Técnicas de la automatización
(Cód. 201987)

4. PLC-II: Programación con Texto Estructurado (ST) y con Diagramas de Bloques de Función (FBD)

Escuela Politécnica Superior
UNIVERSIDAD DE ALCALÁ

Índice

Estructuras de control (i)

- ST es un lenguaje de alto nivel similar Pascal o Ada. [1]
- Un programa ST se compone de una secuencia de estructuras de control separadas por el carácter `;`.

Estructuras de control:

`:=` (asignación) Asigna el valor de una expresión a una variable:

```
1 K3 := (K2 AND KT3) OR (K3 AND NOT S1);
```

`IF` (selección) Selección entre alternativas por medio de expresiones Boolean:

```
1 IF d<e THEN f := 1;
2 ELSIF d=e THEN f := 2;
3 ELSE f := 3;
4 END_IF;
```

[1] [IEC, 2006, pp. 129–134] y [John, 2010, pp. 116–133].

Estructuras de control (ii)

CASE (selección por casos) Selección entre alternativas por medio de una expresión:

```
1 CASE expr OF
2         1: g:=11;
3       2,3: g:=12;
4     5..10: g:=13;
5      ELSE: g:=-1;
6 END_CASE;
```

FOR Bucle con inicialización, condición para continuar y progresión:

```
1 VAR
2    V : ARRAY[1..5] OF INT :=
3          [2, 16, 4, 7, 32];
4    I : INT;
5    nV : INT:=5;
6    Max: INT:=0;
7 END_VAR;
```

```
1 FOR I:=1 TO nV BY 1 DO
2    IF V[I] > Max THEN
3       Max:=V[I];
4    END_IF;
5 END_FOR;
```

Estructuras de control (iii)

WHILE Bucle con condición para continuar:

```
1 I:=1; (* Inicialización. *)
2 WHILE I<=nV DO (* Condición. *)
3    IF V[I] > Max THEN
4       Max:=V[I];
5    END_IF;
6    I:=I+1; (* Progresión. *)
7 END_WHILE;
```

REPEAT Bucle con condición para finalizar:

```
1 I:=1; (* Inicialización. *)
2 REPEAT
3    IF V[I] > Max THEN
4       Max:=V[I];
5    END_IF;
6    I:=I+1; (* Progresión. *)
7 UNTIL I>nV END_REPEAT; (* Condición. *)
```

Estructuras de control (iv) y operadores

`EXIT` Finaliza el bucle donde se encuentra.

`RETURN` Abandona el POU actual.

Operadores

- Aritméticos: `-`(unario), `**` [a], `*`, `/`, `MOD`, `+`, `-`.
- De comparación: `<`, `>`, `<=`, `>=`, `=`, `<>`.
- Lógicos: `NOT`, `AND`, `&` [b], `OR`, `XOR`.
- Paréntesis: `(`, `)`. Modifica la prioridad de los operadores anteriores.

[a] Potencia `a**b` $\equiv$ a^b.

[b] `&`$\equiv$`AND`.

Tipos de datos en ST

Tipo	Descripción	Nro. de bits	Rango
BOOL	Bit	1	0, 1
BYTE	Byte	8	0..#FF
WORD	Cadena	16	#0..#FFFF
DWORD	Cadena doble	32	#0..#FFFFFFFF
LWORD	Cadena larga	64	#0..#FFFFFFFFFFFFFFFF
INT	Entero	16	$-32568..32567$
UINT	Entero sin signo	16	0..65535
DINT	Entero doble	32	$-2^{31}..2^{31}-1$
UDINT	Entero doble sin signo	32	$0..2^{32}-1$
LINT	Entero largo	64	$2^{63}..2^{63}-1$
ULINT	Entero largo sin signo	64	$0..2^{64}-1$
REAL	Número real	32	$\pm 10^{-38}.. \pm 10^{38}$
LREAL	Número real largo	64	$\pm 10^{-308}.. \pm 10^{308}$
STRING	Cadena de caracteres		de 1 a 125 caracteres

Algunas funciones disponibles en ST

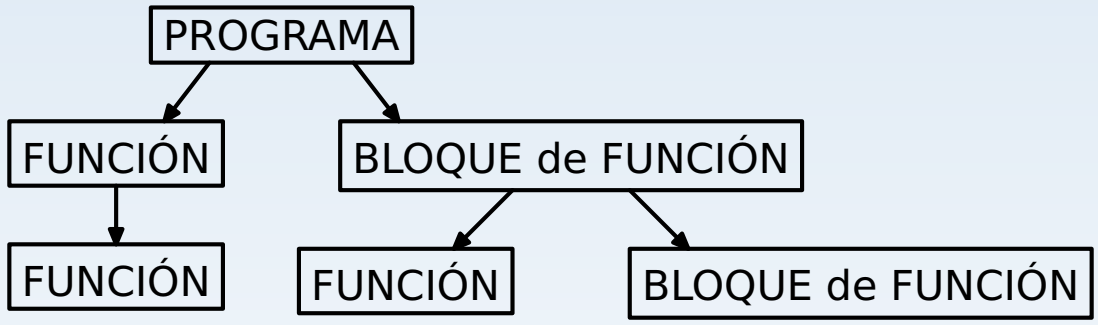
Función	Nombre	Operandos	Resultado
`y:=ABS(x)`	Valor absoluto, $y = \|x\|$	$\mathbf{Z}$ o $\mathbf{R}$	$\mathbf{Z}$ o $\mathbf{R}$
`y:=SQRT(x)`	Raíz cuadrada, $y = \sqrt{x}$	$\mathbf{R}$	$\mathbf{R}$
`y:=LN(x)`	Logaritmo natural	$\mathbf{R}$	$\mathbf{R}$
`y:=LOG(x)`	Logaritmo decimal	$\mathbf{R}$	$\mathbf{R}$
`y:=EXP(x)`	Exponencial, $y = e^x$	$\mathbf{R}$	$\mathbf{R}$
`y:=EXPT(x,n)`	Potencia, $y = x^n$	$x \in \mathbf{R}, n \in \mathbf{Z}$	$\mathbf{R}$
`y:=SIN(x)`	Seno	$\mathbf{R}$	$\mathbf{R}$
`y:=COS(x)`	Coseno	$\mathbf{R}$	$\mathbf{R}$
`y:=TAN(x)`	Tangente	$\mathbf{R}$	$\mathbf{R}$
`y:=ASIN(x)`	Arco seno	$\mathbf{R}$	$\mathbf{R}$
`y:=ACOS(x)`	Arco coseno	$\mathbf{R}$	$\mathbf{R}$
`y:=ATAN(x)`	Arco tangente	$\mathbf{R}$	$\mathbf{R}$

Índice

Llamadas entre POUs

Ya hemos visto que la parte de código de un programa se compone de:

1. secuencias de instrucciones,
2. llamadas a funciones y
3. llamadas a bloques de función.

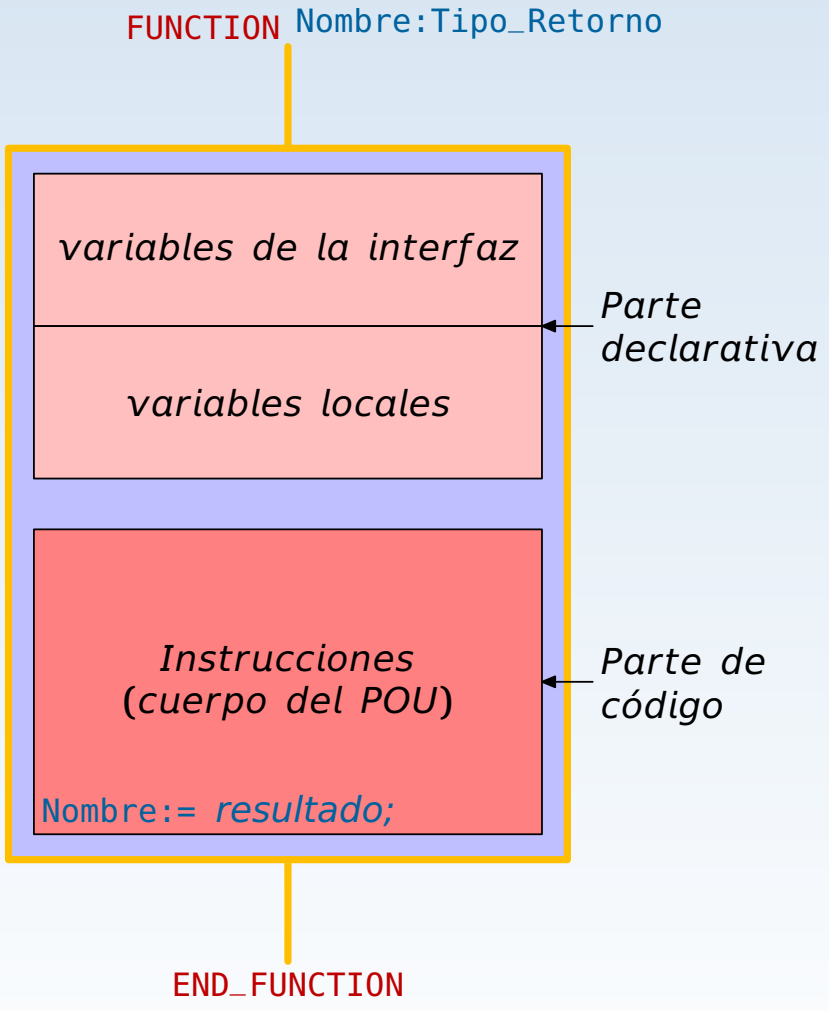


1. Un programa puede llamar a funciones y a bloques de funciones.
2. Un bloque de función puede llamar a funciones y bloque de funciones.
3. Una función solo puede llamar a funciones.
4. Ningún POU puede llamarse de forma recursiva.

Partes de una función

Función

1. Secuencia de instrucciones y llamadas a otras funciones que
2. operan sobre unos datos de entrada para producir una salida.
3. Carece de memoria o estado interno.



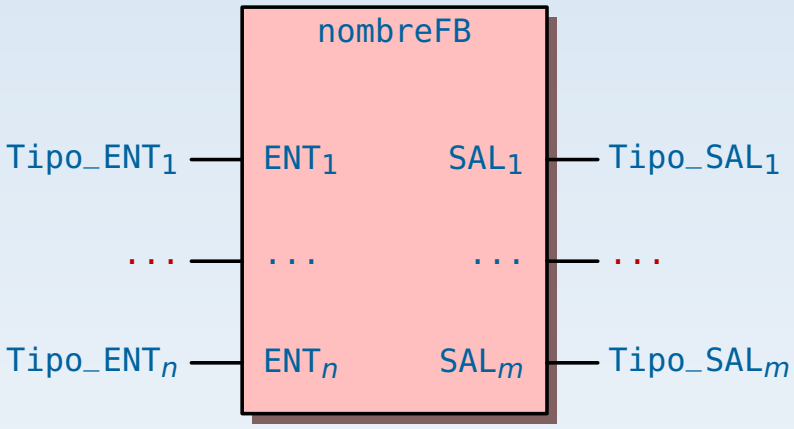
Índice

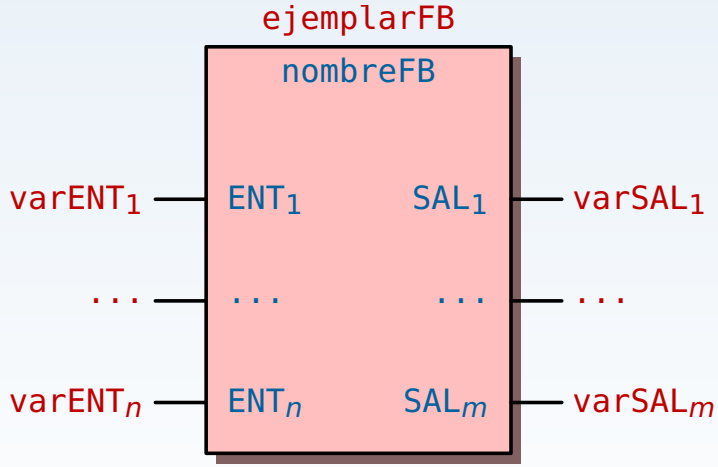
Diagramas de bloques de función (FBD)

1. FBD es un lenguaje gráfico que trabaja con conexiones de funciones y de bloques de función. [2]
2. Los programas dibujados con FBD se asemejan a los diagramas con circuitos integrados.
3. Un bloque de función es un POU que consta de un conjunto de datos encapsulados e independientes y los algoritmos para trabajar con esos datos.
4. A diferencia de las funciones, los bloques de función pueden tener en estado interno asociado.
5. Para utilizar un bloque de función es necesario crear un ejemplar (*instance*) derivado.
6. Los bloques de función se pueden emplear en programas escritos en lenguaje IL, LD, ST y FBD.

[2] [IEC, 2006, p. 143] y [John, 2010, pp. 134–147].

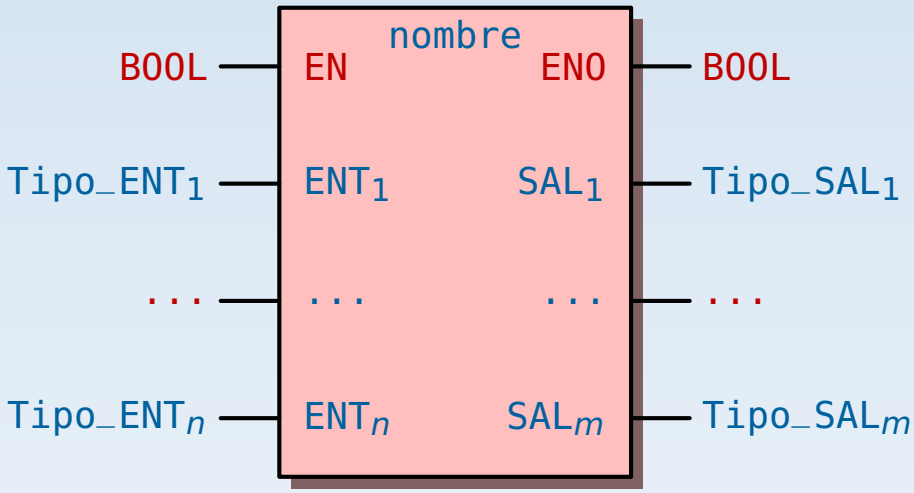
Declaración y uso de un bloque de función






```
1  VAR
2     ejemplarFB: nombreFB;
3     varENT1: Tipo_ENT1;
4     ...
5     varENTn: Tipo_ENTn;
6     varSAL1: Tipo_SAL1;
7     ...
8     varSALm: Tipo_SALm;
9  END_VAR
10 ...
11 ejemplarFB(ENT1:=varENT1,...,
12            ENTn:=varENTn);
13 ...
14 varSAL1 := nombreFB.SAL1;
15 ...
16 varSALm := nombreFB.SALm;
17 ...
18 ejemplarFB(ENT1:=varENT1,...,
19            SAL1=>varSAL1,...);
```

Entrada EN y salida ENO



Tanto EN (*enable input*) como ENO (*enable output*) son optativas.

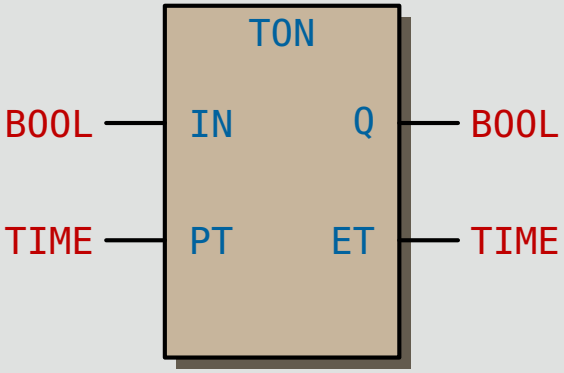
EN	Significado	ENO
FALSE	No se ejecuta el código del bloque al llamarlo	ENO:=FALSE (no se actualizan las salidas)
TRUE	Se ejecuta el código del bloque	ENO:=TRUE (se actualizan las salidas)
	En el bloque se puede fijar el valor de ENO	ENO:=TRUE o FALSE
	Cuando se detecta algún fallo	ENO:=FALSE

Índice

Temporizadores IEC 61131-3 – TON

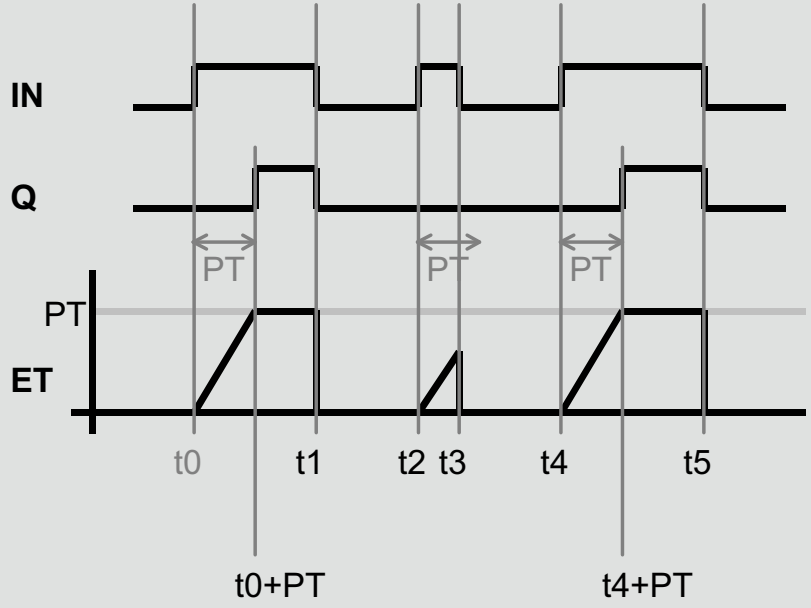
Los temporizadores son los primeros ejemplos de bloques de función IEC 61131-3.

TON — temporizador con retardo a la conexión



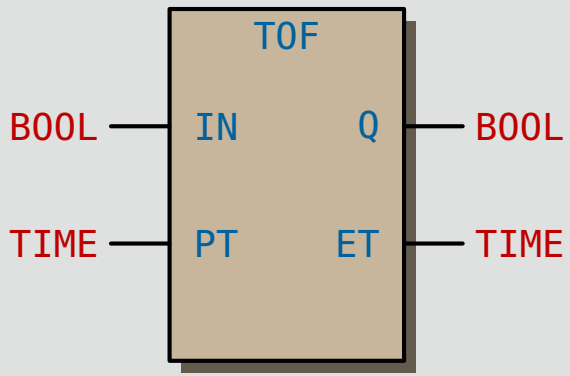
- IN$^{\lceil}$ → ET:=0 y empieza a contar el tiempo.
- ET alcanza PT → Q:=TRUE.
- IN$^{\rfloor}$ → Q:=FALSE y ET:=0.

TON – cronograma



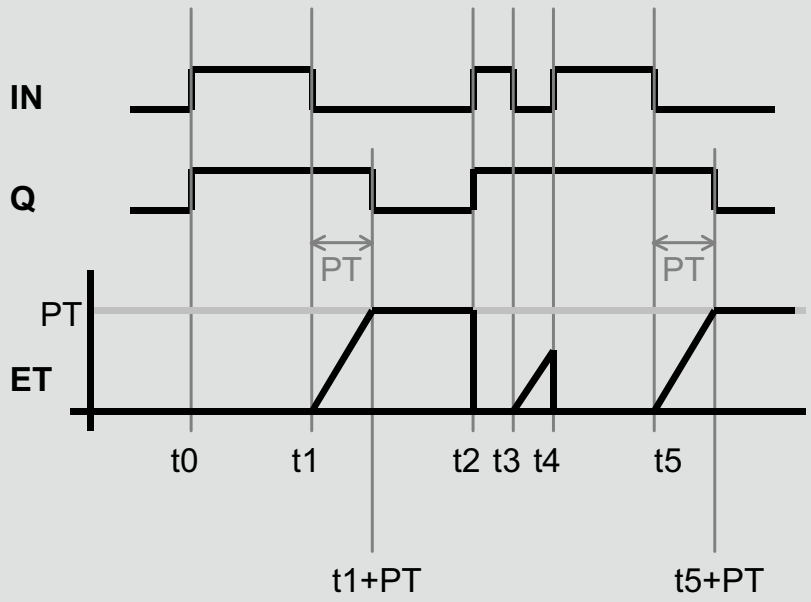
Temporizadores IEC 61131-3 – TOF

TOF — temporizador con retardo a la desconexión



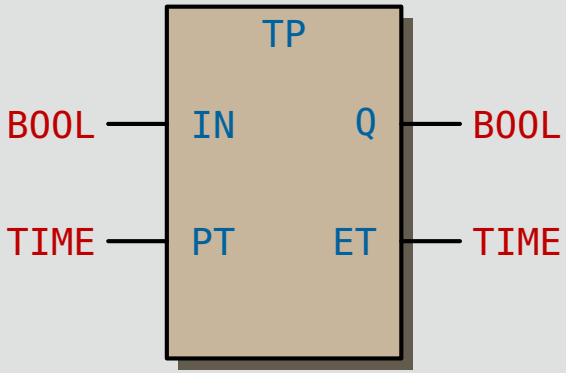
- IN↑ → Q:=TRUE.
- IN↓ → ET:=0 y empieza a contar el tiempo.
- ET alcanza PT → Q:=FALSE.

TOF – cronograma



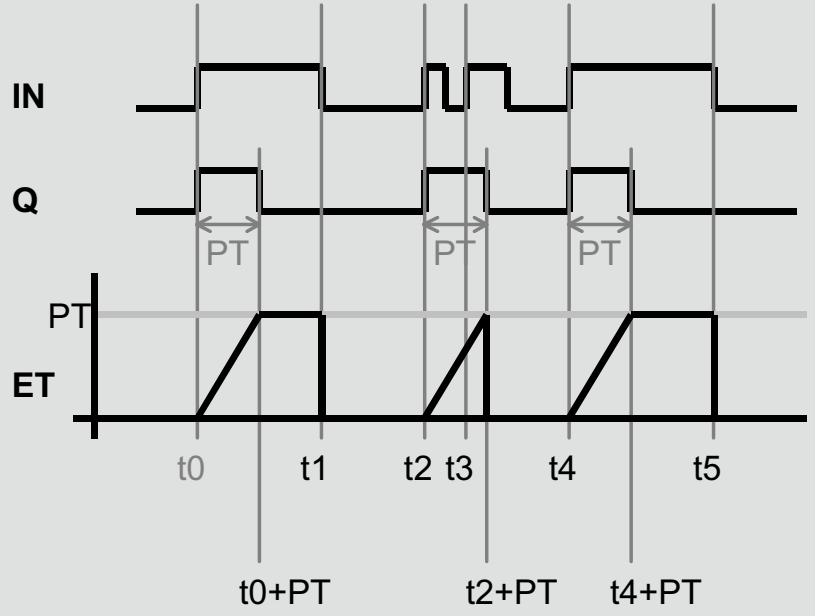
Temporizadores IEC 61131-3 – TP

TP — aestable



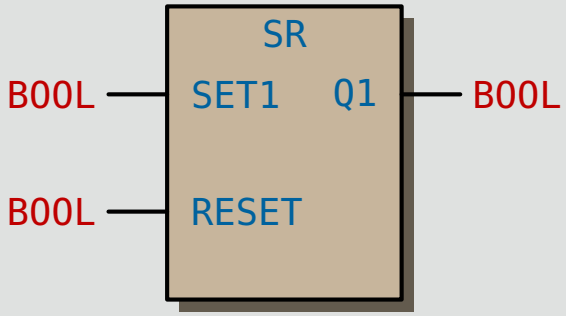
- IN↑ → Q:=TRUE, ET:=0 y empieza a contar el tiempo.
- ET alcanza PT → Q:=FALSE.
- IN↑ o IN↓ antes de que ET alcance PT → no ocurre nada.
- IN↓ después de que ET haya alcanzado PT → ET:=0.

TP – cronograma



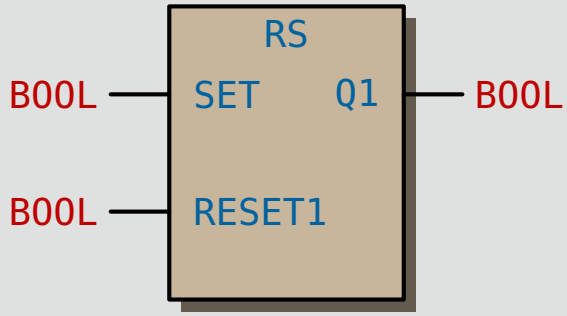
Biestables

SR: Set (prioritario)–Reset




```
1  FUNCTION_BLOCK SR
2     VAR_INPUT
3        SET1: BOOL;
4        RESET : BOOL;
5     END_VAR
6     VAR_OUTPUT
7        Q1: BOOL;
8     END_VAR
9  Q1 := SET1 OR (NOT RESET AND Q1);
10 END_FUNCTION_BLOCK
```

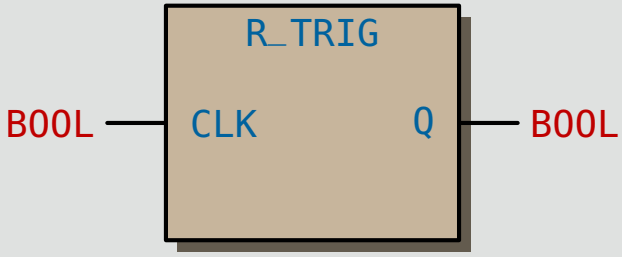
RS: Reset (prioritario)–Set




```
1  FUNCTION_BLOCK RS
2     VAR_INPUT
3        SET : BOOL;
4        RESET1: BOOL;
5     END_VAR
6     VAR_OUTPUT
7        Q1: BOOL;
8     END_VAR
9  Q1 := NOT RESET1 AND (SET OR Q1);
10 END_FUNCTION_BLOCK
```

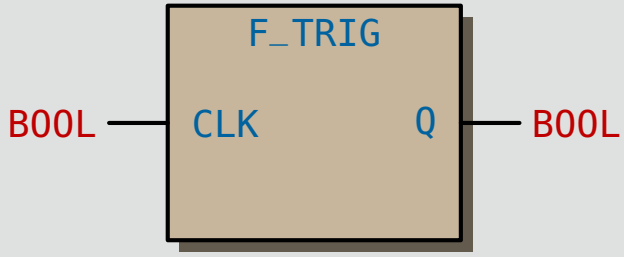
Detectores de flanco

R_TRIG: flanco de subida



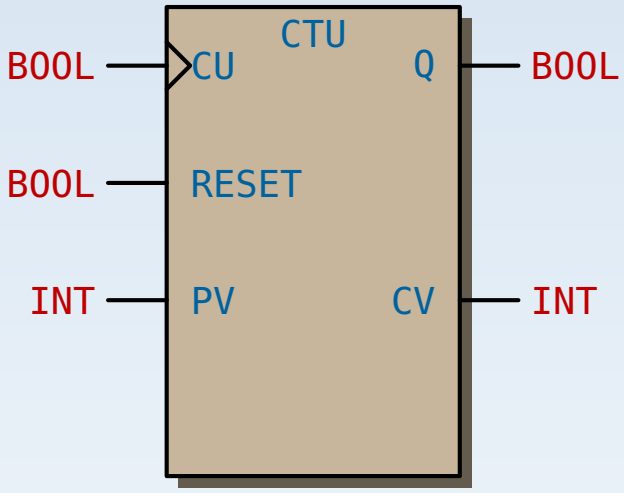

```
1  FUNCTION_BLOCK R_TRIG
2     VAR_INPUT
3        CLK: BOOL;
4     END_VAR
5     VAR_OUTPUT
6        Q: BOOL;
7     END_VAR
8     VAR
9        MEM: BOOL:= 0;
10    END_VAR
11 Q := CLK AND NOT MEM;
12 MEM := CLK;
13 END_FUNCTION_BLOCK
```

F_TRIG: flanco de bajada




```
1  FUNCTION_BLOCK F_TRIG
2     VAR_INPUT
3        CLK: BOOL;
4     END_VAR
5     VAR_OUTPUT
6        Q: BOOL;
7     END_VAR
8     VAR
9        MEM: BOOL:= 0;
10    END_VAR
11 Q := NOT CLK AND MEM;
12 MEM := CLK;
13 END_FUNCTION_BLOCK
```

CTU– Contador ascendente



CU– *count up* (flanco de subida)

RESET– *reset* (CV:=0)

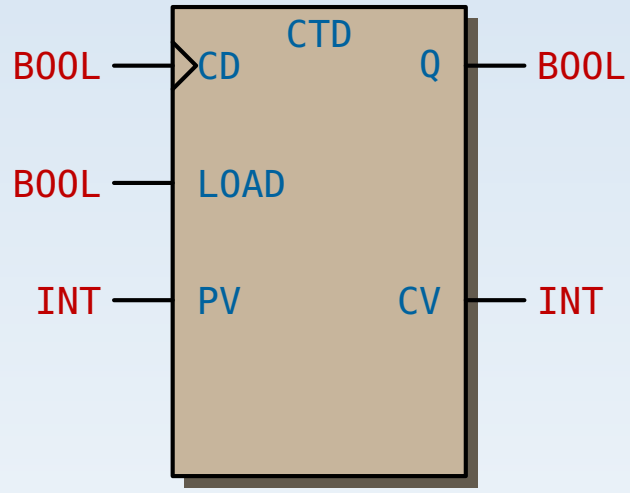
PV– *program value*

Q– cuenta finalizada

CV– *current value*

```
1  FUNCTION_BLOCK CTU
2     VAR_INPUT
3        CU: BOOL R_EDGE;
4        RESET : BOOL;
5        PV: INT;
6     END_VAR
7     VAR_OUTPUT
8        Q : BOOL;
9        CV: INT;
10    END_VAR
11 IF RESET THEN (* Puesta a cero. *)
12    CV := 0;
13 ELSIF CU AND (CV<PV) THEN
14    CV := CV+1; (* Cuenta ascendente. *)
15 END_IF;
16 Q := (CV >= PV); (* Cuenta finalizada. *)
17 END_FUNCTION_BLOCK
```

CTD– Contador descendente



CD– *count down* (flanco de subida)

LOAD– *load* (CV:=PV)

PV– *program value*

Q– cuenta finalizada

CV– *current value*

Código ST:

```
1  FUNCTION_BLOCK CTD
2     VAR_INPUT
3        CU: BOOL R_EDGE;
4        LOAD: BOOL;
5        PV: INT;
6     END_VAR
7     VAR_OUTPUT
8        Q : BOOL;
9              CV: INT;
10    END_VAR
11 IF LOAD THEN (* Carga del contador. *)
12    CV := PV;
13 ELSIF CU AND (CV>0) THEN
14    CV := CV-1; (* Cuenta descendente. *)
15 END_IF;
16 Q := (CV<=0); (* Cuenta finalizada. *)
17 END_FUNCTION_BLOCK
```

Índice

Referencias


International Electrotechnical Commission.
Programmable controllers - Part 3: Programming languages.
IEC, 2006.


Karl-Heinz John and Michael Tiegelkamp.
IEC 61131–3: Programming Industrial Automation Systems.
Springer, 2010.