



**HO  
GENT**

# **Workshop PLC**

IEC-61131-3 Structured Text

## **IEC 61 131**

- IEC standaard voor programmable controllers
- snelle en brede adoptie
- structureel en object georiënteerd programmeren

## **IEC 61 131**

- Part 1: General Information
- Part 2: Equipment requirements & test
- Part 3: Programming Languages
- Part 4: User Guidelines
- Part 5: Communications
- Part 6: Functional Safety
- Part 7: Fuzzy Control Programming
- Part 8: Guidelines for the application and implementation of programming languages
- Part 9: Single-drop digital communication interface for small sensors and actuators

## IEC 61 131 - 3

- Part 1: General Information
- Part 2: Equipment requirements & test
- **Part 3: Programming Languages**
- Part 4: User Guidelines
- Part 5: Communications
- Part 6: Functional Safety
- Part 7: Fuzzy Control Programming
- Part 8: Guidelines for the application and implementation of programming languages
- Part 9: Single-drop digital communication interface for small sensors and actuators

## **IEC 61 131 - 3**

- 6 programmeertalen
- door elkaar te gebruiken
- niet alle talen moeten geïmplementeerd worden om aan de standaard te voldoen

## **IEC 61 131 - 3**

- Ladder (LD)
- Function Block Diagram (FBD)
- Continuous Function Chart (CFC)
- Structured Text (ST)
- Instruction List (IL)
- Sequential Function Chart (SFC)

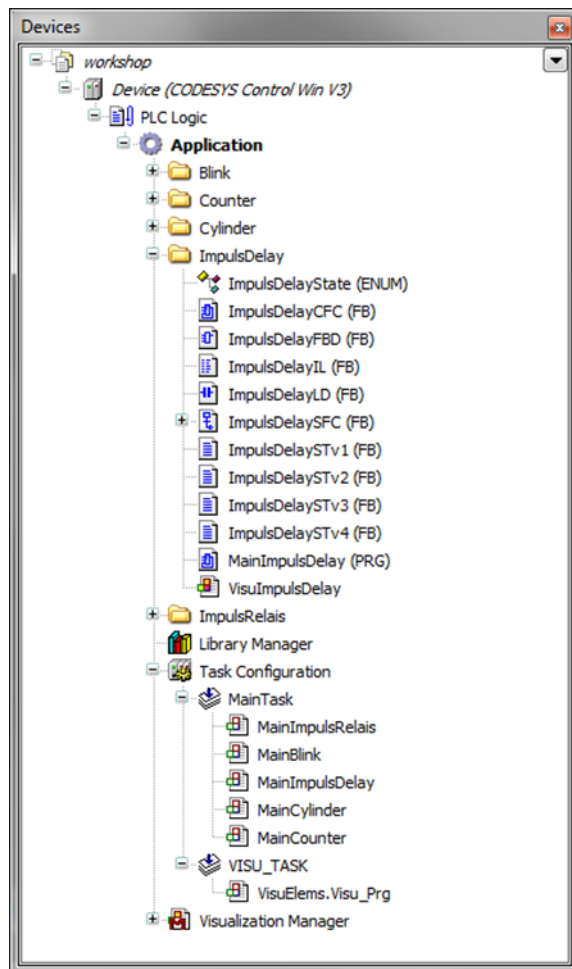
## **IEC 61 131 - 3**

- Ladder (LD)
- Function Block Diagram (FBD)
- Continuous Function Chart (CFC)
- **Structured Text (ST)**
- Instruction List (IL)
- Sequential Function Chart (SFC)



## **IEC 61 131 - 3**

- Resource
- Task
- Program Organisation Unit (POU)
  - Program
  - Function Block
  - Function



HO  
GENT

# IEC 61 131 - 3

## DataTypes

- BOOL
- BYTE / WORD / DWORD / LWORD
- SINT / INT / DINT / LINT / ... / UINT...
- REAL / LREAL
- TIME / DATE
- CHAR / STRING
- WCHAR / WSTRING
  
- array / struct / enum
  
- functieblok

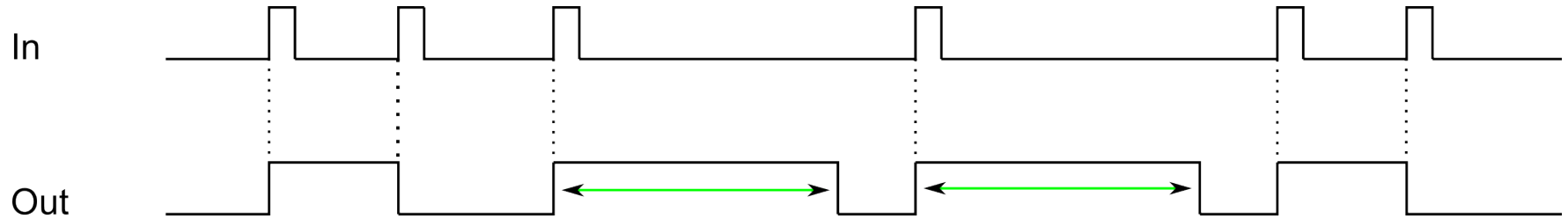
## **IEC 61 131 - 3**

### Variabele scope

- temp / lokaal / globaal
- Input / output / In-Out
- constanten
- retain / persistent

## IEC 61 131 - 3

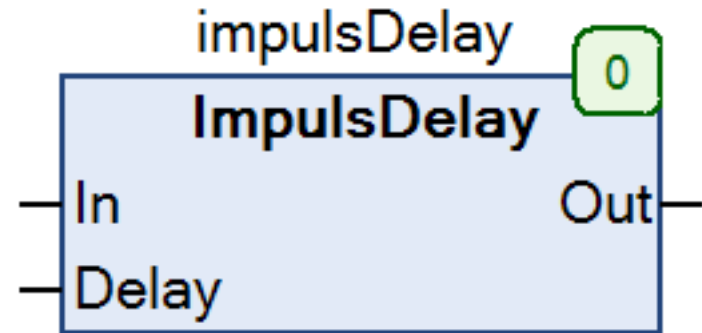
Voorbeeld: ImpulsDelay



## IEC 61 131 - 3

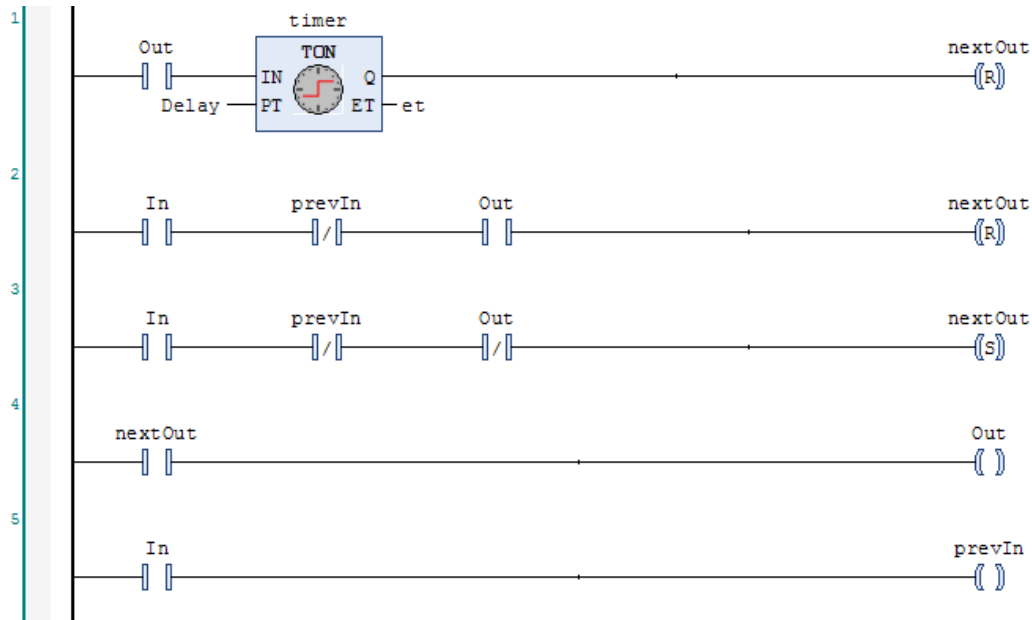
ImpulsDelay: Variabelen / Datatypes

```
1  FUNCTION_BLOCK ImpulsDelay
2  VAR_INPUT
3      In : BOOL;
4      Delay: TIME := T#5s;
5  END_VAR
6  VAR_OUTPUT
7      Out : BOOL;
8  END_VAR
9  VAR
10     timer: TON;
11 END_VAR
```



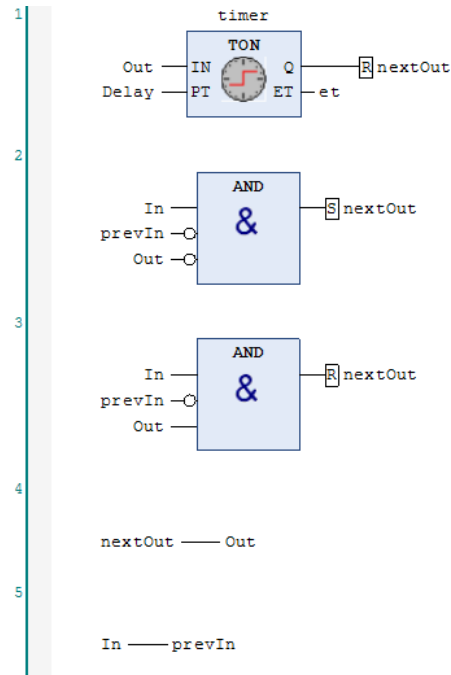
## IEC 61 131 - 3

### ImpulsDelay: Ladder (LD)



## IEC 61 131 - 3

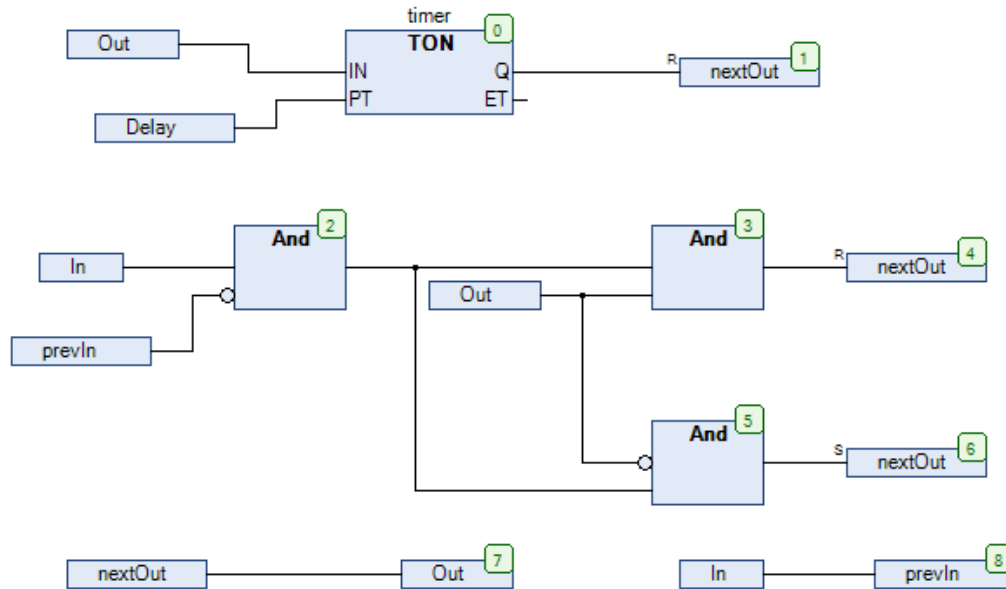
### ImpulsDelay: Function Block Diagram (FBD)





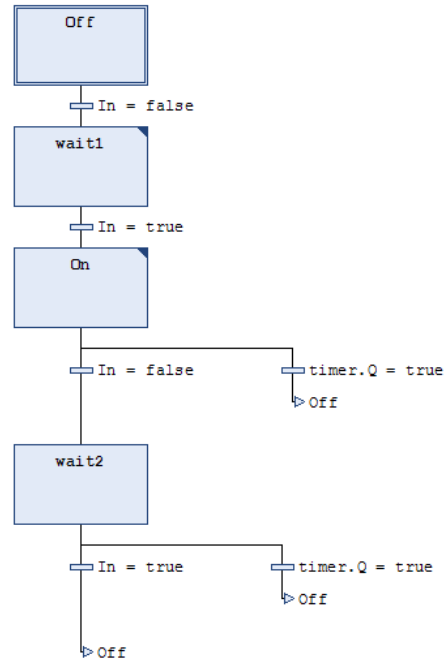
## IEC 61 131 - 3

### ImpulsDelay: Continuous Function Chart (CFC)



## IEC 61 131 - 3

### ImpulsDelay: Sequential Function Chart (SFC)



## IEC 61 131 - 3

### ImpulsDelay: Instruction List (IL)

1	CAL	timer( IN:= Out, PT:= Delay)
2	LD	In
	ANDN	prevIn
	ANDN	Out
	S	nextOut
3	LD	In
	ANDN	prevIn
	AND	Out
	R	nextOut
4	LD	timer.Q
	R	nextOut
5	LD	nextOut
	ST	Out
6	LD	In
	ST	prevIn

## IEC 61 131 - 3

### ImpulsDelay: Structured Text (ST)

```
1  timer(In := Out, PT := Delay);  
2  
3  IF In AND NOT prevIn THEN  
4      Out := NOT Out;  
5  END_IF;  
6  
7  IF timer.Q THEN  
8      Out := FALSE;  
9  END_IF;  
10  
11 prevIn := In;
```

## **IEC 61 131 – 3 Structured Text**

- hogere tekstuele programmeertaal
- $\pm$  pascal syntax
- instructies worden sequentieel uitgevoerd

## IEC 61 131 – 3 Structured Text

### Commentaar

```
// here be dragons
```

```
(*  
Dear maintainer,  
Once you're done trying to optimize this routine, and you have realized  
what a terrible mistake that was, please increment the following counter  
as a warning to the next guy:
```

```
total_hours_wasted_here = 42  
*)
```

## **IEC 61 131 – 3 Structured Text**

### Variabelen / Datatypes

- identiek voor alle IEC 61 131 – 3 programmeertalen

## **IEC 61 131 – 3 Structured Text**

### Statements

- eindigt altijd met een ;
- leeg
- variabele
- toekenning
- functie(-block) aanroep
- conditionele structuren
- samengestelde statements



## IEC 61 131 – 3 Structured Text

### Toekenning

- `<variabele><toekenning><expressie><;>`
- van rechts naar links!
- datatypes respecteren!
  
- `:=`
- `S=`
- `R=`

## IEC 61 131 – 3 Structured Text

### Expressies

- 33
- switch1
- sqrt(25)
- a AND b
- $1 * (2 + 5)$
- out := in1

constante

variabele

functieaanroep

bewerking

samengestelde bewerking

toekenning

## IEC 61 131 – 3 Structured Text

### Constanten

- True
- 0
- WORD#16#2EF1
- BYTE#2#01010101
- INT#12
- 3.14
- 'Hello World'

## **IEC 61 131 – 3 Structured Text**

### Booleaanse Operatoren

- AND
- OR
- NOT
- XOR

# IEC 61 131 – 3 Structured Text

## Rekenkundige Operatoren

- +
- -
- \*
- \*\*
- /
- MOD

# IEC 61 131 – 3 Structured Text

## Comparatoren

- =
- <>
- <
- <=
- >
- >=

# IEC 61 131 – 3 Structured Text

## Functies

- `<naam functie><(>[arg1[,arg2]]<)>`
  - NEG, INC, DEC, TRUNC, FRAC, ABS, FLOOR, SQR, SQRT, LN, LOG, EXP, EXPT
  - SIN, COS, TAN, ASIN, ACOS, ATAN
  - MIN, MAX
  - EQ, NE, GT, GE, LT, LE
  - ROL, ROR, SHL, SHR
  - CONCAT, DELETE, FIND, INSERT, LEFT, LEN, MID, REPLACE, RIGHT
  - TO\_TYPE, TYPE\_TO\_TYPE
  - TIME
  - ...

## IEC 61 131 – 3 Structured Text

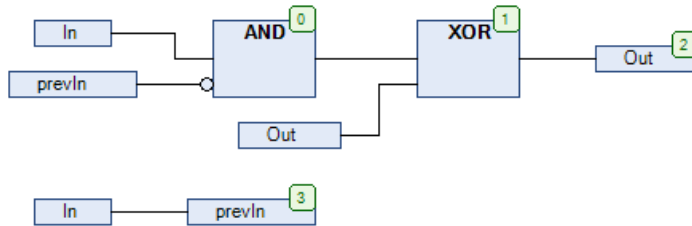
Volgorde bewerkingen

- ()
- functie aanroep
- -, NOT
- <, <=, >, >=
- =, <>
- AND
- XOR
- OR



## IEC 61 131 – 3 Structured Text

### Oefening 1.1: ImpulsRelais

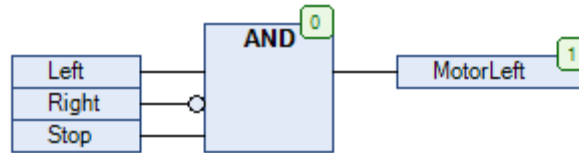


```
Out := Out XOR (in AND NOT prevIn);  
prevIn := In;
```

## IEC 61 131 – 3 Structured Text

### Oefening 3.1: Conveyor - manueel

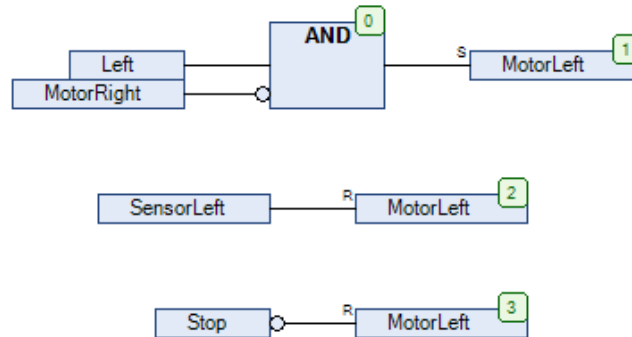
- manuele mode vraagt om een puur booleaanse oplossing met een gewone toekenning



## IEC 61 131 – 3 Structured Text

### Oefening 3.1: Conveyor - automatisch

- automatische mode is dan weer een typisch probleem om op te lossen met set/reset



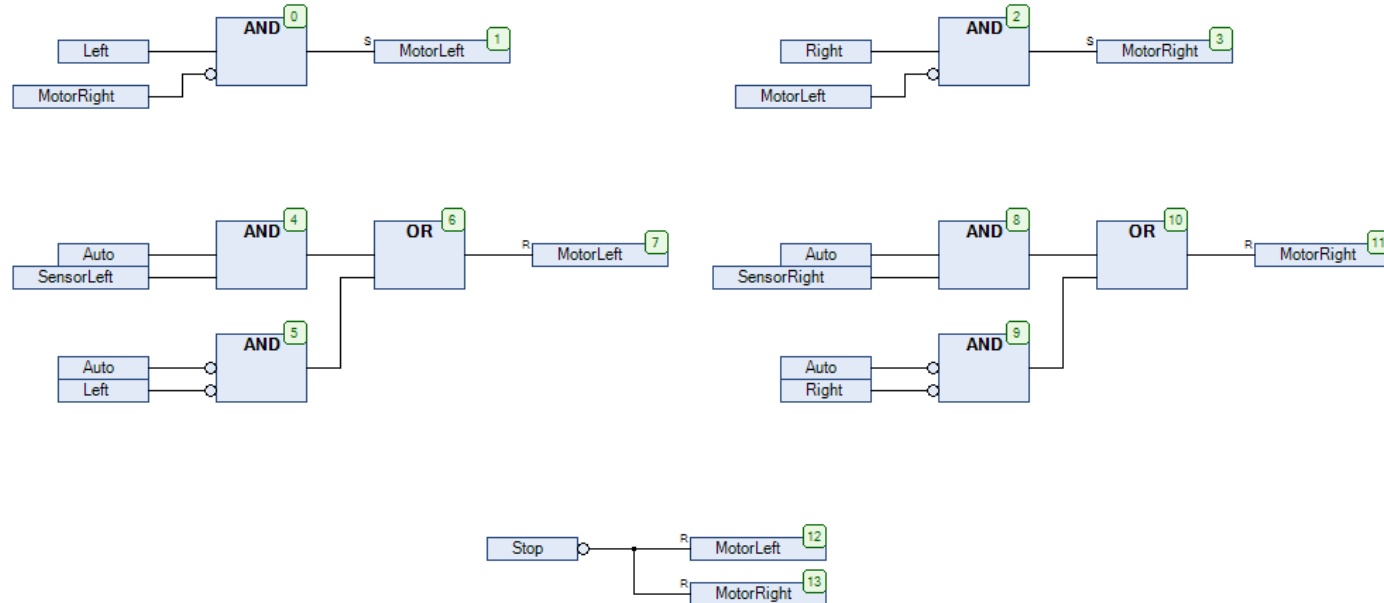
## **IEC 61 131 – 3 Structured Text**

### **Oefening 3.1: Conveyor - volledig**

- manuele en automatische werking samenvoegen lukt niet zomaar. Je kan immers geen set/reset en gewone toekenning door elkaar gebruiken
- Het vraagt dus wat knip en plakwerk...

# IEC 61 131 – 3 Structured Text

## Oefening 3.1: Conveyor



## IEC 61 131 – 3 Structured Text

### Oefening 3.1: Conveyor

```
MotorLeft S= Left AND NOT MotorRight;  
MotorRight S= Right AND NOT MotorLeft;
```

```
MotorLeft R= (Auto AND SensorLeft) OR (NOT Auto AND NOT Left);  
MotorRight R= (Auto AND SensorRight) OR (NOT Auto AND NOT Right);
```

```
MotorLeft R= NOT Stop;  
MotorRight R= NOT Stop;
```

## **IEC 61 131 – 3 Structured Text**

Functieblok aanroep

in tegenstelling tot functies moet er voor een functieblok een variabele aangemaakt worden => instantie van die functieblok

## **IEC 61 131 – 3 Structured Text**

### **Functieblok aanroep**

in tegenstelling tot functies moet er voor een functieblok een variabele aangemaakt worden => instantie van die functieblok

die variabele gedraagt zich als een struct met daaronder alle in-, uit- en interne variabelen die de functieblok nodig heeft om te kunnen functioneren



# IEC 61 131 – 3 Structured Text

## Voorbeeld: Timer

PLC\_PRG

```
1 PROGRAM PLC_PRG
2 VAR
3     motor: BOOL;
4     timer: TON;
5 END_VAR
```



PLC\_PRG

Device.Application.PLC_PRG			
Expression	Type	Value	Prepared value
motor	BOOL	FALSE	
timer	TON		
IN	BOOL	FALSE	
PT	TIME	T#0ms	
Q	BOOL	FALSE	
ET	TIME	T#0ms	

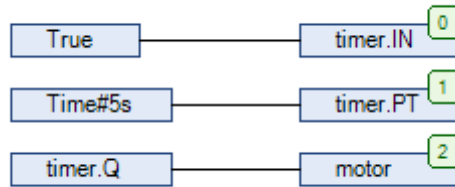
## **IEC 61 131 – 3 Structured Text**

### **Functieblok aanroep**

Variabelen in een struct zijn individueel aanspreekbaar. Op die manier kunnen de in- en uitgangen van een functieblok aangestuurd en afgevraagd worden.

## IEC 61 131 – 3 Structured Text

Voorbeeld: Timer



```
timer.IN := True;  
timer.PT := Time#5s;  
motor := timer.Q;
```

## **IEC 61 131 – 3 Structured Text**

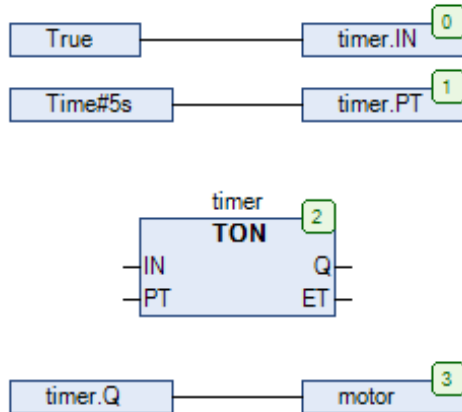
Functieblok aanroep

MAAR:

Daarmee is de functieblok zelf nog niet opgeroepen, en dus de achterliggende code nog niet uitgevoerd.

## IEC 61 131 – 3 Structured Text

Voorbeeld: Timer



```
timer.IN := True;  
timer.PT := Time#5s;  
timer();  
motor := timer.Q;
```

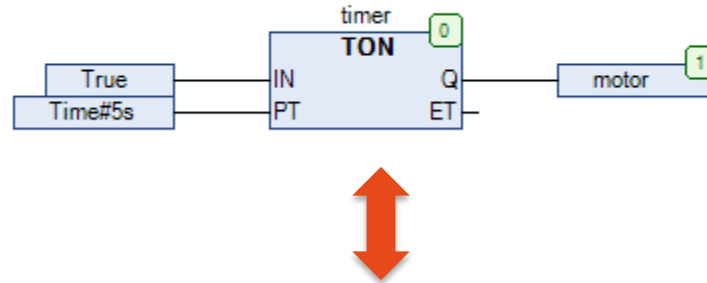
## **IEC 61 131 – 3 Structured Text**

### **Functieblok aanroep**

aansturen van ingangen en afvragen van uitgangen kan gecombineerd worden met de functieaanroep

## IEC 61 131 – 3 Structured Text

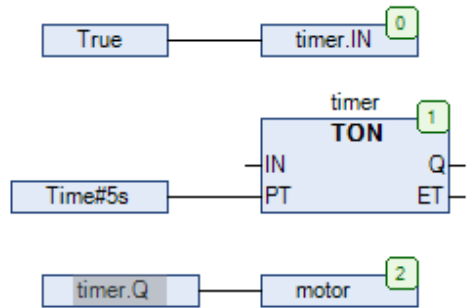
Voorbeeld: Timer



```
timer(IN := true, PT := Time#5s, Q => motor);
```

## IEC 61 131 – 3 Structured Text

Voorbeeld: Timer



```
timer.IN := True;  
timer(PT := Time#5s);  
motor := timer.Q;
```



## **IEC 61 131 – 3 Structured Text**

Functieblok aanroep

LET OP met de volgorde van de verschillende instructies!

## IEC 61 131 – 3 Structured Text

Voorbeeld: Set-Reset

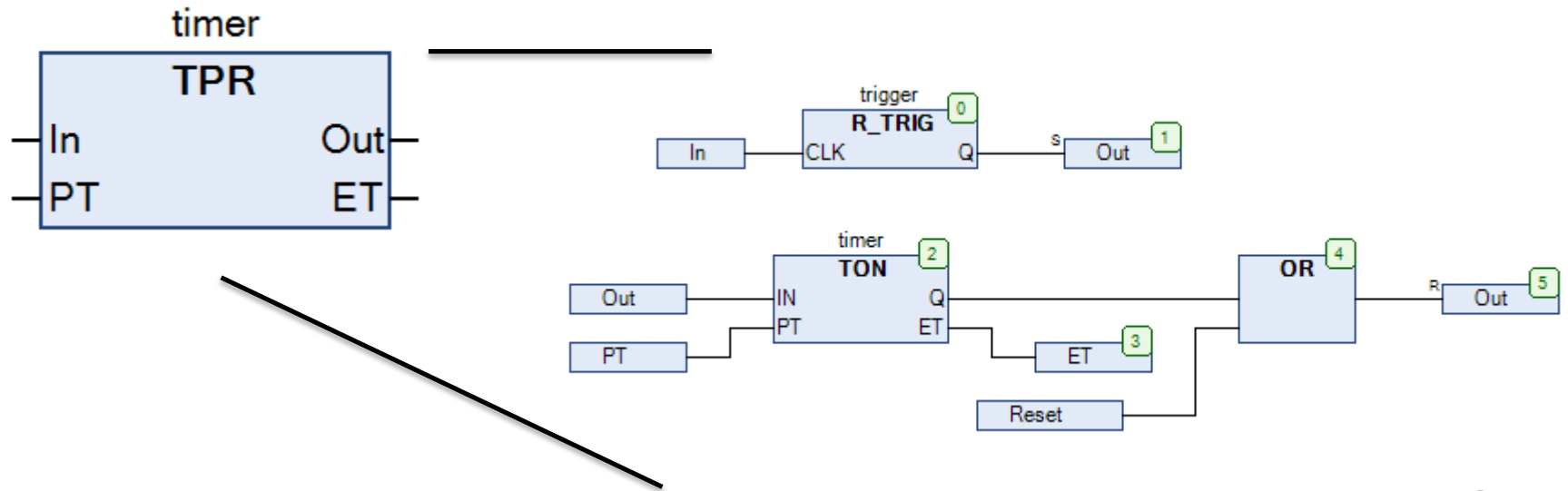
```
latch.SET := True;  
latch();  
lamp := latch.Q1;
```



```
lamp := latch.Q1;  
latch();  
latch.SET := True;
```

## IEC 61 131 – 3 Structured Text

### Oefening 2.1: Puls Timer met Reset



## IEC 61 131 – 3 Structured Text

### Oefening 2.1: Puls Timer met Reset

```
trigger.CLK := In;  
trigger();  
Out S= trigger.Q;  
  
timer.IN := Out;  
timer.PT := PT;  
timer();  
ET := timer.ET;  
Out R= timer.Q OR Reset;
```

## IEC 61 131 – 3 Structured Text

### Oefening 2.1: Puls Timer met Reset

```
trigger(CLK := In);  
Out S= trigger.Q;
```

```
timer(IN := Out, PT := PT, ET => ET);  
Out R= timer.Q OR Reset;
```

## IEC 61 131 – 3 Structured Text

### Conditionele Structuren

- (blokken) code 0, 1 of meerdere keren uitvoeren
- IF, CASE
- FOR, WHILE, REPEAT
- LET OP: Lussen MOETEN in één cyclus kunnen uitgevoerd worden!!

## IEC 61 131 – 3 Structured Text

### IF structuur

```
IF <booleaanse expressie> THEN  
    ;  
ELSIF <booleaanse expressie> THEN  
    ;  
ELSE  
    ;  
END_IF;
```

## IEC 61 131 – 3 Structured Text

### Oefening 1.2: ImpulsRelais

```
Out := Out XOR (In AND NOT prevIn);  
prevIn := In;
```



```
IF In AND NOT prevIn THEN  
    Out := NOT Out;  
END_IF;  
prevIn := In;
```



# IEC 61 131 – 3 Structured Text

## Oefening 3.2: Conveyor

```
MotorLeft S= Left AND NOT MotorRight;  
MotorRight S= Right AND NOT MotorLeft;  
  
MotorLeft R= (Auto AND SensorLeft) OR (NOT Auto AND NOT Left);  
MotorRight R= (Auto AND SensorRight) OR (NOT Auto AND NOT Right);  
  
MotorLeft R= NOT Stop;  
MotorRight R= NOT Stop;
```



```
IF Auto THEN      // AUTOMATISCH  
  
    MotorLeft S= Left AND NOT MotorRight;  
    MotorRight S= Right AND NOT MotorLeft;  
  
    MotorLeft R= SensorLeft;  
    MotorRight R= SensorRight;  
  
    MotorLeft R= NOT Stop;  
    MotorRight R= NOT Stop;  
  
ELSE              // MANUEEL  
  
    MotorLeft := Left AND NOT Right AND Stop;  
    MotorRight := Right AND NOT left AND Stop;  
END_IF
```

## IEC 61 131 – 3 Structured Text

### CASE structuur

```
CASE <integer variabele> OF  
  <const int>:  
    ;  
  <const int>:  
    ;  
ELSE  
  ;  
END_CASE;
```

## IEC 61 131 – 3 Structured Text

FOR structuur

```
FOR <int var> := <int exp> TO <int exp> BY <int exp> DO  
    ;  
END_FOR;
```

## IEC 61 131 – 3 Structured Text

WHILE structuur

```
WHILE <boolean expression> DO  
    ;  
END_WHILE;
```

## IEC 61 131 – 3 Structured Text

WHILE structuur

```
REPEAT  
    ;  
UNTIL <boolean expression>  
END_WHILE;
```

## IEC 61 131 – 3 Structured Text

### Conditionele structuren - Lussen

Wat gebeurt er indien je d.m.v. een lus wacht totdat een ingang van toestand veranderd?

```
MotorRight := False;  
WHILE NOT Right DO  
    ;  
END_WHILE;  
MotorRight := True;  
WHILE NOT SensorRight DO  
    ;  
END_WHILE;  
MotorRight := False;
```

## IEC 61 131 – 3 Structured Text

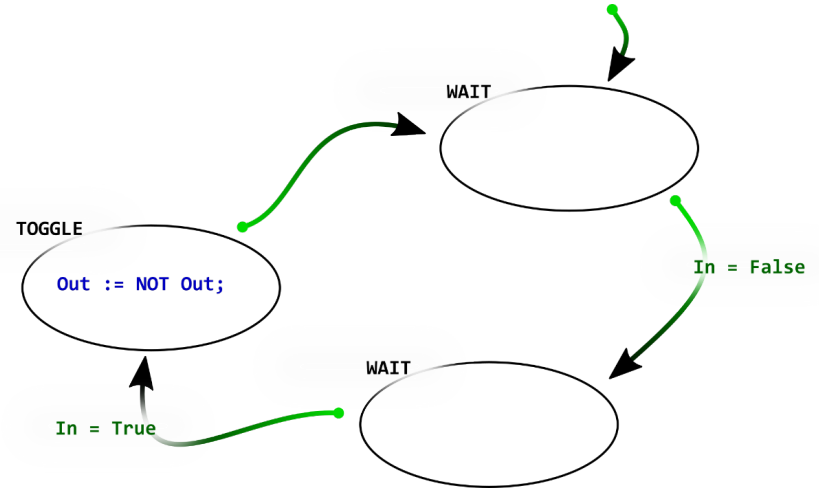
### Finite State Machines

- meerdere toestanden
- per cyclus wordt slechts 1 toestand uitgevoerd
- transities met booleaanse voorwaarden om van toestand te veranderen
- ideaal voor sequentiële oefeningen
- grafisch → Sequential Function Charts

## IEC 61 131 – 3 Structured Text

### Finite State Machines

de structuur van een programma kan grafisch voorgesteld worden d.m.v. een toestanden diagram



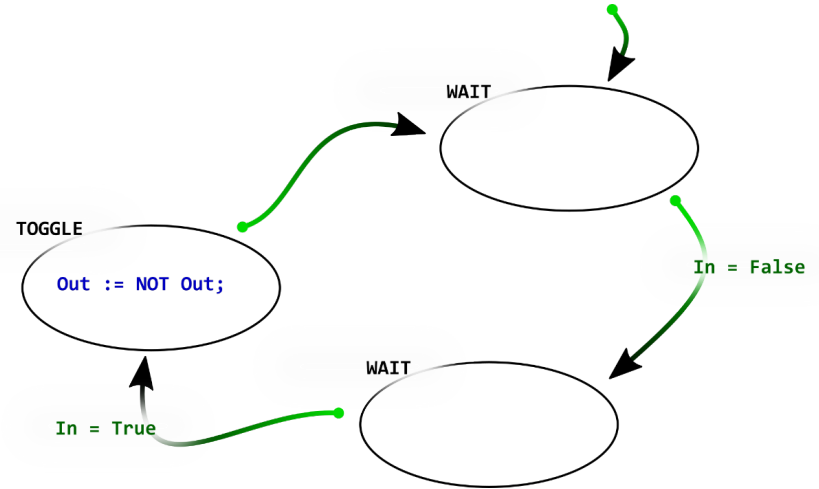


## IEC 61 131 – 3 Structured Text

### Finite State Machines

toestanden worden voorgesteld  
d.m.v. eilanden.

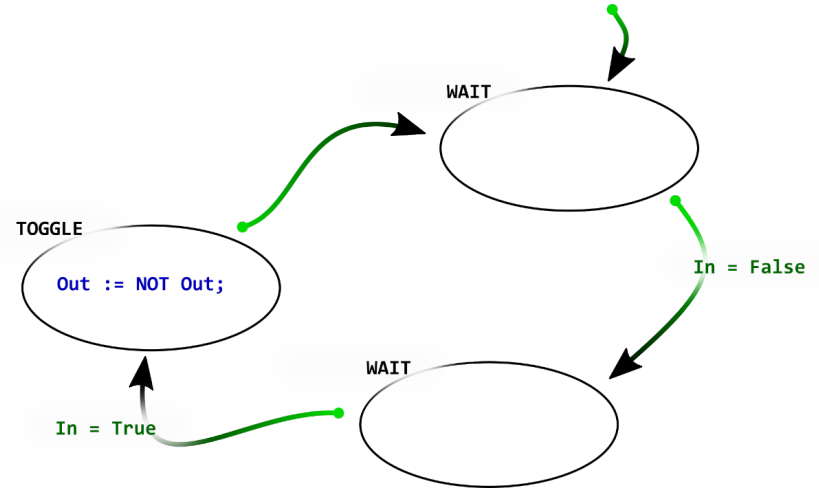
De FSM “bevindt” zich altijd in exact  
1 toestand



## IEC 61 131 – 3 Structured Text

### Finite State Machines

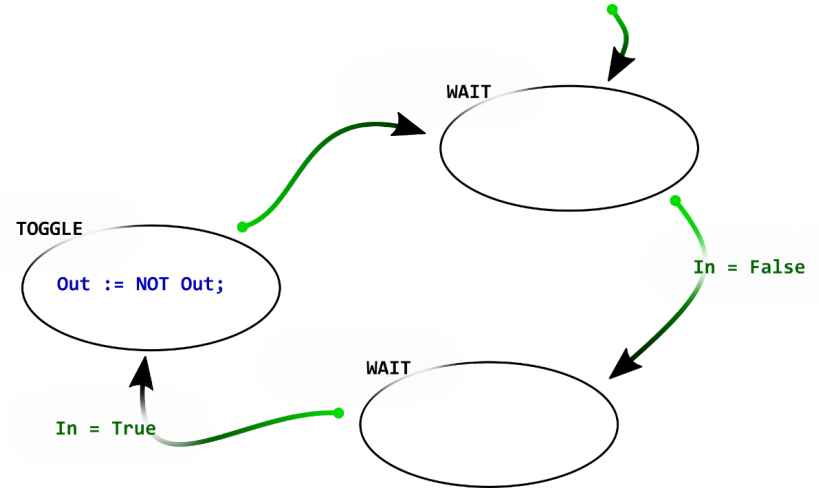
In iedere toestand geven we weer wat moet uitgevoerd worden wanneer de finite state machine zich in die toestand bevind.



## IEC 61 131 – 3 Structured Text

### Finite State Machines

tussen de toestanden zijn transities getekend. Van zodra de voorwaarde van een vertrekkende pijl in de huidige toestand waar is, verspringt de FSM naar de volgende toestand



# IEC 61 131 – 3 Structured Text

## Oefening 1.3 – ImpulsRelais

### STAP 1

definieer – naast alle andere  
variabelen – een interne  
variabele “state” van het type  
INT

```
VAR_INPUT  
    In: BOOL;  
END_VAR  
VAR_OUTPUT  
    Out: BOOL;  
END_VAR  
VAR  
    state: INT;  
END_VAR
```

## **IEC 61 131 – 3 Structured Text**

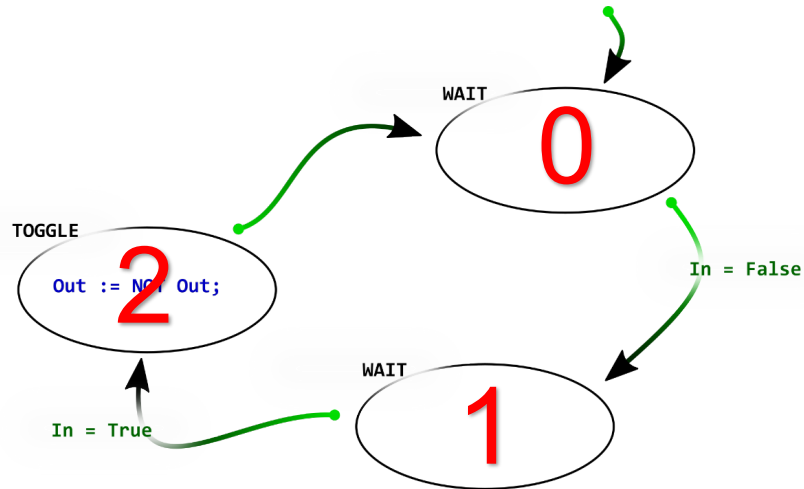
### Oefening 1.3 – ImpulsRelais

#### STAP 2

- programmeer een CASE structuur.
- voorzie evenveel cases als er toestanden zijn in het toestanden diagram
- geef de initiële toestand waarde 0

# IEC 61 131 – 3 Structured Text

## Oefening 1.3 – ImpulsRelais



```
CASE state OF  
0:  
    ;  
1:  
    ;  
2:  
    ;  
END_CASE;
```

## **IEC 61 131 – 3 Structured Text**

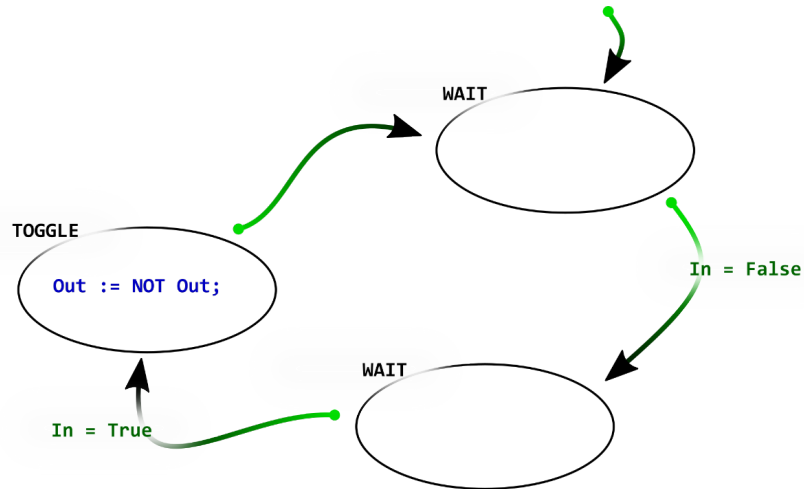
### Oefening 1.3 – ImpulsRelais

#### STAP 3

- programmeer in de verschillende cases de code die in die betreffende toestand moet uitgevoerd worden

# IEC 61 131 – 3 Structured Text

## Oefening 1.3 – ImpulsRelais



```
CASE state OF  
0:  
    ;  
1:  
    ;  
2:  
    Out := NOT Out;  
END_CASE;
```



## **IEC 61 131 – 3 Structured Text**

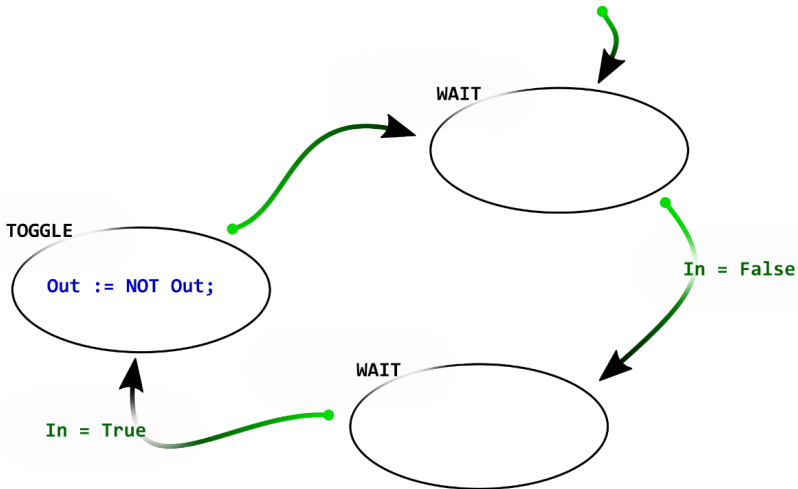
### **Oefening 1.3 – ImpulsRelais**

#### **STAP 4**

- transities programmeer je in de toestand waar de pijl vertrekt
- een transitie programmeren is zoveel als de variabele state gelijk stellen aan het nummer van de volgende toestand
- indien er een voorwaarde gekoppeld is aan de transitie dan programmeer je deze in een IF structuur

# IEC 61 131 – 3 Structured Text

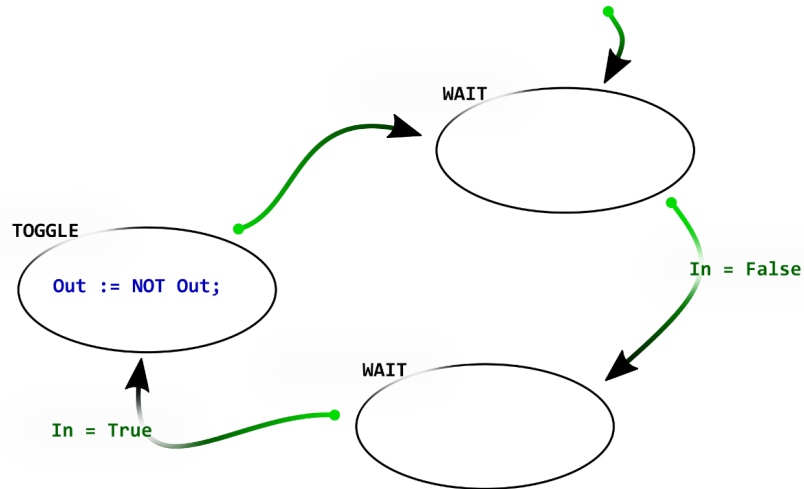
## Oefening 1.3 – ImpulsRelais



```
CASE state OF
0:
    IF In = False THEN
        state := 1;
    END_IF;
1:
    ;
2:
    Out := NOT Out;
END_CASE;
```

# IEC 61 131 – 3 Structured Text

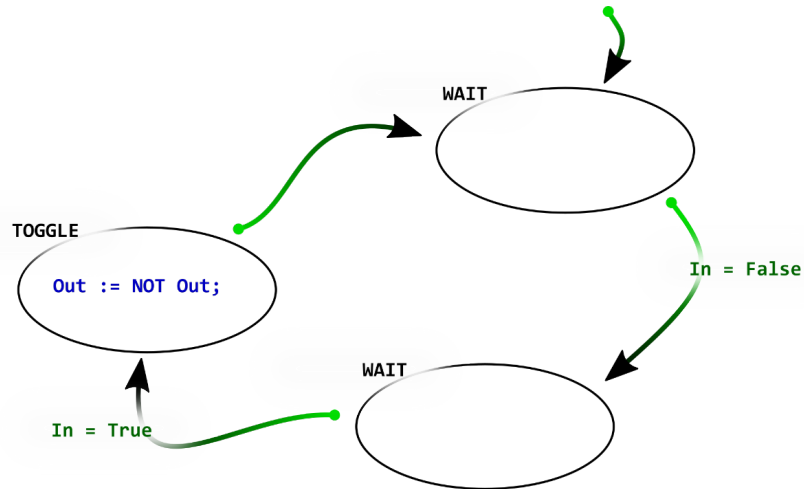
## Oefening 1.3 – ImpulsRelais



```
CASE state OF
0:
    IF In = False THEN
        state := 1;
    END_IF;
1:
    IF In = True THEN
        state := 2;
    END_IF;
2:
    Out := NOT Out;
END_CASE;
```

# IEC 61 131 – 3 Structured Text

## Oefening 1.3 – ImpulsRelais



```
CASE state OF
0:
    IF In = False THEN
        state := 1;
    END_IF;
1:
    IF In = True THEN
        state := 2;
    END_IF;
2:
    Out := NOT Out;
    state := 0;
END_CASE;
```

## **IEC 61 131 – 3 Structured Text**

### Oefening 2.2 – Puls Timer met Reset

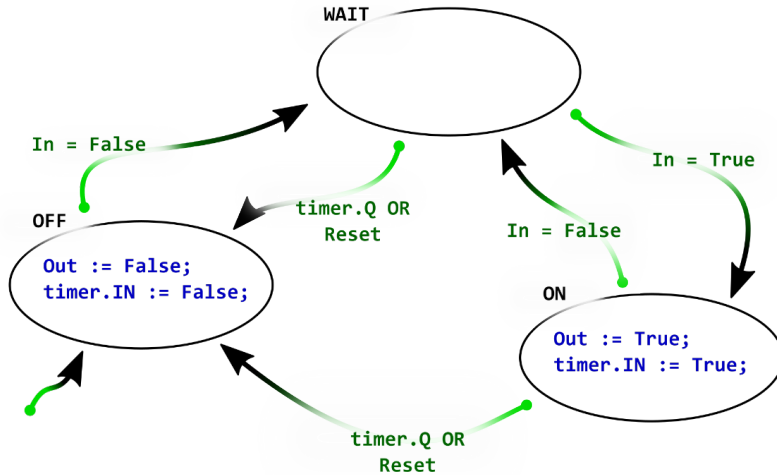
Indien je in een FSM een andere functieblok wil gebruiken dan:

- roep je de functieblok aan BUITEN de case structuur
- schrijven naar de ingangen, afvragen van de uitgangen doe je in de case structuur

# IEC 61 131 – 3 Structured Text

## Oefening 2.2 – Puls Timer met Reset

```
timer(PT := PT, ET => ET);
```



```
timer(PT := PT, ET => ET);
```

```
CASE state OF
```

```
0:
```

```
Out := FALSE;  
timer.IN := FALSE;
```

```
IF In = FALSE THEN  
    state := 1;  
END_IF
```

```
1:
```

```
IF timer.Q OR Reset THEN  
    state := 0;  
ELSIF In = TRUE THEN  
    state := 2;  
END_IF
```

```
2:
```

```
Out := TRUE;  
timer.IN := TRUE;
```

```
IF timer.Q OR Reset THEN  
    state := 0;  
ELSIF In = FALSE THEN  
    state := 1;  
END_IF
```

```
END_CASE
```

## **IEC 61 131 – 3 Structured Text**

### Oefening 4.1 – Blokken groeperen

- ideale oefening om sequentieel op te lossen
- ontwerp het toestanden diagram al programmerend / debuggen
- veel eenvoudige toestanden  $\leftrightarrow$  beperkt aantal complexere
- vermijd counters en flank detectoren
  
- oefening baart kunst