PROCESS AUTOMATION / PROGRAMMING WORKSHOP

Structured Text in practice

Octavian M. Machidon · UL FRI

Winter semester 2026/27

01 IF and Boolean logic

`p05_ST_if`

Make a light follow a switch
or deliberately keep its state.

02 A button press, once

`p06_ST_rtrig`

Detect a rising edge and toggle
a light once per press.

03 Motor direction control

`p07_ST_motor`

Use two timer instances
to block immediate reversal.

04 Material records

`p08_ST_materialTracking`

Store a workpiece and recipe;
copy the record to a location.

The same code runs again next scan

PlcTask
every 10 ms

MAIN (Ladder)

4 networks, evaluated in order

MAIN

1	**p05_ST_if** PROGRAM	IF examples update its own light variable.
2	**p06_ST_rtrig** PROGRAM	Detects a rising edge using buttonOld.
3	**p07_ST_motor_0** FB INSTANCE	buttonRight → forward · buttonLeft → backward T#2s → lockTime · movement/direction → MAIN
4	**FB_materialTracking** FB INSTANCE	Calls p08_ST_materialTracking; uses location[0..7].

Read the few symbols these examples use

```
VAR
    switch: BOOL; light: BOOL;
    button1: BOOL; button2: BOOL;
END_VAR
```

```
IF switch = TRUE THEN
    light := TRUE;
END_IF
light := (switch AND button1) OR
         (NOT switch AND button2);
```

```
forwardTimer(IN:=forward, PT:=lockTime,
             Q => backwardLock);
```

```
location[1].workpiece :=
    location[0].workpiece;
```

Symbols in the code

`VAR`	Start declarations
`:=`	Assign a value
`=`	Compare two values
`AND / OR / NOT`	Boolean logic
`[1]`	Choose array element
`.workpiece`	Field of a record
`=>`	Connect a block output

No ELSE: the old value is kept

FIRST FORM

```
IF switch = TRUE THEN
    light := TRUE;
END_IF
```

EXPLICITLY THE SAME BEHAVIOR

```
IF switch = TRUE THEN
    light := TRUE;
ELSE
    light := light;
END_IF
```

Scan	switch	light before	Assignment	light after
1	FALSE	FALSE	None	FALSE
2	TRUE	FALSE	light := TRUE	TRUE
3	FALSE	TRUE	None	TRUE

Assign both cases when the light must follow

IF / ELSE

```
IF switch THEN
    light := TRUE;
ELSE
    light := FALSE;
END_IF
```

DIRECT BOOLEAN ASSIGNMENT

```
light := switch;
```

switch	light after
FALSE	FALSE
TRUE	TRUE

Choose the behavior first: “keep the previous value” and “follow the input” are different requirements.

Choose which button controls the light

NESTED IF

```
IF switch THEN
    IF button1 THEN
        light := TRUE;
    ELSE
        light := FALSE;
    END_IF
ELSE
    light := button2;
END_IF
```

EQUIVALENT EXPRESSION

```
light := (switch AND button1)
      OR (NOT switch AND button2);
```

Which value is selected?

- switch = TRUE → button1
- switch = FALSE → button2

Toggle once when the button becomes TRUE

EXECUTABLE CODE

```
IF button AND NOT buttonOld THEN
    light := NOT light;
END_IF


buttonOld := button;
```

button · buttonOld · light : BOOL

WHY HOLDING DOES NOT RETRIGGER

buttonOld is updated after the IF, so it represents the previous scan during the comparison.

SIX SCANS · TWO BUTTON PRESSES



Scans 2 & 6: rising edges → light toggles.

A motor controller is a function block

INPUTS AND OUTPUTS

```
FUNCTION_BLOCK p07_ST_motor
VAR_INPUT
    forward: BOOL;
    backward: BOOL;
    lockTime: TIME;
END_VAR
VAR_OUTPUT
    movement: BOOL;
    direction: BOOL;
END_VAR
```

INTERNAL STATE

```
VAR
    forwardTimer: TOF;
    backwardTimer: TOF;
    rotateForward: BOOL;
    rotateBackward: BOOL;
    backwardLock: BOOL;
    forwardLock: BOOL;
END_VAR
```

Call the instance and connect its parameters

INSTANCE DECLARATION IN MAIN

```
p07_ST_motor_0: p07_ST_motor;
```

SAME LADDER CALL, SHOWN IN ST

```
p07_ST_motor_0(
    forward := buttonRight,
    backward := buttonLeft,
    lockTime := T#2s,
    movement => motorMovement,
    direction => motorDirection);
```

:= inputs

Pass commands and settings into the block.

=> outputs

Copy the block outputs to variables in MAIN.

lockTime = T#2s

Each request locks the opposite direction

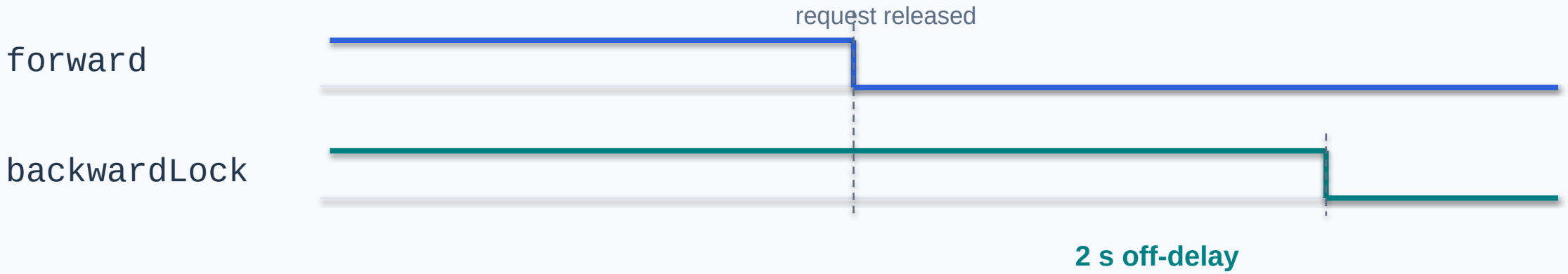
TWO INDEPENDENT TOF INSTANCES

```
forwardTimer(IN:=forward,
    PT:=lockTime, Q => backwardLock);

backwardTimer(IN:=backward,
    PT:=lockTime, Q => forwardLock);
```

TOF: off-delay

- Q becomes TRUE with IN.
- After IN falls, Q stays TRUE for lockTime.



Separate the request from permission to move

PERMITTED MOTION

```
rotateForward := forward AND NOT forwardLock;
rotateBackward := backward AND NOT backwardLock;
movement := rotateForward OR rotateBackward;
```

After the request values and timers have settled

forward	backward	rotateForward	rotateBackward	movement
FALSE	FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	TRUE	FALSE	FALSE	FALSE

During a reversal delay, the active lock can still block a single requested direction.

Keep direction unchanged while stopped

LONG FORM — COMMENTED OUT IN THE FILE

```
IF rotateBackward THEN
    direction := TRUE;
ELSIF rotateForward THEN
    direction := FALSE;
ELSE
    direction := direction;
END_IF
```

ACTIVE SHORT FORM

```
IF movement THEN
    direction := rotateBackward;
END_IF
```

Encoding in this example

FALSE = forward TRUE = backward

No movement → no assignment to direction → keep its previous value.

Why reversing the motor must wait

The request changes immediately, but permission to reverse stays blocked for 2 seconds.

1 FORWARD RUNNING

forward = TRUE
backward = FALSE
movement = TRUE
direction = FALSE (forward)

2 REVERSE REQUEST

forward = FALSE
backward = TRUE
backwardLock = TRUE
movement = FALSE
direction is held (forward)

3 AFTER 2 SECONDS

backwardLock = FALSE
rotateBackward = TRUE
movement = TRUE
direction = TRUE (backward)

Why? forwardTimer keeps backwardLock TRUE after forward falls (TOF off-delay).

Represent a workpiece as a structured record

LOCAL VARIABLE IN THE MATERIAL-TRACKING FB

```
location: ARRAY[0..7] OF ST_LOCATION;
```

ST_LOCATION

occupied · isDone
workpiece

→

ST_WORKPIECE

id
recipe

→

ST_RECIPE

Machine times +
directions
positionSequence[0..9]

→

ST_POSITION

rotation · extension
lift · grip

[index] selects an element; .field selects something inside that element.

Start with position and recipe types

ST_POSITION.TcDUT

```
TYPE ST_POSITION :
STRUCT
    rotation  : INT;
    extension : INT;
    lift      : INT;
    grip      : INT;
END_STRUCT
END_TYPE
```

ST_RECIPE.TcDUT

```
TYPE ST_RECIPE :
STRUCT
    // L2N
    timeMachine1 : TIME;
    timeMachine2 : TIME;
    directionMachine1 : BOOL;
    directionMachine2 : BOOL;
    // RR
    positionSequence :
        ARRAY [0..9] OF ST_POSITION;
END_STRUCT
END_TYPE
```

A location contains a workpiece and flags

ST_WORKPIECE.TcDUT

```
TYPE ST_WORKPIECE :
STRUCT
    id : INT;
    recipe : ST_RECIPE;
END_STRUCT
END_TYPE
```

ST_LOCATION.TcDUT

```
TYPE ST_LOCATION :
STRUCT
    workpiece : ST_WORKPIECE;
    occupied  : BOOL;
    isDone    : BOOL;
END_STRUCT
END_TYPE
```

workpiece is the item record. occupied and isDone belong to the location around it.

Set up the example workpiece

THE EIGHT ASSIGNMENTS IN THE FILE

```
location[0].occupied := TRUE;
location[0].workpiece.id := 42;
location[0].workpiece.recipe.directionMachine1 := FALSE;
location[0].workpiece.recipe.timeMachine1 := T#1S;
location[0].workpiece.recipe.positionSequence[0].rotation := 100;
location[0].workpiece.recipe.positionSequence[0].lift := 50;
location[0].workpiece.recipe.positionSequence[0].extension := 30;
location[0].workpiece.recipe.positionSequence[0].grip := 15;
```

These assignments are in the implementation.

The sample values are written again on every scan.

Copy the whole workpiece record in one line

TRANSFER LOGIC AS SUPPLIED

```
IF location[0].isDone THEN
    location[1].workpiece := location[0].workpiece;
    location[0].isDone := FALSE;
END_IF
```

location[0].workpiece

id = 42

recipe + its position sequence

copy

location[1].workpiece

The same record values are written here.

A data copy is not yet a complete transfer

Field	Before the IF	After the IF
location[0].workpiece.id	42	42
location[1].workpiece.id	0	42
location[0].occupied	TRUE	TRUE
location[1].occupied	FALSE	FALSE
location[0].isDone	TRUE	FALSE

The sample copies data; it does not clear the source.
It also leaves occupancy unchanged and does not check a sensor.

Before writing your first ST program

Decide when a value changes

No assignment keeps the previous value.
A direct assignment updates it each call.

Remember only what you need

A previous input detects an edge.
An FB instance keeps its own state.

Read the execution order

Timer calls come before lock tests.
The last assignment can override earlier ones.

Know what the code does not do

A record can be copied without an object moving.
Read the flags as carefully as the values.

PROCESS AUTOMATION / PROGRAMMING WORKSHOP

Structured Text in practice

Octavian M. Machidon · UL FRI

Winter semester 2026/27