



ПЛК И АВТОМАТИЗАЦИЯ

**Язык ST в промышленной автоматизации: от
основ до роботизированной ячейки**

Практическое руководство

Telegram: <https://t.me/plcmasters>

Любое копирование, перепечатка или передача данного контента без явного письменного разрешения правообладателя запрещена законом и может привести к юридической ответственности.

Введение

Данное практическое руководство было разработано как учебно-методическое пособие для учащихся учреждений среднего специального образования по специальности **5-04-0713-08 «Техническая эксплуатация технологического оборудования и средств робототехники в автоматизированном производстве»** в рамках дисциплины **«Программирование для автоматизированного оборудования»**.

Руководство создавалось с конкретной учебной целью: дать учащимся колледжа практические навыки программирования промышленных контроллеров на языке **Structured Text (ST)** стандарта МЭК 61131-3 — с нуля, без предварительного опыта программирования ПЛК, используя только бесплатное программное обеспечение **CODESYS 3.5** и обычный учебный компьютер.

В процессе подготовки материал неоднократно дорабатывался, обрстал подробными пояснениями, схемами, таблицами тестирования и разборами типичных ошибок. В результате руководство приобрело форму, полезную значительно более широкому кругу читателей, чем студенты одной группы.

Поэтому данное руководство публикуется в открытом виде — для всех, кто хочет освоить программирование ПЛК на языке ST.

Оно будет полезно:

- **Студентам технических специальностей** — колледжей и университетов по направлениям автоматизации, робототехники, мехатроники, электротехники
- **Начинающим инженерам и техникам** — кто впервые сталкивается с программированием ПЛК на производстве
- **Специалистам смежных профессий** — электрикам, наладчикам, операторам ЧПУ, которые хотят разобраться в логике работы автоматических линий
- **Самоучкам и энтузиастам** — всем, кто интересуется промышленной автоматизацией и робототехникой

Как устроено это руководство

Материал построен по принципу «от простого к сложному»: каждая следующая работа опирается на знания, полученные в предыдущей, и добавляет новый слой сложности.

Работа	Тема	Новые концепции
№1	Управление конвейером	Переменные, IF/ELSIF, CTU
№2	Управление захватом	Таймер TON, диагностика
№3	Машина состояний	ENUM, CASE/OF
№4	Функциональный блок	FB, VAR_INPUT/OUTPUT, RETURN
№5	Массивы и циклы	ARRAY, FOR, алгоритмы поиска
№6	Полный цикл ячейки	Интеграция всего изученного
Практ.	Отладка программы	Поиск и исправление ошибок

Что потребуется для работы

- Компьютер с операционной системой Windows 7/10/11
- **CODESYS 3.5** — бесплатно на сайте store.codesys.com
- Никакого дополнительного оборудования — все программы запускаются на встроенном **виртуальном ПЛК** прямо на вашем компьютере

Важное замечание

Программы, приведённые в данном руководстве, предназначены для **учебных целей и виртуального моделирования**. Перед применением любого кода на реальном производственном оборудовании необходимо провести полное тестирование, адаптацию под конкретное оборудование и соблюдение всех требований

промышленной безопасности. Авторы не несут ответственности за последствия применения учебных примеров на реальных установках без надлежащей проверки специалистом.

Желаем успехов в изучении — промышленная автоматизация открывает перед вами мир, где программный код управляет реальными машинами.

ПОДГОТОВКА К РАБОТЕ

Установка и настройка CODESYS

1. Скачайте CODESYS 3.5 SP19 с официального сайта **store.codesys.com** → раздел «Downloads» → «CODESYS Development System» → бесплатная версия
2. Установите с параметрами по умолчанию
3. При первом запуске: **Tools** → **Options** → **Text editor** → установите шрифт Consolas 12pt для удобного чтения кода

Создание нового проекта (выполняется перед каждой работой)

1. **File** → **New Project**
2. Выберите **Standard project** → OK
3. В диалоге выбора устройства:
 - Device: **CODESYS Control Win V3** (виртуальный ПЛК — работает прямо на компьютере)
 - PLC_PRG language: **Structured Text (ST)**
 - → OK
4. В дереве проекта слева появятся:
 - Device — конфигурация ПЛК
 - Application — приложение
 - PLC_PRG (ST) — главная программа

Запуск виртуального ПЛК

1. **Build** → **Build** (F11) — компиляция, убедитесь: «0 errors, 0 warnings»
2. **Online** → **Login** (Alt+F8) — подключение к виртуальному ПЛК
3. При вопросе «Download?» → **Yes**
4. **Debug** → **Start** (F5) — запуск программы
5. Для остановки: **Debug** → **Stop**, для отключения: **Online** → **Logout**

Как форсировать (вручную задавать) значения переменных

Во время работы программы (Online режим):

- Двойной клик на значение переменной в редакторе → вводите новое значение → **Ctrl+F7** (записать форсированное значение)
 - Или откройте **View → Watch** — добавьте переменные и меняйте значения вручную
 - Это имитирует нажатие кнопок и срабатывание датчиков
-

РАБОТА №1 — Управление конвейером (2 ч)

Цель

Написать программу автоматического управления конвейерной линией с кнопками Пуск/Стоп, аварийной остановкой E-Stop и счётчиком деталей.

Что изучается

- Объявление переменных типа BOOL и INT
- Условные операторы IF/ELSIF/ELSE
- Стандартный функциональный блок CTU (счётчик)
- Приоритет условий в цепочке IF

Схема работы системы

[btnStart] → motorRun = TRUE (конвейер работает)
[btnStop] → motorRun = FALSE (конвейер остановлен)
[btnEStop] → motorRun = FALSE + alarm = TRUE + lampAlarm = TRUE
[sensorPart] → CTU считает каждую деталь (передний фронт)

Пошаговое выполнение

Шаг 1 — Создать проект

Создайте новый Standard project с именем **Lab01_Conveyor** (см. «Подготовка»).

Шаг 2 — Открыть PLC_PRG

В дереве проекта двойной клик на **PLC_PRG (ST)** — откроется окно редактора с двумя зонами:

- **Верхняя зона** — объявление переменных (между VAR и END_VAR)
- **Нижняя зона** — тело программы

Шаг 3 — Ввести объявление переменных

В верхнюю зону введите:

VAR

```
btnStart      : BOOL;    (* кнопка Пуск *)
btnStop       : BOOL;    (* кнопка Стоп *)
btnEStop      : BOOL;    (* аварийная остановка *)
sensorPart    : BOOL;    (* датчик наличия детали *)
motorRun      : BOOL;    (* выход: двигатель конвейера *)
```

```

    lampAlarm    : BOOL;    (* выход: лампа аварии *)
    cntParts     : CTU;      (* счётчик деталей *)
    partCount    : INT;      (* текущее количество деталей *)
    alarm        : BOOL;      (* флаг аварии *)
END_VAR

```

Внимание: CTU — это функциональный блок (FB), не простая переменная. CODESYS создаёт для него отдельный экземпляр в памяти.

Шаг 4 — Ввести тело программы

В нижнюю зону введите:

```

(* =====
  УПРАВЛЕНИЕ ДВИГАТЕЛЕМ КОНВЕЙЕРА
  Приоритет: E-Stop > Авария > Пуск > Стоп
  ===== *)

IF btnEStop THEN
    (* E-Stop имеет наивысший приоритет – мгновенная остановка *)
    motorRun := FALSE;
    alarm    := TRUE;
    lampAlarm := TRUE;

ELSIF alarm AND NOT btnEStop THEN
    (* Авария зафиксирована – ждём сброса оператором.
       Пуск невозможен пока alarm = TRUE *)
    motorRun := FALSE;

ELSIF btnStart THEN
    (* Нормальный пуск: запускаем конвейер, сбрасываем аварию *)
    motorRun := TRUE;
    alarm    := FALSE;
    lampAlarm := FALSE;

ELSIF btnStop THEN
    (* Плановая остановка *)
    motorRun := FALSE;

END_IF;

(* =====
  СЧЁТЧИК ДЕТАЛЕЙ
  CTU считает каждый передний фронт sensorPart
  ===== *)
cntParts(CU := sensorPart, R := FALSE, PV := 32767);
partCount := cntParts.CV;

```

Шаг 5 — Скомпилировать

Build → Build (F11). В нижней панели «Messages» должно быть:

Build complete -- 0 error(s), 0 warning(s)

Если есть ошибки — двойной клик на сообщение об ошибке переведёт курсор на проблемную строку.

Шаг 6 — Запустить и протестировать

1. **Online** → **Login** → **Yes** → **F5**
2. Откройте **View** → **Watch Window** (Ctrl+W)
3. Добавьте все переменные: btnStart, btnStop, btnEStop, motorRun, alarm, lampAlarm, partCount

Тест 1 — Пуск конвейера:

- Дважды кликните btnStart → установите TRUE → Ctrl+F7
- Убедитесь: motorRun = TRUE, alarm = FALSE, lampAlarm = FALSE
- Верните btnStart = FALSE

Тест 2 — Остановка:

- Установите btnStop = TRUE
- Убедитесь: motorRun = FALSE
- Верните btnStop = FALSE

Тест 3 — Аварийная остановка:

- Сначала запустите: btnStart = TRUE, затем FALSE
- Затем: btnEStop = TRUE
- Убедитесь: motorRun = FALSE, alarm = TRUE, lampAlarm = TRUE
- Попробуйте нажать btnStart = TRUE — конвейер **не должен** запуститься (блокировка аварией)
- Верните btnEStop = FALSE — авария всё ещё активна (alarm = TRUE)
- Теперь нажмите btnStart = TRUE — конвейер запускается, alarm = FALSE

Тест 4 — Счётчик деталей:

- Запустите конвейер
- Несколько раз: sensorPart = TRUE, затем FALSE (имитация прохождения детали)
- Наблюдайте изменение partCount (увеличивается на 1 при каждом переднем фронте)

Шаг 7 — Добавить сброс счётчика (самостоятельная доработка)

Добавьте переменную btnResetCount : BOOL и строку:

```
cntParts(CU := sensorPart, R := btnResetCount, PV := 32767);
```

Проверьте: при btnResetCount = TRUE счётчик обнуляется.

Вопросы для самопроверки

1. Почему условие btnEStop стоит первым в цепочке IF?
2. Что произойдёт, если поменять местами ELSIF alarm... и ELSIF btnStart?
3. Почему STU считает по переднему фронту, а не по уровню сигнала?

4. Что означает параметр PV := 32767 у счётчика?

Содержание отчёта

- Листинг программы с комментариями
 - Скриншоты Watch Window для каждого теста
 - Ответы на вопросы для самопроверки
 - Блок-схема алгоритма управления (нарисовать вручную или в draw.io)
-

РАБОТА №2 — Управление захватом с таймерами (2 ч)

Цель

Написать программу управления пневматическим захватом с контролем времени выполнения операций и диагностикой неисправностей.

Что изучается

- Функциональный блок TON (таймер с задержкой включения)
- Параметры таймера: IN, PT, Q, ET
- Диагностика по таймауту
- Взаимная блокировка клапанов

Схема работы захвата

```
cmdGrip → valveGrip = TRUE → (ждём sensGripped)
├ sensGripped пришёл вовремя → gripOK = TRUE, valveGrip = FALSE
└ за 2 с не пришёл → errGrip = TRUE (неисправность!)
```

```
cmdRelease → valveRelease = TRUE → (ждём sensReleased)
├ sensReleased пришёл вовремя → releaseOK = TRUE
└ за 2 с не пришёл → errRelease = TRUE
```

Пошаговое выполнение

Шаг 1 — Создать проект

Создайте новый проект **Lab02_Gripper**.

Шаг 2 — Ввести переменные

VAR

cmdGrip	: BOOL;	(* команда захватить *)
cmdRelease	: BOOL;	(* команда разжать *)
sensGripped	: BOOL;	(* датчик: деталь захвачена *)
sensReleased	: BOOL;	(* датчик: захват открыт *)
valveGrip	: BOOL;	(* клапан зажима *)
valveRelease	: BOOL;	(* клапан разжима *)
gripOK	: BOOL;	(* захват выполнен успешно *)
releaseOK	: BOOL;	(* разжим выполнен успешно *)
tGrip	: TON;	(* таймер контроля захвата *)
tRelease	: TON;	(* таймер контроля разжима *)

```

errGrip      : BOOL;    (* ошибка: нет подтверждения захвата *)
errRelease   : BOOL;    (* ошибка: нет подтверждения разжима *)
END_VAR

```

Шаг 3 — Ввести тело программы

```

(* =====
УПРАВЛЕНИЕ ЗАЖИМОМ
===== *)

(* Открываем клапан зажима, если команда есть и захват ещё не подтверждён *)
IF cmdGrip AND NOT gripOK THEN
    valveGrip      := TRUE;
    valveRelease   := FALSE; (* взаимная блокировка – оба клапана одновременно не
открываем *)
END_IF;

(* Подтверждение захвата от датчика *)
IF sensGripped THEN
    gripOK         := TRUE;
    valveGrip      := FALSE; (* закрываем клапан – давление держит пружина *)
END_IF;

(* Таймер контроля: если за 2 с не пришло подтверждение – неисправность *)
tGrip(IN := cmdGrip AND NOT gripOK, PT := T#2S);
IF tGrip.Q THEN
    errGrip        := TRUE;
    valveGrip      := FALSE; (* снимаем команду – безопасное состояние *)
END_IF;

(* =====
УПРАВЛЕНИЕ РАЗЖИМОМ
===== *)

IF cmdRelease AND NOT releaseOK THEN
    valveRelease   := TRUE;
    valveGrip      := FALSE;
    gripOK         := FALSE; (* сбрасываем флаг успешного захвата *)
END_IF;

IF sensReleased THEN
    releaseOK      := TRUE;
    valveRelease   := FALSE;
END_IF;

tRelease(IN := cmdRelease AND NOT releaseOK, PT := T#2S);
IF tRelease.Q THEN
    errRelease     := TRUE;
    valveRelease   := FALSE;
END_IF;

```

Шаг 4 — Понять работу таймера TON

Параметры вызова `tGrip(IN := ..., PT := T#2S)`:

Параметр	Тип	Описание
IN	BOOL (вход)	При TRUE таймер начинает отсчёт
PT	TIME (вход)	Уставка времени (T#2S = 2 секунды)
Q	BOOL (выход)	Становится TRUE когда ET >= PT
ET	TIME (выход)	Текущее прошедшее время

Ключевое правило: Таймер сбрасывается когда IN становится FALSE. Поэтому как только `sensGripped = TRUE` и `gripOK = TRUE`, условие `IN := cmdGrip AND NOT gripOK` становится FALSE → таймер сбрасывается автоматически.

Шаг 5 — Добавить переменные для мониторинга таймера

Добавьте в Watch Window: `tGrip.ET` — будете видеть текущее прошедшее время таймера в реальном времени.

Шаг 6 — Протестировать

Тест 1 — Нормальный захват:

1. `cmdGrip = TRUE` → убедитесь: `valveGrip = TRUE`
2. Через ~0.5 с установите `sensGripped = TRUE`
3. Убедитесь: `gripOK = TRUE`, `valveGrip = FALSE`, `errGrip = FALSE`

Тест 2 — Таймаут захвата (неисправность):

1. Сбросьте: `gripOK = FALSE`, `errGrip = FALSE`, `cmdGrip = FALSE`
2. Установите `cmdGrip = TRUE`
3. НЕ устанавливайте `sensGripped` — ждите 2 секунды
4. Убедитесь: `errGrip = TRUE`, `valveGrip = FALSE`
5. Наблюдайте в Watch Window как `tGrip.ET` нарастает до T#2S

Тест 3 — Нормальный разжим:

1. `cmdRelease = TRUE` → `valveRelease = TRUE`
2. Установите `sensReleased = TRUE`
3. Убедитесь: `releaseOK = TRUE`, `valveRelease = FALSE`

Тест 4 — Взаимная блокировка:

- Установите одновременно `cmdGrip = TRUE` и `cmdRelease = TRUE`
- Убедитесь, что `valveGrip` и `valveRelease` не могут быть одновременно TRUE

Вопросы для самопроверки

1. Что произойдёт если убрать строку `gripOK := FALSE` в блоке разжима?
2. Почему таймер сбрасывается сам после получения подтверждения датчика?
3. Как изменить время контроля с 2 с на 1.5 с?

4. Что такое «взаимная блокировка» клапанов и зачем она нужна?

Содержание отчёта

- Листинг программы
 - Временная диаграмма: сигналы cmdGrip, valveGrip, sensGripped, gripOK, tGrip.ET (нарисовать от руки)
 - Скриншоты Watch Window для тестов 1 и 2
-

РАБОТА №3 — Управление режимами через CASE (2 ч)

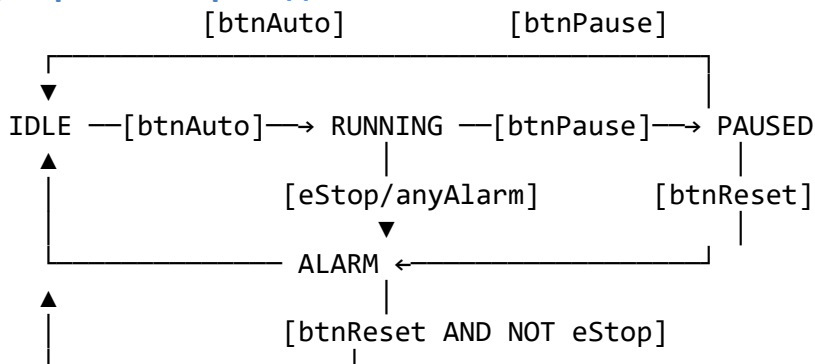
Цель

Реализовать машину состояний (State Machine) для управления производственной ячейкой с пятью режимами работы.

Что изучается

- Пользовательский тип данных ENUM
- Оператор CASE/OF
- Принцип машины состояний (State Machine)
- Управление переходами между состояниями

Диаграмма переходов состояний



IDLE —[btnSetup]→ SETUP —[btnReset]→ IDLE

Пошаговое выполнение

Шаг 1 — Создать проект

Создайте проект **Lab03_StateMachine**.

Шаг 2 — Создать тип данных ENUM

В дереве проекта правой кнопкой на **Application** → **Add Object** → **DUT (Data Unit Type)**:

- Name: E_CellMode
- Type: **Enumeration**

- → Open

В открывшемся редакторе введите:

```

TYPE E_CellMode :
(
  IDLE      := 0,    (* ожидание *)
  RUNNING   := 1,    (* автоматический цикл *)
  PAUSED    := 2,    (* пауза *)
  ALARM     := 3,    (* авария *)
  SETUP     := 4     (* наладка *)
);
END_TYPE

```

Зачем ENUM? Вместо магических чисел (0, 1, 2...) используем понятные имена. Код `mode := RUNNING` читается как текст, а не `mode := 1`. CODESYS не даст присвоить недопустимое значение.

Шаг 3 — Ввести переменные в PLC_PRG

```

VAR
  mode           : E_CellMode := IDLE; (* текущий режим *)
  btnAuto        : BOOL;    (* кнопка "Автомат" *)
  btnPause       : BOOL;    (* кнопка "Пауза" *)
  btnSetup       : BOOL;    (* кнопка "Наладка" *)
  btnReset       : BOOL;    (* кнопка "Сброс" *)
  eStop          : BOOL;    (* аварийная остановка *)
  anyAlarm       : BOOL;    (* любая авария в системе *)
  outConveyor    : BOOL;    (* выход: конвейер *)
  outRobotEnable : BOOL;    (* выход: разрешение роботу *)
  outAlarmLamp   : BOOL;    (* выход: лампа аварии *)
END_VAR

```

Шаг 4 — Ввести тело программы

```

(* =====
  ПЕРЕХОД В АВАРИЮ – ВЫСШИЙ ПРИОРИТЕТ
  Проверяется ДО оператора CASE
  ===== *)
IF eStop OR anyAlarm THEN
  mode := ALARM;
END_IF;

(* =====
  МАШИНА СОСТОЯНИЙ
  ===== *)
CASE mode OF

  IDLE:
    (* Все выходы выключены – безопасное состояние ожидания *)
    outConveyor   := FALSE;
    outRobotEnable := FALSE;
    outAlarmLamp  := FALSE;
    (* Переходы *)
    IF btnAuto THEN mode := RUNNING; END_IF;

```

```

    IF btnSetup THEN mode := SETUP;    END_IF;

RUNNING:
    (* Автоматический режим – всё работает *)
    outConveyor      := TRUE;
    outRobotEnable    := TRUE;
    outAlarmLamp      := FALSE;
    (* Переходы *)
    IF btnPause THEN mode := PAUSED; END_IF;
    (* E-Stop обработан выше – перейдёт в ALARM автоматически *)

PAUSED:
    (* Пауза – оборудование остановлено, готово к продолжению *)
    outConveyor      := FALSE;
    outRobotEnable    := FALSE;
    outAlarmLamp      := FALSE;
    (* Переходы *)
    IF btnAuto THEN mode := RUNNING; END_IF;    (* продолжить *)
    IF btnReset THEN mode := IDLE;    END_IF;    (* вернуться в начало *)

ALARM:
    (* Авария – всё остановлено, мигает лампа *)
    outConveyor      := FALSE;
    outRobotEnable    := FALSE;
    outAlarmLamp      := TRUE;
    (* Сброс только если E-Stop снят и нажат Reset *)
    IF btnReset AND NOT eStop THEN
        anyAlarm := FALSE;
        mode      := IDLE;
    END_IF;

SETUP:
    (* Наладка – оборудование остановлено, оператор работает вручную *)
    outConveyor      := FALSE;
    outRobotEnable    := FALSE;
    outAlarmLamp      := FALSE;
    IF btnReset THEN mode := IDLE; END_IF;

END_CASE;

```

Шаг 5 — Добавить визуализацию (опционально, 20 мин)

CODESYS имеет встроенный редактор визуализации:

1. Правой кнопкой на **Application** → **Add Object** → **Visualization**
2. Добавьте прямоугольники для каждого состояния
3. Добавьте кнопки (Toggle): привяжите каждую к соответствующей переменной
4. Добавьте текстовое поле с текстом %s → Expression: mode — будет показывать текущее состояние

Шаг 6 — Протестировать все переходы

Заполните таблицу тестирования:

№	Начальное состояние	Действие	Ожидаемый результат	Факт
1	IDLE	btnAuto = TRUE	mode = RUNNING	
2	RUNNING	btnPause = TRUE	mode = PAUSED	
3	PAUSED	btnAuto = TRUE	mode = RUNNING	
4	PAUSED	btnReset = TRUE	mode = IDLE	
5	RUNNING	eStop = TRUE	mode = ALARM	
6	ALARM	btnReset = TRUE, eStop = FALSE	mode = IDLE	
7	ALARM	btnReset = TRUE, eStop = TRUE	mode остаётся ALARM	
8	IDLE	btnSetup = TRUE	mode = SETUP	

Вопросы для самопроверки

1. Почему проверка eStop OR anyAlarm стоит **до** оператора CASE, а не внутри него?
 2. Что произойдёт если нажать btnAuto находясь в состоянии ALARM?
 3. Зачем в состоянии ALARM проверяется NOT eStop перед сбросом?
 4. Можно ли добавить новое состояние (например, MAINTENANCE) не меняя структуру программы?
-

РАБОТА №4 — Пользовательский функциональный блок FB_Motor (2 ч)

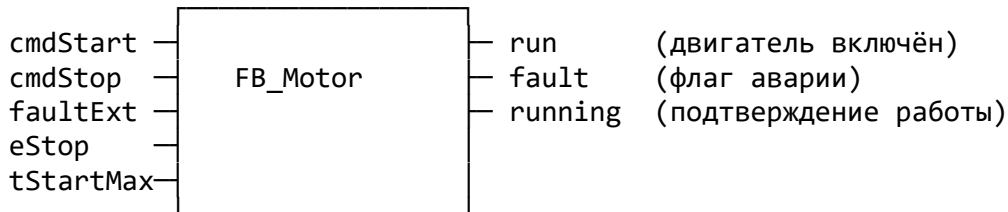
Цель

Создать универсальный функциональный блок управления двигателем с защитами, пригодный для многократного использования в любом проекте.

Что изучается

- Создание FUNCTION_BLOCK в CODESYS
- VAR_INPUT / VAR_OUTPUT / VAR
- Оператор RETURN
- Машина состояний внутри FB
- Инстанцирование FB (создание экземпляров)

Интерфейс функционального блока



Пошаговое выполнение

Шаг 1 — Создать проект и FB

1. Создайте проект **Lab04_FB_Motor**
2. Правой кнопкой на **Application** → **Add Object** → **POU**
3. Name: **FB_Motor**, Type: **Function Block**, Language: **ST** → Open

Шаг 2 — Ввести объявление переменных FB

FUNCTION_BLOCK FB_Motor

VAR_INPUT

```
cmdStart    : BOOL;    (* команда пуск *)
cmdStop     : BOOL;    (* команда стоп *)
faultExt    : BOOL;    (* внешняя авария (сигнал от частотника/термореле) *)
eStop       : BOOL;    (* аварийная остановка *)
tStartMax   : TIME;    (* максимально допустимое время пуска *)
```

END_VAR

VAR_OUTPUT

```
run         : BOOL;    (* выходной сигнал на пускатель/частотник *)
fault       : BOOL;    (* флаг аварии (требуется сброс) *)
running     : BOOL;    (* вход: подтверждение работы от частотника *)
```

END_VAR

VAR

```
tStart      : TON;    (* внутренний таймер контроля пуска *)
state       : INT := 0; (* внутреннее состояние: 0=стоп, 1=пуск, 2=работа *)
```

END_VAR

Важно понять разницу: - VAR_INPUT — то, что задаёт вызывающая программа -
VAR_OUTPUT — то, что FB возвращает вызывающей программе - VAR —
внутренние переменные, снаружи не видны

Шаг 3 — Ввести тело FB

```
(* =====
ЗАЩИТА: E-Stop и внешняя авария
Выполняется в первую очередь при любом состоянии
===== *)
IF eStop OR faultExt THEN
    run    := FALSE;
    fault  := TRUE;
    state  := 0;
    tStart(IN := FALSE, PT := tStartMax); (* сбросить таймер *)
```

```

    RETURN;  (* Выйти из FB – код ниже не выполняется *)
END_IF;

(* =====
   МАШИНА СОСТОЯНИЙ ДВИГАТЕЛЯ
   ===== *)
CASE state OF

    0: (* СТОП *)
        run := FALSE;
        (* Пуск разрешён только если нет аварии *)
        IF cmdStart AND NOT fault THEN
            state := 1;
        END_IF;

    1: (* ПУСК – ожидаем подтверждения работы *)
        run := TRUE;
        tStart(IN := TRUE, PT := tStartMax);

        IF running THEN
            (* Двигатель разогнался – переходим в нормальную работу *)
            tStart(IN := FALSE, PT := tStartMax);  (* сброс таймера *)
            state := 2;

        ELSIF tStart.Q THEN
            (* Таймаут пуска – двигатель не запустился *)
            run := FALSE;
            fault := TRUE;
            state := 0;
        END_IF;

    2: (* РАБОТА *)
        run := TRUE;
        tStart(IN := FALSE, PT := tStartMax);  (* таймер не нужен *)

        IF cmdStop THEN
            run := FALSE;
            state := 0;
        END_IF;

END_CASE;

```

Шаг 4 — Создать главную программу с тремя экземплярами FB

Откройте PLC_PRG (ST) и введите:

```

VAR
    (* Три экземпляра FB_Motor – для трёх разных двигателей *)
    mConveyor : FB_Motor;  (* двигатель конвейера *)
    mRobot    : FB_Motor;  (* двигатель оси робота *)
    mSpindle   : FB_Motor;  (* шпиндель станка *)

```

```

    (* Входные сигналы для каждого двигателя *)
    btnStartConv    : BOOL;
    btnStopConv     : BOOL;
    runningConv     : BOOL;  (* подтверждение от частотника конвейера *)
    faultConv       : BOOL;  (* авария частотника конвейера *)

    btnStartSpin    : BOOL;
    btnStopSpin     : BOOL;
    runningSpin     : BOOL;
    faultSpin       : BOOL;

    globaleEStop    : BOOL;  (* общая кнопка E-Stop *)
END_VAR

(* Вызов FB для двигателя конвейера *)
mConveyor(
    cmdStart := btnStartConv,
    cmdStop  := btnStopConv,
    faultExt := faultConv,
    eStop    := globaleEStop,
    tStartMax := T#5S,      (* конвейеру даём 5 с на разгон *)
    running  := runningConv
);

(* Вызов FB для шпинделя – даём больше времени на разгон *)
mSpindle(
    cmdStart := btnStartSpin,
    cmdStop  := btnStopSpin,
    faultExt := faultSpin,
    eStop    := globaleEStop,
    tStartMax := T#10S,     (* шпинделю даём 10 с на разгон *)
    running  := runningSpin
);

(* Чтение выходов – обращение через точку *)
(* mConveyor.run    – сигнал на пускатель конвейера *)
(* mConveyor.fault  – индикатор аварии конвейера *)
(* mSpindle.run     – сигнал на частотник шпинделя *)

```

Шаг 5 — Протестировать

Добавьте в Watch Window: mConveyor.run, mConveyor.fault, mConveyor.state, mSpindle.run, mSpindle.fault

Тест 1: Нормальный пуск конвейера:

1. btnStartConv = TRUE/FALSE
2. mConveyor.state = 1 (пуск)
3. Установите runningConv = TRUE (имитация подтверждения от частотника)
4. mConveyor.state = 2, mConveyor.run = TRUE

Тест 2: Таймаут пуска:

1. Сбросьте `mConveyor.fault = FALSE` вручную
2. `btnStartConv = TRUE/FALSE`
3. НЕ устанавливайте `runningConv` — ждите 5 секунд
4. `mConveyor.fault = TRUE, mConveyor.run = FALSE`

Тест 3: E-Stop при работе:

1. Запустите конвейер до состояния 2
2. Установите `globalEStop = TRUE`
3. Оба двигателя: `run = FALSE, fault = TRUE`

Вопросы для самопроверки

1. Чем отличается `VAR_INPUT` от `VAR` в функциональном блоке?
 2. Зачем используется `RETURN` в начале тела FB?
 3. Почему нужно явно сбрасывать таймер `tStart(IN := FALSE)` при переходе в состояние 2?
 4. Как добавить выход `stateOut : INT` чтобы главная программа знала текущее состояние FB?
-

РАБОТА №5 — Массивы и циклы: накопитель на 8 позиций (2 ч)

Цель

Освоить работу с массивами и циклом `FOR` для управления многопозиционным накопителем деталей.

Что изучается

- Объявление и использование массивов `ARRAY[ .. ] OF`
- Цикл `FOR` с индексной переменной
- Алгоритмы поиска (первый свободный/занятый элемент)
- Агрегирование данных (подсчёт суммы)

Схема накопителя

Позиции: [0] [1] [2] [3] [4] [5] [6] [7]

Датчики: D0 D1 D2 D3 D4 D5 D6 D7

Состояние: 0=свободно, 1=занято

Задача робота: положить деталь в `freeIndex`

Задача станка: взять деталь из `fullIndex`

Пошаговое выполнение

Шаг 1 — Создать проект Lab05_Buffer

Шаг 2 — Ввести переменные

VAR

```
(* Массив состояний ячеек – обновляется с датчиков *)
cells      : ARRAY[0..7] OF BOOL;
(* Массив датчиков наличия деталей *)
sensorCell : ARRAY[0..7] OF BOOL;

freeIndex  : INT := -1;  (* -1 означает "не найдено" *)
fullIndex  : INT := -1;
i          : INT;       (* счётчик цикла *)
totalParts : INT;       (* общее количество деталей *)

(* Дополнительно: управление *)
bufferFull : BOOL;      (* накопитель полон *)
bufferEmpty : BOOL;     (* накопитель пуст *)
```

END_VAR

Шаг 3 — Ввести тело программы

```
(* =====
ШАГ 1: Синхронизация с датчиками
Обновляем массив cells по реальным датчикам
===== *)
FOR i := 0 TO 7 DO
    cells[i] := sensorCell[i];
END_FOR;

(* =====
ШАГ 2: Поиск первой свободной ячейки
(для робота – куда положить деталь)
===== *)
freeIndex := -1;          (* сбрасываем перед поиском *)
FOR i := 0 TO 7 DO
    IF NOT cells[i] AND freeIndex = -1 THEN
        freeIndex := i;   (* нашли – запоминаем индекс *)
        (* НЕ выходим из цикла – в ST нет break, продолжаем *)
        (* Условие freeIndex = -1 не даст перезаписать найденный индекс *)
    END_IF;
END_FOR;

(* =====
ШАГ 3: Поиск первой занятой ячейки
(для станка – откуда взять деталь)
===== *)
fullIndex := -1;
FOR i := 0 TO 7 DO
    IF cells[i] AND fullIndex = -1 THEN
        fullIndex := i;
    END_IF;
END_FOR;
```

```
END_FOR;
```

```
(* =====  
    ШАГ 4: Подсчёт деталей в накопителе  
    ===== *)
```

```
totalParts := 0;  
FOR i := 0 TO 7 DO  
    IF cells[i] THEN  
        totalParts := totalParts + 1;  
    END_IF;  
END_FOR;
```

```
(* =====  
    ШАГ 5: Флаги состояния накопителя  
    ===== *)
```

```
bufferFull  := (totalParts = 8);    (* все ячейки заняты *)  
bufferEmpty := (totalParts = 0);    (* все ячейки свободны *)
```

Шаг 4 — Протестировать с Watch Window

Добавьте в Watch Window массив `cells` — кликните на переменную, она раскроется как таблица:

```
cells[0] = FALSE  
cells[1] = FALSE  
...  
cells[7] = FALSE
```

Тест 1 — Поиск свободной ячейки:

1. Все `sensorCell[i] = FALSE` → `freeIndex = 0`, `totalParts = 0`, `bufferEmpty = TRUE`
2. Установите `sensorCell[0] = TRUE`, `sensorCell[1] = TRUE`
3. Ожидаем: `freeIndex = 2` (первая незанятая), `totalParts = 2`

Тест 2 — Полный накопитель:

1. Установите все `sensorCell[0..7] = TRUE`
2. Ожидаем: `freeIndex = -1`, `totalParts = 8`, `bufferFull = TRUE`

Тест 3 — Поиск занятой ячейки:

1. Только `sensorCell[3] = TRUE` и `sensorCell[5] = TRUE`
2. Ожидаем: `fullIndex = 3` (первая занятая), `totalParts = 2`

Шаг 5 — Самостоятельная доработка

Добавьте функцию **LIFO** (последний вошёл — первый вышел): найдите **последнюю** занятую ячейку (проход цикла от 7 до 0):

```
(* Поиск последней занятой ячейки – LIFO *)
```

```
VAR  
    lastFullIndex : INT := -1;  
END_VAR
```

```

lastFullIndex := -1;
FOR i := 7 TO 0 BY -1 DO    (* BY -1 — шаг цикла отрицательный *)
    IF cells[i] AND lastFullIndex = -1 THEN
        lastFullIndex := i;
    END_IF;
END_FOR;

```

Вопросы для самопроверки

1. Почему перед каждым поиском нужно сбрасывать freeIndex := -1?
2. Что означает значение -1 для freeIndex — как его интерпретирует вызывающий код?
3. Почему в ST нет оператора break для выхода из цикла и как это обходится?
4. Как изменить программу чтобы накопитель имел 16 позиций?

РАБОТА №6 — Полный цикл роботизированной ячейки (2 ч)

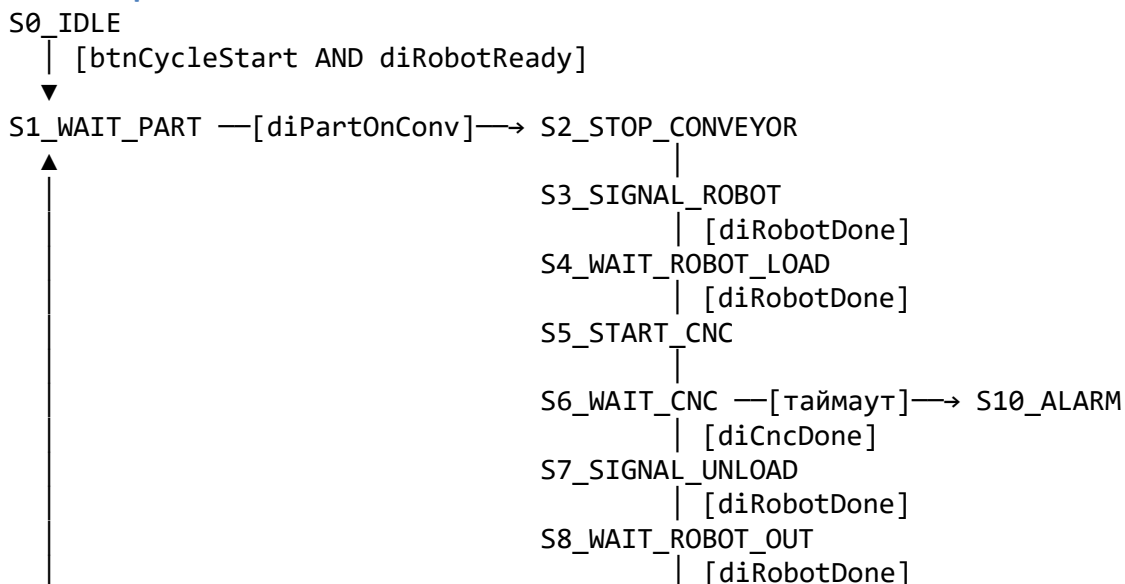
Цель

Написать полную программу ПЛК управления ячейкой «станок с ЧПУ + робот + конвейер», объединив все изученные конструкции языка ST.

Что изучается

- Сложная машина состояний с 11 состояниями
- Таймер таймаута для критических операций
- CTU для учёта производительности
- Приоритетная обработка E-Stop
- Архитектура реальной промышленной программы

Схема цикла ячейки



```

┌────────────────── S9_START_CONVEYOR ─────────────────┐
└────────────────── (счётчик +1) ─────────────────┘

```

Пошаговое выполнение

Шаг 1 — Создать проект Lab06_RoboCell

Шаг 2 — Создать ENUM E_CycleStep

(Аналогично работе №3 — через Add Object → DUT):

```

TYPE E_CycleStep :
(
    S0_IDLE           := 0,
    S1_WAIT_PART      := 1,
    S2_STOP_CONVEYOR := 2,
    S3_SIGNAL_ROBOT   := 3,
    S4_WAIT_ROBOT_LOAD := 4,
    S5_START_CNC       := 5,
    S6_WAIT_CNC        := 6,
    S7_SIGNAL_UNLOAD  := 7,
    S8_WAIT_ROBOT_OUT  := 8,
    S9_START_CONVEYOR := 9,
    S10_ALARM          := 10
);
END_TYPE

```

Шаг 3 — Ввести переменные PLC_PRG

```

VAR
    step           : E_CycleStep := S0_IDLE;

    (* Входные сигналы (DI – Digital Input) *)
    btnCycleStart   : BOOL;      (* кнопка запуска цикла *)
    diPartOnConv     : BOOL;      (* датчик: деталь на конвейере *)
    diRobotReady     : BOOL;      (* робот готов к работе *)
    diRobotDone      : BOOL;      (* робот завершил операцию *)
    diCncDone        : BOOL;      (* станок завершил обработку *)
    diEStop          : BOOL;      (* аварийная остановка *)

    (* Выходные сигналы (DO – Digital Output) *)
    doConveyor       : BOOL;      (* конвейер вкл/выкл *)
    doRobotStart      : BOOL;      (* старт робота – операция загрузки *)
    doRobotUnload     : BOOL;      (* старт робота – операция выгрузки *)
    doCncStart        : BOOL;      (* запуск цикла обработки на станке *)
    doAlarmLamp       : BOOL;      (* лампа аварии *)

    (* Сервисные переменные *)
    tCncTimeout       : TON;      (* таймер таймаута станка *)
    cycleCount        : CTU;      (* счётчик завершённых циклов *)
    totalCycles       : INT;      (* текущее количество деталей *)
    prevStep          : E_CycleStep; (* предыдущий шаг – для отладки *)
END_VAR

```

Шаг 4 — Ввести тело программы

```
(* =====
E-STOP — АБСОЛЮТНЫЙ ПРИОРИТЕТ
Выполняется каждый цикл ПЛК
===== *)
IF diEStop THEN
    step := S10_ALARM;
END_IF;

(* =====
СОХРАНЕНИЕ ПРЕДЫДУЩЕГО ШАГА
Полезно для отладки
===== *)
prevStep := step;

(* =====
ОСНОВНАЯ МАШИНА СОСТОЯНИЙ
===== *)
CASE step OF

    S0_IDLE:
        (* Начальное безопасное состояние *)
        doConveyor    := FALSE;
        doRobotStart   := FALSE;
        doRobotUnload  := FALSE;
        doCncStart      := FALSE;
        doAlarmLamp     := FALSE;
        (* Запуск только если нажата кнопка И робот готов *)
        IF btnCycleStart AND diRobotReady THEN
            step := S1_WAIT_PART;
        END_IF;

    S1_WAIT_PART:
        (* Запускаем конвейер и ждём появления детали *)
        doConveyor := TRUE;
        IF diPartOnConv THEN
            step := S2_STOP_CONVEYOR;
        END_IF;

    S2_STOP_CONVEYOR:
        (* Деталь обнаружена — останавливаем конвейер *)
        doConveyor := FALSE;
        (* Сразу переходим к следующему шагу (без условия) *)
        step := S3_SIGNAL_ROBOT;

    S3_SIGNAL_ROBOT:
        (* Даем роботу команду забрать деталь с конвейера и загрузить в станок
        *)
        doRobotStart := TRUE;
        IF diRobotDone THEN
            doRobotStart := FALSE; (* снимаем команду *)
```

```

        step := S4_WAIT_ROBOT_LOAD;
    END_IF;

S4_WAIT_ROBOT_LOAD:
    (* Ждём подтверждения что деталь загружена в станок *)
    IF diRobotDone THEN
        step := S5_START_CNC;
    END_IF;

S5_START_CNC:
    (* Даём команду станку начать обработку *)
    doCncStart := TRUE;
    step       := S6_WAIT_CNC;  (* переходим сразу *)

S6_WAIT_CNC:
    (* Ждём окончания обработки с контролем таймаута *)
    tCncTimeout(IN := TRUE, PT := T#120S);  (* 2 минуты максимум *)
    IF diCncDone THEN
        (* Нормальное завершение *)
        doCncStart := FALSE;
        tCncTimeout(IN := FALSE, PT := T#120S);  (* сброс таймера *)
        step := S7_SIGNAL_UNLOAD;
    ELSIF tCncTimeout.Q THEN
        (* Станок не завершил за 2 минуты – авария *)
        doCncStart := FALSE;
        step := S10_ALARM;
    END_IF;

S7_SIGNAL_UNLOAD:
    (* Даём роботу команду выгрузить готовую деталь из станка *)
    doRobotUnload := TRUE;
    IF diRobotDone THEN
        doRobotUnload := FALSE;
        step := S8_WAIT_ROBOT_OUT;
    END_IF;

S8_WAIT_ROBOT_OUT:
    (* Ждём подтверждения что деталь уложена на выход *)
    IF diRobotDone THEN
        step := S9_START_CONVEYOR;
    END_IF;

S9_START_CONVEYOR:
    (* Цикл завершён – увеличиваем счётчик и начинаем снова *)
    cycleCount(CU := TRUE, R := FALSE, PV := 32767);
    totalCycles := cycleCount.CV;
    step := S1_WAIT_PART;  (* возврат к ожиданию следующей детали *)

S10_ALARM:
    (* Аварийное состояние – всё остановлено *)
    doConveyor := FALSE;
    doRobotStart := FALSE;

```

```

doRobotUnload := FALSE;
doCncStart     := FALSE;
doAlarmLamp    := TRUE;
tCncTimeout(IN := FALSE, PT := T#120S); (* сброс таймера *)
(* Сброс: по кнопке Reset (добавить самостоятельно) *)

```

END_CASE;

Шаг 5 — Протестировать полный цикл

Создайте Watch Window с переменными: step, doConveyor, doRobotStart, doRobotUnload, doCncStart, totalCycles

Полный цикл (шаг за шагом):

Шаг	Действие оператора	Ожидаемый step	Что включено
1	diRobotReady=TRUE, btnCycleStart=TRUE/FALSE	S1_WAIT_PART	doConveyor=TRUE
2	diPartOnConv=TRUE/FALSE	S2→S3	doConveyor=FALSE, doRobotStart=TRUE
3	diRobotDone=TRUE/FALSE (2 раза)	S4→S5→S6	doCncStart=TRUE
4	diCncDone=TRUE/FALSE	S7	doRobotUnload=TRUE
5	diRobotDone=TRUE/FALSE (2 раза)	S8→S9→S1	totalCycles=1

Тест аварии по E-Stop:

- В любой момент цикла: diEStop=TRUE
- Все выходы = FALSE, doAlarmLamp=TRUE, step=S10_ALARM

Тест таймаута станка:

- Дойдите до шага S6_WAIT_CNC
- НЕ устанавливайте diCncDone
- Через 120 секунд (в тесте — уменьшите PT до T#10S): step=S10_ALARM

Вопросы для самопроверки

1. Почему doCncStart устанавливается в TRUE в S5, а снимается в S6?
2. Что произойдёт если diRobotDone уже был TRUE до прихода в шаг S3?
3. Зачем нужен prevStep и как он помогает при отладке?
4. Добавьте самостоятельно: кнопку сброса аварии (переход из S10_ALARM → S0_IDLE)

ПРАКТИЧЕСКАЯ РАБОТА 2.3 — Отладка программы с ошибками (2 ч)

Цель

Развить навык поиска и исправления ошибок в программах ST — синтаксических, логических и временных.

Задание

Ниже приведена программа управления сортировочной ячейкой с **9 намеренно внесёнными ошибками**. Найдите все ошибки, исправьте и проверьте корректную работу.

Программа с ошибками

Создайте проект **PractWork_Debug**, введите программу как есть (с ошибками):

VAR

```
sensorType1 : BOOL;  
sensorType2 : BOOL;  
btnStart    : BOOL;  
btnStop     : BOOL;  
eStop       : BOOL;
```

```
conveyorRun : BOOL;  
pusher1     : BOOL;  
pusher2     : BOOL;  
lampAlarm   : BOL;
```

(* ОШИБКА 1 *)

```
count1      : CTU;  
count2      : CTU;  
parts1      : INT;  
parts2      : INT;
```

```
tPusher1    : TON;  
tPusher2    : TON;  
running     : BOOL;  
alarm       : BOOL;
```

END_VAR

(* E-Stop *)

```
IF eStop THEN  
    conveyorRun := FALSE;  
    alarm       := TRUE;  
    lampAlarm   = TRUE;
```

(* ОШИБКА 2 *)

END_IF

(* Управление конвейером *)

(* ОШИБКА 3 *)

```
IF btnStart AND NOT alarm THEN  
    running := TRUE;  
ELSIF btnStop THEN  
    running := FALSE
```

```

END_IF;                                (* ОШИБКА 4 *)

conveyorRun := running AND NOT eStop;

(* Сортировка *)
IF sensorType1 AND conveyorRun THEN
    pusher1 := TRUE;
END_IF;

tPusher1(IN := pusher1, PT := T#0.5S);
IF tPusher1.Q THEN
    pusher1 := FALSE;
    tPusher1(IN := TRUE, PT := T#0.5S); (* ОШИБКА 5 *)
END_IF;

IF sensorType2 AND conveyorRun THEN
    pusher2 := TRUE;
END_IF;

tPusher2(IN := pusher2, PT := T#0.5S);
IF tPusher2.Q THEN
    pusher2 := FALSE;
END_IF;
(* Счётчики *)
count1(CU := sensorType1, R := FALSE, PV := 9999);
parts1 := count1.CV;

count2(CU := sensorType2, R := FALSE, PV := 9999);
parts2 := count2.Q; (* ОШИБКА 6 *)

(* Проверка: суммарное количество *)
IF parts1 + parts2 > 100 THEN
    alarm := TRUE; (* ОШИБКА 7 – логическая *)
END_IF;

(* Сброс аварии *)
IF btnStart AND NOT eStop THEN (* ОШИБКА 8 – логическая *)
    alarm := FALSE;
    lampAlarm := FALSE;
END_IF;

(* Защита от одновременной работы толкателей *)
IF pusher1 = TRUE THEN
    pusher2 := FALSE;
END_IF; (* ОШИБКА 9 – логическая: неполная защита *)

```

Таблица для заполнения учащимся

№	Строка	Тип ошибки	Описание ошибки	Исправление
1	lampAlarm : BOL	Синтаксическая		
2	lampAlarm = TRUE	Синтаксическая		
3	END_IF без ;	Синтаксическая		
4	running := FALSE без ;	Синтаксическая		
5	tPusher1(IN := TRUE...)	Временная		
6	parts2 := count2.Q	Синтаксическая/логическая		
7	Счётчик > 100 → alarm	Логическая		
8	Сброс по btnStart	Логическая		
9	Неполная защита толкателей	Логическая		

Подсказки (не читать сразу!)

Подсказка к ошибке 1

Правильный тип данных — BOOL (четыре буквы)

Подсказка к ошибке 5

Таймер TON сбрасывается когда IN := FALSE. Если после Q вы снова ставите IN := TRUE — таймер немедленно начинает новый отсчёт и никогда не сбросится. Нужно:
tPusher1(IN := FALSE, PT := T#0.5S);

Подсказка к ошибке 6

count2.Q — это BOOL (флаг переполнения), а не значение счётчика. Значение счётчика — count2.CV (Current Value)

Подсказка к ошибке 7

Сброс аварии должен выполняться только по специальной кнопке btnReset, а не автоматически по количеству деталей. Иначе авария будет сбрасываться и выставляться каждый цикл ПЛК.

Подсказка к ошибке 8

Сброс аварии по кнопке Пуск — опасная практика. Нужна отдельная кнопка btnReset. Кроме того, при сбросе нужно также сбросить running := FALSE и перезапустить с чистого состояния.

Подсказка к ошибке 9

Защита работает только в одну сторону: если pusher1 активен — pusher2 выключается. Но если pusher2 активен первым — pusher1 не блокируется. Нужно добавить симметричное условие.

Содержание протокола отладки

ПРОТОКОЛ ОТЛАДКИ

Дисциплина: Программирование для автоматизированного оборудования

Учащийся: _____ Группа: _____ Дата: _____

1. СИНТАКСИЧЕСКИЕ ОШИБКИ (найдено: ____ из 5)
 Ошибка №1: строка ____, описание: ____, исправление: ____
 ...
2. ЛОГИЧЕСКИЕ ОШИБКИ (найдено: ____ из 3)
 Ошибка №__: строка ____, описание: ____, последствия: ____, исправление: ____
 ...
3. ВРЕМЕННЫЕ ОШИБКИ (найдено: ____ из 1)
 Ошибка №__: строка ____, описание: ____, последствия: ____, исправление: ____
4. ПРОВЕРКА ПОСЛЕ ИСПРАВЛЕНИЯ
 Тест 1 (пуск/стоп): ПРОЙДЕН / НЕ ПРОЙДЕН
 Тест 2 (сортировка): ПРОЙДЕН / НЕ ПРОЙДЕН
 Тест 3 (E-Stop): ПРОЙДЕН / НЕ ПРОЙДЕН
5. ВЫВОД:
 Самая опасная ошибка: ____
 Ошибка труднее всего обнаруживалась: ____

ИТОГОВАЯ ПАМЯТКА ПО ВСЕМ РАБОТАМ

Ключевые конструкции ST — краткий справочник

Конструкция	Синтаксис	Когда применять
Присваивание	x := 5;	Всегда с := и ;
Условие	IF ... THEN ... ELSIF ... ELSE ... END_IF;	Ветвление по условию
Выбор	CASE var OF val: ... END_CASE;	Машина состояний, режимы
Цикл	FOR i:=0 TO N DO ... END_FOR;	Обработка массивов
Таймер	t(IN:=cond, PT:=T#2S); IF t.Q THEN...	Выдержки, таймауты
Счётчик	c(CU:=sig, R:=rst, PV:=max);	Подсчёт событий
Фронт	rt(CLK:=sig); IF rt.Q THEN...	Однократная реакция
Выход из FB	RETURN;	Прерывание выполнения

Типичные ошибки начинающих

Ошибка	Неверно	Верно
Оператор присваивания	x = 5	x := 5
Пропущена точка с запятой	x := 5 (без ;)	x := 5;
Незакрытый блок	IF ... THEN ...	IF ... THEN ... END_IF;
Чтение выхода таймера	tGrip = TRUE	tGrip.Q = TRUE
Значение счётчика	cnt.Q	cnt.CV
Таймер не сбрасывается	t(IN:=TRUE...) после Q	t(IN:=FALSE...) для сброса

Telegram: <https://t.me/plcmasters>

Любое копирование, перепечатка или передача данного контента без явного письменного разрешения правообладателя запрещена законом и может привести к юридической ответственности.