24V production line

Conversion GRAFCET to program code
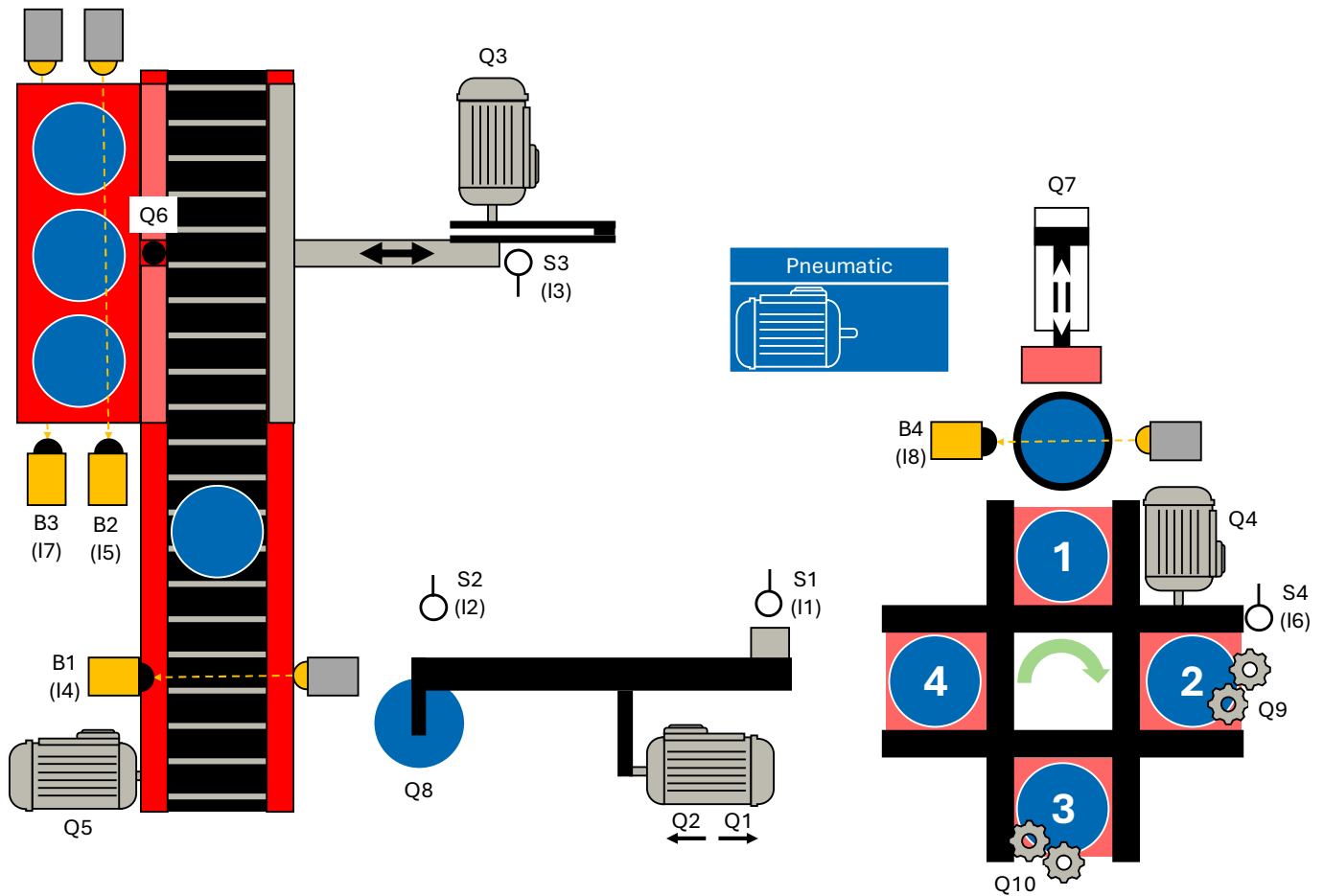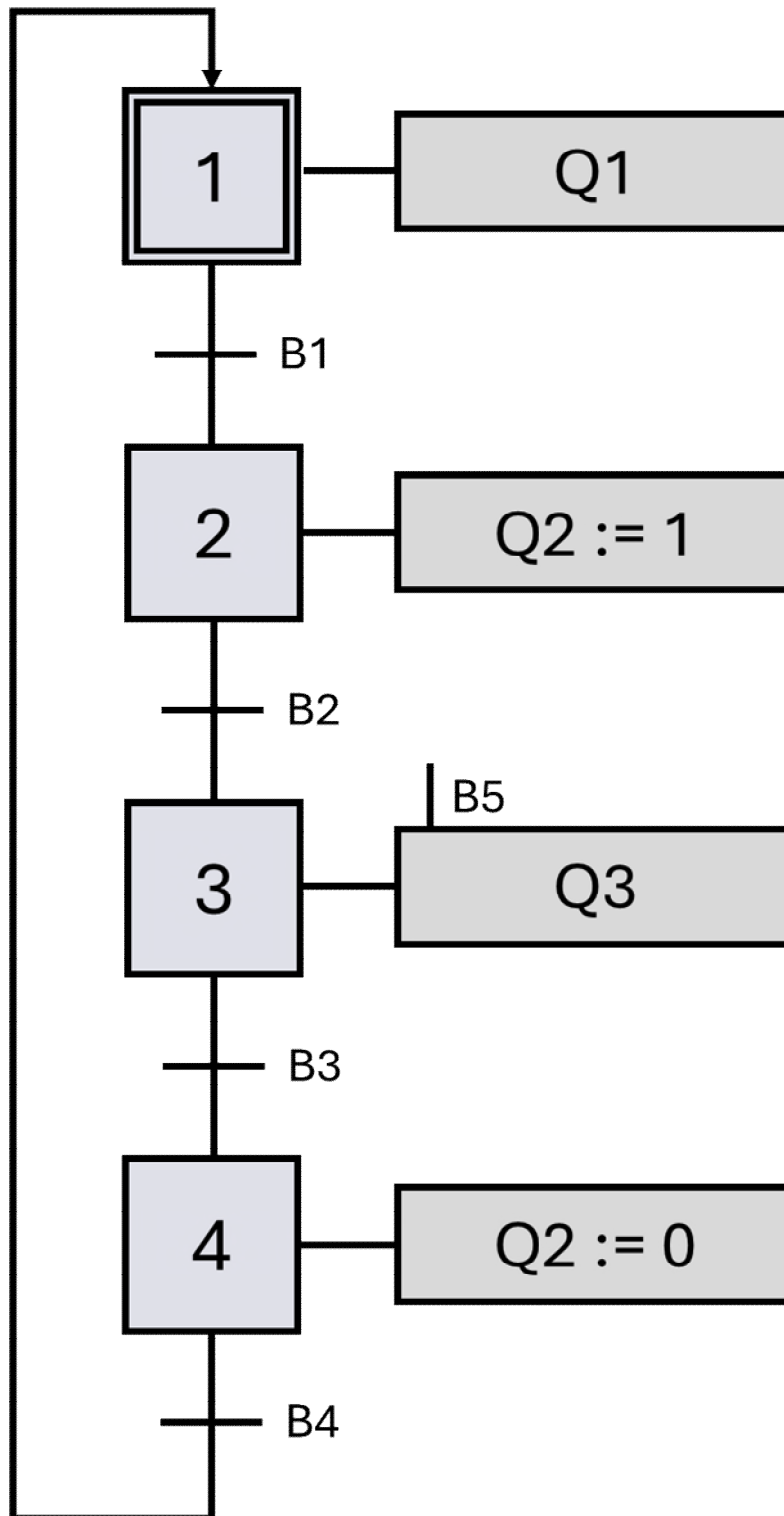
Table of contents

9 Conversion GRAFCET to program code

9.1 Functional description of the process chain

A PLC program is now to be created from the GRAFCET given for an example system. The basic procedure for converting a Grafcet into program code is illustrated below using a simple step chain:

Picture 1 Exemplary GRAFCET chain

GROLLMUS

fischertechnik

Functional description

Step 1 (initial step)

Step 1 is the initial step.
As long as the system is in this step, actuator Q1 is activated by means of a continuously acting action.
If the sensor delivers a B1 1 signal, the system jumps to the next step.

Step 2

Actuator Q2 is activated by saving ("TRUE").
If the sensor B2 delivers a 1 signal, the system jumps to the next step.

Step 3

As long as the system is in this step, actuator Q3 is activated by means of a continuously acting action with condition. In order for the actuator to be controlled with the value "TRUE", sensor B5 must also supply a 1 signal in addition to the active step.
If sensor B3 delivers a 1 signal, the system jumps to the next step.

Step 4

Actuator Q2 is reset ("FALSE").
If the sensor B4 delivers a 1 signal, the system jumps back to the initial step.

GROLLMUS

fischertechnik

9.2 Implement GRAFCET process chain in FBD

In this section, we will implement the sequence chain planned in GRAFCET in the FBD programming language.

9.2.1 Implement steps and transitions in FBD

The reset-dominant flip-flop is the central element in the implementation of the GRAFCET sequence chain in FBD. An active step is represented by an activated flip-flop. A step is activated via the set input and can therefore be regarded as part of the preceding transition. An active step is exited via the reset input and is therefore part of the subsequent transition.
A separate step marker representing the activated step is connected as a memory for the flip-flop.
Global flags, variables from global data blocks or local variables from the static area of the block interface can be used as variables for the step flags.
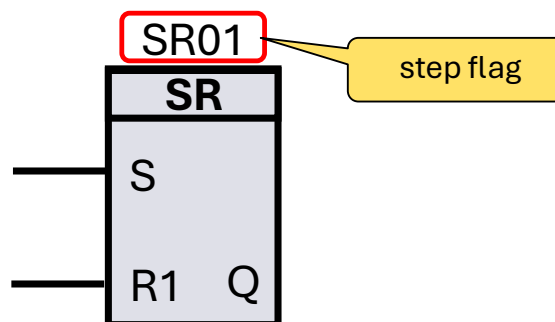
Picture 2 Flip-flop with step marker

Initial step
Step 1 is the initial step, which must be activated when the chain is initialized (Init). The step is also activated if the chain is in the last step (SR04) and the subsequent transition (B4) is fulfilled.
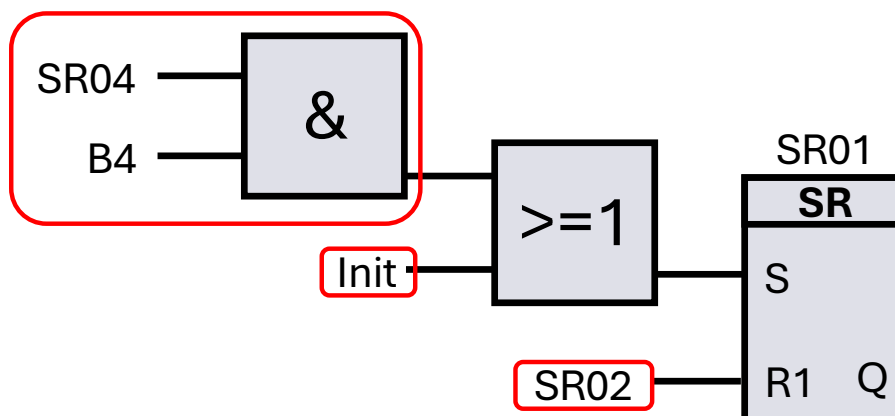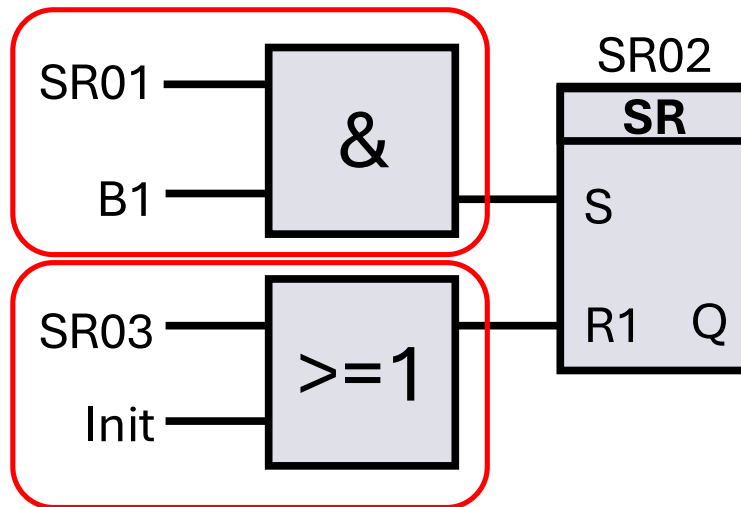

Picture 3 Programming initial step

The step must be reset if the following step (SR02) is active.

GROLLMUS · fischertechnik

Standard step

All further steps are set if the previous step and the corresponding transition are fulfilled. They are reset when the subsequent step is active or the chain is initialized.

Picture 4 Programming step

The last step is followed by a jump to the first step of the chain, so the last step is reset with the first step, as this is the following step.

This results in the following step chain for the GRAFCET shown at the beginning:
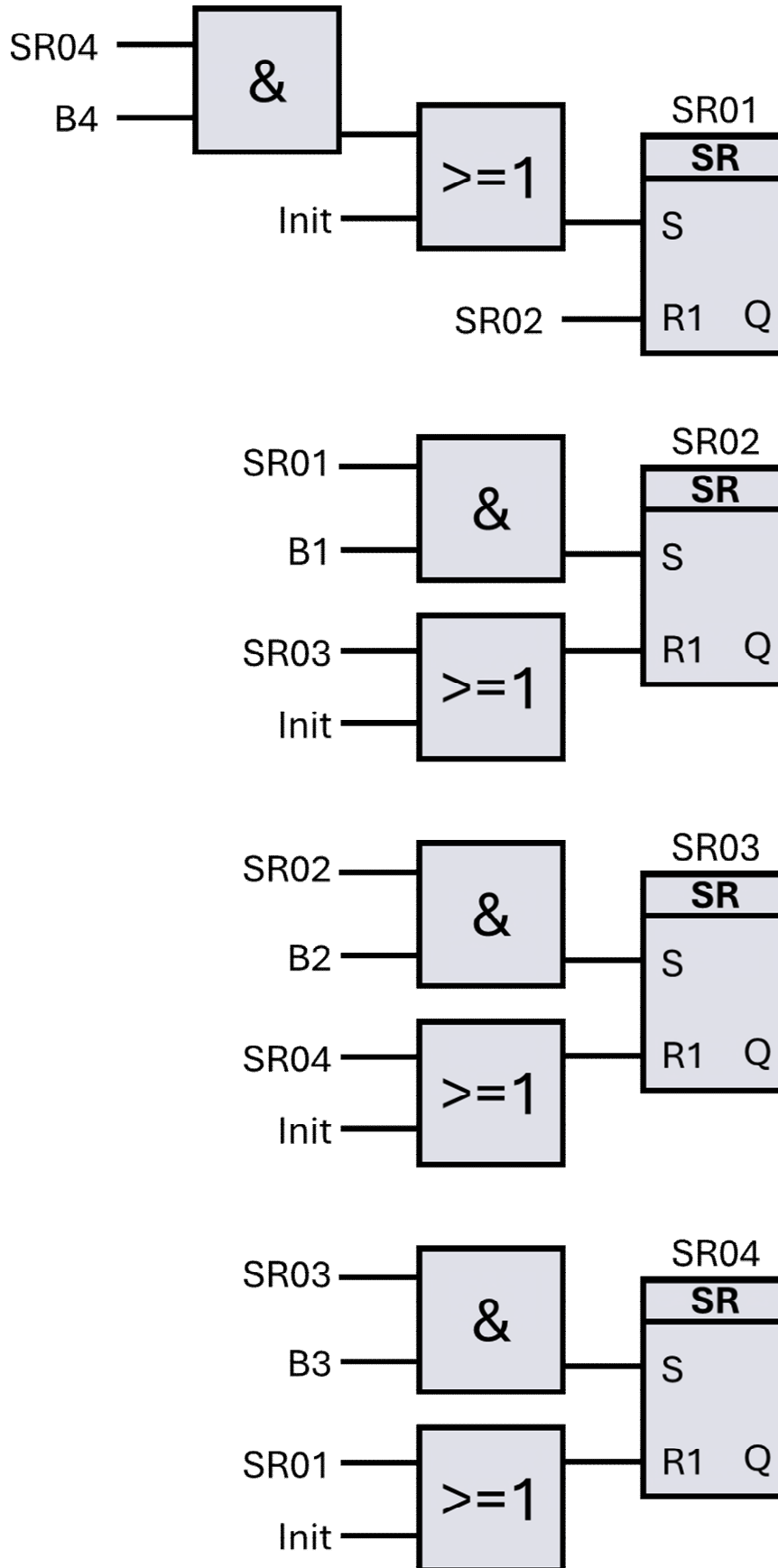
Picture 5 Programming step chain

9.2.2 Implement actions in FUP

Due to functional structuring aspects, the actions in the respective steps of the sequence chain can be programmed either in a separate control module or directly in the networks immediately after the chain in the same module.

Programming in a separate control module

One option is to program the actions of the steps in a separate control module. This method has the advantage that the control logic of the step chain and the execution of the actions are clearly separated from each other. This increases the clarity of the program, which is particularly advantageous for complex control systems. This separation also makes it easier to maintain and expand the program, as changes to the control logic or actions can be made independently of each other.

In this approach, the step chain is implemented in the main module. When a step becomes active, a signal is sent to the control module, which then executes the corresponding actions. The step flags are usually used as signals for the exchange. This structure enables modular programming in which each module has a clearly defined task.

Programming in the networks directly after the step chain

An alternative method is to program the actions directly in the networks immediately after the step chain in the same block. This approach has the advantage that the entire logic is concentrated in one place, which makes it easier to trace the execution of the program. This can be a more efficient and faster solution, especially for smaller and less complex projects.

Here, the actions are programmed directly in the networks that follow the corresponding steps in the step chain. This method can optimize the program runtime, as no additional block calls are required, and the structure of the program remains compact.
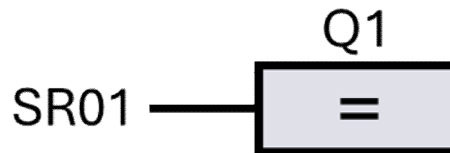
Continuous action in FBD:

Default:

The action is executed for as long as the step is active.

Implementation:

The corresponding step marker is written directly to the output via an assignment.
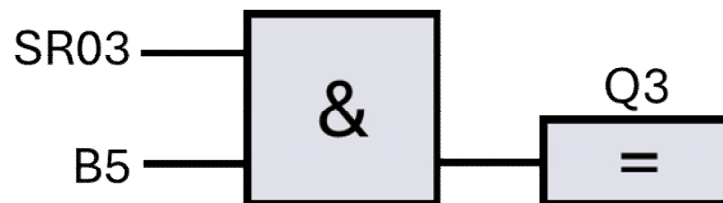

Picture 6 Continuous allocation

Continuous allocation with additional allocation condition:

Default:

The action is executed as long as the step is active and the assignment condition is fulfilled.

Implementation:

The corresponding step flag is logically linked to the assignment condition and the result of the link is assigned to the output.

Picture 7 Continuous assignment with condition

Saving action:

Default:

The saved action remains active over several steps.

Implementation:

The corresponding step flags for setting and resetting the action are connected to a flip-flop. The flip-flop has a saving effect on the output.
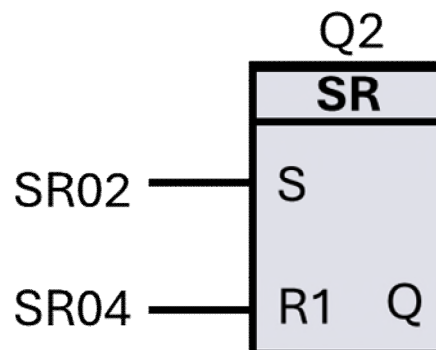
Picture 8 Storing action

When the chain is initialized, the action may remain set as step 4 (SR04) has not been activated. To prevent this, the variable for the initialization request (Init) can also be connected to the reset input using an OR link.

9.3 Implement GRAFCET process chain in ST / SCL
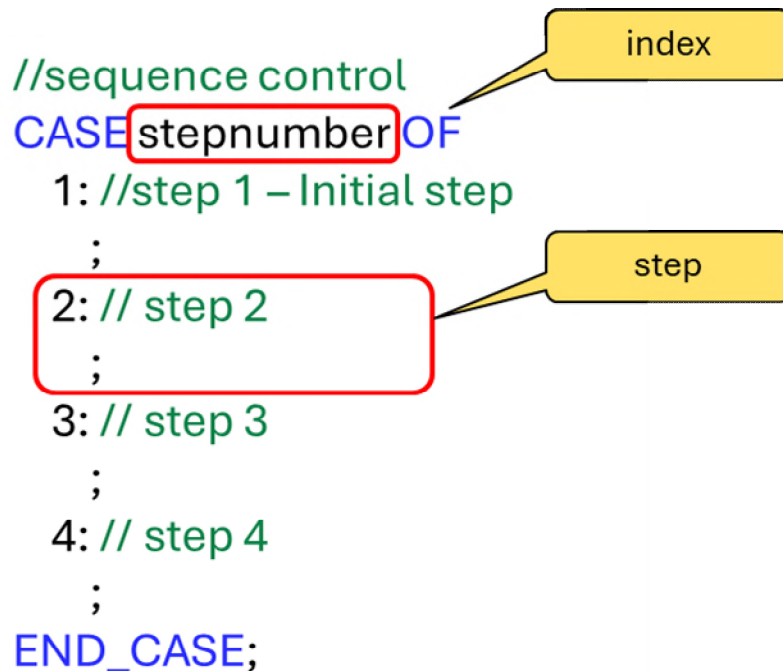
In this section, we will implement the sequence chain planned in GRAFCET in the ST / SCL programming language.

It is recommended to use a CASE structure to map the individual steps of the GRAFCET. To do this, an index variable with an integer data type (e.g. INT) must first be defined. This is the count variable that represents the current step.

```
//sequence control
CASE stepnumber OF         [index]
    1: //step 1 – Initial step
       ;
    2: // step 2               [step]
       ;
    3: // step 3
       ;
    4: // step 4
       ;
END_CASE;
```

Picture 9 CASE structure

### 9.3.1 Initializing the process chain

Step 1 is the initial step, which must be activated when the chain is initialized (Init). For this purpose, an IF statement must be placed before the CASE structure, which resets the index variable (step number) to 1 when initialization is requested.

```
//initialize sequence control
IF Init THEN
    stepnumber := 1;
END_IF;
```

Picture 10 Initialization

9.3.2 Structure of a step (CASE)

Each step (CASE) is structured according to the following scheme:
First, the actions that are to be carried out in the step are implemented. This is followed by an IF statement, which represents the transition and sets the index variable of the step chain to the next step if the transition conditions are met.
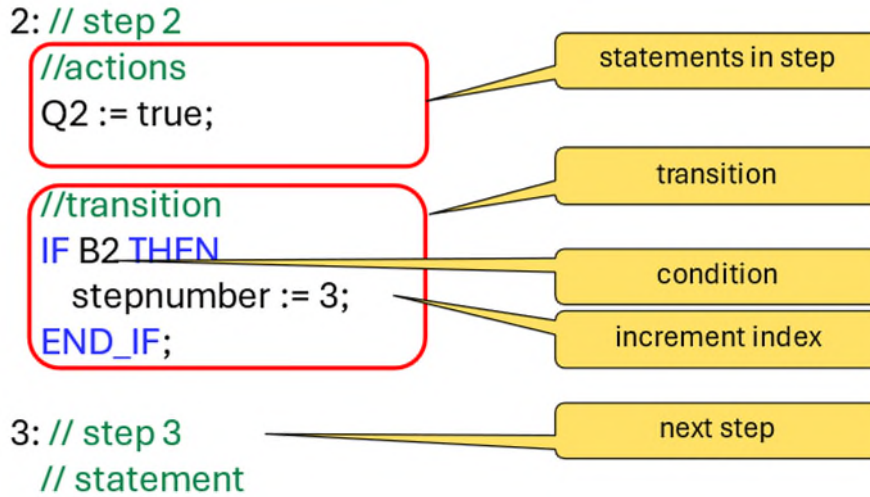
Picture 11 Structure step

Depending on the planned action, these must be implemented differently:

Continuous allocation:
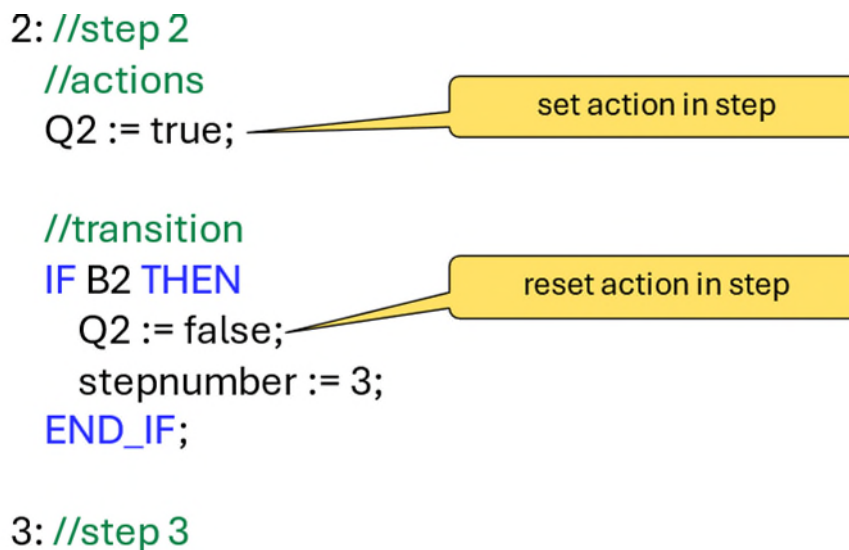Default
  The action is executed for as long as the step is active.
Implementation
  By assigning the signal state "TRUE" to the variable, a saving behavior is achieved. However, this is not desired at this point. Therefore, the action must be reset when leaving the step, which follows the fulfillment of the transition condition.

Picture 12 Continuous assignment with condition

10

Continuous allocation with additional allocation condition:

Default

The action is executed as long as the step is active and the assignment condition is fulfilled.

Implementation

The additional assignment condition can be assigned directly to the action. As with continuous assignment, the corresponding action must also be reset when leaving the step.

```
3: //step 3
    //action
    Q3 := B5;                    set action in step,
                                 depending on the signal of
                                 the assignment condition

    //transition
    IF B3 THEN                   reset action in step
        Q3 := false;
        stepnumber := 4;
    END_IF;


    4: //step 4
```

Picture 13 Continuous assignment with condition

Saving action:

Default

With stored actions, the action is executed over several steps.

Implementation

In principle, all actions are actions with a saving effect due to the direct assignment of the signal states. If this behavior is desired, the action must not be reset in the IF statement of the transition, but in the corresponding step.

```
2: //step 2
    //action
    Q2 := true;                  set action in step

    //transition
    IF B2 THEN
        stepnumber := 3;        //next step
    END_IF;

    4: //step 4
    //action
    Q2 := false;                 reset action in step   //reset action

    //transition
    IF B4 THEN
```

Picture 14 Storing action

GROLLMUS   **fischertechnik**

This results in the following step chain for the GRAFCET shown at the beginning:

```
//initialize sequence control
IF Init THEN
   stepnumber := 1;
END_IF;

//sequence control
CASE stepnumber OF
   1: //step 1 – Initial step
      //action
      Q1 := true;                  //set action

      //transition
      IF B1 THEN
         Q1 := false;              //reset action
         stepnumber := 2;          //next step
      END_IF;

   2: //step 2
      //action
      Q2 := true;

      //transition
      IF B2 THEN
         stepnumber := 3;          //next step
      END_IF;

   3: //step 3
      //action
      Q3 := B5;                    //set action as long as B5 is active

      //transition
      IF B3 THEN
         Q3 := false;              //reset action
         stepnumber := 4;          //next step
      END_IF;

   4: //step 4
      //action
      Q2 := false;                 //reset action

      //transition
      IF B4 THEN
         stepnumber := 1;          //jump to step 1
      END_IF;
END_CASE;
```

Picture 15 Programming step chain

12