# SUPORTE DE CURSO

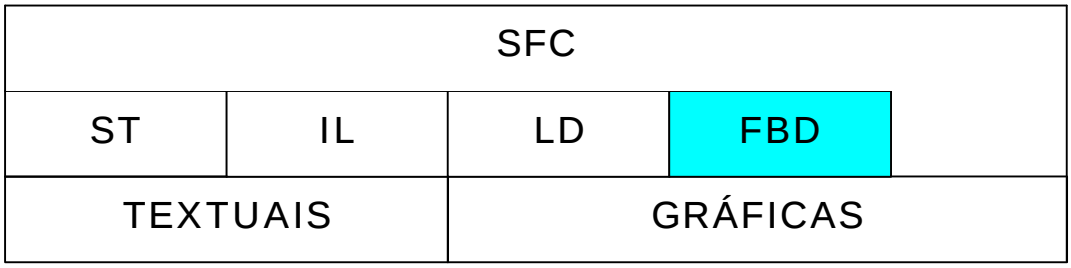
# IEC 1131-3 Function Block Diagram

**Livro Texto:** Programming industrial control systems using IEC 1131-3 – R.W. Lewis

UFMG – Informática Industrial
Prof. Constantino Seixas Filho

# IEC 1131-3 Function Block Diagram

| | |
|---|---|
| Structured Text (ST) | Textuais |
| Instruction List (IL) | |
| Function Block Diagram (FBD) | Gráficas |
| Ladder Diagram (LD) | |
| Sequential Function Charts (SFC) | |

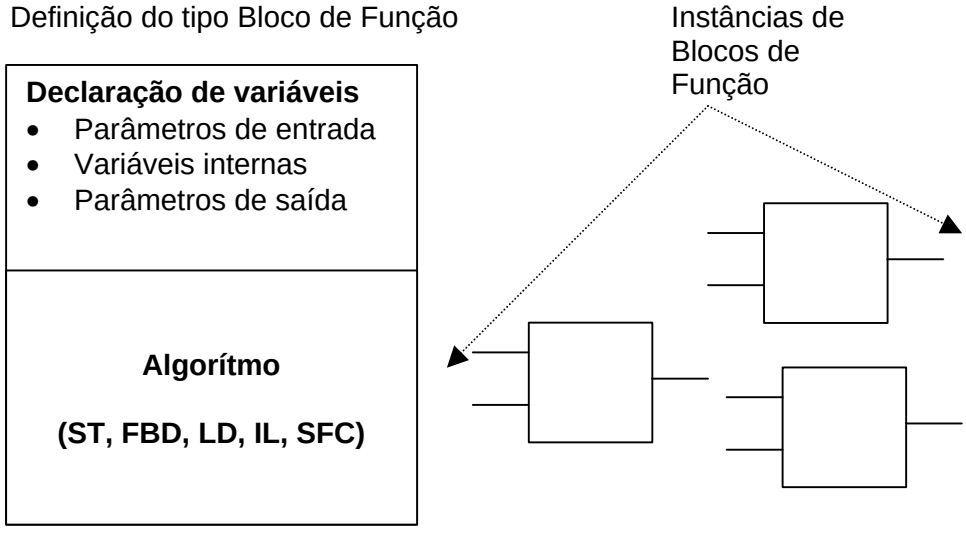



## É usada para descrever o comportamento de :

- Funções
- Blocos de funções
- Programas
- Em SFC para expressar o comportamento de passos, ações e transições.

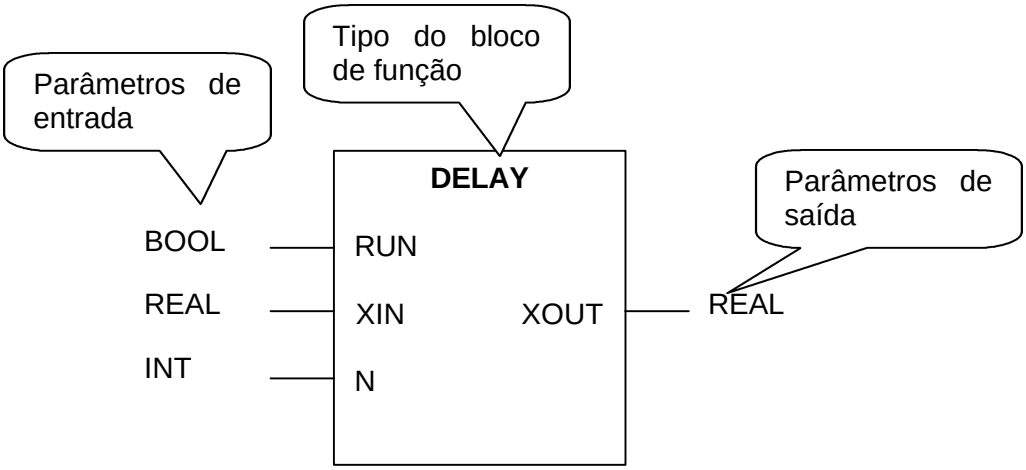
## Características

Modela o sistema em termos do fluxo de sinais entre elementos de processamento.



- Os blocos avaliam todas variáveis internas e de saída a cada ciclo (*scan*).
- O bloco de função possui persistência de dados. Os valores dos dados são mantidos entre dois *scans*.
- Se as avariáveis internas forem definidas com o atributo **RETAIN**, os valores serão mantidos mesmo em caso de falta de energia no CLP.

## Notação:




- Apenas os parâmetros de entrada e de saída de um bloco podem ser acessados externamente. Variáveis internas não são acessíveis.

- Um bloco de função só será executado:
  - Se for parte de uma rede de blocos conectados que formam uma POU.
  - Se chamado por um programa em ST ou IL.

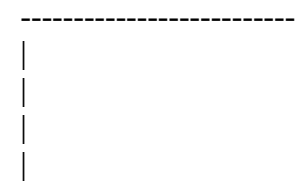
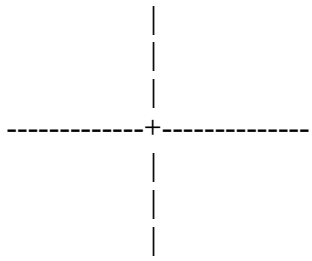
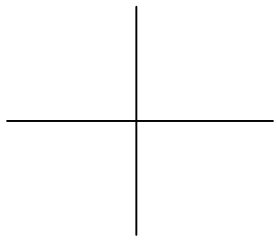
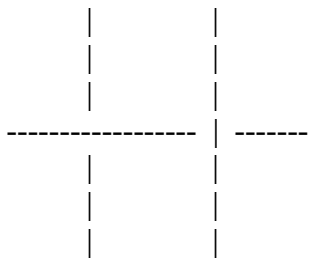
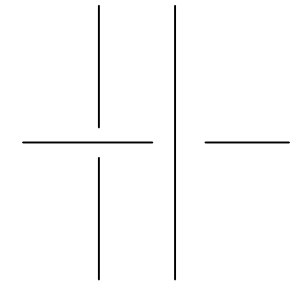
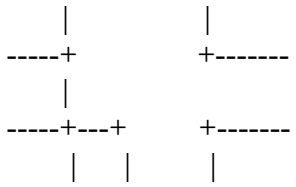
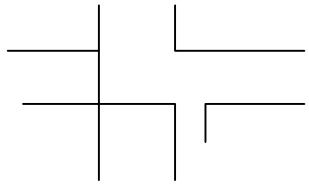
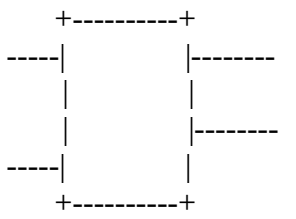
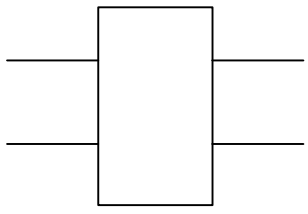
- Instancias de um bloco podem ser utilizadas na definição de outros blocos ou programas, mas não na definição do seu mesmo tipo.
- Instâncias de blocos de função declarados como globais através de **VAR**_GLOBAL são acessíveis em qualquer parte de um recurso ou configuração.

- Ë possível passar uma instância de bloco de função como entrada para outro programa, bloco de função ou função.

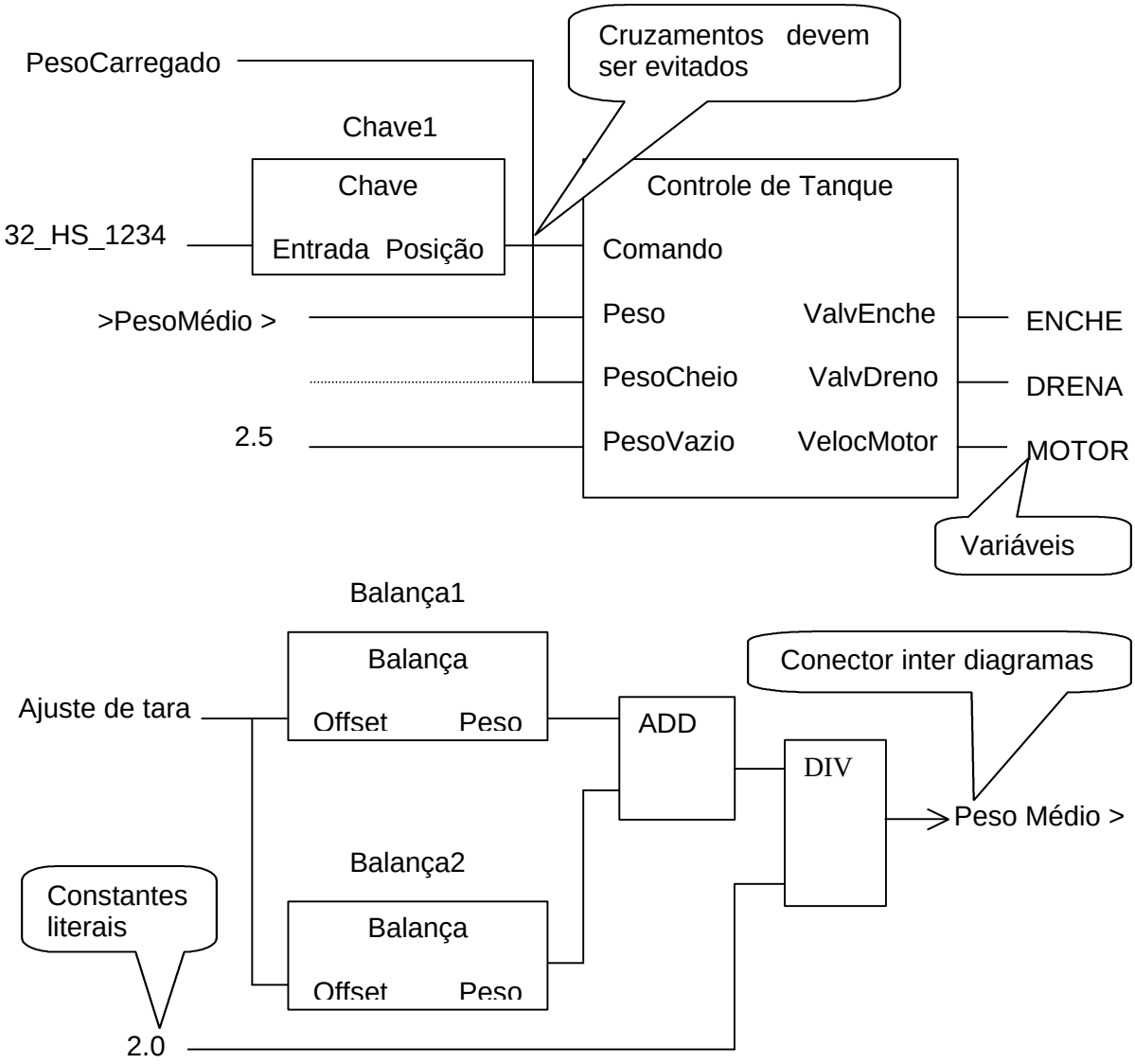
- Os parâmetros de saída e um bloco de função podem ser acessados através da notação: DELAY.XOUT.

## Elementos gráficos

Podem ser representados em forma semi gráfica através dos caracteres - | ou totalmente gráfica.

| **Entidade gráfica** | **Forma semi gráfica** | **Forma gráfica** |
|---|---|---|
| Linhas horizontais e verticais |  |  |
| Interconexão de fluxos de sinais horizontais e verticais |  |  |
| Cruzamento de fluxos de sinais vertical e horizontal |  |  |
| Esquinas de fluxo de sinais |  |  |
| Blocos com conexões |  |  |
| Conectores | -----→ LOAD_JOB><br>>LOAD_JOB>-------- | ⟶ LOAD_JOB ><br>> LOAD_JOB ⟶ |

O standard permite alterações no formato dos blocos tais como arredondamento de cantos, sombreamento, aplicação de cores, etc.



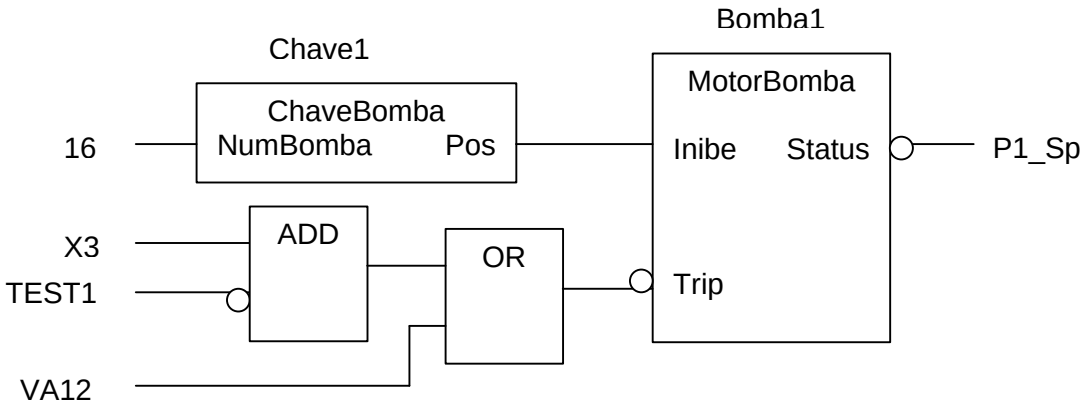

O bloco de função Chave1 é uma instância do bloco de funções Chave. Ele lê o valor de uma *thumbwheel* (chave de codificação decádica) e seleciona o modo de operação do tanque.

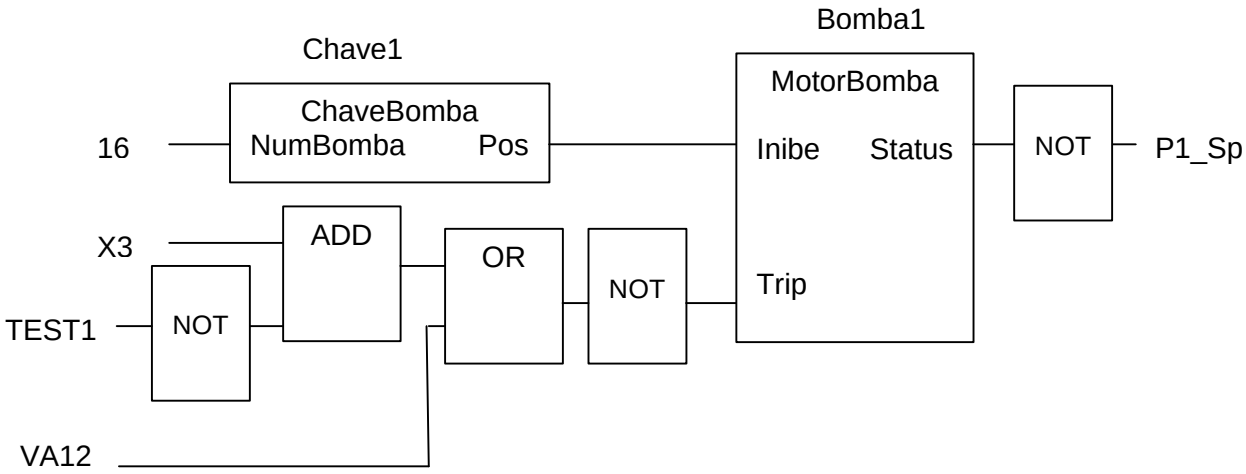
## Fluxo de sinal

Os sinais se propagam da saídas de blocos de função para as entradas de outras funções ou blocos de função.

Tipos de dados diferentes são representados por linhas da mesma espessura. Entretanto é permitido usar linhas de cores diferentes para tipos de dados diferentes.

**Negação de sinais booleanos**

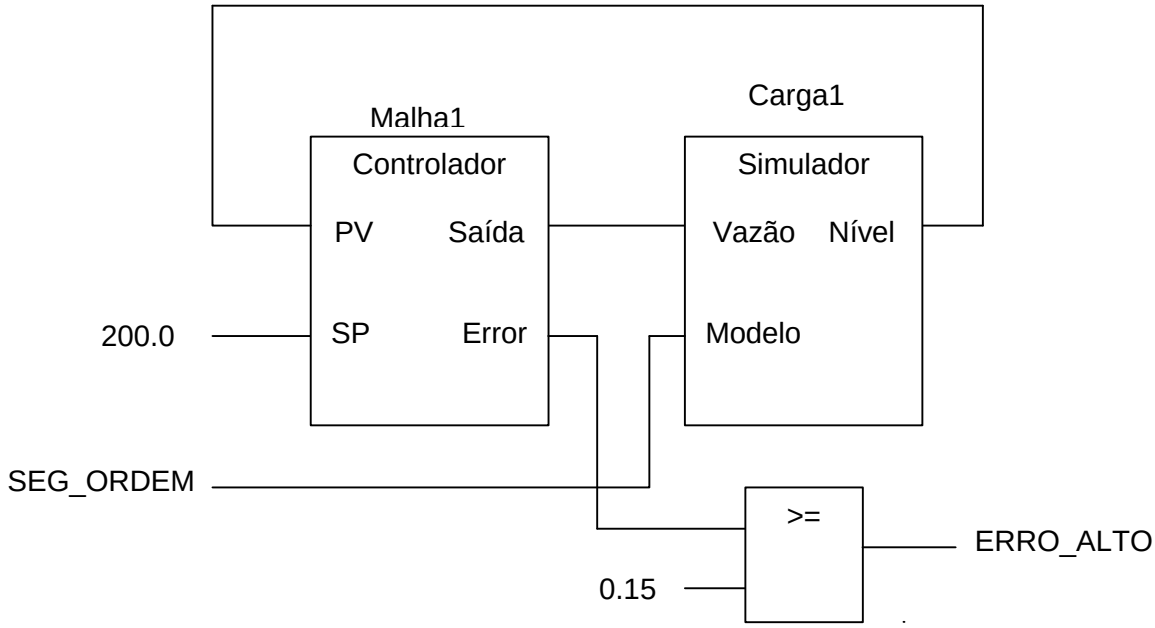



**Realimentação:**

Um sinal de saída de um bloco pode ser realimentado na entrada de um bloco precedente.

O padrão não estabelece que blocos de uma malha será avaliado primeiro. Como a ordem pode variar entre implementações, pode haver diferenças de comportamento entre diferentes compiladores com diferentes resultados finais.




**Limitações no tamanho de FBDs**

O número de FBDs e sua complexidade não são limitados no padrão.

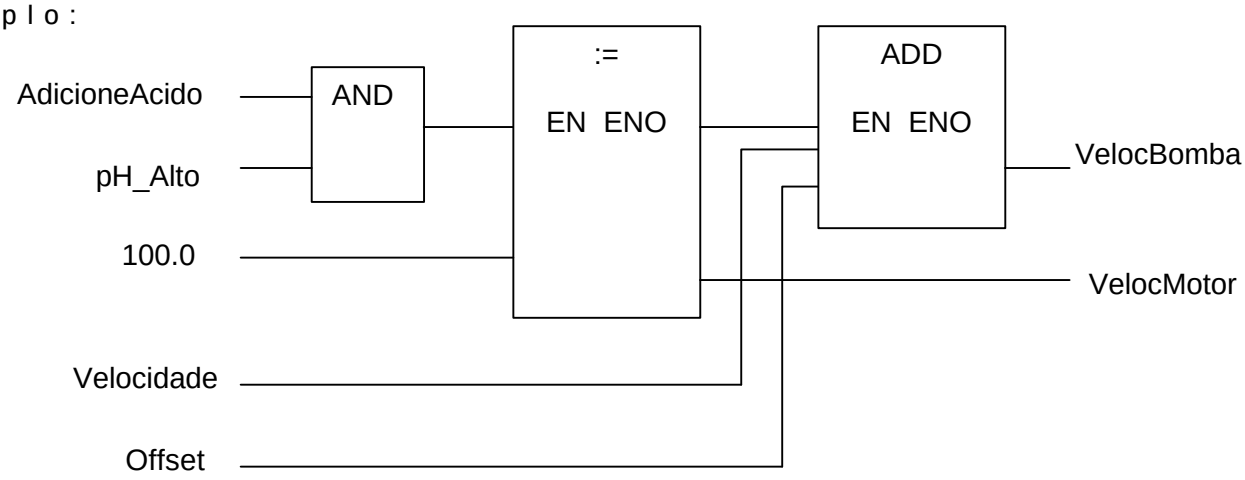
## Controle de execução dos blocos de função

A ordem de execução segue o fluxo natural do diagrama. Por exemplo na figura anterior o bloco >= é executado após o bloco Loop1.

Ë possível avaliar seletivamente blocos de função através da entrada booleana EN (*enable*). Se EN é **TRUE** o bloco é avalizado e produz ima saída, caso contrário não é avaliado.

A saída ENO (*enable output*) é uma saída booleana que indica que uma função completou a sua avaliação. Esta saída pode ser encadeada com as entradas EN de outros blocos.

E x e m p l o :



Quando pH_Alto é **TRUE** e AdicioneAcido é **TRUE**, o Bloco ":=" será ativado e o valor 100 será carregado em VelocMotor.
Em seguida o bloco ADD será ativado e Velocidade e Offset serão somados para produzir VelocBomba.

**Observações:**

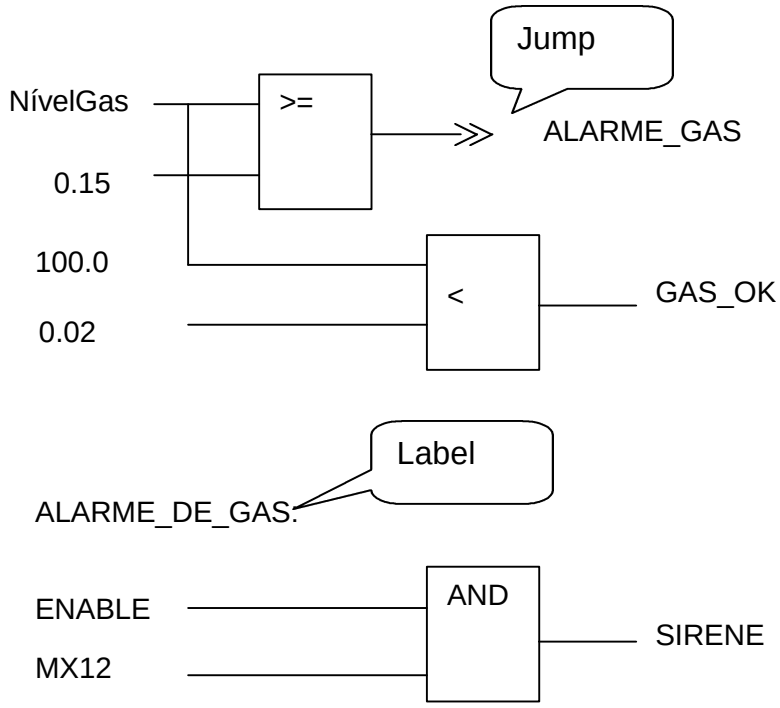
- Se houver erro no processamento do bloco, por exemplo, divisão por zero, EN0 será **FALSE**. Logo ENO serve para indicar se o processamento do bloco foi OK.

- Se EN é **FALSE** qual o valor da saída do Bloco ?
  O standard NÃO diz.

## Jumps e labels

É possível rotular um trecho de uma rede FBD com um label.
É possível pular de um trecho de uma rede FBD para este label:

Exemplo




**Observações:**

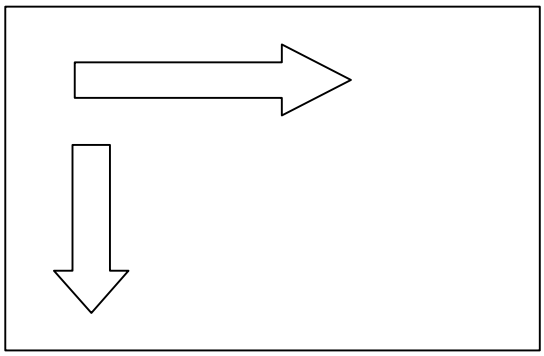
- Que acontece com a avaliação do resto da rede de FBD após o jump ocorrer ?
  A avaliação continua ?
  A avaliação é interrompida ?
  standard não esclarece.
- **Jumps apesar de existentes são construções não recomendadas**.

# Avaliação de rede de FDB

**Regras:**

A ordem de avaliação varia de um produto para outro.

**Geralmente…**

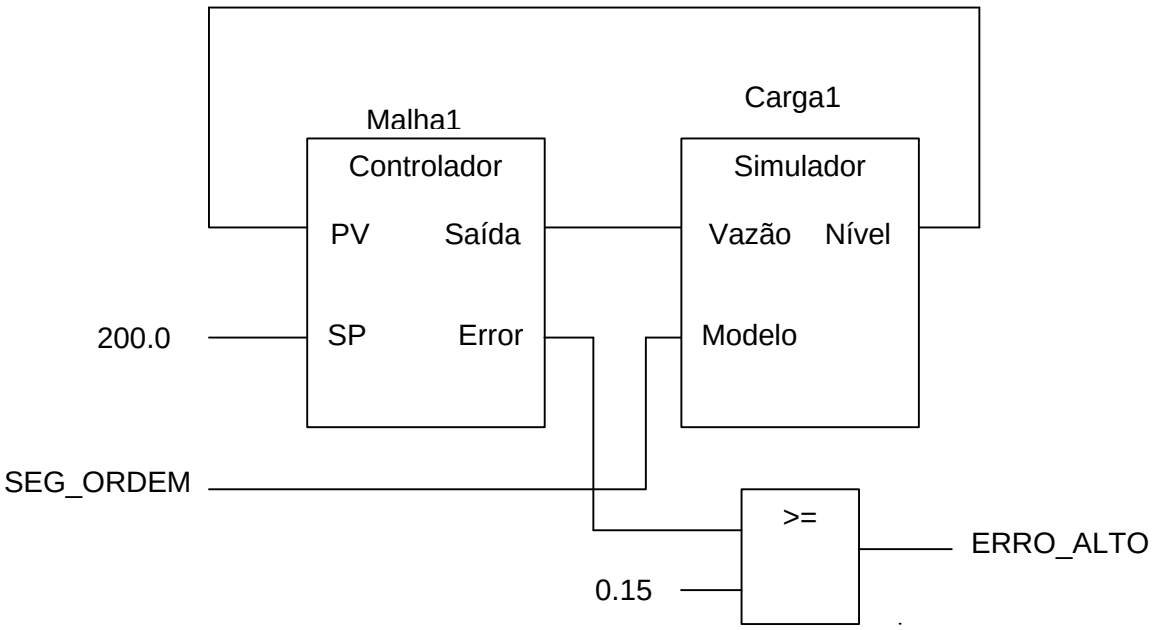


- Nenhum elemento da rede será avaliado a menos que todas as suas entradas proveniente de outros elementos tenham sido avaliadas.
- A avaliação não estará completa até que todas as saídas tenham sido definidas.
- A avaliação de uma rede só estará concluída quando a avaliação de todos os blocos de função tiver sido concluída.
- Quando dados são transferidos de um a rede para outra, todos os valores indo de uma rede a outra devem ter sido produzidos pela mesma avaliação da rede.

## Portabilidade entre ST e FBD

Nem todas as *features* de uma linguagem IEC1131-3 pode ser traduzida em outras linguagens.
Apenas um núcleo de funções básicas é portável.

Exemplo Traduzir o exemplo seguinte em ST




```
VAR
        Malha1: Controlador;
        Carga1: Simulador;
END_VAR

(* Invoque os blocos de função com conexão entre parâmetros *)
Malha1(PV := Carga1.Nível, SP:=200.0);
Carga1(Vazão:= Malha1.Saída, Modelo:= SEG_ORDEM);
ERRO_ALTO := Malha1.Error >= 0.15;
```

## Dificuldades na tradução FBD-> ST

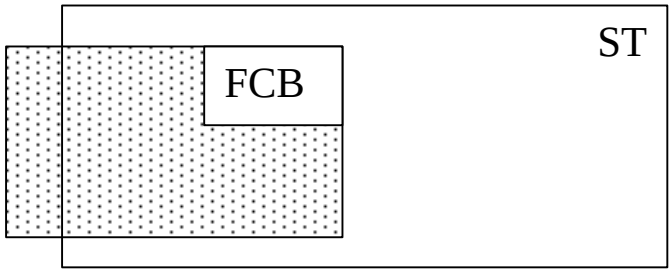
Funções que controlam sua execução explicitamente com o uso de EN e ENO, NÃO podem ser representadas em texto estruturado. Não existe sintaxe para endereçar a saída ENO.

## Dificuldades na tradução ST-> FDB

Dificuldade de mapear construções:

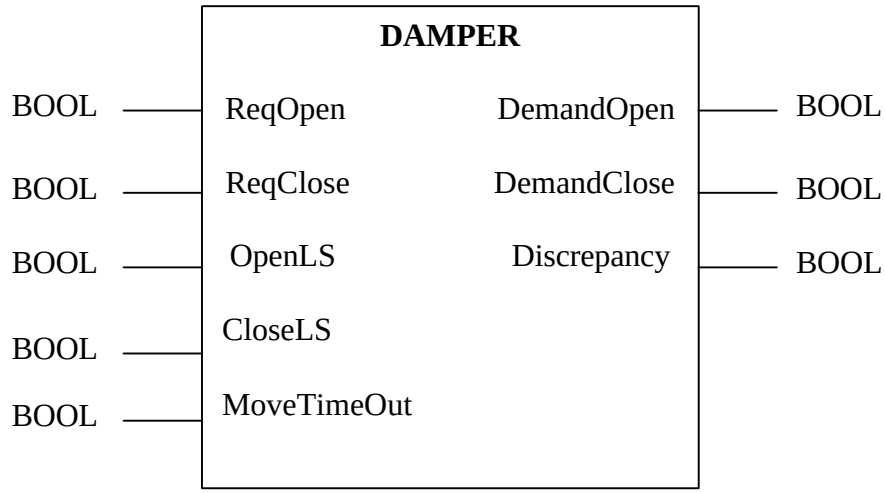
- **IF**..**THEN**
- **CASE**
- **FOR**
- **WHILE**
- **REPEAT**
- Acessar elementos em um array:

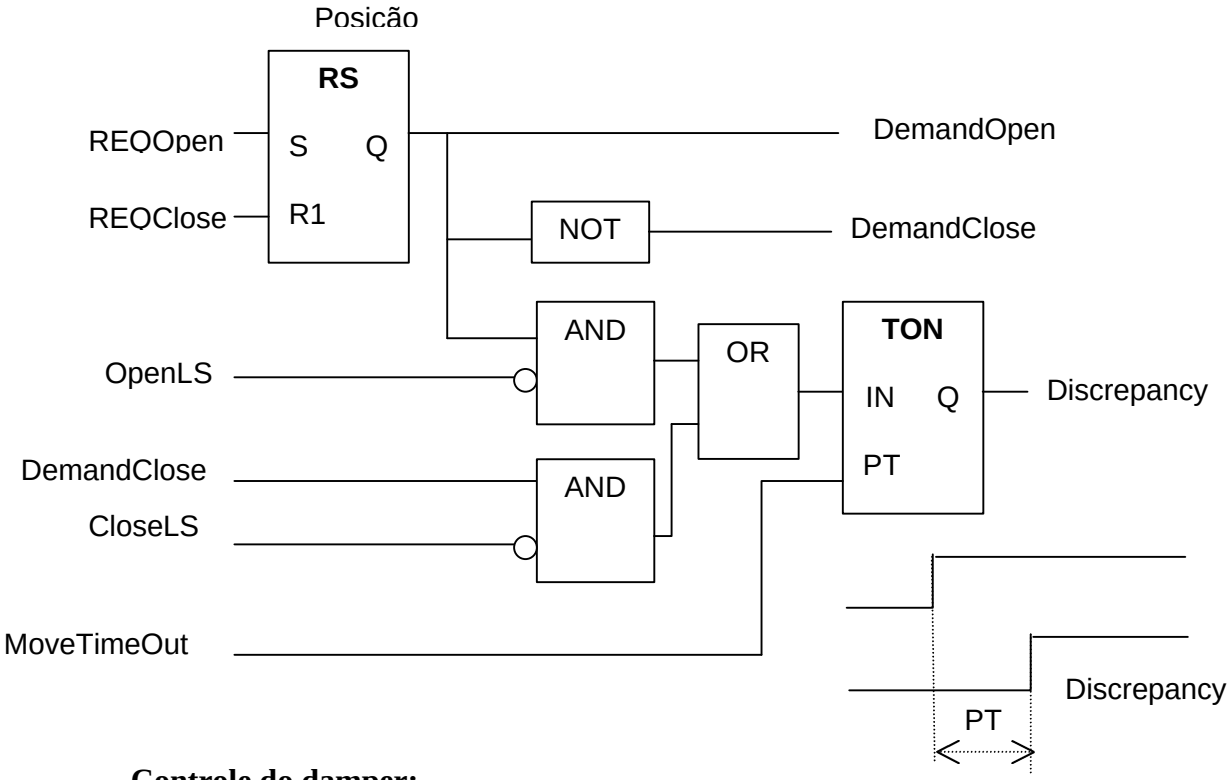
```
FOR I:= 1 TO 100 DO
        rate[I] := 100;
END_FOR;
```



Definido um bloco com FCBs.

Example

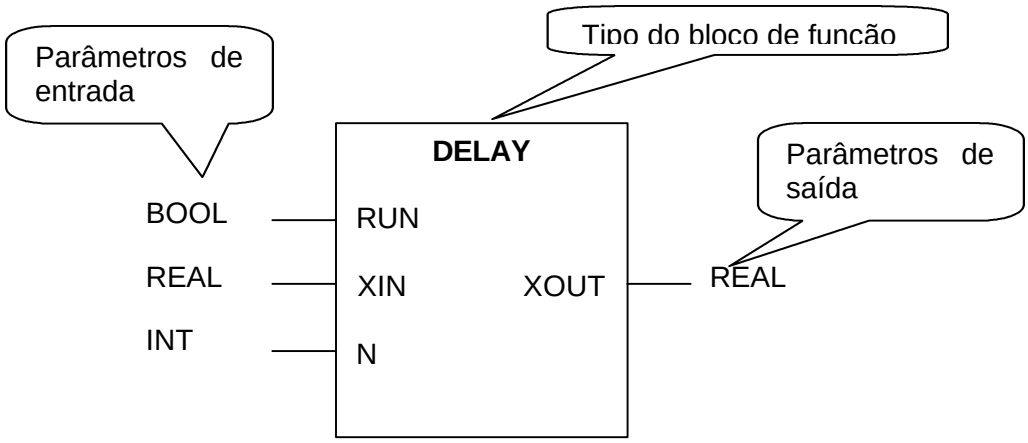



**Controle do damper:**

- O damper possui duas posições aberto e fechado. Os fins de curso OpenLS e Close LS denotam a posição real do damper.
- O biestável RS memoriza o pedido para abrir ou fechar.
- As saídas DemandClose e DemandOpen ligam os atuadores, que movem o damper para a posição desejada.
- Se após o comando, a posição final não é alcançada, um bit de erro é alimentado na entrada de um temporizador. Se após um tempo suficiente para o atuador funcionar, dado por MoveTimeOut o fim de curso não tiver sido atingido, o bit de discrepância é ativado.

Exemplo: Definição de um bloco de funções usando ST



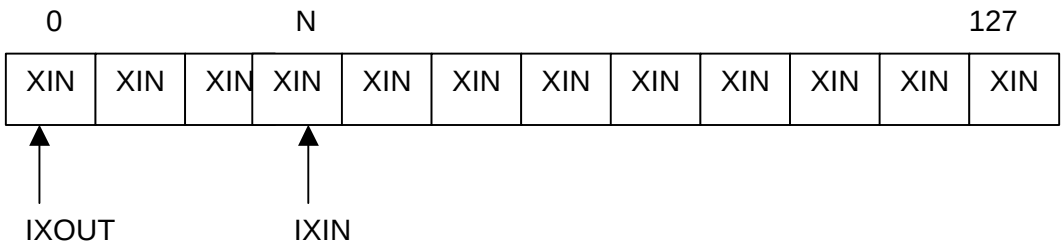

**Funcionamento:**

- DELAY estabelece um atraso de N amostras da entrada XIN para a saída XOUT.
- Será usado um buffer circular como memória intermediária.
- Cada vez que o bloco executar, um ponto será introduzido no bufffer.
- O valor de saída é obtido buscando uma amostra no buffer N posições atrás do ponto de entrada.
- atraso será proporcional ao tempo de scan.

```
FUNCTION_BLOCK DELAY
  (* Declaração de variáveis *)
  VAR_INPUT              (* Parâmetros de entrada *)
        RUN: BOOL;       (* Inicializar para 1 para executar o bloco,
                            i.e. ativar o atraso *)
        XIN: REAL;       (* Entrada a ser atrasad *)
        N: INT;          (* Largura da amostras de delay *)
  END_VAR

  VAR_OUTPUT             (* Parâmetros de saída *)
        XOUT: REAL;      (* Saída com atraso *)
  END_VAR

  VAR
        X: ARRAY[0..127] OF REAL;  (* Buffer circular *)
        I, IXIN, IXOUT: INT := 0;
  END_VAR

  (* Algoritmo *)
  IF RUN THEN
        (* Incrementa o índice da entrada *)
        (* Armazena o valor de entrada no buffer circular *)
        IXIN:= MOD(IXIN+1, 128);  X(IXIN) := XIN;
        (* Incrementa o índice da saída *)
```

```
            (* salva valor de saída no buffer circular *)
            IXOUT:= MOD(IXOUT+1, 128);  XOUT:=X(IXOUT);
    ELSE
            (* reseta valores dos índices *)
            IXIN := N; IXOUT := 0;
            (* saída = entrada *)
            XOUT := XIN;
            (* limpa buffer *)
            FOR I:+ 0 TO N DO
                        X[I] := XIN;
            END_FOR;
    END_IF;
END_FUNCTION_BLOCK
```

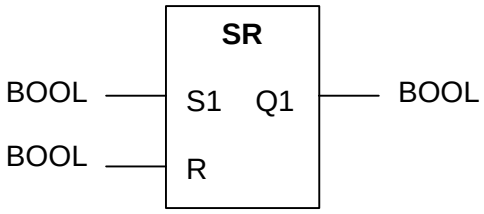
O vetor X armazena o último valor da entrada antes de iniciar a execução.
Este valor aparecerá na saída até que novos valores, N casas à frente, estejam disponíveis.



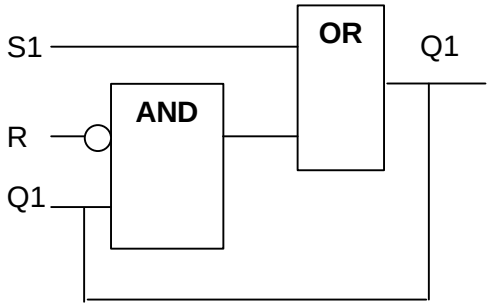
# Blocos de função padrões

Biestáveis:

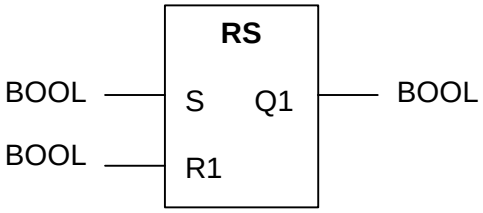
## BIESTÁVEL SR



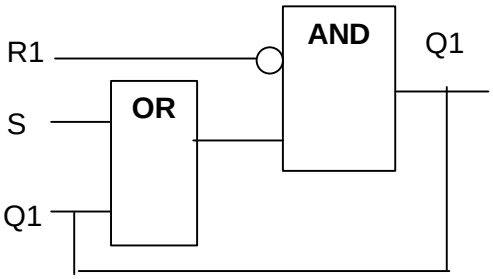

A entrada set é a dominante.
Se S1 = R = 1 então Q1=_____



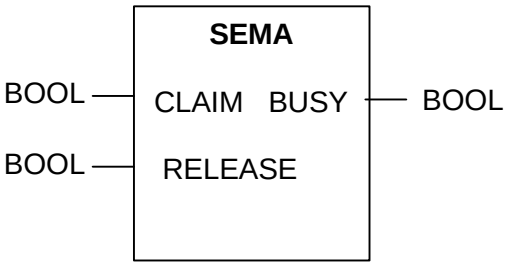

## BIESTÁVEL RS




A entrada reset é a dominante.
Se S1 = R = 1 então Q1=_____




## SEMÄFORO



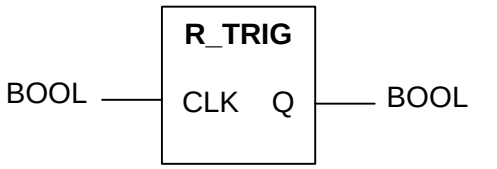

```
FUNCTION_BLOCK SEMA
    VAR_INPUT
        CLAIM, RELEASE: BOOL;
    END_VAR
    VAR_OUTPUT
        BUSY:BOOL;
    END_VAR
    VAR
        X: BOOL := 0;
    END_VAR
    BUSY := X;
    IF CLAIM THEN
        X:= 1;
    ELSEIF RELEASE THEN
        BUSY := 0; X := 0;
    END_IF
END_FUNCTION_BLOCK
```

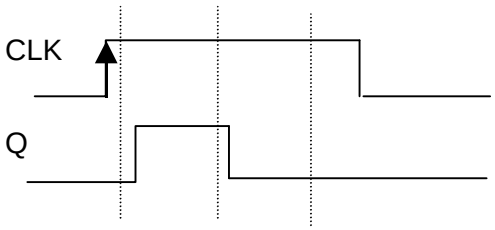
Explique o funcionamento do semáforo:

a) O que acontece se CLAIM é **TRUE** e X = 0 ?
b) O que acontece se CLAIM é **TRUE** e X = 1 ?
c) O que acontece se RELEASE é **TRUE** e X = 1 ?
d) Para que serve um semáforo ?
e) O semáforo necessita ser declarado como global ?

Detetor de bordas

## DETETOR DE BORDA DE SUBIDA

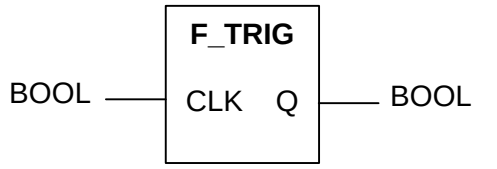


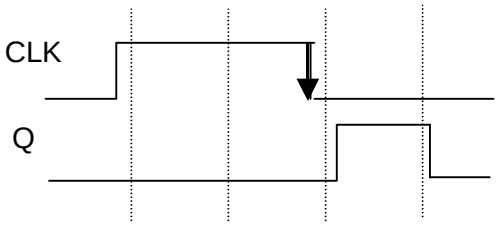




A saída permanece em 1 durante um scan

```
FUNCTION_BLOCK R_TRIG
    VAR_INPUT
        CLK: BOOL;
    END_VAR
    VAR_OUTPUT
        Q: BOOL;
    END_VAR
    VAR
        M: BOOL := 0;
    END_VAR
    Q := CLK AND NOT M;
    M := CLK;
END_FUNCTION_BLOCK
```

## DETETOR DE BORDA DE DESCIDA



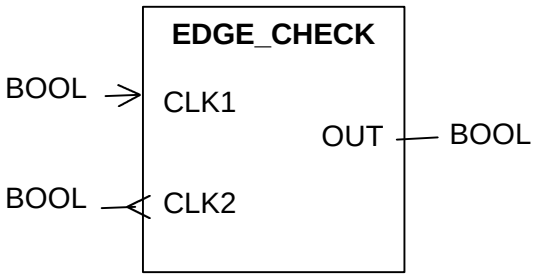

A saída permanece em 1 durante um scan

```
FUNCTION_BLOCK F_TRIG
    VAR_INPUT
        CLK: BOOL;
    END_VAR
    VAR_OUTPUT
        Q: BOOL;
    END_VAR
    VAR
        M: BOOL := 1;
    END_VAR
    Q := NOT CLK AND NOT M;
    M := NOT CLK;
END_FUNCTION_BLOCK
```

**OBS:** Estes blocos podem detectar bordas durante a ocorrência ou recuperação de falha de alimentação, dependendo da implementação e se o valor de M é mantido em memória retentiva.

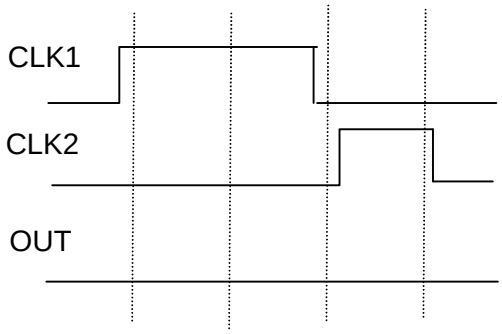
## ATRIBUTO SENSÍVEL À BORDA

R_EDGE e F_EDGE podem ser usados como atributos de variável de entrada booleana em blocos de função ou programas. Um bloco de função apropriado será então associado à entrada.



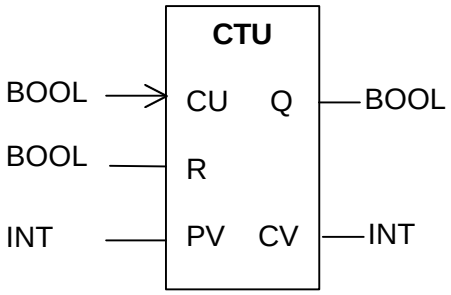

```
FUNCTION_BLOCK EDGE_TRIG
    VAR_INPUT
        CLK1: BOOL R_EDGE;
        CLK2: BOOL F_EDGE;
    END_VAR
    VAR_OUTPUT
        OUT: BOOL;
    END_VAR

    OUT := CLK1 OR CLK2;
END_FUNCTION_BLOCK
```

Desenhe a forma de onda gerada por OUT:



C o n t a d o r e s :

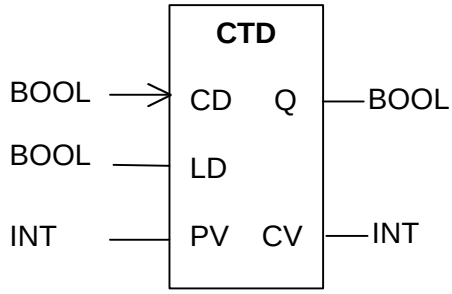
## CONTADOR INCREMENTAL




```
FUNCTION_BLOCK CTU
   VAR_INPUT
        CU : BOOL R_TRIG;
        R  : BOOL;
        PV  : INT;
   END_VAR
   VAR_OUTPUT
        Q : BOOL;
        CV: INT;
   END_VAR
   IF R THEN
        CV := 0;
   ELSIF CU AND (CV < PV) THEN
        CV := CV + 1;
   END_IF;
   Q := (CV >= PV);
END_FUNCTION_BLOCK
```

**Operação:** Este contador conta pulsos positivos em CU até que um valor máximo PV seja atingido. Neste instante o valor de contagem é congelado e a saída Q é ativada.
R reseta o contador e faz Q := 0;

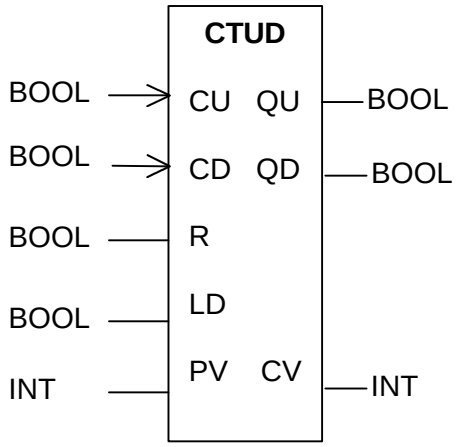
## CONTADOR DECREMENTAL




```
FUNCTION_BLOCK CTD
   VAR_INPUT
        CD : BOOL R_TRIG;
        LD  : BOOL;
        PV  : INT;
   END_VAR
   VAR_OUTPUT
        Q : BOOL;
        CV: INT;
   END_VAR
   IF LD THEN
        CV := PV;
   ELSIF CD AND (CV > PV) THEN
        CV := CV - 1;
   END_IF;
   Q := (CV <= 0);
END_FUNCTION_BLOCK
```

**Operação:** Este contador decrementa um contador a cada pulso positivos em CD até atingir o valor 0. Aí a contagem é interrompida e Q vai para **TRUE**. A entrada LOAD (LD) é usada para carregar o valor inicial CV do contador e fazer Q igual a 0.

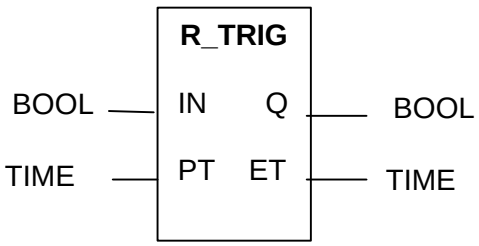
## CONTADOR INCREMENTAL/DECREMENTAL

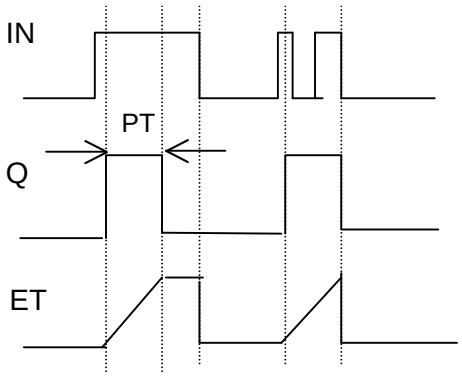



```
FUNCTION_BLOCK CTUD
   VAR_INPUT
      CU, CD : BOOL R_TRIG;
      R, LD  : BOOL;
      PV  : INT;
   END_VAR
   VAR_OUTPUT
      QU, QD : BOOL;
      CV: INT;
   END_VAR
   IF R THEN
      CV := 0;
   ELSIF LD THEN
      CV := PV;
   ELSIF CU AND (CV< PV) THEN
      CV := CV +1;
   ELSIF CD AND (CV > 0) THEN
      CV := CV - 1;
   END_IF;
   QU := (CV >= PV);
   QD := (CV <= 0);
END_FUNCTION_BLOCK
```

**Operação:** Incrementa o contador a cada pulso em CU e decrementa a cada pulso em CD. R reseta o contador. LD carrega um valor inicial. PV é o valor inicial de contagem. QU sinaliza que o valor máximo foi atingido. QD indica que o contador chegou a 0.

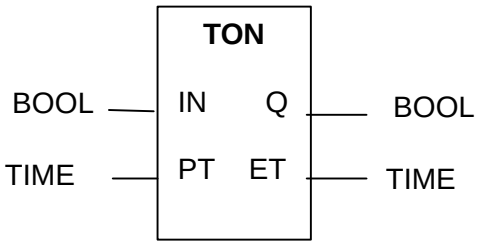
## TEMPORIZADOR DE PULSO

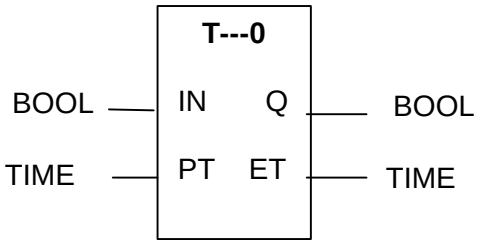


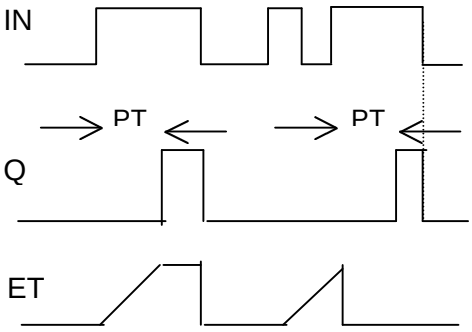




**OBS:** Quando a entrada IN vai para 1, a saída Q vai para 1 e permanece neste estado durante o tempo PT, independente do estado de IN. A saída ET será incrementada. Enquanto Q é **TRUE**, o tempo decorrido ET será incrementado. Ao final do pulso, o tempo decorrido será mantido até o início do próximo pulso.

## TEMPORIZADOR DE ATRASO NA SUBIDA

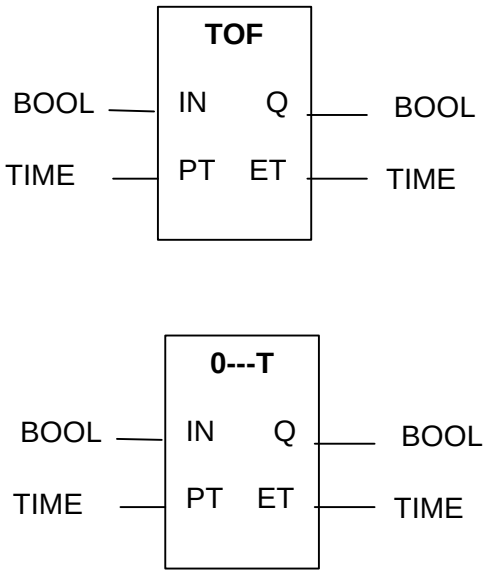


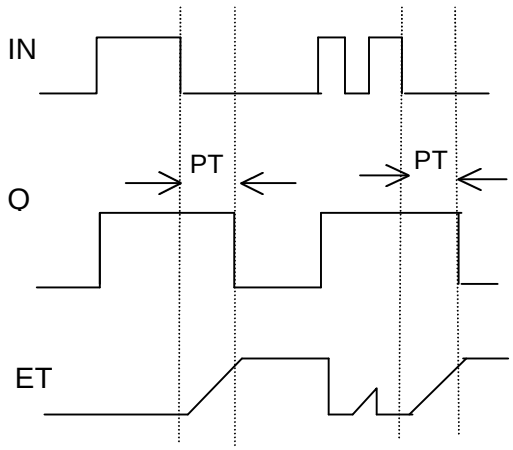




**OBS:** Após o início do pulso, a saída ficará em 0 durante o tempo especificado por PT e depois irá para 1 até que IN volte a zero. Durante a temporização, a saída ET refletirá o valor do tempo decorrido.

## TEMPORIZADOR DE ATRASO NA DESCIDA



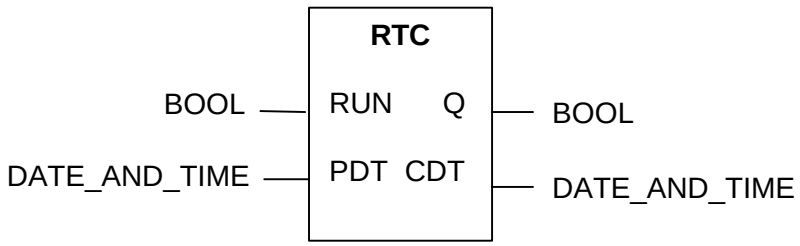




**OBS:** A saída Q é mantida em 1 durante o tempo PT após IN voltar para 0. Durante a temporização, a saída ET refletirá o valor do tempo decorrido.

Exercício:

Explique como este temporizador pode ser utilizado para eliminar ruído na forma de *glitches* da entrada.

## REAL TIME CLOCK




**Operação:**

Inicialmente o valor inicial do real time clock deve ser feito igual à data e hora iniciais, com RUN igual a 1, sendo que:

a) O bloco está sendo executado pela primeira vez.
b) O bloco havia sido executado logo antes com RUN = 0.

Após isso o bloco deve ser chamado com RUN = 1 e apresentará na saída CDT o valor da data e hora correntes.

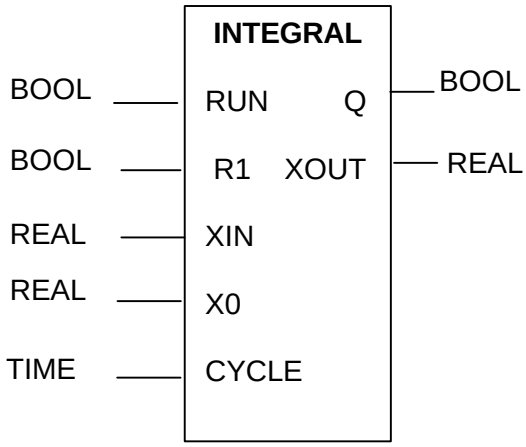
Este bloco deve ser declarado como uma variável global.

Em geral apenas um bloco é definido a menos que se deseje trabalhar com diversos fuso horários.

**Exemplo:**

```
CONFIGURATION SYTEM_ABC
VAR_GLOBAL
        RTC_Clock1: RTC;
END_VAR
PROGRAM PROG1
VAR_EXTERNAL
        RTC_CLOCK1 : RTC;
END_VAR
***
IF SetClck = True THEN
        (* Inicializa relógio *)
      RTC_Clock1(RUN := 0);
      RTC_Clock1(RUN : =1, PDT := InputTime);
ELSE
      (* Atualiza data hora correntes *)
      RTC_Clock1(RUN := 1);
      CurrentTime:= RTC_Clock1.CDT;
***
```

## INTEGRAL




```
FUNCTION_BLOCK INTEGRAL
   VAR_INPUT
        RUN: BOOL;          (* Integra: 1, Mantém: 0)
        R1: BOOL;           (* Reset *)
        XIN: REAL;          (* Entrada *)
        X0:REAL;            (*Valor Inicial *)
        CYCLE: TIME;        (* Tempo de ciclo *)
   END_VAR
   VAR_OUTPUT
        Q: BOOL;            (* 1 = Não reset *)
        XOUT: REAL;         (* Valor integrado *)
   END_VAR
   Q:= NOT R1;
   IF R1 THEN
        XOUT := X0;
   ELSIF RUN THEN
        XOUT := XOUT + XIN * TIME_TO_REAL(CYCLE);
   END_IF;
END_FUNCTION_BLOCK
```

**Operação:**

Integra o valor da entrada XIN no tempo enquanto RUN = **TRUE**, caso contrário, mantém o valor da saída.
valor de integração pode ser igualado a um valor X0 definindo-se a entrada R1 = true.
CYCLE define o tempo entre execuções.

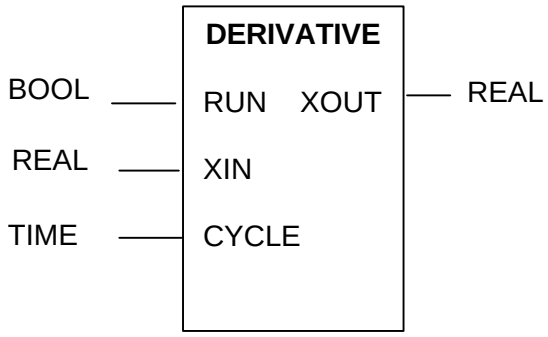
- Q = 1 se o bloco não estiver em reset.

**Utilização:**

- Algoritmos PID.
- Para calcular energia devemos integrar a _____.
- Para calcular o volume de um tanque devemos integrar a _____.

D e r i v a d a

## DERIVADA



```
FUNCTION_BLOCK DERIVATIVE
   VAR_INPUT
        RUN: BOOL;          (* 0 = Reset *)
        XIN: REAL;          (* Entrada *)
        CYCLE: TIME;        (* Tempo de ciclo *)
   END_VAR
   VAR_OUTPUT
        XOUT: REAL;         (* Valor da derivada *)
   END_VAR
   VAR
        X1, X2, X3: REAL;
   END_VAR
   IF RUN THEN
        XOUT := (3.0 * (XIN - X3) + (X1 - X2) / (10.0 *
        TIME_TO_REAL(CYCLE));
        X3 := X2;  X2 := X1;
        X1 := XIN;
   ELSE
        (* Reset *)
        XOUT := 0.0; X1 := XIN;
        X2 := XIN; X3 := XIN;
   END_IF;
END_FUNCTION_BLOCK
```

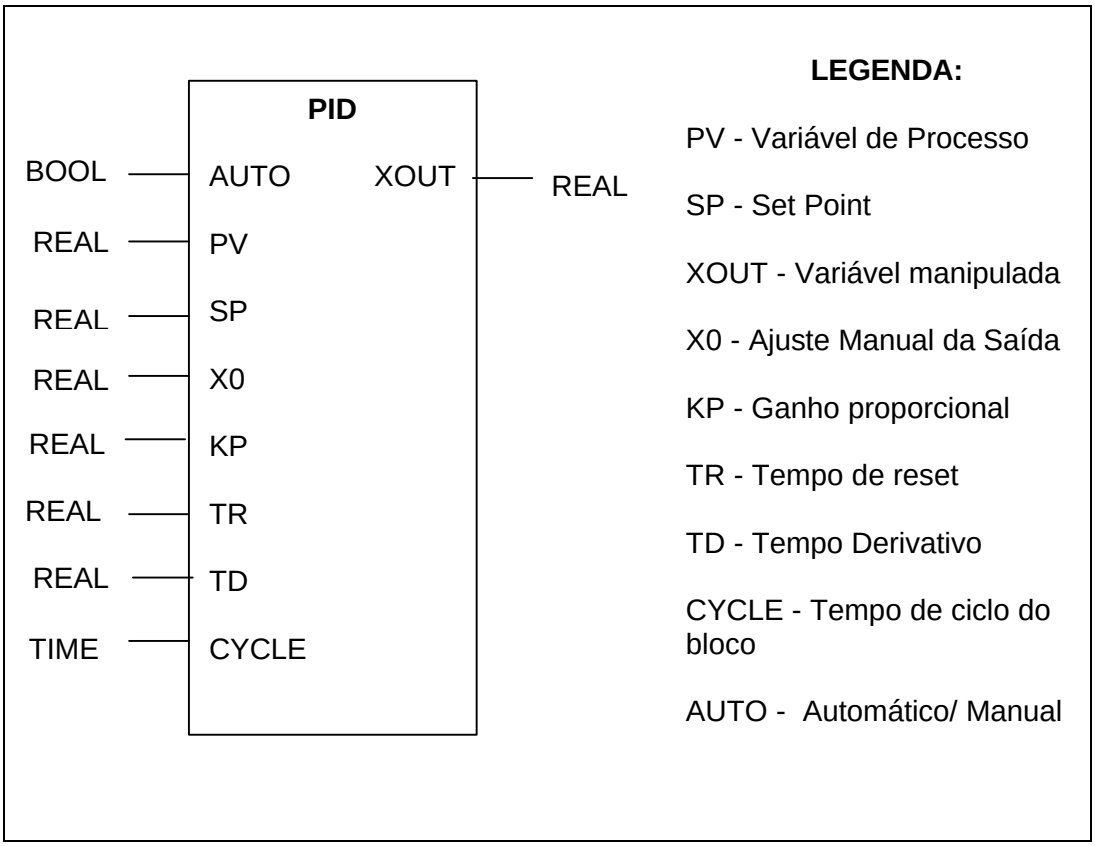
**Operação:**

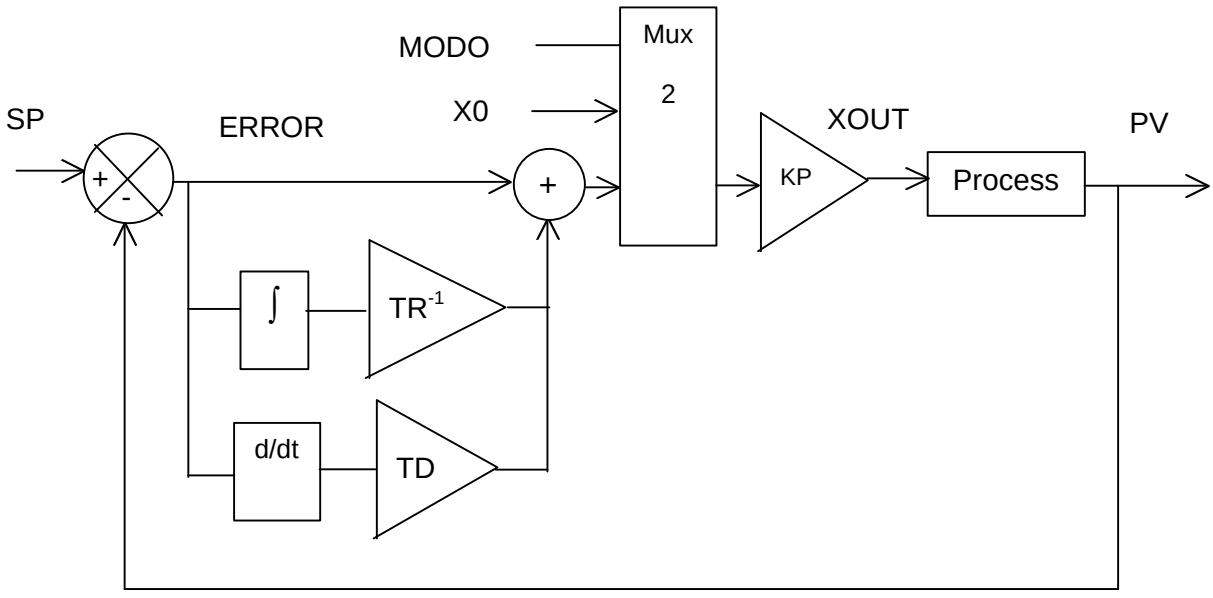
Produz saída XOUT equivalente à taxa de variação de XIN.
Se RUN = True: a derivada é calculada.
Se RUN = False, a derivada é zero.

**Utilização:**

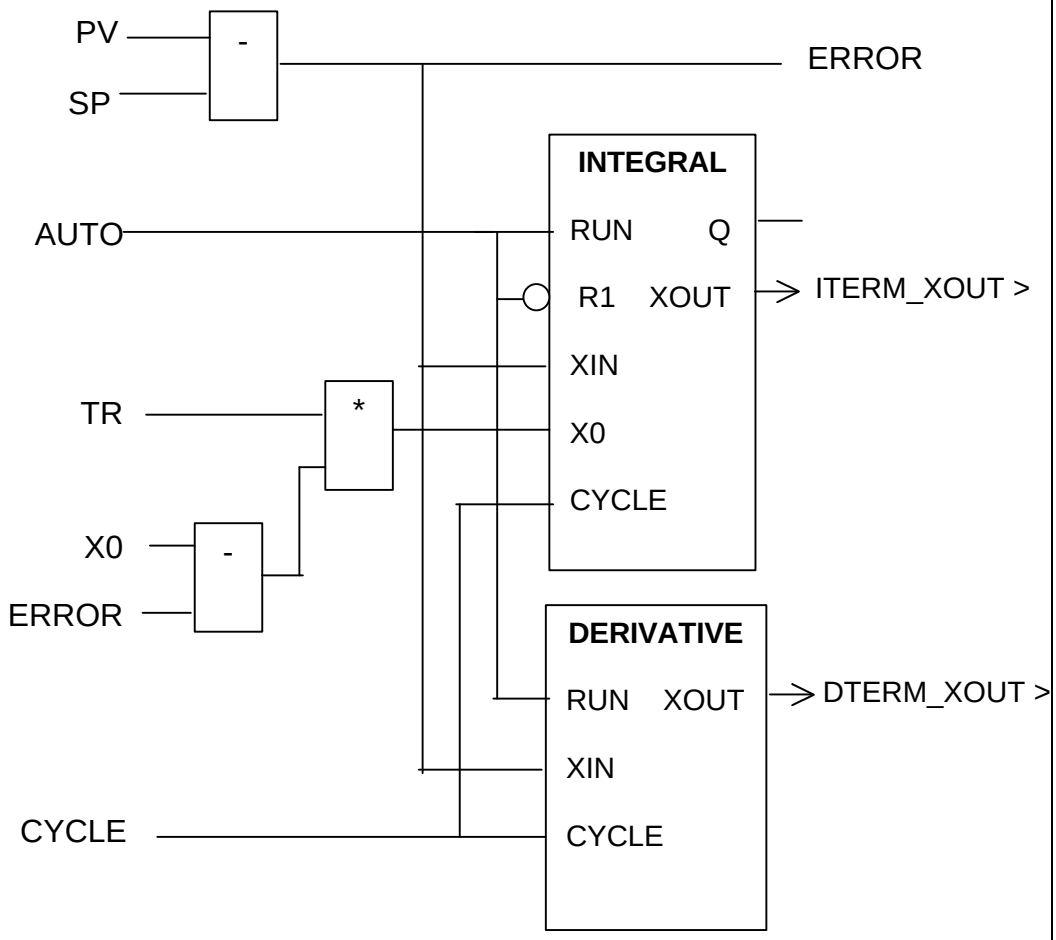
- Algoritmos PID.
- Cálculo de taxa de variação da posição para obter _____.
- Cálculo da taxa de variação de variáveis de processo para validação.

PID

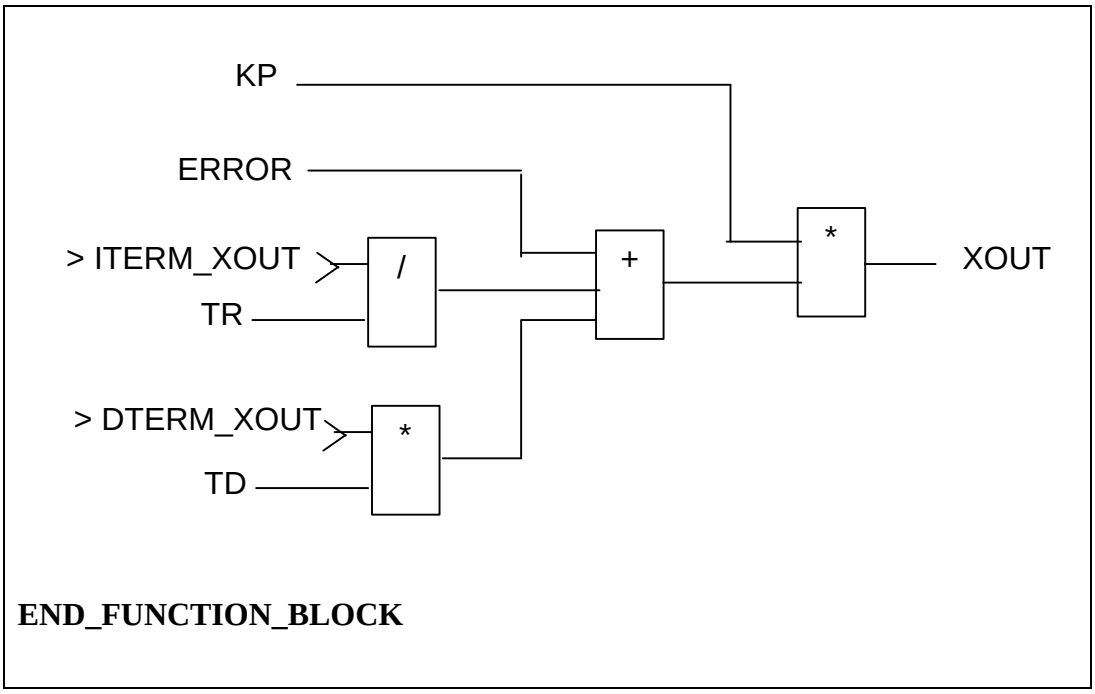

PID
BOOL
AUTO
XOUT
REAL
REAL
PV
REAL
SP
REAL
X0
REAL
KP
REAL
TR
REAL
TD
TIME
CYCLE
LEGENDA:
PV - Variável de Processo
SP - Set Point
XOUT - Variável manipulada
X0 - Ajuste Manual da Saída
KP - Ganho proporcional
TR - Tempo de reset
TD - Tempo Derivativo
CYCLE - Tempo de ciclo do bloco
AUTO - Automático/ Manual


MODO
Mux
2
X0
SP
ERROR
XOUT
PV
+
-
+
KP
Process
∫
TR-1
d/dt
TD

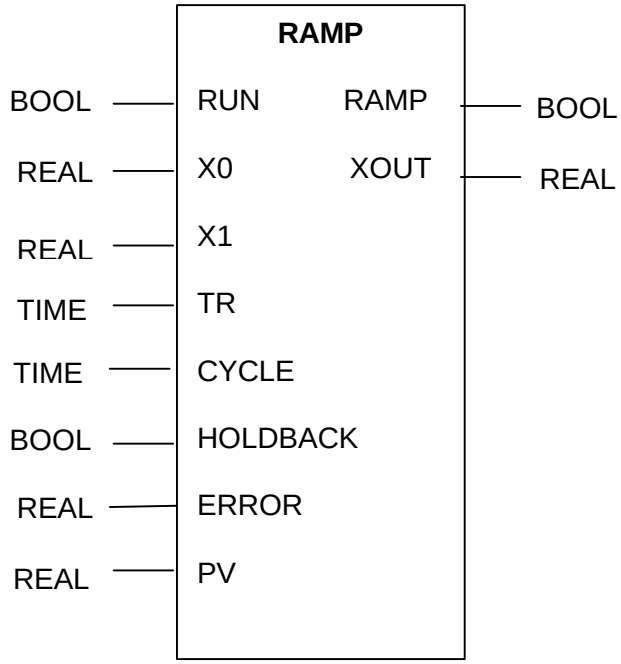
```
UNCTION_BLOCK PID
   VAR_INPUT
        AUTO: BOOL;        (* 0  =  Manual,  1 = Automático *)
        PV: REAL;          (* Variável de Processo *)
        SP: REAL;          (* Set Point *)
        X0: REAL;          (* Ajuste manual de saída *)
        KP: REAL;          (* Constante Proporcional *)
        TR: REAL;          (* Tempo de Reset *)
        TD: REAL;          (* Tempo derivativo *)
        CYCLE: TIME;       (* Tempo de ciclo do bloco *)
   END_VAR
   VAR_OUTPUT
        XOUT: REAL;        (* Valor da derivada *)
   END_VAR
   VAR
        ERROR: REAL;               (* Erro *)
        ITERM: INTEGRAL;           (* Componente Integral *)
        DTERM: DERIVATIVE;         (* Componente Derivativo *)
   END_VAR

   (* Algoritmo PID expresso em FBD *)
```



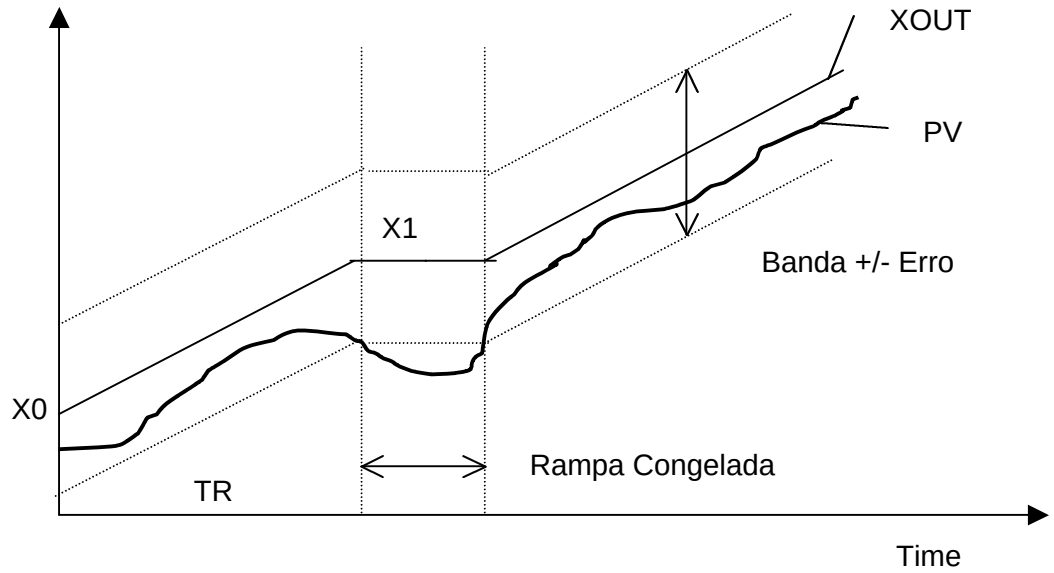

```
END_FUNCTION_BLOCK
```


KP
ERROR
> ITERM_XOUT
TR
/
+
*
XOUT
> DTERM_XOUT
TD
*
END_FUNCTION_BLOCK

## RAMPA




Saída do gerador de rampa



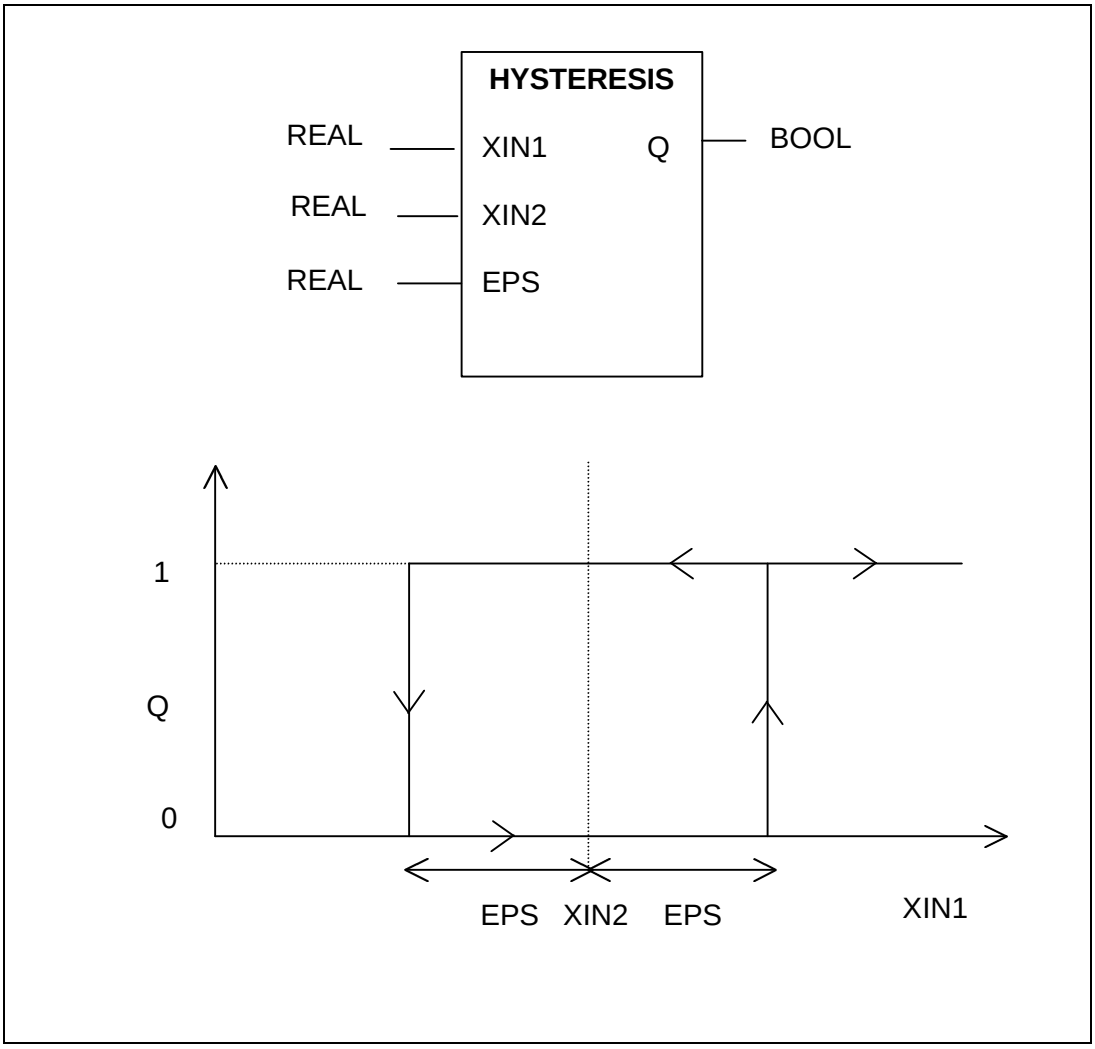
```
FUNCTION_BLOCK RAMP
   VAR_INPUT
        RUN: BOOL;              (* 0 = Reset *)
        X0: REAL;               (* Valor inicial *)
        X1: REAL;               (* Valor alvo *)
        TR: REAL;               (* Duração da rampa *)
        CYCLE: TIME;            (* Tempo de ciclo *)
        HOLDBACK: BOOL;         (* Mantém valor quando true *)
        ERROR: REAL;            (* Erro para Holdback *)
        PV: REAL;               (* Valor da variável medida *)
   END_VAR
   VAR_OUTPUT
        RAMP: BOOL;         (* 1 enquanto  gera      ndo rampa *)
        XOUT: REAL;         (* Valor da derivada *)
   END_VAR
   VAR
        XI: REAL;           (* Valor intermediário da rampa *)
        T: TIME := T#0s;    (* Tempo de geração da rampa *)
   END_VAR

   (* Algoritmo *)
   IF RUN THEN

        (* Mantém saída se ABS(PV - XOUT ) >  ERRO *)
        IF (HOLDBACK AND ABS(PV - XOUT) < ERROR)
        OR NOT HOLDBACK THEN
                XOUT := X0 + (X1 - X0) * TIME_TO_REAL(T) /
                TIME_TO_REAL(TR);
                T:= T + CYCLE;
                RAMP := TRUE;
        ELSE
                (* Não está gerando rampa devido a Holdback *)
                RAMP := FALSE;
                XOUT := X0; XI := X0; T := T#0s;
        END_IF;
END_FUNCTION_BLOCK
```

**Funcionamento:**

Gera uma rampa de referência em geral usada como *set point* de velocidade ou temperatura de uma malha de controle.
A saída XOUT produz uma rampa que parte de um valor inicial X0 e tende a um valor final X1 em um período de tempo estipulado TR.
A função rampa é ativada fazendo-se RUN = 1, caso contrário XOUT = 0.
Se a flag HOLDBACK estiver ativada, a função rampa irá ficar congelada enquanto o valor da PV estiver acima de um certo valor de erro.
Isto facilita a implementação de malhas de controle onde o processo não consegue seguir uma determinada rampa. Ao invés de deixar o erro crescer indefinidamente, o *set point* é mantido, até que a saída do processo consiga acompanha-la convenientemente.

H i s t e r e s e

## HISTERESE



```
FUNCTION_BLOCK HYSTERESIS
    VAR_INPUT
         XIN1, XIN2: REAL;
         EPS: REAL;          (* Banda de histerese *)
    END_VAR
    VAR_OUTPUT
         Q:  BOOL := 0;      (* Valor inicial é 0 *)
    END_VAR

     IF Q THEN

         IF XIN1 < (XIN2 - EPS) THEN
                Q : = 0;     (* XIN1 diminuindo *)
         END_IF
    ELSEIF NIN1 > (XIN2 + EPS) THEN
         Q := 1;             (* XIN1 aumentando *)
    END_IF
END_FUNCTION_BLOCK
```

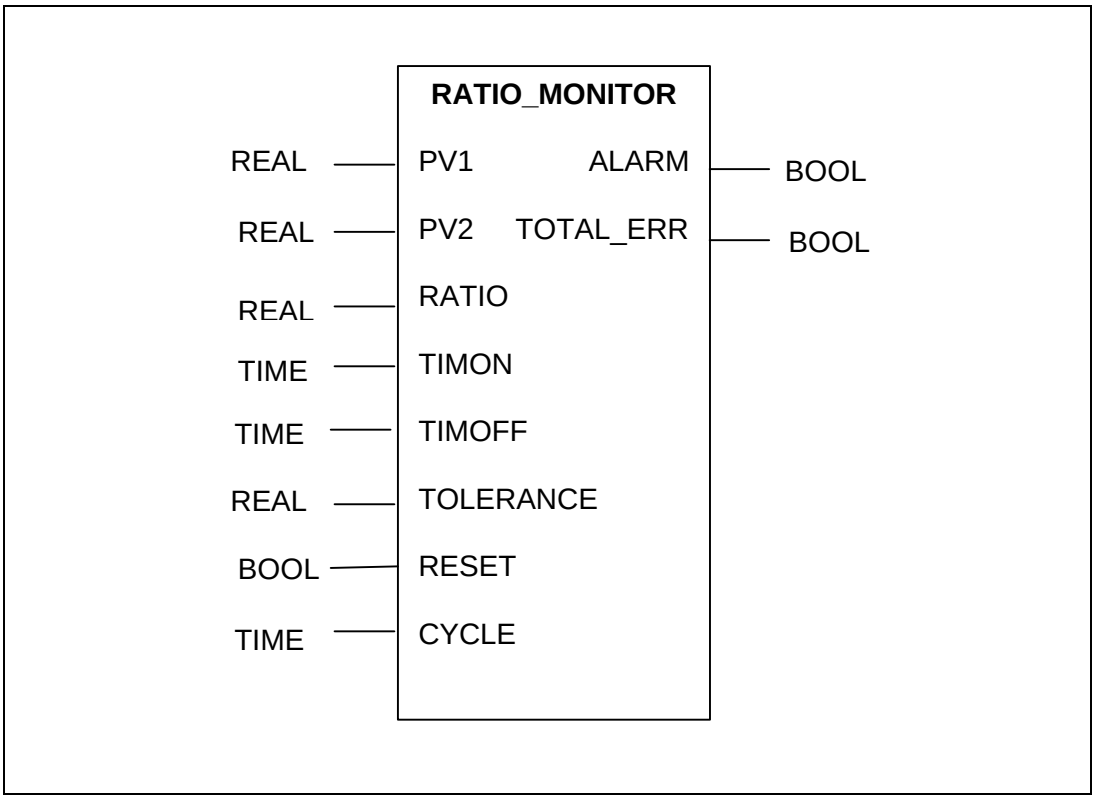
**Operação:**

Se a entrada XIN1 superar um nível dado XIN2 de um valor dado por EPS, a saída Q irá para 1. A saída só volta ao nível 0 se XIN1 se tornar inferior a XIN2 de um valor igual ou superior a EPS.

**Utilidades:**
Cite duas utilidades para o bloco de histerese.

a) _____
b) _____

## RATIO MONITOR




**Comentários:**

PV1 deve estar sempre a uma razão definida por RATIO de uma outra variável de processo PV2. Se o valor do erro superar uma tolerância dada, um temporizador de alarme é disparado. Se o alarme persistir por um tempo superior a TIMON, a saída ALARM vai para 1. Um segundo temporizador é disparado para garantir que ALARM ficará em 1 por pelo menos por um tempo TIMOFF.
O erro integrado no tempo é mostrado na saída TOTAL_ERR.

```
FUNCTION_BLOCK RATIO_MONITOR
   VAR_INPUT
        PV1, PV2: REAL;          (* Variáveis de processo *)
        RATIO: REAL;             (* Razão entre PV1 e PV2 *)
        TIMON: TIME;             (* Tempo antes de Alarme ser ativado *)
        TIMOFF: TIME;            (* Tempo que alarme permanece ativo *)
        TOLERANCE: REAL;         (* Erro absoluto aceitável *)
        RESET: BOOL;             (* 1 zera erro total *)
        CYCLE: TIME;             (* Tempo de scan do bloco *)
   END_VAR
   VAR_OUTPUT
        ALARM: BOOL := 0;        (* Erro de relação em alarme *)
        TOTAL_ERR:  REAL := 0;   (* Erro integrado *)
   END_VAR

   VAR
        ErrorInt: INTEGRAL;      (* Integral do erro *)
        TimerOn: TON;            (* Temporização de alarme em ON *)
        TimerOff: TOF;           (* Temporização de alarme em OFF *)
        Erro: REAL;
   END_VAR

   (* Verifica erro entre variáveis de processo *)
   Error := ABS(PV1 - RATIO * PV2);

   IF RESET THEN
        (* zera integrador *)
        ErrorInt(R1:= 1; X0:= 0.0);
   ELSE
        (* Integra o erro *)
        ErrorInt(RUN:=1, R1 := 0, XIN:= Error, CYCLE:=CYCLE);
        TOTAL_ERR := ErrorInt.XOUT;
   END_IF;
   (* chama temporizadores de alarme se alarmes ativos *)
   TimerOn(EN:=Error > TOLERANCE, PT:= TIMON);
   TimerOff(EN:=TimerOn.Q, PT:= TIMOFF);
   ALARM:= TimerOff.Q;
END_FUNCTION_BLOCK
```

## Leitura Complementar:


- Bonfatti, Monari, Sampieri, IEC1131-3 Programming Methodology, CJ International, 1997.

## Exercícios:

1. Escreva um bloco de função para sintetizar a função ou exclusivo de três entradas.

2. Escreva um *bloco de função* para sintetizar um sensor de rotação baseado em dois trens de pulso de entrada, gerados por sensores ópticos ligados a um codificador de rotação. O número de dentes do codificador (N) constitui uma entrada do bloco de função. O primeiro trem está defasado do segundo de 45°. O bloco deve indicar o sentido de rotação e a velocidade de rotação em revoluções/s.

3. Desenhe um *deboucer* de uma entrada de teclado, utilizando um bloco de histerese.

4. Enumere as vantagens de se utilizar blocos de função:
   a)
   b)
   c)
   d)

5. Desenhe o FCB para o comando de partida de um motor reversível:
   Um motor elétrico pode ser acionado nos dois sentidos com reversão manual no sentido de rotação. O motor não deve entrar em funcionamento ou não deve ser desligado, se qualquer uma das condições estiver presente:
   a) Se a chave de emergência for acionada.
   b) Se a pressão de óleo nos mancais for igual a zero.
   c) Se a temperatura da carcaça do motor for superior a 70°C.

| Entradas | Saídas |
|---|---|
| Emergência | MotorDireita |
| Pressão do óleo | MotorEsquerda |
| Temperatura de carcaça | |
| Desliga | |
| LigaDireita | |
| LigaEsquerda | |

   Note que:
   - O temporizador com retardo no desligamento deve garantir que a saída ficará energizada durante o tempo em que a entrada se encontra energizada e ainda durante um retardo de tempo t2.
   - O ligamento / desligamento do motor se dá via memórias biestáveis.
   - Parar evitar que o motor que se encontra em um sentido de rotação seja ligado em sentido contrário, deve ser incluído um intertravamento no circuito.
   - O desligamento deve ser único.