基于Simulink模型确定计算顺序的研究

李 俊 朱长皓 陆梦寒
(中国科学技术大学自动化系 合肥 230027)

摘 要 将Simulink仿真与代码生成相结合具有很大的实用价值。通过分析Simulink模型特点,将模型转变为C语言代码可达到模型仿真和代码生成相结合的目的,而其中亟需解决的问题是在生成代码过程中计算顺序的确定。通过提取Simulink模型文件中的信息来分析Simulink模型中模块的特性,并将模块间的关系以图的形式存储进而得到Simulink模型中模块间的依赖关系。由模块特性和模块间依赖关系得出两种计算顺序:考虑到底层模块的模块化计算顺序和顾及子系统的层次化计算顺序。通过分析比较这两种计算顺序,发现层次化计算顺序优于模块化计算顺序。最后对Simulink自带的f14模型的测试说明了两种计算顺序是可行的。

关键词 计算顺序,层次化,Simulink仿真,基于模型,依赖关系
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.11.044

Research of Confirming Calculate Order Based on Simulink Model

LI Jun ZHU Chang-hao LU Meng-han
(Department of Automation, University of Science and Technology of China, Hefei 230027, China)

Abstract The combination of Simulink simulation and code generation has great practical value. By analyzing the feature of models, Simulink models can be converted into C language code, which realizes the purpose of combination of Simulink simulation and code generation. And the primary problem is the confirmation of calculate order during the code generation process. From the information extracted from Simulink models file, the feature of module in Simulink model is analyzed and the dependence between modules from the relationship of the modules which is stored in graphsis is gotten. From the feature of module in Simulink model and dependence, two calculated orders are goten: the calculate order based on module which takes into account the underlying module and the calculate order based on hierarchy with consideration of subsystem. By analyzing and comparing the two calculate order, it was found that the calculate order based on hierarchy is better than the calculate order based on module. Finally by testing the model of f14 in Simulink, it was found that the two calculate orders are feasible.

Keywords Calculation order, Hierarchy, Simulink simulation, Based-models, Dependence

1 引言

Matlab是Mathworks公司推出的一种科学计算仿真软件。在工程领域中,可以用Matlab对实际系统进行数值计算和仿真,特别是Matlab集成图形界面仿真软件Simulink[1-3]。Simulink是一种面向多域仿真并基于模型设计的框模块环境,支持系统级设计、仿真、自动代码生成以及嵌入式系统的连续测试和验证[4]。

为了使离线仿真能对接口和通信等实时参量进行评价,Simulink增加了Real Time Workshop(RTW)工具箱。文献[5]中介绍了RTW在仿真中的应用。文献[6]则介绍了RTW的机制。文献[7]阐述了RTW在仿真中的应用以及实现Simulink图形模型到C语言的转换,相应的相关文献则研究了RTW代码生成技术[8]。通过RTW将Simulink模型转换为C语言代码能满足实时仿真的不同需求,有利于缩短项目的开发周期和减少项目开发成本以及硬件开发次数,具有较好的经济价值和市场前景。

但通过分析RTW[9]生成的代码可以看出RTW生成代码有些不足,以Simulink自带模型f14经RTW生成的部分代码说明不足之处。如图1所示,1-11行是结构体*ContinuousStates_f14*,由注释可看出该结构中包含各个模块的信息,耦合性比较强,不便分开分析。后面截取的代码描述从*ContinuousStates_f14*结构体中的元素到x数组中的元素所经过的多次转换,可以看出指针指向路径很长,在具体的代码中分析比较难。综合而言,RTW生成代码有以下不足:(a)代码耦合性比较强,无法分拆模型的各个部分;(b)指针指向繁琐;(c)结构复杂,不便分析。

针对RTW生成代码的不足,我们构建新的代码生成工

到稿日期:2014-01-20 返修日期:2014-03-24 本文受国家自然基金重点项目(61331017)资助。
李 俊(1973-),男,博士,副教授,主要研究方向为先进控制与优化、网络传播系统与控制,E-mail:Ljun@ustc.edu.cn;朱长皓(1989-),男,硕士,主要研究方向为先进控制与优化;陆梦寒(1989-),男,硕士,主要研究方向为网络传播系统与控制。

具。该工具可将 Simulink 模型转换为 C 语言代码并克服以上不足。代码生成流程如下：(a)分析模型文件，提取模型结构信息、模块信息及模块间的关联信息；(b)以 C 语言函数形式重构模块功能；(c)由提取的信息确定计算顺序；(d)综合上面信息生成代码。

```
typedef struct {
  real_T TransferFcn2_CSTATE; /* '<S1>/Transfer Fcn.2' */
  real_T TransferFcn1_CSTATE; /* '<S1>/Transfer Fcn.1' */
  real_T ActuatorModel_CSTATE; /* '<Root>/Actuator Model'
  */
  real_T Wgustmodel_CSTATE[2]; /* '<S3>/W-gust model' */
  real_T Qgustmodel_CSTATE; /* '<S3>/Q-gust model' */
  real_T AlphasensorLowpassFilter_CSTATE;/* '<S2>/Alpha-sen-
  sor Low-pass Filter' */
  real_T StickPrefilter_CSTATE; /* '<S2>/Stick Prefilter' */
  real_T PitchRateLeadFilter_CSTATE; /* '<S2>/Pitch Rate Lead
  Filter' */
  real_T Proportionalplusintegralcompens;/* '<S2>/Proportional
  plus integral compensator' */
} ContinuousStates_f14;
ContinuousStates_f14 f14_X;
f14_M->ModelData.contStates = ((real_T *) &f14_X);
rtsiSetContStatesPtr(&f14_M->solverInfo,&f14_M->ModelDa-
ta.contStates);
#define rtsiSetContStatesPtr(S,cp) ((S)->contStatesPtr = (cp))
typedef ssSolverInfo RTWSolverInfo;
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
real_T *x = rtsiGetContStates(si);
#define rtsiGetContStates(S) *((S)->contStatesPtr)
for (i=0;i<nXc;i++){
    x[i]=y[i]+(f0[i]*hB[0]+f1[i]*hB[1]+f2[i]*hB[2]+f3
        [i]*hB[3]+f4[i]*hB[4]+f5[i]*hB[5]);
  }
```

图 1　Simulink 模型 f14 经 RTW 生成代码截取的部分代码(非连续)示例

计算顺序的确定是代码生成中最关键的环节，直接影响着代码的运行结果。确定计算顺序的根本准则是不能改变 Simulink 模型中模块间的依赖关系和运算规律。本文分析 Simulink 模型的依赖关系和运算规律并基于不同的分析粒度，提出了模块化计算顺序和层次化计算顺序。

2　计算顺序确定前预备知识

2.1　模块特性及依赖关系

Simulink 模块库中模块有着不同的功能和特性。对于计算顺序而言，所需分析的特性有 7 点：(a)直通性(模块的输出与输入关系可直接由线性表达式表示)；(b)模块的输入个数；(c)延迟性；(d)内部更新状态；(e)信号源模块；(f)控制信号；(g)输入模块或输出模块。

Simulink 模型仿真时，模块间有数据流，因此模块间有数据依赖关系，有些模块的输出会受触发信号控制，即有控制依赖关系。数据依赖关系表现在模块的输出值依赖其输入值。控制依赖关系表现在一个模块的输出作为另一个模块的触发信号，后一个模块只有在触发信号有效时才更新输出信号，反之保持原值。

计算顺序确定的基本原则是在保持模块特性的前提下不违背模块的依赖关系。体现在 7 点：(a)直通模块(具有直通性的模块)可直接计算；(b)多输入模块输出等其输入更新完再计算；(c)有延迟特性的模块输出值计算要考虑次序；(d)有内部更新状态的模块需要更新其状态；(e)信号源模块可优先计算；(f)有控制信号模块要满足控制依赖关系；(g)输入模块或输出模块在模块化计算顺序和层次化计算顺序表现不同。

2.2　抽象模型图的建立

模块之间的依赖关系是计算顺序确定的基础，而依赖关系与模型的结构相关，这就引入了建立抽象模型图的概念。抽象模型图的建立是按照图论的方法将模型图抽象成图的过程。

有向图是由顶点集合和边集合组成的数据结构[10-12]，通常用 $G(V,E)$ 来表示，其中顶点集合和边集合分用 $V(G)$ 和 $E(G)$ 表示。$V(G)$ 中元素为顶点，用 u、v 表示；$E(G)$ 中元素为有向边，用〈u、v〉顶点对表示，其中 u 为起点，v 为终点。顶点的入度是以 u 为终点的有向边的数目，记为 $id(u)$。顶点的出度是以 u 为起点的有向边的数目，记作 $od(u)$。

抽象模型图的建立过程需要分析模型图文件，以从中提取模块信息及模块间的关联信息。遇到子系统时，基于模块的抽象模型图的建立需将子系统解封装，而基于层次化的抽象模型图的建立直接将子系统作为节点对待。

2.2.1　基于模块的抽象模型图的建立

将 Simulink 模型抽象成图，图中的节点对应模型中的模块，边对应模块间的依赖关系，即〈u、v〉表示 v 依赖 u。如果有子系统模块，抽象模型图建立前需要将子系统解封装。在解封装子系统时，将子系统一层层展开，得到基于模块的抽象模型图。

基于模块的抽象模型图建立的具体过程如下：

(1)将所有的模块，即模型文件中的 *Block*，抽象为图的节点，把这些节点加入顶点集合 $V(G)$ 中；

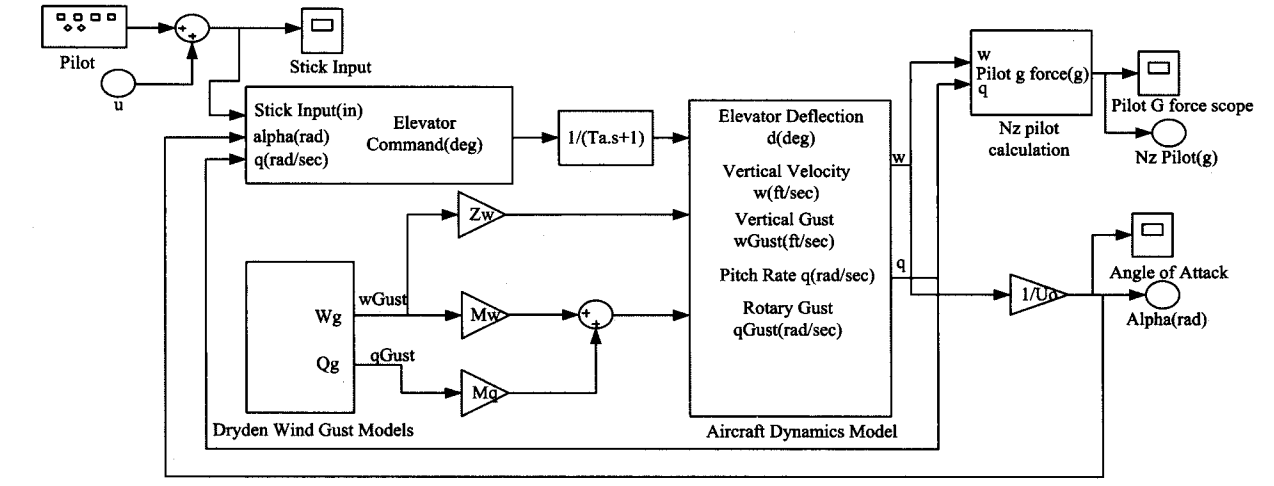
(2)将模块间的连线，即模型文件中 *Line*，抽象为图的边，把这些边加到边的集合 $E(G)$ 中；

(3)标记边 $E(G)$ 中的方向信息，在模型文件 *Line* 中，一条 *Line* 中有 *SrcBlock* 和 *DstBlock* 信息，将 *SrcBlock* 所对应的节点作为 $E(G)$ 元素〈u,v〉中的起始节点 u，将 *DstBlock* 对应的节点作为〈u,v〉中的终止节点 v，如果有一 *SrcBlock* 对应多个 *DstBlock*，就需要同样数量的元素〈u,v〉来对应；

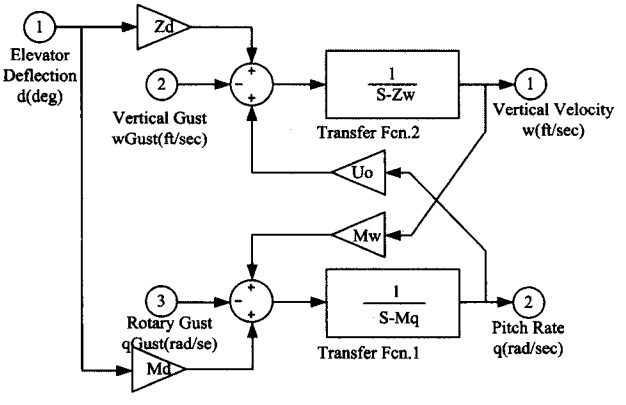
(4)将 $V(G)$ 和 $E(G)$ 都加到 $G(V,E)$ 中，得到一张有向图。

将模型中所有模块及模块间连线信息处理完后，就得到

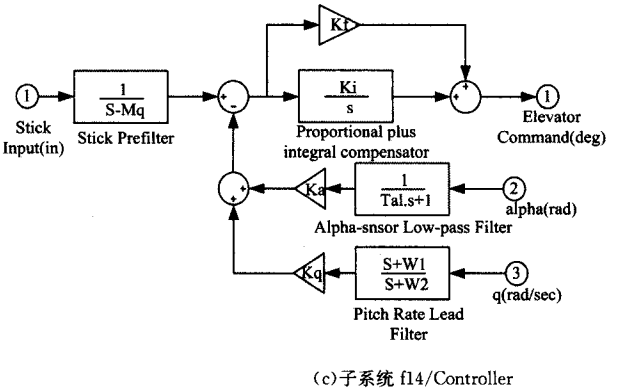
一张基于模块的抽象模型图。从抽象模型图可知整个模型的拓扑结构，由边的信息可知节点的依赖关系。图 2 是 Simulink 自带模型 f14；图 3 是图 2 所示模型图的基于模块的抽象模型图。



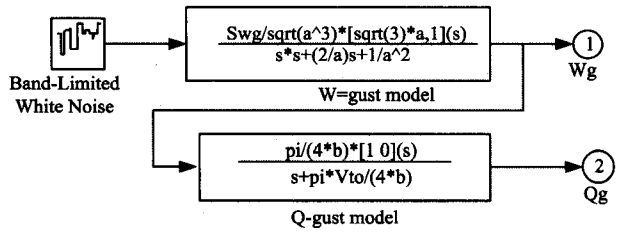
(a) 模型 f14



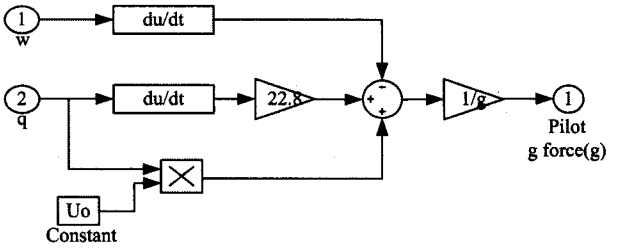
(b)子系统 f14/Aircraft Dynamics Model



(c)子系统 f14/Controller



(d)子系统 f14/Dryden Wind Gust Models



(e)子系统 f14/Nz pilot calculation

图 2　模型 f14 及其子系统图

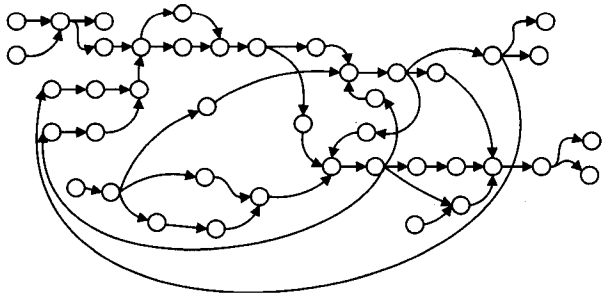


图 3　模型 f14 的基于模块的抽象模型图

2.2.2　基于层次的抽象模型图的建立

考虑到子系统是实现一定功能的集合，可将子系统作为模块对待，无需解封装。在分析子系统时，再将子系统内部系统递归地建立基于层次的抽象模型图。

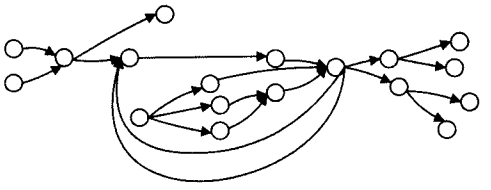
下面是基于层次的抽象模型图建立的具体过程：

(1)将根层次的模型抽象为抽象模型图，具体方法与基于模块的抽象模型图方法一样，在碰到子系统时将子系统作为普通的模块抽象成节点，并加入 $V(G)$ 中。

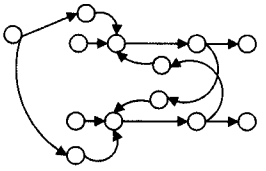
(2)将在根层次中的子系统递归地建立基于层次的抽象模型图，在子系统内部将该子系统作为根层次(相对该子系统而言)，并按照(1)中方法建立该子系统。

(3)如此递归，直到所有的模块和连线信息都被图形化，即得到基于层次的抽象模型图。

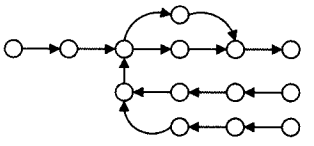
在模型图经过基于层次的抽象模型图建立后，就得到一张整个模型的根层次抽象模型图和各个子系统的抽象模型图。图 4 是图 2 模型的层次化的抽象模型图。



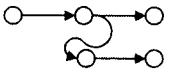
(a) 模型 f14 根层次抽象模型图



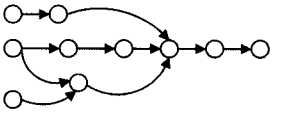
(b)子系统 f14/Aircraft Dynamics Model 抽象模型图



(c)子系统 f14/Controller 抽象模型图



(d) 子系统 f14/Dryden Wind Gust Models 抽象模型图



(e)子系统 f14/Nz pilot calculation 抽象模型图

图 4 模型 f14 及其子系统抽象模型图

3 计算顺序的确定

从 Simulink 模型图的抽象模型图中可以得到模块间的依赖关系，再结合模块的特性，分别针对基于模块的抽象模型图和基于层次的抽象模型图两种情况，可分别得到模块化计算顺序的规则和层次化计算顺序的规则。

3.1 模块化计算顺序的确定

由基于模块的抽象模型图可知模型的拓扑结构和模块间的依赖关系，再结合模块特性得到确定模块化计算顺序的规则：

(1)从基于模块的抽象模型图中分析模型是否有只包含直通模块的环路[13]

首先从抽象模型图中判断是否有环路。如果没有环路，可直接从输入模块或信号源模块开始计算，沿着主通道依次到最后的输出模块或显示模块。

如果有环路，通过环路中的模块特性判断环路是否全部为直通模块。如果是，就无法确定计算顺序来生成代码；否则按照后面的规则确定计算顺序。

(2)多输入模块

由抽象模型图的边集合可知，多输入模块依赖其输入端口连接的模块。模块抽象成节点后，其输入端口数量是该节点的入度。因此需要定义一个变量来描述入度信息。入度初始化为模块的输入端口数，每得到一个输入值入度减 1。入度为零时，计算模块的输出值并将入度值复原。

(3)初始入度为零的节点、初始出度为零的节点

初始入度为零的节点一般为信号源模块或输入模块。由于其没有输入端口，可在计算顺序中先计算。初始出度为零的节点一般是显示模块(如 Scope)或输出模块，由于没有其他依赖该模块的模块，因此在计算顺序中后计算。

(4)有内部状态的模块

Simulink 模型库中存在一些有内部状态的模块。这些内部状态更新不仅与其依赖模块的输出值相关，也与状态的前一轮值及模型所用算法有关。因此在计算顺序中，这类模块可以作为环路中的首选模块(第一个计算的模块)来分割环路。

(5)与时间相关的模块

Simulink 模型库中有一些模块的更新与时间有关。分析这类模块时，需把握其时间更新周期，在更新时间点更新模块输出值。

(6)其他模块

在确定计算顺序时满足抽象模型图中边集合所表示的依赖关系即可。

3.2 层次化计算顺序的确定

由前面得到的基于层次的抽象模型图，可知模型的各个层次的拓扑结构和模块间依赖关系，再结合模块特性可确定层次化计算顺序的规则：

(1)从基于模块的抽象模型图中判断是否有只包含直通模块的环路

这与模块化计算顺序一样。若没有只包含直通模块的环路，就按后面规则来确定计算顺序。

(2)有子系统层次的计算顺序

由该层次的抽象模型图的边集合得到依赖关系。依照模块化计算顺序分析该层次的计算顺序。但在遇到子系统模块时，进入子系统后需将子系统内部所有模块计算顺序确定完后才可离开子系统，然后去确定该层次中其他模块的计算顺序。

(3)子系统模块的输入和输出

子系统一般是多输入多输出的系统。按照依赖关系，计算子系统内部模块需要等待子系统的输入端口的值全部更新完；子系统的输出要等子系统内部模块状态更新完并且子系统的输出全部计算完才能离开子系统。

(4)子系统内部的子系统

由基于层次的抽象模型图可知子系统也抽象为抽象模型图，可将子系统中的子系统作为普通模块依照模块化计算顺序来确定子系统内部模块的计算顺序。

(5)根层次首选计算模块

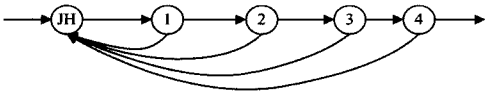
模型中没有环路，就以输入模块或信号源模块作为首选模块。在有环路的情况下，以有内部状态的模块或子系统来作首选模块。

(6)其他各层次中模块计算顺序确定可参照模块化计算顺序

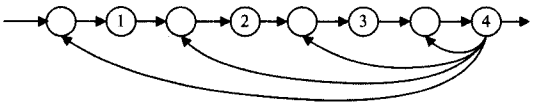
(7)复杂图中首选计算模块

规则(5)适用于该层次只有一个环路的情况，对于比较复杂的抽象模型图的首选计算模块按照图 5 所示 4 种情况来确定。

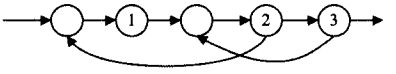
图 5(a)多个模块的输出都反馈到模块 JH 的输入端，则首选计算模块应选这些回路的公共模块 1；图 5(b)中同一个模块的输出都反馈到不同模块输入端，则取回路的最后一个公共模块 4 作为首选计算模块；图 5(c)中将交叉的两个回路的公共模块 2 作为首选计算模块；图 5(d)中两个回路无公共模块，将两个回路中有内部状态的模块作为首选模块并且第一个回路在第二个回路前计算。更复杂的环路一般由上面 4 种情况组合而成，分析时可将复杂环路分拆成不同情况来分析。



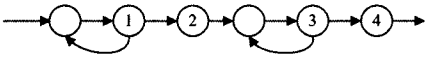
(a)多条反馈回到同一模块的输入端



(b)同一模块输出反馈到不同模块输入端



(c)不同模块的输出反馈到不同模块输入端,两回路有公共模块



(d)不同模块的输出反馈到不同模块输入端,两回路间无公共模块

图 5　4 种复杂抽象模型图结构

4　计算顺序的比较及测试

4.1　与 RTW 生成代码的比较

由引言中分析可知 RTW 生成的代码有 3 个方面不足。以两种计算顺序为基础所构建的新的代码生成工具,均能克服 RTW 的不足之处。

首先,以模块为单位,不像 RTW 将模块的不同信息分拆再组合起来,而是将模块的信息放在一起,以函数来重构模块功能以达到解耦和的目的。其次,生成的代码指针指向明确,没有 RTW 生成代码中指针指向路径过长的问题,分析起来很明确。最后,由前面两点所带来的效应使得由这两种计算顺序为基础所构建的代码生成工具生成的代码结构简单易懂,特别是层次化计算顺序将子系统作为功能群,解耦性较好,为代码的串并行分析提供方便。图 6 是生成的关于计算传递函数的部分代码,图 6(a)是新的代码生成工具生成的代码,图 6(b)是 RTW 生成的代码。可以看出新的代码生成工具生成的代码比较简洁,其不需要调用很多函数和设置些变量。

```
for (i=0;i<nf;i++)
{
    nq+=nX[i];
}
dx[nq]=inputb;
for (i=nX[nf]-1;i>0;i--){
    dx[nq]+=A[i]*((*X)[nq+i]);
    dx[nq+i]=(*X)[nq+i-1];
}
dx[nq]+=A[0]*((*X)[nq]);
hB[0]=h*rt_ODE5_B[0][0];
for (i=0;i<nX[nf]);i++)
{
    (*X)[nq+i]=y[nq+i]+(f[0][nq+i]*hB[0]);
}
out=D*input c+C[0]*((*X)[nq]);
i=1;
while (--n){
    out+=C[i]*((*X)[nq+i]);
    i++;
    }
return out;
```

(a)新的代码生成工具生成的代码

```
time_T tnew=rtsiGetSolverStopTime(si);
time_T h=rtsiGetStepSize(si);
real_T *x=rtsiGetContStates(si);
OEDG5_IntgData *id=(OED5_IntgData*)rtsiGetSolverData(si);
real_T *y=id->y;
real_T *f0=id->f[0];
real_T *f1=id->f[1];
real_T *f2=id->f[2];
real_T *f3=id->f[3];
real_T *f4=id->f[4];
real_T *f5=id->f[5];
real_T hB[6];
int_T i;
int_T nXc=10;
rtsiSetSimTimeStep(si,MINOR_TIME_STEP);
(void) memcpy(y,x,nXc*sizeof(real_T));
rtsiSetdX(si,f0);
f14_derivatives();
hB[0]=h*rt_ODE5_B[0][0];
for (i=0;i<nXc;i++){
  x[i]=y[i]+(f0[i]*hB[0]);
}
rtsiSetT(si,t+h*rt_OED5_A[0]);
rtsiSetdX(si,f1);
f14_output(0);
f14_derivatives();
for (i=0;i<=1;i++)
  hB[i]=h*  rt_ODE5_B[1][i];
for (i=0;i<nXc;i++){
  x[i]=y[i]+(f0[i]*hB[0]+f1[i]*hB[1]);
}
rtsiSetT(si,t+h*rt_ODE5_A[1]);
rtsiSetdX(si,f2);
```

(b)RTW 生成的代码

图 6　关于计算传递函数的部分代码

4.2　两种计算顺序的比较

分析两种计算顺序,可知层次化计算顺序是在模块化计算顺序的基础上发展而来的,二者之间有着联系及区别。两种计算顺序都要确定是否有只包含直通模块的环路存在。但在将模型图建立成抽象模型图时,模块化计算顺序直接将所有的模块作为节点,而层次化计算顺序将子系统也作为节点递归地建立层次的抽象模型图。层次化计算顺序的确定会参照模块化计算顺序的规则。

由于分析的粒度不同,层次化计算顺序有些特性优于模块化计算顺序。主要体现在下面 4 点:(1)子系统之间的独立性,层次化计算顺序中不需要将子系统解封装,每个子系统内部的模块是内聚的,子系统间耦合性比较小;(2)子系统可以单独分析,可在该子系统的层次分析该子系统;(3)分析整个模型的原理更加明朗,层次化计算顺序不拘泥于模块,将子系统作为实现某种功能的模块,分析时更能抓住主要部分;(4)

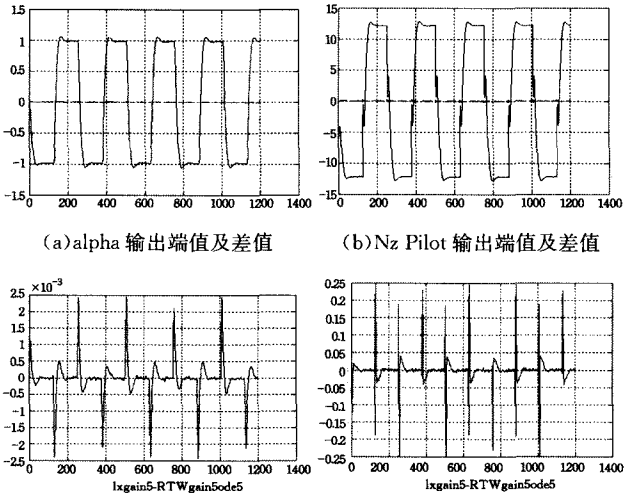
子系统之间低耦合为对生成代码作串并行分析奠定了基础。

4.3 两种计算顺序的测试

以 Simulink 自带的模型 f14 为例测试两种计算顺序并和 Simulink 仿真结果比较。在 Simulink 中打开 f14 模型，在两个输出端 alpha 和 Nz Pilot 都加上 Toworkspace 模块，以数组的形式记录两个输出端的值，并将两个输出端命名为 RTWgain5ode5 和 RTWgain2ode5。在 RTW 中选用 ode5 算法并将仿真步长设为 0.05，仿真记录两个输出端值。

在 VS2010 中建立两个工程分别用来测试两种计算顺序。将模块的实现函数以头文件的形式包含到工程中，在主函数中实现整个模型的计算顺序。运行这两个工程得到模型的两个输出端的值。将模块化计算顺序结果与 Simulink 仿真结果比较，发现值的误差在万分之一以内，这是可行的。下面主要分析层次化计算顺序。

将层次化计算顺序对应于 alpha 和 Nz Pilot 的输出端分别命名为 lxgain2 和 lxgain5。利用 Matlab 画图的功能，将层次化计算顺序的结果和 Simulink 仿真的结果以及二者的差值以图形的形式表现出来并进行比较分析。如图 7 所示，图 7(a)中实线型是层次化计算顺序的 lxgain5 值，虚线型是 Simulink 仿真出来的 RTWgain5ode5 值，中间双划线型是 lxgain5 与 RTWgain5ode5 的差值，即 lxgain5-RTWgain5ode5；图 7(b)中实线型是层次化计算顺序的 lxgain2 值，虚线型是 Simulink 仿真出来的 RTWgain2ode5 值，中间双划线型是 lxgain2 与 RTWgain2ode5 的差值，即 lxgain2-RTWgain2ode5；图 7(c)是 lxgain5 与 RTWgain5ode5 的差值放大；图 7(d)是 lxgain2 与 RTWgain2ode5 的差值放大。



(a)alpha 输出端值及差值　(b)Nz Pilot 输出端值及差值

(c)alpha 端输出差值放大图　(d)Nz Pilot 端输出差值放大图

图 7　模型 f14 层次化计算顺序结果与 Simulink 仿真结果的比较

从图 7(a)和图 7(b)中可以看出，两个输出端的值几乎没有差别，图 7(a)中 alpha 输出端的 Simulink 仿真结果的实线都盖住层次化计算顺序结果的虚线，图 7(b)中 Nz Pilot 输出端的 Simulink 仿真结果的实线几乎都盖住层次化计算顺序结果的虚线，只有放大才能看到一些虚线点；图 7(a)和图 7(b)中差值的双线几乎是零值；图 7(c)和图 7(d)分别是 alpha 端和 Nz Pilot 端输出的层次化计算顺序和 Simulink 仿真出来值的差值。从图 7(c)中统计得知 alpha 端输出差值中最大的误差在千分二点三内(1201 个点中有 137 个点超过 0.0005)，其他点在 0.0005 内，即误差在万分之四点七内；从图 7(d)中统计得知 Nz Pilot 端输出差值中最大的误差在百分之一点九内(1201 个点中有 58 个点超过 0.05)，其他点在 0.05 内，即误差在千分之三点八内。因此层次化计算顺序是可行的。

结束语　本文在分析 Simulink 模型的基础上提出两种计算顺序，一种是模块化计算顺序，一种是层次化计算顺序。两种计算顺序在代码生成过程中都起着非常重要的作用，影响着代码生成的构架。由这两种计算顺序所构建的代码生成工具改进了 RTW 在代码生成 4 个方面的不足。比较分析两种计算顺序，发现层次化计算顺序优于模块化计算顺序。最后测试两种计算顺序并与 Simulink 仿真结果比较可知两种计算顺序是可行的。由这两种计算顺序为基础所构建的代码生成工具可用于嵌入式开发，因此可将 PC 机上的 Simulink 模型仿真和嵌入式平台中代码运行有机结合起来。

参 考 文 献

[1] Mokhtari M, Marie M. MATLAB 与 SIMULINK 工程应用[M]. 北京：电子工业出版社，2002

[2] 黄永安，马路，刘慧敏. MATLAB7.0/Simulink6.0 建模仿真开发与高级工程应用[M]. 北京：清华大学出版社，2005

[3] 李颖. Simulink 动态系统建模与仿真(第 2 版)[M]. 西安：西安电子科技大学出版社，2009

[4] Works M. MathWorks® Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow®. R2011b [OL]. http://www.docin.com/p-173565475.html

[5] Shen E, Zhang Tao, Huang Liang-wei, et al. A real-time simulation system for satellite based on RTW and VxWorks[C]// 2010 3rd International Symposium on Systems and Control in Aeronautics and Astronautics (ISSCAA). IEEE, 2010: 859-864

[6] 王勃. Real-time Workshop 机制研究——动态数据流模型代码生成器的研究与实现[D]. 成都：电子科技大学，2008

[7] 任传俊，蒋志文. Real-Time Workshop 实时仿真研究与应用[J]. 计算机仿真，2007，24(8)：268-271

[8] 卞学飞. 基于 DSP 的 RTW 代码自动生成技术研究[D]. 成都：电子科技大学，2012

[9] Math Works. Real-Time Workshop® 7 Target Language Compiler[OL]. http://www.docin.com/p-81745259.html

[10] 魏丽侠，王涛. 图论及其应用[M]. 徐州：中国矿业大学出版社，2012

[11] 王桂平，王衍，任嘉辰. 图论算法理论、实现及应用[M]. 北京：北京大学出版社，2011

[12] 徐俊明. 图论及其应用[M]. 合肥：中国科学技术大学出版社，2010

[13] 邱杰，原渭兰. 数字计算机仿真中消除代数环问题的研究[J]. 计算机仿真，2003，20(7)：33-35