# Contents

# Chapter 1

# Simulation Using MSPICE

We use *MSPICE* to simulate any circuit before formal verification. MSPICE is a simple approximation of SPICE. It uses ODE to model the circuit and solves the ODE by ODE solver in matlab.

## 1.1 RK method

MSPICE uses ODE45 for integration. ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{n-1})$. In general, ode45 is the best function to apply as a "first try" for most problems.

RK is a family of methods to solver ODE. Let an *initial value problem* be specified as follows.

$$y' = f(t, y), \quad y(t_0) = y_0.$$

Then, the RK4 method for this problem is given by the following equations:

$$\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\
k_3 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \\
k_4 &= f(t_n + h, y_n + hk_3) \\
y_{n+1} &= y_n + \frac{1}{6}h\,(k_1 + 2k_2 + 2k_3 + k_4) \\
t_{n+1} &= t_n + h
\end{aligned}$$

where $y_{n+1}$ is the RK4 approximation of $y(t_{n+1})$, and

The RK4 method is a fourth-order method, meaning that the error per step is on the order of $h^5$, while the total accumulated error has order $h^4$.

The general explicit RK method is

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i \\ k_i &= f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j\right). \end{aligned}$$

where

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \dots & b_{s-1} & b_s \end{array} = \begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b^T} \end{array}$$

The RungeKutta method is consistent if

$$\sum_{j=1}^{i-1} a_{ij} = c_i \text{ for } i = 2, \dots, s$$

The general implicit RK method is

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i \\ k_i &= f\left(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} k_j\right). \end{aligned}$$

where

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} = \begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b^T} \end{array}$$

ode23 is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness. Like ode45, ode23 is a one-step solver.

ode113 is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than ode45 at stringent tolerances and when the ODE file function is particularly expensive to evaluate. ode113 is a multistep solver - it normally needs the solutions at several preceding time points to compute the current solution.

Table 1.1: The comparison of different ODE solvers in Matlab

| Solver | Problem | Accuracy | When to Use |
|---|---|---|---|
| ode45 | Nonstiff | Medium | Most of the time. This should be the first solver you try. |
| ode23 | Nonstiff | Low | If using crude error tolerances or solving moderately stiff problems. |
| ode113 | Nonstiff | Low to high | If using stringent error tolerances or solving a computationally intensive ODE file. |
| ode15s | Stiff | Low to medium | If ode45 is slow (stiff systems) or there is a mass matrix. |
| ode23s | Stiff | Low | If using crude error tolerances to solve stiff systems or there is a constant mass matrix. |

The above algorithms are intended to solve non-stiff systems. If they appear to be unduly slow, try using one of the stiff solvers (ode15s and ode23s) instead.

ode15s is a variable order solver based on the numerical differentiation formulas, NDFs. Optionally, it uses the backward differentiation formulas, BDFs (also known as Gear's method) that are usually less efficient. Like ode113, ode15s is a multistep solver. If you suspect that a problem is stiff or if ode45 has failed or was very inefficient, try ode15s.

ode23s is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than ode15s at crude tolerances. It can solve some kinds of stiff problems for which ode15s is not effective.

The comparison of solvers is listed on table 1.1.

## 1.2 Random Brockett Input

To represent a family of signals, we use the brockett annulus to specify the input signals. To simulate the input, we want to generate a random signal whose derivative satisfies the specification.

Our idea is to pickup a random trace within the annulus. The first step is to selects a set of points randomly within the outer and inner ellipses, which is trivial. The next step is to compute a interpolation based on these points. Figure 1.1 shows the idea.

There are several different interpolation method. The matlab *interp*1 provides methods including *nearest, linear, spline, pchip/cubic, v5cubic.* The *pp-form* (piecewise polynomial form) can be computed as

```
PP = INTERP1(X,Y,METHOD,'pp')
```

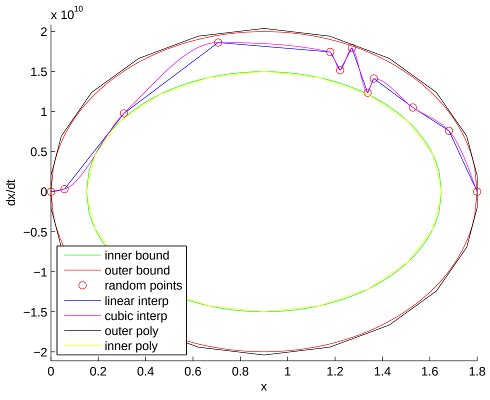

Figure 1.1: The random input trace

The result $pp$ is a structure with $breaks, coefs, pieces, order, dim$ fileds, which can be evaluated by *ppval* function or extracted by *unmkpp* function as

```
V = PPVAL(PP,XX)
[br,co,l,k,d] = unmkpp(pp)
```

Given $pp$, the piecewise polynomial[1] is

$$f(x) = \begin{cases} co(1,:)\cdot\begin{bmatrix}(x-br(1))^k\\ \vdots\\ (x-br(1))\\ 1\end{bmatrix} & : \quad x < br(2)\\ \vdots & \vdots\\ co(i,:)\cdot\begin{bmatrix}(x-br(i))^k\\ \vdots\\ (x-br(i))\\ 1\end{bmatrix} & : \quad br(i)\le x < br(i+1)\\ \vdots & \vdots\\ co(l,:)\cdot\begin{bmatrix}(x-br(l))^k\\ \vdots\\ (x-br(l))\\ 1\end{bmatrix} & : \quad br(l)\le x \end{cases}$$

The simplest case is the linear interpolation. If the two known points are given by the coordinates $(x_0, y_0)$ and $(x_1, y_1)$, the linear interpolant is the straight line between these points. For a value x in the interval $(x_0, x_1)$, the

[1]Similar for all polynomial interpolation methos?

value y along the straight line is given from the equation

$$\begin{aligned}\frac{y-y_0}{y_1-y_0} &= \frac{x-x_0}{x_1-x_0}\\ y &= y_0+(x-x_0)\frac{y_1-y_0}{x_1-x_0}\end{aligned}$$

*Cubic interpolation*[2] or *cubic Hermite spline* is a third-degree spline with each polynomial of the spline in hermite form.

On the unit interval (0,1), given a starting point p0 at t = 0 and an ending point p1 at t = 1 with starting tangent m0 at t = 0 and ending tangent m1 at t = 1, the polynomial can be defined by

$$\begin{aligned}\mathbf{p}(t) &= (2t^3-3t^2+1)\mathbf{p}_0+(t^3-2t^2+t)\mathbf{m}_0+(-2t^3+3t^2)\mathbf{p}_1+(t^3-t^2)\mathbf{m}_1\\ &= h_{00}(t)\mathbf{p}_0+h_{10}(t)\mathbf{m}_0+h_{01}(t)\mathbf{p}_1+h_{11}(t)\mathbf{m}_1.\end{aligned}$$

where $t\in[0,1]$.

Interpolating $x$ in the interval $(x_k, x_{k+1})$ can now be done with the formula

$$\mathbf{p}(x)=h_{00}(t)\mathbf{p}_0+h_{10}(t)h\mathbf{m}_0+h_{01}(t)\mathbf{p}_1+h_{11}(t)h\mathbf{m}_1.$$

with $h=x_{k+1}-x_k$ and $t=\frac{x-x_k}{h}$. Note that the tangent values have been scaled by h compared to the equation on the unit interval.

Of course, we can use the random trace within the annulus as the differential function and use ODE45 to solve it. However, it has the problem when the derivative is zero which causes the input stay stable for infinite time! It also requires to change the mspice code, which is not a big problems of course.

Thus, given the trace, we need to solve the ODE function to get the input function $x=f(t)$.

## 1.3 Solving the differential function

Given a trace $\dot{x}=\frac{dx}{dt}=f(x)$ within the brockett annulus, what is the transistion time(trace) from $x_0$ to $x_1$?

The ODE for linear trace is easy to solve as

$$\begin{aligned}\dot{x} &= ax+b\\ dt &= \frac{dx}{ax+b}\\ \int dt &= \int\frac{dx}{ax+b}dx\\ t &= \frac{1}{a}ln|ax+b|+c\\ x &= \frac{1}{a}(ce^{at}-b)\end{aligned}$$

[2]It is provided by *pchip* function in matlab. Cubic hermite interpolation is not related to hermite polynomials

Given the condition that $x = x_0$ when $t = 0$, we have

$$\begin{aligned} c &= ax_0 + b \\ x &= (x_0 + \frac{b}{a})e^{at} - \frac{b}{a} \end{aligned} \tag{1.1}$$

The time to arrive $x_1$ is

$$t(x_1) = \frac{1}{a} \ln \left( \frac{ax_1 + b}{ax_0 + b} \right) \tag{1.2}$$

Remarkly, $ax + b$ should have the same sign (can not be zero) for $\forall x \in [x_0, x_1]$ (of course, $\frac{ax_1+b}{ax_0+b} \geq 1$) and $a$ can not be zero. When $a = 0$, it degrades to

$$\begin{aligned} x &= bt + x_0 \\ t(x_1) &= \frac{x_1 - x_0}{b} \end{aligned}$$

For the ellipse trace,

$$\frac{(x - x_0)^2}{a^2} + \frac{\dot{x}}{b^2} = 0$$

the differential equation can be solved by variable substitution. As we know for differential equation

$$F(y, \frac{dy}{dx}) = 0$$

if there is a substitution

$$\begin{aligned} y &= \phi(x) \\ \frac{dy}{dx} &= \sigma(x) \end{aligned}$$

which satisfies the differential equation, then the general solution is

$$\begin{aligned} y &= \phi(x) \\ x &= \int \frac{\phi'(t)}{\sigma(t)} dt + c \end{aligned}$$

We use the substitution

$$\begin{aligned} x &= a\cos(\nu) + x_0 \\ \frac{dx}{dt} &= b\sin(\nu) \end{aligned}$$

and get the solution

$$\begin{aligned} x &= a\cos(\nu) + x_0 \\ t &= -\tfrac{a}{b}\nu + c \end{aligned} \quad \Rightarrow \quad \begin{aligned} \nu &= \tfrac{b}{a}(c - t) \\ x &= a\cos(\tfrac{b}{a}(c - t)) + x_0 \end{aligned}$$

Given the initial condition $x = (x_0 - a)$ when $t = 0$, we have

$$\begin{aligned} c &= \frac{a}{b}\pi \\ x &= a\cos(\pi - \frac{b}{a}t) + x_0 \end{aligned} \tag{1.3}$$

Therefore, the arriving time for $x = x_0 + a \cdot \delta p, \delta p \in [-1, 1]$ is

$$t(\delta p) = (\pi - \arccos(\delta p))\frac{a}{b}$$

The transition time from $x_0 - a$ to $x_0 + a$ is

$$t(1) = \frac{a}{b}\pi \tag{1.4}$$

Thus, for a brockett annulus defined by $ll, lh, hl, hh$ and $r_{max}, r_{min}$, the transition time from $lh$ to $hl$ is in the interval

$$\left[\frac{hh - ll}{2r_{max}}\left(\pi - 2\arccos\left(\frac{hl - lh}{hh - ll}\right)\right), \frac{hl - lh}{2r_{min}}\pi\right] \tag{1.5}$$

The differential function for cubic interpoloation is more complicated. However, the transition time must be contained by this interval of ellipse trace.

### 1.3.1 Connecting of different period

Therefore, we use linear interpolation of random points within the annulus. However, we have the problem that the derivative can not be zeros. Thus, for each peroid, we use positive derivative for rising phase and negative for falling phase. Then the problem, how to connect these two traces? Our solution is to use linear trace from $x_1^{fall}$ to $x_0^{rise}$ as shown in Figure 1.2[3].

The code is

```
function brockett = brockett_generate(brockett,t,n)
ir = brockett.radius(1); or = brockett.radius(2);
ll = brockett.vbnds(1); lh = brockett.vbnds(2);
hl = brockett.vbnds(3); hh = brockett.vbnds(4);
minT = brockett.minT;

% the minimum transition time from lo/hi to hi/lo is
minTransT = (hh-ll)/(2*or)*(pi-2*acos((hl-lh)/(hh-ll)));
hpt = (minT+minTransT);
np = ceil(t/hpt);

bnds = linspace(ll,hh,n+1)';
xs = repmat(bnds(1:n),1,np+1) + repmat(diff(bnds),1,np+1).*rand(n,np+1);
oys = sqrt( 1 - ((xs-(ll+hh)/2)/((hh-ll)/2)).^2 ) * or;
iys = sqrt(max( 0, 1 - ((xs-(lh+hl)/2)/((hl-lh)/2)).^2 )) * ir;
ys = iys + (oys-iys).*rand(n,np+1);
xs(:,2:2:end) =  xs(end:-1:1,2:2:end); % fall edge;
ys(:,2:2:end) = -ys(end:-1:1,2:2:end);

as  = zeros(n,np); bs  = zeros(n,np);
```

[3]We have the problem that the signal may never goes to 0 or 1.8 now

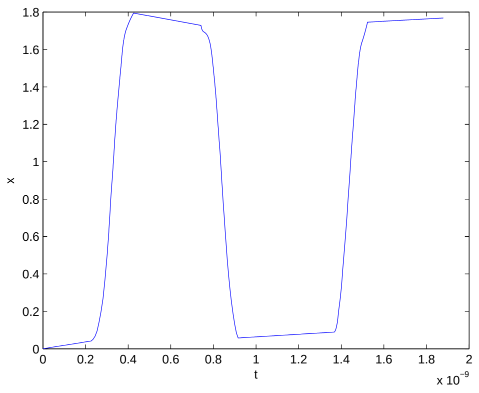

Figure 1.2: The random input from brockett annulus

```
x0s = zeros(n,np); ts  = zeros(n,np);
for i=1:np
    % rise/fall phase
    x = xs(:,i); y = ys(:,i);
    % linear interpolation
    x0 = x(1:end-1); x1 = x(2:end);
    y0 = y(1:end-1); y1 = y(2:end);
    a = (y1-y0)./(x1-x0);
    b = y0-a.*x0;
    t = (log(x1+b./a)-log(x0+b./a))./a;

    % stable phase
    st = (1+rand)*minT; % stable
    slop = (xs(1,i+1)-x(end))/st;
    a = [a; 0]; b = [b; slop ];
    x0 = [x0; x(end)]; t = [t;st];

    as(:,i) = a; bs(:,i) = b; x0s(:,i) = x0; ts(:,i) = t;
end;

t0 = rand*minT;
slope = (x0s(1)-ll)/t0;
trace.as  = [0;reshape(as,[],1)];
trace.bs  = [slope;reshape(bs,[],1)];
trace.x0s = [ll;reshape(x0s,[],1)];
trace.ts  = cumsum([0;t0; reshape(ts,[],1)]);
brockett.trace = trace;

function v = f(brockett,t)
trace = brockett.trace;
ts = trace.ts; x0s = trace.x0s; as = trace.as; bs = trace.bs;

% For the rise/fall phase, the function is
%   x(t) = (x0s(i)+bs(i)/as(i))*exp(as(i)*(t-ts(i)) - bs(i)/as(i) for t IN [ts(i),ts(i+1)]
% For the stable phase, the function is
%   x(t) = x0s(i)+bs(i)*t for t IN [ts(i),ts(i+1)]
n = length(t);
v = zeros(n,1);
for i=1:n % vectorize later
    ind = find(ts<=t(i),1,'last'); % find the interval
    dt = t(i)-ts(ind); % relative time
    a = as(ind); b = bs(ind); x0 = x0s(ind);
    if(a~=0)
        v(i) = (x0+b/a)*exp(a*dt) - b/a; % rise/fall;
    else
        v(i) = x0+b*dt; % stable region
    end;
end;
```

### 1.3.2 Other issues

The input specification of latch circuit requires that the data input can not change when clock is falling. Therefore, we have the restriction that *clock in fall phase* $\Rightarrow$ *data in low or high phase*.

Our solution is to genearate the input for clock first as described above. Then we compute all time intervals when the input can not change. Then we genearate the input with this constraints.

## 1.4 Measure the capacitance

The capacitance used in COHO is different from the real circuit. Thus, we want to adjust it close to real circuit. We use ring oscillator shown in figure 1.3 as an example. When $\overline{reset}$ is low, the circuit is stable, and when $\overline{reset}$ is high, the circuit starts to osccillate.

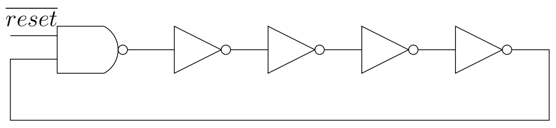

Figure 1.3: The Ring Oscillator Circuit

I simulate it first by hspice and measure it period as $3.4e{-}10$. And then simulate it by mspice and its period is $3.8e{-}10$. Thus, mspice is about 11.5% slower than hspice. Thus, the default value $2e{-}9$ used in mspice is close enough, we do not need to change it.

## 1.5 Support macro models

Currently, MSPICE only support nmos or pmos. Thus, when I verify the arbiter circuit with a four terminal macro model, codes are messed up. It is time to let mspice supports more macro models. I think this will increase performance and may reduce error.

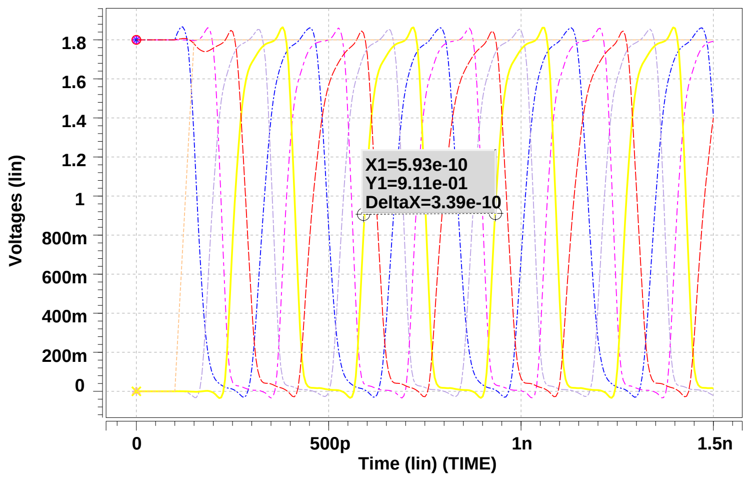

Figure 1.4: The HSPICE simulation result

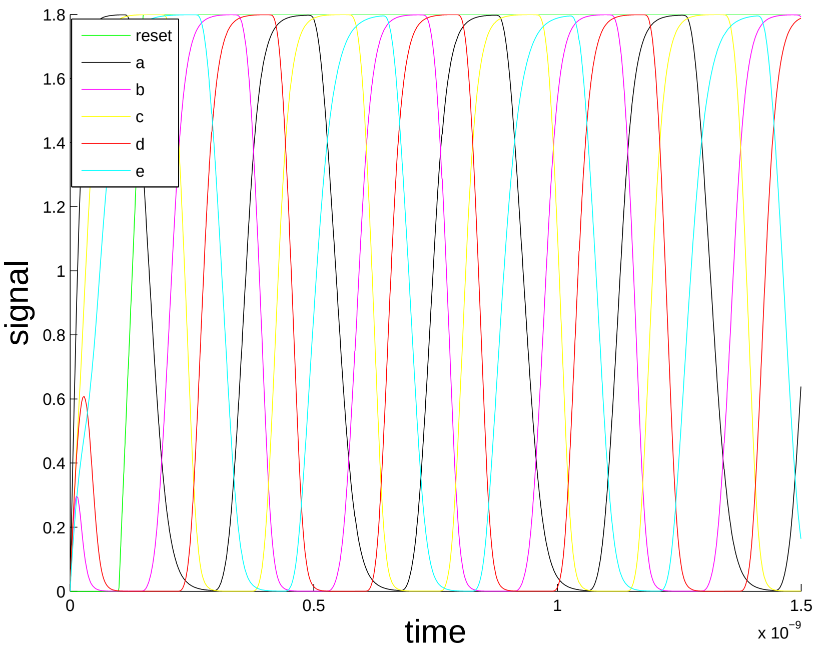

Figure 1.5: The MSPICE simulation result