MAST 建模讲义

一、引言

所谓仿真，就是描述一个实际系统的数学模型的特征，对仿真器而言，系统的模型无非就是一系列的代数或微分方程（组）。MAST 语言主要是用来创建模拟、数字或系统模型的，而用 MAST 语言建模实际上就是指定要被仿真的模型，它实际上就是要建立一系列的方程，因此用 MAST 语言建模的核心就是用线性（或非线性）的代数、微分方程（组）来描述对象的特征。它包括电、机械、光和流体等。

从上面的定义可以看出，Saber 仿真器并不是单纯的一个电路模型仿真器，从理论上讲如果你能用 MAST 语言建立任何建模，通过 Saber 仿真器就能进行仿真，但在实际情况是它讲受系统硬件的限制。仿真器实际上要作的工作就是解方程。

既然 Saber 仿真器的主要工作就是解方程，那么可不可以将描述系统特征的数学方程用 MATLAB 来求解呢？因该说是可以的。但是有时建立一个系统的模型非常困难，特别是一些结构复杂的系统。用 MAST 建模时可以首先建立系统中元件的模型，然后将各个元件按照一定的要求连接起来就构成系统模型，因此在这种情况下描述系统模型方程由仿真器自动完成。

正如上所述，只要能写出描述对象特征的方程就能用 MAST 语言建模，因此 MAST 语言不仅可以建立模拟元件的模型，还可以建立数字元件的模型，对于数字模型是用元件在各离散时刻的离散值来描述的。

在 MAST 语言中，被 Saber 仿真器使用的最核心的单元就是模板(template)，在创建模型中，模板是分层结构的。所谓分层结构就是在创建模板中可以引用其它模板。这样的结构有几个好处：

1）在你创建模板的过程中可以直接调用 Saber 库中元件模型，这样将大大的减少你编写模板的工作量。

2）对于你经常使用到的电路结构（该结构中可以包括其它电路结构），可以将其构成一个子系统，而其它模板可以调用这个子模板。

3）可以建立一个顶层模板，在该模板中调用系统中的其它所有模板，它只反映各模板之间的连接及各模板所需要传递的参数。该模板通常叫网表。这样在仿真中修改参数就很方便。

模板的命名。在这个问题上有两点需要注意，1、模板的扩展名必须是 .sin，即 templatename.sin。2、模板名必须以字母开头。

如果是用 SPICE 语言写的模板，Saber 仿真器也可以对它进行仿真，但之前必须把它转换为 Saber 格式，用 spitos 来转换。

根据仿真对象的不同，所建模板的表现形式也不同，描述模板程序不同，构成模板的复杂程度不同，但作为模板它们都有一些共同的特征，首先它们都是由 MAST 语言写成的（Saber 的模型可以由其它语言写，如 VHDL，但那不是本课程的内容），其次是模板的结构具有相似性。与此相对应，在本部分内容中应注意 MAST 语言本身的特征以及模板的一些结构特征。

//

二、模板的结构

一个模板可能有部分以下的一个或几个部分，也可能有以下的全部

Units definitions

Connection point definitions

Template header

Header declarations

{

local declarations

Parameters sections

When statements

Values section

Control section

Equations section

}

在编写模板时，没有上面顺序的限制，可以按任意顺序编写，但有两点需特别注意，在使用一个量之前必须首先定义这个量，你定义的量的位置就决定了它是全局的或是局部的。如果你要在模板中引用文件，你可以在任何地方引用文件，关于文件的引用将在后面作简单介绍。但是为了增加程序的可读性，建议编写模板程序时采用上述顺序。另外，如果要调用程序，如果该文件为全局调用，建议调用句放在 header declarations 部分，如果该文件为局部调用，建议放在 local declarations 部分。

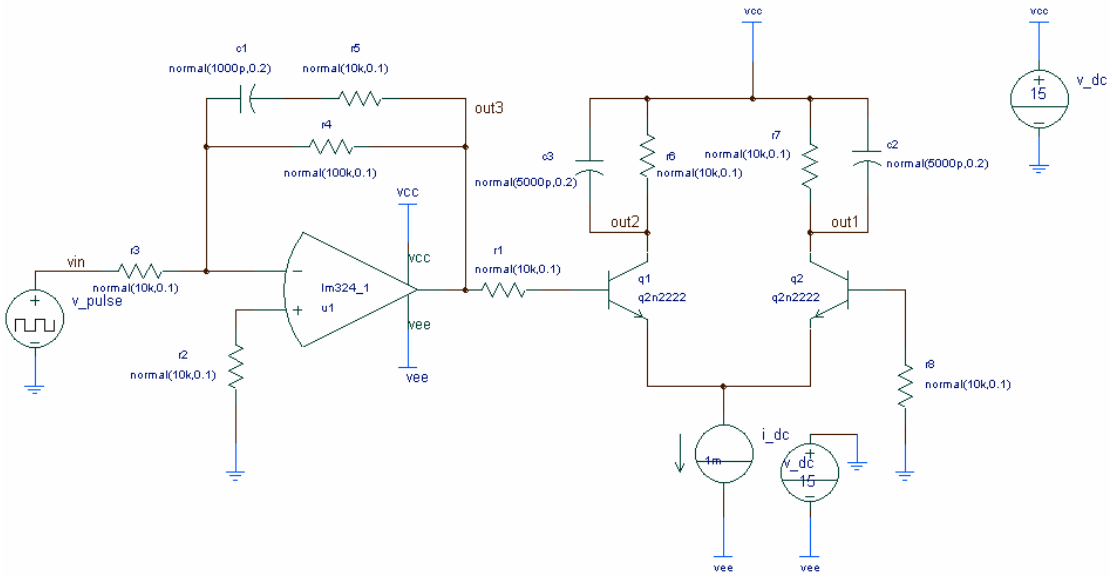


图 1 一个差分放大器的原理图

你的模板可能只有以上的某些部分，至于那些部分是你需要的则是根据你模型的需要。例如，顶层模板就不包括模板头，头定义以及函数体{ }。下面将分别介绍在模板中各部分的情况。如下例：

```
c.c2 p:vcc m:out1 = c=normal(5000p,0.2)
c.c3 p:vcc m:out2 = c=normal(5000p,0.2)
c.c1 p:n4 m:n6 = c=normal(1000p,0.2)
r.r1 p:out3 m:n8 = rnom=normal(10k,0.1)
r.r4 p:n4 m:out3 = rnom=normal(100k,0.1)
r.r3 p:vin m:n4 = rnom=normal(10k,0.1)
r.r5 p:n6 m:out3 = rnom=normal(10k,0.1)
r.r6 p:vcc m:out2 = rnom=normal(10k,0.1)
r.r7 p:vcc m:out1 = rnom=normal(10k,0.1)
```

```
r.r8 p:n5 m:0 = rnom=normal(10k,0.1)
r.r2 p:n9 m:0 = rnom=normal(10k,0.1)
i_dc.isink p:n1 m:vee = dc_value=1m
q2n2222.q1 b:n8 c:out2 e:n1
q2n2222.q2 b:n5 c:out1 e:n1
v_pulse.v2 p:vin m:0 = period=10m, ac_phase=0, initial=-.01, delay=1m, width=5m,
\
        ac_mag=.1, tr=1u, tf=1u, pulse=0.01
v_dc.sym12 p:0 m:vee = dc_value=15
lm324_1.u1 vee:vee vcc:vcc inp:n9 inm:n4 out:out3
v_dc.v3 p:vcc m:0 = dc_value=15
```

这就是与图 1 对应的 MAST 语言的结构。从上面这段程序可以看出，这段程序就是一个网表。下面是一个电阻模型

```
template resistor p m = res
electrical p,m    number res
{
equations   {       i(p->m) += (v(p)-v(m))/res    }
}
```

下面是一个电容模型

```
template capacitor p m = cap
electrical p,m
number cap
{
equations   {       i(p->m) += d_by_dt(cap*(v(p)-v(m)))    }
}
```

在上述网表中所用的电容的模型如下：

```
element template c  p m = c,model,l,w,ic,esr,rleak,tc,tnom,
                ratings,rth_ja,part_type,part_class
                                                                    //template header
#...declaration of connections:

electrical  p,m

process..imodel model = (tc1 = 0,tc2 = 0)    #  Process  model  for  defining
capacitance.

#...declaration of argument

number  c = undef,       # Capacitance.
            esr = 0,         # Equivalent Series Resistance
            rleak = inf,# Parallel leakage resistance
            l = 0,           # Optional Capacitor length specification.
```

```
            w = 0,          # Optional Capacitor width specification.
            ic = undef,     # initial condition on vcap
            tc[2] = [0,0],  # Temperature coefficients.
            tnom = 27,      # Nominal temperature
            rth_ja = undef  # thermal resistance

# Bring in temperature - in degrees celsius
external number temp, include_stress, c_tol, c_vmax, c_vrmax
external standard..pdist pdist

struc {
    number  vmax = undef,       # Max. forward voltage
                vrmax   = undef,# Max. reverse voltage
                tjmax   = undef,# Max. junction temperature
                tjmin   = undef,# Min. junction temperature
                pdmax_ja= undef # Max. power dissipation
    } ratings=()

string part_type    = "capacitor",    # type of the device
       part_class   = "generic"       # class of the device

export val p     pwrd       # instantaneous power dissipation
export val tc    tempj  # instantaneous junction temperature
export val joule energy # energy stored in capacitor (1/2)*(c*v**2)
                                                        // header declaration
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# Start the definition

{
    #...Make working array for tc assignment in parameter section
    #...Assign the current instance values to initialize
    number wtc[2] = [tc[1],tc[2]]   # Temperature coefficients

    #...declare internal node
    electrical pm

    #...Quantities useful for output:

    val v v,            # Total Capacitor voltage.
          vesr,         # Voltage across ESR
          vcap          # Voltage across capacitor and leakage resistor
    val q q             # Capacitor charge.
    val i ileak,    # leakage current (non-dissipative)
```

```
        iesr          # equiv. series resistance current (dissipative)
    val nu ceff

    #...Define the "undefined" number

    number  cap,    # Final value of capacitance.
                xw,     # Final value of capacitor width.
                xl,     # Final value of capacitor length.
                dl,     # Final value of geometry reduction.
                xesr,   # Final value of equivalent series resistance.
                xgleak, # Final value of leakage conductance.
                rth_eff,# Final value of thermal resistance.
                pdmax,  # Final value of maximum power dissipation rating.
                xvmax,  # Final value of ratings->vmax
                xvrmax, # Final value of ratings->vrmax
                xtjmax, # Final value of ratings->tjmax
                xtjmin  # Final value of ratings->tjmin


    number tempj_tnom=25

    group{v,vesr,vcap,q,ileak,iesr,ceff} hsp
                                                        // local claration
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

parameters {

    #...Check input parameters.
    if ( (c == undef) & ((model->cj == 0)|(model->cjw == 0)|(l == 0)|
        ((w == 0)&(model->wdf == 0))) ) {
        # Capacitance is not specified.
        saber_message("TMPL_S_ALT_SPEC",  instance(),
            "Capacitance","c","model->cj, model->cjw, l, and w")
        }
   # If the tc argument (array) is not defined, use the model->tc's
   # else, the working array wtc has already been set to equal the
   # instance values 'arguments' for tc[1,2]

   if ( (tc[1] == 0) & (tc[2] == 0)  ) {
      wtc = [model->tc1,model->tc2]
      }


    if (c ~= undef) {
```

```
    #...Capacitance specification.
    if(c ~= inf) {
        cap = distfunc(c,c_tol,pdist)
        cap = cap*(1 + wtc[1]*(temp-tnom) + wtc[2]*(temp-tnom)**2)
        }
    else {
        saber_message("TMPL_S_RANGE_NE_INF",instance(),"c")
        }
    }
else {
    #...Process specification.
    #...Check input parameters.
    if ((model->dl == undef)|(model->dl < 0)) {
        dl = 0
        }
    else {
        dl = model->dl
        }

    if ( ((w == 0)|(w == undef)) &
        ((model->wdf == 0)|(model->wdf == undef)) ) {
        saber_message("TMPL_S_ALT_SPEC",  instance(),
            "capacitor width","w","model->wdf")
        }
    #...Take into account the geometry reduction of the length.
    xl = l - dl
    if (xl <= 0) {
        saber_message("TMPL_S_POS",instance(),
            "effective capacitor length")
        }

    #...Take into account the geometry reduction of the width.
    if ((w > 0)&(w ~= undef)) {
        xw = w - dl
        }
    else {
        xw = model->wdf
        }

    #...Calculate the capacitance from process parameters.
    if (xw > 0) {
        cap = model->cj*xl*xw + model->cjw*2.0*(xl+xw)
            cap = distfunc(cap,c_tol,pdist)
            cap = cap*(1 + wtc[1]*(temp-tnom) + wtc[2]*(temp-tnom)**2)
```

```
        }
    else {
        saber_message("TMPL_S_POS",instance(),
            "effective capacitor width")
        }
    }

if (cap < 0) {
    # Capacitance is negative.
    saber_message("TMPL_W_GE_REL_VALUE",instance(),
        "capacitance value","zero")
    }

if(esr >= 0 & esr ~= undef & esr ~= inf) {
    #...just assign the value if >= 0.
    xesr = esr
    }
else if(esr == undef) {
    #...Saber warning for changing value to zero (removes it)
    #...will collapse nodes in control section
    saber_message("TMPL_W_CHANGE_POSEQ",instance(),"esr","undef",0)
    xesr = 0
    }
else if (esr == inf) {
    #...Saber fatal error (open circuit)
    saber_message("TMPL_S_RANGE_NE_INF",instance(),"esr")
    }
else { #...esr < 0
    #...Saber fatal error (nonphysical)
    saber_message("TMPL_S_RANGE_GT_0",instance(),"esr",esr)
    }

if(rleak > 0 & rleak ~= undef & rleak ~= inf) {
    #...just assign the value if > 0.
    xgleak = 1/rleak
    }
else if (rleak == inf) {
    xgleak = 0
    }
else if(rleak == undef) {
    #...Saber warning for changing value to inf (removes it)
    saber_message("TMPL_W_CHANGE_POS",instance(),"rleak","undef","inf")
    xgleak = 0
    }
```

```
else {   #...rleak <= 0
    #...Saber fatal error (nonphysical)
    saber_message("TMPL_S_POS",instance(),"rleak")
    }

(rth_eff,pdmax) = thermpar(rth_ja,undef,undef,ratings->pdmax_ja,undef)

#...Check remaining ratings

xvmax = if(ratings->vmax == undef) then c_vmax else ratings->vmax
if(xvmax ~= undef & xvmax ~= inf) {
    if(xvmax < 0) {
    saber_message("TMPL_W_CHANGE_POS",instance(),"vmax",xvmax,
                            -xvmax)
    xvmax = -xvmax
    }
    else if(xvmax == 0) {
    saber_message("TMPL_W_CHANGE_POS",instance(),"vmax",xvmax,
                            "undef")
    xvmax = undef
    }
    }
xvrmax = if(ratings->vrmax == undef) then c_vrmax else ratings->vrmax
if(xvrmax ~= undef & xvrmax ~= inf) {
    if(xvrmax < 0) {
    saber_message("TMPL_W_CHANGE_POS",instance(),"vrmax",xvrmax,
                                -xvrmax)
    xvrmax = -xvrmax
    }
    else if(xvrmax == 0) {
    saber_message("TMPL_W_CHANGE_POS",instance(),"vrmax",xvrmax,
                                "undef")
    xvrmax = undef
    }
    }

xtjmax = ratings->tjmax
xtjmin = ratings->tjmin
if(xtjmin ~= undef & xtjmin ~= inf) {
    if(xtjmin ~= undef & xtjmax < xtjmin) {
    saber_message("TMPL_S_REL_VALUE",instance(),"tjmax","tjmin")
    xtjmin = undef
    }
    }
```

```
    }
                                                    // Parameters section
```

在本程序中没有Netlist section 和When statement。

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

values {

    #...Definition of output quantities.

    ceff = cap
    v = v(p) - v(m)
    vesr = v(p) - v(pm)
    vcap = v(pm) - v(m)
    q = cap*vcap
    ileak = xgleak*vcap

    if(xesr ~= 0) {
        iesr = vesr/xesr
        }
    else {
        iesr = 0
        }

    pwrd = iesr*iesr*xesr
    tempj = temp + pwrd*rth_eff
    energy = 0.5*cap*vcap*vcap

    }
                                                              // Value section
control_section {
    #...device type and class
    device_type(part_type,part_class)

    #...collapse nodes if possible
    if(xesr == 0) collapse(p,pm)

    # Initial condition for v
    initial_condition(vcap,ic)

    #...Stress measures
    if(include_stress) {
        stress_measure(vmax,voltage,"Max Fwd Voltage",v,max,xvmax)
        stress_measure(vrmax,voltage,"Max Rev Voltage",-v,max,xvrmax)
```

```
        stress_measure(tjmax,temperature,"Max Temperature",tempj,winmax,
               xtjmax, tempj_tnom)
        stress_measure(tjavg,temperature,"Avg Temperature",tempj,average,
               xtjmax, tempj_tnom)
        stress_measure(tjmin,temperature,"Min Temperature",tempj,min,
               xtjmin, tempj_tnom)
        stress_measure(pdmax,power,"Max Power Diss.",pwrd,winmax,pdmax)
        stress_measure(pdavg,power,"Avg Power Diss.",pwrd,average,pdmax)
        }
    }
                                                    // control section
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

equations {
    if(xesr == 0) {
           i(p->m) += d_by_dt(q) + ileak
    }
    else {
       i(p->pm) += iesr
           i(pm->m) += d_by_dt(q) + ileak
    }
    }

}
```

// equation section

下面是一个电源的 MAST 语言的程序

```
#       Ideal voltage source - called by v_sin                          *
#                                                                       *
element template v_sin  p m  =  amplitude,   frequency,   phase,
                offset,      delay,       damping,
                ac_mag,      ac_phase,
                white_noise,flicker_noise                // template header
electrical p,m

number  amplitude,
    frequency,
    phase         = 0,
    offset        = 0,
    delay         = 0,
    damping       = 0,
    ac_mag        = 0,
    ac_phase      = 0,
    white_noise   = 0,
    flicker_noise = 0                               // header declarations
```

```
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
{

var i  i

val v  v
val nv nsv,nsf
val p  power

group {nsv,nsf} noise
    group {v,i,power} hsp

number  work[10]# Work array for argument evaluation.
                                                    // local declaration
srctype..tran    tran=(sin=(va    = amplitude,
                 vo    = offset,
                 f     = frequency,
                 phase = phase,
                 td    = delay,
                 theta = damping))

srctype..trw trw                                    // 调用外模板的参数

foreign trsrc
                                                    // local declaration
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
parameters {

    work = trsrc(1,work,tran)

    }
                                                     // Parameters section
```
在该程序中没有 Netlist section 和 When statement section
```
values {

    #...Determine v

    if (dc_domain | time_domain) {
        # Note that the tran structure needs to be passed for the
        # pwl and ppwl sources. They do not use the work array.
        (v,next_time,step_size) = trsrc(2,time,v,work,tran,trw)
        }
```

```
    else if (freq_mag) {
        v = ac_mag
        }
    else if (freq_phase) {
        v = ac_phase
        }

    #...Determine noise terms
    nsv = white_noise
    if (freq ~= 0.0) {
        nsf = flicker_noise/(freq**0.5)
        }
    else {
        nsf = 0.0
        }

    #...Determine power term for extraction
    power = v*i

    }
                                                    //value section
control_section {
    noise_source(nsv,i)
    noise_source(nsf,i)
    }
                                                    // control section
equations {
    i(p) += i
    i(m) -= i

    i: v(p) - v(m) = v
    }                                               //equation section
}
```

从上面的几个程序都可以看出，在上面的程序中都没有单位和连接点的定义，这是应为在 Saber 程序中有一个标准的 Units.sin 文件，在该文件中有单位和端点的定义。

1、单位和连接点的定义

我们在作仿真时，对各个器件的设置时只需要输入一个数，例如在对电阻设置只输入 1 就是电阻值为 1 欧，输入 1k 就是 1000 欧；在对电压源设置时输入 1000 就表示 1000V；在对频率设置时 50 就表是 50Hz。如果你在设置时加入量纲则系统还将提示你有错误。这是因为在 Saber 文件夹中有一个 Units.sin 文件，在该文件中规定了一些基本量的量纲，Saber 在以 default 方式启动时该文件将被自动加载。在 Saber 中，单位的定义格式如下：

unit {“symbol”,”unit”,”definition”} identifier

对于一些数字量，其单位的定义格式如下：

unit state {MASTname,”Boolen_value”,”printmap”,”plotmap”,

MASTname,”Boolen_value”,”printmap”,”plotmap”,

…….

MASTname,”Boolen_value”,”printmap”,”plotmap”}

例如，电流、电压的定义如下

```
unit {"A","Amperes","Current"} i
unit {"V","Volts","Voltage"} v
```

四值逻辑变量的定义：

```
 unit state {l4_0,  "0","0","low.1",
    l4_1,   "1","1","high.1",
    l4_x,   "x","x","middle.1",
    l4_z,   "x","z","middle.1"} logic_4 = l4_x \
   {CONFLICT_RESOLUTION : foreign l4cnfr}
```

同样，在仿真中也遇到过这样的问题，在图形输入界面方式下，如果将一个电阻接到一个电机的转轴上（在实际系统中不可能有这样的接法，但在仿真图形输入界面中有这种可能），则此时系统将报错。这就是因为元件端点的特征不一致。Saber 中的 Units.sin 中文件也定义了元件端点的特征。在编写模板程序时对于新的端点可能要自己定义，对于单位的定义在编程中用得很少。

如果在编程过程中的确需要自己编写 Units.sin 文件，则在启动 Saber 后需将其设置为 p 方式。

在连接点的定义中就隐含申明了一对变量，它们是 through variable 和 across variable，因此在申明一个端点的同时就隐含的申明了一对变量，同时还要申明这对变量的量纲。

在 MAST 语言中的 through variable 和 cross variable 是与连接点相对应的，所谓 through variable 是指离开该连接点的该量的和为零。如电流 i ，离开任意一节点的电流和为零。Across variable 是指从该点出发，经任一回路回到该点，该量的和为零。如电压 v 就是一 cross variable。经任意一回路的电压和为零。

连接点的定义形式如下：

pin identifier through unit1 across unit2

pin identifier across unit1 through unit2

例如一些连接点的定义如下：

```
pin electrical across v through i
```

下面是常用到的 through variable 和 cross variable 。

	through variable	cross variable
Electrical	i	v
Rotational	torque	angular velocity
Mechanical	force	position
Fluid	flow rate	pressure

//

//

下面是 Saber 中的 Units.sin 文件。

```
#*****************************************************************

# The following units have been defined by Analogy, and are used
# in many of the Analogy supported templates. Changing these unit
```

```
# definitions could affect the operation or displayed results of
# these templates. Please do not change these unit declarations.
#
# List of units already used, listed alphabetically:
#   a,accl_cps2,accl_fps2,accl_ips2,accl_mps2,ang_deg,ang_rad,asi,
#   b,bg,bsi,c,cd,cm,deg,dufs,dw_rpmps,dw_rpmps2,
#   f,fr,frc_d,frc_N,frc_o,frc_p,freq,
#   g,g_kgps,h,hg,hm,hp,hsi,i,j,joule,jsi,kg,
#   l,lb,li,lm,logic_3,logic_4,lps,lx,
#   m,meter,mm,mmf,msi,mu,n,ni,nu,nv,
#   omega,
#   p,p_atm_a,p_bar_a,p_pa_a,p_ppi2_a,pa,pcm3,pm,pos_c,pos_f,pos_i,
#       pos_m,prsr,
#   q,q_s_f3pm,q_s_lps,q_s_m3pm,r,rad,rm,
#   s,s_vol_f3,s_vol_l,s_vol_m3,ssi
#   t,tc,tf,time,tk,tq_dc,tq_fp,tq_io,tq_Nm,tsi,
#   v,v_Nspm2,v_cP,v_m2ps,v_cSt,vel_cps,vel_fps,vel_ips,vel_mps,vps,
#   w,w_radps,w_rpm,wsi

unit {"-","No unit","Unitless number"} nu
unit {"s","Seconds","Time"} time
unit {"Hz","Hertz","Frequency"} freq
unit {"rad/sec","Radians/second","Frequency"} omega
unit {"rad","radians","Angle"} rad
unit {"deg","degrees","Angle"} deg
unit {"m","meters","length"} meter

#*****************************************************************************

unit {"cm","centimeter","length"} cm
unit {"v/s","volt/second","slew_rate"} vps
unit {"1/cm**3","per cubic centimeter","density"} pcm3
unit {"cm**2/sec","square centimeter/sec","diffusivity"} dufs
unit {"cm**2/V/sec","square centimeter/Volt/sec","mobility"} mu

#*****************************************************************************

unit {"V","Volts","Voltage"} v
unit {"A","Amperes","Current"} i
unit {"C","Coulombs","Charge"} q
unit {"F","Farads","Capacitance"} c
unit {"H","Henries","Inductance"} l
unit {"O","Ohms","Resistance"} r
unit {"Mho","Mhos","Conductance"} g
```

```
unit {"S","Siemens","Conductance"} gs
unit {"V/rt(Hz)","Volts/sqrt(Hertz)","Noise Voltage"} nv
unit {"A/rt(Hz)","Amperes/sqrt(Hertz)","Noise Current"} ni


pin electrical across v through i

#*****************************************************************************

unit {"Wb","Webers","Flux"} f
unit {"Wb/sec","Webers/second","Flux rate"} fr
unit {"A-turn","Ampere-turns","Magneto-Motive Force"} mmf
unit {"Mho","Mhos","Magnetic Resistance"} rm
unit {"P","Wb/A-turn","Permeance"} pm
unit {"B","line/in**2","Magnetic Flux Density"} b
unit {"Tesla","Tesla","Magnetic Flux Density"} bsi
unit {"Gauss","Gauss","Magnetic Flux Density"} bg
unit {"A/m","A/m","Magnetic Field Strength"} hm
unit {"Oe","Oersted","Magnetic Field Strength"} hg
unit {"Wb*m/m**3","A/m","Magnetic Dipole Moment Density"} mm


pin magnetic across mmf through f

#*****************************************************************************

unit {"J","Joules","energy"} joule
unit {"W","Watts","Power"} p
unit {"K","Kelvin","Absolute Temperature"} tk
unit {"C","Degrees Celsius","Temperature"} tc
unit {"Fht","Degrees Fahrenheit","Temperature"} tf

pin thermal_k across tk through p
pin thermal_c across tc through p
pin thermal across tc through p

#*****************************************************************************

unit {"rad","radians","rotation angle"} ang_rad
unit {"deg","degrees","rotation angle"} ang_deg
unit {"rad/s","radians per second","angular velocity"} w_radps
unit {"rpm","revolutions per minute","angular velocity"} w_rpm
unit {"rad/s**2","radians per second square","angular acceleration"} dw_radps2
unit {"rpm/s","RPM per second","angular acceleration"} dw_rpmps
```

```
unit {"N.m","newton.meters","torque"} tq_Nm
unit {"dyne.cm","dyne.centimeters","torque"} tq_dc
unit {"ft.lb","foot.pounds","torque"} tq_fp
unit {"in.oz","inch.ounces","torque"} tq_io
unit {"hp","horsepower","hpower"} hp
unit {"oz.in**2.rpm","ounce.inches**2.revolutions/minute","angular momentum"} h
unit {"N.m.rad.s","newton.meters.radians.sec","angular momentum"} hsi
unit {"J","ounce.inches.sec**2","moment of inertia"} j
unit {"J","newton.meters.sec**2","moment of inertia"} jsi

pin rotational_vel across w_radps through tq_Nm
pin rotational_ang across ang_rad through tq_Nm

######## The following rotational unit and pin definitions are obsolete.
######## The above units and pins should be used.
unit {"oz.in","ounce.inches","torque"} t
unit {"N.m","newton.meters","torque"} tsi
unit {"rpm","revolutions per minute","angular velocity"} w
unit {"rad/s","radians per second","angular velocity"} wsi

pin rotational across w through t
########

#*****************************************************************************

unit {"m","meters","position"} pos_m
unit {"cm","centimeters","position"} pos_c
unit {"ft","feet","position"} pos_f
unit {"in","inches","position"} pos_i
unit {"m/s","meters per second","velocity"} vel_mps
unit {"cm/s","centimeters per second","velocity"} vel_cps
unit {"ft/s","feet per second","velocity"} vel_fps
unit {"in/s","inches per second","velocity"} vel_ips
unit {"km/h","kilometer per hour","velocity"} vel_kmh
unit {"mile/h","mile per hour","velocity"} vel_mph
unit {"m/s**2","meters per second square","acceleration"} accl_mps2
unit {"cm/s**2","centimeters per second square","acceleration"} accl_cps2
unit {"ft/s**2","feet per second square","acceleration"} accl_fps2
unit {"in/s**2","inches per second square","acceleration"} accl_ips2
unit {"N","newtons","force"} frc_N
unit {"dyne","dynes","force"} frc_d
unit {"lbf","pounds","force"} frc_p
unit {"ozf","ounces","force"} frc_o
unit {"lb.sec","pound.sec"," linear momentum"} m
```

```
unit {"N.sec","newton.sec","linear momentum"} msi

pin translational_pos across pos_m through frc_N

######## The following translational unit and pin definitions are obsolete.
######## The above units and pins should be used.
unit {"in/s","inches/sec","speed"} s
unit {"m/s","meters/sec","speed"} ssi
unit {"in/s/s","inches/sec**2","linear acceleration"} a
unit {"m/s/s","meters/sec**2","linear acceleration"} asi
unit {"lb","pounds","force"} lb
unit {"N","newtons","force"} n

pin translational across s through lb
########
#*****************************************************************************

unit {"lm","Lumens","luminous flux"} lm      # cd*sr
unit {"lx","Lux","illuminance"} lx       # lm/m**2
unit {"cd","Candelas","luminous intensity"} cd

pin light across lx through lm

#*****************************************************************************

# New for hydraulics library
unit {"p/rt(Hz)","Pressure/sqrt(Hertz)","Noise Pressure"} np
unit {"q/rt(Hz)","flow rate/sqrt(Hertz)","Noise Flow"} nq

unit {"m^3/s","cubic_meters/sec","fluid flow"} q_m3ps
unit {"m^3","cubic_meters","fluid volume"} vol_m3
unit {"m^2","square_meters","surface area"} area_m2
unit {"N/m^2","newton/m**2","pressure"} p_Npm2
unit {"N_s/m^2","newton_s/m**2","dyn.viscosity"} v_Nspm2
unit {"cP","centiPoise","dyn.viscosity"} v_cP
unit {"m^2/s","square_meter/s","kin.viscosity"} v_m2ps
unit {"cSt","centiStoke","kin.viscosity"} v_cSt

unit {"cm^3/s","cubic_centimeters/sec","fluid flow"} q_cm3ps
unit {"cm^3","cubic_centimeters","fluid volume"} vol_cm3
unit {"cm^2","square_centimeters","surface area"} area_cm2
unit {"dynes/cm^2","dynes/cm**2","pressure"} p_dpcm2

unit {"ft^3/s","cubic_feet/sec","fluid flow"} q_f3ps
```

```
unit {"ft^3","cubic_feet","fluid volume"} vol_f3
unit {"ft^2","square_feet","surface area"} area_f2
unit {"ft","feet","length"} foot
unit {"lbs/ft^2","pounds/ft**2","pressure"} p_ppf2

unit {"in^3/s","cubic_inches/sec","fluid flow"} q_i3ps
unit {"i^3","cubic_inches","fluid volume"} vol_i3
unit {"i^2","square_inches","surface area"} area_i2
unit {"in","inch","length"} inch
unit {"oz/in^2","ounces/in**2","pressure"} p_opi2

unit {"psi","pounds/in^2","pressure"} p_ppi2
unit {"atm","atmospheres","pressure"} p_atm
unit {"bar","bars","pressure"} p_bar

unit {"gpm","gallons/min","fluid flow"} q_gpm
unit {"lpm","liters/min","fluid flow"} q_lpm
unit {"cfm","cubic_feet/min","fluid flow"} q_f3pm

unit {"liters","liters","fluid volume"} vol_l
unit {"gal","gallons","fluid volume"} vol_g

unit {"in.lb","inch.pounds","torque"} tq_ip

pin hyd_mks across p_Npm2 through q_m3ps

# Old pressure and flow units
unit {"l/s","liters/sec","fluid flow"} lps
unit {"l","liters","fluid"} li
unit {"psi","pounds/in**2","pressure"} prsr
unit {"Pa","Pascals","pressure"} pa

pin flow across pa through lps

#*****************************************************************************
# units for thermal
#*****************************************************************************
unit {"C/W","degrees/Watt","thermal resistance"} rth
unit {"J/C","Joules/degree","thermal capacitance"} cth
unit {"Sec","second","thermal time constant"} tauth

#*****************************************************************************
# Units for pneumatic components
#*****************************************************************************
```

```
unit {"kg/s","kilogram/s","mass flow"} g_kgps
unit {"st_m^3/min","standard cubic_meter/min","gas flow"} q_s_m3pm
unit {"st_ft^3/min","standard cubic_feet/min","gas flow"} q_s_f3pm
unit {"st_lps","standard liter/s","gas flow"} q_s_lps

unit {"Pa_a","pascal_a","pressure"} p_pa_a
unit {"psi_a","pound/in^2","pressure"} p_ppi2_a
unit {"atm_a","atmosphere","pressure"} p_atm_a
unit {"bar_a","bar","pressure"} p_bar_a

unit {"kg","kilogram","mass"} kg
unit {"st_m^3","standard cubic meter","gas volume"} s_vol_m3
unit {"st_ft^3","standard cubic feet","gas volume"} s_vol_f3
unit {"st_liter","standard liter","gas volume"} s_vol_l

pin pneumatic across p_Npm2 through g_kgps

#*****************************************************************************
# The discrete logic families
#*****************************************************************************
#
# Note: The Analogy supported templates depend on the following
#   declarations. Please do not make any changes to them.

unit state {l4_0,    "0","0","low.1",
        l4_1,    "1","1","high.1",
        l4_x,    "x","x","middle.1",
        l4_z,    "x","z","middle.1"} logic_4 = l4_x \
       {CONFLICT_RESOLUTION : foreign l4cnfr}

unit state {l3_0,    "0","0","low.1",
        l3_1,    "1","1","high.1",
        l3_x,    "x","x","middle.1"} logic_3 = l3_x \
       {CONFLICT_RESOLUTION : foreign l3cnfr}
```

2、模板头

模板头文件定义了模板名、端点名和可以通过网表赋值的变量。如果你调用模板，只需包含该文件的头文件即可。

3、说明部分（declarations）

从上面的结构可以看出，在模板结构中可能有两个部分包含说明，一个是头说明，另一个足部说明，对于 MAST 定义的关键词则不需要作进一步的说明。说明的作用是定义一个“名”相联系的量的类型，以及定义如何使用这个“名”。在说明中可以对一些“名”赋初

值。在使用别人编写的程序时，读懂这一部分很重要，只有读懂了别人模板的这一部分，才有可能正确使用这个模板。

4、参数段 Parameters section

在模板中，参数和变量（arguments）的类型是相似的，它们的说明部分也是相似，但参数是局部的。用 Saber 的 alter 命令可以变量同时也可以改变参数段中的参数。

在这一段中的参数仅在读入文件时计算一次，在用 alter 命令改变参数时 Saber 仿真器将要计算一次参数段中的参数；另外在作 Monte Carlo 分析时每次都要计算一次参数段中的参数。

这一段通常是由一些数学表达式和 MAST 的内部函数构成，但在这一段中不能有微分 d_by_dt(x)，和 delay 函数。

5、网表部分

在模板内的网表主要式在模板中调用其它模板，并且该模板在主模板中有固定的连接关系。其调用形式为

templatename.refdes connection_pt_list=argument_assignment。

Templatename 是调用模板的头文件。

Refdes 是在设计中的器件编号，该值在设计中只能有一个。

Connection_pt_list 它是系统与模板相联的节点，与模板相联的节点数一定要与模板的端点数相等。

6、When statement

When statement 部分主要是用来构造状态机的，即模板的状态与系统的状态有关，或者模板的状态与数字门的值有关。在作一些离散值仿真时常用到 When statement。首先看一个理想二极管的 MAST 语言程序。

```
template ideal_d p n                    // template header
electrical p n                          // header declarations
{
var i id
val v vpn
state r rdiode
state nu a, b
number ron=1u
number roff=100meg                      // local declarations
                                        // 在该程序中没有参数段
when(dc_domain|time_init)
{
schedule_event(time,rdiode,roff)
}
when(threshold(id,0,a,b)
{
if(a==1|0&b==-1)
{
schedule_event(time,rdiode,roff)
schedule_next_time(time)
schedule_next_time(time+1n)
}
```

```
}
when(threshold(vpn,1,a,b)
{
if(a==1|0&b==1)
{
schedule_event(time,rdiode,ron)
schedule_next_time(time)
schedule_next_time(time+1n)
}
}                                                    // When statement section
values
{
vpn=v(p)-v(n)
}                                                      // Value section
equations
{
i(p->n)+=id
id: vpn=id*rdiode
}
}                                                     // equations section
```

When statement 的用法是：

When(condition)

{

statement

}

下面是一个采用保持模板

```
template smplhold in gate out gnd = dt,rt
electrical in,out,gnd
state logical_4 gate
number dt=1n      # delay time in seconds
number rt=1n      # rise and fall time in seconds
{
var i iout            #output current
state v held          #held voltage
state time_next=0     #time when sample is valid
state nu sample       #flag: 1=> sample, 0=> hold
val v vout, vin       #output voltage,input voltage
#detect event on gate
when(event_on(gate))
{
if(gate==l4_1)
schedule_event(time+dt+rt,sample,0)
else if(gate==l4_0)
schedule_event(time+dt,sample,1)
```

```
}
#sample input waveform
when(event_on(sample))
{
schedule_next_time(time)
if(sample==1)
{
next=time+rt
schedule_next_time(next)
}
else held=v(in)-v(gnd)
}
values
{
vin=v(in)-v(gnd)
if(time<next&sample==1)
{
vout=vin-((next-time)/dt)*(vin-held)
else vout=vin
if(sample==0)
vout=held
}
equations
{
i(out->gnd)+=iout
iout: v(out)-v(gnd)=vout
}
}
```

下面是一个数字逆变器的模板

```
template inverter in out                    //template header
state logic_4 in,out                        //header declaration
{
when(event_on(in))
{
if(in==l4_1)
schedule_event(time,out,;4_0)
else if(in==l4_0)
schedule_event(time,out,l4_1)
else schedule_event(time,out,l4_x)
}
}
```

在 When statement 中的条件是一个逻辑表达式，通常是由 Saber 的内部函数 threshold、event_on 和系统变量 dc_done 和 time_step_done 构成，而 statement 通常是由 schedule_event、

schedule_next_time 和 deschedule 函数构成。When statement 与其它段不同，它根据对象的需要可以有许多段 When statement。

与 When statement 相关的几个 MAST 内部函数和系统变量

dc_done 当系统完成直流分析之后，将该变量值为 1。

Time_step_done 在暂态分析中，每一个时间步长完成时将该值设为 1。在 When statement 常用到的系统变量还有 time_init，tr_start，tr_done，dc_init，dc_start。

time_init 在暂态分析开始时设为 1，其它情况下它为零。

tr_start 在暂态分析开始时它为 1，其它情况下它为零。它指的是在任何情况下只要系统一进入暂态分析，该变量就置为 1。而如果系统从暂态分析到暂态分析 time_init 变量不会为 1 而 tr_start 变量将置为 1。

dc_init 与 dc_start 变量的关系与 tr_init 和 tr_start 的关系相似。

以上这些变量都是用来反映系统在仿真过程中的状态，因此如果需要在系统的甚么状态下对系统进行控制时就可以在 When statement。

另外还有一个比较重要的系统变量那就是 time，该变量用来描述仿真器在目前状态下的仿真时间。与之相对应的一个变量是 time_next，这个变量描述的是仿真器将要进入的一个非常短暂的未来。

在 When statement 的条件中经常用到的还有 event_on 函数，threshold 函数，schedule_event 函数，schedule_next_time 函数。

event_on 是一个事件驱动函数，它也经常用到 When statement 段的条件中，当给指定变量赋值时则该函数的返回值为 1，否则该函数的返回值为 0。

event_on (state_var [, old_value])

在表达式中，state_var 是一个被监视量，当它被

schedule_event 函数也经常用到 When statement 的条件中，该函数是一个事件指定函数，该函数的作用是在指定的时刻，通过一个表达式对指定变量赋值。其用法是：

schedule_event(time, state_var, expression)

Threshold 函数，该函数也经常用到 When statement 段的条件，当一个指定表达式“跨越”一个指定的值时，该函数的返回值为 1，其用法为：

threshold(expression,value [,before_state [,after_state]])

在以上的表达式中，expression 是与 value 进行比较，判断 threshold 的条件是否满足。before_state 和 after_state 变量是输出变量，它们可以用来决定系统的状态。

before_state 的值如下：

当 expression 中的值在“跨越”value 值之前大于 value 的值时，该变量的返回值为 1；

当 expression 中的值在“跨越”value 值之前小于 value 的值时，该变量的返回值为－1；

当 expression 中的值在“跨越”value 值之前等于 value 的值时，该变量的返回值为 0。

after_state 的值如下：

当 expression 中的值在“跨越”value 值之后大于 value 的返回值时，该变量的值为 1；

当 expression 中的值在“跨越”value 值之后小于 value 的返回值时，该变量的值为－1；

当 expression 中的值在“跨越”value 值之后等于 value 的返回值时，该变量的值为 0。

schedule_next_time 函数确定算法的采样点，它是由仿真器决定系统的仿真步长，其用法是：

[scheduling_id=] schedule_next_time(time)。

为了加深对这一部分函数的理解，下面在编写几个程序。

下面是一个时钟模板。

```
template clock out = hightime,lowtime            //header template
```

```
#this template models a square wave
state logic_4 out
number hightime   #time when signal is high
number loetime    #time when signal is low     //header declaration
{
state nu wake_up                                  //local declaration
when(dc_init)
{
schedule_event(time,out,l4_1)
}
when(time_init)
{
schedule_event(time,wake_up,0)
}
when(event_on(wake_up))
{
schedule_event(time+hightime,out,l4_0)
schedule_event(time+hightime+lowtime,out,l4_1)
schedule_event(time+hightime+lowtime,wake_up,0)      // when statement
}
}
```

这一部分主要是有条件句，数学表达式和 MAST 内部函数构成，但这一部分不能有 d_by_dt(x)函数和 delay 以及 random 函数。它可以调用有 Fortran 或 C 语言写的函数，将 C 或 Fortran 写的程序的运算结果的返回值作为判据。

7、Values 段

模板 Values 段是用来定义在处理传递过程中将要提取的数据，它也要将变量转换为在方程段中所需要的变量形式。

8、控制部分

这一部分不是必须的，它要求处理器在处理该模板时如遇到特殊情况的处理方法。

a) 在条件控制中可能使两个节点合并为一个节点（collapse two nodes into a single node），如遇到这种情况可以加大仿真步长，否则，一旦节点 collapse，只有退出重新仿真。
b) 为了折线化非线性曲线，定义一组值。
c) 定义采样点
d) 确定自变量的牛顿步长。
e) 小信号噪声源的定义。

在仿真中，我们经常遇到一些问题，如报的一些错误并不是由于电路结构的错误，对于修改一些元件的参数后仿真器就能通过仿真，引起这些问题的主要原因就是一些元件中没有控制段段，如果在这些元件中增加一些控制，则系统的仿真就可以顺利进行。下面就控制的一些主要问题作一个简单的介绍。

9、方程段

方程段是描述所定义模板元件的端点特征，实际上模板的这一部分定义了元件在系统中的作用。

方程段中的方程表明了指定变量与系统中其它变量的关系，这一部分主要是数学表达式、MAST 内部函数，在这一段中，可以使用 delay 函数和 d_by_dt(x)函数，但 random 函数不能使用。

在方程段中，经常使用两个符号＋＝和－＝；符号＋＝是 is added to；而－＝是 is subtracted from。

在编写元件模板时，方程段中通常定义的 through 变量，系统的 ref 变量，以及与 var 变量相对应的描述系统特征的方程数。

从上面可以看出，一个模板可能比较复杂，但比较主要的部分是 declaration 部分，Values 段和方程段，这几部分是编写模板最重要的几部分。

上面讲述了在调用模板的方法，如果需要在模板中调用文件则是用符号<，该符号要放在第一列。

< includefilename　　< consts.sin

三、典型模板的建立

在前面已经建立了一些模板，我们已经知道在建立模板时有几个需要注意的问题。首先应该明确知道模板的结构。在 MAST 建模的各段虽然没有固定的顺序，但在编写模板时应该按照一定的顺序编写，这样使整个程序更有可读性；同时还应注意编写模板的语法。下面将从另一角度来看一看模板的写法。

1）非线性模板

在前面讲述了一个二极管模型，在该模型中，采用了一个状态变量 r 来描述二极管的状态，通过对状态变量 r 的赋值来描述二极管的导通和关断。这个二极管模型的建立比较简单，但该模型比较粗着，它没有反映二极管的开通过程，在有些系统级的仿真中调用该模型将导致仿真停止，下面将介绍一个相对好一点的模型。

我们知道，二极管是一个非线性模型，二极管在反向电压下是处于关断状态，在正向电压下二极管是导通状态，在导通状态下其电压电流的关系是：

```
1   element template diode p m = is, ic # template header
2     electrical p, m                   # header declarations
3     number is = 1e-16,
4            ic = undef
5     external number temp
6   {                                   # start of template body
7     number k = 1.318e-23,             # local declarations
8            qe = 1.602e-19,
9            vt
10    val v vd
11    val i id
12    struc {
13      number bp, inc;         # Newton steps
14    }  nvd[*] = [(0,.001),(2,0)]
15    parameters {                  # start of parameters section
16      vt = k * (temp+273.15) / qe # compute thermal voltage
17    }                            # end of parameters section
18    values {                      # start of values section
19      vd = v(p) - v(m)            # diode voltage
```

```
20      id = is * (limexp(vd/vt)-1) # diode current
21    }                             # end of values section
22    control_section {              # start of control section
23      newton_step (vd,nvd)         # Newton steps assigned to vd
24      initial_condition(vd,ic)
25      start_value(vd,0.6)
26      device_type("diode","example")
27      small_signal(vd,voltage,"p-m voltage", vd)
28    }                             # end of control section
29    equations {                    # start of equations section
30      i(p->m) += id                # current contribut. of diode
31    }                             # end of equations section
32  }                               # end of template body
```

在这个例子中是采用控制牛顿步长来解非线性方程的。

模 板

用 MAST 语言来描述系统连接式元件特性的单位是模板，模板是 MAST 语言的核心，使用 MASY 语言的目的就是要建立反映系统连接或元件特性的模板。

一个通用模板可能包括 11 个部分，它们是单位定义、指针定义、头文件、全局申明、局部申明、参数段、网表、when 段、vulues 段、控制段、方程段。

一个模板的各个部分主要有二个部分，申明部分主要是指明仿真器可用的变量和引用的一些变量。

一、模板头的定义，它是为了给一个模板去一个名，并且指定模板类型，该模板的连接点以及使用该模板需要的参数。它决定了该模板的使用方法。

模板的类型有两种，一个是标准模板，另一种是元件模板。元件模板应在模板前加 element 关键词。这两种模板的区别一种是模板内部的节点可见，另一种是模板内的节点不可见。模板头的定义规则维：

[element] template template-name connections [argument]

定义模板的关键词是 template，如果是元件模板，则其关键词为 element template。

template-name 是模板名，它应与文件名一致。

例子：element template vesistor p m [= r,tnom]。

二、全局申明：在头文件种，只有 template-name 项是可知的，而连接点和 argument 没有说明。

Pin-type	(implied unit)	id,id
State	unite	id,id
Ref	unite	id,id
Var	unite	id,id
Argument	三类基本数	三类复合数

在全局申明中就是用来说明这一部分的，它主要是说明连接点上的类型和 argument 的类型。（互感元件 k 用了 ref 端点）。

三、局部申明

局部申明是用来说明在函数体内需要用到的指针，连接点，argument 等的说明。这一部分的使用规则与全局申明的规则一样。

四、参数段

参数段是一个操作段，它是用来处理参数。并不是在每个步长中都要处理该段，只是在仿真器读输入文件时，以及改变 argument 只时或进行 monte carlo 分析处理该段。对参数段的执行是从上到下逐条执行，这与 C 或 fortran 程序相似。参数段的关键词 parameters，其定义规则为：

```
parameters{
    statements
}
```

在参数段中的 statements 可以是赋值语句，可以是条件语句，也可以调用 C 或 fortran 函数，利用其函数计算的返回值，它可以用 MAST 语言的库函数，但它布能用 d_dy_dt 函数和 delay 函数，在该段中唯一可用的仿真变量 simvar 就是 statistical

例子

```
parameters{
    if (model→type==-n{
[
```

```
            p=1
}
else (model→type==-p{
[
            p=-1
}
rb=area*model→rb
work=spq(1,model,rb,temperature)
}
```

五、在前面已经看到，用 MAST 语言解一个反映系统连接的模板其实就是一个网表，在解一个元件模型时，有时也要用到网表。网表的目的就是在一个元件中调用已有的模板，并对调用模板的 argument 量赋值。元件模板中的网表与系统网表的用法相同，其调用语法规则:

templatename. refdes connection-pt-list[=argument]

templatename 是调用的模板名，它是由模板头所确定的，refdes 是设计的参考，它在设计中必须是唯一的，connection pt-list 模板的连接方式。

e.g.

```
element template v p m =dc ,tran ,ac
electrical( p ,m
number dc=0
union{
 number off
 struc{number v0=0, va , f, ,td, theta}sin
 struc{number v1,v2, ,td, tr, tf, pw, per}pulse
 struc{number v1,v2, ,td1, tau1, td2,tau2}exp
 number pwl[*]
 struc{
      number v1,v2,period,rtime,width,ftime,delay
      }clock
}tran=( off=1)
struc{number mag=0,phase=0}ac=(0,0)
```

```
v.v1 p:a m:0=dc=5
v.v2 b     0=5
v.v3 c     0=vcc
v.v4 d     0=tran=(sin=(va=1,f=10k,td=0.theta=0))
v.v5 e     0=tran=(sin=(0,1,10k,0,0))
v.v6 f     0=tran=(pwl=[0,0,10n,0,11n,5,20n,5,21n,0])
v.v7g     0=tran=(sin=(0,1,10k,0,0)),ac=(mag=1,phase=0)
```

六、when 段

when 段是一个操作段，该段的操作总是和时间联系在一起的，即在什么时候操作。它可以离散模拟信号，操作数字信号，测试模拟波形穿越门槛，确定触发事件，确定仿真器的时间等。

when 段语法定义为：

```
when(condition){
      statements
```

}

在 when 段时，关键词是 when，而条件将被监视，一旦条件得到满足，函数体中的语句将逐条被执行，在 when 段中的条件通常由事件触发函数 event-next-time 和门槛比较函数 threshold 充当，同时仿真器变量也经常作为 when 中的条件来使用，而在 statements 中紧张用到的函数 schedule-event,schedyle-next-time。

在 statements 中可以使用赋值语句，被赋值量一定要是状态量。

Values 段

values 段既是声明段又是操作段，在这一段中主要是对 val 类型的变量赋值，它主要是为方程段服务的。

values 段并不是在每一个时间步长上都要进行计算，只是在系统需要时候才进行计算。当只有在该段设有与时间相关的量时在每个时间步长上才至少计算一次，但斐然当然如果有非线性部分，如果有必要的话将进行叠代。在该段中被赋值的变量一定要是 val 型（仿真器变量中 next-time 和 step_size 也能在该段作为被赋值的变量。

values 段的声明语法：

```
values{
    statements
}
```

其中关键词是 values。在 statements 中可以是赋值语句、条件语句，也可以是有 MAST 语言的内部函数（除 d_dy_dt,delay 函数外）。

要注意在 val 段中赋值的变量。

在 values 段中，可以调用用 C 或 Fortran 所写的函数。如下例

```
electrical p,n
struc{
number is,cs,rs
} model=()
number area=1
{
electrical pi
val i idi,id
val q qd
val v vdi,vres
number work[17]
external number temp
foreign diodesub
values{
vdi=v(pi)-v(n)
vres=v(p)-v(pi)
id=vres/model->rs
(idi,qd)=diodesub(work,model,temp,area,vdi)
}
equations{
I(p)+=id
I(pi)+=idi+d_by_dt(qd)-id
I(n)-=idi+d_by_dt(qd)
```

}

在作仿真时，有时需要将有些节点或变量上加上一些扰动，当然可以用其它一些办法来实现这样的功能，在该节点或变量上加上噪声是最简单的方法，关于噪声源的加法在控制段中实现，但是对噪声源变量将是 val 变量，对于这类变量将在 value 段中赋值。

```
Val nv nsv, nsvf
Val ni nsim,nsip,nsipf,nsimf
Var I I
Values{
Nsv=20n
Nsip=0.71
Nsim=.71p
If(freq~=0){
Nsvf=200n/freq
Nsipf=70p/freq
Nsimf=70p/freq
}
else
Nsvf=0
Nsipf=0
Nsimf=0
}
control_section{
noise_source(nsv,I)
noise_source(nsvf,I)
noise_source(nsim,I)
noise_source(nsimf,I)
noise_source(nsip,I)
noise_source(nsipf,I)
}
```

控制段

控制段是一个申明段而不是一个操着段，对模板而言，这一部分并不是必须的，它提供给仿真器的信息不同于其它段，它提供给的是仿真器的分析方式。

控制段的申明如下：

```
control_section{
statements
}
```

其中 control_section 是该段的关键词，而控制段中 statements 只能是下面几种方式中的一种。

```
Collapse(node1,node2)
Noise_source(val,pin_or_var[,pin_or_var])
Sample_points(variable,sapoints)
Sample_points((variable,variable),sapoints)
Pl_set((dep_id[,dep_id]),(indep_id[,indep_id]))
```

Newton_step((variable,variable),nstep)

在一些条件判据中，可能要将两个节点合并，这是将改变系统电路的结构，如果是这样的话，将改变系统的网表，这时应该加快仿真器的仿真速度跳过对这两点间的计算，否则仿真器将会退出运行。这是就应该在控制段中加入 collapse 函数,该函数的用法为 collapse(node1,node2)。

在控制段中，可以对非线性方程进行线性化处理。对于非线性模型而言，如对其进行线性化处理，通常由三种方法。

第一种方法就是在建立非线性系统的模型时，用折线段来代替非线性。如果采用这种方法来线性化，则不需要仿真器作任何其它的工作，这是系统的仿真精度取决于所建模型的精度，如果发现仿真精度不够时，只有改变其近似模型。这种方法使用时不太方便，在 saber 中不用这种方法。

第二种方法建立模型时还是用系统自身的非线性模型,然后给出猜测点及在这一点的增加量，仿真器根据给出的猜测点计算出在该点的斜率，根据计算的斜率和给出增加量来计算下一点，然后反复计算，得出一条折线，它就是用这条折线来代替系统的曲线。

如果要对非线性模型采用这类方法来进行线性化处理，就可以在控制段中采用 sample_points 命令。

sample_points 的用法为 sample_points（variable [,variable],sampoint）

其中 variable 是某个需要线性化的变量（或某些变量），而 sampoint 是一对数据，分别表示变量的猜测点和增加量。

```
Struc{
Number breakpoint,increment
}sp[*]=[(-100, 10), (-10, 1), (0, 0.2), (10,0)]
val v vd
control_section{
sample_points(vd, sp)
}
```

在 saber 中，有一个用 Fortran 写成的采样函数，logsap，在编写程序时可以调用该函数来实现采样。该函数的用法为

[*]=logsap(a, b, c, d)

该函数的返回值为一对数组成的数组，返回数组的长度与该函数的调用参数有关。该函数的调用的意义。

1、logsap(1u,1meg,1,x)

-1e6, -1e5, ……,-1e-6, 0, 1e-6, ……, 1e5, 1e6

2、logsap(1u,1meg,3,x)

-1e6, -4.641e5, -2.154e5, -1e5, -4.641e4, -2.154e4, -1e3, ……, -1e-6, 0, 1e-6, -4.641e-5, -2.154e-5, , ……, 1e6

3、logsap(1u,1meg,0.5,x)

-1e6, -1e4, -1e2, -1 , ……, -1e-6, 0, 1e-6, , ……, 1e6

4、logsap(1u,1meg,1,90)

(-1e6, 1e4), (-1e5, 1e3), , ……, (-1e-6, 1e-8), (0, 1e-8), (-1e-6, 1e-7), ……, (1e6, 0)

对于线性化的另一种办法就是在非线性曲线上描述一些点,将这些点连接起来就构成一条折线，当然只需要给出自变量。该种方法在建模时仍然采用的是非线性模型。在 saber 中如果用这类方法来进行线性化处理，就可以用 pl_set 命令来实现。

Pl_set(id, vd)

在上述的关系是中，vd 是自变量，id 是因变量。在用上述关系时，首先要确定自变量 vd 的采样点，然后就可以调用上面的函数。

Pl_set((idi,qd),vdi)

用上述函数的条件是 idi 和 qd 都是 vdi 的函数。

三、对牛顿步长的控制

在仿真中，有时需要对某些变量的牛顿步长进行控制，特别是函数斜率变化较快的时候。牛顿步长的控制如下

Newton step(variable, nstep)

newton_step((variable,variable),nstep)

e.g

```
struc{
number breakpoint,increment
} nv=[(0.2, 20m) , (0.6, 1m) ,(1, 0)]
val v vdi
control_section{
newton_step(vdi,nv)
}
```

方程段

方程段是用来描述元件端点特性，以及对系统变量的定义。方程段的定义

```
equations{
statements
}
```

在 statements 中包含两种方程，一种是对系统变量的定义，另一种是描述元件端点的特性。如果要正确的描述元件端点的特性，则描述元件特性的方程数应与系统变量的个数相等。

在定义系统变量时，经常用到的操作符为＋＝或－＝。＋＝的含义是（to add to）－＝含义是(to subtract from)。

在描述元件特性的方程可以用 MAST 语言的任意函数，但用 d_by_dt 时前面只能用＋或—。

```
Electrical p, m
Var id
Number resistance
Equations{
I(p->m)+=id
(v(p)-v(m))/resistance
I(p)+=id
(v(p)-v(m))/resistance
I(m)-=id
(v(p)-v(m))/resistance
```