2 Programação de Simulink S-functions

2.1 S-function

É uma descrição de um bloco do simulink numa linguagem de programação, que pode ser codificada em Matlab, C / C++, Fortran ou Ada.

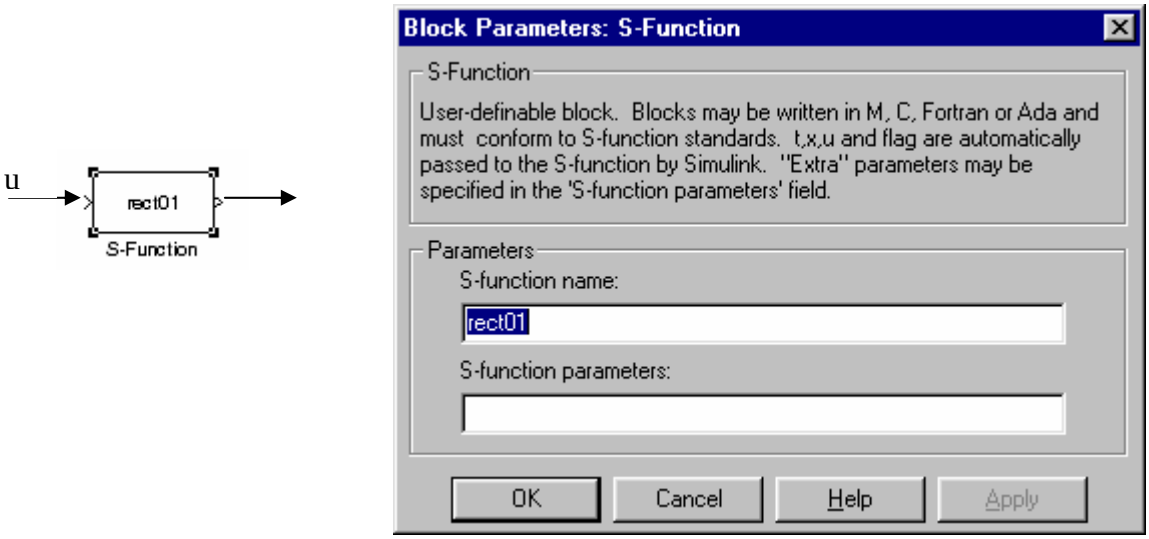

Fig. 2-1: (S-function) bloco do Simulink e parâmetros do bloco

Na janela de parâmetros do bloco S-function deve ser indicado o nome da função. Se fôr codificada em Matlab, o código deve ser guardado numa m-file com o mesmo nome da função (e extensão .m).

Um bloco Simulink pode ser visto como:

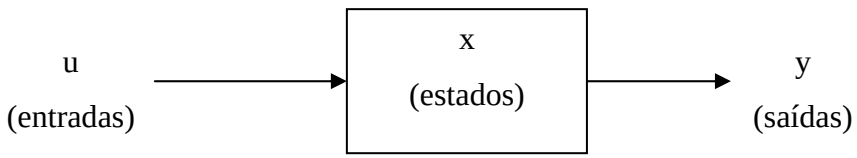

Fig. 2-2: Representação de um bloco do Simulink

E segundo um modelo de espaço de estados, e sabendo que se tem estados contínuos e discretos, isto é:

$x = x_d + x_c$

tem-se as equações dinâmicas:

$\dot{x}_c = Ax_c + Bu$ (derivada dos estados contínuos)

$y = Cx + Du$ (equação de saída)

No caso de se ter estados discretos em cada instante de amostragem deve ser feita a actualização destes (o equivalente discreto da derivada dos estados contínuos):

$x_d^{k+1} = Ax_d^k + Bu^k$ ou $x_d^{k+1} = f(t, x_d^k, u^k)$

i. e., como uma função dos estados anteriores e da entrada actual.

A codificação em Matlab toma a forma de um função:

[sys,x0,str,ts] = f (t,x,u,flag,p1,p2,...)

parâmetros de entrada:

f	Nome da S-function's
t	Tempo corrente
x	vector de estado
u	entrada do bloco
flag	Indica uma tarefa a ser desempenhada durante a simulação
p1, p2, ...	São parâmetros adicionais Quando o tempo de amostragem varia normalmente p1 é usado para indicar esse valor à S-function. Os valores destes parâmetros devem ser indicados na janela de parâmetros do bloco, mostrada na Fig. 2-1, na linha de entrada intitulada S-function parameters separados por vígulas. **Podem ser constantes ou variáveis definidas no workspace do Matlab.**

parâmetros de saída

sys	argumento de retorno genérico. Os valores dependem de flag. (Ex. se flag = 3, sys contém a saída da S-function (ou do bloco))

x0	Valores iniciais dos estados
str	Reservado para uso futuro. S-functions devem retornar a matriz vazia [ ]
ts	Matriz com duas colunas contendo o intervalo de amostragem

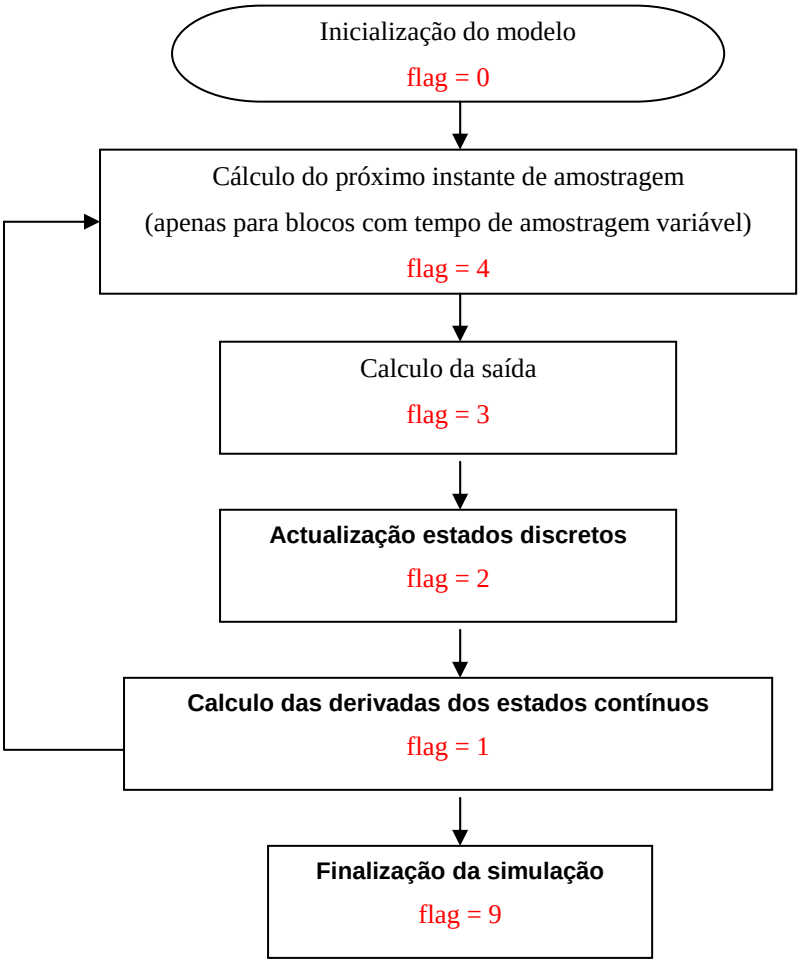

Fig. 2-3: Etapas de uma simulação e correspondência com o valor de flag.

A simulação segue o diagrama anterior. No inicio todos os blocos são inicializados (flag=0). Entra-se então num ciclo de simulação onde primeiro é calculado o próximo instante de amostragem (flag=4) de modo que o cálculo da saída (flag=3) seja executado apenas após a amostragem de novos dados. Seguidamente são actualizados os estados discretos (flag=2). Antes de se aguardar pela próxima amostra, são ainda calculadas as derivadas dos estados contínuos (flag=1).

Podem ainda ser efectuadas algumas tarefas antes do fim da simulação (flag=9).

Como ilustra o diagrama as tarefas que são efectuadas durante a simulação dependem do valor do parâmetro de entrada flag. Este parâmetro é fornecido pelo próprio Simulink, cada vez que chama a S-function de modo a indicar qual a etapa em que a simulação se encontra.

2.2 Implementação de s-functions

Neste ponto serão apresentados alguns exemplos de S-functions. Basicamente uma S-function é dividida em segmentos de código que executam uma tarefa dada a etapa corrente da simulação e que é especificada pelo valor de flag:

```
[sys, x0, str, ts] = sfunc (t, x, u, flag)

if flag==0
(...)

elseif flag == 4
(...)
```

2.2.1 Entrada e saída únicas

No exemplo seguinte admite-se que o tempo de amostragem é variável e portanto é passado como primeiro parâmetro adicional, **ts**. Deste modo o valor deste parâmetro deve ser especificado na janela de parâmetros do bloco, Fig. 2-1. Este exemplo consiste simplesmente num ganho que duplica o valor do sinal de entrada:

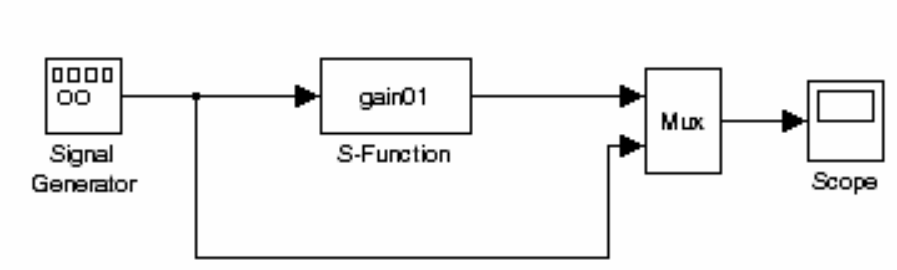

Fig. 2-4: Diagrama para a s-function gain01

Nota: O sinal é uma sinusóide com amplitude 1 e frequência 1 rad / s.
Um período de amostragem conveniente será 0.1. (Pode-se tentar 1 para ver o efeito)

```
function [sys, x0, str, ts] = gain01(t, x, u, flag, ts)

%inicialização
if flag == 0
        %estados contínuos = 0
        %estados discretos = 0
        %saídas = 1
        %entradas = 1
        %raizes continuas. reservado deve ser 0
        %direct feedtrough deve ser 1 se u usado em flag=3
        %sample times = 1
        sys = [0 0 1 1 0 1 1];
        x0 = [ ];
        str = [ ];
        ts = [-2 0];                                    %tempo de amostragem variável

%Calcula próximo instante de amostragem
elseif flag == 4
        ns = t / ts;                                    %ns nº de amostras
        sys = (1 + floor(ns + 1e-13*(1+ns)))*ts;        %momento da próxima amostra

%Calcula a saída como o dobro da entrada
elseif flag == 3
        sys = 2*u(1);

%default
else
        sys = [ ];                                              %não faz nada
end
```

2.2.2 Entradas e saídas múltiplas

No entanto um função pode receber mais de uma entrada. Se como 2ª entrada se encontrar o factor de ganho, o diagrama de blocos passa a ser:

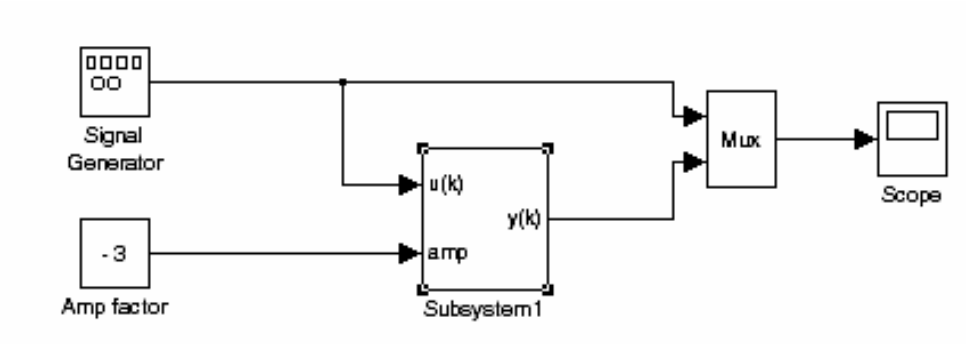

Fig. 2-5: Ganho variável

Repare-se que a S-function, mais as suas entradas e saídas foram agrupadas num subsistema. No caso de múltiplas entradas é usado um *Multiplexer* de modo a transformar os sinais independentes num vector. Quanto à saída consiste num vector com tantas posições como o número de saídas de modo que se for ligada a um *Scope* apresenta tantos sinais como o número de saídas.

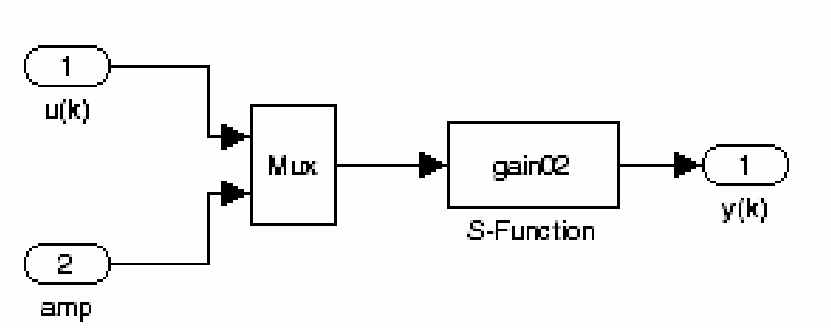

Fig. 2-6: S-function com 2 entradas

As alterações que devem ser efectuadas ao algoritmo anterior:

```
else if flag == 0          %Inicalização
        sys = [ 0 0 1 2 0 1 1 ]

elseif flag == 3           %Calcula a saída como um factor da 1ª entrada dado pela 2ª
        sys = u(2)*u(1);
```

2.2.3 Memorização de estados da S-function entre iterações

Nos casos anteriores não foram necessários estados. Vamos no entanto supor que é necessário guardar um valor do passo anterior para efectuar um qualquer cálculo. Uma forma de fazer isto (*the hard way*), consiste em guardar num estado discreto esse valor.

O seguinte exemplo calcula a diferença entre a entrada corrente u(k) e a anterior u(k-1). Para isso deve ser memorizado o valor da anterior.

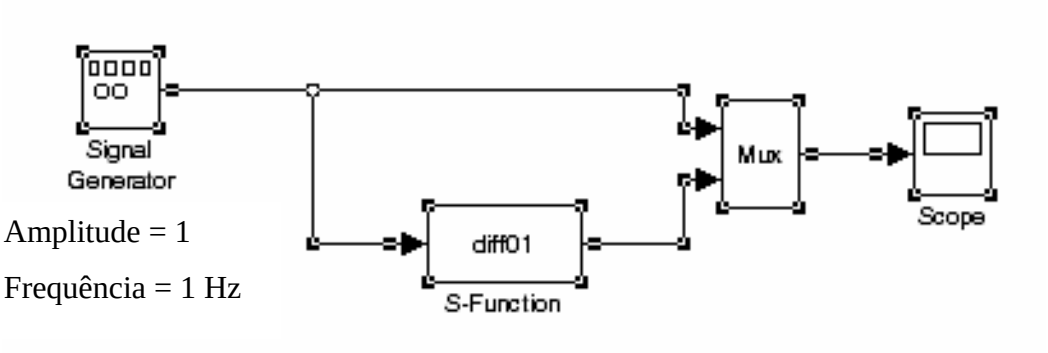

Fig. 2-7: diferença entre a entrada actual e a anterior

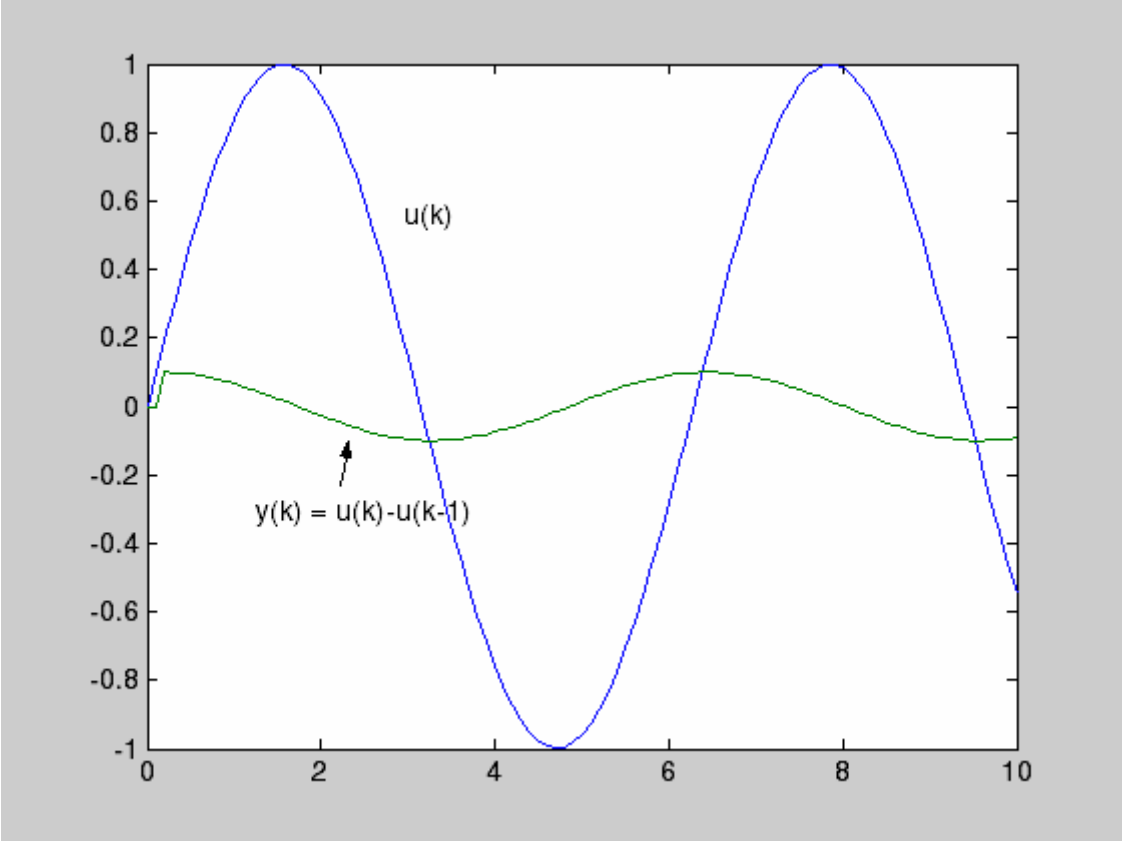

Fig. 2-8: Entrada e saída do diagrama anterior

O código para a S-Function (a vermelho as alterações em relação a gain01):

```
function [sys, x0, str, ts] = diff01 (t, x, u, flag, ts)     %<---
%inicialização
if flag == 0
        %estados contínuos = 0
        %estados discretos = 2                          %<--- 2 estados discretos armazenam:
                                                                u(k-1) e y(k)
        %saídas = 1
        %entradas = 1                                   %<---
        %raizes continuas. reservado deve ser 0
        %direct feedtrough deve ser 1 se u usado em flag=3
        %sample times = 1
        sys = [0 2 1 1 0 1 1];                          %<---
        x0 = [0 0];                                     %<--- 2 Estados iniciais = 0
        str = [ ];
        ts = [-2 0];                                    %tempo de amostragem variável

%Calcula próximo instante de amostragem
elseif flag == 4
        ns = t / ts;                                        %ns nº de amostras
        sys = (1 + floor(ns + 1e-13*(1+ns)))*ts;        %momento da próxima amostra

%Actualiza os estados discretos e envia o vector de estado  %<---
elseif flag == 2
        y=u(1)-x(1);                    %y=u(k)-u(k-1)
        sys = [u(1) y];                 %x=[u(k-1) y(k)]

%Retira a saida do vector de estado
elseif flag == 3
        sys = x(2);                                         %<---

%default
else
        sys = [ ];                                              %não faz nada
end
```

Se bem que exista uma meio mais simples este exemplo serve também para ilustrar o uso da actualização dos estados discretos.

Outra forma de implementar o exemplo anterior (*the easy way*), possível apenas nas versões recentes do Matlab, consiste em declarar uma variável local que mantém o seu valor entre diferentes chamadas a uma função, da mesma forma que as variáveis estáticas em C / C++. Para isso usa-se a palavra reservada **presistent**. O código para a S-Function (a vermelho as alterações em relação a gain01):

```
function [sys, x0, str, ts] = diff02 (t, x, u, flag, ts)     %<---
persistent uk_1;                                             %<---
%inicialização
if flag == 0
        sys = [0 0 1 1 0 1 1];
        x0 = [  ];
        str = [ ];
        ts = [-2 0];                                         %tempo de amostragem variável
        uk_1 = 0;                                            %<---

%Calcula próximo instante de amostragem
elseif flag == 4
        ns = t / ts;                                         %ns nº de amostras
        sys = (1 + floor(ns + 1e-13*(1+ns)))*ts;             %momento da próxima amostra

%Calcula a saída como a diferença entre a entrada actual e a anterior        %<---
elseif flag == 3
        sys = u(1)-uk_1;
        uk_1=u(1);

%default
else
        sys = [ ];                                                  %não faz nada
end
```

2.2.4 Mascarando blocos ou sub-sistemas

Pode ser útil mascarar um subsistema de modo a ter um interface para passagem de parâmetros. No caso do sub-sistema da Fig. 2-5 e da Fig. 2-6, a única variável que interessa colocar na máscara é o tempo de amostragem ts. Assim primeiro deve ser colocado na janela de parâmetros da S-function a variável que vem da mascara, por exemplo ts0. Para isso deve ser aberto o sub-sistema:

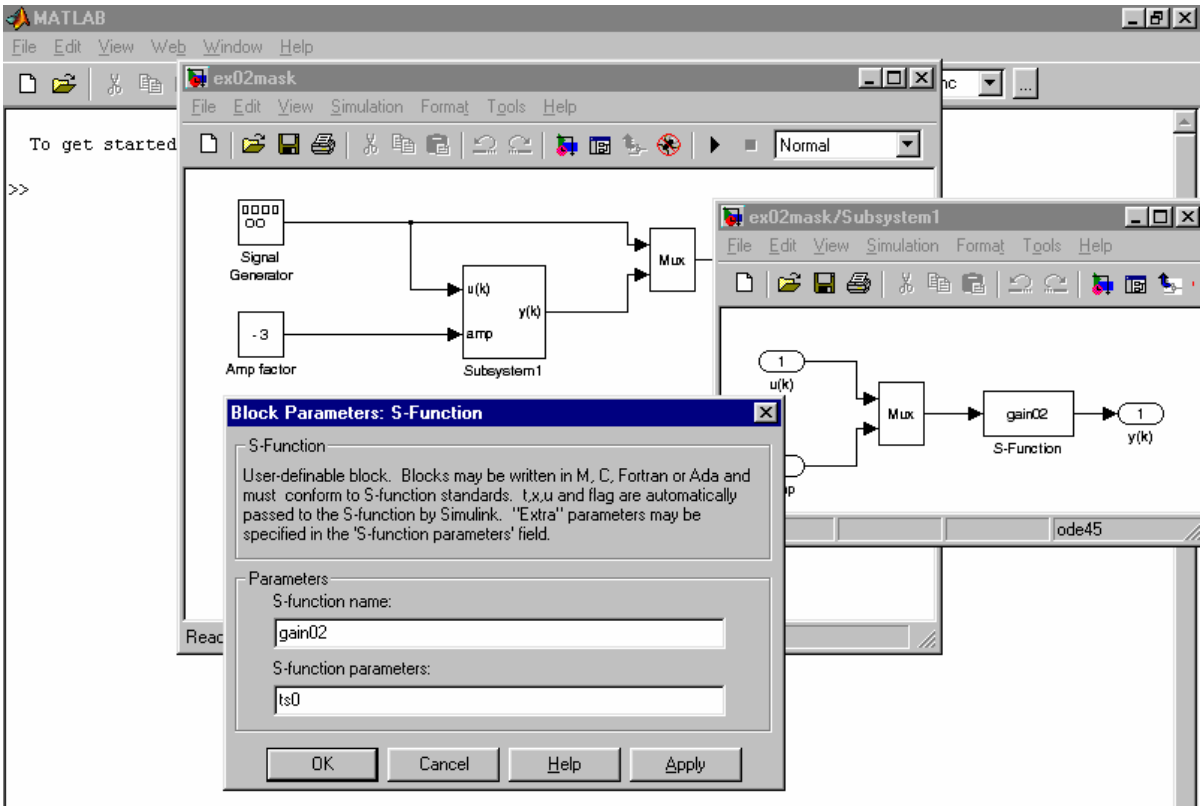

Fig. 2-9: Definição da variável proveniente da mascara

Fig. 2-10: Definição da variável proveniente da mascara

Seguidamente deve ser editada a mascara (Edit Masc). Finalmente o valor do parâmetro ts da S-function pode ser editado na janela de parâmetros do bloco (sub-sistema):

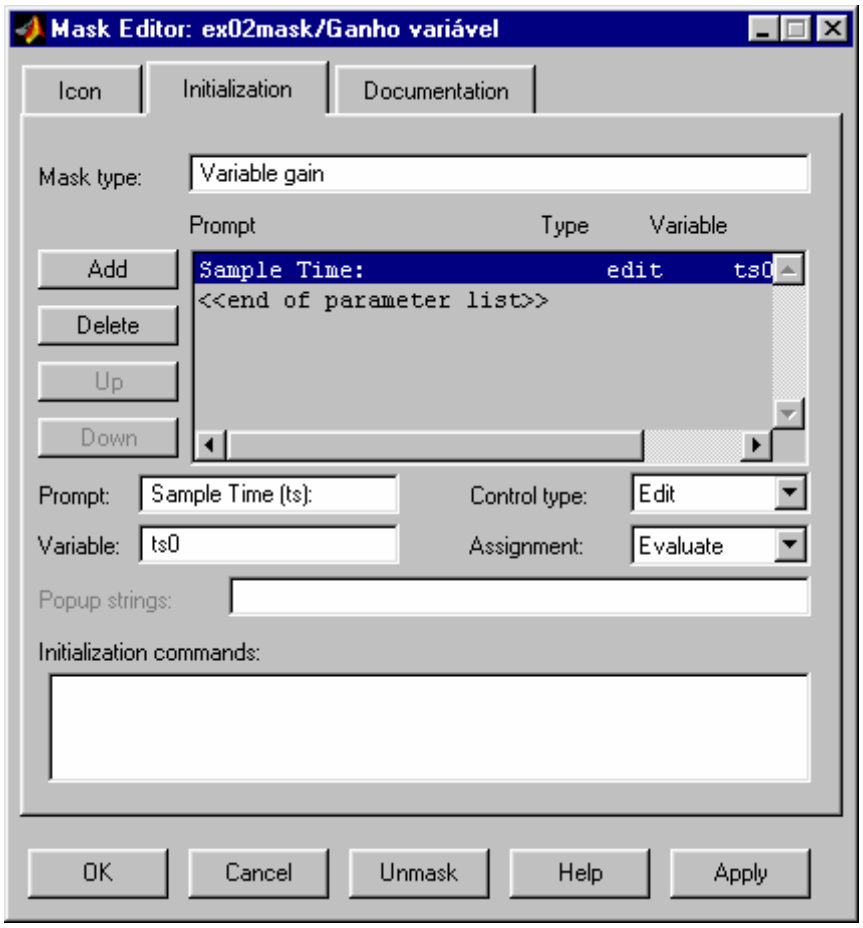

Fig. 2-11: Definição da máscara