

Automatické třídění na dopravníku

Automated Conveyor Sorting · Structured Text source

Verze zdroje v3.25.1

Build 10 / 2026

CODESYS · ST · IEC 61131-3

Datové typy

Globální proměnné

Funkce

Funkční bloky

PLC_PRG_ST

Kompletní zdrojový kód řízení třídící linky – datové typy, globální proměnné, funkce, funkční bloky a hlavní program, ve stejném členění jako projekt CODESYS.

Co má linka dělat, popisuje funkční specifikace **FSD v3.25.1**; tento dokument ukazuje, jak je to naprogramované. Komentáře popisují jen aktuální chování, historie verzí je v [CHANGELOG.md](#).

ŘÍZENÍ
CODESYS V3.5 SP22

SIMULACE
Factory I/O 2.5.9

HMI
HMI_Control

AUTOR
David Neumann

VYDÁNÍ 1.0 · 10 / 2026

OBSAH

1	Přehled zdrojového kódu	02
	1.1 Kód v číslech	02
	1.2 Pořadí scanu a volané bloky	02
	1.3 Přehled POU	03
	1.4 Konvence a čtení výpisu	04
2	Datové typy	05
	E_Mode ... ST_AlarmEntry	
3	Globální proměnné	08
	GVL ... GVL_HMI	
4	Funkce	15
	F_ClassifyWeight ... F_FormatAlarmTime	
5	Funkční bloky	20
	FB_SafetyInterlock ... FB_AlarmHistory	
6	Hlavní program	40

KAPITOLA 1

Přehled zdrojového kódu

Source overview

Program je napsaný ve Structured Text podle IEC 61131-3 a běží v CODESYS Control Win V3.5 SP22; Factory I/O 2.5.9 simuluje stroj a vystupuje jako OPC server. Kód je rozdělený do 24 objektů (POU) – ve výpisu jsou ve stejném pořadí jako v projektu.

1.1 Kód v číslech

24

objektů POU v projektu

11

funkčních bloků

5

funkcí

5

datových typů · 4
ENUM, 1
STRUCT

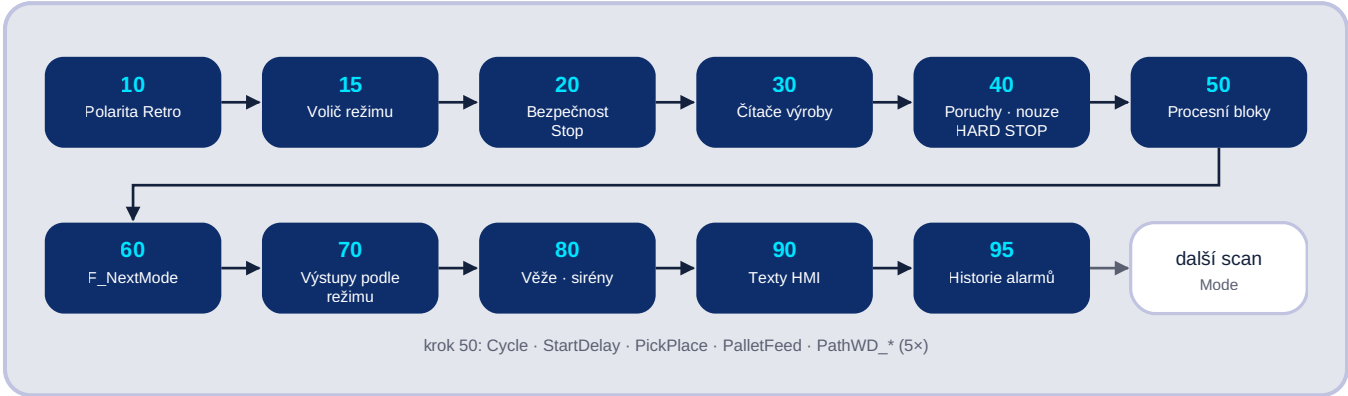
2

globální listy
proměnných

1 913

řádků kódu bez
komentářů

1.2 Pořadí scanu a volané bloky



Hlavní program `PLC_PRG_ST` prochází kroky 10–95 v každém scanu ve stejném pořadí; kroky jsou v kódu označené hlavičkami.

KROK	CO SE DĚJE	VOLANÉ POU
10	Retro_S1..S4 (light-operate) převedeny na *_Present.	–
15	Dvoupolohový volič Auto / Manual z HMI.	–
20	Sloučení tlačítek s HMI, hrany Reset / Drain, západka Stop.	FB_SafetyInterlock
30	Šest čítačů výroby, režim ze začátku scanu.	FB_ProductionCounter × 6
40	Poruchy watchdogů, HARD STOP a vynucení EMERGENCY → ModeThisScan.	–
50	Procesní bloky v režimu AUTO a DRAINING.	FB_SortingCycle · FB_StartDelay · FB_PickPlace · FB_PalletFeed · FB_PathWatchdog × 5
60	Režim pro další scan.	F_NextMode
70	Výstupy podle ModeThisScan, HARD STOP větví EMERGENCY.	–
80	Světelné věže a sirény.	F_TowerColors · FB_Blink · FB_PulseAlarm × 5 · FB_TimedAlarm
90	Stavové texty pro HMI.	–
95	Historie alarmů – pozorovatel, nic nezapisuje zpět.	FB_AlarmHistory · F_FormatAlarmTime

1.3 Přehled POU

POU	TYP	ÚČEL	FSD	STR.
DATOVÉ TYPY				
E_Mode	ENUM	Provozní režim linky	kap. 3, 9	05
E_SortState	ENUM	Stavy třídícího cyklu	kap. 5	05
E_WeightClass	ENUM	Hmotnostní třída S / M / L	kap. 6	06
E_PickPlaceState	ENUM	Stavy manipulátoru PnP1	kap. 7	06
ST_AlarmEntry	STRUCT	Řádek historie alarmů	kap. 13.3	07
GLOBÁLNÍ PROMĚNNÉ				
GVL	GVL	I/O, čítače, alarmy, zobrazení	kap. 2.3, 2.5 · příl. A	08
GVL_HMI	GVL	Povely obsluhy z HMI	kap. 12, 13 · příl. A.6	13
FUNKCE				
F_ClassifyWeight	Funkce	Klasifikace hmotnosti	kap. 6	15
F_NextMode	Funkce	Přechodová tabulka režimů	kap. 3.1, 9	15
F_TowerColors	Funkce	Barvy světelných věží	kap. 4.1	17
F_Pad2	Funkce	Číslo na dva znaky	kap. 13.3	18
F_FormatAlarmTime	Funkce	Formát času alarmu	kap. 13.3	18
FUNKČNÍ BLOKY				
FB_SafetyInterlock	Funkční blok	Nouze, dveře, západka Stop	kap. 10	20
FB_ProductionCounter	Funkční blok	Čítač událostí	kap. 14	21
FB_SortingCycle	Funkční blok	Třídící cyklus	kap. 5, 6, 11	21
FB_PickPlace	Funkční blok	Manipulátor PnP1	kap. 7	27
FB_PalletFeed	Funkční blok	Paletový přívod	kap. 8, 9	29
FB_PathWatchdog	Funkční blok	Hlídnání úseku (FIFO)	kap. 11	32
FB_StartDelay	Funkční blok	Zpoždění startu 3 s	kap. 3.1	33
FB_Blink	Funkční blok	Blikání věží	kap. 4.1	34
FB_PulseAlarm	Funkční blok	Siréna – pípnutí	kap. 4.2	35
FB_TimedAlarm	Funkční blok	Siréna – souvislý tón	kap. 4.2	36
FB_AlarmHistory	Funkční blok	Historie alarmů	kap. 13.3	36
HLAVNÍ PROGRAM				
PLC_PRG_ST	Program	Hlavní program	kap. 3, 4, 9 · příl. C	40

1.4 Konvence a čtení výpisu

VZOR	VÝZNAM	PŘÍKLAD
E_ · ST_	Výčtový typ (qualified_only) · struktura	E_Mode.MODE_AUTO
F_	Funkce – bez vnitřního stavu	F_NextMode
FB_	Funkční blok; instance bez předpony	FB_PathWatchdog → PathWD_S
GVL.	Fyzická I/O, čítače, zobrazovací proměnné	GVL.Retro_S1
*_Cmd_HMI	Povel z HMI, nikdy namapovaný na I/O	Start_Cmd_HMI
Man_*	Ruční vstup obsluhy pro MANUAL	Man_Conv2
Dis_* · CounterDis_*	Zobrazovací proměnná, čte jen HMI	Dis_SystemText
Raw* · *_Present	Surový vstup a jeho opravená podoba	Retro_S1_Present
R_*	Detektor náběžné hrany (R_TRIG)	R_Reset
T_* · *Tmr	Časová konstanta · časovač TON	T_STOP_DRAIN · DrainTmr
Fault_* · InFlight_*	Porucha zdroje · počet kusů na úseku	Fault_ExitPath_S

DEKLARACE A IMPLEMENTACE

Každé POU je rozdělené jako v editoru CODESYS: deklarace (hlavička, proměnné) a implementace, obě s vlastním číslováním řádků.

HLAVIČKA POU

Komentář **PURPOSE** · **NOTES** · **FSD REF** : účel, nezřejmé souvislosti a kapitoly (ch.) či přílohy (app.) funkční specifikace.

BARVY VÝPISU

Klíčová slova tučně, **typy** navy, komentáře šedě. Komentáře jsou anglicky.

Rozsah: bezpečnostní funkce jsou v projektu realizovány standardní logikou PLC v simulaci. Reálná instalace vyžaduje certifikovaný bezpečnostní obvod podle požadované úrovně PL / SIL (FSD kap. 10).

KAPITOLA 2

| Datové typy Data types · DUT

Čtyři výčtové typy pro stavy linky a cyklů a jedna struktura pro historii alarmů. Výčty mají atribut `qualified_only` – hodnota se vždy píše s názvem typu, například `E_Mode.MODE_AUTO`.

2.1 E_Mode

ENUM · 6 HODNOT · 12 ŘÁDKŮ KÓDU · FSD KAP. 3, 9

Šest provozních režimů linky. Drží je jediná proměnná `Mode` v `PLC_PRG_ST`; HARD STOP není samostatný režim.

DEKLARACE · DECLARATION

26 ř.

```

1 {attribute 'qualified_only'}
2 {attribute 'strict'}
3 (* =====
4   DUT (ENUM) : E_Mode
5   PURPOSE : Operating mode of the line. Held in one variable only,
6             PLC_PRG_ST.Mode; transitions are computed by F_NextMode or
7             forced in PLC_PRG_ST step 40 (emergency, HARD STOP).
8   NOTES  : MODE_DRAINING is entered only from MODE_AUTO (Drain request)
9             and ends in MODE_DRAINED once the pallet side and the box
10            side are both confirmed empty. MODE_DRAINED is left only by
11            Reset, to MODE_STOPPED. HARD STOP is not a mode of its own:
12            it is MODE_STOPPED with emergency outputs (PLC_PRG_ST step 70).
13   FSD REF : ch. 3   Operating modes
14             ch. 9   System Drain
15   ===== *)
16
17 TYPE E_Mode :
18 (
19   MODE_STOPPED,
20   MODE_AUTO,
21   MODE_MANUAL,
22   MODE_EMERGENCY,
23   MODE_DRAINING,
24   MODE_DRAINED
25 );
26 END_TYPE

```

2.2 E_SortState

ENUM · 10 HODNOT · 16 ŘÁDKŮ KÓDU · FSD KAP. 5

Deset stavů třídícího cyklu od `IDLE` po `SORT_FAULT`; tři z nich hlídá časový limit (watchdog).

DEKLARACE · DECLARATION

25 ř.

```

1 {attribute 'qualified_only'}
2 {attribute 'strict'}
3 (* =====
4   DUT (ENUM) : E_SortState
5   PURPOSE : States of the box sorting cycle (FB_SortingCycle).
6   NOTES  : WAIT_SCALE_ENTRY, WAIT_EXIT and SORT_CONFIRM each have a
7             timeout. On expiry the cycle enters SORT_FAULT, which is held
8             until Reset.
9   FSD REF : ch. 5   Sorting cycle
10   ===== *)
11
12 TYPE E_SortState :
13 (
14   IDLE,
15   RUN,
16   WAIT_SCALE_ENTRY,
17   WEIGHING,
18   CLASSIFICATION,
19   SORTING,
20   WAIT_EXIT,
21   SORT_CONFIRM,

```

```

22     STOPPING,
23     SORT_FAULT
24 );
25 END_TYPE

```

2.3 E_WeightClass

ENUM · 4 HODNOTY · 10 ŘÁDKŮ KÓDU · FSD KAP. 6

Hmotnostní třída krabice; **CLS_NONE** mimo třídění.

DEKLARACE · DECLARATION

18 ř.

```

1 {attribute 'qualified_only'}
2 {attribute 'strict'}
3 (* =====
4   DUT (ENUM) : E_WeightClass
5   PURPOSE : Weight class of a box - small / medium / large.
6   NOTES   : CLS_NONE = no box being sorted (before classification and
7             after the lane confirmation).
8   FSD REF : ch. 6      Weight classification
9   ===== *)
10
11 TYPE E_WeightClass :
12 (
13     CLS_NONE,
14     CLS_S,
15     CLS_M,
16     CLS_L
17 );
18 END_TYPE

```

2.4 E_PickPlaceState

ENUM · 8 HODNOT · 14 ŘÁDKŮ KÓDU · FSD KAP. 7

Stavy dvouosého manipulátoru PnP1 včetně poruchy **PP_FAULT**.

DEKLARACE · DECLARATION

20 ř.

```

1 {attribute 'qualified_only'}
2 {attribute 'strict'}
3 (* =====
4   DUT (ENUM) : E_PickPlaceState
5   PURPOSE : States of the two-axis Pick & Place PnP1 (FB_PickPlace).
6   FSD REF : ch. 7      Pick & Place PnP1
7   ===== *)
8
9 TYPE E_PickPlaceState :
10 (
11     PP_HOME,
12     PP_WAIT,
13     PP_LOWER_PICK,
14     PP_GRAB,
15     PP_LIFT,
16     PP_MOVE_TO_DROP,
17     PP_RELEASE,
18     PP_FAULT
19 );
20 END_TYPE

```

2.5 ST_AlarmEntry

STRUCT · 4 POLOŽKY · 8 ŘÁDKŮ KÓDU · FSD KAP. 13.3

Jeden výskyt alarmu – číslo alarmu, čas vzniku a zániku. Text se dohledává až při zobrazení.

DEKLARACE · DECLARATION

21 ř.

```

1  (* =====
2  DUT (STRUCT) : ST_AlarmEntry
3  PURPOSE : One alarm occurrence in GVL.AlarmHist - which alarm, when it
4             came and when it went. One row per occurrence: the falling
5             edge closes the SAME row (TimeOff) instead of adding a new one.
6  NOTES   : Only the numeric AlarmId is stored; the text is looked up in
7             GVL.ALARM_TEXT when the HMI lines are formatted. This keeps
8             the RETAIN footprint small and lets the wording change
9             without touching logged history.
10 FSD REF : ch. 13.3 Alarm history
11 ===== *)
12
13 TYPE ST_AlarmEntry :
14 STRUCT
15     AlarmId : INT;      (* 1..GVL.NUM_ALARMS, index into GVL.ALARM_TEXT;
16                          0 = empty slot *)
17     TimeOn  : DT;      (* local time the alarm came *)
18     TimeOff : DT;      (* local time it went - only valid if NOT Active *)
19     Active  : BOOL;    (* TRUE until the falling edge is logged *)
20 END_STRUCT
21 END_TYPE

```

KAPITOLA 3

Globální proměnné

Global variables · GVL

Dva listy proměnných: `GVL` pro fyzická I/O, čítače a zobrazovací proměnné, `GVL_HMI` pro povely obsluhy. Díky oddělení HMI nikdy nezapisuje do proměnných namapovaných na I/O.

3.1 GVL

GVL · 184 ŘÁDKŮ KÓDU · FSD KAP. 2.3, 2.5 · PŘÍL. A

Fyzická I/O ze scény Factory I/O, konstanty, čítače výroby a historie alarmů (RETAIN) a zobrazovací proměnné `Dis_*`. Proměnné čte a zapisuje jen `PLC_PRG_ST`.

DEKLARACE · DECLARATION

347 ř.

```

1  (* =====
2  GVL : Global Variable List
3  PURPOSE : All physical I/O (exchanged with Factory I/O over OPC), the
4            retained production counters and alarm history, and the
5            display tags Dis_* and CounterDis_* read by the HMI.
6  NOTES   : Only PLC_PRG_ST reads and writes these variables; function
7            blocks use just the constants below (array bounds, alarm
8            texts). The HMI never writes here - its commands are in
9            GVL_HMI.
10           NC inputs (Stop, E-stops, doors) read TRUE when healthy, so
11           a wire break acts like a pressed button. Retro_S1..S4 are
12           light-operate (TRUE = beam clear) and are inverted once in
13           PLC_PRG_ST step 10.
14  FSD REF : ch. 2.3   Control architecture
15           ch. 2.5   Signal conventions
16           app. A    I/O list
17  ===== *)
18
19  VAR_GLOBAL CONSTANT
20      NUM_ESTOPS : INT := 6;   (* number of E-stop channels *)
21      NUM_DOORS  : INT := 5;   (* number of safety door sensors *)
22      NUM_TOWERS : INT := 8;   (* number of stack light towers *)
23      NUM_SIRENS : INT := 2;   (* number of sound alarms *)
24
25      (* Alarm history (FB_AlarmHistory, PLC_PRG_ST step 95). NUM_ALARMS must match
26      AlarmBits[1..19] in step 95 and ALARM_TEXT below, entry for entry.
27      ALARM_HIST_SIZE = occurrences kept (oldest overwritten first) = number of
28      Dis_AlarmLine rows on the HMI. *)
29      NUM_ALARMS : INT := 19;
30      ALARM_HIST_SIZE : INT := 20;
31
32      (* Alarm texts, index = AlarmId, max. 24 characters. The wording can be changed
33      here freely; logged history shows the new text on the next refresh of the
34      HMI lines. *)
35      ALARM_TEXT : ARRAY[1..NUM_ALARMS] OF STRING(24) := [
36          'E-STOP main',           (* 1 Emergency_Tripped[1] *)
37          'E-STOP lane S',         (* 2 Emergency_Tripped[2] *)
38          'E-STOP lane M',         (* 3 Emergency_Tripped[3] *)
39          'E-STOP lane L',         (* 4 Emergency_Tripped[4] *)
40          'E-STOP pallet load L',   (* 5 Emergency_Tripped[5] *)
41          'E-STOP pallet load R',   (* 6 Emergency_Tripped[6] *)
42          'Door sorting 1 open',    (* 7 Door_Open[1] *)
43          'Door sorting 2 open',    (* 8 Door_Open[2] *)
44          'Door sorting 3 open',    (* 9 Door_Open[3] *)
45          'Door sorting 4 open',    (* 10 Door_Open[4] *)
46          'Door loading 1 open',    (* 11 Door_Open[5] *)
47          'JAM sort cycle',         (* 12 Fault_SortCycle *)
48          'JAM PnP1 -> scale',      (* 13 Fault_PnP1_Path *)
49          'FAULT PnP1 robot',       (* 14 Fault_PnP1_Robot *)
50          'JAM exit lane S',        (* 15 Fault_ExitPath_S *)
51          'JAM exit lane M',        (* 16 Fault_ExitPath_M *)
52          'JAM exit lane L',        (* 17 Fault_ExitPath_L *)
53          'JAM pallet feed',        (* 18 Fault_PalletPath *)
54          'DRAIN timeout'          (* 19 Fault_Draining *)
55      ];
56  END_VAR

```

```

57
58 VAR_GLOBAL
59 (* ===== DIGITAL INPUTS ===== *)
60 Start_Button : BOOL;
61 Stop_Button : BOOL; (* NC: TRUE = at rest, FALSE = pressed or wire break.
62                      Inverted to StopActive and latched in
63                      FB_SafetyInterlock. *)
64 Reset_Button : BOOL; (* NO. Reset edge (R_Reset): leaves MODE_EMERGENCY /
65                      MODE_DRAINED and clears SORT_FAULT, PP_FAULT, all
66                      path watchdogs and Fault_Draining. Does not reset the
67                      production counters. *)
68
69 ResButtCounting1 : BOOL; (* NO. Resets all six production counters at once;
70                          independent of Reset_Button. *)
71
72 Drain_Button : BOOL; (* NO. System Drain request, accepted only in MODE_AUTO.
73                      Edge-detected (R_Drain) together with
74                      GVL_HMI.Drain_Cmd_HMI. *)
75
76 (* E-stop channels packed in PLC_PRG_ST step 20 - NC: FALSE = pressed, wire
77    break or a trip simulated from the HMI. *)
78 Emergency_Buttons : ARRAY[1..NUM_ESTOPS] OF BOOL;
79 Emergency_Tripped : ARRAY[1..NUM_ESTOPS] OF BOOL; (* active-high, per E-stop *)
80 Door_Open : ARRAY[1..NUM_DOORS] OF BOOL; (* active-high, per door *)
81 Safety_Doors : ARRAY[1..NUM_DOORS] OF BOOL; (* packed in step 20, NC: TRUE = closed *)
82
83 (* Individual E-stop and door inputs (NC), packed into Emergency_Buttons /
84    Safety_Doors in PLC_PRG_ST step 20. *)
85 Emergency_Button : BOOL;
86 Emergency_Button_S : BOOL;
87 Emergency_Button_M : BOOL;
88 Emergency_Button_L : BOOL;
89 EButt_PallLoad_L : BOOL;
90 EButt_PallLoad_R : BOOL;
91
92 SafetyDoorSorting1 : BOOL;
93 SafetyDoorSorting2 : BOOL;
94 SafetyDoorSorting3 : BOOL;
95 SafetyDoorSorting4 : BOOL;
96 SafetyDoorLoading1 : BOOL;
97
98 Sel_Mode_Manual : BOOL; (* mode selector - Manual position *)
99 Sel_Mode_Auto : BOOL; (* mode selector - Auto position; in MODE_MANUAL it
100                       acts as Stop -> HARD STOP (PLC_PRG_ST step 40) *)
101
102 Retro_S1 : BOOL; (* RAW light-operate: TRUE = beam clear. Box at the scale
103                  entry. Do not read directly - use Retro_S1_Present
104                  (PLC_PRG_ST step 10). *)
105 Diffuse_S1 : BOOL; (* weight capture position (stop point) *)
106 Diffuse_S2 : BOOL; (* scale exit confirmation *)
107
108 Diffuse_S3 : BOOL; (* sort confirm - left path (S), after the pop-up wheel *)
109 Diffuse_S4 : BOOL; (* sort confirm - forward path (M), after the pop-up wheel *)
110 Diffuse_S5 : BOOL; (* sort confirm - right path (L), after the pop-up wheel *)
111
112 Retro_S2 : BOOL; (* RAW light-operate, see Retro_S1. Exit of lane S: end of
113                  PathWD_S, Counter_S. *)
114 Retro_S3 : BOOL; (* RAW light-operate. Exit of lane M: end of PathWD_M,
115                  Counter_M. *)
116 Retro_S4 : BOOL; (* RAW light-operate. Exit of lane L: end of PathWD_L,
117                  Counter_L. *)
118
119 Weight_Analog : REAL; (* continuous weight measurement, kg *)
120
121 (* Manual mode - physical / simulated operator inputs Man_*. Each one is
122    combined with its GVL_HMI *_Cmd_HMI counterpart in PLC_PRG_ST step 70: BOOL
123    channels by OR, REAL channels by LIMIT(Man_* + Cmd_HMI). RollStop_Entry has
124    no Man_input - in MANUAL it is driven from the HMI only. *)
125 Man_Conv1 : BOOL;
126 Man_Conv2 : REAL;
127 Man_Conv3 : REAL;
128 Man_Conv4 : REAL;
129 Man_Scale : BOOL;
130 Man_Sorter_Left : BOOL;
131 Man_Sorter_Forward : BOOL;

```

```

132   Man_Sorter_Right   : BOOL;
133
134   (* PnP1 (position 0..10, grip), load conveyors, box line roller stops *)
135   Man_PnP1_Xset      : REAL;
136   Man_PnP1_Zset      : REAL;
137   Man_PnP1_Grab      : BOOL;
138
139   Man_LoadConv1 : REAL;
140   Man_LoadConv2 : REAL;
141   Man_LoadConv3 : REAL;
142
143   Man_RollStop_S : BOOL;
144   Man_RollStop_M : BOOL;
145   Man_RollStop_L : BOOL;
146
147   (* Pallet line drives, -10..+10 *)
148   Man_PallConvLoad_R : REAL;
149   Man_PallConvCurv_R1 : REAL;
150   Man_PallConvCurv_R2 : REAL;
151   Man_PallConvCurv_R3 : REAL;
152   Man_PallConvStrait_1 : REAL;
153   Man_PallConvStrait_2 : REAL;
154   Man_PallConvStrait_3 : REAL;
155   Man_PallConvCurv_L1 : REAL;
156   Man_PallConvCurv_L2 : REAL;
157   Man_PallConvCurv_L3 : REAL;
158   Man_PallConvLoad_L : REAL;
159
160   (* Pallet line roller stops *)
161   Man_RollStop_PallPick1 : BOOL;
162   Man_RollStop_PallPick2 : BOOL;
163   Man_RollStop_LoadR : BOOL;
164   Man_RollStop_LoadL : BOOL;
165   Man_RollStop_Entry1 : BOOL;
166
167   (* ===== DIGITAL OUTPUTS ===== *)
168   Conv1 : BOOL;
169   Scale_Conveyor : BOOL;
170   Sorter_Left : BOOL; (* direction input - S path; no memory, must
171                        be held (see FB_SortingCycle) *)
172   Sorter_Forward : BOOL; (* sorter drive - runs continuously in RUN and
173                          STOPPING *)
174   Sorter_Right : BOOL; (* direction input - L path; no memory, must
175                        be held (see FB_SortingCycle) *)
176
177   (* Box line roller stops - TRUE = engaged *)
178   RollStop_Entry : BOOL;
179   RollStop_S : BOOL;
180   RollStop_M : BOOL;
181   RollStop_L : BOOL;
182
183   (* Signaling - written in PLC_PRG_ST step 80, one element per tower / siren *)
184   Tower_Red_Out : ARRAY[1..NUM_TOWERS] OF BOOL;
185   Tower_Yellow_Out : ARRAY[1..NUM_TOWERS] OF BOOL;
186   Tower_Green_Out : ARRAY[1..NUM_TOWERS] OF BOOL;
187   Siren_Out : ARRAY[1..NUM_SIRENS] OF BOOL;
188
189   (* ===== ANALOG OUTPUTS ===== *)
190   (* Exit lane drives S / M / L: -10..+20 (Factory I/O range), 5.0 while running,
191      0.0 = stopped. *)
192   Conv2 : REAL;
193   Conv3 : REAL;
194   Conv4 : REAL;
195
196   (* Load conveyors, -10..+10 *)
197   LoadConv1 : REAL;
198   LoadConv2 : REAL;
199   LoadConv3 : REAL;
200
201   (* Display copies of the production counters for the HMI or a display device -
202      the RETAIN counters themselves are never bound to the HMI. Recomputed every
203      scan in PLC_PRG_ST step 30. *)
204   CounterDis_Exit_S : DINT;
205   CounterDis_Exit_M : DINT;
206   CounterDis_Exit_L : DINT;

```

```

207 CounterDis_Exit_Total : DINT;    (* S + M + L = all sorted boxes *)
208
209 CounterDis_BoxIn      : DINT;    (* boxes placed on LoadConv1 (DiffS_PosDrop1) *)
210 CounterDis_PalletIn   : DINT;    (* pallets entering the feed (LArray_LoadR) *)
211 CounterDis_PalletOut  : DINT;    (* pallets leaving the feed (LArray_LoadL) *)
212
213 (* HMI status texts, written in PLC_PRG_ST step 90. Nothing else reads them, so
214    the wording can be changed there freely. *)
215 Dis_ModeText         : STRING(20);
216 Dis_SortStatusText   : STRING(20);
217
218 Dis_PnP1StatusText   : STRING(20);
219
220 (* Dis_SystemText follows the same conditions as the stack lights.
221    Dis_BoxClassText = class of the box being sorted (CLASSIFICATION ..
222    SORT_CONFIRM), '-' otherwise. Dis_CurrentWeight = last settled weight, held
223    until the next box is weighed. *)
224 Dis_SystemText       : STRING(20);
225 Dis_BoxClassText     : STRING(6);
226 Dis_CurrentWeight    : REAL;
227
228 (* Alarm history for the HMI, written in PLC_PRG_ST step 95 from
229    FB_AlarmHistory. Index 1 = newest occurrence. Fixed columns - use a
230    monospace font on the HMI:
231    '25.09. 14:03:27 14:05:10 JAM exit lane S' (cleared)
232    '25.09. 14:07:02 ACTIVE E-STOP main' (still active)
233    Unused rows are ''. Not retained: rebuilt from the RETAIN
234    GVL.AlarmHist on the first scan after a restart. *)
235 Dis_AlarmLine         : ARRAY[1..ALARM_HIST_SIZE] OF STRING(60);
236 Dis_AlarmLineActive   : ARRAY[1..ALARM_HIST_SIZE] OF BOOL;    (* per row -
237    bind to the row's Colors -> Toggle color
238    variable to highlight still-active alarms *)
239 Dis_AlarmActiveCount  : INT;    (* alarms active right now, 0..NUM_ALARMS -
240    e.g. a badge/counter on the main screen *)
241 Dis_AlarmHistCount    : INT;    (* rows in use, 0..ALARM_HIST_SIZE *)
242
243 (* ===== PICK & PLACE (PnP1) ===== *)
244 DiffS_PosPick1 : BOOL;    (* pallet at the pick point (Strait_2). Shared by
245    FB_PickPlace (PosPick) and FB_PalletFeed
246    (DiffS_PosPick). *)
247 DiffS_PosDrop1 : BOOL;    (* LoadConv1 drop point occupied *)
248
249 (* PnP1 axes: position setpoint / actual 0..10, grip command and sensor *)
250 PnP1_Xset      : REAL;
251 PnP1_Zset      : REAL;
252 PnP1_GrabCmd   : BOOL;
253 PnP1_Xact      : REAL;
254 PnP1_Zact      : REAL;
255 PnP1_Detected  : BOOL;
256
257 (* ===== PALLET FEED (FB_PalletFeed) =====
258    Carries a loaded pallet from the Emitter (Factory I/O scene object,
259    not PLC-controlled) to the PnP1 pick point and the empty pallet on to
260    its exit:
261    PallConvLoad_R -> Curv_R1 -> Curv_R2 -> Curv_R3 -> Strait_1 ->
262    Strait_2 (pick point) -> Strait_3 -> Curv_L1 -> Curv_L2 ->
263    Curv_L3 -> PallConvLoad_L
264    All eleven drives are REAL, -10..+10; the sign follows the drive
265    orientation in the scene. *)
266 DiffS_PallEntry : BOOL;    (* pallet waiting to transfer from
267    Strait_1 onto Strait_2 *)
268 LArray_LoadR    : BOOL;    (* pallet entering the feed: start of PathWD_Pallet,
269    Counter_PalletIn; closes the entrance during Drain *)
270 LArray_LoadL    : BOOL;    (* pallet leaving the feed: end of PathWD_Pallet,
271    Counter_PalletOut *)
272
273 PallConvLoad_R   : REAL;
274 PallConvCurv_R1 : REAL;
275 PallConvCurv_R2 : REAL;
276 PallConvCurv_R3 : REAL;
277 PallConvStrait_1 : REAL;
278 PallConvStrait_2 : REAL;
279 PallConvStrait_3 : REAL;
280 PallConvCurv_L1 : REAL;
281 PallConvCurv_L2 : REAL;

```

```

282   PallConvCurv_L3 : REAL;
283   PallConvLoad_L  : REAL;
284
285   RollStop_PallPick1 : BOOL; (* pick point clamp, side 1 - TRUE = engaged *)
286   RollStop_PallPick2 : BOOL; (* pick point clamp, side 2 *)
287   RollStop_LoadR  : BOOL; (* entrance stop at PallConvLoad_R *)
288   RollStop_LoadL  : BOOL; (* exit stop at PallConvLoad_L *)
289   RollStop_Entry1 : BOOL; (* stop at the Strait_1 entry (not the box line
290                           RollStop_Entry) *)
291
292   (* ===== JAM / PATH WATCHDOGS =====
293   One fault bit per source, so the HMI and the alarm history show which
294   segment jammed. LineJamFault = OR of all eight and forces
295   MODE_EMERGENCY (PLC_PRG_ST step 40). InFlight_* = items currently
296   tracked on each path: the exit lanes also gate FB_SortingCycle's
297   drain check, the PnP1 and pallet paths the System Drain completion. *)
298   Fault_SortCycle : BOOL; (* FB_SortingCycle timeout in WAIT_SCALE_ENTRY /
299                           WAIT_EXIT / SORT_CONFIRM *)
300   Fault_PnP1_Path : BOOL; (* jam between the Pick&Place drop point
301                           (DiffS_PosDrop1) and the scale entry
302                           (Retro_S1) *)
303   Fault_PnP1_Robot : BOOL; (* PP_FAULT inside FB_PickPlace: axis or grip
304                           timeout, item lost *)
305   Fault_ExitPath_S : BOOL; (* jam on Conv2, Diffuse_S3 -> Retro_S2 *)
306   Fault_ExitPath_M : BOOL; (* jam on Conv3, Diffuse_S4 -> Retro_S3 *)
307   Fault_ExitPath_L : BOOL; (* jam on Conv4, Diffuse_S5 -> Retro_S4 *)
308   Fault_PalletPath : BOOL; (* jam on the pallet feed, LArray_LoadR ->
309                           LArray_LoadL *)
310   Fault_Draining : BOOL; (* MODE_DRAINED not reached within T_DRAIN_TIMEOUT.
311                           Latched, cleared only by Reset (PLC_PRG_ST step
312                           40). *)
313   LineJamFault : BOOL; (* OR of the eight faults above *)
314
315   InFlight_PnP1_Path : INT;
316   InFlight_ExitPath_S : INT;
317   InFlight_ExitPath_M : INT;
318   InFlight_ExitPath_L : INT;
319   InFlight_PalletPath : INT;
320 END_VAR
321
322 (* ===== PRODUCTION COUNTERS AND ALARM HISTORY (RETAIN) ===== *)
323 (* Written only by PLC_PRG_ST step 30, from six FB_ProductionCounter
324 instances. Counted on confirmed physical events (rising edges): the
325 exit lanes on Retro_S2/S3/S4 - the box's real destination, not the
326 class predicted on the scale - boxes in on DiffS_PosDrop1, pallets in
327 and out on LArray_LoadR / LArray_LoadL.
328 Reset only by ResButtCounting1 / ResetCounting_Cmd_HMI, all six at
329 once; the machine Reset does not clear them.
330 Declared RETAIN (no PERSISTENT list needed - survival of a program
331 download is not required). *)
332 VAR_GLOBAL RETAIN
333   Counter_S : DINT;
334   Counter_M : DINT;
335   Counter_L : DINT;
336   Counter_BoxIn : DINT;
337   Counter_PalletIn : DINT;
338   Counter_PalletOut : DINT;
339
340   (* Alarm history ring buffer, see FB_AlarmHistory. Kept here as RETAIN and
341   passed to the FB by VAR_IN_OUT, so the non-retained FB instance can never
342   overwrite it after a restart. AlarmHistHead = newest row (0 = empty),
343   AlarmHistCount = rows in use. *)
344   AlarmHist : ARRAY[1..ALARM_HIST_SIZE] OF ST_AlarmEntry;
345   AlarmHistHead : INT;
346   AlarmHistCount : INT;
347 END_VAR

```

3.2 GVL_HMI

GVL · 50 ŘÁDKŮ KÓDU · FSD KAP. 12, 13 · PŘÍL. A.6

Všechno, co zapisuje HMI. Proměnné nejsou namapované na I/O a se vstupy se slučují na jediném místě.

DEKLARACE · DECLARATION

92 ř.

```

1  (* =====
2  GVL_HMI : HMI command variables
3  PURPOSE : Everything the HMI writes. These variables are never mapped
4             to I/O; each one is combined with its physical input at a
5             single point in PLC_PRG_ST.
6  NOTES   : Why a separate list: the visualization task and the main
7             task run independently. An HMI button writing directly into
8             an I/O-mapped variable could be overwritten by the next I/O
9             update before PLC_PRG_ST reads it - the press would be lost.
10           Combination: NO buttons by OR, NC channels (Stop, E-stops)
11           by AND NOT, manual REAL channels by LIMIT(Man_* + Cmd_HMI) -
12           an HMI slider adds to the physical input instead of
13           fighting it (PLC_PRG_ST step 70).
14           SelModeManual_Cmd_HMI / SelModeAuto_Cmd_HMI form a
15           2-position selector, exactly one TRUE (step 15). Both HMI
16           buttons must write TRUE to their own bit (Execute ST-Code /
17           OnMouseClicked), not Toggle Variable. Manual is preselected
18           after a cold download.
19           AlarmHistClear_Cmd_HMI has no physical button; configure it
20           as Tap (momentary), not Toggle.
21           In AUTO / DRAINING the analog trims are overwritten every
22           scan with the live outputs; a HARD STOP clears all manual
23           trims (PLC_PRG_ST step 40).
24  FSD REF : ch. 12   Manual mode
25           ch. 13   HMI interface
26           app. A.6 HMI commands
27  ===== *)
28  VAR_GLOBAL
29      Start_Cmd_HMI       : BOOL; (* OR'd into StartDelay's StartButton input *)
30      Stop_Cmd_HMI        : BOOL; (* AND NOT'd into Safety's RawStopButton input *)
31      Reset_Cmd_HMI       : BOOL; (* OR'd into R_Reset.CLK *)
32      ResetCounting_Cmd_HMI : BOOL; (* OR'd into R_ResetCounters.CLK *)
33      Drain_Cmd_HMI       : BOOL; (* OR'd into R_Drain.CLK *)
34      AlarmHistClear_Cmd_HMI : BOOL; (* sole input of R_AlarmClear.CLK *)
35      (* 2-position selector - exactly one bit TRUE, see header *)
36      SelModeManual_Cmd_HMI : BOOL := TRUE; (* default-selected on a cold download *)
37      SelModeAuto_Cmd_HMI  : BOOL;
38
39      (* MANUAL trims - combined with Man_* in PLC_PRG_ST step 70 *)
40      PnP1_Xset_Cmd_HMI    : REAL; (* position trim, 0-10 - see step 70 *)
41      PnP1_Zset_Cmd_HMI    : REAL; (* position trim, 0-10 - see step 70 *)
42      PnP1_GrabCmd_Cmd_HMI : BOOL;
43
44      LoadConv1_Cmd_HMI    : REAL; (* -10..+10 *)
45      LoadConv2_Cmd_HMI    : REAL; (* -10..+10 *)
46      LoadConv3_Cmd_HMI    : REAL; (* -10..+10 *)
47
48      RollStop_Entry_Cmd_HMI : BOOL; (* sole MANUAL control for RollStop_Entry
49                                     - no Man_RollStop_Entry exists *)
50
51      Conv1_Cmd_HMI        : BOOL;
52      Conv2_Cmd_HMI        : REAL; (* -10..+10 *)
53      Conv3_Cmd_HMI        : REAL; (* -10..+10 *)
54      Conv4_Cmd_HMI        : REAL; (* -10..+10 *)
55
56      Scale_Conveyor_Cmd_HMI : BOOL;
57      Sorter_Left_Cmd_HMI    : BOOL;
58      Sorter_Forward_Cmd_HMI : BOOL;
59      Sorter_Right_Cmd_HMI   : BOOL;
60
61      RollStop_S_Cmd_HMI    : BOOL;
62      RollStop_M_Cmd_HMI    : BOOL;
63      RollStop_L_Cmd_HMI    : BOOL;
64
65      (* Pallet line drives *)
66      PallConvLoad_R_Cmd_HMI : REAL; (* -10..+10 *)
67      PallConvCurv_R1_Cmd_HMI : REAL; (* -10..+10 *)

```

```
68 PallConvCurv_R2_Cmd_HMI : REAL; (* -10..+10 *)
69 PallConvCurv_R3_Cmd_HMI : REAL; (* -10..+10 *)
70 PallConvStrait_1_Cmd_HMI : REAL; (* -10..+10 *)
71 PallConvStrait_2_Cmd_HMI : REAL; (* -10..+10 *)
72 PallConvStrait_3_Cmd_HMI : REAL; (* -10..+10 *)
73 PallConvCurv_L1_Cmd_HMI : REAL; (* -10..+10 *)
74 PallConvCurv_L2_Cmd_HMI : REAL; (* -10..+10 *)
75 PallConvCurv_L3_Cmd_HMI : REAL; (* -10..+10 *)
76 PallConvLoad_L_Cmd_HMI : REAL; (* -10..+10 *)
77
78 (* Pallet line roller stops *)
79 RollStop_PallPick1_Cmd_HMI : BOOL;
80 RollStop_PallPick2_Cmd_HMI : BOOL;
81 RollStop_LoadR_Cmd_HMI : BOOL;
82 RollStop_LoadL_Cmd_HMI : BOOL;
83 RollStop_Entry1_Cmd_HMI : BOOL;
84
85 (* NC channels - Cmd=TRUE means "HMI wants to simulate a trip" *)
86 EStop_Cmd_HMI : BOOL; (* combined with Emergency_Button *)
87 EStop_S_Cmd_HMI : BOOL; (* combined with Emergency_Button_S *)
88 EStop_M_Cmd_HMI : BOOL; (* combined with Emergency_Button_M *)
89 EStop_L_Cmd_HMI : BOOL; (* combined with Emergency_Button_L *)
90 EStop_PallL_Cmd_HMI : BOOL; (* combined with EButt_PallLoad_L *)
91 EStop_PallR_Cmd_HMI : BOOL; (* combined with EButt_PallLoad_R *)
92 END_VAR
```

KAPITOLA 4

Funkce Functions

Bezstavové funkce – stejné vstupy dávají vždy stejný výsledek. Časovače a detektory hran, které potřebují paměť, leží ve funkčních blocích nebo v `PLC_PRG_ST`.

4.1 F_ClassifyWeight

FUNKCE · 11 ŘÁDKŮ KÓDU · FSD KAP. 6

Prahy 4,5 a 6,0 kg; obě hranice patří do třídy M.

DEKLARACE · DECLARATION

12 ř.

```

1  (* =====
2  FUNCTION : F_ClassifyWeight
3  PURPOSE : Classifies a settled weight reading into S / M / L.
4  NOTES   : < 4.5 kg = S, 4.5 .. 6.0 kg = M (both limits belong to M),
5             > 6.0 kg = L. Nominal boxes: S 4 kg, M 5 kg, L 7.5 kg.
6  FSD REF : ch. 6      Weight classification
7  ===== *)
8
9  FUNCTION F_ClassifyWeight : E_WeightClass
10 VAR_INPUT
11     Weight : REAL;    (* CurrentWeight - Weight_Analog after settling *)
12 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

7 ř.

```

1  IF Weight < 4.5 THEN
2     F_ClassifyWeight := E_WeightClass.CLS_S;
3  ELSIF Weight <= 6.0 THEN
4     F_ClassifyWeight := E_WeightClass.CLS_M;
5  ELSE
6     F_ClassifyWeight := E_WeightClass.CLS_L;
7  END_IF

```

4.2 F_NextMode

FUNKCE · 48 ŘÁDKŮ KÓDU · FSD KAP. 3.1, 9

Režim pro další scan. Nouze a HARD STOP se vynucují už v kroku 40, proto v tabulce nejsou.

DEKLARACE · DECLARATION

56 ř.

```

1  (* =====
2  FUNCTION : F_NextMode
3  PURPOSE : Mode for the next scan - the transition table of the line.
4  NOTES   : Called in PLC_PRG_ST step 60 with ModeThisScan, i.e. after
5             the pre-emption of step 40. Those transitions are therefore
6             not in this table:
7             any mode    -> EMERGENCY : E-stop, door or jam fault
8             AUTO/DRAINING -> STOPPED   : selector on Manual (HARD STOP)
9             MANUAL       -> STOPPED   : Stop, HMI Stop or selector
10                                on Auto (HARD STOP)
11
12             Transitions computed here:
13             EMERGENCY -> STOPPED : Reset AND no emergency AND no jam
14             STOPPED   -> AUTO    : Start confirmed (3 s) AND no jam
15             STOPPED   -> MANUAL  : Manual selected (masked by the
16                                HARD STOP latch in PLC_PRG_ST)
17             AUTO      -> STOPPED : Stop latched AND line drained
18             AUTO      -> DRAINING : Drain request - a simultaneous
19                                Stop wins (checked first)
20             DRAINING  -> STOPPED : Stop latched AND line drained -
21                                Stop interrupts a drain
22             DRAINING  -> DRAINED : pallet side AND box side settled
23             DRAINED   -> STOPPED : Reset AND no emergency AND no jam
24             MANUAL    -> STOPPED : StopActive (defensive - normally
25                                already forced in step 40)
26
27             Recovery from a jam takes two Resets: in the scan of the

```

```

26         first Reset JamFault still holds the previous value while
27         the blocks clear their faults (text JAM -> E-STOP); the
28         second Reset leaves EMERGENCY (FSD ch. 11.3).
29         In STOPPED -> AUTO the jam condition is only defensive.
30         Stateless: timers and edges (start delay, settle timers,
31         Reset / Drain edges) live in PLC_PRG_ST.
32     FSD REF : ch. 3.1 Transitions
33              ch. 3.2 Command priority
34              ch. 9 System Drain
35     ===== *)
36
37     FUNCTION F_NextMode : E_Mode
38     VAR_INPUT
39         CurrentMode      : E_Mode;    (* ModeThisScan - after step 40 *)
40         StartConfirmed    : BOOL;      (* StartDelay.StartConfirmed - 3 s after Start *)
41         SelModeManual     : BOOL;      (* SelModeManualEff AND NOT HardStopLatch
42                                         (PLC_PRG_ST) *)
43         ResetTrig         : BOOL;      (* R_Reset.Q *)
44         StopRequested     : BOOL;      (* FB_SafetyInterlock.StopRequested - latched *)
45         StopActive        : BOOL;      (* FB_SafetyInterlock.StopActive - raw
46                                         active-high level *)
47         EmergencyActive   : BOOL;      (* FB_SafetyInterlock.EmergencyActive *)
48         JamFault          : BOOL;      (* GVL.LineJamFault - latched watchdog faults *)
49         LineDrained       : BOOL;      (* Cycle.Drained AND PickPlace.Drained,
50                                         see PLC_PRG_ST *)
51         DrainTrig         : BOOL;      (* R_Drain.Q - Drain_Button OR Drain_Cmd_HMI *)
52         PalletSideSettled : BOOL;      (* LastPalletSettleTmr.Q - no pallet in the feed
53                                         for 1 s *)
54         BoxSideSettled    : BOOL;      (* BoxSideSettleTmr.Q - line drained and PnP1 path
55                                         empty for 1 s *)
56     END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

48 ř.

```

1  F_NextMode := CurrentMode;    (* default: stay put unless a condition below fires *)
2
3  CASE CurrentMode OF
4
5      E_Mode.MODE_EMERGENCY:
6          IF (NOT EmergencyActive) AND (NOT JamFault) AND ResetTrig THEN
7              F_NextMode := E_Mode.MODE_STOPPED;
8          END_IF
9
10     E_Mode.MODE_STOPPED:
11         (* NOT JamFault is defensive: while LineJamFault is latched, step 40 forces
12            MODE_EMERGENCY every scan, so Mode never rests in STOPPED with it set. *)
13         IF StartConfirmed AND (NOT JamFault) THEN
14             F_NextMode := E_Mode.MODE_AUTO;
15         ELSIF SelModeManual THEN
16             F_NextMode := E_Mode.MODE_MANUAL;
17         END_IF
18
19     E_Mode.MODE_AUTO:
20         IF LineDrained AND StopRequested THEN
21             F_NextMode := E_Mode.MODE_STOPPED;
22         ELSIF DrainTrig THEN    (* after Stop - a simultaneous Stop wins *)
23             F_NextMode := E_Mode.MODE_DRAINING;
24         END_IF
25
26         (* Drain in progress. A Stop interrupts it and ends in MODE_STOPPED once the
27            line is clear (raw LineDrained, as in AUTO). Normal completion uses the
28            debounced settle inputs - one scan with a stale InFlight count must not
29            freeze the line in the sticky MODE_DRAINED. *)
30     E_Mode.MODE_DRAINING:
31         IF LineDrained AND StopRequested THEN
32             F_NextMode := E_Mode.MODE_STOPPED;
33         ELSIF PalletSideSettled AND BoxSideSettled THEN
34             F_NextMode := E_Mode.MODE_DRAINED;
35         END_IF
36
37         (* Drain complete, holding - left only by Reset. *)
38     E_Mode.MODE_DRAINED:
39         IF (NOT EmergencyActive) AND (NOT JamFault) AND ResetTrig THEN

```

```

40         F_NextMode := E_Mode.MODE_STOPPED;
41     END_IF
42
43     E_Mode.MODE_MANUAL:
44         IF StopActive THEN
45             F_NextMode := E_Mode.MODE_STOPPED;
46         END_IF
47
48 END_CASE

```

4.3 F_TowerColors

FUNKCE · 43 ŘÁDKŮ KÓDU · FSD KAP. 4.1

Barvy věže podle režimu; stejné podmínky používá stavový text SYSTEM.

DEKLARACE · DECLARATION

26 ř.

```

1  (* =====
2  FUNCTION : F_TowerColors
3  PURPOSE : Stack light colors (red / yellow / green) for all towers.
4  NOTES   : STOPPED (incl. HARD STOP) and DRAINED: steady yellow - the
5             text Dis_SystemText tells them apart. Start countdown: plus
6             blinking green. AUTO: steady green, blinking yellow once Stop
7             is latched (STOPPING). MANUAL: yellow and green blink.
8             DRAINING: all three blink. EMERGENCY: red blinks; a jam adds
9             blinking yellow, so it is distinct from an E-stop or door.
10 FSD REF : ch. 4.1 Stack lights and system text
11 ===== *)
12
13 FUNCTION F_TowerColors
14 VAR_INPUT
15     Mode          : E_Mode;
16     StopRequested : BOOL;
17     StartPending  : BOOL; (* StartDelay.Armed - TRUE while the post-Start
18                           countdown is running *)
19     JamFault      : BOOL; (* GVL.LineJamFault *)
20     BlinkPulse    : BOOL;
21 END_VAR
22 VAR_OUTPUT
23     Red    : BOOL;
24     Yellow : BOOL;
25     Green  : BOOL;
26 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

36 ř.

```

1  Red    := FALSE;
2  Yellow := FALSE;
3  Green  := FALSE;
4
5  CASE Mode OF
6      E_Mode.MODE_EMERGENCY:
7          Red := BlinkPulse;
8          IF JamFault THEN
9              Yellow := BlinkPulse; (* Red+Yellow together = jam, not a safety trip *)
10         END_IF
11
12     E_Mode.MODE_STOPPED:
13         Yellow := TRUE; (* steady yellow = "ready to start" *)
14         IF StartPending THEN
15             Green := BlinkPulse; (* blinking green warns the line is about to start *)
16         END_IF
17
18     E_Mode.MODE_MANUAL:
19         Yellow := BlinkPulse;
20         Green  := BlinkPulse; (* any part of the line can start/stop at any time in MANUAL *)
21
22     E_Mode.MODE_AUTO:
23         IF StopRequested THEN
24             Yellow := BlinkPulse;
25         ELSE

```

```

26         Green := TRUE;
27     END_IF
28
29     E_Mode.MODE_DRAINING:
30         Red := BlinkPulse;
31         Yellow := BlinkPulse;
32         Green := BlinkPulse; (* all three together *)
33
34     E_Mode.MODE_DRAINED:
35         Yellow := TRUE; (* steady yellow, same as MODE_STOPPED *)
36 END_CASE

```

4.4 F_Pad2

FUNKCE · 9 ŘÁDKŮ KÓDU · FSD KAP. 13.3

Pomocná funkce pro formát času: 0..99 na „00“...„99“.

DEKLARACE · DECLARATION

9 ř.

```

1  (* =====
2  FUNCTION : F_Pad2
3  PURPOSE : 0..99 -> '00'..'99'. Helper for F_FormatAlarmTime.
4  ===== *)
5
6  FUNCTION F_Pad2 : STRING(2)
7  VAR_INPUT
8      Value : DINT; (* expected 0..99 *)
9  END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

5 ř.

```

1  IF Value < 10 THEN
2      F_Pad2 := CONCAT('0', DINT_TO_STRING(Value));
3  ELSE
4      F_Pad2 := DINT_TO_STRING(Value);
5  END_IF

```

4.5 F_FormatAlarmTime

FUNKCE · 40 ŘÁDKŮ KÓDU · FSD KAP. 13.3

Čas alarmu ve tvaru DD.MM. HH:MM:SS, počítaný aritmeticky z hodnoty DT.

DEKLARACE · DECLARATION

27 ř.

```

1  (* =====
2  FUNCTION : F_FormatAlarmTime
3  PURPOSE : Formats an alarm timestamp for the HMI history lines.
4              WithDate = TRUE -> 'DD.MM. HH:MM:SS' (15 characters)
5              WithDate = FALSE -> 'HH:MM:SS' (8 characters)
6  NOTES : Computed arithmetically from the DT value (seconds since
7              1970) instead of slicing DT_TO_STRING(), so the result does
8              not depend on that function's text format. Day and month
9              use the standard days-to-civil-date algorithm. No year - the
10             HMI line has no room for it and the buffer holds only recent
11             history.
12  FSD REF : ch. 13.3 Alarm history
13  ===== *)
14
15  FUNCTION F_FormatAlarmTime : STRING(15)
16  VAR_INPUT
17      Stamp : DT;
18      WithDate : BOOL;
19  END_VAR
20  VAR
21      Secs : UDINT;
22      Days : DINT;
23      TimeOfDay : DINT;
24      Z, era, doe, yoe, doy, mp : DINT;

```

```

25     DayOfMon, MonthNo      : DINT;
26     TimeTxt      : STRING(8);
27 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

30 ř.

```

1  Secs      := DT_TO_UDINT(Stamp);
2  Days      := UDINT_TO_DINT(Secs / 86400);
3  TimeOfDay := UDINT_TO_DINT(Secs MOD 86400);
4
5  TimeTxt := CONCAT(CONCAT(CONCAT(CONCAT(
6      F_Pad2(TimeOfDay / 3600), ':'),
7      F_Pad2((TimeOfDay MOD 3600) / 60), ':'),
8      F_Pad2(TimeOfDay MOD 60));
9
10 IF WithDate THEN
11     z := Days + 719468;
12     era := z / 146097;
13     doe := z - era * 146097;
14     yoe := (doe - doe / 1460 + doe / 36524 - doe / 146096) / 365;
15     doy := doe - (365 * yoe + yoe / 4 - yoe / 100);
16     mp := (5 * doy + 2) / 153;
17     DayOfMon := doy - (153 * mp + 2) / 5 + 1;
18     IF mp < 10 THEN
19         MonthNo := mp + 3;
20     ELSE
21         MonthNo := mp - 9;
22     END_IF
23
24     F_FormatAlarmTime := CONCAT(CONCAT(CONCAT(CONCAT(
25         F_Pad2(DayOfMon), '.'),
26         F_Pad2(MonthNo)), '.'),
27         TimeTxt);
28 ELSE
29     F_FormatAlarmTime := TimeTxt;
30 END_IF

```

KAPITOLA 5

Funkční bloky Function blocks

Procesní logika linky. Bloky nečtou ani nezapisují proměnné GVL – signály dostávají přes `VAR_INPUT` a vracejí přes `VAR_OUTPUT`; historie alarmů pracuje s retain bufferem přes `VAR_IN_OUT`. Každý blok lze proto testovat a použít samostatně.

5.1 FB_SafetyInterlock

FUNKČNÍ BLOK · 34 ŘÁDKŮ KÓDU · FSD KAP. 10

Převod NC vstupů na aktivní-high a západka Stopu. Instance `Safety`, krok 20.

DEKLARACE · DECLARATION

39 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_SafetyInterlock
3  PURPOSE : Converts the NC E-stop, door and Stop inputs to active-high
4            signals and latches the Stop request.
5  NOTES   : NC wiring: FALSE = pressed / open / wire break, so a broken
6            wire acts like a pressed button. EmergencyActive = any
7            E-stop or door. StopRequested is set on the rising edge of
8            StopActive and held until the line is in MODE_STOPPED - one
9            Stop press carries through the whole controlled run-down.
10           Standard PLC logic in simulation; a real machine needs a
11           certified safety circuit (safety relay or safety PLC).
12  FSD REF : ch. 10   Safety interlocks
13            ch. 3.5  Stop, HARD STOP and emergency
14  ===== *)
15
16 FUNCTION_BLOCK FB_SafetyInterlock
17 VAR_INPUT
18   RawEmergencyButtons : ARRAY[1..GVL.NUM_ESTOPS] OF BOOL;
19   RawSafetyDoors      : ARRAY[1..GVL.NUM_DOORS] OF BOOL;
20   RawStopButton       : BOOL; (* Stop_Button AND NOT Stop_Cmd_HMI; in MANUAL
21                                also AND NOT Auto selected *)
22   SystemAtStopped     : BOOL; (* Mode = MODE_STOPPED at scan entry - resets the
23                                Stop latch *)
24 END_VAR
25 VAR_OUTPUT
26   EmergencyActive : BOOL; (* logical/active-high; NOT RawEmergencyButtons[i] *)
27   EmergencyTripped : ARRAY[1..GVL.NUM_ESTOPS] OF BOOL;
28   DoorOpen        : ARRAY[1..GVL.NUM_DOORS] OF BOOL;
29   StopActive       : BOOL; (* NOT RawStopButton - level; MANUAL exit / HARD
30                             STOP *)
31   StopRequested    : BOOL; (* StopLatch.Q1 - latched *)
32 END_VAR
33 VAR
34   R_StopButton : R_TRIG; (* rising edge of StopActive - feeds StopLatch.SET1 *)
35   StopLatch    : SR; (* SET1 := R_StopButton.Q, RESET := SystemAtStopped *)
36
37   i : INT;
38   AnyTripped : BOOL;
39 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

19 ř.

```

1  AnyTripped := FALSE;
2  FOR i := 1 TO GVL.NUM_ESTOPS DO
3     EmergencyTripped[i] := NOT RawEmergencyButtons[i];
4     IF EmergencyTripped[i] THEN AnyTripped := TRUE; END_IF
5  END_FOR;
6
7  FOR i := 1 TO GVL.NUM_DOORS DO
8     DoorOpen[i] := NOT RawSafetyDoors[i];
9     IF DoorOpen[i] THEN AnyTripped := TRUE; END_IF
10 END_FOR;
11
12 EmergencyActive := AnyTripped;
13

```

```

14 StopActive := NOT RawStopButton;
15
16 R_StopButton(CLK := StopActive);
17
18 StopLatch(SET1 := R_StopButton.Q, RESET := SystemAtStopped);
19 StopRequested := StopLatch.Q1;

```

5.2 FB_ProductionCounter

FUNKČNÍ BLOK · 18 ŘÁDKŮ KÓDU · FSD KAP. 14

Obecný čítač náběžných hran. Šest instancí v kroku 30: CounterS/M/L, CounterBoxIn, CounterPalletIn, CounterPalletOut.

DEKLARACE · DECLARATION

25 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_ProductionCounter
3  PURPOSE : Generic event counter: counts rising edges of ConfirmSignal
4            while enabled and resets to zero on command.
5  NOTES   : Reset works in any mode and has priority over counting.
6            Six instances in PLC_PRG_ST step 30 - exit lanes S / M / L,
7            boxes in, pallets in and out - enabled in MODE_AUTO and
8            MODE_DRAINING. The caller copies Count to the RETAIN
9            counters in GVL.
10 FSD REF : ch. 14   Production counters
11 ===== *)
12
13 FUNCTION_BLOCK FB_ProductionCounter
14 VAR_INPUT
15     ConfirmSignal : BOOL; (* counted event - rising edge *)
16     ResetCommand  : BOOL; (* R_ResetCounters.Q - reset to 0, any mode *)
17     CountEnable   : BOOL; (* Mode = AUTO OR DRAINING, at scan entry (before step
18                           40) *)
19 END_VAR
20 VAR_OUTPUT
21     Count : DINT;
22 END_VAR
23 VAR
24     R_Confirm : R_TRIG;
25 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

7 ř.

```

1 R_Confirm(CLK := ConfirmSignal);
2
3 IF ResetCommand THEN
4     Count := 0;
5 ELSIF CountEnable AND R_Confirm.Q THEN
6     Count := Count + 1;
7 END_IF

```

5.3 FB_SortingCycle

FUNKČNÍ BLOK · 247 ŘÁDKŮ KÓDU · FSD KAP. 5, 6, 11

Vážení, klasifikace, nastavení třídiče, potvrzení dráhy a řízené zastavení. Instance Cycle, krok 50.10.

DEKLARACE · DECLARATION

98 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_SortingCycle
3  PURPOSE : Box sorting cycle IDLE .. SORT_FAULT: brings the box onto the
4            scale, weighs and classifies it, sets the sorter and waits
5            for the lane confirmation; runs the controlled stop.
6  NOTES   : Self-contained - sensors in through VAR_INPUT, actuators out
7            through VAR_OUTPUT, no GVL access. Called every scan with
8            Enable = AUTO OR DRAINING, so its timers never skip a scan.
9            Enable = FALSE: everything off, roller stops engaged, back to
10           IDLE - except SORT_FAULT, which is kept until Reset.

```

```

11      Stop: the load conveyors and Conv1 stop at once, the exit
12      lanes and the sorter run on until the line is empty
13      (STOPPING). A box already between Conv1 and the scale is
14      still weighed and sorted. After T_STOP_DRAIN the stop is
15      completed regardless of the sensors (safety net).
16      Empty exit lanes are judged by the InFlight counts of the
17      lane watchdogs, not by point sensors alone - a box between
18      two sensors is invisible to them.
19      Drained = IDLE, or RUN with all exit lanes empty (RUN is the
20      steady state while no Stop is latched).
21      WAIT_SCALE_ENTRY, WAIT_EXIT and SORT_CONFIRM have a timeout
22      -> SORT_FAULT -> GVL.LineJamFault -> MODE_EMERGENCY.
23  FSD REF : ch. 5      Sorting cycle
24            ch. 5.2    Controlled stop
25            ch. 6      Weight classification
26            ch. 11     Watchdogs and faults
27  ===== *)
28
29  FUNCTION_BLOCK FB_SortingCycle
30  VAR_INPUT
31      Enable      : BOOL;    (* ProcessEnable: Mode = AUTO OR DRAINING *)
32      StopRequested : BOOL;    (* FB_SafetyInterlock.StopRequested *)
33      ResetTrig    : BOOL;    (* R_Reset.Q - clears SORT_FAULT *)
34
35      Retro_S1     : BOOL;    (* box present at scale transfer entry *)
36      Diffuse_S1    : BOOL;    (* weight capture position (stop point) *)
37      Diffuse_S2    : BOOL;    (* scale exit confirmation *)
38      Diffuse_S3    : BOOL;    (* sort confirm - left path (S) *)
39      Diffuse_S4    : BOOL;    (* sort confirm - forward path (M) *)
40      Diffuse_S5    : BOOL;    (* sort confirm - right path (L) *)
41      Retro_S2     : BOOL;    (* left exit path - used here only for the
42                               STOPPING drain-clear check *)
43      Retro_S3     : BOOL;    (* forward exit path - ditto *)
44      Retro_S4     : BOOL;    (* right exit path - ditto *)
45
46      (* Boxes between Diffuse_S3/4/5 and Retro_S2/3/4 (PathWD_S/M/L, value from the
47       previous scan). Used by the drain checks. *)
48      InFlight_S : INT;
49      InFlight_M : INT;
50      InFlight_L : INT;
51
52      WeightAnalog : REAL;    (* continuous weight measurement, kg *)
53  END_VAR
54  VAR_OUTPUT
55      Conv1      : BOOL;
56      Conv2      : REAL;    (* exit lane drive, 5.0 = run, 0.0 = stop *)
57      Conv3      : REAL;    (* ditto *)
58      Conv4      : REAL;    (* ditto *)
59      ScaleConveyor : BOOL;
60      SorterLeft  : BOOL;
61      SorterForward : BOOL;
62      SorterRight : BOOL;
63      Drained     : BOOL;    (* IDLE, or RUN with exit lanes empty *)
64      Fault       : BOOL;    (* State = SORT_FAULT *)
65      State       : E_SortState := E_SortState.IDLE;    (* current state - HMI display *)
66      (* WeightClass: class of the box being sorted, CLS_NONE after the lane
67       confirmation *)
68      WeightClass : E_WeightClass := E_WeightClass.CLS_NONE;
69      CurrentWeight : REAL;    (* last settled weight, kg *)
70
71      RollStop_Entry : BOOL;    (* roller-stop brakes, entry section *)
72      RollStop_S     : BOOL;    (* roller-stop brakes, S exit path *)
73      RollStop_M     : BOOL;    (* roller-stop brakes, M exit path *)
74      RollStop_L     : BOOL;    (* roller-stop brakes, L exit path *)
75
76      LoadConv1 : REAL;    (* load conveyor drive, 0.0 = stop *)
77      LoadConv2 : REAL;    (* load conveyor drive, 0.0 = stop *)
78      LoadConv3 : REAL;    (* load conveyor drive, 0.0 = stop *)
79  END_VAR
80  VAR
81      ScaleOccupied : BOOL;
82
83      WeightSettleTimer : TON;
84      T_SETTLE          : TIME := T#300MS;    (* tune to your scale / load cell response *)
85

```

```

86   StopDrainTimer : TON;
87   T_STOP_DRAIN   : TIME := T#20S;      (* max. controlled run-down after Stop
88                                         (FSD ch. 5.2) *)
89
90   (* --- per-state jam watchdogs --- *)
91   EntryTimer     : TON;
92   T_ENTRY_TIMEOUT : TIME := T#4S;      (* Retro_S1 -> Diffuse_S1, tune to line *)
93   ExitTimer      : TON;
94   T_EXIT_TIMEOUT  : TIME := T#4S;      (* Diffuse_S1 -> Diffuse_S2, tune to line *)
95   ConfirmTimer   : TON;
96   T_CONFIRM_TIMEOUT : TIME := T#6S;    (* Diffuse_S2 -> Diffuse_S3/4/5, tune to
97                                         line *)
98   END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

276 ř.

```

1  (* Fault reset - evaluated ALWAYS, regardless of Enable/Mode, same
2  pattern as FB_PickPlace.PP_FAULT. *)
3  IF (State = E_SortState.SORT_FAULT) AND ResetTrig THEN
4      State := E_SortState.IDLE;
5      ScaleOccupied := FALSE;
6      WeightClass := E_WeightClass.CLS_NONE;
7      EntryTimer(IN := FALSE);
8      ExitTimer(IN := FALSE);
9      ConfirmTimer(IN := FALSE);
10  END_IF;
11
12  (* Runs every scan, independent of State, driven directly by StopRequested. *)
13  StopDrainTimer(IN := StopRequested, PT := T_STOP_DRAIN);
14
15  (* ---LOAD SIDE--- load conveyors run and the entry stop is open until Stop is latched *)
16  IF StopRequested THEN
17      RollStop_Entry := TRUE;
18      LoadConv1 := 0.0;
19      LoadConv2 := 0.0;
20      LoadConv3 := 0.0;
21  ELSE
22      RollStop_Entry := FALSE;
23      LoadConv1 := 4.0;
24      LoadConv2 := 1.0;
25      LoadConv3 := 1.0;
26  END_IF;
27
28  (* ---DISABLED (Enable = FALSE)--- everything off, roller stops engaged *)
29  IF NOT Enable THEN
30      Conv1 := FALSE;
31      Conv2 := 0.0;
32      Conv3 := 0.0;
33      Conv4 := 0.0;
34      ScaleConveyor := FALSE;
35      SorterLeft := FALSE;
36      SorterForward := FALSE;
37      SorterRight := FALSE;
38      RollStop_Entry := TRUE;
39      RollStop_S := TRUE;
40      RollStop_M := TRUE;
41      RollStop_L := TRUE;
42      LoadConv1 := 0.0;
43      LoadConv2 := 0.0;
44      LoadConv3 := 0.0;
45
46      (* SORT_FAULT survives Enable = FALSE - including the MODE_EMERGENCY it
47      triggers itself - and is cleared only by ResetTrig (top of the FB). *)
48      IF State <> E_SortState.SORT_FAULT THEN
49          State := E_SortState.IDLE;
50          ScaleOccupied := FALSE;
51          WeightClass := E_WeightClass.CLS_NONE;
52          WeightSettleTimer(IN := FALSE);
53      END_IF;
54      EntryTimer(IN := FALSE);
55      ExitTimer(IN := FALSE);
56      ConfirmTimer(IN := FALSE);
57      Drained := (State = E_SortState.IDLE);

```

```

58
59 ELSE
60     (* Safety net: once T_STOP_DRAIN has elapsed, force STOPPING from any state, so
61        a stop always completes even if a sensor never confirms. *)
62     IF StopDrainTimer.Q AND (State <> E_SortState.IDLE) THEN
63         State := E_SortState.STOPPING;
64     END_IF;
65
66 CASE State OF
67
68     E_SortState.IDLE:
69         ScaleOccupied := FALSE;
70         WeightClass   := E_WeightClass.CLS_NONE;
71         SorterLeft    := FALSE;
72         SorterForward := FALSE;
73         SorterRight   := FALSE;
74
75         (* Waits for StopRequested to clear so the machine can't
76            auto-restart out of IDLE while STOP is still active. *)
77         IF NOT StopRequested THEN
78             State := E_SortState.RUN;
79         END_IF
80
81     (* ---RUN--- *)
82     E_SortState.RUN:
83         Conv1      := NOT StopRequested;
84         Conv2      := 5.0;
85         Conv3      := 5.0;
86         Conv4      := 5.0;
87         SorterForward := TRUE; (* sorter drive runs continuously *)
88
89         LoadConv1 := 3.0;
90         LoadConv2 := 1.0;
91         LoadConv3 := 1.0;
92         RollStop_S := FALSE;
93         RollStop_M := FALSE;
94         RollStop_L := FALSE;
95
96         IF StopRequested THEN
97             State := E_SortState.STOPPING;
98         ELSIF Retro_S1 AND (NOT ScaleOccupied) THEN
99             ScaleOccupied := TRUE;
100             ScaleConveyor := TRUE;
101             State := E_SortState.WAIT_SCALE_ENTRY;
102         END_IF
103
104     (* ---WAIT_SCALE_ENTRY--- *)
105     E_SortState.WAIT_SCALE_ENTRY:
106         ScaleConveyor := TRUE;
107         (* Conv1 runs unconditionally, regardless of StopRequested -
108            this state exists specifically to retrieve a box already
109            caught between Conv1 and the scale during a stop. Gating
110            this on NOT StopRequested would strand that box before it
111            ever reaches Diffuse_S1. *)
112         Conv1 := TRUE;
113
114         (* Watchdog: box seen at Retro_S1 but not at Diffuse_S1 in time - jam
115            before the scale. *)
116         EntryTimer(IN := TRUE, PT := T_ENTRY_TIMEOUT);
117         IF EntryTimer.Q THEN
118             State := E_SortState.SORT_FAULT;
119         ELSIF Diffuse_S1 THEN
120             ScaleConveyor := FALSE;
121             EntryTimer(IN := FALSE);
122             WeightSettleTimer(IN := TRUE, PT := T_SETTLE);
123             State := E_SortState.WEIGHING;
124         END_IF
125
126     (* ---WEIGHING--- *)
127     E_SortState.WEIGHING:
128         ScaleConveyor := FALSE;
129         WeightSettleTimer(IN := TRUE, PT := T_SETTLE);
130
131         IF WeightSettleTimer.Q THEN
132             CurrentWeight := WeightAnalog;

```

```

133         WeightSettleTimer(IN := FALSE);
134         State := E_SortState.CLASSIFICATION;
135     END_IF
136
137     (* ---CLASSIFICATION--- *)
138     E_SortState.CLASSIFICATION:
139         (* Thresholds in F_ClassifyWeight. Counting does not happen here - the
140            counters use the confirmed lane exit (FSD ch. 14). *)
141         WeightClass := F_ClassifyWeight(Weight := CurrentWeight);
142         State := E_SortState.SORTING;
143
144     (* ---SORTING--- *)
145     E_SortState.SORTING:
146         (* SorterForward is NOT set here - it already runs
147            continuously from RUN/STOPPING. Only the direction
148            inputs are class-driven, and only S/L need one. *)
149         CASE WeightClass OF
150             E_WeightClass.CLS_S: SorterLeft := TRUE;
151             E_WeightClass.CLS_M: ; (* no direction output - default forward path *)
152             E_WeightClass.CLS_L: SorterRight := TRUE;
153         ELSE
154             ;
155         END_CASE
156
157         ScaleConveyor := TRUE; (* falls through to WAIT_EXIT *)
158         State := E_SortState.WAIT_EXIT;
159
160     (* ---WAIT_EXIT--- *)
161     E_SortState.WAIT_EXIT:
162         ScaleConveyor := TRUE;
163
164         (* Watchdog: weighed box does not leave the scale (Diffuse_S2) in time
165            - stuck on the scale. *)
166         ExitTimer(IN := TRUE, PT := T_EXIT_TIMEOUT);
167         IF ExitTimer.Q THEN
168             State := E_SortState.SORT_FAULT;
169         ELSIF Diffuse_S2 THEN
170             (* Scale freed immediately for throughput - independent
171                of whether the box has reached the sorter yet. The
172                sorter output set in SORTING is still held TRUE. *)
173             ScaleOccupied := FALSE;
174             ExitTimer(IN := FALSE);
175             State := E_SortState.SORT_CONFIRM;
176         END_IF
177
178     (* ---SORT_CONFIRM--- *)
179     E_SortState.SORT_CONFIRM:
180         (* Hold the direction output until the box passes the lane confirmation
181            sensor - the sorter inputs have no memory. SorterForward unchanged. *)
182
183         (* Watchdog: box left the scale but no lane confirmation in time -
184            stuck before or in the sorter. *)
185         ConfirmTimer(IN := TRUE, PT := T_CONFIRM_TIMEOUT);
186         IF ConfirmTimer.Q THEN
187             State := E_SortState.SORT_FAULT;
188         ELSIF Diffuse_S3 OR Diffuse_S4 OR Diffuse_S5 THEN
189             SorterLeft := FALSE;
190             SorterRight := FALSE;
191             WeightClass := E_WeightClass.CLS_NONE;
192             ConfirmTimer(IN := FALSE);
193
194             IF StopRequested THEN
195                 State := E_SortState.STOPPING;
196             ELSE
197                 State := E_SortState.RUN;
198             END_IF
199         END_IF
200
201     (* ---STOPPING--- *)
202     E_SortState.STOPPING:
203         Conv1 := FALSE; (* infeed stops immediately *)
204         Conv2 := 5.0; (* run-out belts keep running *)
205         Conv3 := 5.0;
206         Conv4 := 5.0;
207         SorterForward := TRUE; (* sorter keeps running *)

```

```

208      RollStop_Entry := TRUE;
209
210      RollStop_S := FALSE;
211      RollStop_M := FALSE;
212      RollStop_L := FALSE;
213      LoadConv1 := 0;
214      LoadConv2 := 0;
215      LoadConv3 := 0;
216
217      (* Line empty (or safety net elapsed): exit side off, back to IDLE.
218      A box arriving at Retro_S1 is still taken onto the scale. *)
219      IF StopDrainTimer.Q OR
220      ((NOT Retro_S1) AND (NOT Diffuse_S1) AND (NOT Diffuse_S2)
221      AND (NOT Retro_S2) AND (NOT Retro_S3) AND (NOT Retro_S4)
222      AND (InFlight_S = 0) AND (InFlight_M = 0) AND (InFlight_L = 0)
223      AND (NOT ScaleOccupied)) THEN
224
225          Conv2      := 0.0;
226          Conv3      := 0.0;
227          Conv4      := 0.0;
228          ScaleConveyor := FALSE;
229          SorterForward := FALSE;
230          SorterLeft   := FALSE;
231          SorterRight  := FALSE;
232          LoadConv1 := 0;
233          LoadConv2 := 0;
234          LoadConv3 := 0;
235
236          State := E_SortState.IDLE;
237
238      ELSIF Retro_S1 AND (NOT ScaleOccupied) THEN
239          ScaleOccupied := TRUE;
240          ScaleConveyor := TRUE;
241          State := E_SortState.WAIT_SCALE_ENTRY;
242      END_IF
243
244      (* ---SORT_FAULT--- *)
245      E_SortState.SORT_FAULT:
246          (* A watchdog expired. Hold the safe state until ResetTrig (top of the
247          FB). Needed here as well as in the disabled branch: Enable is still
248          TRUE in the scan the fault trips; MODE_EMERGENCY follows next scan. *)
249          Conv1      := FALSE;
250          Conv2      := 0.0;
251          Conv3      := 0.0;
252          Conv4      := 0.0;
253          ScaleConveyor := FALSE;
254          SorterLeft   := FALSE;
255          SorterForward := FALSE;
256          SorterRight  := FALSE;
257          RollStop_Entry := TRUE;
258          RollStop_S := TRUE;
259          RollStop_M := TRUE;
260          RollStop_L := TRUE;
261          LoadConv1 := 0.0;
262          LoadConv2 := 0.0;
263          LoadConv3 := 0.0;
264
265      END_CASE
266
267      (* Drained: IDLE, or RUN with nothing in flight on the exit lanes. It cannot be
268      TRUE in RUN while Stop is latched (RUN leaves to STOPPING in the same scan),
269      so the AUTO -> STOPPED gate is unaffected. *)
270      Drained := (State = E_SortState.IDLE)
271      OR (State = E_SortState.RUN
272      AND (InFlight_S = 0) AND (InFlight_M = 0) AND (InFlight_L = 0));
273
274      END_IF
275
276      Fault := (State = E_SortState.SORT_FAULT);

```

5.4 FB_PickPlace

FUNKČNÍ BLOK · 118 ŘÁDKŮ KÓDU · FSD KAP. 7

Dvouosý přesun krabice z palety na nakládací dopravník, řízený polohou. Instance `PickPlace`, krok 50.30.

DEKLARACE · DECLARATION

53 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_PickPlace
3  PURPOSE : Two-axis Pick & Place PnP1 (X and Z, no rotation): takes the
4             box off the pallet at the pick point and places it on
5             LoadConv1.
6  NOTES  : Xset / Zset are POSITION setpoints 0..10, not speeds.
7             Enable = FALSE freezes the arm: the actual position is
8             re-commanded every scan, GrabCmd and State are kept, so the
9             cycle continues from the same step when Enable returns.
10             Stop blocks only the start of a new cycle in PP_WAIT - a
11             transfer in progress is always completed.
12             Watchdogs: axis moves (T_AXIS_TIMEOUT), grip (T_GRAB_TIMEOUT),
13             item lost while lifting or moving -> PP_FAULT, held until
14             Reset (evaluated regardless of Enable) -> GVL.Fault_PnP1_Robot
15             -> MODE_EMERGENCY. Drained = PP_HOME or PP_WAIT (arm idle).
16  FSD REF : ch. 7      Pick & Place PnP1
17  ===== *)
18
19 FUNCTION_BLOCK FB_PickPlace
20 VAR_INPUT
21     Enable      : BOOL;    (* ProcessEnable - same gating as Cycle *)
22     StopRequested : BOOL;  (* blocks a NEW cycle starting from PP_WAIT -
23                             does not abort one in progress *)
24     ResetTrig   : BOOL;    (* R_Reset.Q - clears a Fault *)
25     PosPick     : BOOL;    (* GVL.Diffs_PosPick1 - pallet at the pick point *)
26     PosDrop     : BOOL;    (* GVL.Diffs_PosDrop1 - LoadConv1 drop point occupied *)
27     Detected    : BOOL;    (* GVL.PnP1_Detected - gripper proximity sensor *)
28     Xact        : REAL;    (* GVL.PnP1_Xact, position 0..10 *)
29     Zact        : REAL;    (* GVL.PnP1_Zact, position 0..10 *)
30 END_VAR
31 VAR_OUTPUT
32     Xset        : REAL;
33     Zset        : REAL;
34     GrabCmd     : BOOL;
35     Drained     : BOOL;    (* TRUE in PP_HOME or PP_WAIT - feeds LineDrained
36                             together with Cycle.Drained *)
37     Fault       : BOOL;
38     State       : E_PickPlaceState := E_PickPlaceState.PP_HOME;    (* current state - HMI display *)
39 END_VAR
40 VAR
41     X_HOME      : REAL := 0.0;
42     X_WAIT      : REAL := 10.0;
43     Z_UP        : REAL := 0.0;
44     Z_DOWN      : REAL := 10.0;    (* safety limit for the descent - NOT the
45                                     target depth, see PP_LOWER_PICK *)
46     POS_TOL     : REAL := 0.2;    (* position tolerance for 'axis reached' *)
47
48     XTmr        : TON;
49     ZTmr        : TON;
50     GrabTmr     : TON;
51     T_AXIS_TIMEOUT : TIME := T#3S;    (* axis-move watchdog *)
52     T_GRAB_TIMEOUT  : TIME := T#2S;    (* grip-confirm watchdog *)
53 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

112 ř.

```

1  IF (State = E_PickPlaceState.PP_FAULT) AND ResetTrig THEN
2     State := E_PickPlaceState.PP_HOME;    (* evaluated ALWAYS, regardless of Enable/Mode *)
3  END_IF;
4
5  IF NOT Enable THEN
6     Xset := Xact;
7     Zset := Zact;
8     XTmr(IN := FALSE);
9     ZTmr(IN := FALSE);
10    GrabTmr(IN := FALSE);

```

```

11      (* GrabCmd and State kept on purpose - resume from the same step *)
12
13  ELSE
14
15      CASE State OF
16
17          E_PickPlaceState.PP_HOME:
18              Xset := X_HOME;
19              Zset := Z_UP;
20              GrabCmd := FALSE;
21              XTmr(IN := FALSE);
22              ZTmr(IN := FALSE);
23              GrabTmr(IN := FALSE);
24              State := E_PickPlaceState.PP_WAIT;
25
26          E_PickPlaceState.PP_WAIT:
27              Xset := X_WAIT;
28              Zset := Z_UP;
29              GrabCmd := FALSE;
30
31              XTmr(IN := (ABS(Xact - X_WAIT) > POS_TOL), PT := T_AXIS_TIMEOUT);
32              IF XTmr.Q THEN
33                  State := E_PickPlaceState.PP_FAULT;
34              ELSIF (ABS(Xact - X_WAIT) <= POS_TOL)
35                  AND PosPick AND (NOT PosDrop) AND (NOT StopRequested) THEN
36                  (* StopRequested only blocks a NEW cycle starting here *)
37                  State := E_PickPlaceState.PP_LOWER_PICK;
38              END_IF
39
40          E_PickPlaceState.PP_LOWER_PICK:
41              Zset := Z_DOWN;
42
43              ZTmr(IN := (ABS(Zact - Z_DOWN) > POS_TOL) AND (NOT Detected),
44                  PT := T_AXIS_TIMEOUT);
45
46              IF Detected THEN
47                  (* Box detected - at whatever depth. Stop pushing
48                     further and hold Zset at the current position. *)
49                  Zset := Zact;
50                  ZTmr(IN := FALSE);
51                  State := E_PickPlaceState.PP_GRAB;
52              ELSIF ZTmr.Q THEN
53                  (* Still descending without detection after T_AXIS_TIMEOUT - axis
54                     stuck or no box under the gripper. *)
55                  State := E_PickPlaceState.PP_FAULT;
56              END_IF
57
58          E_PickPlaceState.PP_GRAB:
59              GrabCmd := TRUE;
60
61              GrabTmr(IN := (NOT Detected), PT := T_GRAB_TIMEOUT);
62              IF GrabTmr.Q THEN
63                  State := E_PickPlaceState.PP_FAULT; (* nothing detected in time *)
64              ELSIF Detected THEN
65                  GrabTmr(IN := FALSE);
66                  State := E_PickPlaceState.PP_LIFT;
67              END_IF
68
69          E_PickPlaceState.PP_LIFT:
70              Zset := Z_UP;
71              IF NOT Detected THEN
72                  State := E_PickPlaceState.PP_FAULT; (* lost the item while lifting *)
73              ELSE
74                  ZTmr(IN := (ABS(Zact - Z_UP) > POS_TOL), PT := T_AXIS_TIMEOUT);
75                  IF ZTmr.Q THEN
76                      State := E_PickPlaceState.PP_FAULT;
77                  ELSIF ABS(Zact - Z_UP) <= POS_TOL THEN
78                      State := E_PickPlaceState.PP_MOVE_TO_DROP;
79                  END_IF
80              END_IF
81
82          E_PickPlaceState.PP_MOVE_TO_DROP:
83              Xset := X_HOME;
84              IF NOT Detected THEN
85                  State := E_PickPlaceState.PP_FAULT;

```

```

86      ELSE
87          XTmr(IN := (ABS(Xact - X_HOME) > POS_TOL), PT := T_AXIS_TIMEOUT);
88          IF XTmr.Q THEN
89              State := E_PickPlaceState.PP_FAULT;
90          ELSIF ABS(Xact - X_HOME) <= POS_TOL THEN
91              State := E_PickPlaceState.PP_RELEASE;
92          END_IF
93      END_IF
94
95      E_PickPlaceState.PP_RELEASE:
96          GrabCmd := FALSE;
97          State := E_PickPlaceState.PP_HOME;
98
99      E_PickPlaceState.PP_FAULT:
100          (* Frozen just like the Enable = FALSE branch - hold position, wait for Reset *)
101          Xset := Xact;
102          Zset := Zact;
103          XTmr(IN := FALSE);
104          ZTmr(IN := FALSE);
105          GrabTmr(IN := FALSE);
106
107  END_CASE
108
109 END_IF
110
111 Drained := (State = E_PickPlaceState.PP_HOME) OR (State = E_PickPlaceState.PP_WAIT);
112 Fault := (State = E_PickPlaceState.PP_FAULT);

```

5.5 FB_PalletFeed

FUNKČNÍ BLOK · 108 ŘÁDKŮ KÓDU · FSD KAP. 8, 9

Jedenáct úseků a pět zarážek paletové trasy; v režimu Drain uzavírá vstup. Instance `PalletFeed`, krok 50.35.

DEKLARACE · DECLARATION

66 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_PalletFeed
3  PURPOSE : Pallet infeed: carries a loaded pallet from the Emitter
4            (Factory I/O scene, not PLC-controlled) to the PnP1 pick
5            point, holds it there until the box has been taken and sends
6            the empty pallet on to its exit.
7  NOTES   : Self-contained - sensors in through VAR_INPUT, drives and
8            roller stops out through VAR_OUTPUT, no GVL access.
9            Route: PallConvLoad_R -> Curv_R1..R3 -> Strait_1 -> Strait_2
10             (pick point) -> Strait_3 -> Curv_L1..L3 -> PallConvLoad_L.
11             Only Strait_1, Strait_2 and (during Drain) the entrance have
12             a stop condition; all other segments run constantly -
13             pallets queuing behind a stopped one is normal.
14             Unloaded pallet: DiffS_PosPick cannot tell a loaded pallet
15             from an empty one (the bare pallet also breaks the beam). A
16             pallet at the pick point counts as unloaded when PnP1 returns
17             to idle (rising edge of PnP1_Drained); PalletCleared then
18             releases it and is reset when the pallet leaves the sensor.
19             Stop: all drives stop and all five roller stops release at
20             once, ahead of Enable (Mode can still be AUTO while the box
21             line runs down).
22             Draining: only the entrance closes (PallConvLoad_R stops,
23             RollStop_LoadR engages), after a pallet still on LArray_LoadR
24             has passed. Whether the whole line is empty is decided in
25             PLC_PRG_ST / F_NextMode only.
26  FSD REF : ch. 8      Pallet infeed
27            ch. 9      System Drain
28  ===== *)
29
30 FUNCTION_BLOCK FB_PalletFeed
31 VAR_INPUT
32     Enable      : BOOL;    (* ProcessEnable: Mode = AUTO OR DRAINING *)
33     StopRequested : BOOL;  (* Safety.StopRequested - acts ahead of Enable *)
34     Draining     : BOOL;   (* Mode = MODE_DRAINING - closes the entrance only *)
35     DiffS_PallEntry : BOOL; (* GVL.DiffS_PallEntry - pallet waiting to
36                               transfer from Strait_1 onto Strait_2 *)
37     DiffS_PosPick  : BOOL;  (* GVL.DiffS_PosPick1 - pallet at the pick point

```

```

38      (same sensor as FB_PickPlace.PosPick *)
39      PnP1_Drained    : BOOL;  (* PickPlace.Drained - rising edge = pick cycle
40                               finished *)
41      LArray_LoadR    : BOOL;  (* GVL.LArray_LoadR - entrance sensor, used by the
42                               Drain close only *)
43  END_VAR
44  VAR_OUTPUT
45      PallConvLoad_R   : REAL;
46      PallConvCurv_R1 : REAL;
47      PallConvCurv_R2 : REAL;
48      PallConvCurv_R3 : REAL;
49      PallConvStrait_1 : REAL;
50      PallConvStrait_2 : REAL;
51      PallConvStrait_3 : REAL;
52      PallConvCurv_L1 : REAL;
53      PallConvCurv_L2 : REAL;
54      PallConvCurv_L3 : REAL;
55      PallConvLoad_L   : REAL;
56      RollStop_PallPick1 : BOOL;  (* pick point clamp, side 1 *)
57      RollStop_PallPick2 : BOOL;  (* pick point clamp, side 2 *)
58      RollStop_LoadR    : BOOL;  (* entrance stop *)
59      RollStop_LoadL    : BOOL;  (* exit stop *)
60      RollStop_Entry1   : BOOL;  (* Strait_1 entry stop *)
61  END_VAR
62  VAR
63      PalletCleared : BOOL;  (* TRUE once PnP1 has taken the box off the pallet at
64                              the pick point *)
65      R_PnP1Drained : R_TRIG;
66  END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

110 ř.

```

1  IF StopRequested THEN
2      (* Stop: immediate, ahead of Enable - same priority as the load conveyors in
3       FB_SortingCycle. *)
4      PallConvLoad_R   := 0.0;
5      PallConvCurv_R1 := 0.0;
6      PallConvCurv_R2 := 0.0;
7      PallConvCurv_R3 := 0.0;
8      PallConvStrait_1 := 0.0;
9      PallConvStrait_2 := 0.0;
10     PallConvStrait_3 := 0.0;
11     PallConvCurv_L1 := 0.0;
12     PallConvCurv_L2 := 0.0;
13     PallConvCurv_L3 := 0.0;
14     PallConvLoad_L   := 0.0;
15     (* all roller stops released *)
16     RollStop_PallPick1 := FALSE;
17     RollStop_PallPick2 := FALSE;
18     RollStop_LoadR    := FALSE;
19     RollStop_LoadL    := FALSE;
20     RollStop_Entry1   := FALSE;
21
22     (* PalletCleared is kept. The edge detector is called with FALSE, so a pick
23        finished during the stop is not evaluated. *)
24     R_PnP1Drained(CLK := FALSE);
25
26  ELSIF NOT Enable THEN
27      PallConvLoad_R   := 0.0;
28      PallConvCurv_R1 := 0.0;
29      PallConvCurv_R2 := 0.0;
30      PallConvCurv_R3 := 0.0;
31      PallConvStrait_1 := 0.0;
32      PallConvStrait_2 := 0.0;
33      PallConvStrait_3 := 0.0;
34      PallConvCurv_L1 := 0.0;
35      PallConvCurv_L2 := 0.0;
36      PallConvCurv_L3 := 0.0;
37      PallConvLoad_L   := 0.0;
38      (* Roller stops released, as in the Stop branch. PLC_PRG_ST does not use these
39         outputs outside AUTO / DRAINING. *)
40      RollStop_PallPick1 := FALSE;
41      RollStop_PallPick2 := FALSE;

```

```

42 RollStop_LoadR      := FALSE;
43 RollStop_LoadL      := FALSE;
44 RollStop_Entry1     := FALSE;
45
46 (* Clean idle - the pallet state is re-evaluated from the sensors on the next
47    start. *)
48 PalletCleared := FALSE;
49 R_PnP1Drained(CLK := FALSE);
50
51 ELSE
52   R_PnP1Drained(CLK := PnP1_Drained);
53
54   (* exit stop and Strait_1 stop released in normal operation *)
55   RollStop_LoadL := FALSE;
56   RollStop_Entry1 := FALSE;
57
58   IF NOT DiffS_PosPick THEN
59     (* Pick point clear - reset for the next pallet, whether this
60        scan's clearing is the current one finally leaving or simply
61        nothing having arrived yet. *)
62     PalletCleared := FALSE;
63   ELSIF R_PnP1Drained.Q THEN
64     PalletCleared := TRUE;
65   END_IF;
66
67   (* ---Entrance--- closes during Drain once LArray_LoadR is clear, and stays
68      closed: no new pallet can reach the sensor any more. *)
69   IF Draining AND (NOT LArray_LoadR) THEN
70     (* Drain: entrance closed *)
71     PallConvLoad_R := 0.0;
72     RollStop_LoadR := TRUE;
73   ELSE
74     (* not draining, or a pallet still on the sensor - let it pass *)
75     PallConvLoad_R := -4.0;
76     RollStop_LoadR := FALSE;
77   END_IF;
78
79   PallConvCurv_R1 := -8.0;
80   PallConvCurv_R2 := 8.0;
81   PallConvCurv_R3 := 8.0;
82
83   (* ---Strait_1--- holds the next pallet while the pick point is occupied *)
84   IF DiffS_PallEntry AND DiffS_PosPick THEN
85     PallConvStrait_1 := 0.0;
86   ELSE
87     PallConvStrait_1 := 8.0;
88   END_IF;
89
90   (* ---Strait_2, pick point--- belt stopped and both clamps engaged until the
91      box has been taken (PalletCleared). *)
92   IF (NOT DiffS_PosPick) OR PalletCleared THEN
93     PallConvStrait_2 := 8.0;
94     RollStop_PallPick1 := FALSE;
95     RollStop_PallPick2 := FALSE;
96   ELSE
97     PallConvStrait_2 := 0.0;
98     RollStop_PallPick1 := TRUE;
99     RollStop_PallPick2 := TRUE;
100  END_IF;
101
102  (* ---Exit side--- runs constantly *)
103  PallConvStrait_3 := 8.0;
104  PallConvCurv_L1 := 8.0;
105
106  PallConvCurv_L2 := 8.0;
107  PallConvCurv_L3 := -8.0;
108  PallConvLoad_L := 6.0;
109
110 END_IF

```

5.6 FB_PathWatchdog

FUNKČNÍ BLOK · 72 ŘÁDKŮ KÓDU · FSD KAP. 11

Fronta až 15 kusů s vlastním časovačem pro každý kus. Pět instancí PathWD_*, krok 50.40.

DEKLARACE · DECLARATION

58 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_PathWatchdog
3  PURPOSE : Transit-time watchdog for a path between two sensors that can
4             hold several items at once (boxes or pallets). Faults
5             (latched) if an item does not reach EndSignal within Timeout,
6             or if more items enter than can be tracked.
7  NOTES  : A single start/stop timer is not enough: on several paths
8             the next box enters before the previous one has left (e.g.
9             SORT_CONFIRM returns the cycle to RUN while the box is still
10            travelling to Retro_S2/S3/S4). A second StartSignal edge
11            would re-arm the timer and hide the first box if it stuck.
12            So each StartSignal edge enqueues a slot with its own TON
13            (FIFO ring buffer, N_MAX slots) and each EndSignal edge
14            dequeues the oldest. FIFO order holds because items cannot
15            overtake each other on one conveyor path.
16            Enable = FALSE (outside AUTO / DRAINING): sensor edges are
17            ignored and the slot timers stop; a TON restarts from zero,
18            so after an emergency every tracked item gets the full
19            Timeout again. The queue itself is kept.
20            OverflowFault: more than N_MAX items - EndSignal most likely
21            not responding. UnexpectedEnd: EndSignal with nothing queued
22            (sensor mismatch) - diagnostic only, does not set Fault.
23            N_MAX is an array bound, so it is a CONSTANT - raise it and
24            recompile if a path can queue more items.
25  FSD REF : ch. 11.1 Watching a path with several items
26  ===== *)
27
28  FUNCTION_BLOCK FB_PathWatchdog
29  VAR_INPUT
30      Enable      : BOOL;    (* ProcessEnable - edges and timers active only in AUTO /
31                               DRAINING *)
32      StartSignal : BOOL;    (* rising edge = an item entered the path *)
33      EndSignal   : BOOL;    (* rising edge = an item left the path - FIFO order *)
34      Timeout     : TIME;    (* max. transit time per item *)
35      ResetTrig   : BOOL;    (* R_Reset.Q - clears Fault and empties the queue *)
36  END_VAR
37  VAR_OUTPUT
38      Fault       : BOOL;    (* TimeoutFault OR OverflowFault, latched *)
39      TimeoutFault : BOOL;    (* a tracked item exceeded Timeout *)
40      OverflowFault : BOOL;    (* more items entered than N_MAX can track *)
41      UnexpectedEnd : BOOL;    (* EndSignal with nothing queued - diagnostic only *)
42      InFlight     : INT;     (* items currently tracked - diagnostic / HMI *)
43  END_VAR
44  VAR_CONSTANT
45      N_MAX : INT := 15;    (* max. items tracked at once - array bound, must stay
46                               CONSTANT *)
47  END_VAR
48  VAR
49      SlotActive : ARRAY[1..N_MAX] OF BOOL;
50      SlotTmr    : ARRAY[1..N_MAX] OF TON;
51      Head       : INT := 1; (* index of the oldest tracked item (FIFO front) *)
52      Tail       : INT := 1; (* index the next StartSignal edge will occupy *)
53      Count      : INT := 0;
54
55      R_Start : R_TRIG;
56      R_End   : R_TRIG;
57      k       : INT;
58  END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

51 ř.

```

1  IF ResetTrig THEN
2      FOR k := 1 TO N_MAX DO
3          SlotActive[k] := FALSE;
4          SlotTmr[k](IN := FALSE);
5      END_FOR;

```

```

6   Head      := 1;
7   Tail      := 1;
8   Count     := 0;
9   Fault     := FALSE;
10  TimeoutFault := FALSE;
11  OverflowFault := FALSE;
12  UnexpectedEnd := FALSE;
13  END_IF;
14
15  R_Start(CLK := StartSignal);
16  R_End(CLK := EndSignal);
17
18  IF Enable THEN
19    IF R_Start.Q THEN
20      IF Count < N_MAX THEN
21        SlotActive[Tail] := TRUE;
22        Tail := (Tail MOD N_MAX) + 1;
23        Count := Count + 1;
24      ELSE
25        OverflowFault := TRUE;
26      END_IF;
27    END_IF;
28
29    IF R_End.Q THEN
30      IF Count > 0 THEN
31        SlotActive[Head] := FALSE;
32        SlotTmr[Head](IN := FALSE);
33        Head := (Head MOD N_MAX) + 1;
34        Count := Count - 1;
35      ELSE
36        UnexpectedEnd := TRUE;
37      END_IF;
38    END_IF;
39  END_IF;
40
41  (* Slot timers run only while Enable = TRUE - a TON restarts from zero
42     after its IN was FALSE. *)
43  FOR k := 1 TO N_MAX DO
44    SlotTmr[k](IN := (SlotActive[k] AND Enable), PT := Timeout);
45    IF SlotTmr[k].Q THEN
46      TimeoutFault := TRUE;
47    END_IF;
48  END_FOR;
49
50  Fault := TimeoutFault OR OverflowFault;
51  InFlight := Count;

```

5.7 FB_StartDelay

FUNKČNÍ BLOK · 26 ŘÁDKŮ KÓDU · FSD KAP. 3.1

Potvrzení startu po 3 s ve STOPPED. Instance `StartDelay`, krok 50.20.

DEKLARACE · DECLARATION

30 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_StartDelay
3  PURPOSE : Confirms a Start press only after DelayTime (3 s) in
4            MODE_STOPPED; leaving STOPPED cancels the pending start.
5  NOTES   : StartConfirmed feeds F_NextMode instead of the raw button,
6            so MODE_AUTO is entered only after the delay. Armed (count-
7            down running) is also read by PLC_PRG_ST for the stack
8            lights, the start beep and the STARTING text.
9  FSD REF : ch. 3.1 Transitions
10           ch. 4 Signaling
11  ===== *)
12
13  FUNCTION_BLOCK FB_StartDelay
14  VAR_INPUT
15    StartButton : BOOL; (* Start_Button OR Start_Cmd_HMI - rising edge arms the
16                          delay *)
17    InStopped : BOOL; (* ModeThisScan = MODE_STOPPED - delay only runs
18                       while still in STOPPED *)

```

```

19     DelayTime    : TIME := T#3S;
20 END_VAR
21 VAR_OUTPUT
22     StartConfirmed : BOOL;    (* TRUE once DelayTime has elapsed - feeds
23                               F_NextMode instead of raw StartButton *)
24 END_VAR
25 VAR
26     R_Start : R_TRIG;
27     Armed   : BOOL;    (* countdown running - holds the press until the delay
28                       completes or STOPPED is left *)
29     Tmr      : TON;
30 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

15 ř.

```

1  R_Start(CLK := StartButton);
2  IF NOT InStopped THEN
3      (* Left STOPPED - clear the pending request so a stale delay can't
4       fire next time. *)
5      Armed := FALSE;
6      Tmr(IN := FALSE);
7      StartConfirmed := FALSE;
8  ELSE
9      IF R_Start.Q THEN
10         Armed := TRUE;
11     END_IF;
12
13     Tmr(IN := Armed, PT := DelayTime);
14     StartConfirmed := Tmr.Q;
15 END_IF

```

5.8 FB_Blink

FUNKČNÍ BLOK · 30 ŘÁDKŮ KÓDU · FSD KAP. 4.1

Generátor pulzů 400 / 400 ms pro všechny věže. Instance `Blink`, krok 80.

DEKLARACE · DECLARATION

20 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_Blink
3  PURPOSE : Square-wave pulse generator for the stack light blinking
4           (400 / 400 ms).
5  FSD REF : ch. 4.1   Stack lights
6  ===== *)
7
8  FUNCTION_BLOCK FB_Blink
9  VAR_INPUT
10     Enable : BOOL;
11     TimeOn  : TIME := T#400MS;
12     TimeOff : TIME := T#400MS;
13 END_VAR
14 VAR_OUTPUT
15     Pulse : BOOL;
16 END_VAR
17 VAR
18     TmrOn  : TON;
19     TmrOff : TON;
20 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

17 ř.

```

1  IF NOT Enable THEN
2      Pulse := FALSE;
3      TmrOn(IN := FALSE);
4      TmrOff(IN := FALSE);
5  ELSIF NOT Pulse THEN
6      TmrOff(IN := TRUE, PT := TimeOff);
7      IF TmrOff.Q THEN
8          TmrOff(IN := FALSE);
9          Pulse := TRUE;

```

```

10     END_IF
11 ELSE
12     TmrOn(IN := TRUE, PT := TimeOn);
13     IF TmrOn.Q THEN
14         TmrOn(IN := FALSE);
15         Pulse := FALSE;
16     END_IF
17 END_IF

```

5.9 FB_PulseAlarm

FUNKČNÍ BLOK · 44 ŘÁDKŮ KÓDU · FSD KAP. 4.2

N krátkých pípnutí po náběžné hraně. Pět instancí v kroku 80: nouze, start, stop, zaseknutí, Drain.

DEKLARACE · DECLARATION

24 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_PulseAlarm
3  PURPOSE : Siren pattern - PulseCount beeps (TimeOn / TimeOff) after a
4            rising edge on Trigger, then silent. A new edge restarts it.
5  FSD REF : ch. 4.2  Sirens
6  ===== *)
7
8  FUNCTION_BLOCK FB_PulseAlarm
9  VAR_INPUT
10     Trigger      : BOOL;
11     PulseCount   : INT  := 3;
12     TimeOn       : TIME := T#300MS;
13     TimeOff      : TIME := T#300MS;
14 END_VAR
15 VAR_OUTPUT
16     Alarm : BOOL;
17 END_VAR
18 VAR
19     R_Trigger : R_TRIG;
20     TmrOn     : TON;
21     TmrOff    : TON;
22     PulsesDone : INT;
23     Running   : BOOL;
24 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

29 ř.

```

1  R_Trigger(CLK := Trigger);
2
3  IF R_Trigger.Q THEN
4     Running := TRUE;
5     PulsesDone := 0;
6     Alarm := TRUE;
7     TmrOn(IN := FALSE);
8     TmrOff(IN := FALSE);
9  END_IF
10
11 IF Running THEN
12     IF Alarm THEN
13         TmrOn(IN := TRUE, PT := TimeOn);
14         IF TmrOn.Q THEN
15             TmrOn(IN := FALSE);
16             Alarm := FALSE;
17             PulsesDone := PulsesDone + 1;
18             IF PulsesDone >= PulseCount THEN
19                 Running := FALSE;
20             END_IF
21         END_IF
22     ELSE
23         TmrOff(IN := TRUE, PT := TimeOff);
24         IF TmrOff.Q THEN
25             TmrOff(IN := FALSE);
26             Alarm := TRUE;

```

```

27     END_IF
28     END_IF
29 END_IF

```

5.10 FB_TimedAlarm

FUNKČNÍ BLOK · 27 ŘÁDKŮ KÓDU · FSD KAP. 4.2

Souvislý tón 1 s při vstupu do AUTO. Instance `StartupAlarm`, krok 80.

DEKLARACE · DECLARATION

20 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_TimedAlarm
3  PURPOSE : Siren pattern - one continuous tone for Duration after a
4            rising edge on Trigger.
5  FSD REF : ch. 4.2 Sirens
6  ===== *)
7
8  FUNCTION_BLOCK FB_TimedAlarm
9  VAR_INPUT
10     Trigger : BOOL; (* rising edge starts the continuous tone *)
11     Duration : TIME := T#1S;
12 END_VAR
13 VAR_OUTPUT
14     Alarm : BOOL;
15 END_VAR
16 VAR
17     R_Trigger : R_TRIG;
18     Running : BOOL;
19     Tmr : TON;
20 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

16 ř.

```

1  R_Trigger(CLK := Trigger);
2
3  IF R_Trigger.Q THEN
4     Running := TRUE;
5     Alarm := TRUE; (* ON immediately, no pulsing - single continuous tone *)
6     Tmr(IN := FALSE);
7  END_IF;
8
9  IF Running THEN
10     Tmr(IN := TRUE, PT := Duration);
11     IF Tmr.Q THEN
12         Tmr(IN := FALSE);
13         Alarm := FALSE;
14         Running := FALSE;
15     END_IF;
16 END_IF

```

5.11 FB_AlarmHistory

FUNKČNÍ BLOK · 120 ŘÁDKŮ KÓDU · FSD KAP. 13.3

Kruhový zásobník 20 výskytů s časem vzniku a zániku a hotové řádky pro HMI. Instance `AlarmHistory`, krok 95.

DEKLARACE · DECLARATION

44 ř.

```

1  (* =====
2  FUNCTION_BLOCK : FB_AlarmHistory
3  PURPOSE : Watches NUM_ALARMS alarm bits and logs every occurrence in a
4            ring buffer: a rising edge opens a new row (TimeOn, Active),
5            the falling edge closes that SAME row (TimeOff). When the
6            buffer is full the oldest row is overwritten. Also formats
7            the rows into HMI strings, newest first.
8  NOTES : The buffer (Hist / Head / Count) is VAR_IN_OUT and owned by
9            the caller (GVL.AlarmHist*, RETAIN) - the FB keeps nothing
10           that must survive a power cycle. After a restart PrevBits
11           start at FALSE: an alarm still active is logged again with

```

```

12         the restart time, and rows left open at power loss are
13         closed with TimeOff = restart time (the real time is
14         unknown). Clear empties the buffer and resets PrevBits, so
15         live alarms are re-logged at once - clearing never hides a
16         live alarm. Lines are rebuilt only when something changed.
17         Reads only the GVL constants NUM_ALARMS, ALARM_HIST_SIZE and
18         ALARM_TEXT.
19     FSD REF : ch. 13.3 Alarm history
20     ===== *)
21
22 FUNCTION_BLOCK FB_AlarmHistory
23 VAR_INPUT
24     AlarmBits : ARRAY[1..GVL.NUM_ALARMS] OF BOOL; (* level, TRUE = alarm active *)
25     Now       : DT; (* local timestamp, supplied by the caller *)
26     ClearCmd  : BOOL; (* one-scan pulse (R_AlarmClear.Q) *)
27 END_VAR
28 VAR_IN_OUT
29     Hist : ARRAY[1..GVL.ALARM_HIST_SIZE] OF ST_AlarmEntry; (* GVL.AlarmHist *)
30     Head : INT; (* GVL.AlarmHistHead - newest row, 0 = empty *)
31     Count : INT; (* GVL.AlarmHistCount - rows in use *)
32 END_VAR
33 VAR_OUTPUT
34     Lines : ARRAY[1..GVL.ALARM_HIST_SIZE] OF STRING(60); (* [1] = newest *)
35     LineActive : ARRAY[1..GVL.ALARM_HIST_SIZE] OF BOOL;
36     ActiveCount : INT;
37 END_VAR
38 VAR
39     PrevBits : ARRAY[1..GVL.NUM_ALARMS] OF BOOL;
40     FirstScan : BOOL := TRUE;
41     Dirty : BOOL := TRUE; (* TRUE -> Lines rebuilt at the end of this call *)
42     i, k, idx : INT;
43     Txt : STRING(24);
44 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

115 ř.

```

1  (* --- sanity check of the retained indices (e.g. after ALARM_HIST_SIZE
2     was changed and downloaded) --- *)
3  IF (Head < 0) OR (Head > GVL.ALARM_HIST_SIZE) OR
4     (Count < 0) OR (Count > GVL.ALARM_HIST_SIZE) THEN
5     Head := 0;
6     Count := 0;
7     Dirty := TRUE;
8  END_IF
9
10 (* --- first scan after a restart: close rows left open at power loss --- *)
11 IF FirstScan THEN
12     FOR k := 1 TO GVL.ALARM_HIST_SIZE DO
13         IF Hist[k].Active THEN
14             Hist[k].Active := FALSE;
15             Hist[k].TimeOff := Now;
16         END_IF
17     END_FOR
18     FirstScan := FALSE;
19     Dirty := TRUE;
20 END_IF
21
22 (* --- clear --- *)
23 IF ClearCmd THEN
24     FOR k := 1 TO GVL.ALARM_HIST_SIZE DO
25         Hist[k].AlarmId := 0;
26         Hist[k].Active := FALSE;
27     END_FOR
28     FOR i := 1 TO GVL.NUM_ALARMS DO
29         PrevBits[i] := FALSE; (* live alarms get re-logged right below *)
30     END_FOR
31     Head := 0;
32     Count := 0;
33     Dirty := TRUE;
34 END_IF
35
36 (* --- edge detection and logging --- *)
37 ActiveCount := 0;

```

```

38 FOR i := 1 TO GVL.NUM_ALARMS DO
39   IF AlarmBits[i] THEN
40     ActiveCount := ActiveCount + 1;
41   END_IF
42
43   IF AlarmBits[i] AND NOT PrevBits[i] THEN
44     (* came: open a new row, overwriting the oldest when full *)
45     Head := Head + 1;
46     IF Head > GVL.ALARM_HIST_SIZE THEN
47       Head := 1;
48     END_IF
49     Hist[Head].AlarmId := i;
50     Hist[Head].TimeOn := Now;
51     Hist[Head].TimeOff := Now;
52     Hist[Head].Active := TRUE;
53     IF Count < GVL.ALARM_HIST_SIZE THEN
54       Count := Count + 1;
55     END_IF
56     Dirty := TRUE;
57
58   ELSIF PrevBits[i] AND NOT AlarmBits[i] THEN
59     (* went: close the newest still-open row of this alarm. Not found
60      = its row was already overwritten - nothing to do. *)
61     idx := Head;
62     FOR k := 1 TO Count DO
63       IF Hist[idx].Active AND (Hist[idx].AlarmId = i) THEN
64         Hist[idx].TimeOff := Now;
65         Hist[idx].Active := FALSE;
66         EXIT;
67       END_IF
68       idx := idx - 1;
69       IF idx < 1 THEN
70         idx := GVL.ALARM_HIST_SIZE;
71       END_IF
72     END_FOR
73     Dirty := TRUE;
74   END_IF
75   PrevBits[i] := AlarmBits[i];
76 END_FOR
77
78 (* --- HMI lines, newest first - only when something changed --- *)
79 IF Dirty THEN
80   idx := Head;
81   FOR k := 1 TO GVL.ALARM_HIST_SIZE DO
82     IF k <= Count THEN
83       IF (Hist[idx].AlarmId >= 1) AND (Hist[idx].AlarmId <= GVL.NUM_ALARMS) THEN
84         Txt := GVL.ALARM_TEXT[Hist[idx].AlarmId];
85       ELSE
86         Txt := '?';
87       END_IF
88
89       IF Hist[idx].Active THEN
90         Lines[k] := CONCAT(CONCAT(
91           F_FormatAlarmTime(Stamp := Hist[idx].TimeOn, WithDate := TRUE),
92           ' ACTIVE '),
93           Txt);
94       ELSE
95         Lines[k] := CONCAT(CONCAT(CONCAT(CONCAT(
96           F_FormatAlarmTime(Stamp := Hist[idx].TimeOn, WithDate := TRUE),
97           ' '),
98           F_FormatAlarmTime(Stamp := Hist[idx].TimeOff, WithDate := FALSE)),
99           ' '),
100           Txt);
101       END_IF
102       LineActive[k] := Hist[idx].Active;
103
104       idx := idx - 1;
105       IF idx < 1 THEN
106         idx := GVL.ALARM_HIST_SIZE;
107       END_IF
108     ELSE
109       Lines[k] := '';
110       LineActive[k] := FALSE;
111     END_IF
112

```

```
113     END_FOR
114     Dirty := FALSE;
115 END_IF
```

KAPITOLA 6

Hlavní program Main program · PLC_PRG_ST

PLC_PRG_ST drží jediný stav linky, slučuje fyzické vstupy s povely HMI, volá všechny bloky v pevném pořadí kroků 10–95 (diagram 1.2) a zapisuje výstupy podle režimu.

6.1 PLC_PRG_ST

PROGRAM · 624 ŘÁDKŮ KÓDU · FSD KAP. 3, 4, 9 · PŘÍL. C

Režim linky, pořadí scanu, sloučení povelů a zápis výstupů. Kroky 10–95 jsou v kódu označené hlavičkami.

DEKLARACE · DECLARATION

160 ř.

```

1  (* =====
2  PROGRAM : PLC_PRG_ST
3  PURPOSE : Main program. Holds the line mode (Mode - the single source
4            of truth), merges physical inputs with HMI commands, calls
5            every block once per scan in a fixed order and writes all
6            outputs according to the mode. Process logic lives in the
7            function blocks; this program connects them.
8  NOTES   : Scan order
9            10 Retro sensor polarity (light-operate -> *_Present)
10           15 Mode selector interlock (HMI 2-position selector)
11           20 Safety and Stop, Reset / counter reset / Drain edges
12           30 Production counters
13           40 Faults, HARD STOP, emergency pre-emption
14              -> ModeThisScan, ProcessEnable
15           50 Process blocks: Cycle, StartDelay, PickPlace,
16              PalletFeed, five path watchdogs
17           60 F_NextMode -> Mode for the next scan
18           70 Outputs per mode
19           80 Stack lights and sirens
20           90 HMI status texts
21           95 Alarm history (last, observer only)
22           Steps 20 to 40 use Mode as it was at scan entry; steps 50
23           to 90 use ModeThisScan. Steps 40 and 70 must both use
24           ModeThisScan, never the result of step 60.
25           One-scan lag: Fault_* and InFlight_* in step 40 come from
26           the previous scan, because their blocks run in step 50.
27           Harmless at PLC scan rates - e.g. a Reset clears a jam
28           within two scans.
29           HARD STOP (step 40): AUTO / DRAINING with the selector on
30           Manual, or MANUAL with Stop, HMI Stop or the selector on
31           Auto -> MODE_STOPPED in the same scan, without run-down and
32           without Reset. Outputs use the MODE_EMERGENCY branch (drives
33           0, roller stops released, PnP1 holds its position) with the
34           grip frozen. From MANUAL a latch blocks the return to MANUAL
35           until Manual is selected again or AUTO is reached. All
36           manual HMI trims are cleared; physical Man_* inputs cannot
37           be cleared by the PLC and act again in MANUAL if still on.
38           System Drain: Drain in AUTO -> MODE_DRAINING. The process
39           blocks keep running (ProcessEnable), only the pallet
40           entrance closes. MODE_DRAINED needs both sides empty for
41           1 s continuously; DrainTmr limits the whole drain to
42           T_DRAIN_TIMEOUT (Fault_Draining).
43  FSD REF : ch. 3   Operating modes
44            ch. 4   Signaling
45            ch. 9   System Drain
46            ch. 13  HMI interface
47            app. C  Program structure
48  ===== *)
49
50 PROGRAM PLC_PRG_ST
51 VAR
52     Mode          : E_Mode := E_Mode.MODE_STOPPED;  (* persists across
53                                                         scans - the single source of truth for Mode *)
54     ModeThisScan : E_Mode;  (* Mode after step 40 (emergency, HARD STOP) - used
55                             for dispatch *)
56
57     R_Reset       : R_TRIG;  (* Reset edge: Reset_Button OR Reset_Cmd_HMI. Leaves

```

```

58             EMERGENCY / DRAINED, clears all faults - not the
59             counters *)
60 R_ResetCounters : R_TRIG;  (* counter reset edge: ResButtCounting1 OR
61             ResetCounting_Cmd_HMI - all six counters *)
62 R_Drain : R_TRIG;  (* Drain request edge: Drain_Button OR Drain_Cmd_HMI *)
63
64 Safety      : FB_SafetyInterlock;
65 CounterS    : FB_ProductionCounter;
66 CounterM    : FB_ProductionCounter;
67 CounterL    : FB_ProductionCounter;
68 CounterBoxIn : FB_ProductionCounter;  (* GVL.Diffs_PosDrop1 edge *)
69 CounterPalletIn : FB_ProductionCounter;  (* GVL.LArray_LoadR edge *)
70 CounterPalletOut : FB_ProductionCounter;  (* GVL.LArray_LoadL edge *)
71 Cycle       : FB_SortingCycle;
72 PickPlace   : FB_PickPlace;
73 PalletFeed   : FB_PalletFeed;  (* pallet infeed to PnP1 *)
74 StartDelay   : FB_StartDelay;
75
76 (* --- path / jam watchdogs (FB_PathWatchdog) --- *)
77 PathWD_PnP1  : FB_PathWatchdog;  (* DiffS_PosDrop1 -> Retro_S1 *)
78 PathWD_S     : FB_PathWatchdog;  (* Diffuse_S3 -> Retro_S2, Conv2 *)
79 PathWD_M     : FB_PathWatchdog;  (* Diffuse_S4 -> Retro_S3, Conv3 *)
80 PathWD_L     : FB_PathWatchdog;  (* Diffuse_S5 -> Retro_S4, Conv4 *)
81 PathWD_Pallet : FB_PathWatchdog;  (* LArray_LoadR -> LArray_LoadL, whole
82             pallet feed *)
83
84 T_DROP_TO_ENTRY : TIME := T#30S;  (* longest/least certain segment -
85             tune to measured transit time *)
86 T_EXIT_S        : TIME := T#6S;  (* independent per lane in case
87             Conv2/3/4 ever differ in length *)
88 T_EXIT_M        : TIME := T#6S;
89 T_EXIT_L        : TIME := T#6S;
90 T_PALLET_PATH   : TIME := T#120S;  (* whole pallet feed - tune to measured
91             transit time *)
92
93 (* --- System Drain --- *)
94 DrainTmr       : TON;  (* whole-process watchdog, IN := (Mode
95             = MODE_DRAINING) *)
96 T_DRAIN_TIMEOUT : TIME := T#120S;  (* independent of T_PALLET_PATH *)
97 LastPalletSettleTmr : TON;  (* IN := GVL.InFlight_PalletPath <= 0 *)
98 T_PALLET_SETTLE : TIME := T#1S;
99 BoxSideSettleTmr : TON;  (* IN := LineDrained AND PnP1PathClear *)
100 T_BOX_SIDE_SETTLE : TIME := T#1S;  (* same value as T_PALLET_SETTLE,
101             kept as its own named constant since
102             the two sides may need different
103             tuning later *)
104 ProcessEnable   : BOOL;  (* ModeThisScan = AUTO OR DRAINING - Enable of
105             every process block in step 50 *)
106
107 Blink           : FB_Blink;
108 EmergencyAlarm  : FB_PulseAlarm;
109 StartupAlarm    : FB_TimedAlarm;
110 StartBeep       : FB_PulseAlarm;  (* 1 short beep when the start countdown
111             begins *)
112 StopBeep        : FB_PulseAlarm;  (* 3 short beeps when Stop is pressed *)
113 JamBeep         : FB_PulseAlarm;  (* 4 short beeps when a jam fault latches *)
114 DrainBeep       : FB_PulseAlarm;  (* 2 short beeps on entering MODE_DRAINING *)
115
116 j : INT;
117 TowerRed, TowerYellow, TowerGreen : BOOL;
118 LineDrained : BOOL;  (* Cycle.Drain AND PickPlace.Drain - Stop gate in
119             F_NextMode, debounced for the drain completion *)
120 PnP1PathClear : BOOL;  (* no box between PnP1 and the scale *)
121
122 (* --- Retro_S1..S4, active-high after step 10 --- *)
123 Retro_S1_Present : BOOL;
124 Retro_S2_Present : BOOL;
125 Retro_S3_Present : BOOL;
126 Retro_S4_Present : BOOL;
127
128 (* --- mode selector, see step 15 --- *)
129 R_SelAutoCmd     : R_TRIG;  (* rising edge of GVL_HMI.SelModeAuto_Cmd_HMI *)
130 R_SelManCmd      : R_TRIG;  (* rising edge of GVL_HMI.SelModeManual_Cmd_HMI *)
131 SelModeManualEff : BOOL;  (* Sel_Mode_Manual OR SelModeManual_Cmd_HMI, after
132             the interlock *)

```

```

133 SelModeAutoEff : BOOL; (* Sel_Mode_Auto OR SelModeAuto_Cmd_HMI - Stop
134                      source in MANUAL (step 20) *)
135
136 (* --- alarm history, see step 95 --- *)
137 AlarmHistory : FB_AlarmHistory;
138 R_AlarmClear : R_TRIG; (* GVL_HMI.AlarmHistClear_Cmd_HMI edge *)
139 AlarmBits : ARRAY[1..GVL.NUM_ALARMS] OF BOOL;
140 RtcResult : UDINT; (* SysTimeRtc result code, 0 = OK *)
141 RtcUtc : UDINT; (* seconds since 1970, UTC *)
142 RtcLocal : DWORD; (* same, converted to local time (incl. DST) -
143                   DWORD because the library's VAR_IN_OUT
144                   pdwTimestampLocal is DWORD *)
145 NowLocal : DT;
146
147 (* --- HARD STOP, see step 40 --- *)
148 HardStopAuto : BOOL; (* this scan: AUTO/DRAINING + selector Manual *)
149 HardStopManual : BOOL; (* this scan: MANUAL + Safety.StopActive *)
150 HardStopNow : BOOL; (* HardStopAuto OR HardStopManual *)
151 HardStopLatch : BOOL; (* set by HardStopManual; blocks STOPPED -> MANUAL
152                       until Manual is selected again or AUTO is
153                       reached *)
154 HardStopOut : BOOL; (* HardStopNow OR HardStopLatch - selects the
155                     emergency output branch in MODE_STOPPED *)
156 HardStopGrabHold : BOOL; (* PnP1_GrabCmd frozen at the hard stop *)
157 R_SelManEff : R_TRIG; (* rising edge of SelModeManualEff *)
158 OutMode : E_Mode; (* step 70 dispatch: ModeThisScan, or
159                   MODE_EMERGENCY for a hard stop *)
160 END_VAR

```

IMPLEMENTACE · IMPLEMENTATION

780 ř.

```

1  (* =====
2  10. RETRO SENSOR POLARITY
3  Retro_S1..S4 are light-operate: beam clear = TRUE, box present = FALSE -
4  the opposite of every Diffuse_* sensor. Inverted once, here, because
5  the raw signals have three consumers (sorting cycle, counters, path
6  watchdogs).
7  ===== *)
8  Retro_S1_Present := NOT GVL.Retro_S1;
9  Retro_S2_Present := NOT GVL.Retro_S2;
10 Retro_S3_Present := NOT GVL.Retro_S3;
11 Retro_S4_Present := NOT GVL.Retro_S4;
12
13 (* =====
14 15. MODE SELECTOR INTERLOCK
15 The two HMI selector bits behave as a 2-position switch: the rising
16 edge of one clears the other; if both arrive in the same scan, Auto
17 wins. Requires both HMI buttons to write TRUE to their own bit - a
18 Toggle Variable could switch itself off without an edge, which this
19 interlock cannot undo. The physical selector is exclusive by
20 construction and is only OR-ed in.
21 ===== *)
22 R_SelAutoCmd(CLK := GVL_HMI.SelModeAuto_Cmd_HMI);
23 R_SelManCmd(CLK := GVL_HMI.SelModeManual_Cmd_HMI);
24
25 IF R_SelAutoCmd.Q THEN
26   GVL_HMI.SelModeManual_Cmd_HMI := FALSE;
27 ELSIF R_SelManCmd.Q THEN
28   GVL_HMI.SelModeAuto_Cmd_HMI := FALSE;
29 END_IF
30
31 SelModeManualEff := GVL.Sel_Mode_Manual OR GVL_HMI.SelModeManual_Cmd_HMI;
32 SelModeAutoEff := GVL.Sel_Mode_Auto OR GVL_HMI.SelModeAuto_Cmd_HMI;
33
34 (* =====
35 20. SAFETY, STOP AND COMMAND EDGES (FSD ch. 10)
36 Physical buttons merged with HMI commands. The Stop latch is reset
37 with Mode as at scan entry.
38 ===== *)
39 (* NO buttons: physical input OR HMI command (see GVL_HMI) *)
40 R_Reset(CLK := GVL.Reset_Button OR GVL_HMI.Reset_Cmd_HMI);
41 R_ResetCounters(CLK := GVL.ResButtCounting1 OR GVL_HMI.ResetCounting_Cmd_HMI);
42 R_Drain(CLK := GVL.Drain_Button OR GVL_HMI.Drain_Cmd_HMI);

```

```

43
44 (* NC channels: an HMI command simulates a trip (AND NOT *)
45 GVL.Emergency_Buttons[1] := GVL.Emergency_Button AND NOT GVL_HMI.EStop_Cmd_HMI;
46 GVL.Emergency_Buttons[2] := GVL.Emergency_Button_S AND NOT GVL_HMI.EStop_S_Cmd_HMI;
47 GVL.Emergency_Buttons[3] := GVL.Emergency_Button_M AND NOT GVL_HMI.EStop_M_Cmd_HMI;
48 GVL.Emergency_Buttons[4] := GVL.Emergency_Button_L AND NOT GVL_HMI.EStop_L_Cmd_HMI;
49 GVL.Emergency_Buttons[5] := GVL.EButt_PallLoad_L AND NOT GVL_HMI.EStop_PallL_Cmd_HMI;
50 GVL.Emergency_Buttons[6] := GVL.EButt_PallLoad_R AND NOT GVL_HMI.EStop_PallR_Cmd_HMI;
51
52 (* door channels (NC) *)
53 GVL.Safety_Doors[1] := GVL.SafetyDoorSorting1;
54 GVL.Safety_Doors[2] := GVL.SafetyDoorSorting2;
55 GVL.Safety_Doors[3] := GVL.SafetyDoorSorting3;
56 GVL.Safety_Doors[4] := GVL.SafetyDoorSorting4;
57 GVL.Safety_Doors[5] := GVL.SafetyDoorLoding1;
58
59 (* Stop (NC): physical Stop AND NOT HMI Stop. In MODE_MANUAL, selecting
60 Auto also acts as Stop -> HARD STOP in step 40. *)
61 Safety(RawEmergencyButtons := GVL.Emergency_Buttons,
62       RawSafetyDoors      := GVL.Safety_Doors,
63       RawStopButton       := GVL.Stop_Button AND NOT GVL_HMI.Stop_Cmd_HMI
64       AND NOT ((Mode = E_Mode.MODE_MANUAL) AND SelModeAutoEff),
65       SystemAtStopped     := (Mode = E_Mode.MODE_STOPPED));
66 GVL.Emergency_Tripped := Safety.EmergencyTripped;
67 GVL.Door_Open        := Safety.DoorOpen;
68
69 (* =====
70 30. PRODUCTION COUNTERS (FSD ch. 14)
71 Enabled in AUTO and DRAINING, with Mode as at scan entry - boxes and
72 pallets finished during a drain are real production. All six
73 counters share one reset edge.
74 ===== *)
75 CounterS(ConfirmSignal := Retro_S2_Present, ResetCommand := R_ResetCounters.Q,
76         CountEnable := (Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING));
77 CounterM(ConfirmSignal := Retro_S3_Present, ResetCommand := R_ResetCounters.Q,
78         CountEnable := (Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING));
79 CounterL(ConfirmSignal := Retro_S4_Present, ResetCommand := R_ResetCounters.Q,
80         CountEnable := (Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING));
81
82 (* boxes in (placed by PnP1), pallets in / out *)
83 CounterBoxIn(ConfirmSignal := GVL.DiffS_PosDrop1, ResetCommand := R_ResetCounters.Q,
84             CountEnable := (Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING));
85 CounterPalletIn(ConfirmSignal := GVL.LArray_LoadR, ResetCommand := R_ResetCounters.Q,
86              CountEnable := (Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING));
87 CounterPalletOut(ConfirmSignal := GVL.LArray_LoadL, ResetCommand := R_ResetCounters.Q,
88               CountEnable := (Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING));
89
90 (* copy to the RETAIN counters *)
91 GVL.Counter_S := CounterS.Count;
92 GVL.Counter_M := CounterM.Count;
93 GVL.Counter_L := CounterL.Count;
94 GVL.Counter_BoxIn := CounterBoxIn.Count;
95 GVL.Counter_PalletIn := CounterPalletIn.Count;
96 GVL.Counter_PalletOut := CounterPalletOut.Count;
97
98 (* display copies for the HMI *)
99 GVL.CounterDis_Exit_S := CounterS.Count;
100 GVL.CounterDis_Exit_M := CounterM.Count;
101 GVL.CounterDis_Exit_L := CounterL.Count;
102 GVL.CounterDis_Exit_Total := CounterS.Count + CounterM.Count + CounterL.Count;
103 GVL.CounterDis_BoxIn := CounterBoxIn.Count;
104 GVL.CounterDis_PalletIn := CounterPalletIn.Count;
105 GVL.CounterDis_PalletOut := CounterPalletOut.Count;
106
107 (* =====
108 40. FAULTS, HARD STOP AND EMERGENCY PRE-EMPTION (FSD ch. 3.1, 10, 11)
109 Highest priority - applied before any mode-specific logic, with Mode
110 as at scan entry. Fault_* and InFlight_* come from the previous scan
111 (their blocks run in step 50). ModeThisScan is captured at the end.
112 ===== *)
113 GVL.Fault_SortCycle := Cycle.Fault;
114 GVL.Fault_PnP1_Path := PathWD_PnP1.Fault;
115 GVL.Fault_PnP1_Robot := PickPlace.Fault;
116 GVL.Fault_ExitPath_S := PathWD_S.Fault;
117 GVL.Fault_ExitPath_M := PathWD_M.Fault;

```

```

118 GVL.Fault_ExitPath_L := PathWD_L.Fault;
119 GVL.Fault_PalletPath := PathWD_Pallet.Fault;
120
121 (* System Drain watchdog. Latched and cleared only by Reset: the fault
122    itself ends MODE_DRAINING, so an unlatched value would clear before
123    the operator could see it. *)
124 DrainTmr(IN := (Mode = E_Mode.MODE_DRAINING), PT := T_DRAIN_TIMEOUT);
125 IF DrainTmr.Q THEN
126     GVL.Fault_Draining := TRUE;
127 END_IF;
128 IF R_Reset.Q THEN
129     GVL.Fault_Draining := FALSE;
130 END_IF;
131
132 GVL.LineJamFault      := GVL.Fault_SortCycle OR GVL.Fault_PnP1_Path OR
133                          GVL.Fault_PnP1_Robot OR
134                          GVL.Fault_ExitPath_S OR GVL.Fault_ExitPath_M OR
135                          GVL.Fault_ExitPath_L OR GVL.Fault_PalletPath OR
136                          GVL.Fault_Draining;
137
138 GVL.InFlight_PnP1_Path := PathWD_PnP1.InFlight;
139 (* A box picked by PnP1 but not yet at Retro_S1 is not part of Cycle's
140    state - the drain completion checks this path separately. *)
141 PnP1PathClear          := (GVL.InFlight_PnP1_Path <= 0);
142 GVL.InFlight_ExitPath_S := PathWD_S.InFlight;
143 GVL.InFlight_ExitPath_M := PathWD_M.InFlight;
144 GVL.InFlight_ExitPath_L := PathWD_L.InFlight;
145 GVL.InFlight_PalletPath := PathWD_Pallet.InFlight;
146
147 (* Pallet side empty for T_PALLET_SETTLE. The FIFO count drops when
148    LArray_LoadL trips, before the pallet has physically left. *)
149 LastPalletSettleTmr(IN := (GVL.InFlight_PalletPath <= 0), PT := T_PALLET_SETTLE);
150
151 (* HARD STOP - before the emergency pre-emption, so an emergency in the
152    same scan still wins. *)
153 R_SelManEff(CLK := SelModeManualEff);
154 IF R_SelManEff.Q OR (Mode = E_Mode.MODE_AUTO) THEN
155     HardStopLatch := FALSE;
156 END_IF
157
158 HardStopAuto      := ((Mode = E_Mode.MODE_AUTO) OR (Mode = E_Mode.MODE_DRAINING))
159                      AND SelModeManualEff;
160 HardStopManual    := (Mode = E_Mode.MODE_MANUAL) AND Safety.StopActive;
161 HardStopNow       := HardStopAuto OR HardStopManual;
162
163 IF HardStopNow THEN
164     Mode := E_Mode.MODE_STOPPED;
165     IF HardStopManual THEN
166         HardStopLatch := TRUE;
167     END_IF
168
169     (* Freeze the grip as it is right now (GVL still holds last scan's
170        output), and clear every MANUAL command trim so that entering
171        MODE_MANUAL afterwards starts from a fully stopped machine. *)
172     HardStopGrabHold := GVL.PnP1_GrabCmd;
173
174     GVL_HMI.Conv1_Cmd_HMI      := FALSE;
175     GVL_HMI.Conv2_Cmd_HMI      := 0.0;
176     GVL_HMI.Conv3_Cmd_HMI      := 0.0;
177     GVL_HMI.Conv4_Cmd_HMI      := 0.0;
178     GVL_HMI.Scale_Conveyor_Cmd_HMI := FALSE;
179     GVL_HMI.Sorter_Left_Cmd_HMI := FALSE;
180     GVL_HMI.Sorter_Forward_Cmd_HMI := FALSE;
181     GVL_HMI.Sorter_Right_Cmd_HMI := FALSE;
182     GVL_HMI.RollStop_Entry_Cmd_HMI := FALSE;
183     GVL_HMI.RollStop_S_Cmd_HMI := FALSE;
184     GVL_HMI.RollStop_M_Cmd_HMI := FALSE;
185     GVL_HMI.RollStop_L_Cmd_HMI := FALSE;
186     GVL_HMI.LoadConv1_Cmd_HMI := 0.0;
187     GVL_HMI.LoadConv2_Cmd_HMI := 0.0;
188     GVL_HMI.LoadConv3_Cmd_HMI := 0.0;
189
190     GVL_HMI.PallConvLoad_R_Cmd_HMI := 0.0;
191     GVL_HMI.PallConvCurv_R1_Cmd_HMI := 0.0;
192     GVL_HMI.PallConvCurv_R2_Cmd_HMI := 0.0;

```

```

193   GVL_HMI.PallConvCurv_R3_Cmd_HMI := 0.0;
194   GVL_HMI.PallConvStrait_1_Cmd_HMI := 0.0;
195   GVL_HMI.PallConvStrait_2_Cmd_HMI := 0.0;
196   GVL_HMI.PallConvStrait_3_Cmd_HMI := 0.0;
197   GVL_HMI.PallConvCurv_L1_Cmd_HMI := 0.0;
198   GVL_HMI.PallConvCurv_L2_Cmd_HMI := 0.0;
199   GVL_HMI.PallConvCurv_L3_Cmd_HMI := 0.0;
200   GVL_HMI.PallConvLoad_L_Cmd_HMI := 0.0;
201   GVL_HMI.RollStop_PallPick1_Cmd_HMI := FALSE;
202   GVL_HMI.RollStop_PallPick2_Cmd_HMI := FALSE;
203   GVL_HMI.RollStop_LoadR_Cmd_HMI := FALSE;
204   GVL_HMI.RollStop_LoadL_Cmd_HMI := FALSE;
205   GVL_HMI.RollStop_Entry1_Cmd_HMI := FALSE;
206
207   (* PnP1: X/Z are POSITIONS - zeroing them would drive the arm to 0/0.
208      Set the trims so MANUAL's LIMIT(Man_* + Cmd_HMI) lands exactly on
209      the actual position instead; grip trim takes the held grip. *)
210   GVL_HMI.PnP1_Xset_Cmd_HMI := GVL.PnP1_Xact - GVL.Man_PnP1_Xset;
211   GVL_HMI.PnP1_Zset_Cmd_HMI := GVL.PnP1_Zact - GVL.Man_PnP1_Zset;
212   GVL_HMI.PnP1_GrabCmd_Cmd_HMI := HardStopGrabHold;
213 END_IF
214
215   HardStopOut := HardStopNow OR HardStopLatch;
216
217   (* Emergency pre-emption: E-stop, door or any latched fault *)
218   IF Safety.EmergencyActive OR GVL.LineJamFault THEN
219     Mode := E_Mode.MODE_EMERGENCY;
220 END_IF
221   ModeThisScan := Mode;
222
223   (* Enable of all process blocks in step 50 *)
224   ProcessEnable := (ModeThisScan = E_Mode.MODE_AUTO) OR (ModeThisScan = E_Mode.MODE_DRAINING);
225
226   (* =====
227      50. PROCESS BLOCKS (FSD ch. 5, 7, 8, 11)
228      Enabled in MODE_AUTO and MODE_DRAINING (ProcessEnable). Each block
229      brings its own outputs to a safe idle when disabled.
230      ===== *)
231   (* 50.10. Main sorting cycle - see FB_SortingCycle header. *)
232   Cycle(Enable := ProcessEnable,
233         StopRequested := Safety.StopRequested,
234         ResetTrig := R_Reset.Q,
235         Retro_S1 := Retro_S1_Present,
236         Diffuse_S1 := GVL.Diffuse_S1,
237         Diffuse_S2 := GVL.Diffuse_S2,
238         Diffuse_S3 := GVL.Diffuse_S3,
239         Diffuse_S4 := GVL.Diffuse_S4,
240         Diffuse_S5 := GVL.Diffuse_S5,
241         Retro_S2 := Retro_S2_Present,
242         Retro_S3 := Retro_S3_Present,
243         Retro_S4 := Retro_S4_Present,
244         InFlight_S := GVL.InFlight_ExitPath_S,
245         InFlight_M := GVL.InFlight_ExitPath_M,
246         InFlight_L := GVL.InFlight_ExitPath_L,
247         WeightAnalog := GVL.Weight_Analog);
248
249   (* 50.20. Start delay - Start_Button OR Start_Cmd_HMI. *)
250   StartDelay(StartButton := GVL.Start_Button OR GVL_HMI.Start_Cmd_HMI,
251             InStopped := (ModeThisScan = E_Mode.MODE_STOPPED));
252
253   (* 50.30. Two-axis Pick & Place - see FB_PickPlace header. *)
254   PickPlace(Enable := ProcessEnable,
255            StopRequested := Safety.StopRequested,
256            ResetTrig := R_Reset.Q,
257            PosPick := GVL.Diffs_PosPick1,
258            PosDrop := GVL.Diffs_PosDrop1,
259            Detected := GVL.PnP1_Detected,
260            Xact := GVL.PnP1_Xact,
261            Zact := GVL.PnP1_Zact);
262
263   (* line drained: sorting cycle and PnP1 both idle *)
264   LineDrained := Cycle.Drained AND PickPlace.Drained;
265
266   (* Box side settled: line drained and PnP1 path empty, continuously for
267      T_BOX_SIDE_SETTLE. Cycle runs before PathWD_S/M/L, so in the scan a

```

```

268     box reaches Diffuse_S3/4/5 Cycle is already back in RUN while
269     InFlight still holds the previous 0 - LineDrained can be TRUE for one
270     scan, and MODE_DRAINED is sticky. The timer filters that out. *)
271 BoxSideSettleTmr(IN := (LineDrained AND PnP1PathClear), PT := T_BOX_SIDE_SETTLE);
272
273 (* 50.35. Pallet infeed - uses this scan's PickPlace.Drained (PickPlace
274     runs first). *)
275 PalletFeed(Enable      := ProcessEnable,
276            StopRequested := Safety.StopRequested,
277            Draining      := (ModeThisScan = E_Mode.MODE_DRAINING),
278            DiffS_PallEntry := GVL.DiffS_PallEntry,
279            DiffS_PosPick  := GVL.DiffS_PosPick1,
280            PnP1_Drained   := PickPlace.Drained,
281            LArray_LoadR   := GVL.LArray_LoadR);
282
283 (* 50.40. Path watchdogs - see FB_PathWatchdog. *)
284 PathWD_PnP1(Enable      := ProcessEnable,
285             StartSignal := GVL.DiffS_PosDrop1,
286             EndSignal   := Retro_S1_Present,
287             Timeout      := T_DROP_TO_ENTRY,
288             ResetTrig    := R_Reset.Q);
289
290 PathWD_S(Enable      := ProcessEnable,
291          StartSignal := GVL.Diffuse_S3,
292          EndSignal   := Retro_S2_Present,
293          Timeout      := T_EXIT_S,
294          ResetTrig    := R_Reset.Q);
295
296 PathWD_M(Enable      := ProcessEnable,
297          StartSignal := GVL.Diffuse_S4,
298          EndSignal   := Retro_S3_Present,
299          Timeout      := T_EXIT_M,
300          ResetTrig    := R_Reset.Q);
301
302 PathWD_L(Enable      := ProcessEnable,
303          StartSignal := GVL.Diffuse_S5,
304          EndSignal   := Retro_S4_Present,
305          Timeout      := T_EXIT_L,
306          ResetTrig    := R_Reset.Q);
307
308 (* whole pallet feed; LArray_* read TRUE = pallet present, no inversion *)
309 PathWD_Pallet(Enable      := ProcessEnable,
310              StartSignal := GVL.LArray_LoadR,
311              EndSignal   := GVL.LArray_LoadL,
312              Timeout      := T_PALLET_PATH,
313              ResetTrig    := R_Reset.Q);
314
315 (* =====
316 60. MODE FOR THE NEXT SCAN                                     (FSD ch. 3.1)
317 ===== *)
318 Mode := F_NextMode(CurrentMode      := ModeThisScan,
319                   StartConfirmed := StartDelay.StartConfirmed,
320                   SelModeManual   := SelModeManualEff AND NOT HardStopLatch, (* latch masks Manual *)
321                   ResetTrig      := R_Reset.Q,
322                   StopRequested  := Safety.StopRequested,
323                   StopActive     := Safety.StopActive,
324                   EmergencyActive := Safety.EmergencyActive,
325                   JamFault       := GVL.LineJamFault,
326                   LineDrained    := LineDrained,
327                   DrainTrig      := R_Drain.Q,
328                   PalletSideSettled := LastPalletSettleTmr.Q,
329                   BoxSideSettled  := BoxSideSettleTmr.Q);
330
331 (* =====
332 70. OUTPUTS PER MODE                                           (FSD ch. 3.4)
333 Dispatched on OutMode = ModeThisScan, except a HARD STOP in
334 MODE_STOPPED, which uses the MODE_EMERGENCY branch. PnP1_GrabCmd is
335 then overridden after the CASE.
336 ===== *)
337 OutMode := ModeThisScan;
338 IF (ModeThisScan = E_Mode.MODE_STOPPED) AND HardStopOut THEN
339     OutMode := E_Mode.MODE_EMERGENCY;
340 END_IF
341
342 CASE OutMode OF

```

```

343
344 E_Mode.MODE_EMERGENCY:
345     GVL.Conv1      := FALSE;
346     GVL.Conv2      := 0.0;
347     GVL.Conv3      := 0.0;
348     GVL.Conv4      := 0.0;
349     GVL.Scale_Conveyor := FALSE;
350     GVL.Sorter_Left  := FALSE;
351     GVL.Sorter_Forward := FALSE;
352     GVL.Sorter_Right := FALSE;
353     (* box line roller stops released *)
354     GVL.RollStop_Entry := FALSE;
355     GVL.RollStop_S     := FALSE;
356     GVL.RollStop_M     := FALSE;
357     GVL.RollStop_L     := FALSE;
358     GVL.LoadConv1      := 0.0;
359     GVL.LoadConv2      := 0.0;
360     GVL.LoadConv3      := 0.0;
361     (* PnP1 holds position and grip (PickPlace re-commands the actual position
362        while disabled) *)
363     GVL.PnP1_Xset      := PickPlace.Xset;
364     GVL.PnP1_Zset      := PickPlace.Zset;
365     GVL.PnP1_GrabCmd    := PickPlace.GrabCmd;
366     (* pallet line stopped, roller stops released *)
367     GVL.PallConvLoad_R := 0.0;
368     GVL.PallConvCurv_R1 := 0.0;
369     GVL.PallConvCurv_R2 := 0.0;
370     GVL.PallConvCurv_R3 := 0.0;
371     GVL.PallConvStrait_1 := 0.0;
372     GVL.PallConvStrait_2 := 0.0;
373     GVL.PallConvStrait_3 := 0.0;
374     GVL.PallConvCurv_L1 := 0.0;
375     GVL.PallConvCurv_L2 := 0.0;
376     GVL.PallConvCurv_L3 := 0.0;
377     GVL.PallConvLoad_L := 0.0;
378     GVL.RollStop_PallPick1 := FALSE;
379     GVL.RollStop_PallPick2 := FALSE;
380     GVL.RollStop_LoadR     := FALSE;
381     GVL.RollStop_LoadL     := FALSE;
382     GVL.RollStop_Entry1    := FALSE;
383
384 E_Mode.MODE_STOPPED:
385     GVL.Conv1      := FALSE;
386     GVL.Conv2      := 0.0;
387     GVL.Conv3      := 0.0;
388     GVL.Conv4      := 0.0;
389     GVL.Scale_Conveyor := FALSE;
390     GVL.Sorter_Left  := FALSE;
391     GVL.Sorter_Forward := FALSE;
392     GVL.Sorter_Right := FALSE;
393     (* box line roller stops engaged at rest *)
394     GVL.RollStop_Entry := TRUE;
395     GVL.RollStop_S     := TRUE;
396     GVL.RollStop_M     := TRUE;
397     GVL.RollStop_L     := TRUE;
398     GVL.LoadConv1      := 0.0;
399     GVL.LoadConv2      := 0.0;
400     GVL.LoadConv3      := 0.0;
401     (* PnP1 holds position and grip *)
402     GVL.PnP1_Xset      := PickPlace.Xset;
403     GVL.PnP1_Zset      := PickPlace.Zset;
404     GVL.PnP1_GrabCmd    := PickPlace.GrabCmd;
405     (* pallet line stopped, roller stops released *)
406     GVL.PallConvLoad_R := 0.0;
407     GVL.PallConvCurv_R1 := 0.0;
408     GVL.PallConvCurv_R2 := 0.0;
409     GVL.PallConvCurv_R3 := 0.0;
410     GVL.PallConvStrait_1 := 0.0;
411     GVL.PallConvStrait_2 := 0.0;
412     GVL.PallConvStrait_3 := 0.0;
413     GVL.PallConvCurv_L1 := 0.0;
414     GVL.PallConvCurv_L2 := 0.0;
415     GVL.PallConvCurv_L3 := 0.0;
416     GVL.PallConvLoad_L := 0.0;
417     GVL.RollStop_PallPick1 := FALSE;

```

```

418     GVL.RollStop_PallPick2 := FALSE;
419     GVL.RollStop_LoadR    := FALSE;
420     GVL.RollStop_LoadL    := FALSE;
421     GVL.RollStop_Entry1   := FALSE;
422
423     (* MODE_DRAINED = MODE_STOPPED outputs, except that the pallet line roller
424        stops are engaged, so the emptied line stays secured. Same steady yellow
425        light; Dis.SystemText shows DRAINED. *)
426     E_Mode.MODE_DRAINED:
427         GVL.Conv1          := FALSE;
428         GVL.Conv2          := 0.0;
429         GVL.Conv3          := 0.0;
430         GVL.Conv4          := 0.0;
431         GVL.Scale_Conveyor := FALSE;
432         GVL.Sorter_Left    := FALSE;
433         GVL.Sorter_Forward := FALSE;
434         GVL.Sorter_Right   := FALSE;
435         GVL.RollStop_Entry := TRUE;
436         GVL.RollStop_S     := TRUE;
437         GVL.RollStop_M     := TRUE;
438         GVL.RollStop_L     := TRUE;
439         GVL.LoadConv1      := 0.0;
440         GVL.LoadConv2      := 0.0;
441         GVL.LoadConv3      := 0.0;
442         GVL.PnP1_Xset      := PickPlace.Xset;
443         GVL.PnP1_Zset      := PickPlace.Zset;
444         GVL.PnP1_GrabCmd   := PickPlace.GrabCmd;
445         GVL.PallConvLoad_R := 0.0;
446         GVL.PallConvCurv_R1 := 0.0;
447         GVL.PallConvCurv_R2 := 0.0;
448         GVL.PallConvCurv_R3 := 0.0;
449         GVL.PallConvStrait_1 := 0.0;
450         GVL.PallConvStrait_2 := 0.0;
451         GVL.PallConvStrait_3 := 0.0;
452         GVL.PallConvCurv_L1 := 0.0;
453         GVL.PallConvCurv_L2 := 0.0;
454         GVL.PallConvCurv_L3 := 0.0;
455         GVL.PallConvLoad_L  := 0.0;
456         (* the one difference from MODE_STOPPED: engaged *)
457         GVL.RollStop_PallPick1 := TRUE;
458         GVL.RollStop_PallPick2 := TRUE;
459         GVL.RollStop_LoadR     := TRUE;
460         GVL.RollStop_LoadL     := TRUE;
461         GVL.RollStop_Entry1    := TRUE;
462
463     (* AUTO and DRAINING: outputs straight from the process blocks *)
464     E_Mode.MODE_AUTO, E_Mode.MODE_DRAINING:
465         GVL.Conv1          := Cycle.Conv1;
466         GVL.Conv2          := Cycle.Conv2;
467         GVL.Conv3          := Cycle.Conv3;
468         GVL.Conv4          := Cycle.Conv4;
469         GVL.Scale_Conveyor := Cycle.ScaleConveyor;
470         GVL.Sorter_Left    := Cycle.SorterLeft;
471         GVL.Sorter_Forward := Cycle.SorterForward;
472         GVL.Sorter_Right   := Cycle.SorterRight;
473         GVL.RollStop_Entry := Cycle.RollStop_Entry;
474         GVL.RollStop_S     := Cycle.RollStop_S;
475         GVL.RollStop_M     := Cycle.RollStop_M;
476         GVL.RollStop_L     := Cycle.RollStop_L;
477         GVL.LoadConv1      := Cycle.LoadConv1;
478         GVL.LoadConv2      := Cycle.LoadConv2;
479         GVL.LoadConv3      := Cycle.LoadConv3;
480         (* PnP1 *)
481         GVL.PnP1_Xset      := PickPlace.Xset;
482         GVL.PnP1_Zset      := PickPlace.Zset;
483         GVL.PnP1_GrabCmd   := PickPlace.GrabCmd;
484
485         (* pallet feed *)
486         GVL.PallConvLoad_R := PalletFeed.PallConvLoad_R;
487         GVL.PallConvCurv_R1 := PalletFeed.PallConvCurv_R1;
488         GVL.PallConvCurv_R2 := PalletFeed.PallConvCurv_R2;
489         GVL.PallConvCurv_R3 := PalletFeed.PallConvCurv_R3;
490         GVL.PallConvStrait_1 := PalletFeed.PallConvStrait_1;
491         GVL.PallConvStrait_2 := PalletFeed.PallConvStrait_2;
492         GVL.PallConvStrait_3 := PalletFeed.PallConvStrait_3;

```

```

493   GVL.PallConvCurv_L1 := PalletFeed.PallConvCurv_L1;
494   GVL.PallConvCurv_L2 := PalletFeed.PallConvCurv_L2;
495   GVL.PallConvCurv_L3 := PalletFeed.PallConvCurv_L3;
496   GVL.PallConvLoad_L   := PalletFeed.PallConvLoad_L;
497   GVL.RollStop_PallPick1 := PalletFeed.RollStop_PallPick1;
498   GVL.RollStop_PallPick2 := PalletFeed.RollStop_PallPick2;
499   GVL.RollStop_LoadR    := PalletFeed.RollStop_LoadR;
500   GVL.RollStop_LoadL    := PalletFeed.RollStop_LoadL;
501   GVL.RollStop_Entry1   := PalletFeed.RollStop_Entry1;
502
503   (* The live analog outputs are copied into the HMI trims every scan. A HARD
504      STOP clears them (step 40). *)
505   GVL_HMI.LoadConv1_Cmd_HMI := GVL.LoadConv1;
506   GVL_HMI.LoadConv2_Cmd_HMI := GVL.LoadConv2;
507   GVL_HMI.LoadConv3_Cmd_HMI := GVL.LoadConv3;
508   GVL_HMI.Conv2_Cmd_HMI     := GVL.Conv2;
509   GVL_HMI.Conv3_Cmd_HMI     := GVL.Conv3;
510   GVL_HMI.Conv4_Cmd_HMI     := GVL.Conv4;
511   GVL_HMI.PnP1_Xset_Cmd_HMI := GVL.PnP1_Xset;
512   GVL_HMI.PnP1_Zset_Cmd_HMI := GVL.PnP1_Zset;
513   GVL_HMI.PallConvLoad_R_Cmd_HMI := GVL.PallConvLoad_R;
514   GVL_HMI.PallConvCurv_R1_Cmd_HMI := GVL.PallConvCurv_R1;
515   GVL_HMI.PallConvCurv_R2_Cmd_HMI := GVL.PallConvCurv_R2;
516   GVL_HMI.PallConvCurv_R3_Cmd_HMI := GVL.PallConvCurv_R3;
517   GVL_HMI.PallConvStrait_1_Cmd_HMI := GVL.PallConvStrait_1;
518   GVL_HMI.PallConvStrait_2_Cmd_HMI := GVL.PallConvStrait_2;
519   GVL_HMI.PallConvStrait_3_Cmd_HMI := GVL.PallConvStrait_3;
520   GVL_HMI.PallConvCurv_L1_Cmd_HMI := GVL.PallConvCurv_L1;
521   GVL_HMI.PallConvCurv_L2_Cmd_HMI := GVL.PallConvCurv_L2;
522   GVL_HMI.PallConvCurv_L3_Cmd_HMI := GVL.PallConvCurv_L3;
523   GVL_HMI.PallConvLoad_L_Cmd_HMI := GVL.PallConvLoad_L;
524
525   E_Mode.MODE_MANUAL:
526   (* MANUAL: every actuator from its physical input Man_* combined with the
527      HMI command - BOOL by OR, REAL by LIMIT(Man_* + Cmd_HMI) within the
528      channel range. Nothing commanded = drives stopped, roller stops
529      released. *)
530   GVL.Conv1      := GVL.Man_Conv1 OR GVL_HMI.Conv1_Cmd_HMI;
531   (* exit lanes limited to -10..+10 in MANUAL (drive range -10..+20) *)
532   GVL.Conv2      := LIMIT(-10.0, GVL.Man_Conv2 + GVL_HMI.Conv2_Cmd_HMI, 10.0);
533   GVL.Conv3      := LIMIT(-10.0, GVL.Man_Conv3 + GVL_HMI.Conv3_Cmd_HMI, 10.0);
534   GVL.Conv4      := LIMIT(-10.0, GVL.Man_Conv4 + GVL_HMI.Conv4_Cmd_HMI, 10.0);
535   GVL.Scale_Conveyor := GVL.Man_Scale OR GVL_HMI.Scale_Conveyor_Cmd_HMI;
536   GVL.Sorter_Left  := GVL.Man_Sorter_Left OR GVL_HMI.Sorter_Left_Cmd_HMI;
537   GVL.Sorter_Forward := GVL.Man_Sorter_Forward OR GVL_HMI.Sorter_Forward_Cmd_HMI;
538   GVL.Sorter_Right := GVL.Man_Sorter_Right OR GVL_HMI.Sorter_Right_Cmd_HMI;
539
540   (* RollStop_Entry from the HMI only - there is no Man_ input *)
541   GVL.RollStop_Entry := GVL_HMI.RollStop_Entry_Cmd_HMI;
542   GVL.RollStop_S     := GVL.Man_RollStop_S OR GVL_HMI.RollStop_S_Cmd_HMI;
543   GVL.RollStop_M     := GVL.Man_RollStop_M OR GVL_HMI.RollStop_M_Cmd_HMI;
544   GVL.RollStop_L     := GVL.Man_RollStop_L OR GVL_HMI.RollStop_L_Cmd_HMI;
545
546   GVL.LoadConv1 := LIMIT(-10.0, GVL.Man_LoadConv1 + GVL_HMI.LoadConv1_Cmd_HMI, 10.0);
547   GVL.LoadConv2 := LIMIT(-10.0, GVL.Man_LoadConv2 + GVL_HMI.LoadConv2_Cmd_HMI, 10.0);
548   GVL.LoadConv3 := LIMIT(-10.0, GVL.Man_LoadConv3 + GVL_HMI.LoadConv3_Cmd_HMI, 10.0);
549
550   (* PnP1: position setpoints, limited to 0..10 *)
551   GVL.PnP1_Xset := LIMIT(0.0, GVL.Man_PnP1_Xset + GVL_HMI.PnP1_Xset_Cmd_HMI, 10.0);
552   GVL.PnP1_Zset := LIMIT(0.0, GVL.Man_PnP1_Zset + GVL_HMI.PnP1_Zset_Cmd_HMI, 10.0);
553   GVL.PnP1_GrabCmd := GVL.Man_PnP1_Grab OR GVL_HMI.PnP1_GrabCmd_Cmd_HMI;
554
555   (* pallet line: drives -10..+10, roller stops by OR *)
556   GVL.PallConvLoad_R := LIMIT(-10.0, GVL.Man_PallConvLoad_R + GVL_HMI.PallConvLoad_R_Cmd_HMI, 10.0);
557   GVL.PallConvCurv_R1 := LIMIT(-10.0, GVL.Man_PallConvCurv_R1 + GVL_HMI.PallConvCurv_R1_Cmd_HMI, 10.0);
558   GVL.PallConvCurv_R2 := LIMIT(-10.0, GVL.Man_PallConvCurv_R2 + GVL_HMI.PallConvCurv_R2_Cmd_HMI, 10.0);
559   GVL.PallConvCurv_R3 := LIMIT(-10.0, GVL.Man_PallConvCurv_R3 + GVL_HMI.PallConvCurv_R3_Cmd_HMI, 10.0);
560   GVL.PallConvStrait_1 := LIMIT(-10.0, GVL.Man_PallConvStrait_1 + GVL_HMI.PallConvStrait_1_Cmd_HMI, 10.0);
561   GVL.PallConvStrait_2 := LIMIT(-10.0, GVL.Man_PallConvStrait_2 + GVL_HMI.PallConvStrait_2_Cmd_HMI, 10.0);
562   GVL.PallConvStrait_3 := LIMIT(-10.0, GVL.Man_PallConvStrait_3 + GVL_HMI.PallConvStrait_3_Cmd_HMI, 10.0);
563   GVL.PallConvCurv_L1 := LIMIT(-10.0, GVL.Man_PallConvCurv_L1 + GVL_HMI.PallConvCurv_L1_Cmd_HMI, 10.0);
564   GVL.PallConvCurv_L2 := LIMIT(-10.0, GVL.Man_PallConvCurv_L2 + GVL_HMI.PallConvCurv_L2_Cmd_HMI, 10.0);
565   GVL.PallConvCurv_L3 := LIMIT(-10.0, GVL.Man_PallConvCurv_L3 + GVL_HMI.PallConvCurv_L3_Cmd_HMI, 10.0);
566   GVL.PallConvLoad_L := LIMIT(-10.0, GVL.Man_PallConvLoad_L + GVL_HMI.PallConvLoad_L_Cmd_HMI, 10.0);
567   GVL.RollStop_PallPick1 := GVL.Man_RollStop_PallPick1 OR GVL_HMI.RollStop_PallPick1_Cmd_HMI;

```

```

568     GVL.RollStop_PallPick2 := GVL.Man_RollStop_PallPick2 OR GVL_HMI.RollStop_PallPick2_Cmd_HMI;
569     GVL.RollStop_LoadR    := GVL.Man_RollStop_LoadR    OR GVL_HMI.RollStop_LoadR_Cmd_HMI;
570     GVL.RollStop_LoadL    := GVL.Man_RollStop_LoadL    OR GVL_HMI.RollStop_LoadL_Cmd_HMI;
571     GVL.RollStop_Entry1   := GVL.Man_RollStop_Entry1   OR GVL_HMI.RollStop_Entry1_Cmd_HMI;
572
573 END_CASE
574
575 (* HARD STOP: the grip stays as it was at the moment of the stop - a box
576    in the gripper is never dropped. *)
577 IF (ModeThisScan = E_Mode.MODE_STOPPED) AND HardStopOut THEN
578     GVL.PnP1_GrabCmd := HardStopGrabHold;
579 END_IF
580
581 (* =====
582    80. STACK LIGHTS AND SIRENS (FSD ch. 4)
583    All eight towers show the same pattern; both sirens play the same
584    signal. Each event sounds once, on its rising edge.
585    ===== *)
586 Blink(Enable := TRUE, TimeOn := T#400MS, TimeOff := T#400MS);
587
588 F_TowerColors(Mode           := ModeThisScan,
589               StopRequested  := Safety.StopRequested,
590               StartPending   := StartDelay.Armed,
591               JamFault       := GVL.LineJamFault,
592               BlinkPulse     := Blink.Pulse,
593               Red             => TowerRed,
594               Yellow          => TowerYellow,
595               Green           => TowerGreen);
596
597 FOR j := 1 TO GVL.NUM_TOWERS DO
598     GVL.Tower_Red_Out[j] := TowerRed;
599     GVL.Tower_Yellow_Out[j] := TowerYellow;
600     GVL.Tower_Green_Out[j] := TowerGreen;
601 END_FOR;
602
603 (* 9 beeps on E-stop / door only - a jam has its own JamBeep *)
604 EmergencyAlarm(Trigger := Safety.EmergencyActive,
605               PulseCount := 9,
606               TimeOn := T#300MS,
607               TimeOff := T#300MS);
608
609 (* continuous 1 s tone on entering AUTO *)
610 StartupAlarm(Trigger := (ModeThisScan = E_Mode.MODE_AUTO), Duration := T#1S);
611
612 (* 1 beep when the start countdown begins *)
613 StartBeep(Trigger := StartDelay.Armed,
614           PulseCount := 1,
615           TimeOn := T#150MS,
616           TimeOff := T#150MS);
617
618 (* 3 beeps when Stop is latched *)
619 StopBeep(Trigger := Safety.StopRequested,
620          PulseCount := 3,
621          TimeOn := T#150MS,
622          TimeOff := T#150MS);
623
624 (* 4 beeps when a jam fault latches *)
625 JamBeep(Trigger := GVL.LineJamFault,
626         PulseCount := 4,
627         TimeOn := T#150MS,
628         TimeOff := T#150MS);
629
630 (* 2 beeps on entering MODE DRAINING *)
631 DrainBeep(Trigger := (ModeThisScan = E_Mode.MODE_DRAINING),
632          PulseCount := 2,
633          TimeOn := T#150MS,
634          TimeOff := T#150MS);
635
636 (* both sirens play the same signal *)
637 GVL.Siren_Out[1] := EmergencyAlarm.Alarm OR StartupAlarm.Alarm
638                 OR StartBeep.Alarm OR StopBeep.Alarm OR JamBeep.Alarm
639                 OR DrainBeep.Alarm;
640
641 GVL.Siren_Out[2] := GVL.Siren_Out[1];
642

```

```

643 (* =====
644 90. HMI STATUS TEXTS (FSD ch. 13.2)
645 Display only - nothing else reads these, so the wording can be edited
646 here. Dis_SystemText uses the same conditions as F_TowerColors, so
647 the text and the stack lights always agree.
648 ===== *)
649 CASE ModeThisScan OF
650   E_Mode.MODE_STOPPED:   GVL.Dis_ModeText := 'Stopped';
651   E_Mode.MODE_AUTO:     GVL.Dis_ModeText := 'Auto';
652   E_Mode.MODE_MANUAL:    GVL.Dis_ModeText := 'Manual';
653   E_Mode.MODE_EMERGENCY: GVL.Dis_ModeText := 'EMERGENCY';
654   E_Mode.MODE_DRAINING:  GVL.Dis_ModeText := 'Draining';
655   E_Mode.MODE_DRAINED:   GVL.Dis_ModeText := 'Drained';
656 END_CASE
657
658 CASE Cycle.State OF
659   E_SortState.IDLE:       GVL.Dis_SortStatusText := 'Idle';
660   E_SortState.RUN:        GVL.Dis_SortStatusText := 'Running';
661   E_SortState.WAIT_SCALE_ENTRY: GVL.Dis_SortStatusText := 'Waiting for box';
662   E_SortState.WEIGHING:    GVL.Dis_SortStatusText := 'Weighing';
663   E_SortState.CLASSIFICATION: GVL.Dis_SortStatusText := 'Classifying';
664   E_SortState.SORTING:     GVL.Dis_SortStatusText := 'Sorting';
665   E_SortState.WAIT_EXIT:   GVL.Dis_SortStatusText := 'Waiting for exit';
666   E_SortState.SORT_CONFIRM: GVL.Dis_SortStatusText := 'Confirming sort';
667   E_SortState.STOPPING:    GVL.Dis_SortStatusText := 'Stopping';
668   E_SortState.SORT_FAULT:  GVL.Dis_SortStatusText := 'FAULT';
669 END_CASE
670
671 (* PnP1 state *)
672 CASE PickPlace.State OF
673   E_PickPlaceState.PP_HOME:       GVL.Dis_PnP1StatusText := 'Home';
674   E_PickPlaceState.PP_WAIT:       GVL.Dis_PnP1StatusText := 'Waiting for pallet';
675   E_PickPlaceState.PP_LOWER_PICK: GVL.Dis_PnP1StatusText := 'Lowering to pick';
676   E_PickPlaceState.PP_GRAB:       GVL.Dis_PnP1StatusText := 'Grabbing';
677   E_PickPlaceState.PP_LIFT:       GVL.Dis_PnP1StatusText := 'Lifting';
678   E_PickPlaceState.PP_MOVE_TO_DROP: GVL.Dis_PnP1StatusText := 'Moving to drop';
679   E_PickPlaceState.PP_RELEASE:    GVL.Dis_PnP1StatusText := 'Releasing';
680   E_PickPlaceState.PP_FAULT:      GVL.Dis_PnP1StatusText := 'FAULT';
681 END_CASE
682
683 (* class of the box being sorted ('-' outside CLASSIFICATION .. SORT_CONFIRM) and
684 last settled weight *)
685 CASE Cycle.WeightClass OF
686   E_WeightClass.CLS_NONE: GVL.Dis_BoxClassText := '-';
687   E_WeightClass.CLS_S:    GVL.Dis_BoxClassText := 'S';
688   E_WeightClass.CLS_M:    GVL.Dis_BoxClassText := 'M';
689   E_WeightClass.CLS_L:    GVL.Dis_BoxClassText := 'L';
690 END_CASE
691 GVL.Dis_CurrentWeight := Cycle.CurrentWeight;
692
693 (* SYSTEM text - same conditions as the stack lights (step 80) *)
694 CASE ModeThisScan OF
695   E_Mode.MODE_EMERGENCY:
696     IF GVL.LineJamFault THEN
697       GVL.Dis_SystemText := 'JAM';
698     ELSE
699       GVL.Dis_SystemText := 'E-STOP';
700     END_IF
701
702   E_Mode.MODE_STOPPED:
703     IF StartDelay.Armed THEN
704       GVL.Dis_SystemText := 'STARTING';
705     ELIF HardStopLatch THEN
706       GVL.Dis_SystemText := 'HARD STOP';
707     ELSE
708       GVL.Dis_SystemText := 'STOPPED';
709     END_IF
710
711   E_Mode.MODE_MANUAL:
712     GVL.Dis_SystemText := 'MANUAL';
713
714   E_Mode.MODE_AUTO:
715     IF Safety.StopRequested THEN
716       GVL.Dis_SystemText := 'STOPPING';
717     ELSE

```

```

718     GVL.Dis_SystemText := 'RUNNING';
719     END_IF
720
721     E_Mode.MODE_DRAINING:
722         GVL.Dis_SystemText := 'DRAINING';
723
724     E_Mode.MODE_DRAINED:
725         GVL.Dis_SystemText := 'DRAINED';
726 END_CASE
727
728 (* =====
729 95. ALARM HISTORY (FSD ch. 13.3)
730 Last step of the scan - every alarm source is final. Observer only:
731 writes nothing that another step reads.
732 ===== *)
733 (* Timestamp: RTC in UTC, converted to the controller's local time (time
734 zone + DST as set in Windows for CODESYS Control Win). If the
735 conversion fails, fall back to UTC rather than logging no time. *)
736 (* The SysTimeRtc library declares pResult / pdwTimestampLocal as
737 VAR_IN OUT (by reference), so the variables themselves are passed -
738 NOT ADR(...). *)
739 RtcUtc := SysTimeRtcGet(pResult := RtcResult);
740 IF SysTimeRtcConvertUtcToLocal(UDINT_TO_DWORD(RtcUtc), RtcLocal) <> 0 THEN
741     RtcLocal := UDINT_TO_DWORD(RtcUtc);
742 END_IF
743 NowLocal := DWORD_TO_DT(RtcLocal);
744
745 (* Alarm sources - index MUST match GVL.ALARM_TEXT, entry for entry.
746 Individual bits, not the aggregate LineJamFault/EmergencyActive, so the
747 history shows WHICH E-stop/door/segment it was. *)
748 AlarmBits[1] := GVL.Emergency_Tripped[1]; (* Emergency_Button *)
749 AlarmBits[2] := GVL.Emergency_Tripped[2]; (* Emergency_Button_S *)
750 AlarmBits[3] := GVL.Emergency_Tripped[3]; (* Emergency_Button_M *)
751 AlarmBits[4] := GVL.Emergency_Tripped[4]; (* Emergency_Button_L *)
752 AlarmBits[5] := GVL.Emergency_Tripped[5]; (* EButt_PallLoad_L *)
753 AlarmBits[6] := GVL.Emergency_Tripped[6]; (* EButt_PallLoad_R *)
754 AlarmBits[7] := GVL.Door_Open[1]; (* SafetyDoorSorting1 *)
755 AlarmBits[8] := GVL.Door_Open[2]; (* SafetyDoorSorting2 *)
756 AlarmBits[9] := GVL.Door_Open[3]; (* SafetyDoorSorting3 *)
757 AlarmBits[10] := GVL.Door_Open[4]; (* SafetyDoorSorting4 *)
758 AlarmBits[11] := GVL.Door_Open[5]; (* SafetyDoorLoding1 *)
759 AlarmBits[12] := GVL.Fault_SortCycle;
760 AlarmBits[13] := GVL.Fault_PnP1_Path;
761 AlarmBits[14] := GVL.Fault_PnP1_Robot;
762 AlarmBits[15] := GVL.Fault_ExitPath_S;
763 AlarmBits[16] := GVL.Fault_ExitPath_M;
764 AlarmBits[17] := GVL.Fault_ExitPath_L;
765 AlarmBits[18] := GVL.Fault_PalletPath;
766 AlarmBits[19] := GVL.Fault_Draining;
767
768 R_AlarmClear(CLK := GVL_HMI.AlarmHistClear_Cmd_HMI);
769
770 AlarmHistory(AlarmBits := AlarmBits,
771             Now := NowLocal,
772             ClearCmd := R_AlarmClear.Q,
773             Hist := GVL.AlarmHist,
774             Head := GVL.AlarmHistHead,
775             Count := GVL.AlarmHistCount);
776
777 GVL.Dis_AlarmLine := AlarmHistory.Lines;
778 GVL.Dis_AlarmLineActive := AlarmHistory.LineActive;
779 GVL.Dis_AlarmActiveCount := AlarmHistory.ActiveCount;
780 GVL.Dis_AlarmHistCount := GVL.AlarmHistCount;

```

Zdrojový kód – Automatické třídění na dopravníku

Verze zdroje v3.25.1 · Build 10 / 2026 · Vydání 1.0 · 10 / 2026

Reference: FSD Conveyor Sorting v3.25.1, vydání 1.0 · historie změn: CHANGELOG.md

David Neumann · davidneumann.info