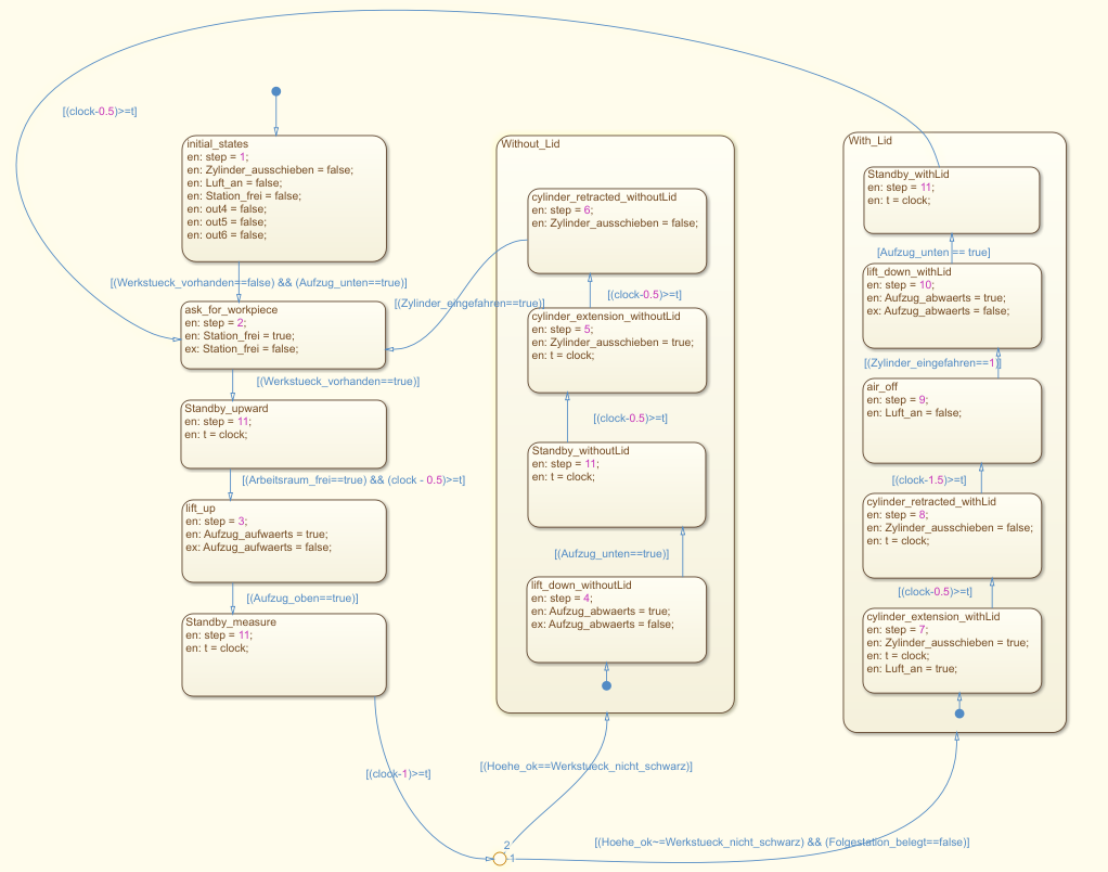
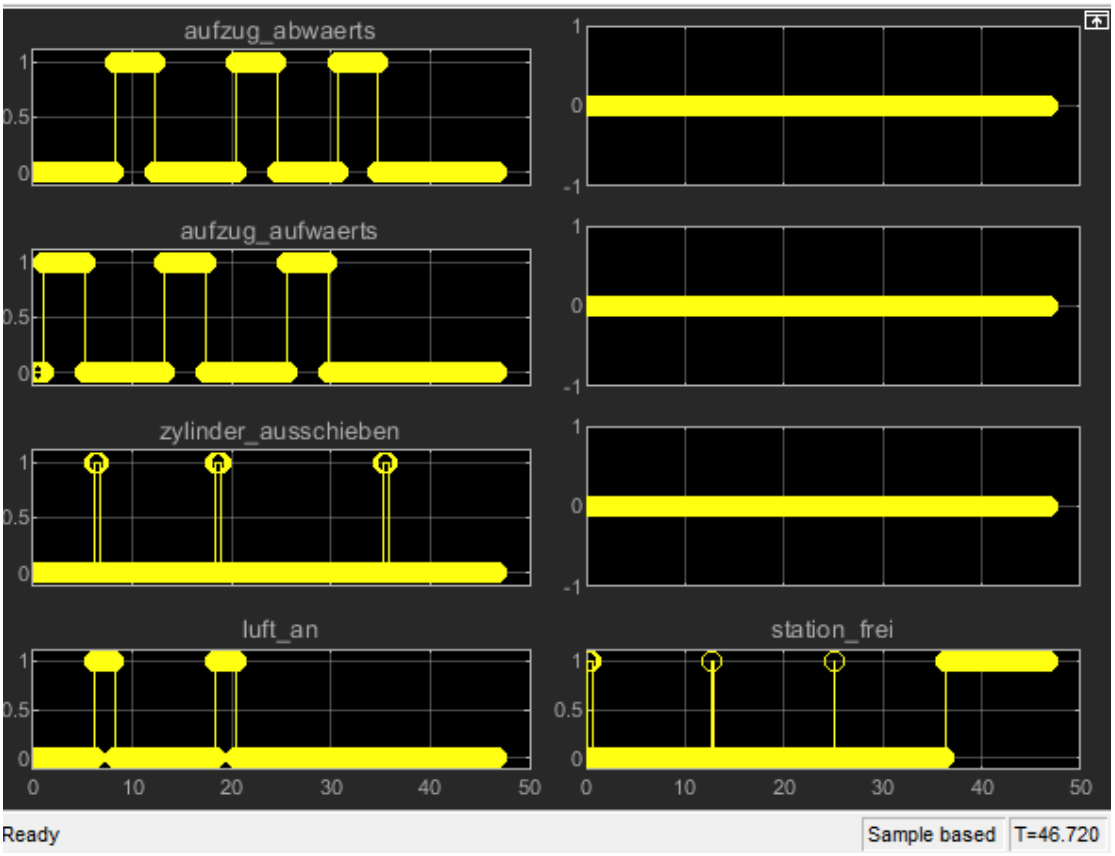
Topic 2

Task 1 Stateflow



According to the description of the control system, i divided the entire „testing“ task into multiple small parts, for example: initial_state, ask_for_workpiece, lift_up u.s.w.. Each part ist described by a „state“ Block. The output value is in the „state“, and the input value is used as the condition of the „transition“. All the functions of „time waiting“ in the description are expressed in the form of the combination of definitions „t = clock“ and „(clock - x) > t“. In order to make the conditions in some states or transitions

not too complicated, some of the „time waiting“ functions are listed separately, represented by the state of „standby_x, and these „standby_x“ states occupy the step 11 together. In addition, because there is a „fork“ in the „testing“ process, i.e. the workpieces are assigned to the air-cushion chute or reject chute according to different conditions, so the „connective junction“ block is used. To make the system more intuitive, the following two cases (with_Lid or without_Lid) are programmed in two „or state“, what is not a necessary structure.



The simulation results are basically the same as those given in the document_topic2. Because some

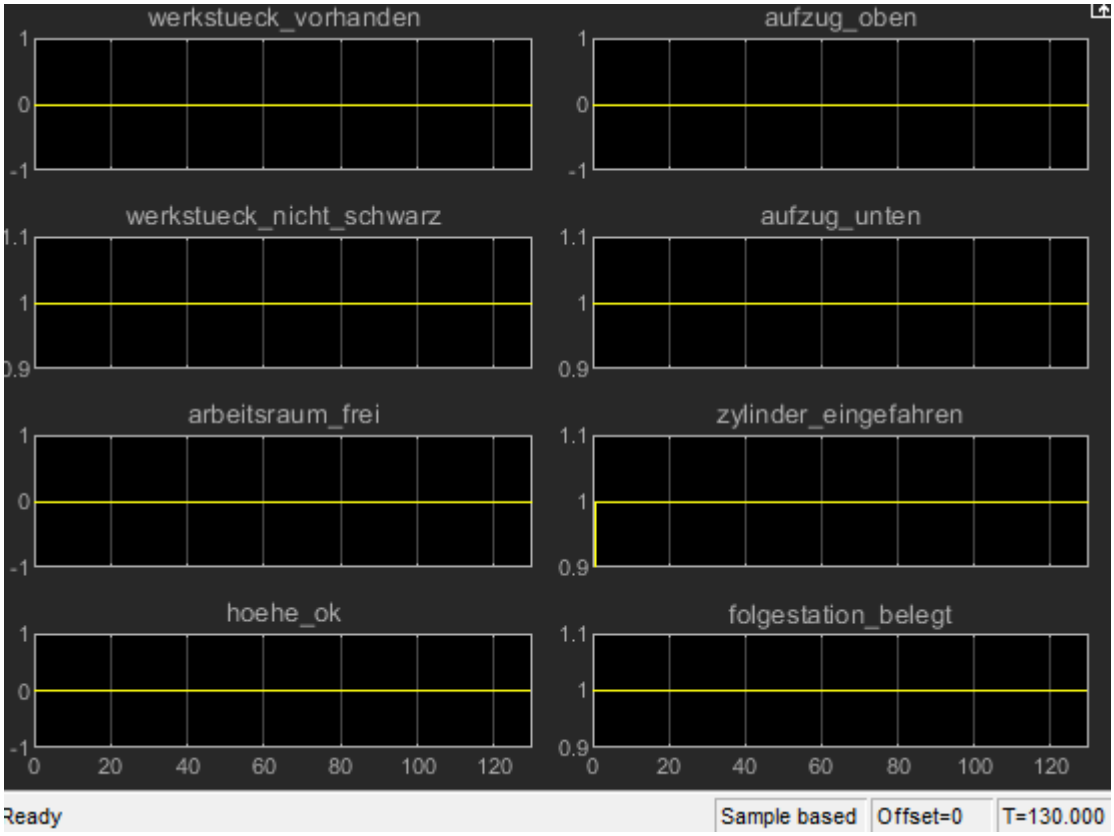
waiting times are not precisely defined, I tried to make some small changes to the values of these time (0.05s/0.1s/0.5s), and these changes have only little effect on the simulation results.

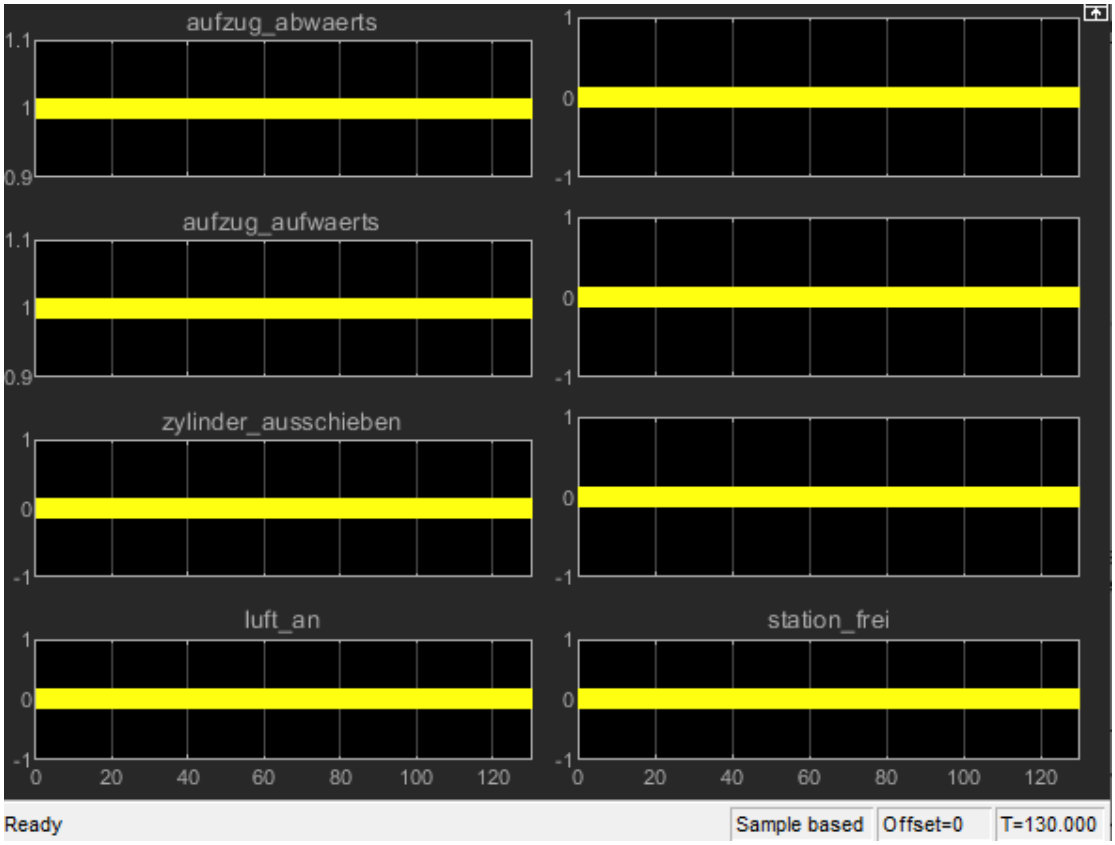
Task 2 Codesys programming

```
IF f = 0 THEN
    f := 1;
    a_zylinder_ausschieben := FALSE;
    a_luft_an := FALSE;
    a_station_frei := FALSE;
    a_aufzug_abwaerts := FALSE;
    a_aufzug_aufwaerts := FALSE;
END_IF

CASE f OF
 1: (*ask_for_workpiese*)
    a_zylinder_ausschieben := FALSE;
    a_Luft_an := FALSE;
    IF ( NOT e_werkstueck_vorhanden) AND e_aufzug_unten THEN
        a_station_frei := TRUE;
    END_IF
    f := 2;

 2: IF e_werkstueck_vorhanden THEN
        a_station_frei := FALSE;
        f := 3;
    END_IF

 3: (*Aufzug aufwaerts*)
    a_aufzug_aufwaerts := TRUE;
    IF e_aufzug_oben THEN
       a_aufzug_aufwaerts := FALSE;
       f := 4;
    END_IF

 4: (*Measure*)
    Timer(PT:=T#1S,IN:=TRUE);
    IF (e_hoehe_ok <> e_werkstueck_nicht_schwarz) AND (NOT e_folgestation_frei) THEN
        mit_Deckel := TRUE;
        a_zylinder_ausschieben := TRUE;
        a_luft_an := TRUE;
        f := 5;
    ELSE
        IF e_hoehe_ok = e_werkstueck_nicht_schwarz THEN
            mit_Deckel := FALSE;
            a_aufzug_abwaerts := TRUE;
            f := 6;
        END_IF
    END IF
```

```
5: (*Mit Deckel*)
   Timer(PT:=T#0.5S,IN:=TRUE);
   a_zylinder_ausschieben := FALSE;
   Timer(PT:=T#1.5S,IN:=TRUE);
   a_luft_an := FALSE;
   Timer(PT:=T#0.5S,IN:=TRUE);
   IF e_zylinder_eingefahren THEN
       a_aufzug_abwaerts := TRUE;
       IF e_aufzug_unten THEN
          a_aufzug_abwaerts := FALSE;
       END_IF
   END_IF
   Timer(PT:=T#0.5S,IN:=TRUE);
   f := 1;

6:  (*Ohne Deckel*)
   IF e_aufzug_unten THEN
       a_aufzug_abwaerts := FALSE;
       a_zylinder_ausschieben := TRUE;
   END_IF
   Timer(PT:=T#0.5S,IN:=TRUE);
   a_zylinder_ausschieben := FALSE;
   Timer(PT:=T#0.5S,IN:=TRUE);
   f := 1;
END_CASE
```

For codesys programming, I have not got the same results as stateflow after repeated simulations.





Device | GVL | PLC_PRG | **Steuerung**

Device.Application.PLC_PRG.strg

Expression	Type	Value
e_werkstueck_vorhanden	BOOL	FALSE
e_werkstueck_nicht_schwarz	BOOL	FALSE
e_arbeitsraum_frei	BOOL	FALSE
e_hoehe_ok	BOOL	FALSE
e_aufzug_oben	BOOL	FALSE
e_aufzug_unten	BOOL	FALSE
e_zylinder_eingefahren	BOOL	FALSE
e_folgestation_frei	BOOL	FALSE
a_aufzug_abwaerts	BOOL	TRUE
a_aufzug_aufwaerts	BOOL	TRUE
a_zylinder_ausschieben	BOOL	FALSE
a_luft_an	BOOL	FALSE
a_station_frei	BOOL	FALSE
f	SINT	2
mit_Deckel	BOOL	FALSE
Timer	TON	

There are no changes in input and output during the simulation. It can be seen from the simulation

results that the „folgestation_belegt“ in the input signal is always 1, what makes the entire process to be unable to proceed, and the values of „aufzug_abwaerts“ and „aufzug_aufwaerts“ in the output signal are both always 1, which does not conform to the actual situation. In addition, in the above program, I use the given variable *f* as the identifier of each part of the process/program, which is similar to the *step* in stateflow, and to point to each part through the *case* function, but in the result, f is always stuck in f = 2 position.

Compared with stateflow, I think the possible reason is that the Timer (TON) function is not used correctly, or I have not configured the CODESYS program correctly, or the given variables in stateflow and codesys are different (folgestation_frei/belegt, no out4/5/6).

I have spent more than 1 day on debugging, but I still can’t get the correct results, so I plan to complete the following tasks first. If there is still enough time in the end, I will come back and try to continue to optimize this program.

Task 3 Code generation

```
(*
 *
 * File: Stateflow_code.st
 *
 * IEC 61131-3 Structured Text (ST) code generated for subsystem
"Stateflow_code/Pruefen_Angabe"
 *
 * Model name                     : Stateflow_code
 * Model version                  : 1.388
 * Model creator                  : Jan
 * Model last modified by         : LENOVO
 * Model last modified on         : Tue Sep 15 19:28:20 2020
 * Model sample time              : 0.01s
 * Subsystem name                 : Stateflow_code/Pruefen_Angabe
 * Subsystem sample time          : 0.01s
 * Simulink PLC Coder version     : 3.2 (R2020a) 18-Nov-2019
 * ST code generated on           : Tue Sep 15 20:54:59 2020
 *
 * Target IDE selection           : 3S CoDeSys 3.5
 * Test Bench included            : No
 *
 *)
FUNCTION_BLOCK Pruefen_Angabe
VAR_INPUT
    ssMethodType: SINT;
    werkstueck_vorhanden: BOOL;
    werkstueck_nicht_schwarz: BOOL;
    arbeitsraum_frei: BOOL;
    hoehe_ok: BOOL;
    aufzug_oben: BOOL;
    aufzug_unten: BOOL;
    zylinder_eingefahren: BOOL;
    Folgestation_belegt: BOOL;
    clock: LREAL;
END_VAR
VAR_OUTPUT
    Aufzug_abwaerts: BOOL;
    Aufzug_aufwaerts: BOOL;
    Zylinder_ausschieben: BOOL;
    Luft_an: BOOL;
    out4: BOOL;
    out5: BOOL;
```

```
42      out6: BOOL;
43      Station_frei: BOOL;
44      b_step: LREAL;
45  END_VAR
46  VAR
47      is_active_c3_Pruefen_Angabe: USINT;
48      is_c3_Pruefen_Angabe: USINT;
49      t: LREAL;
50      is_With_Lid: USINT;
51      is_Without_Lid: USINT;
52  END_VAR
53  CASE ssMethodType OF
54      SS_INITIALIZE:
55          (* SystemInitialize for Chart: '<Root>/Pruefen_Angabe' *)
56          is_With_Lid := c_Pruefen_Anga_IN_NO_ACTIVE;
57          is_Without_Lid := c_Pruefen_Anga_IN_NO_ACTIVE;
58          is_active_c3_Pruefen_Angabe := 0;
59          is_c3_Pruefen_Angabe := c_Pruefen_Anga_IN_NO_ACTIVE;
60      SS_STEP:
61          (* Chart: '<Root>/Pruefen_Angabe' incorporates:
62           *  Outport: '<Root>/Aufzug_abwaerts'
63           *  Outport: '<Root>/Aufzug_aufwaerts'
64           *  Outport: '<Root>/Luft_an'
65           *  Outport: '<Root>/Station_frei'
66           *  Outport: '<Root>/Zylinder_ausschieben'
67           *  Outport: '<Root>/out4'
68           *  Outport: '<Root>/out5'
69           *  Outport: '<Root>/out6' *)
70          (* Gateway: Pruefen_Angabe *)
71          (* During: Pruefen_Angabe *)
72          IF is_active_c3_Pruefen_Angabe = 0 THEN
73              (* Entry: Pruefen_Angabe *)
74              is_active_c3_Pruefen_Angabe := 1;
75              (* Entry Internal: Pruefen_Angabe *)
76              (* Transition: '<S1>:91' *)
77              is_c3_Pruefen_Angabe := c_Pruefen_Angab_IN_initial_;
78              (* Outport: '<Root>/step' *)
79              (* Entry 'initial_states': '<S1>:66' *)
80              (* '<S1>:66:2' step = 1; *)
81              b_step := 1.0;
82              (* '<S1>:66:3' Zylinder_ausschieben = false; *)
83              Zylinder_ausschieben := FALSE;
84              (* '<S1>:66:4' Luft_an = false; *)
85              Luft_an := FALSE;
```

```
86              (* '<S1>:66:5' Station_frei = false; *)
87              Station_frei := FALSE;
88              (* '<S1>:66:6' out4 = false; *)
89              out4 := FALSE;
90              (* '<S1>:66:7' out5 = false; *)
91              out5 := FALSE;
92              (* '<S1>:66:8' out6 = false; *)
93              out6 := FALSE;
94          ELSE
95              CASE is_c3_Pruefen_Angabe OF
96                  c_Pruefen_Anga_IN_Standby_m:
97                      (* Outport: '<Root>/step' *)
98                      b_step := 11.0;
99                      (* During 'Standby_measure': '<S1>:96' *)
100                     (* '<S1>:83:1' sf_internal_predicateOutput = (clock-1)>=t; *)
101                     IF (clock - 1.0) >= t THEN
102                         (* Transition: '<S1>:83' *)
103                         (* '<S1>:94:1' sf_internal_predicateOutput =
(Hoehe_ok~=Werkstueck_nicht_schwarz) && (Folgestation_belegt==false); *)
104                         IF (hoehe_ok <> werkstueck_nicht_schwarz) AND ( NOT
Folgestation_belegt) THEN
105                             (* Transition: '<S1>:94' *)
106                             is_c3_Pruefen_Angabe := Pruefen_Angabe_IN_With_Lid;
107                             (* Entry Internal 'With_Lid': '<S1>:63' *)
108                             (* Transition: '<S1>:95' *)
109                             is_With_Lid := c_P_IN_cylinder_extension_w;
110                             (* Outport: '<Root>/step' *)
111                             (* Entry 'cylinder_extension_withLid': '<S1>:82' *)
112                             (* '<S1>:82:2' step = 7; *)
113                             b_step := 7.0;
114                             (* '<S1>:82:3' Zylinder_ausschieben = true; *)
115                             Zylinder_ausschieben := TRUE;
116                             (* '<S1>:82:4' t = clock; *)
117                             t := clock;
118                             (* '<S1>:82:5' Luft_an = true; *)
119                             Luft_an := TRUE;
120                         ELSE
121                             (* '<S1>:90:1' sf_internal_predicateOutput =
(Hoehe_ok==Werkstueck_nicht_schwarz); *)
122                             IF hoehe_ok = werkstueck_nicht_schwarz THEN
123                                 (* Transition: '<S1>:90' *)
124                                 is_c3_Pruefen_Angabe := c_Pruefen_Angabe_IN_Without;
125                                 (* Entry 'Without_Lid': '<S1>:72' *)
126                                 (* Entry Internal 'Without_Lid': '<S1>:72' *)
```

```
                                (* Transition: '<S1>:61' *)
                                is_Without_Lid := c_Pruefen_IN_lift_down_with;
                                (* Outport: '<Root>/step' *)
                                (* Entry 'lift_down_withoutLid': '<S1>:73' *)
                                (* '<S1>:73:2' step = 4; *)
                                b_step := 4.0;
                                (* '<S1>:73:3' Aufzug_abwaerts = true; *)
                                Aufzug_abwaerts := TRUE;
                            END_IF;
                        END_IF;
                    END_IF;
                c_Pruefen_Angab_IN_Standby_:
                    (* Outport: '<Root>/step' *)
                    b_step := 11.0;
                    (* During 'Standby_upward': '<S1>:62' *)
                    (* '<S1>:87:1' sf_internal_predicateOutput = (Arbeitsraum_frei==true) &&
(clock - 0.5)>=t; *)
                    IF arbeitsraum_frei AND ((clock - 0.5) >= t) THEN
                        (* Transition: '<S1>:87' *)
                        is_c3_Pruefen_Angabe := Pruefen_Angabe_IN_lift_up;
                        (* Outport: '<Root>/step' *)
                        (* Entry 'lift_up': '<S1>:71' *)
                        (* '<S1>:71:2' step = 3; *)
                        b_step := 3.0;
                        (* '<S1>:71:3' Aufzug_aufwaerts = true; *)
                        Aufzug_aufwaerts := TRUE;
                    END_IF;
                Pruefen_Angabe_IN_With_Lid:
                    (* During 'With_Lid': '<S1>:63' *)
                    CASE is_With_Lid OF
                        c_Pruefen_Anga_IN_Standby_w:
                            (* Outport: '<Root>/step' *)
                            b_step := 11.0;
                            (* During 'Standby_withLid': '<S1>:93' *)
                            (* '<S1>:79:1' sf_internal_predicateOutput = (clock-0.5)>=t; *)
                            IF (clock - 0.5) >= t THEN
                                (* Transition: '<S1>:79' *)
                                is_With_Lid := c_Pruefen_Anga_IN_NO_ACTIVE;
                                is_c3_Pruefen_Angabe := c_Pruefen_An_IN_ask_for_wor;
                                (* Outport: '<Root>/step' *)
                                (* Entry 'ask_for_workpiece': '<S1>:64' *)
                                (* '<S1>:64:2' step = 2; *)
                                b_step := 2.0;
                                (* '<S1>:64:3' Station_frei = true; *)
```

```
170                                 Station_frei := TRUE;
171                             END_IF;
172                         Pruefen_Angabe_IN_air_off:
173                             (* Outport: '<Root>/step' *)
174                             b_step := 9.0;
175                             Luft_an := FALSE;
176                             (* During 'air_off': '<S1>:86' *)
177                             (* '<S1>:78:1' sf_internal_predicateOutput =
(Zylinder_eingefahren==1); *)
178                             IF zylinder_eingefahren THEN
179                                 (* Transition: '<S1>:78' *)
180                                 is_With_Lid := c_Pruefen_An_IN_lift_down_w;
181                                 (* Outport: '<Root>/step' *)
182                                 (* Entry 'lift_down_withLid': '<S1>:80' *)
183                                 (* '<S1>:80:2' step = 10; *)
184                                 b_step := 10.0;
185                                 (* '<S1>:80:3' Aufzug_abwaerts = true; *)
186                                 Aufzug_abwaerts := TRUE;
187                             END_IF;
188                         c_P_IN_cylinder_extension_w:
189                             (* Outport: '<Root>/step' *)
190                             b_step := 7.0;
191                             Zylinder_ausschieben := TRUE;
192                             Luft_an := TRUE;
193                             (* During 'cylinder_extension_withLid': '<S1>:82' *)
194                             (* '<S1>:88:1' sf_internal_predicateOutput = (clock-0.5)>=t; *)
195                             IF (clock - 0.5) >= t THEN
196                                 (* Transition: '<S1>:88' *)
197                                 is_With_Lid := c_P_IN_cylinder_retracted_w;
198                                 (* Outport: '<Root>/step' *)
199                                 (* Entry 'cylinder_retracted_withLid': '<S1>:92' *)
200                                 (* '<S1>:92:2' step = 8; *)
201                                 b_step := 8.0;
202                                 (* '<S1>:92:3' Zylinder_ausschieben = false; *)
203                                 Zylinder_ausschieben := FALSE;
204                                 (* '<S1>:92:4' t = clock; *)
205                                 t := clock;
206                             END_IF;
207                         c_P_IN_cylinder_retracted_w:
208                             (* Outport: '<Root>/step' *)
209                             b_step := 8.0;
210                             Zylinder_ausschieben := FALSE;
211                             (* During 'cylinder_retracted_withLid': '<S1>:92' *)
212                             (* '<S1>:75:1' sf_internal_predicateOutput = (clock-1.5)>=t; *)
```

```
                        IF (clock - 1.5) >= t THEN
                            (* Transition: '<S1>:75' *)
                            is_With_Lid := Pruefen_Angabe_IN_air_off;
                            (* Outport: '<Root>/step' *)
                            (* Entry 'air_off': '<S1>:86' *)
                            (* '<S1>:86:2' step = 9; *)
                            b_step := 9.0;
                            (* '<S1>:86:3' Luft_an = false; *)
                            Luft_an := FALSE;
                        END_IF;
                    ELSE
                        (* Outport: '<Root>/step' *)
                        b_step := 10.0;
                        (* During 'lift_down_withLid': '<S1>:80' *)
                        (* '<S1>:76:1' sf_internal_predicateOutput = Aufzug_unten == true;
*)
                        IF aufzug_unten THEN
                            (* Transition: '<S1>:76' *)
                            (* Exit 'lift_down_withLid': '<S1>:80' *)
                            (* '<S1>:80:4' Aufzug_abwaerts = false; *)
                            Aufzug_abwaerts := FALSE;
                            is_With_Lid := c_Pruefen_Anga_IN_Standby_w;
                            (* Outport: '<Root>/step' incorporates:
                             *  Outport: '<Root>/Aufzug_abwaerts' *)
                            (* Entry 'Standby_withLid': '<S1>:93' *)
                            (* '<S1>:93:2' step = 11; *)
                            b_step := 11.0;
                            (* '<S1>:93:3' t = clock; *)
                            t := clock;
                        END_IF;
                END_CASE;
            c_Pruefen_Angabe_IN_Without:
                (* During 'Without_Lid': '<S1>:72' *)
                CASE is_Without_Lid OF
                    c_Pruefen_A_IN_Standby_with:
                        (* Outport: '<Root>/step' *)
                        b_step := 11.0;
                        (* During 'Standby_withoutLid': '<S1>:68' *)
                        (* '<S1>:70:1' sf_internal_predicateOutput = (clock-0.5)>=t; *)
                        IF (clock - 0.5) >= t THEN
                            (* Transition: '<S1>:70' *)
                            is_Without_Lid := c_IN_cylinder_extension_wit;
                            (* Outport: '<Root>/step' *)
                            (* Entry 'cylinder_extension_withoutLid': '<S1>:65' *)
```

```
256                                 (* '<S1>:65:2' step = 5; *)
257                                 b_step := 5.0;
258                                 (* '<S1>:65:3' Zylinder_ausschieben = true; *)
259                                 Zylinder_ausschieben := TRUE;
260                                 (* '<S1>:65:4' t = clock; *)
261                                 t := clock;
262                             END_IF;
263                         c_IN_cylinder_extension_wit:
264                             (* Outport: '<Root>/step' *)
265                             b_step := 5.0;
266                             Zylinder_ausschieben := TRUE;
267                             (* During 'cylinder_extension_withoutLid': '<S1>:65' *)
268                             (* '<S1>:67:1' sf_internal_predicateOutput = (clock-0.5)>=t; *)
269                             IF (clock - 0.5) >= t THEN
270                                 (* Transition: '<S1>:67' *)
271                                 is_Without_Lid := c_IN_cylinder_retracted_wit;
272                                 (* Outport: '<Root>/step' *)
273                                 (* Entry 'cylinder_retracted_withoutLid': '<S1>:69' *)
274                                 (* '<S1>:69:2' step = 6; *)
275                                 b_step := 6.0;
276                                 (* '<S1>:69:3' Zylinder_ausschieben = false; *)
277                                 Zylinder_ausschieben := FALSE;
278                             END_IF;
279                         c_IN_cylinder_retracted_wit:
280                             (* Outport: '<Root>/step' *)
281                             b_step := 6.0;
282                             Zylinder_ausschieben := FALSE;
283                             (* During 'cylinder_retracted_withoutLid': '<S1>:69' *)
284                             (* '<S1>:77:1' sf_internal_predicateOutput =
(Zylinder_eingefahren==true); *)
285                             IF zylinder_eingefahren THEN
286                                 (* Transition: '<S1>:77' *)
287                                 is_Without_Lid := c_Pruefen_Anga_IN_NO_ACTIVE;
288                                 is_c3_Pruefen_Angabe := c_Pruefen_An_IN_ask_for_wor;
289                                 (* Outport: '<Root>/step' *)
290                                 (* Entry 'ask_for_workpiece': '<S1>:64' *)
291                                 (* '<S1>:64:2' step = 2; *)
292                                 b_step := 2.0;
293                                 (* '<S1>:64:3' Station_frei = true; *)
294                                 Station_frei := TRUE;
295                             END_IF;
296                         ELSE
297                             (* Outport: '<Root>/step' *)
298                             b_step := 4.0;
```

```
                        (* During 'lift_down_withoutLid': '<S1>:73' *)
                        (* '<S1>:74:1' sf_internal_predicateOutput = (Aufzug_unten==true);
*)
                        IF aufzug_unten THEN
                            (* Transition: '<S1>:74' *)
                            (* Exit 'lift_down_withoutLid': '<S1>:73' *)
                            (* '<S1>:73:4' Aufzug_abwaerts = false; *)
                            Aufzug_abwaerts := FALSE;
                            is_Without_Lid := c_Pruefen_A_IN_Standby_with;
                            (* Outport: '<Root>/step' incorporates:
                             *  Outport: '<Root>/Aufzug_abwaerts' *)
                            (* Entry 'Standby_withoutLid': '<S1>:68' *)
                            (* '<S1>:68:2' step = 11; *)
                            b_step := 11.0;
                            (* '<S1>:68:3' t = clock; *)
                            t := clock;
                        END_IF;
                    END_CASE;
                c_Pruefen_An_IN_ask_for_wor:
                    (* Outport: '<Root>/step' *)
                    b_step := 2.0;
                    (* During 'ask_for_workpiece': '<S1>:64' *)
                    (* '<S1>:84:1' sf_internal_predicateOutput = (Werkstueck_vorhanden==true);
*)
                    IF werkstueck_vorhanden THEN
                        (* Transition: '<S1>:84' *)
                        (* Exit 'ask_for_workpiece': '<S1>:64' *)
                        (* '<S1>:64:4' Station_frei = false; *)
                        Station_frei := FALSE;
                        is_c3_Pruefen_Angabe := c_Pruefen_Angab_IN_Standby_;
                        (* Outport: '<Root>/step' incorporates:
                         *  Outport: '<Root>/Station_frei' *)
                        (* Entry 'Standby_upward': '<S1>:62' *)
                        (* '<S1>:62:2' step = 11; *)
                        b_step := 11.0;
                        (* '<S1>:62:3' t = clock; *)
                        t := clock;
                    END_IF;
                c_Pruefen_Angab_IN_initial_:
                    (* Outport: '<Root>/step' *)
                    b_step := 1.0;
                    Zylinder_ausschieben := FALSE;
                    Luft_an := FALSE;
                    out4 := FALSE;
```

```
341                         out5 := FALSE;
342                         out6 := FALSE;
343                         (* During 'initial_states': '<S1>:66' *)
344                         (* '<S1>:85:1' sf_internal_predicateOutput = (Werkstueck_vorhanden==false)
&& (Aufzug_unten==true); *)
345                         IF ( NOT werkstueck_vorhanden) AND aufzug_unten THEN
346                             (* Transition: '<S1>:85' *)
347                             is_c3_Pruefen_Angabe := c_Pruefen_An_IN_ask_for_wor;
348                             (* Outport: '<Root>/step' *)
349                             (* Entry 'ask_for_workpiece': '<S1>:64' *)
350                             (* '<S1>:64:2' step = 2; *)
351                             b_step := 2.0;
352                             (* '<S1>:64:3' Station_frei = true; *)
353                             Station_frei := TRUE;
354                         END_IF;
355                     ELSE
356                         (* Outport: '<Root>/step' *)
357                         b_step := 3.0;
358                         (* During 'lift_up': '<S1>:71' *)
359                         (* '<S1>:81:1' sf_internal_predicateOutput = (Aufzug_oben==true); *)
360                         IF aufzug_oben THEN
361                             (* Transition: '<S1>:81' *)
362                             (* Exit 'lift_up': '<S1>:71' *)
363                             (* '<S1>:71:4' Aufzug_aufwaerts = false; *)
364                             Aufzug_aufwaerts := FALSE;
365                             is_c3_Pruefen_Angabe := c_Pruefen_Anga_IN_Standby_m;
366                             (* Outport: '<Root>/step' incorporates:
367                              *  Outport: '<Root>/Aufzug_aufwaerts' *)
368                             (* Entry 'Standby_measure': '<S1>:96' *)
369                             (* '<S1>:96:2' step = 11; *)
370                             b_step := 11.0;
371                             (* '<S1>:96:3' t = clock; *)
372                             t := clock;
373                         END_IF;
374                 END_CASE;
375             END_IF;
376             (* End of Chart: '<Root>/Pruefen_Angabe' *)
377     END_CASE;
378     END_FUNCTION_BLOCK
379     VAR_GLOBAL CONSTANT
380         c_Pruefen_Anga_IN_NO_ACTIVE: USINT := 0;
381         c_Pruefen_Anga_IN_Standby_m: USINT := 1;
382         c_Pruefen_Angab_IN_Standby_: USINT := 2;
383         Pruefen_Angabe_IN_With_Lid: USINT := 3;
```

```
384       c_Pruefen_Angabe_IN_Without: USINT := 4;
385       c_Pruefen_An_IN_ask_for_wor: USINT := 5;
386       c_Pruefen_Angab_IN_initial_: USINT := 6;
387       Pruefen_Angabe_IN_lift_up: USINT := 7;
388       c_Pruefen_Anga_IN_Standby_w: USINT := 1;
389       Pruefen_Angabe_IN_air_off: USINT := 2;
390       c_P_IN_cylinder_extension_w: USINT := 3;
391       c_P_IN_cylinder_retracted_w: USINT := 4;
392       c_Pruefen_An_IN_lift_down_w: USINT := 5;
393       c_Pruefen_A_IN_Standby_with: USINT := 1;
394       c_IN_cylinder_extension_wit: USINT := 2;
395       c_IN_cylinder_retracted_wit: USINT := 3;
396       c_Pruefen_IN_lift_down_with: USINT := 4;
397       SS_INITIALIZE: SINT := 0;
398       SS_STEP: SINT := 1;
399   END_VAR
400
```

Comparing the PLC code, which is generated by stateflow and the codesys code, which is programmed by myself, it can be found that the basic structure of two programs are similar: The *case* function is used to point to different parts of the „testing“ system, and the conditions in each part are expressed by *if* function. The difference is that the generated PLC code does not use the TON function to represent time but use the command "t:=clock" and "(clock-x)>t", which is similar to stateflow.