的用户输入值写入到 TLC 中以备代码生成使用。如此便完成了数据的传递,存储在 rtw 中的数据只等 TLC 文件来获取。

### 18.3.2 模块 TLC 文件的构成

模块 TLC 文件具体实现一个 S 函数是如何生成代码的,与 S 函数的构成类似,TLC 文件也由多个执行顺序有先后的子函数构成。模块 TLC 文件包括如表 18.3-1 所列,子函数按照执行顺序依次排列。

表 18.3-1 模块 TLC 函数的子函数

| 子函数                                    | 输出    | 功能说明   |
|--|-------|--|
| BlockInstanceSetup<br>(block, system)  | 不产生输出 | 同种类的模块存在多个时,每一模块都会执行一次此函数,可以将模块共同的操作或特例的操作写入此函数中   |
| BlockTypeSetup ( block,<br>system)     | 不产生输出 | 同类模块即使存在多个也只执行此函数一次,可以将同一类模块共同地且执行一次的操作写入此函数中,也可以不实现此函数  |
| Enable(block, system)                  | 产生输出  | 为模型中非虚子系统创建 Enable 函数,并将使能某功能的代码生成在该函数中  |
| Disable(block, system)                 | 产生输出  | 为模型中非虚子系统创建 Disable 函数,并将禁止某功能的代码生成在该函数中   |
| Start(block, system)                   | 产生输出  | 为模块中仅执行一次的函数,内部代码会生成到 <code>model_initialize()</code> 函数中,通常将模型各变量、状态或硬件外设初始化的代码写在此函数中,因为它们不需要重复执行 |
| InitializeConditions(block,<br>system) | 产生输出  | 此函数里的代码通常也用于初始化某个子系统的状态变量,但是它不一定仅执行一次,而是在当前模块所在子系统每次被使能时都会执行                                       |
| Outputs(block, system)                 | 产生输出  | 用于编写模块计算输出的代码,并将其生成到 <code>model_step()</code> 函数中   |
| Update(block, system)                  | 产生输出  | 用于编写每个步长更新模块状态变量的代码,其内容生成到 <code>model_update()</code> 中   |
| Derivatives(block, system)             | 产生输出  | 用于计算模块连续变量的函数,其内容生成到 <code>model_Derivatives()</code> 函数中  |
| Terminate(block, system)               | 产生输出  | 此函数用于自定义代码用,可以存储数据,释放内存,复位硬件寄存器等操作。此函数内的代码将生成到 <code>MdlTerminate()</code> 中                       |

每个模块 TLC 文件并非必须囊括上面所有子函数,用户可以根据需要选择其中部分函数。每个子函数都有两个参数:`block` 和 `system`,分别表示当前函数所属的模块名和该模块所属的子系统名。每一个 S 函数的 TLC 文件都拥有相同的函数名,即使多个模块的 TLC 文件同时存在也不会出现访问冲突,这是因为这些子函数都是多态的,它们在执行时才由 TLC 决定调用哪个模块的子函数。

产生输出的函数不需用户指定即可将内部代码创建成文件流生成到特定的位置;不产生输出的函数中则需要用户编写代码创建文件流再指定其生成的位置。产生输出的函数在其函数参数列表后有 `Output` 标注,不产生输出的函数参数列表后则以 `void` 标注,如:

```
% function BlockTypeSetup(block, system) void
% function Start(block, system) Output
```

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

每个模块 TLC 文件都以 %implements 命令开头, 其后注明 S 函数名及生成代码的目标语言。如: %implements sfun\_c\_filter "C"。

C 文件的开头通常是头文件包含、宏定义及函数声明。这些内容可以写入 BlockType-Setup(block, system) 或 BlockInstanceSetup(block, system), 用以生成预处理内容或函数声明, 如:

```
% function BlockTypeSetup(block, system) void
%<LibAddToCommonIncludes("mcu.h")>
% % Place a #define in the model's header file
% openfile buffer
#define A2D_CHANNEL 0
%closefile buffer
%<LibCacheDefine(buffer)>
% openfile buffer
extern real_T mydata;
%closefile buffer
%<LibCacheExtern(buffer)>
% % Place function prototypes in the model's header file
% openfile buffer
void start_a2d(void);
void reset_a2d(void);
%closefile buffer
%<LibCacheFunctionPrototype(buffer)>
%endfunction
```

上述代码中, LibAddToCommonIncludes(incFileName) 将头文件包含语句生成到 model.h 中, 如图 18.3-2 所示。

```
#ifndef RTW_HEADER_sfun_filter_model_h_
#define RTW_HEADER_sfun_filter_model_h_
#ifndef sfun_filter_model_COMMON_INCLUDES_
#define sfun_filter_model_COMMON_INCLUDES_
#include <stddef.h>
#include "rtwtypes.h"
#include "mcu.h"
#endif /* sfun_filter_model_COMMON_INCLUDES_ */
```

图 18.3-2 头文件包含代码生成

LibCacheDefine(buffer) 将文件流中宏定义内容生成到 model\_private.h 中, LibCache-FunctionPrototype(buffer) 将 buffer 表示的文件流内容生成到 model\_private.h 中, 其内容通常为函数的外部声明; LibCacheExtern(buffer) 将文件流中的外部变量声明代码也生成到同一头文件中, 如图 18.3-3 所示。

Start 子函数中写入在程序运行过程中仅执行一次的代码, 如全局变量的初始化、硬件外设寄存器的初始化等。如以下代码:

```
% function Start(block, system) Output
/* Initialize the extern variable and hardware */
mydata = 0.0;
mcu_init();
%endfunction
```

上述的 TLC 代码生成的 C 代码存放在 model\_initialize() 函数的自定义代码段, 如图 18.3-4 所示。

```

#ifndef RTW_HEADER_sfun_filter_model_private_h_
#define RTW_HEADER_sfun_filter_model_private_h_
#include "rtwtypes.h"
#define A2D_CHANNEL 0
#ifndef __RTWTYPES_H__
#error This file requires rtwtypes.h to be included
#else
#define TMWTYPES_PREVIOUSLY_INCLUDED
#error This file requires rtwtypes.h to be included before tmwtypes.h
#endif
#endif
/* TMWTYPES_PREVIOUSLY_INCLUDED */
/* __RTWTYPES_H__ */

extern real_T mydata;
void start_a2d(void);
void reset_a2d(void);

#endif /* RTW_HEADER_sfun_filter_model_private_h_ */

```

图 18.3-3 头文件包含宏定义及函数声明生成

```

/* Model initialize function */
void model_initialize(void)
{
    /* initialize error status */
    rtmSetErrorStatus(sfund filter model M, (NULL));
    /* Start for S-Function (sfund_c_filter): 'Root/filter_cmax' */
    /* Initialize the extern variable and hardware */
    mydata = 0.0;
    mcu_init();
}

```

图 18.3-4 自定义初始化代码

Start 函数体中的 C 代码可以是直接调用函数,也可以是对寄存器的操作。代码生成的位置都是 model\_initialize() 函数的自定义代码段,其他代码段由 Simulink Coder 决定。model\_initialize() 函数中用户代码段可以细分为 4 个子段,按照顺序从前到后依次为 header、declaration、execution 和 trailer。详见图 18.3-5。

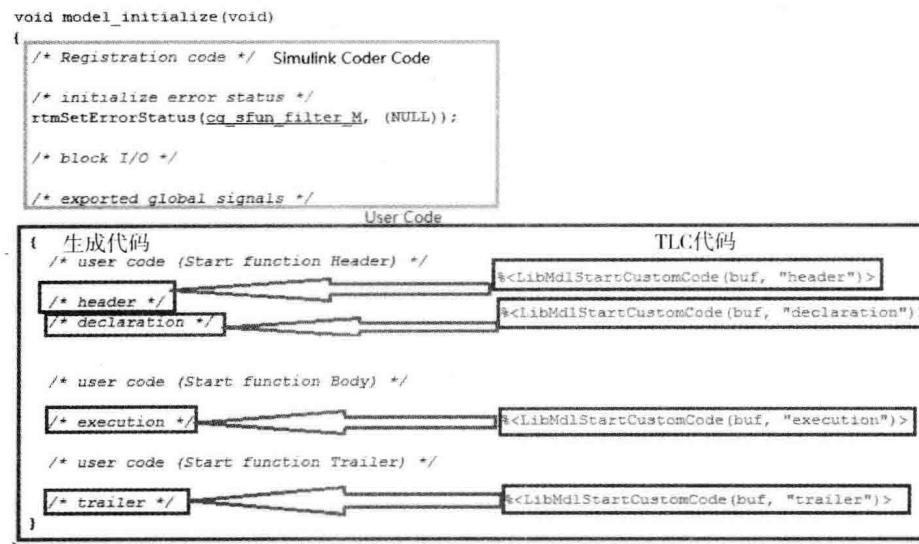


图 18.3-5 model\_initialize() 函数的用户代码段划分

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

使用 TLC 函数 LibMdlStartCustomCode(buffer, section) 可以将文件流 buffer 生成到 model\_initialize() 函数用户代码段中对应的字段中。如果不使用 LibMdlStartCustomCode 进行指定，则默认将代码追加到 execution 段中去。例如：

```
% function Start(block, system) Output
    % openfile buf
    /* Initialize the extern variable and hardware */
    % closefile buf
    % <LibMdlStartCustomCode(buf, "header")>
    % openfile buf
    mydata = 0.0;
    % closefile buf
    % <LibMdlStartCustomCode(buf, "declaration")>
    % openfile buf
    mydata = 1.0;
    % closefile buf
    % <LibMdlStartCustomCode(buf, "execution")>
    % openfile buf
    mydata = 2.0;
    % closefile buf
    % <LibMdlStartCustomCode(buf, "trailer")>
    mcu_init();
% endfunction
```

上述 Start 函数生成的代码如图 18.3-6 所示。

```
/* Model initialize function */
void sfun_filter_model_initialize(void)
{
    /* initialize error status */
    rtmSetErrorStatus(sfun_filter_model_M, (NULL));
    {
        /* user code (Start function Header) */
        /* Initialize the extern variable and hardware */
        mydata = 0.0;

        /* user code (Start function Body) */
        mydata = 1.0;

        /* Start for S-Function (sfun_c_filter): 'Root/filter_cmax' */
        mcu_init();

        /* user code (Start function Trailer) */
        mydata = 2.0;
    }
}
```

图 18.3-6 model\_initialize() 函数内用户代码段指定实例

Update 子函数与 Derivates 子函数的功能与 S 函数的 mdlUpdate 和 mdlDerivates 函数类似，对离散/连续状态变量进行计算和更新。

Output 子函数基本上是每个用于计算功能的模块 TLC 文件都必须的，负责采集模块输入/输出端口、参数及工作向量的数据并生成算法代码，用于目标平台或硬件。Output 子函数的代码生成在 model\_step 函数中。如下面代码：

```
% function Outputs(block, system) Output
    /* call existed function to calculate */
    mydata = custom_algorithm(signal_in);
% endfunction
```

生成的代码如图 18.3-7 所示。

```
/* Model step function */
void model_step(void)
{
    /* S-Function (sfun_c_filter): 'Root>/filter_cmex' */
    /* call existed function to calculate */
    mydata = custom_algorithm(signal_in);
```

图 18.3-7 TLC Output 子函数代码反映到 C model\_step() 函数中

Output 子函数中通常都要使用到模块的输入/输出端口数据,参数数据和工作向量数据等,这时需要使用 TLC 内建函数:LibBlockInputSignal、LibBlockOutputSignal、LibBlockDWork 和 LibBlockParameter 等函数。其参数列表都相同,以 LibBlockInputSignal 为例说明:

`LibBlockInputSignal(portIdx, ucv, lcv, sigIdx)`

其中:portIdx——端口的索引号(LibBlockDWork 中此处为工作向量名);

ucv——用户自定义循环控制变量,通常为空“”;

lcv——生成循环代码的阈值控制变量;

sigIdx——信号的维数索引号。

这 4 个函数的使用方式如下:

```
% assign u = LibBlockInputSignal(0, " ", lcv, sigIdx)
% assign y = LibBlockOutputSignal(0, " ", lcv, sigIdx)
% assign x = LibBlockDWork(dwork, "", lcv, sigIdx)
% assign k = LibBlockParameter(gain, "", lcv, sigIdx)
```

如将 Output 子函数的内容书写为以下形式,调用 custom\_algorithm 函数实现模块的输出计算:

```
% function Outputs(block, system) Output
    % assign u = LibBlockInputSignal(0, " . ", 0)
    % assign y = LibBlockOutputSignal(0, " . ", 0)
    /* call existed function to calculate */
    % <y> = custom_algorithm(% <u>);
% endfunction
```

Ctrl+B 组合键启动代码生成,生成的 model\_step 函数如图 18.3-8 所示。

```
/* Model step function */
void model_step(void)
{
    /* S-Function (sfun_c_filter): 'Root>/filter_cmex' */
    /* call existed function to calculate */
    sfun filter model Y.Out1 = custom_algorithm(sfun filter model U.In1);
```

图 18.3-8 输入/输出端口为模型端口名

生成的代码中输入/输出信号都以模型中输入/输出端口的信号名作为变量展开,通过对信号存储类型的选择优化生成代码的变量形式,此处不作赘述。

Terminate 子函数的有无受模型 Configuration Parameter 里面 Code Generation 页面的 Code Interface 参数的控制,参数勾选框如图 18.3-9 所示。

勾选此项之后,可以在 Terminate 子函数中定义用户代码,主要用于用户执行程序结束前的数据保存,硬件状态复位,内存清除等操作。Terminate 子函数内部也划分为 header、declaration、

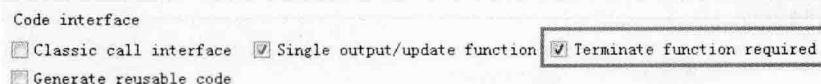


图 18.3-9 Configuration Parameter 中参数勾选框

execution 和 trailer 4 个字段, 使用 LibMdlTerminateCustomCode(buffer, location) 函数将自定义代码的文件流定位到 4 个字段中。4 个字段及其对应的 TLC 代码如图 18.3-10 所示。

```
void model_terminate(void)
{
    /* user code (Terminate function Header) */
    /* header */             %<LibMdlTerminateCustomCode(buf, "header")>
    /* declaration */        %<LibMdlTerminateCustomCode(buf, "declaration")>

    /* user code (Terminate function Body) */

    /* execution */          %<LibMdlTerminateCustomCode(buf, "execution")>

    /* user code (Terminate function Trailer) */

    /* trailer */            %<LibMdlTerminateCustomCode(buf, "trailer")>
}
```

图 18.3-10 Terminate 函数的用户代码段

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

### 18.3.3 模块 TLC 函数实例

以第 10 章“S 函数”的自定义滤波器 S 函数为例,为其编写模块 TLC 文件,以生成具有同样算法功能的 C 代码。filter cmex 模块图标上标注有滤波器算法计算公式,双击模块之后可以打开参数对话框对滤波系数进行编辑或修改。输入/输出端口为 in、out 模块以简化模型,生成代码时便于观察自定义 TLC 文件的功能。滤波器模型及 filter cmex 模块的参数对话框如图 18.3-11 所示。

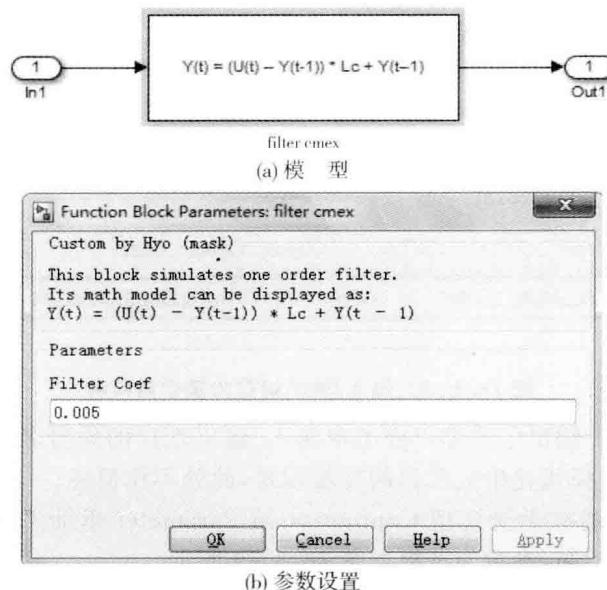


图 18.3-11 自定义一阶滤波器模块

上述模块的 C Mex S 函数中增加获取 GUI 参数的宏和将其写入 rtw 文件的 mdlRTW 函数，并设置 Dwork 变量的名称之后，其 C 代码为：

```
#define S_FUNCTION_NAME sfun_c_filter
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

#define COEF_IDX 0
#define PARAM_NUM 1
#define COEF(S) mxGetScalar(ssGetSFcnParam(S,COEF_IDX))

/* Function: mdlInitializeSizes
=====
* Abstract:
*   Setup sizes of the various vectors.
*/
static void mdlInitializeSizes(SimStruct * S)
{
    ssSetNumSFcnParams(S, 1);
    if(ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

    if(! ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if(! ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);

    ssSetNumDWork(S, 1);
    ssSetDWorkWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetDWorkName(S, 0, "dwork"); /* Dwork Name */

    ssSetNumSampleTimes(S, 1);

    /* set parameter untunable */
    ssSetSFcnParamNotTunable(S, 0);

    /* specify the sim state compliance to be same as a built-in block */
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);

    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
                    SS_OPTION_DISALLOW_CONSTANT_SAMPLE_TIME));
}

/* Function: mdlInitializeSampleTimes
=====
* Abstract:
*   Specifiy that we inherit our sample time from the driving block.
*/

```

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```

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

#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions
= = = = = = = = = = = = = = = = = = = = = = = = = = =
* Abstract:
*     Initialize both discrete states to one.
*/
static void mdlInitializeConditions(SimStruct * S)
{
    real_T * x = (real_T *) ssGetDWork(S,0);
    x[0] = 0.0; // initial to 0.0
}
/* Function: mdlOutputs
= = = = = = = = = = = = = = = = = = = = =
* Abstract:
*     y = (u - x) * coef + x
*/
static void mdlOutputs(SimStruct * S, int_T tid)
{
    int_T i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T y = ssGetOutputPortRealSignal(S,0);
    int_T width = ssGetOutputPortWidth(S,0);
    real_T x = (real_T *) ssGetDWork(S,0);
    real_T Lc = COEF(S); // floating point datatype

    for(i = 0; i < width; i++)
    {
        y[i] = (*uPtrs[i] - x[i]) * Lc + x[i];
    }

    /* save the current output as the DWork Vector */
    for(i = 0; i < width; i++)
    {
        x[i] = y[i];
    }
}

/* Function: mdlTerminate
= = = = = = = = = = = = = = = = = = = = =
* Abstract:
*     No termination needed, but we are required to have this routine.
*/
static void mdlTerminate(SimStruct * S)
{
}

```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```
# define MDL_RTW
void mdlRTW(SimStruct * S)
{
    /* get parameter */
    real_T c_coef = COEF(S);
    /* write parameter into rtw file */
    if(! ssWriterTWPParamSettings(S, 1,
        SSWRITE_VALUE_DTYPE_NUM, "r_coef", &c_coef, DTINFO(SS_DOUBLE, COMPLEX_NO)))
    return;
    /* dwork is automatically written into rtw file */
}

#ifndef MATLAB_MEX_FILE    /* Is this file being compiled as a MEX - file? */
#include "simulink.c"      /* MEX - file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Dwork 变量自动被写入 rtw 文件, 无须再编写 C 代码实现, 其值可以直接在 tlc 文件中获取。有了上面讲述的知识, 可以编写此 S 函数的 tlc 文件如下:

```
% implements sfun_c_filter "C"
% % Function: blockTypeSetup
= = = = = = = = = = = = = = = = = = = =
% %
% % Purpose:
% %     Add some macro defines .
% %
% function BlockTypeSetup(block, system) void

% endfunction

% % Function: Start
= = = = = = = = = = = = = = = = = = = =
% %
% % Purpose:
% %
% %     these code will appear at model.c initialization function
% %
% function Start(block, system) Output
    /* If need user can add custom initialize code here */
% endfunction

% % Function: Outputs
= = = = = = = = = = = = = = = = = = = =
% %
% % Purpose:
% %
% %     these code will appear at model.c step function
% %
% function Outputs(block, system) Output
```

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

```
% assign t_coef = SFcnParamSettings.r_coef
% assign rollVars = ["U", "Y", "DWork"]
% roll sigIdx = RollRegions, lcv = RollThreshold, block, "Roller", rollVars
% assign u = LibBlockInputSignal(0, "", lcv, sigIdx)
% assign y = LibBlockOutputSignal(0, "", lcv, sigIdx)
% assign x = LibBlockDWork(dwork, "", lcv, sigIdx)
/* Calculate the filter result */
% <y> = (% <u> - % <x>) * % <t_coef> + % <x>;
% <x> = % <y>;
%
% endroll
% endfunction
```

在 `ert.tlc` 的系统目标文件作用下, 编译图 18.3-11 所示模型, 生成代码如图 18.3-12 所示。

```
/* Model step function */
void sfun_filter_model_step(void)
{
    /* S-Function (sfun_c_filter): 'Root/filter_cmex' */

    /* Calculate the filter result */
    sfun filter model Y.Out1 = (sfun filter model U.In1 -
        sfun filter model DW.filtercmex_dwork) * 0.005 +
        sfun filter model DW.filtercmex_dwork;
    sfun filter model DW.filtercmex_dwork = sfun filter model Y.Out1;

    /* Model initialize function */
    void sfun_filter_model_initialize(void)
    {
        /* Registration code */

        /* initialize error status */
        rtmSetErrorStatus(sfun filter model M, (NULL));

        /* states (dwork) */
        (void) memset((void *)&sfun filter model DW, 0,
                      sizeof(DW sfun filter model T));

        /* external inputs */
        sfun filter model U.In1 = 0.0;

        /* external outputs */
        sfun filter model Y.Out1 = 0.0;
    }
}
```

图 18.3-12 一阶滤波器生成代码

可见参数 0.005 直接内敛展开在代码中, 而输入/输出端口及 Dwork 变量都生成了 Simulink Coder 内部的结构体变量, 视觉上多少显得繁冗。接下来对它们进行 Exported Global 存储类型的设置。输入/输出信号在信号属性对话框里设置即可; Dwork 变量则在 S 函数 `mdlInitializeSizes()` 函数中使用 `SimStruct` 宏函数设置, 将设置 Dwork 变量的代码更改为以下样式:

```
ssSetNumDWork(S, 1);
ssSetDWorkWidth(S, 0, DYNAMICALLY_SIZED);
ssSetDWorkName(S, 0, "dwork");
/* Identifier */
ssSetDWorkRTWIdentifier(S, 0, "x");
/* Type Qualifier */
ssSetDWorkRTWTypeQualifier(S, 0, "volatile");
/* Storage class */
ssSetDWorkRTWStorageClass(S, 0, SS_RTW_STORAGE_EXPORTED_GLOBAL);
```

再次使用 mex 命令编译 C 文件生成新的 cmex 文件后, Ctrl+B 启动模型编译, 再次生成的代码可读性增强很多, 如图 18.3-13 所示。

```

/* Model step function */
void sfun_filter_model_step(void)
{
  /* S-Function (sfu_c_filter): 'Root/filter_cmex' */

  /* Calculate the filter result */
  out = (in - x) * 0.005 + x;
  x = out;
}

/* Model initialize function */
void sfun_filter_model_initialize(void)
{
  /* Registration code */

  /* initialize error status */
  rtmSetErrorStatus(sfun_filter_model_M, (NULL));

  /* block I/O */

  /* exported global signals */
  out = 0.0;

  /* states (dwork) */
  /* exported global states */
  x = 0.0;

  /* external inputs */
  in = 0.0;

  /* Start for S-Function (sfu_c_filter): 'Root/filter_cmex' */
  /* If need user can add custom initialize code here */
}

```

图 18.3-13 一阶滤波器各变量简化后生成的代码

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

# 第 19 章

## 基于 TSP 的直流电机控制设计

**引言:**Simulink 是 MATLAB 最重要的组件之一,它提供一个动态系统建模、仿真、代码生成和综合分析的集成环境。MBD(Model Based Design)是基于 Simulink 模型,结合仿真与代码自动生成技术进行嵌入式软件开发的方法。使用 MATLAB/Simulink 软件,在计算机上搭建模型进行仿真或实时仿真,从而实现控制算法的开发和验证,模型仿真之后,使用 TSP 及工具链一键生成代码下载到目标板中进行实机验证。

### 19.1 TSP 是什么

基于模型设计是人们在抽象出数学模型的认知基础上,进行 Simulink 建模,将数学模型转换为 Simulink 模型,经过反复的验证和修改,证明算法仿真结果和数据类型都符合要求后,再通过 Simulink Coder/Embedded Coder 针对某种特定硬件进行代码自动生成的技术。MBD 技术将软件工程师从繁重的代码编写任务中解放出来,从而将更多的精力投入到控制算法设计和代码优化方面。

相对于控制算法代码,嵌入式软件的驱动代码则不具有直观的数学逻辑性,并不能直接抽象为数学公式进行模型建立与仿真。驱动代码是通过将值写入到芯片寄存器中,使周边外设电路能够按照指定方式工作的代码。由于无法直观建立数学模型,因此不能够建立 Simulink 模型以自动生成代码。嵌入式工程师往往手写驱动代码,再跟由模型生成的算法代码进行融合后形成一个完整的工程。如此一来,MBD 的优势体现则被打了折扣。为了进一步提高开发效率,针对 Cypress(原 Spansion)的 ARM 芯片 MB9BF568R,提供了便于嵌入式工程师在 Simulink 环境中配置并可以自动生成驱动代码的工具——Target Support Package(简称 TSP)。此款 TSP 实现了完全基于模型开发流程的电机控制嵌入式开发,并支持从代码到硬件实现过程的自动化。主要提供两个组件,Simulink 工具箱形式提供的外设驱动库 Peripheral Simulink Library(简称 PSL)和 Toolchain。其结构图如图 19.1-1 所示,下载地址:<http://www.spansion.com/JP/Support/microcontrollers/development-environment/Pages/TargetSupportPackage.aspx>。

PSL 为嵌入式工程师提供 Cortex M4 的外设驱动模块库,完全支持 Simulink 环境,能够高效实现快速原型化,让开发者很快实装算法模型生成的代码。PSL 库中提供了 ADC、GPIO、MFT 和 MFS 4 个外设模块库和两个通用机能模块 PDL\_USER 模块和 System 模块及一个 Interrupt 模块。PDL\_USER 模块控制外设的使能与否,并根据配置生成 C 文件,其中包含众多资源使能与否的宏定义;System 模块,用于用户设定系统时钟工作方式及自定义 Mainloop 中的代码生成;Interrupt 模块用于用户选择中断类型并连接触发子系统以将其内模型生成代码作为所选中断的服务函数。

开发者可通过模块的对话框配置得到数模转换、多功能计时器、串口通信和通用输入/输出端口的驱动代码,可以直接编译链接下载到 MCU 中运行;并且这些模块都提供与算法层数

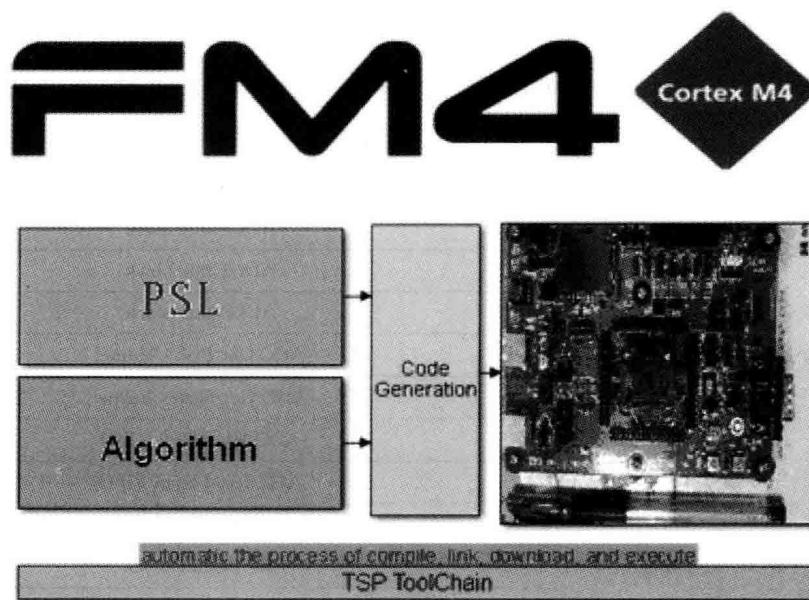


图 19.1-1 TSP 结构示意图

据的传输接口,建模、生成代码之后可以实现驱动代码与算法层代码的自动整合。Simulink Library Browser 中的工具箱及包含的模块,如图 19.1-2 所示。

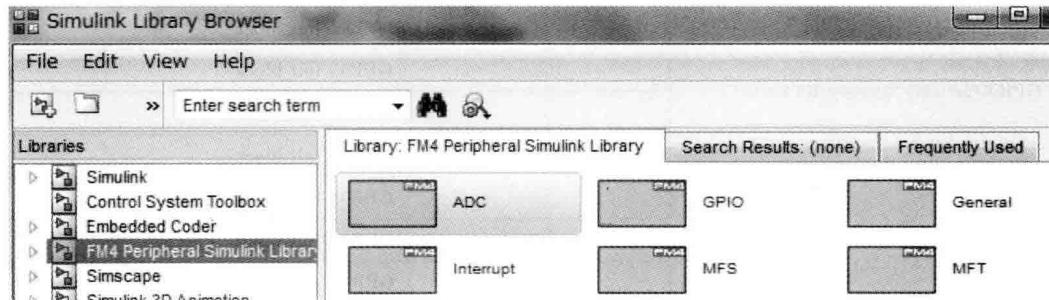


图 19.1-2 PSL 包含的驱动模块

User Defined Algorithm 就是 TSP 的用户自定义 Simulink 环境下的算法模型。

工具链为 TSP 的使用者提供从驱动 GUI 配置后所有自动化流程,对应快捷实现开发流程。它将代码自动生成、IDE(集成开发环境)的启动、工程创建、工程文件目录自动刷新和编译工作全部集成,TSP 用户在使用 PSL 设计完毕之后,可以一键得到能够下载到芯片中的目标文件。

图 19.1-1 所示最右边的图片表示本 TSP 对应的载有 Cortex M4 系列芯片 MB9BF568R 的板子,是最终的目标文件下载的载体。此解决方案适用于各种芯片,各半导体厂商提供针对其 MCU 产品的 TSP。

嵌入式开发者使用此 TSP 产生的驱动代码,可以将自定义的算法模型与其结合,并通过工具链自动拷贝文件及 IDE 工程创建编译等过程,直接下载到实机中进行验证。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

## 19.1.1 PSL 的构成与使用

PSL 包含图 19.1-2 所示 6 大模块，每个模块的子模块构成如表 19.1-1 所列。

表 19.1-1 PSL 所包含的模块列表

| 模 块                            | 子模块                    |
|--------------------------------|------------------------|
| ADC(Analog-Digital Convertor)  | ADC0 Init block        |
|                                | ADC1 Init block        |
|                                | ADC2 Init block        |
|                                | ADC Get Prio Channel   |
|                                | ADC Get Scan Channel   |
|                                | ADC Read Prio FIFO     |
|                                | ADC Read Scan FIFO     |
| MFT(Multi-Functional Timer)    | MFT0 Init block        |
|                                | MFT1 Init block        |
|                                | MFT ICU GetCaptureData |
| GPIO(General-Purpose IO ports) | GPIO0 Init block       |
|                                | GPIO1 Init block       |
|                                | GPIO2 Init block       |
|                                | GPIO3 Init block       |
|                                | GPIO4 Init block       |
|                                | GPIO5 Init block       |
|                                | GPIO6 Init block       |
|                                | GPIO7 Init block       |
|                                | GPIO8 Init block       |
|                                | GPIOE Init block       |
|                                | GPIO Read block        |
|                                | GPIO Write block       |
| MFS(Multi-Functional Serial)   | MFS Init block         |
| General                        | PDL_User block         |
|                                | System block           |
| Interrupt                      | Interrupt              |

部分模块包括初始化与读/写操作两种类型的子模块，其中初始化模块是根据硬件电路板上真实存在的单元数来确定的，MB9BF568R 芯片上有 3 个 ADC 单元，2 个 MFT 单元以及 0~8 和 E 这 10 个 I/O Port，初始化时按照每个个体进行模块的建立。这些模块在 Simulink 中按照表 19.1-1 所列层次的关系存储，如 ADC 模块包含 7 个驱动模块，如图 19.1-3 所示。

接下来以 ADC0 Init block 为例介绍模块的使用方式，首先从模块外观开始。

### 1. 子模块风格

每个子模块都以一个矩形框的形式存在于 Simulink 环境中，从 Simulink Library Browser 中拖曳子模块到模型编辑器中，如图 19.1-4 所示。

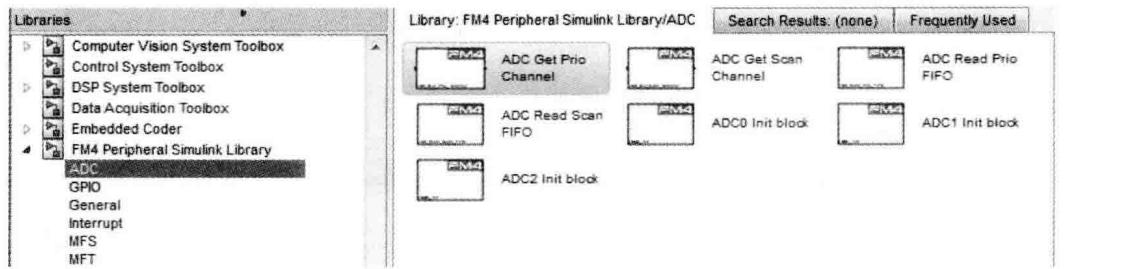


图 19.1-3 ADC 模块与其包含的子模块

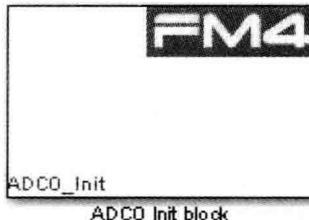


图 19.1-4 ADC0 Init block 模块

模块外观就是一个矩形框,矩形框的右上方显示 FM4 标志图像,左下方显示当前子模块所对应的名称及单元号。不同的 PSL 库模块以不同颜色显示表示不同的功能,如表 19.1-2 所列。

表 19.1-2 模块图标左下角颜色说明

| 颜色 | 功能           |
|----|--------------|
| 蓝色 | 初始化功能        |
| 红色 | 读/写功能        |
| 黑色 | 用户变量(无对应寄存器) |

在 FM4 的 TSP 中,黑色仅存在于 PDL\_USER 中,表示该模块中对应的变量并非实际硬件寄存器,而是自定义的一些宏变量,用来作为一些模块的使能开关变量。

根据子模块的功能不同会有不同个数的输入/输出端子,例如 MFT Init block 在配置了 ICU 通道之后,会根据 ICU 通道的开启数产生对应个数的输入端口;又如 GPIO 的 Read/Write block,将模型计算的值通过 I/O 口与 MCU 交互,那么这个模块则应该具有输入端口,端口数则根据所需要使用的引脚个数来定。另外,对于部分初始化功能模块,本身没有与算法的数据传递,则无输入/输出端口。

对于既有输入端口又有输出端口的模块,通常输入端口显示在子模块左侧,输出端口显示在子模块右侧,端口名称(port label)以对应硬件上的 ICU 通道号或 I/O 口引脚名等标注。

## 2. 子模块的用户图形界面

每个子模块提供与其名称对应的 GUI 界面,双击子模块可以启动模块的参数对话框。GUI 以 Edit、Popupmenu 和 check-box 等控件方式将寄存器的值和模式选择提供给开发者进行驱动工作模式的配置。GUI 最上端菜单栏显示模块的名字,FM4 TSP 字样下是模块的功能概述和版本号,以及使用者可以配置的参数。ADC0 Init block 的参数对话框如图 19.1-5 所示。

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

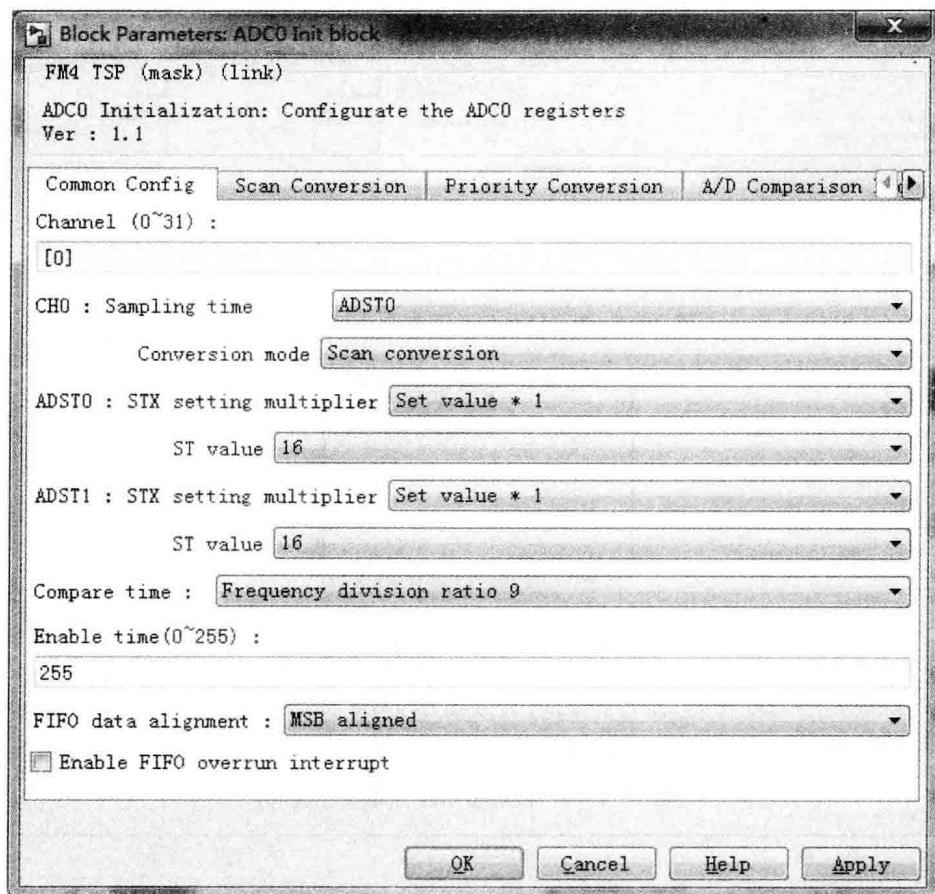


图 19.1-5 ADC0 Init 模块的参数对话框

ADC<sub>x</sub> Init block( $x = 0 \sim 2$ )GUI 有 5 个标签页,以 Common Config 为例进行介绍:

ADC 有 32 个共用的通道,对于 Unit0,用户希望使用哪个通道进行转换,填入通道号即可。例如,此处使用通道 0;对所选通道选择采样方式 ADST0 或者 ADST1;再确定转换方式: Scan 转换或者 Priority 转换。对于采样方式 ADST0 和 ADST1,需要配置其计算因子 STX 和 ST 值;根据计算公式可以得到具体的采样时间:

$$\text{Sample Time} = \text{HCLK Freq} \times 2 \times \{(ST + 1) \times STX + 3\}$$

其中 HCLK Freq 表示主时钟频率。

用户设置比较时钟的 Division Ratio,也就是分频系数,再根据下式计算可得:

$$\text{Comparison time} = \text{Compare clock cycle} \times 14$$

其中  $\text{Compare clock cycle} = \text{HCLK cycle} \times \text{Clock division ratio}$ (分频即周期加倍)。

A/D 转换的结果连同该结果所对应的通道都存放在 FIFO 寄存器中,此 GUI 还提供给用户设置 FIFO 寄存器中 12 位转换结果数据对齐方式。

最下方是 FIFO 溢出中断功能的开关,如果使用者需要中断函数,则勾选,代码自动生成时会使能该中断,并产生相应的中断服务函数框架,使用者可以进一步在其中详细定义中断动作。

ADC 初始化时具体关于设置扫描转换、优先转换和范围比较等内容不再详述。其他子模

块也是以简单 GUI 配置方式替代开发者手动编写驱动代码的工作。

### 3. 子模块自带帮助文档

为了使 TSP 的使用者能够很快理解 GUI 上每个参数的作用, 免去研读数千页文档的时间, PSL 的每一个子模块还提供了一个针对自身嵌入 Simulink 环境的 html 文档, 用户可以通过子模块参数对话框右下角的 Help(如图 19.1-5 所示)按钮启动它, 如图 19.1-6 所示。

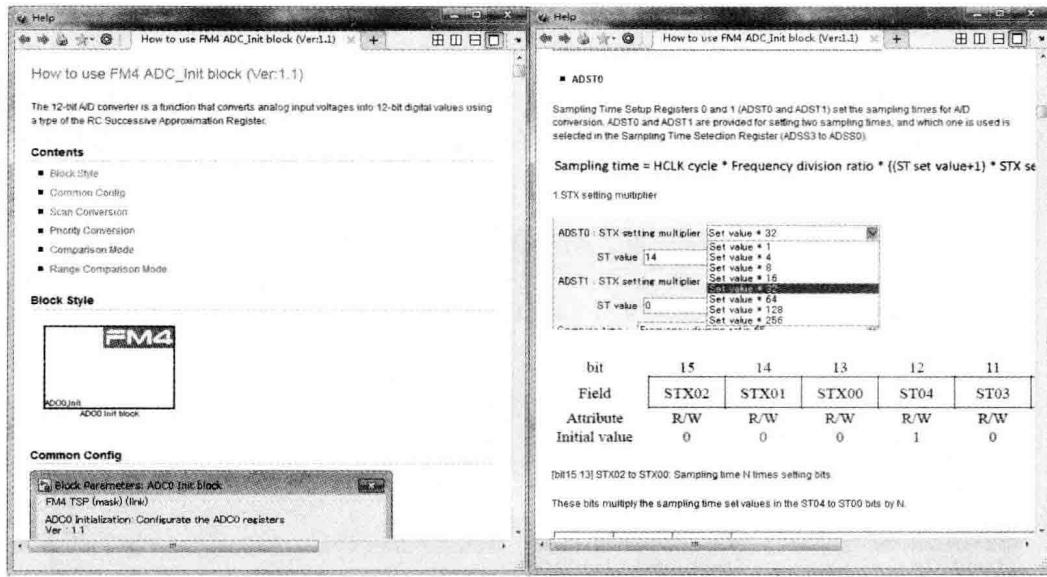


图 19.1-6 启动子模块的 Help 文档

Help 文档主要告诉用户如何使用当前的子模块, 各个 MCU 芯片的驱动部分构成大致相同, 对于经验丰富的工程师来说, 看到 Help 中的提示就足够明了驱动工作方式和配置方法。对于新手来说, 也能够降低学习整套文档的时间。

Help 文档首先概括了子模块所配置的工作模式主要完成哪些工作, 实现什么目的。其编写方法请参考第 12 章“Publish 发布 M 文件”。

Contents 标题下提供了跳转到每个页面说明起始位置的超链接, 方便用户查找需要的参数。Block Style 提供当前模块的外观图示。接下来就是对每个页面 GUI 的总括截图, 告知使用者紧接着要介绍每个参数的使用方法, 一些重要的寄存器对应关系及注意事项等。如图 19.1-6 右侧关于采样时间计算给出了设置该参数的 GUI 控件、计算公式、STX 的可能选择值及在寄存器中对应的位的位置。

有了 PSL 提供的各个子模块和丰富的说明文档, 使用者可以按照自己希望的工作方式设置驱动模型, 接下来的工作就由工具链负责流程的自动化。

## 19.1.2 工具链自动化流程

工具链提供了从用户模型系统级设置、代码生成、IDE 启动和创建项目工程的一系列动作。使用者构建完模型之后, 一键就可以替代繁杂的重复性劳动。除此之外, 工具链还对 PSL 进行检查, 当使用者进行了不恰当或者错误的配置时, 在代码自动生成阶段给出提示。工具链支持的功能如图 19.1-7 所示。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

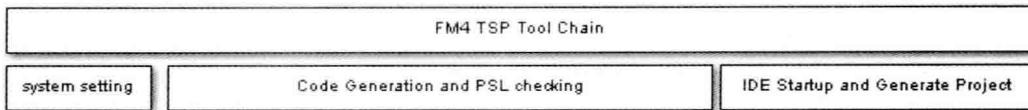


图 19.1-7 工具链支持的功能链

### 1. 模型的系统目标设置

Simulink 新建立的模型默认的设置仅适合于连续系统的仿真,不适用于嵌入式代码自动生成,更不能生成针对某一款芯片的驱动代码。所以工具链的第一步就是将模型设置到一个能够生成 Cortex M4 MB9BF568R 可执行代码的环境。在模型的系统目标文件设置为 FM4.tlc 后即可完成模型环境的配置。系统目标文件选择如图 19.1-8 所示。

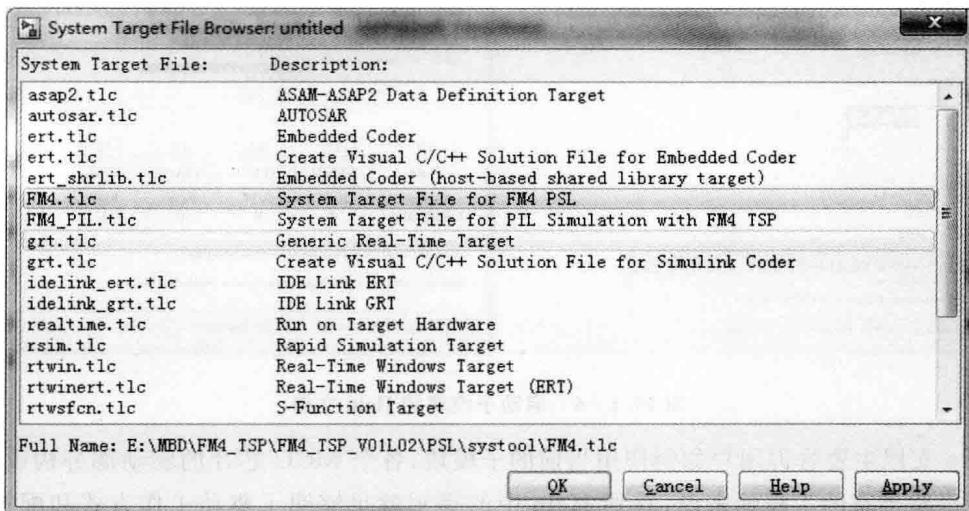


图 19.1-8 选择 FM4 的系统目标文件

系统目标设置会触发系统目标文件选择回调函数(select Callback)一系列的动作:

- ① 模型解算器从变步长设置为定步长,适合嵌入式硬件资源调配。
- ② 自动配置 Hardware Implementation 中硬件目标为 Spansion ARM Cortex M4。
- ③ 添加 FM4 系列硬件目标定义并配置到当前模型。
- ④ 关闭通用嵌入式主函数模板生成功能,自动选用 FM4 用主函数。
- ⑤ 自动打开代码生成完毕启动报告功能,方便 TSP 使用者查看代码文件。
- ⑥ 追加 IDE 选择,TSP 可以选择希望启动的 IDE 软件,或者仅查看代码不启动 IDE。

设置第三方 IDE 工具的方式是通过 Configuration Parameter 中 FM4 的 TSP 自定义的 FM4 Target Options 页面实现,如图 19.1-9 所示。

⑦ 将 FM4 自定义的数据类型对象生成到 Base Workspace 中,自定义数据别名对象如图 19.1-10 所示。

### 2. 代码生成与 PSL 设置检测

以图 19.1-5 所示 ADC0 的初始化配置为例,根据此配置生成初始化外设 ADC Unit0 的驱动代码,能够完全体现 TSP 使用者所做的配置。完整的模型如图 19.1-11 所示,除了 ADC0 Init Block 外,模型中必须包括一个 PDL\_User 模块和一个 System 模块。

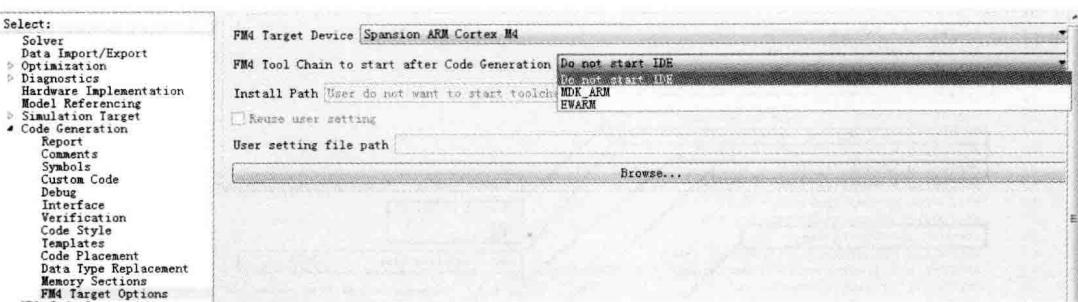


图 19.1-9 FM4 自定义面板配置工具链

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

| Name      | Value                  |
|-----------|------------------------|
| boolean_t | 1x1 Simulink.AliasType |
| char_t    | 1x1 Simulink.AliasType |
| float32_t | 1x1 Simulink.AliasType |
| float64_t | 1x1 Simulink.AliasType |
| int16_t   | 1x1 Simulink.AliasType |
| int32_t   | 1x1 Simulink.AliasType |
| int8_t    | 1x1 Simulink.AliasType |
| uint16_t  | 1x1 Simulink.AliasType |
| uint32_t  | 1x1 Simulink.AliasType |
| uint8_t   | 1x1 Simulink.AliasType |

图 19.1-10 FM4 数据类型对象

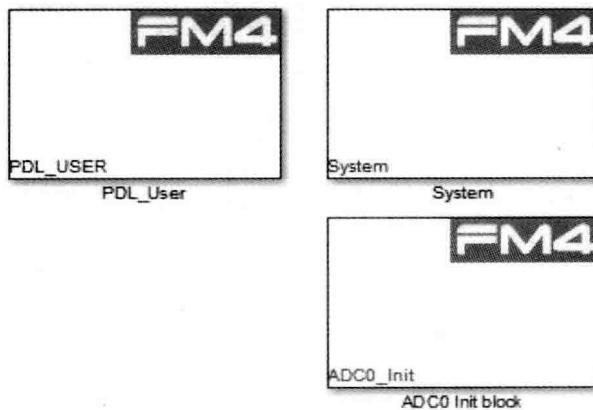


图 19.1-11 FM4 ADC0 初始化模型

该模型生成的代码中关于 ADC0 初始化的驱动代码及参数配置的对应关系如图 19.1-12 所示。

在生成代码前, 工具链会检测当前模型中是否有不合理的设置:

- ① 同一个外设单元是否被多次使用, 如一个模型中使用两个 ADC0 单元子模块即会报错;
- ② 不同外设之间配置冲突, 即错误的硬件依赖关系, 如未进行初始化的模拟信号输入通道被用来做采样通道等。

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```

48 void ADC0_Init(void)
49 {
50     stc_adcn_t* pstcAdc = NULL;
51     stc_adc_config_t stcConfig;
52     PDL_ZERO_STRUCT(stcConfig); // Clear local configuration to zero
53     stcConfig.u82Canne1Select.AD_CH_0 = 1u;
54     stcConfig.bLSEAlignment = TRUE;
55     stcConfig.enMode = SingleConversion;
56     stcConfig.u32SamplingTimeSelect.AD_CH_0 = 0u;
57     stcConfig.enSamplingTime0 = Value8;
58     stcConfig.u8SamplingTime0 = 10u;
59     stcConfig.enSamplingTime1 = Value8;
60     stcConfig.u8SamplingTime1 = 0u;
61     stcConfig.u8SamplingMultiplier = 3u;
62     stcConfig.u8EnableTime = 0u;
63     stcConfig.bScanTimerStartEnable = FALSE;
64     stcConfig.enScanConversionTimerTrigger = AdcNoTimer;
65     stcConfig.u8ScanDepth = 0u;
66     stcConfig.bPrioExtTrigStartEnable = FALSE;
67     stcConfig.bPrioExtTrigEnable = FALSE;
68     stcConfig.bPrioTimerStartEnable = FALSE;
69     stcConfig.enPrioConversionTimerTrigger = AdcNoTimer;
70     stcConfig.u8PrioLevel12AnalogChSel = 0;
71     stcConfig.enPrioStageCount = PrioStageFirst;
72     stcConfig.bComparisonEnable = FALSE;
73     stcConfig.u16ComparisonValue = 0u;
74     stcConfig.bCompIrqEqualGreater = TRUE;
75     stcConfig.bCompareAllChannels = TRUE;
76     stcConfig.u8CompareChannel = 0;
77     stcConfig.bRangeComparisonEnable = FALSE;
78     stcConfig.u16UpperLimitRangeValue = 0u;
79     stcConfig.u16LowerLimitRangeValue = 0u;
80     stcConfig.u8RangeCountValue = 0;
81     stcConfig.bOutOfRange = FALSE;
82     stcConfig.bRangeCompareAllchannels = TRUE;
83     stcConfig.u8RangeComapreChannel = 0u;
84     stcConfig.bFifoOverrunIRQEnable = TRUE;
85     stcConfig.pfnErrorCallbackAdc = &ADC0_FIFO_Overrun_Callback;
86 }

```

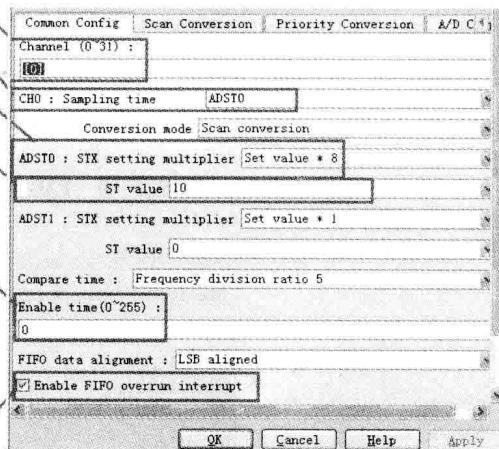


图 19.1-12 根据配置生成驱动代码

工具链还会自动检测当前模型中所使用的模块和中断，并自动开启 PDL\_User 模块中对应的开关，以上述模型为例，PDL\_User 的设置如图 19.1-13 所示。

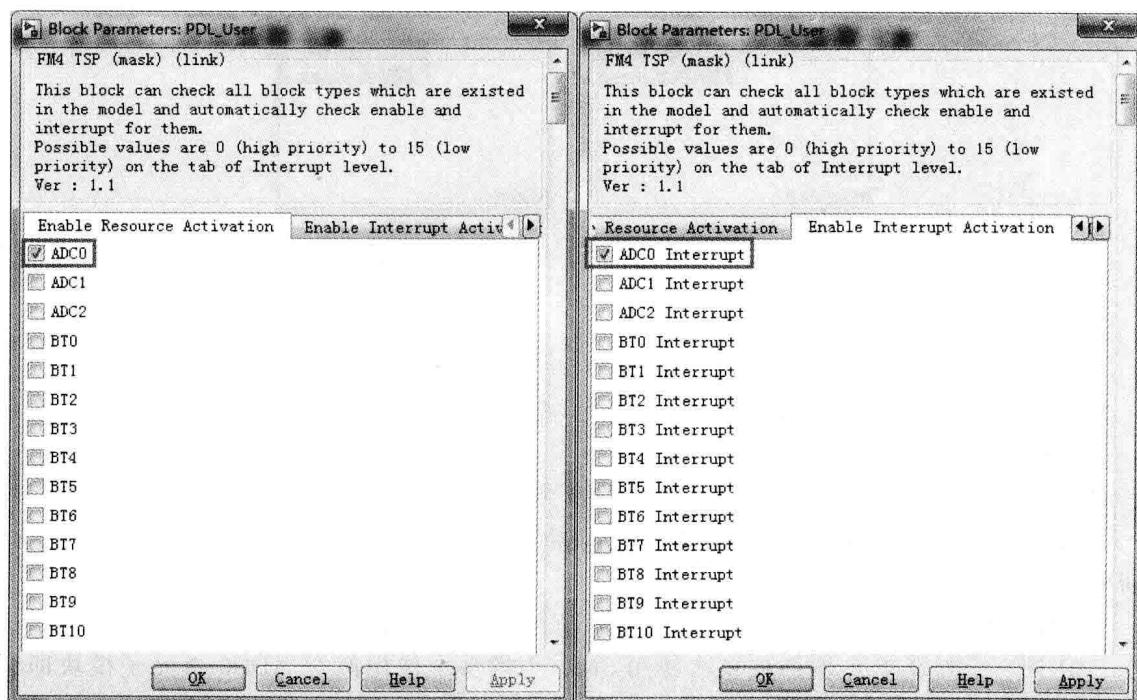


图 19.1-13 根据模型自动开启资源开关

### 3. IDE 启动与工程文件创建

在生成代码之后,根据用户的选择,Simulink 决定是否启动第三方 IDE,如图 19.1-14 所示。

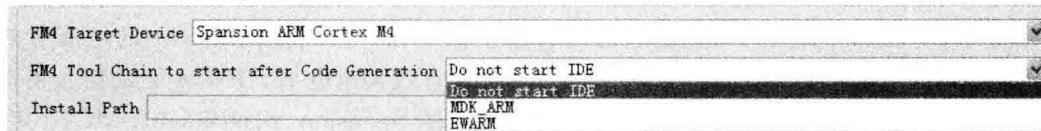


图 19.1-14 IDE 选择启动与否

若选择 MDK\_ARM,则生成代码后,自动启动用户 PC 上安装的 Keil 并生成直接可以编译的工程,只要单击工具栏的 Build 即可生成目标文件,如图 19.1-15 所示。

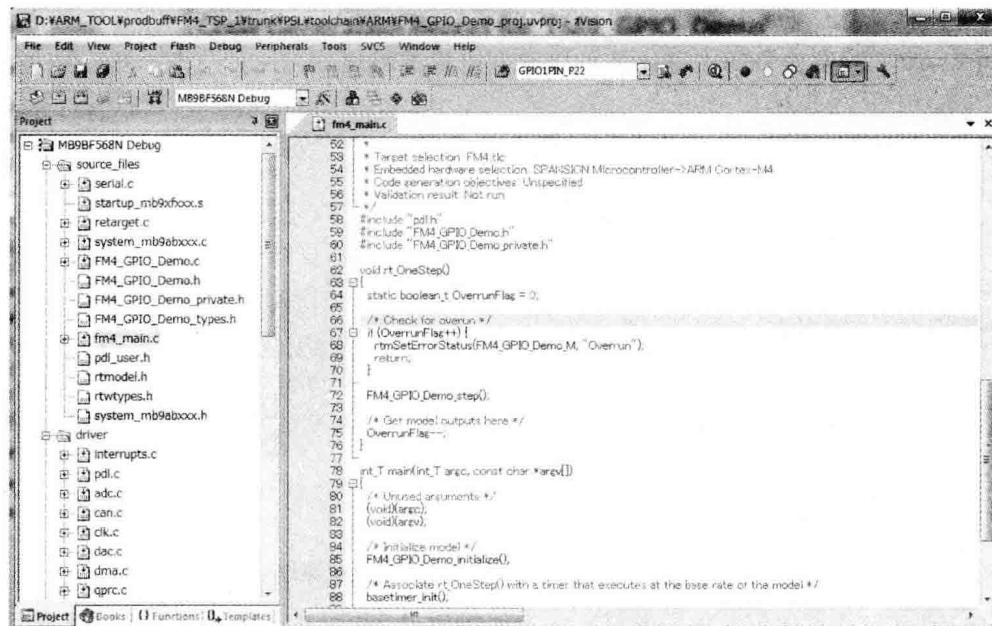


图 19.1-15 IDE 自动启动并生成直接可用的工程文件

TSP 为 MBD 嵌入式开发者提供了 Simulink 环境下的驱动库,驱动部分无须手写代码,只要在模型上设置寄存器工作方式即可自动生成可直接使用的嵌入式 C 代码,并提供自动化支持工具链,这样不仅加速了开发过程,而且除去了手写代码引入的不具合,并且能够通过读/写模块方便地与算法模型连接起来,让嵌入式工程师省下手写驱动代码的时间,更专注于算法设计,真正做到高效开发。接下来,笔者使用此款芯片及 TSP 来快速开发直流电机控制软件。

## 19.2 直流电机控制原理

直流电机调速系统需要一个专门向电机供电的直流可控电源,常用脉宽调制变换器作为直流可控源。脉宽调制变换器通过控制电子开关器件的开与关对脉冲的宽度进行调制,从而产生可变的等效电压。本书是利用电子开关的开闭输出不同占空比的 PWM 波形(脉宽调制)得到不同有效值的等效电压,并以此对直流电机调速。

所谓 PWM 调制，简而言之，就是把恒定的直流电压调制成频率一定、宽度可变的脉冲电压，从而改变平均输出电压的大小。PWM 信号只有两种状态，高电平和低电平，对于一个给定的周期来说，高电平时间和周期之比称为占空比，电机的速度与施加的平均电压成正比，输出转矩则与电流成正比。直流电机高效运行的最常见方法是施加一个 PWM 方波，其通一断比率对应于直流电机的转速，即直流电机的转速正比于在一个周期内 PWM 的电压有效值。电机起到一个平均滤波器作用，将 PWM 信号转换为有效直流电平。PWM 驱动信号很常用，使用微处理器很容易产生 PWM 信号，比如本文使用的 Spansion MB9BF568R MCU，其外设 MFT(多功能计时器)或者 BT(Base Timer)都可以提供 PWM 信号。

改变加到直流电机电枢两端的直流驱动电压，即可改变电机的转速；使用 PWM 方法，可以方便地改变加给电机电枢的平均电压的大小，此处 PWM 信号波形由 MFT 提供，如图 19.2-1(a)所示，设  $U_i$  是三极管基极的控制电压， $U_m$  为电机两端的直流电压，其波形如图 19.2-1(b)所示，在一个周期  $T$  内，平均电压  $U_m$  为  $t_1 \times U_{cc}/T$ ，其中  $t_1/T$  称矩形波的占空比，可知，改变控制信号的占空比就可以改变电机的转速。

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

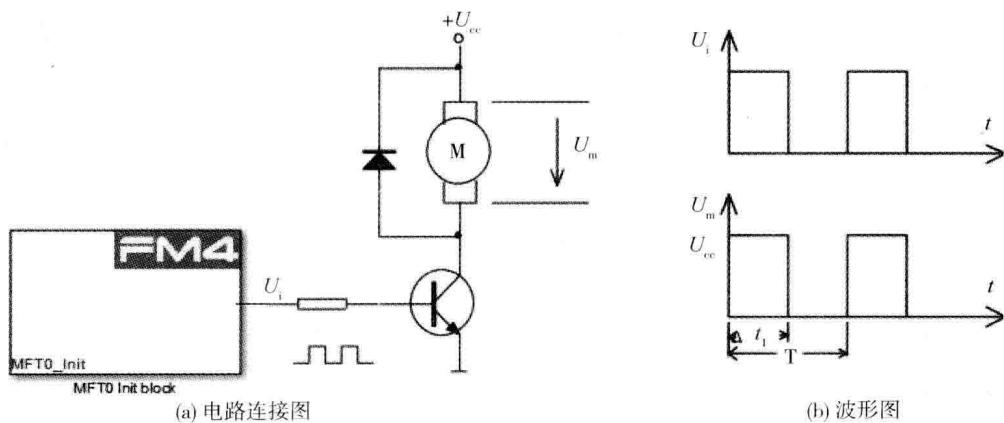


图 19.2-1 PWM 信号控制直流电机原理

那么，直流电机调速控制的问题就转化为如何通过 MCU 的 MFT 外设输出 PWM 信号，以及变化占空比。

### 19.3 系统的构成

直流电机控制的构成包括 FM4 开发板、升压电路、直流电机及控制信号，如图 19.3-1 所示。

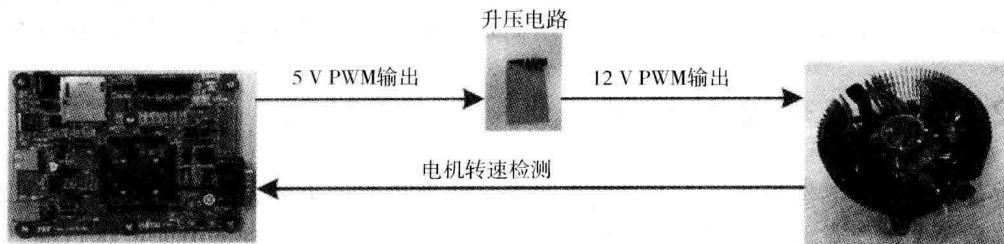


图 19.3-1 直流电机控制系统

如图 19.3-1 所示为整个控制回路的构成,需要硬件包括:搭载 Spansion MB9BF568R MCU 的 FM4-120SD1NQ 开发板、E92f 直流电机(额定电压 12 V,带有霍尔传感器,额定电压 5 V)、电压放大驱动电路、E92f 直流电机是 4 线式的,接口如图 19.3-2 所示。4 根线的功能分别为电源线、地线、使能端及脉冲输出端。

通过控制电机使能端电平的高与低来控制电机的转停及转速的高低,Spansion MB9BF568R 芯片输出的 PWM 信号通过 I/O 口输出电压幅值只有 5 V,不足以驱动需要 12 V 电压的直流电机,故需要一个放大电路来提升 PWM 的输出信号。设计如图 19.3-3 所示升压电路。

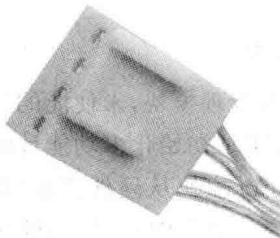


图 19.3-2 直流电机的 4 线接口

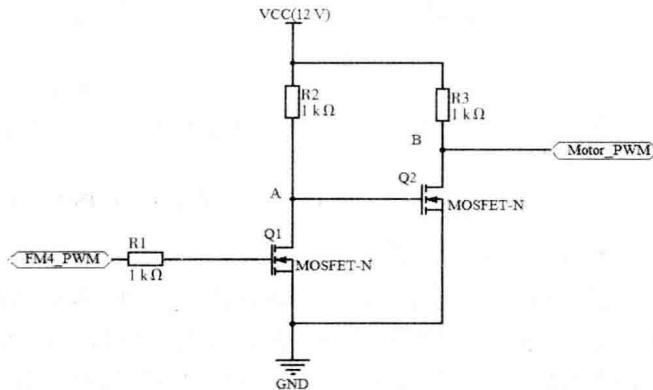


图 19.3-3 电机驱动升压电路原理图

当 FM4\_PWM 端口输入低电平时,Q1 不导通,A 点电平被拉高,Q2 导通,B 点接地,电平为低;当 FM4\_PWM 端口输入高电平时,Q1 导通,A 点电平被拉低,Q2 关闭,B 点输出高电平。VCC 为 12 V,故通过此电路可以将输入的 5 V 电压升高为 12 V 电压。

升压电路输出 Motor\_PWM 给直流电机的使能端。接收 PWM 波而转动起来的电机通过霍尔传感器也输出一系列脉冲信号,通过 I/O 口读入该信号,反馈到 MCU 中可以计算出电机的实际转速,从而进行闭环速度控制。驱动电路板与 FM4 开发板的连接引脚如表 19.3-1 所列。

表 19.3-1 FM4-120SD1NQ 开发板与电压放大驱动电路的连接

| 序号 | FM4 开发板所用引脚 | 驱动电路板    |
|----|-------------|----------|
| 1  | VCC         | VCC(5 V) |
| 2  | VSS         | GND      |
| 3  | P3A(RT0)    | FM4_PWM  |
| 4  | P30(Input)  | 霍尔传感器输出端 |

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

## 19.4 模型的建立

系统模型包括 3 部分:PWM 波产生模块、电机转速计算模块和速度调节模块。PWM 波由 FM4 芯片的多功能计时器(MFT)提供,通过 I/O 口输出给电机。电机接收到 MFT 发出的 PWM 波之后开始以一定转速转动,霍尔传感器反馈 PWM 波给 FM4 另一个 I/O 口,FM4

MCU 对脉冲波进行计数并转化为转速,与目标转速进行对比,根据差进行闭环控制。系统模型如图 19.4-1 所示。

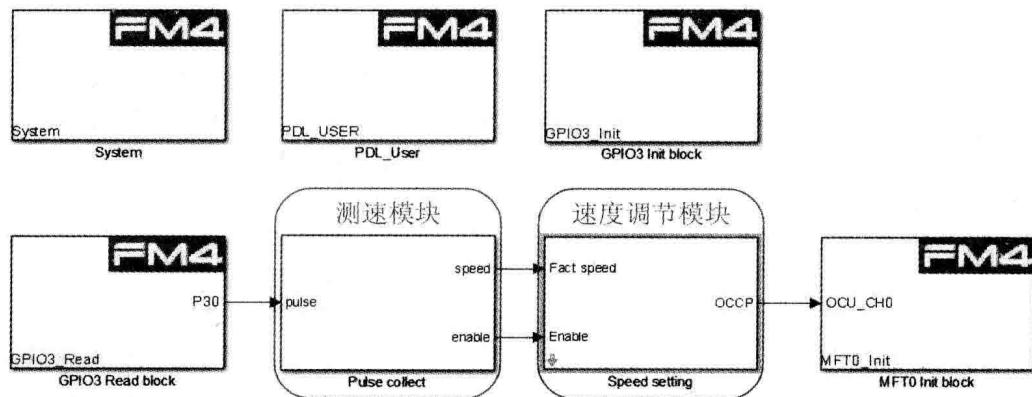


图 19.4-1 基于 FM4\_TSP 的直流电机调速模型

System 模块为硬件时钟初始化配置模块,PDL\_User 是硬件资源管理模块,未使能的外设不开启已达到节约电力降低发热量的功能。GPIO Init、MFT Init 模块则是初始化多功能计时器和通用输入/输出口的配置模块。配置过的 I/O 口可以通过 GPIO Read 模块从硬件获取信号值。针对获取的 PWM 信号进行电机转速计算及反馈控制量计算,再通过 MFT 的 OCU 的输出端口 RT0 将调整后的 PWM 信号(即反馈控制量)传回给 MCU 进行直流电机的转速控制。

### 19.4.1 PWM 波形的产生

多功能计时器 MFT 是 Cypress/Spansion MB9BF568R 微处理器的一个很具特色的外设,专门用于产生控制电机的 PWM 波,由三角波产生器 FRT、输出比较单元 OCU 等模块构成。FRT 产生三角波时有多种模式,本文使用 Up count 模式,即给定一个计数峰值到峰值 16 位寄存器 TCCP,从 0 开始 FRT 每个时钟周期计数加 1,达到 TCCP 的值之后立刻回到 0 值再重新开始计数。输出比较单元 OCU 需要与 FRT 连接使用,通过 OCU 的 RT0 输出波形,初始值设置为高电平。OCU 里面有一个 16 位的比较值寄存器 OCCP,设置 RT 的波形在 FRT 计数达到 OCCP 时进行一次翻转,在 OCCP 达到峰值时再进行一次翻转。这样,只要控制好 OCCP 与 TCCP 的值就能够控制 PWM 波的占空比了。FRT 产生锯齿波的原理图如图 19.4-2 所示。

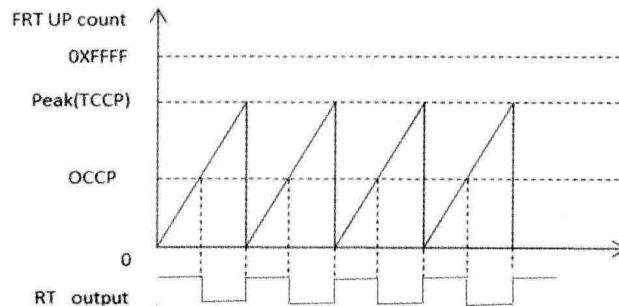


图 19.4-2 FRT 与 OCU 共同作用产生 PWM 波

若您对此书内容有任何疑问,可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

由图 19.4-2 可知 PWM 波的占空比 = OCCP/TCCP。使用 TSP 中的 MFT Init 模块，双击打开其参数对话框，通过其 GUI 可以方便地设置寄存器以为后面驱动代码生成之用。FRT 的页面上设置 FRT 计数模式为 Up-count，由于 FRT 计数周期 = (TCCP+1) × 外设时钟周期，故设置 TCCP 的值为 9999，即 FRT cycle value。MFT 模块 FRT 的 GUI 设置如图 19.4-3 所示。

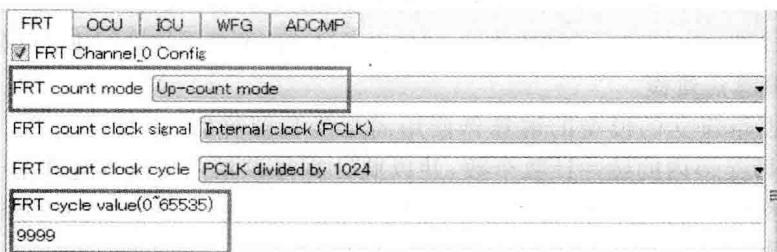


图 19.4-3 FRT 与 OCU 连接产生 PWM 波设置 1

OCU 页面上，设置与 FRT 通道的连接，初始电平设置为高，并在 FRT 上升阶段计数值达到 OCCP 时以及 TCCP 时进行翻转。FRT 超过 OCCP 之后达到峰值也要进行一次翻转。设置如图 19.4-4 所示。

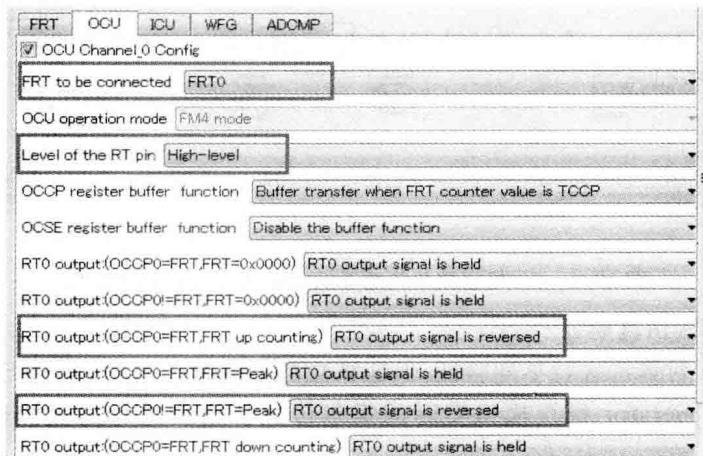


图 19.4-4 FRT 与 OCU 连接产生 PWM 波设置 2

当 OCCP 为不同的值时，RT 输出的 PWM 波对应的占空比也是可变的，如设置 OCCP 值为 2 000，可知 MFT 的 RT 输出的 PWM 波占空比为 0.2。使用带有 FM4 PSL 驱动模块的模型生成代码及项目工程，直接编译下载到 FM4 开发板上使用示波器进行波形观察，其 PWM 波形如图 19.4-5 所示。

## 19.4.2 电机转速计算模块

电机转速与霍尔传感器的输出脉冲波之

若您对此书内容有任何疑问，可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

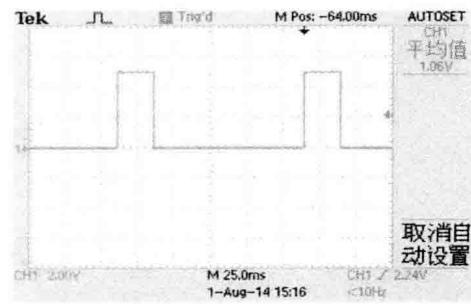


图 19.4-5 OCCP 为 2 000 时的 PWM 波形

间存在一定的关系,也就是说,电机每转一转霍尔传感器输出  $N$  个周期的脉冲波。但是这个  $N$  是多少呢,需要通过实验获得电机转速与单位时间霍尔传感器输出脉冲周期个数之间的关系。再通过实验获得不同 OCCP 值下霍尔传感器单位时间的输出脉冲个数,两个关系桥接就得到了转速和 OCCP 值之间的关系。

将此关系编制为查找表(Lookup Table)在模型中估算不同目标转速时对应的 OCCP 值,这是以空间换取计算速度的方法,常用于嵌入式控制软件开发之中。虽然估算的 OCCP 值达不到精确的目标转速,但是提供了比较接近的初始速度,之后再通过速度调整模块进行速度微调来优化。

### 1. 脉冲检测算法建模

根据上述思路建立的模型的解算器使用固定步长解算器,步长设为 0.000 1 s,统计每 20 000 个步长(即 2 s)内的脉冲计数个数,并以此计算每秒霍尔传感器输出脉冲的个数,再根据实验得到的换算关系将脉冲个数转换为电机转速。图 19.4-6 所示为速度计算模块,输入为霍尔传感器脉冲波形,输出为电机转速和调速使能信号。

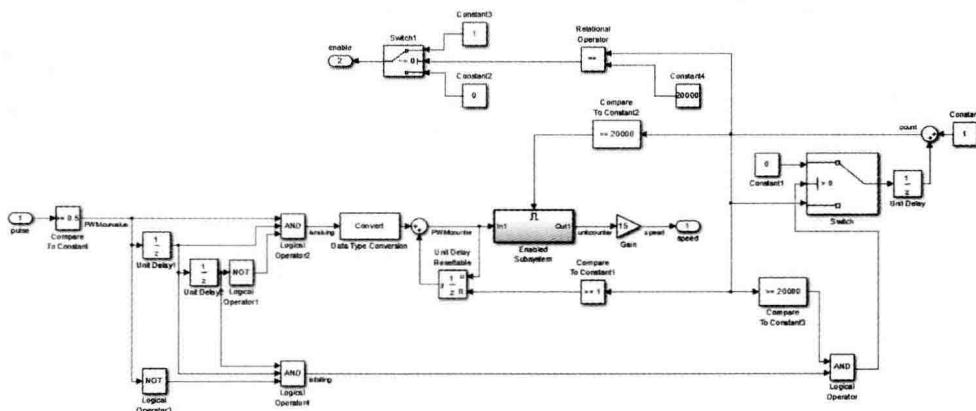


图 19.4-6 根据霍尔传感器的脉冲波形计算转速模型

### 2. 脉冲检测算法仿真验证

通过 Simulink 对算法进行早期验证,保证算法的准确性。这里,通过 Simulink 自带的信号发生器 Pulse Generator 模块模拟实际的霍尔传感器输出进行仿真验证,仿真模型图如图 19.4-7 所示。

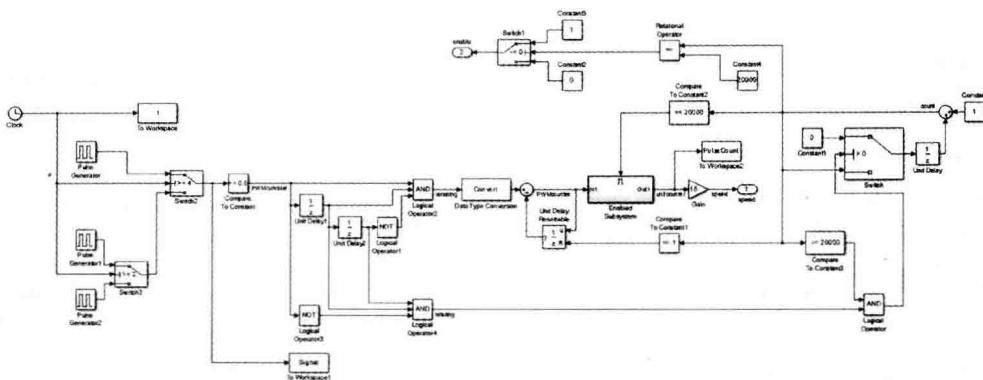


图 19.4-7 转速计算算法验证模型图

2 s 的定时长度下, 转速调节使能验证的图像如图 19.4-8 所示。

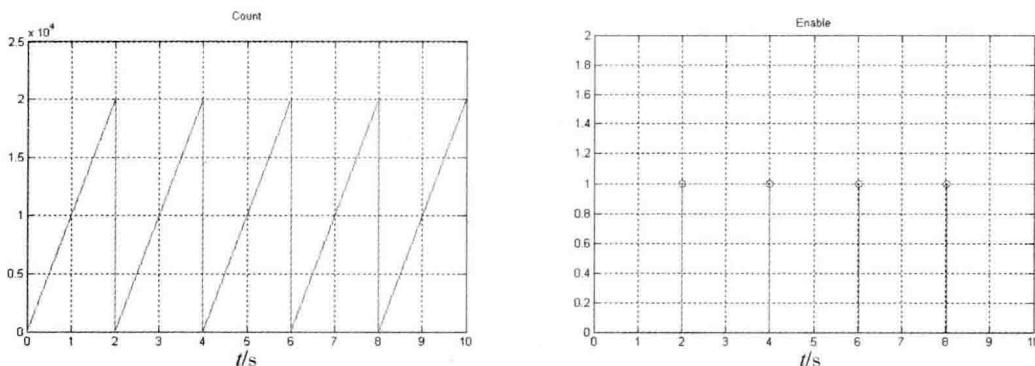


图 19.4-8 Count 和 Enable 波形图

Simulink 中 Pulse Generator 模块可以用于产生脉冲信号, 因此考虑使用 Pulse Generator 模拟电机输出的脉冲信号进行仿真验证。假设, 0~2 s 内电机输出频率为 50 Hz, 幅值为 1, 占空比为 0.5 的脉冲信号(100 个脉冲); 2~4 s 内电机输出频率为 20 Hz, 幅值为 1, 占空比为 0.5 的脉冲信号(40 个脉冲); 4 s 之后 Pulse Generator 输出频率为 40 Hz、幅值为 1、占空比为 0.5 的脉冲信号(2 s 内 80 个脉冲)。仿真结果如图 19.4-9 所示。

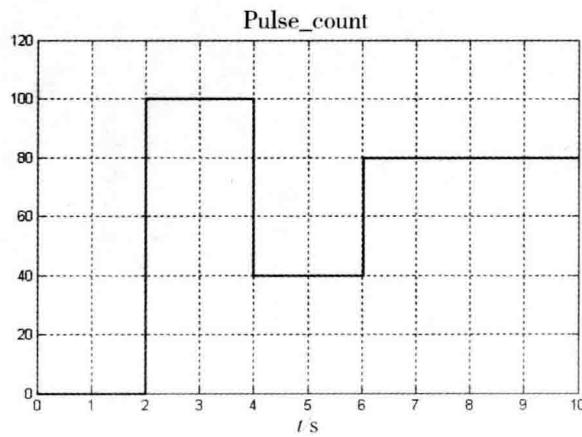


图 19.4-9 Pulse-Count 波形图

由图 19.4-8 可知, Count 值从 0 累加到 20 000, 然后再清 0, 由于模型的采样时间设置为 0.000 1 s, 等同于实现了 2 s 的定时, 同时 Enable 信号也是 2 s 使能一次; 由图 19.4-9 可知, Pulse-Count 等于 Pulse Generator 发生器 2 s 内产生的脉冲个数。由此, 我们可以得出结论, 测速模块的算法设计是正确合理的。

### 19.4.3 电机调速模块

将电机转速计算模块算出的速度与设定的目标转速做对比, 如小于目标转速, 则需要提升平均输出电压以加速电机转动, 即提升 PWM 波的占空比, 本质上就是增大 OCCP 的值; 反之, 则降低 OCCP 的值。OCCP 值调整的基础值根据查找表得到。实际转速每 2 s 更新一次, 因此转速调整模块同样每 2 s 使能一次。调速模块的模型如图 19.4-10 所示。

若您对此书内容有任何疑问, 可以凭在线交流卡登录 MATLAB 中文论坛与作者交流。

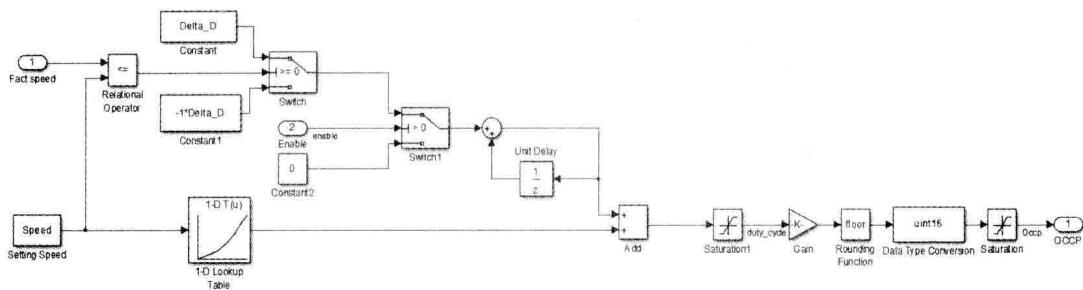


图 19.4-10 调速模块子系统图

上述 3 个模块构建成的模型可以直接用来生成代码与 Keil 工程，并进行编译链接下载运行。模型建立完成之后，嵌入式开发者仅需一键即可实机运行可执行文件。实际运行的硬件环境如图 19.4-11 所示。

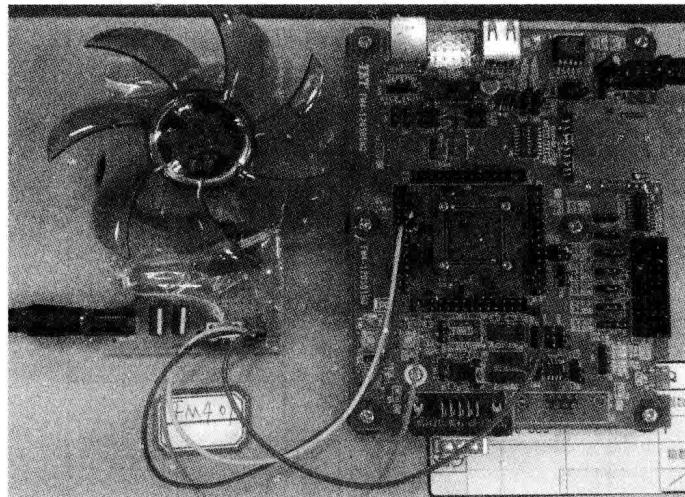


图 19.4-11 硬件执行环境

## 19.5 总 结

基于模型的设计方式对于开发嵌入式控制软件有如下好处：设计阶段即可通过模型仿真对设计思想进行早期验证，降低传统嵌入式开发流程中到了测试阶段发现问题再回头修改设计并重新遍历测试流程这种费时费力的风险。结合 MATLAB/Simulink 环境下的 TSP 工具箱，从设计到代码生成，编译环境启动，项目工程生成及编译下载动作完全自动化实现，大大提高了嵌入式工程师的开发效率，使他们可以专注于控制算法的设计与验证。根据实践，使用 TSP 结合 MBD 流程开发的控制软件是高效可靠的。

## 参考文献

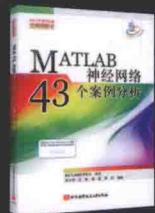
- [1]The MathWorks Inc. Simulink CoderTarget Language Compiler [M],2013.
- [2]孙忠潇. Simulink 的流控制[J]. FNST Technical Journal,2014,23(4):12-18.
- [3]吕欣, 董明盛, 张晓娟, 等. 酒精发酵非结构动力学模型及其参数估计[J]. 西北农林科技大学学报,2005(11):23-28.
- [4]刘瑞叶,任洪林,李志民,等. 计算机仿真技术基础[M]. 北京:电子工业出版社,2002.
- [5]朱成荣,邵惠鹤. 基于 Matlab 的发酵系统仿真与操作优化[J]. 微计算机信息,2008(1):81-83.
- [6]王英臣. 菌体比生长速率的酒精发酵动力学研究[J]. 酿酒科技,2005(9):56-61.
- [7]伍勇,肖泽仪,黄卫星,等. 酿酒酵母在硅胶膜生物反应器中连续发酵的生长动力学[J]. 现代化工,2004,24(1):34-39.
- [8]张君,唐昌平,刘德华,等. 指数流加模型在乙醇气体发酵过程中的应用研究[J]. 食品与发酵工业,2004,30(7):43-47.
- [9]郭训华,邵世煌. Simulink 建模与仿真系统设计方法及应用[J]. 计算机应用,2005,11(2):13-17.
- [10]孙忠潇. 活用 Matlab 注释发布功能[J]. FNST Technical Journal 技术期刊,2013,10(3):13-16.
- [11]孙忠潇. TSP——MBD 的驱动层解决方案[J]. FNST Technical Journal 技术期刊,2014,21(2):13-16.
- [12]颜庆津. 数值分析[M](第 4 版). 北京:北京航空航天大学出版社,2012.
- [13]张延亮. MATLAB/Simulink 在工业界和学术界中的使用及就业市场[R]. 南京:2014 年 MATLAB 中文论坛南京研讨会,2014.

特约策划：张延亮（math）

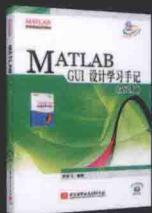
策划编辑：陈守平

封面设计：RUNSIGN

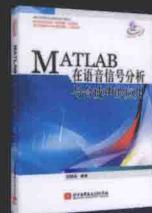
# “在线交流，有问题必答”系列图书



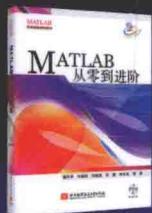
全行业优秀畅销书的升级版  
本，一线实战版主主笔，一问  
一答间提升您的功力。



同类图书中的销量冠军。读者  
评价该书“内容全面，作者负责，  
是学习GUI的首选”。



作者年过70，从事信号处理  
30余年，论坛回帖数过  
2700，靠不靠谱看书便知。



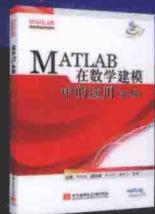
从零开始，五位师傅，口传心  
授，帮你练就MATLAB神功！



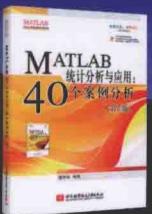
历时三年亮剑之作——国内首  
部用MATLAB函数仿算高等光  
学模型的技术书，辅以丰富实  
例。



国内MATLAB应用的领跑者倾  
情奉献之力作，源码一线实战  
的超强实力，同类专业图书的  
开山之作。



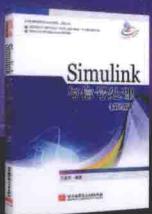
数学建模竞赛大赛季军，用  
80后的执着和创新，助您用  
MATLAB在竞赛中出奇制胜！



跟随一位幽默睿智的导师，将  
“MATLAB+统计”引入课堂、  
引进工作、用于生活……



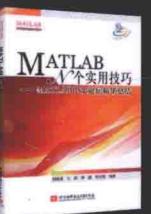
新内容，新思想，新方法，新  
技术，Fast your MATLAB没  
商量！



MathWorks首席工程师执笔，  
所有实例均来自于开发人员和  
用户的反馈，权威，经典……



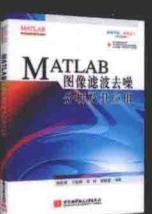
穿越理论，透视技巧，拓宽应  
用，在模式识别与智能算法中  
将MATLAB用到High!



4位精英版主，“99+n”个实  
用技巧，无限次的在线帮助，  
解决您的N个问题。



介绍了MATLAB在光学类课程  
中的应用，并附课程设计综合  
实例。配课件。



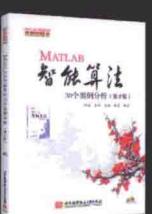
全面而系统地讲解了  
MATLAB图像滤波去噪分  
析及其应用。



最全授课资料免费提供：视  
频、源代码、课件、电子实验  
书、试卷……



以数值分析理论为主线，侧重  
讲解MATLAB下各种算法的实  
现。纯干货！



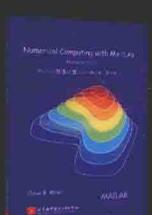
全行业优秀畅销书的升级版  
本，纯案例式讲解，辅以免费  
视频。



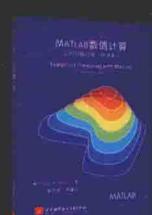
MathWorks工程师之作。有  
读者评论说，看完此书，可  
以高端优雅地进行大型程序  
的开发。



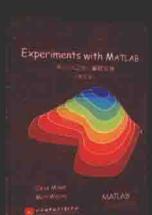
本书作者为MATLAB中文论坛  
的权威会员，技术帖过万，有  
疑问？来论坛找hyowinner！



MATLAB之父Cleve Moler的  
经典之作，经Cleve本人正式  
授权，中国首印，原汁原味。



Numerical Computing with  
MATLAB一书的中译本。张志  
涌编译。



MATLAB之父Cleve Moler的  
“玩票”之作。趣味MATLAB，  
高超尽显。全球首发。



Experiments with MATLAB  
一书的中译本。薛定宇译。

上架建议：计算机软件

## 特别推荐

MathWorks

迈斯沃克软件（北京）有限公司

## 特约技术支持

MATLAB中文论坛 ([www.iLoveMatlab.cn](http://www.iLoveMatlab.cn))

ISBN 978-7-5124-1857-8



定价：69.00元

[General Information]

书名=Simulink仿真及代码生成技术入门到精通

作者=孙忠潇编著

页数=477

SS号=13895531

DX号=

出版日期=2015.10

出版社=北京航空航天大学出版社