使用AMESIM进行控制策略仿真验证

# 概述

在机电液控制系统的早期设计阶段，已经构想出了初步控制策略，但还不具备条件进行RCP仿真或HiL仿真时，在计算机上对控制策略进行数字仿真是验证其性能和减少BUG的唯一手段。

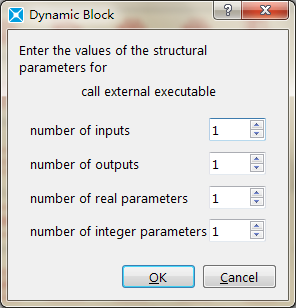
目前电液控制系统的被控对象通常采用AMESIM建模。而控制策略的实现，如果采用AMESIM的”signal, control lib”的话，实现起来相当复杂。因此，很多仿真人员都愿意采用MATLAB/SIMULINK作为控制策略的实现平台，这虽然降低的实现的复杂性，但却必须采用AMESIM-SIMULINK联合仿真的形式执行验证，增加了验证的难度并降低了执行的可靠性。

事实上，AMESIM的”signal, control lib”中除了常规的模块外，还有一个非常特别的“dynamic\_extern\_excutable(DYNEXE0)”模块。顾名思义，它能够调用外部应用程序参与仿真。这个模块为控制策略的实现提供了绝佳的途径。

# DYNEXE0模块介绍

## 描述

DYNEXE0是一个用于与外部应用程序进行交互的接口模块。模块的左侧端口和右侧端口分别代表其输入和输出。在库中选择该模块时，便可对输入和输出数量以及外部应用程序的整型参数和实型参数的数量进行定义，如下图所示。需要注意的是，DYNEXE0不会对参数的有效性进行验证。验证必须在外部程序中完成。

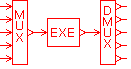


参数在仿真开始时被发送到应用程序中，AMESIM通过特殊“管道”与外部程序进通讯。通讯采用AMESIM内部的“[OpenPipe](qthelp://lmsimagine.lab/ame_dir/doc/stdlib/html/utils/OpenPipe.html)”应用进行初始化。

**注意：AMESIM必须采用C++编译器才能使用****DYNEXE0模块！**

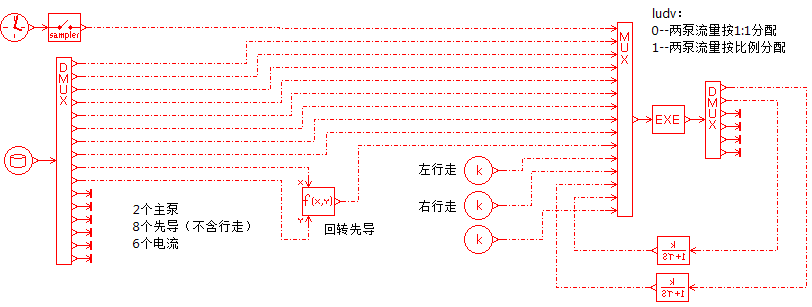
## 用法

由于一般的控制系统都是MIMO系统，因此必须采用DYNMUX2和DYNDMUX2与DYNEXE0进行连接，如下图所示即为一个5输入5输出系统。



## 示例

以如下图所示正流量挖掘机控制策略验证为例。正流量挖掘机控制策略的输入为9个先导压力和2个主泵压力，还有2个主泵反馈电流，以及一个仿真时间输入（后面会讲到它的作用）；输出为2个主泵电流和4个优先控制电流。图中的DYNEXE0模块实际为一个15输入10输出系统，多出的输入和输出端口不使用即可。



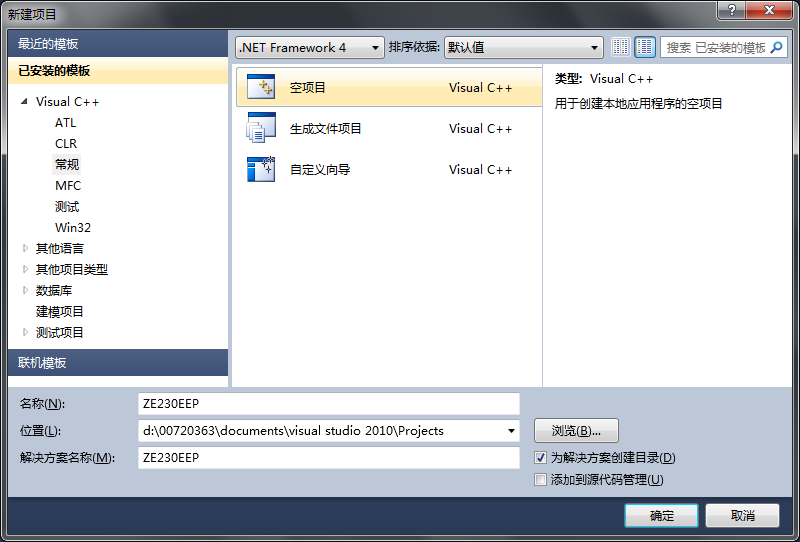
# 控制策略实现

控制策略在外部应用程序中实现。应用程序代码可在Visual Studio环境中采用标准C语言编写，并引用AMESIM提供的头文件。

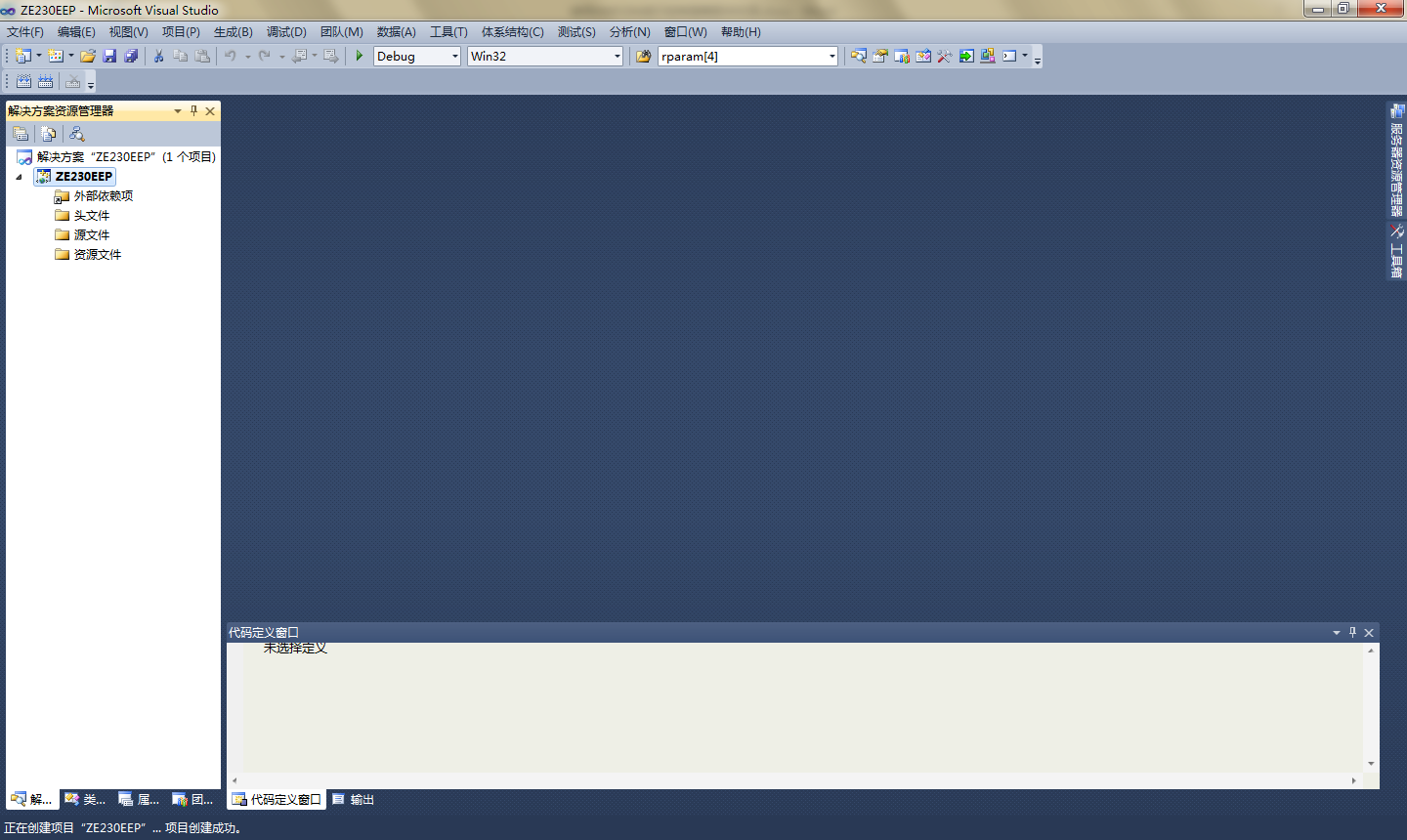
## 编程环境设置

以Visual Studio 2010为例，按以下步骤设置编程环境。

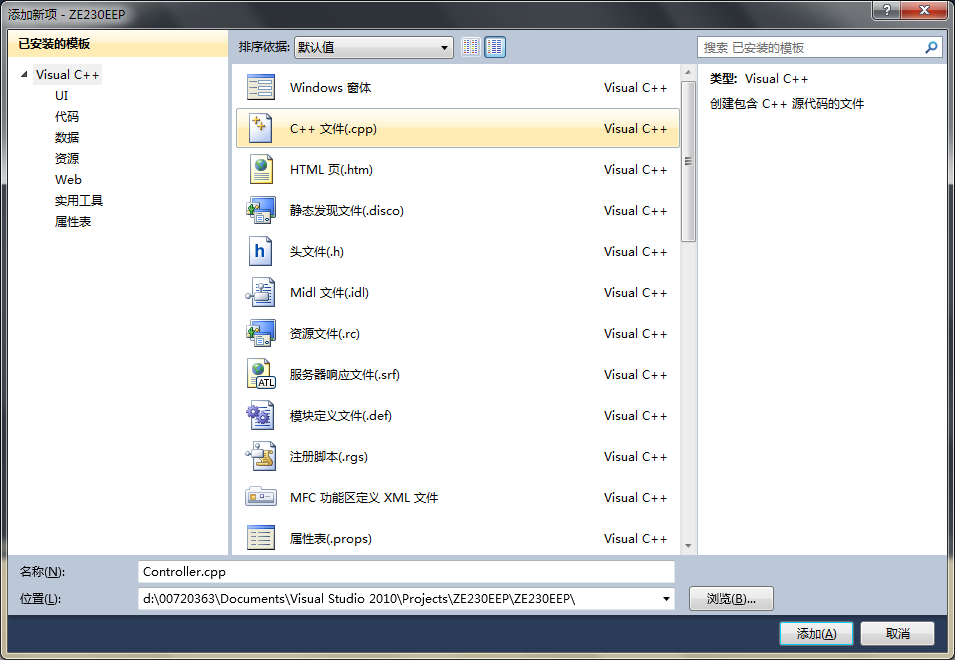
Step 1：新建项目。在Visual Studio中新建一个名为ZE230EEP的C++空项目，如下图所示。



点击确定之后，Visual Studio生成如下图所示窗口。窗口左侧为解决方案资源管理器，可以看到新建的项目“ZE230EEP”已经列在了解决方案树形目录下。

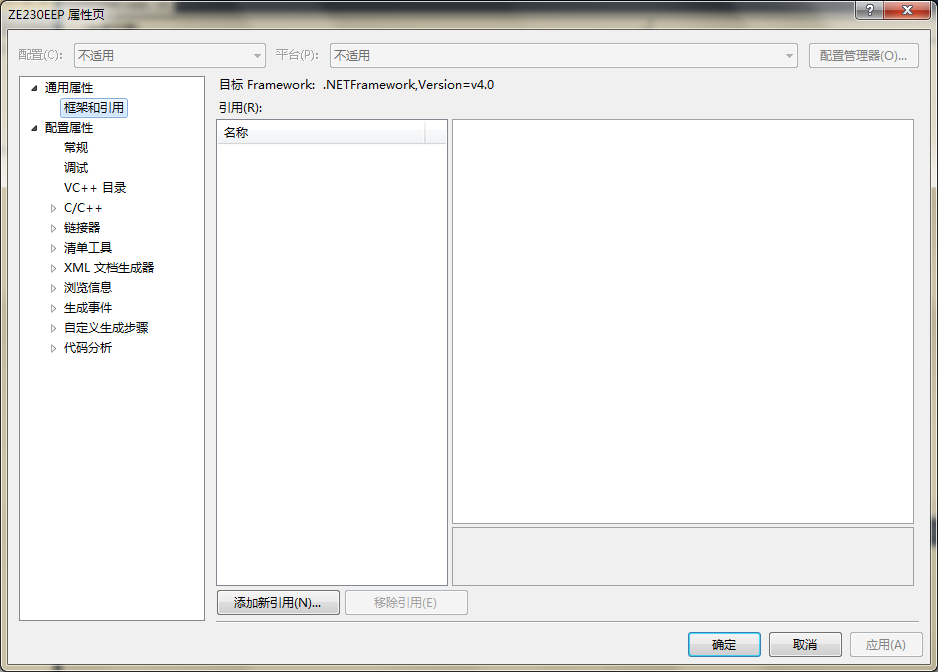


Step 2：添加源文件。右键点击解决方案资源管理器中的“源文件”项，依次选择“添加”、“新建项”菜单，出现如下图所示对话框。

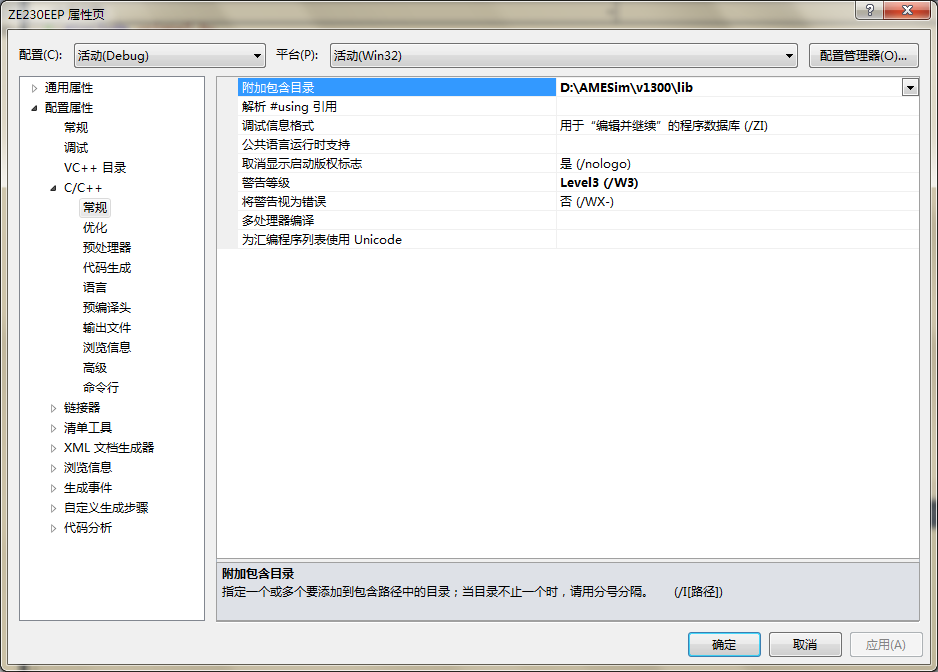


从对话框中添加一个名称为“Controller.cpp”的C++文件，点击确定。

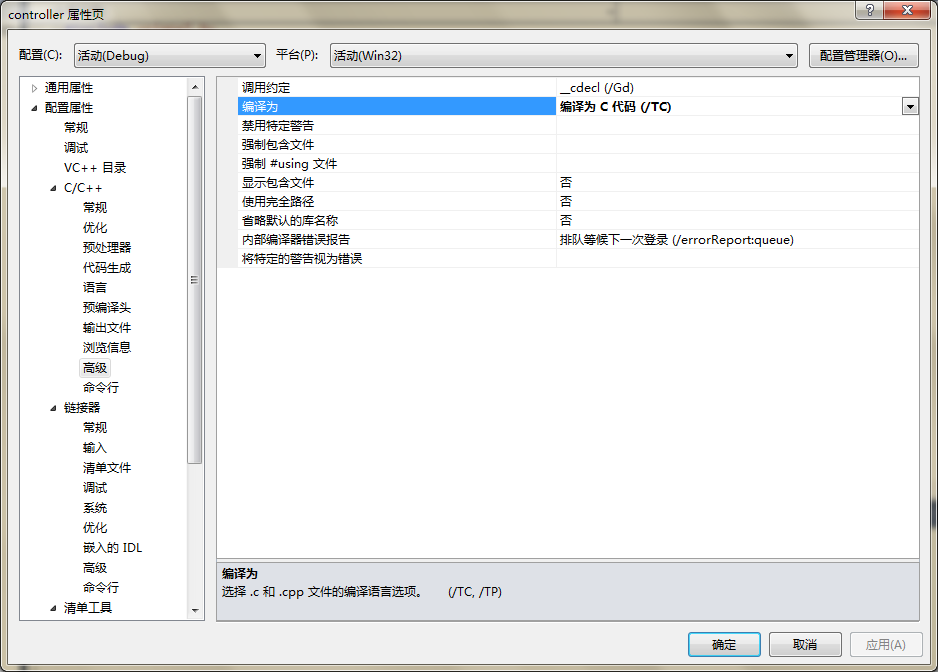
Step 3：配置项目属性。右键点击解决方案资源管理器中的项目名，选择属性菜单，出现如下图所示对话框。



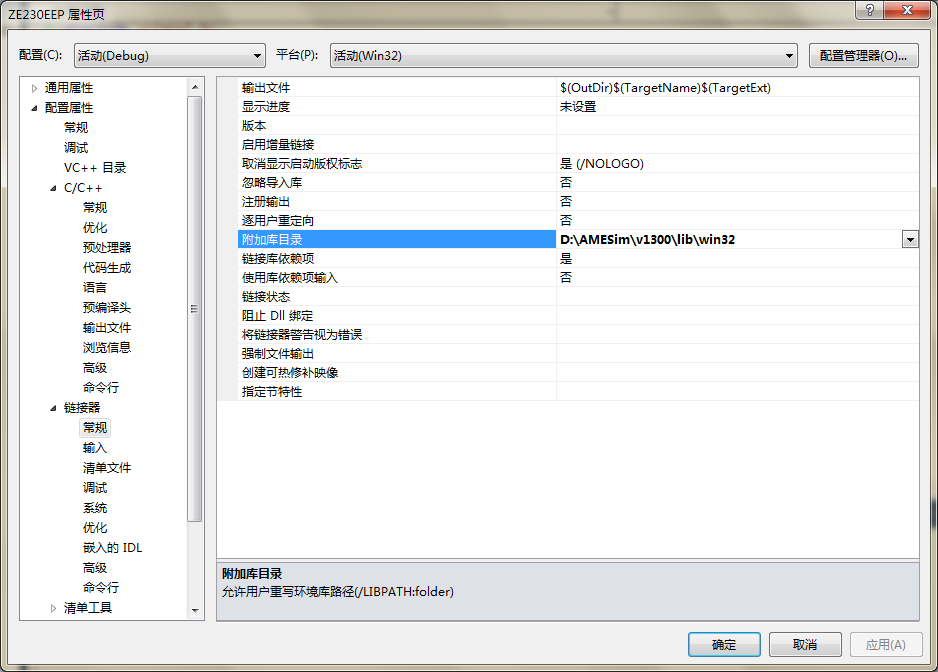
切换到“C/C++”目录项，点击“常规”条目，在右侧的“附加包含目录”中添加“D:\AMESim\v1300\lib”路径，即AMESIM的安装目录。**注意：路径必须使用标准绝对路径，而不能使用****环境变量！**



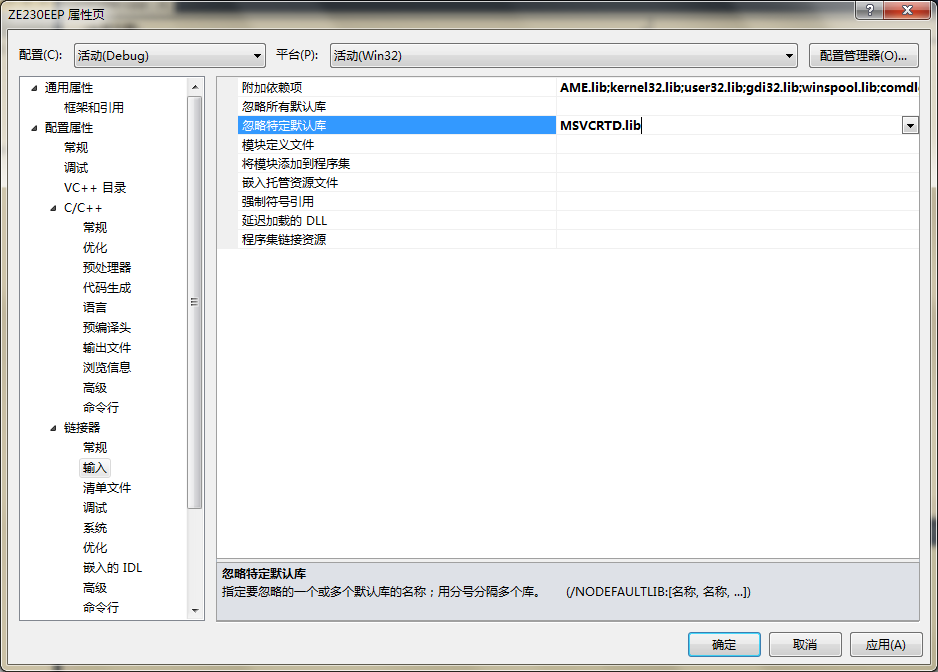
再切换到“高级”条目，在“编译为”选项中选择“编译为C代码(/TC)”，如下图所示。



然后切换到“链接器”目录项，点击“常规”条目，在“附加库目录”中添加“D:\AMESim\v1300\lib\win32”路径，如下图所示。



再切换到“输入”条目，在“附加依赖项”中添加“AME.lib”，在“忽略特定默认库”中添加“MSVCRTD.lib”，如下图所示。



最后点击“确定”，完成项目的配置。

Step 4：在源代码中添加AMESIM头文件。双击打开Controller.cpp，这是一个空白文件，在文件中添加以下代码：

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <ameutils.h>

以上五个头文件都是项目必须的，前四个为编程环境自带，而ameutils.h由AMESIM提供。添加后，可以看到解决方案资源管理器的“外部依赖项”目录中多了很多头文件，且Visual Studio的**自动代码检查**并未报错，表明ameutils.h已经被项目所识别。

## 源代码结构说明

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <ameutils.h>

/\*TODO：添加需要包含的额外头文件\*/

/\*例如：\*/

#include <function.h>//可将自定义函数写在function.h里面

/\*常量定义\*/

/\*注意：此处定义的参数及变量数量必须与AMESIM模型中DYNEXE0模块定义的数量保持一致\*/

#define NUM\_IP 1//整型参数数量

#define NUM\_RP 2//实型参数数量

#define NUM\_INPUTS 2//输入数量

#define NUM\_OUTPUTS 2//输出数量

/\*TODO：添加需要定义的其它常量\*/

/\*例如：\*/

#define GRAVITY 9.81

#define PI 3.1416

/\*TODO：也可在此位置添加所需的自定义函数，而不放在function.h里\*/

/\*例如最大值函数：\*/

static double Max(double \_x, double \_y)

{

return \_x > \_y ? \_x : \_y;

}

/\*主函数\*/

int main()

{

int num\_rp = NUM\_RP;

int num\_ip = NUM\_IP;

int num\_inputs = NUM\_INPUTS;

int num\_outputs = NUM\_OUTPUTS;

/\*数组iparam用于存储整型参数\*/

int file[2], iparam[NUM\_IP], flag;

/\*数组rparam用于存储实型参数，数组input用于存储控制策略的输入，数组output用于存储控制策略的输出\*/

double rparam[NUM\_RP], input[NUM\_INPUTS], output[NUM\_OUTPUTS];

/\*TODO：添加与控制算法有关的程序内部参数、内部变量定义，以及内部变量的初始化\*/

/\*例如：\*/

int control\_mode;//整型参数

double Kc;//实型参数

double current\_max;//实型参数

double speed = 0;//内部变量

double pressure = 0;//内部变量

double current\_pump = 0;//内部变量

double current\_motor = 0;//内部变量

/\*连接AMESIM模型\*/

/\*将AMESIM模型DYNEXE0模块中定义的实型参数读入到rparam数组中，整型参数读入到iparam数组中\*/

connec\_(file, &num\_rp, rparam, &num\_ip, iparam, &num\_inputs, &num\_outputs);

/\*TODO：将模型参数赋值给之前定义的内部参数，以便从参数名即可获知参数的意义\*/

/\*例如：\*/

control\_mode = iparam[0];

Kc = rparam[0];

Current\_max = rparam[1];

/\*循环执行控制策略算法逻辑\*/

while (1)

{

/\*从AMESIM获取输入，并将其存储到数组input中\*/

getinp\_(&(file[0]), &num\_inputs, input, &flag);

/\*TODO：将input数组中的元素赋值给内部变量，以便从变量名即可获知变量的意义\*/

/\*例如：\*/

speed = input[0];

pressure = input[1];

/\*TODO：添加控制策略的算法逻辑\*/

/\*TODO：将算法输出赋值给output数组中的元素\*/

/\*例如：\*/

output[0] = current\_pump;

output[1] = current\_motor;

/\*向AMESIM发送输出\*/

senout\_(&(file[1]), &num\_outputs, output);

}

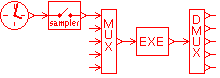
return 0;

}

# 仿真步长与控制周期

AMESIM采用的是变步长仿真机制，每个仿真步长内，外部程序中的控制算法逻辑都会被执行一次。但是实际电液控制系统中，控制算法逻辑具有固定的控制周期，比如0.01s，即控制算法每0.01s才执行一次，输出也是每0.01s即更新一次。如果仿真步长与控制周期不一致（绝对会不一致），就极有可能导致仿真结果出现谬误。

为了解决上述问题，必须从AMESIM建模和控制策略实现两方面进行改进。首先，为DYNEXE0模块增加一个仿真时间采样输入（必须使用零阶保持器），如下图所示。



并将采样器的采样时间设置为与控制周期一致。这样一来，AMESIM的仿真步长就会被限制在控制周期以内（仿真步长不大于控制周期），且必然有一部分仿真时刻与控制时刻重合。再在源代码中插入一段代码：

double time = -1;

connec\_(file, &num\_rp, rparam, &num\_ip, iparam, &num\_inputs, &num\_outputs);

while (1)

{

getinp\_(&(file[0]), &num\_inputs, input, &flag);

if(input[0] == time)//input[0]为仿真时间采样输入

{

senout\_(&(file[1]), &num\_outputs, output);

continue;

}

time = input[0];

/\*TODO：添加控制策略的算法逻辑\*/

这段代码的意义是：当本次仿真时刻与上次仿真时刻相等（零阶保持器特性）时，表明下一控制时刻仍未来临，仿真输出与上一控制周期相同；否则，表明下一控制时刻已经来临，则按照控制算法逻辑计算新的仿真输出，同时更新time变量。

使用上述方法能很好的消除仿真步长与控制周期不一致带来的麻烦，但有的时候，当AMESIM求解器设置的打印间隔（print interval）与控制周期相等时，仿真结果会出现异常（可能是AMESIM的BUG），这时将前述仿真时间采样器的偏置参数（offset，默认为0）设置为采样时间的一半即可。