SEGUNDA PARTE

3.8. Solución completa con MATLAB del cálculo de Vx.

En la primera parte del capítulo se ha mostrado como el desarrollo de ecuaciones para el cálculo de Vx supone un método que ofrece buenos resultados, pero que plantea el riesgo de ser inestable en determinadas situaciones.

En esta segunda parte del capítulo se va a desarrollar un **método alternativo**: éste consiste en seguir el mismo esquema de trabajo descrito en el apartado 3.3, pero dejando que sean las funciones MATLAB las que realicen las operaciones necesarias para calcular las pendientes. Esto tiene dos consecuencias:

- Se simplifican enormemente los cálculos matemáticos necesarios.
- La rutinas MATLAB pasan a ser más complejas ya que asumen el proceso de cálculo.

La nueva metodología se muestra en la figura 3.11:

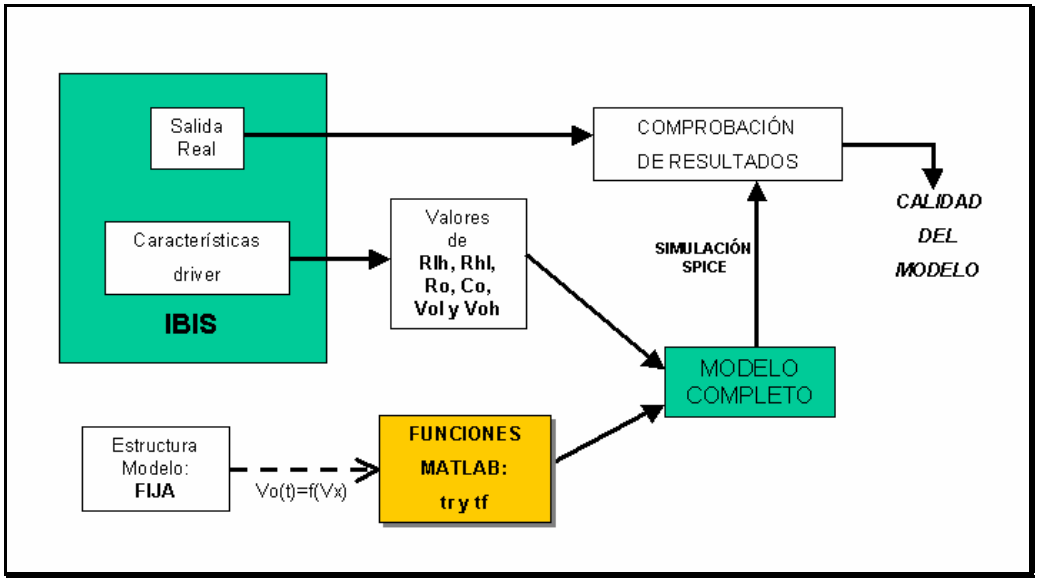

Figura 3.11: Obtención de tr y tf mediante funciones MATLAB.

3.9. Evolución en el tiempo de la salida respecto a Vx.

Se buscan expresiones de la salida Vo en función de la fuente genérica Vx[1] tanto para la característica de subida como de bajada. Éstas se usarán en las funciones MATLAB.

3.9.1. Característica de subida.

El circuito de la figura 3.3 da lugar a la siguiente ecuación diferencial:

$$\frac{dVo(t)}{dt} + \frac{1}{Rlh \cdot C} \cdot \left(1 + \frac{Rlh}{Ro}\right) \cdot Vo(t) = \frac{1}{Rlh \cdot C} \cdot Vx(t)$$

Resolviendo la ecuación se obtiene:

$$Vo(t) = \frac{\alpha}{\beta} \cdot Vx + e^{-\beta \cdot t} \cdot \left(-\frac{\alpha}{\beta} \cdot Vx + vini\right)$$

donde

$$\alpha = \frac{1}{Rlh \cdot C}$$
$$k = 1 + \frac{Rlh}{Ro}$$
$$\beta = \alpha \cdot k$$

Que será la expresión utilizada en las funciones MATLAB de cálculo de característica de subida, independientemente de la complejidad de la fuente Vx. Esta expresión es exactamente igual a la calculada en el apartado 3.4.1.1, mas sin sustituir la expresión concreta de Vx.

[1] En el apartado 3.4.1 se consideró una estructura determinada para Vx, ahora se realiza el cálculo de manera general.

3.9.2. Característica de bajada.

De manera equivalente al apartado anterior, la ecuación diferencial que establece el circuito de la figura 3.5 es:

$$\frac{dVo}{dt}+\frac{1}{Ro\cdot C}\cdot\left(1+\frac{Ro}{Rhl}\right)\cdot Vo=\frac{1}{Ro\cdot C}\cdot Voh+\frac{1}{Rhl\cdot C}\cdot Vx$$

Al resolver la ecuación se tiene:

$$Vo(t)=\frac{k\cdot Voh}{\Omega}+\frac{\tau\cdot Vx}{\Omega}+e^{-\beta\cdot t}\cdot\left(-\frac{k\cdot Voh}{\Omega}-\frac{\tau\cdot Vx}{\Omega}+vini\right)$$

donde

$$\tau=\frac{1}{Rhl\cdot C}$$
$$\Omega=\frac{1}{Ro\cdot C}\cdot\left(1+\frac{Ro}{Rhl}\right)$$
$$k=\frac{1}{Ro\cdot C}$$

De nueva esta expresión es independiente de la complejidad de la fuente Vx, y genera la expresión del apartado 3.4.1.2 sin más que particularizar para la fuente de la figura 3.6.

3.10. Funciones MATLAB para el cálculo de tr y tf.

Conocidas las expresiones de la salida en función de la tensión de control, se desarrollan programas en MATLAB para el cálculo de los tiempos de subida y bajada.

La figura 3.12 muestra un esquema general del funcionamiento del algoritmo para el cálculo de tr. La rutina para el cálculo de tf sigue la misma estructura.

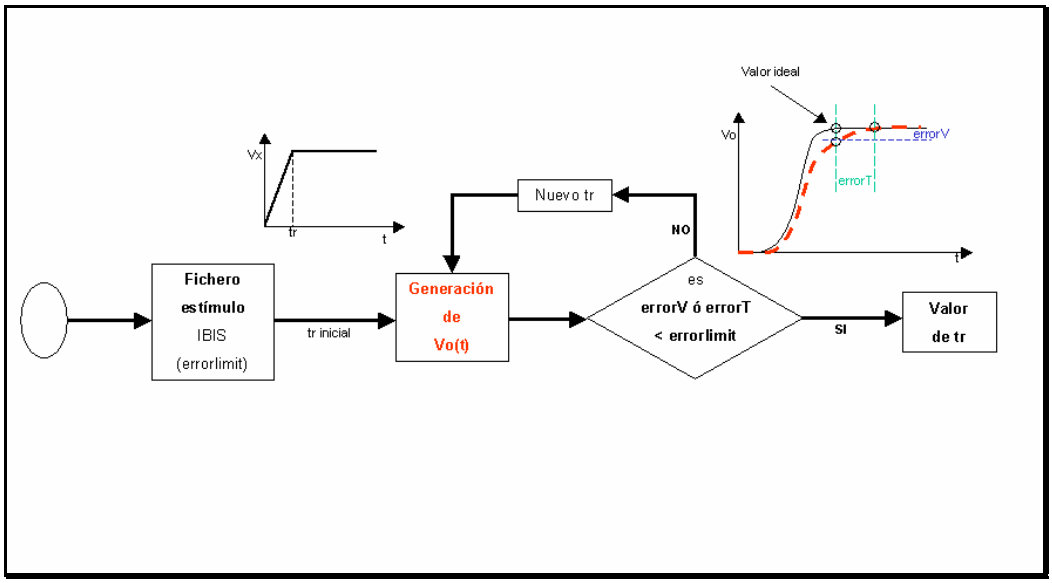

Figura 3.12: Esquema del algoritmo pls_r.m

El fichero de estímulo (o fichero de tecnología), consiste en una forma de agrupar toda la información conseguida de IBIS de manera fácil, para simplificar en gran medida la llamada a las rutinas MATLAB. Cada listado IBIS requerirá un fichero de tecnología diferente.

Ambas funciones se muestra a continuación, pudiéndose encontrar en ellas un gran número de comentarios explicativos.

```
function [tr,time,errorV,errorT] = pls_r(Vol,Voh,Rlh,Ro,C,tfin,dVr,dtr,interv,errorlimit)

% MATLAB function used to calculate automatically the Vx's raise slope,
% necessary for the raise characteristic model. Developed by Mario Palma.
% tfin:Final time for the simulation; it must be greater than Voutput's rise time.
% errorlimit: To define the programme's accuracy.

taux=clock;                 %to control the execution time.
alfa=1/(Rlh*C);                 %Circuit defined constans.
k=1+(Rlh/Ro);
beta=alfa*k;

% Vx: y axis value, it does not change during the execution. The model
% will be controlled by the x axis (time).

Vx=[linspace(Vol,Voh,interv),ones(1,interv)*Voh];
tam=length(Vx);             % Vx (and t) size.

tr=dtr/0.6;         % Initial value for Vx's rise time. This is always
                    % too high.
errorV=-10;         % Initial values so that the program can enter the
errorT=-10;         % loop.
aux0=0;
aux1=tr;
aux2=0;

% The program is in the loop while any error function is bigger than 'errorlimit'

while (abs(errorV)>errorlimit | abs(errorT)>errorlimit)
    vini=Vol/k;                 % Initial output voltage, when the
                                % capacitor is loaded with Vol.
    a=linspace(tr,tfin,(interv+1));
    t=[linspace(0,tr,interv),a(2:(interv+1))];      % x axis (time) creation.
    t1=[0 t];

% Output voltage creation.
    for n=1:tam
        V(n)=((1/beta)*alfa*Vx(n))+exp(-beta*(t1(n+1)-t1(n)))*(1/beta)*alfa*Vx(n)+vini);
        vini=V(n);
    end
```

```
        Vres=interp1(t,V,dtr/0.6);      % Interpolation to get the output voltage
                                        % when the model is in the ideal time.
        Vfin=Vol+dVr/0.6;
        errorV=(Vres-Vfin)/Vfin;        % Definition of the voltage error function.

        if (abs(errorV)>errorlimit & aux0==0)       % Rise time is still too high.
                aux1=tr;
                tr=tr/5;
        elseif (abs(errorV)>errorlimit & aux0==1)   % tr is now %boundered. Rise time is still
                                                    %too high.
                aux1=tr;
                tr=(tr+aux2)/2;

        else                            % Voltage error is already controlled.
            aux0=1;
            fin=V(tam);
            q=find(V >=(floor(fin*100)/100));
            k=q(1);
            tres=t(k);
            errorT=(tres-dtr/0.6)/(dtr/0.6); % Definition of the time error function.

            if errorT<-errorlimit               % Rise time is too high.
                aux2=tr;
                tr=(tr+aux1)/2;
            end

            if errorT>errorlimit                % Rise time is too low.
                aux1=tr;
                tr=(aux1+aux2)/2;
            end
        end
        if (tr<1e-30)
            error('Vx´s rise time (tr) is shorter than 1*e-20 seconds')
        end
end

plot(t,Vx,'--',t,V); grid on; zoom;     % Vx and output voltage are plotted.
xlabel('time'); ylabel('voltage');
time=etime(clock,taux);                 % It is possible to control the execution time
end
```

```
function [tf,time,errorV,errorT] = pls_f(Vol,Voh,Rhl,Ro,C,tfin,dVf,dtf,interv,errorlimit)

    % MATLAB function used to calculate automatically the Vx's falling slope,
    % necessary for the fall characteristic model. Developed by Mario Palma.
    % tfin:Final time for the simulation; it must be greater than Voutput's fall time.
    % interv: Number of points that are going to be used.
    % errorlimit: To define the programme's accuracy.

    taux=clock;                 %to control the execution time.
    tau=1/(Rhl*C);                  %Circuit defined constans.
    ka=1/(Ro*C);
    omega=ka*(1+Ro/Rhl);



    % Vx: y axis value, it does not change during the execution. The model
    % will be controlled by the x axis (time).

    Vx=[linspace(Voh,Vol,interv),ones(1,interv)*Vol];

    tam=length(Vx);             % Vx (and t) size.
    tf=dtf/0.6;             % Initial value for Vx's fall time. This is always
                            % too high.
    errorV=-10;             % Initial values so that the program can enter the
    errorT=-10;             % loop.
    aux0=0;
    aux1=tf;
    aux2=0;

    % The program is in the loop while any error function is bigger than 'errorlimit'.
    while (abs(errorV)>errorlimit | abs(errorT)>errorlimit)
        vini=Voh;                   % Initial output voltage, when the
                                    % capacitor is loaded with Voh.
        a=linspace(tf,tfin,(interv+1));
        t=[linspace(0,tf,interv),a(2:(interv+1))];      % x axis (time) creation.
        t1=[0 t];

    % Output voltage creation.
        for n=1:tam
            V(n)=1/omega*ka*Voh+1/omega*tau*Vx(n)+exp(-omega*(t1(n+1)-t1(n)))*
            (-1/omega*ka*Voh-1/omega*tau*Vx(n)+vini);
```

```
        vini=V(n);
    end
    Vres=interp1(t,V,dtf/0.6);    % Interpolation to get the output voltage
                                  % when the model is in the ideal time.
    Vfin=Voh-dVf/0.6;             % Final value, following IBIS specification.
    errorV=(Vres-Vfin)/Vfin;      % Definition of the voltage error function.

    if (abs(errorV)>errorlimit & aux0==0)       % tf is still too high.
            aux1=tf;
            tf=tf/5;

    elseif (abs(errorV)>errorlimit & aux0==1)   % tf is now boundered. Rise time is still
                                                %too high.
            aux1=tf;
            tf=(tf+aux2)/2;

    else                          % Voltage error is already controlled.
        aux0=1;
        fin=V(tam);
        q=find(V <=(ceil(fin*100)/100));
        k=q(1);
        tres=t(k);
        errorT=(tres-dtf/0.6)/(dtf/0.6); % Definition of the time error function.

        if errorT<-errorlimit             % Fall time is too low.
            aux2=tf;
            tf=(tf+aux1)/2;
        end

        if errorT>errorlimit              % Fall time is too high.
            aux1=tf;
            tf=(aux1+aux2)/2;
        end
    end

    if (tf<1e-20)
    error('Vx´s fall time (tf) is shorter than 1*e-20 seconds. Revise your IBIS file')
    end
end    % end of the while loop.

plot(t,Vx,'--',t,V); grid on; zoom;      % Vx and output voltage are plotted.
```

```
xlabel('time'); ylabel('voltage');

        time=etime(clock,taux);          % It is possible to control the
                                         % execution time.
end
```

3.11. Ejemplo: resultados con el modelo DR-1[2].

Con el **apéndice 1** se puede comprobar la calidad de las salidas mostradas, comparando con los datos del modelo real.

El tiempo de ejecución para estos ejemplos concretos es de 0.5 segundos para la característica de subida y de 0.51 segundos para la de bajada, trabajando con 50 puntos en cada sección. El fichero de tecnología para este modelo que se emplea en MATLAB se lista en el **apéndice 2.**

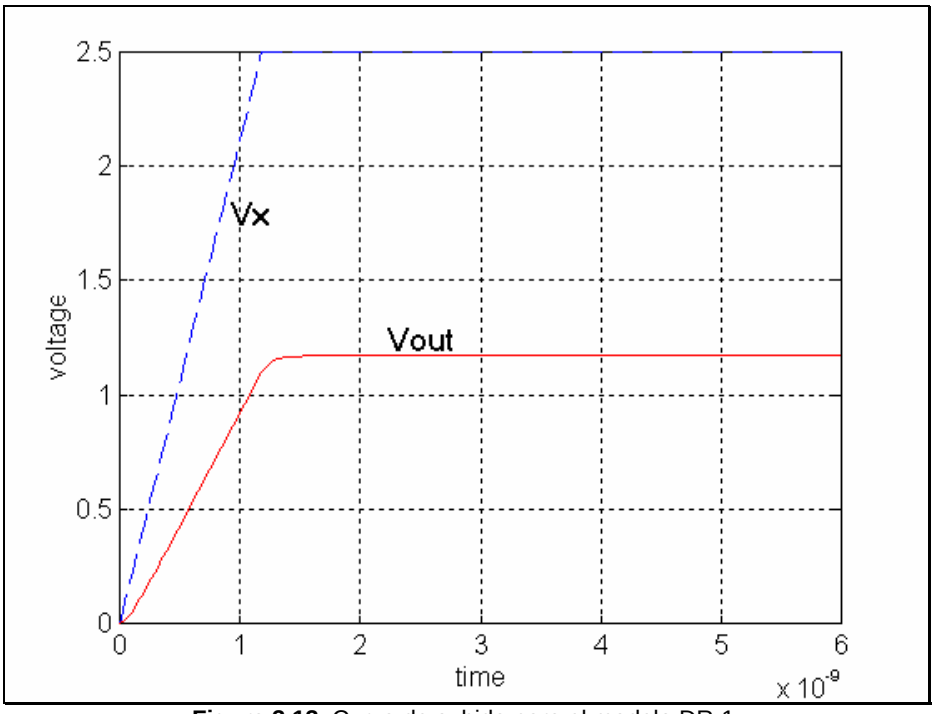

Figura 3.13: Curva de subida para el modelo DR-1.

[2] Al ser un modelo comercial no disponible en Internet se ha tenido que modificar el nombre y alguna información irrelevante para este Proyecto como condición para poder listar el fichero IBIS del mismo.

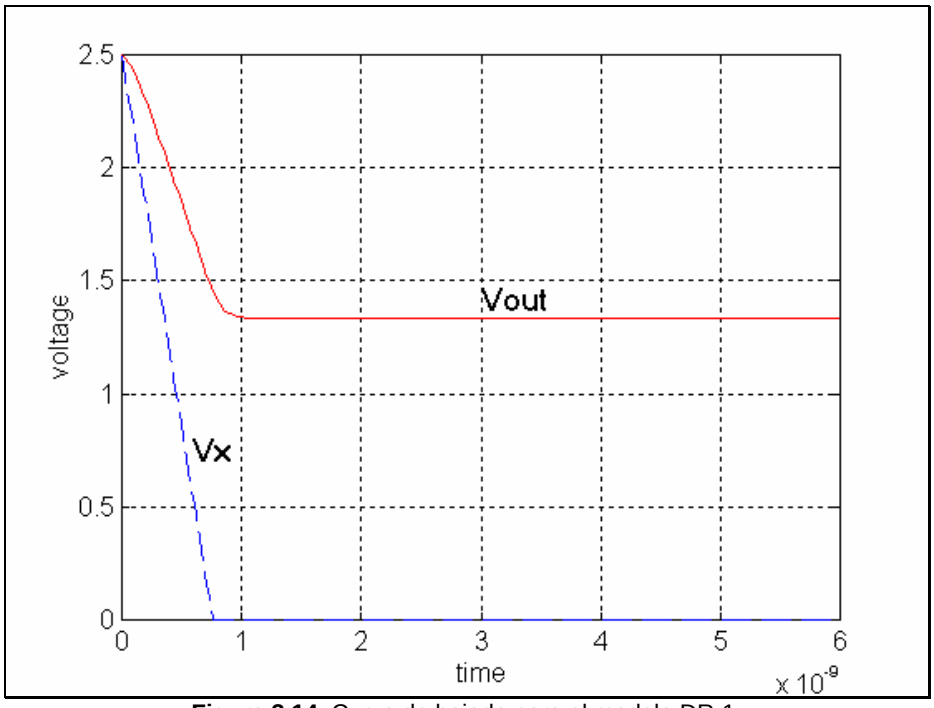

Figura 3.14: Curva de bajada para el modelo DR-1.