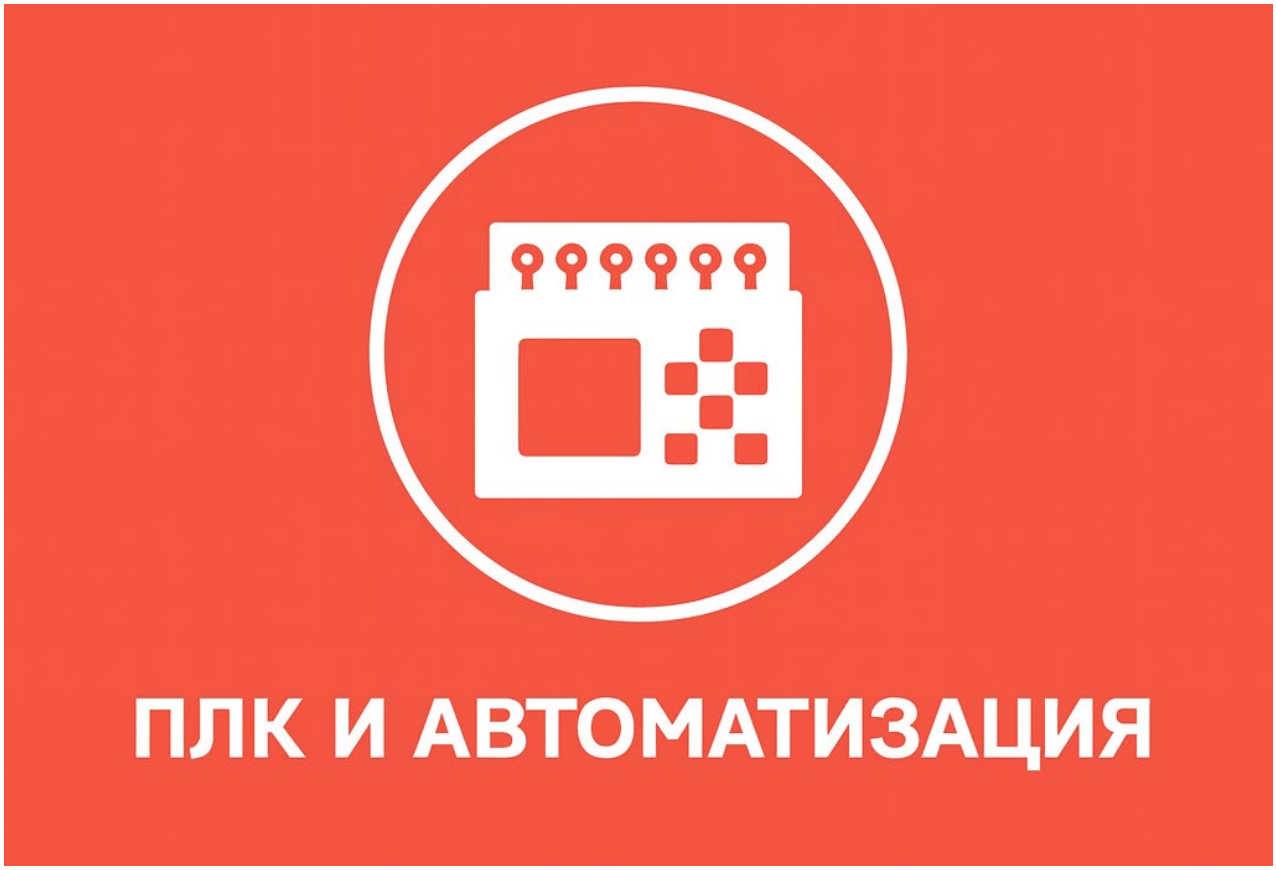

ST в примерах: 50 коротких задач для начинающих

Telegram: https://t.me/plcmasters

Введение

Это пособие предназначено для тех, кто только начинает программировать ПЛК на языке Structured Text (ST) по стандарту IEC 61131-3.
Вместо длинной теории здесь собраны 50 небольших практических задач, каждая на 1–2 страницы: постановка на «человеческом» языке, готовый код ST и подробный разбор.

Для кого это пособие

- Электрики, наладчики и инженеры по АСУТП, знакомые с релейной логикой и лестничными схемами.
- Начинающие программисты ПЛК, которые уже видели ST, но не уверены в правильности решений.
- Студенты технических специальностей, которым нужен «живой» практический курс.

Принятые соглашения

- Язык: стандартный ST по IEC 61131-3 (структурированный текст).
- Таймеры и счётчики: используются стандартные функциональные блоки TON, TOF, TP.
- Типы данных: BOOL, INT, DINT, REAL, TIME, ARRAY, STRUCT.
- Адреса входов/выходов условные: `I_Start`, `Q_Motor` и т.п., чтобы код был переносимым между платформами.
- Все примеры написаны так, чтобы их можно было без труда адаптировать под Siemens, Codesys, Beckhoff и др. — иногда потребуется лишь поправить имена системных переменных или FB.

1. «Мигалка» — простой мигающий выход

Задача

Сделать мигающий выход `Q_Blink` с периодом 1 секунда (0.5 с включено, 0.5 с выключено).

Используем стандартный таймер TON и переменную для состояния.

Программа ST

```
PROGRAM Example01_Blink
VAR
   BlinkTimer : TON;      (* Таймер для полупериода *)
   BlinkState : BOOL := FALSE;
END_VAR

(* Каждый цикл вызываем функциональный блок таймера *)
BlinkTimer(IN := TRUE, PT := T#500ms);

IF BlinkTimer.Q THEN
   (* Полупериод истёк — инвертируем состояние и перезапускаем таймер *)
   BlinkState := NOT BlinkState;
   BlinkTimer(IN := FALSE);   (* сброс таймера *)
   BlinkTimer(IN := TRUE);    (* немедленный запуск нового периода *)
END_IF;

Q_Blink := BlinkState;
END_PROGRAM
```

Разбор

- BlinkTimer : TON — стандартный таймер включения; при IN=TRUE считает время до PT, затем устанавливает Q=TRUE.
- Устанавливаем PT := T#500ms, чтобы получить полупериод 0.5 с; полный период мигания — 1 с.
- Когда BlinkTimer.Q = TRUE, мы:
  - инвертируем BlinkState,
  - кратко выключаем (IN := FALSE), затем снова включаем таймер, чтобы он начал новый отсчёт.
- На каждом цикле присваиваем Q_Blink := BlinkState, поэтому выход повторяет внутреннее логическое состояние.

2. Мигание только при нажатой кнопке

Задача

Выход Q_Blink должен мигать, только пока нажата кнопка I_Button.
Когда кнопку отпускают, выход должен немедленно выключаться.

Программа ST

```
PROGRAM Example02_BlinkOnButton
VAR
   BlinkTimer : TON;
   BlinkState : BOOL := FALSE;
END_VAR

IF I_Button THEN
```

```
    BlinkTimer(IN := TRUE, PT := T#500ms);

    IF BlinkTimer.Q THEN
        BlinkState := NOT BlinkState;
        BlinkTimer(IN := FALSE);
        BlinkTimer(IN := TRUE);
    END_IF;
ELSE
    (* Кнопка не нажата — сбрасываем всё *)
    BlinkTimer(IN := FALSE);
    BlinkState := FALSE;
END_IF;

Q_Blink := BlinkState;
END_PROGRAM
```

Разбор

- Вся логика мигания обёрнута в `IF I_Button THEN`.
- При отпускании кнопки мы:
    - выключаем таймер (`IN := FALSE`),
    - сбрасываем `BlinkState` в `FALSE`, чтобы выход гарантированно погас.
- Таким образом, при новом нажатии мигание всегда начинается с состояния «выключено».

3. Пуск/стоп двигателя с самоподхватом

Задача

Реализовать классическую схему «Пуск–Стоп» двигателя:

- Кнопка I_Start — запускает двигатель.
- Кнопка I_Stop — останавливает двигатель.
- Используем самоподхват: двигатель остаётся включённым после отпускания I_Start.

Программа ST

```
PROGRAM Example03_MotorStartStop
VAR
   MotorRun : BOOL := FALSE;
END_VAR

(* Обработка кнопок *)
IF I_Stop THEN
   MotorRun := FALSE;
ELSIF I_Start THEN
   MotorRun := TRUE;
END_IF;

(* Выход на двигатель *)
Q_Motor := MotorRun;
END_PROGRAM
```

Разбор

- MotorRun — логическое состояние двигателя (аналог реле самоподхвата).
- Приоритет «Стоп» выше: если нажаты обе кнопки, первым срабатывает блок IF I_Stop.
- Состояние MotorRun сохраняется между циклами за счёт того, что это обычная переменная программы.

4. Пуск/стоп с блокировкой по аварии

Задача

Расширить предыдущий пример:

- При I_Emergency = TRUE двигатель должен быть выключен независимо от I_Start.
- После аварии двигатель не должен запускаться, пока авария активна.

Программа ST

```
PROGRAM Example04_MotorWithEmergency
VAR
   MotorRun : BOOL := FALSE;
END_VAR

IF I_Emergency THEN
   MotorRun := FALSE;            (* Жёсткая блокировка *)
ELSIF I_Stop THEN
   MotorRun := FALSE;
```

```
ELSIF I_Start THEN
   MotorRun := TRUE;
END_IF;

Q_Motor := MotorRun;
END_PROGRAM
```

Разбор

- I_Emergency проверяется в самом первом IF.
- Пока I_Emergency = TRUE, MotorRun всегда обнуляется.
- После снятия аварии мотор можно перезапустить обычной кнопкой Start.

5. Антидребезг кнопки по счётчику циклов

Задача

Реализовать программный антидребезг:

- Вход I_ButtonRaw дребезжит при нажатии/отпускании.
- Нужно получить стабильный сигнал ButtonDebounced, который меняется только если I_ButtonRaw удерживается в новом состоянии не менее 50 мс.
- Предположим, что цикл ПЛК — 10 мс.

Программа ST

```
PROGRAM Example05_ButtonDebounce
VAR
  ButtonDebounced : BOOL := FALSE;
  Counter       : INT := 0;
  RequiredCycles  : INT := 5;   (* 5 * 10мс = 50мс *)
END_VAR

IF I_ButtonRaw = ButtonDebounced THEN
  (* Состояние не изменилось — сбрасываем счётчик *)
  Counter := 0;
ELSE
  (* Состояние новое — накапливаем время *)
  Counter := Counter + 1;

  IF Counter >= RequiredCycles THEN
    ButtonDebounced := I_ButtonRaw;   (* принимаем новое состояние *)
    Counter := 0;
  END_IF;
END_IF;

Q_Button := ButtonDebounced;
END_PROGRAM
```

Разбор

- Если «сырое» состояние (I_ButtonRaw) совпадает с уже отфильтрованным (ButtonDebounced), счётчик дребезга обнуляем.
- Если оно отличается, начинаем считать циклы до RequiredCycles.

- Только если новое состояние удерживается достаточно долго, принимаем его как истинное.

6. Однократное срабатывание при нажатии (front detection)

Задача

По отфильтрованной кнопке `ButtonDebounced` нужно получить импульс `ButtonPressed`, который равен TRUE только один цикл при переходе 0→1.

Программа ST

```
PROGRAM Example06_RisingEdge
VAR
   ButtonDebounced : BOOL;    (* предполагаем, получен из Example05 *)
   LastState       : BOOL := FALSE;
   ButtonPressed   : BOOL := FALSE;
END_VAR

ButtonPressed := (ButtonDebounced AND NOT LastState);
LastState := ButtonDebounced;

Q_PressedPulse := ButtonPressed;
END_PROGRAM
```

Разбор

- `LastState` хранит состояние кнопки на предыдущем цикле.
- Импульс возникает, когда сейчас TRUE, а раньше было FALSE.

- Такой подход используется во множестве примеров далее.

7. Счётчик деталей по датчику

Задача

Подсчитать количество деталей, проходящих мимо датчика I_Sensor.
На каждый фронт (0→1) увеличиваем PartCount. Реакция не должна зависеть от длительности импульса.

Программа ST

```
PROGRAM Example07_PartCounter
VAR
   LastSensor : BOOL := FALSE;
   PartCount  : DINT := 0;
END_VAR

IF (I_Sensor AND NOT LastSensor) THEN
   PartCount := PartCount + 1;
END_IF;

LastSensor := I_Sensor;

Q_PartCount := PartCount;    (* может быть, передаём в HMI *)
END_PROGRAM
```

Разбор

- Логика такая же, как в детекторе фронта, но вместо импульса мы увеличиваем счётчик.
- PartCount : DINT, чтобы избежать переполнения при большом числе деталей.

8. Счётчик с кнопкой сброса

Задача

Расширить предыдущий пример:

- Кнопка I_Reset сбрасывает счётчик в ноль.
- Защита от дребезга кнопки сброса не обязательна (по желанию можно добавить).

Программа ST

```
PROGRAM Example08_PartCounterWithReset
VAR
   LastSensor : BOOL := FALSE;
   PartCount  : DINT := 0;
END_VAR

IF I_Reset THEN
   PartCount := 0;
ELSE
   IF (I_Sensor AND NOT LastSensor) THEN
      PartCount := PartCount + 1;
   END_IF;
```

```
END_IF;

LastSensor := I_Sensor;

Q_PartCount := PartCount;
END_PROGRAM
```

Разбор

- Сброс имеет приоритет: пока I_Reset = TRUE, счётчик всегда 0.
- При отпускании кнопки счётчик снова начинает увеличиваться при новых деталях.

9. Простая задержка включения выхода (TON)

Задача

Включать выход Q_Delayed только если вход I_Command удерживается TRUE не менее 3 секунд.
Если вход пропал раньше — ничего не включаем.

Программа ST

```
PROGRAM Example09_DelayOn
VAR
   DelayTimer : TON;
END_VAR

DelayTimer(IN := I_Command, PT := T#3s);
```

```
Q_Delayed := DelayTimer.Q;
END_PROGRAM
```

Разбор

- Классический таймер включения: Q становится TRUE после непрерывного TRUE на IN в течение PT.
- Если IN сбрасывается раньше 3 с, таймер сбрасывается, и Q не успевает стать TRUE.

10. Задержка отключения выхода (TOF)

Задача

Выход Q_Hold должен оставаться TRUE ещё 2 секунды после того, как команда I_Command стала FALSE.
Используем таймер отключения (TOF).

Программа ST

```
PROGRAM Example10_DelayOff
VAR
   OffTimer : TOF;
END_VAR

OffTimer(IN := I_Command, PT := T#2s);

Q_Hold := OffTimer.Q;
```

```
END_PROGRAM
```

Разбор

- При IN=TRUE таймер TOF сразу даёт Q=TRUE.
- Когда IN переходит в FALSE, запускается отсчёт времени PT, и только по его окончании Q становится FALSE.

11. Импульс фиксированной длительности (TP)

Задача

По фронту сигнала I_Trigger сформировать импульс на выходе Q_Pulse длительностью 200 мс, независимо от длительности входного импульса.

Программа ST

```
PROGRAM Example11_PulseTP
VAR
   PulseTimer : TP;
END_VAR

PulseTimer(IN := I_Trigger, PT := T#200ms);

Q_Pulse := PulseTimer.Q;
END_PROGRAM
```

Разбор

- Стандартный таймер импульса TP: по фронту IN выдаёт Q=TRUE на время PT.
- Если I_Trigger дребезжит, стоит добавить антидребезг как в Example05.

12. Аварийный мигающий сигнал

Задача

Если I_Fault = TRUE, включить мигающий аварийный свет Q_AlarmBlink (1 Гц).
Если аварии нет — выход выключен.

Программа ST

```
PROGRAM Example12_AlarmBlink
VAR
  BlinkTimer : TON;
  BlinkState : BOOL := FALSE;
END_VAR

IF I_Fault THEN
  BlinkTimer(IN := TRUE, PT := T#500ms);

  IF BlinkTimer.Q THEN
    BlinkState := NOT BlinkState;
    BlinkTimer(IN := FALSE);
    BlinkTimer(IN := TRUE);
  END_IF;
ELSE
```

```
   BlinkTimer(IN := FALSE);
   BlinkState := FALSE;
END_IF;

Q_AlarmBlink := BlinkState;
END_PROGRAM
```

Разбор

- Логика похожа на Example02, только условие — сигнал аварии.
- Когда авария снимается, мигание останавливается и выход гаснет.

13. Масштабирование аналогового сигнала 0–27648 в 0–100%

Задача

Есть аналоговый вход `AI_Raw` (тип INT), где 0 соответствует 0%, а 27648 — 100%.
Нужно получить `AI_Percent` (REAL) в процентах.

Программа ST

```
PROGRAM Example13_AnalogScale
VAR
   AI_Raw     : INT;   (* 0 .. 27648 *)
   AI_Percent : REAL;
END_VAR

AI_Percent := REAL(AI_Raw) * 100.0 / 27648.0;
```

```
Q_AI_Percent := AI_Percent;
END_PROGRAM
```

Разбор

- Приводим AI_Raw к REAL, чтобы избежать целочисленного деления.
- Формула: (сырое_значение / максимум) * 100%.
- По желанию можно ограничить диапазон 0–100 с помощью MIN/MAX.

14. Гистерезис по уровню (верхний/нижний порог)

Задача

По сигналу уровня в процентах Level управлять насосом Q_Pump:

- Включать, когда уровень поднимется выше 80%.
- Выключать, когда опустится ниже 20%.
- В промежутке сохранять прошлое состояние насоса (гистерезис).

Программа ST

```
PROGRAM Example14_LevelHysteresis
VAR
  Level    : REAL;    (* 0 .. 100 % *)
  PumpOn   : BOOL := FALSE;
  OnLevel  : REAL := 80.0;
  OffLevel : REAL := 20.0;
```

```
END_VAR

IF Level > OnLevel THEN
   PumpOn := TRUE;
ELSIF Level < OffLevel THEN
   PumpOn := FALSE;
END_IF;

Q_Pump := PumpOn;
END_PROGRAM
```

Разбор

- При выходе за верхний порог включаем, при падении ниже нижнего — выключаем.
- В зоне между 20% и 80% состояние не меняем, что предотвращает частые переключения.

15. Автоматическое управление насосом по уровню

Задача

Использовать гистерезис (как выше) для простого управления насосом:

- `Level` — уровень в баке.
- Насос `Q_Pump` должен перекачивать воду из бака:
    - включается при уровне выше 80%,

- выключается при уровне ниже 30%.

Программа ST

```
PROGRAM Example15_PumpControl
VAR
   Level    : REAL;     (* 0 .. 100 % *)
   PumpOn   : BOOL := FALSE;
   OnLevel  : REAL := 80.0;
   OffLevel : REAL := 30.0;
END_VAR

IF Level >= OnLevel THEN
   PumpOn := TRUE;
ELSIF Level <= OffLevel THEN
   PumpOn := FALSE;
END_IF;

Q_Pump := PumpOn;
END_PROGRAM
```

Разбор

- Логика та же, границы можно настраивать через HMI.
- Простая, но типичная задача для АСУТП.

16. Простой конвейер: датчик — мотор

Задача

Конвейерный мотор Q_ConvMotor должен работать, только если:

- Есть общий разрешающий сигнал I_Enable.
- Нет аварии I_Emergency.
- Концевой датчик I_EndSwitch не сработал.

Программа ST

```
PROGRAM Example16_Conveyor
VAR
   MotorCmd : BOOL;
END_VAR

MotorCmd := I_Enable AND NOT I_Emergency AND NOT I_EndSwitch;

Q_ConvMotor := MotorCmd;
END_PROGRAM
```

Разбор

- Прямое логическое выражение читается практически как фраза.
- Если логика становится сложнее, её лучше раскладывать на промежуточные переменные с понятными именами.

17. Кольцевой буфер последних N событий

Задача

Сохранять последние 10 моментов времени нажатия кнопки I_ButtonDebounced (по фронту) в массиве Events[0..9].
Новая запись должна перезаписывать самую старую (кольцевой буфер).

Программа ST

```
PROGRAM Example17_RingBuffer
VAR
   I_ButtonDebounced : BOOL;
   LastButton       : BOOL := FALSE;
   Events           : ARRAY[0..9] OF TIME;
   Index            : INT := 0;
END_VAR

IF I_ButtonDebounced AND NOT LastButton THEN
   (* Фронт кнопки — запоминаем текущее время *)
   Events[Index] := TIME();   (* стандартная функция системного времени, зависит от ПЛК *)
   Index := Index + 1;
   IF Index > 9 THEN
      Index := 0;
   END_IF;
END_IF;

LastButton := I_ButtonDebounced;
END_PROGRAM
```

Разбор

- TIME() — абстрактная функция получения текущего времени (в конкретной системе может называться иначе).
- Index циклически проходит значения 0..9, перезаписывая старые записи.

18. Переключение режимов «ручной/авто»

Задача

Есть два режима управления насосом:

- Автоматический: по уровню (как в Example15).
- Ручной: оператор напрямую подаёт команду I_ManualCmd.

Режим выбирается переключателем I_ModeAuto (TRUE — авто, FALSE — ручной).

Программа ST

```
PROGRAM Example18_ManualAuto
VAR
  Level      : REAL;
  AutoCmd    : BOOL := FALSE;
  PumpOn     : BOOL := FALSE;
  OnLevel    : REAL := 80.0;
  OffLevel   : REAL := 30.0;
END_VAR

(* Автоматическая команда *)
IF Level >= OnLevel THEN
```

```
   AutoCmd := TRUE;
ELSIF Level <= OffLevel THEN
   AutoCmd := FALSE;
END_IF;

(* Выбор команды по режиму *)
IF I_ModeAuto THEN
   PumpOn := AutoCmd;
ELSE
   PumpOn := I_ManualCmd;
END_IF;

Q_Pump := PumpOn;
END_PROGRAM
```

Разбор

- Разделяем вычисление автоматической команды и выбор источника.
- Такая структура легко расширяется до нескольких режимов («ручной», «авто», «дистанционный» и т.п.).

19. Две скорости двигателя

Задача

У двигателя есть два режима:

- Низкая скорость `Q_SpeedLow`.

- Высокая скорость Q_SpeedHigh.

Правила:

- При I_SlowCmd = TRUE — включать только низкую скорость.
- При I_FastCmd = TRUE — включать только высокую (имеет приоритет над медленной).
- Оба выхода никогда не должны быть включены одновременно.

Программа ST

```
PROGRAM Example19_TwoSpeeds
VAR
   SpeedLow  : BOOL := FALSE;
   SpeedHigh : BOOL := FALSE;
END_VAR

IF I_FastCmd THEN
   SpeedHigh := TRUE;
   SpeedLow  := FALSE;
ELSIF I_SlowCmd THEN
   SpeedLow  := TRUE;
   SpeedHigh := FALSE;
ELSE
   SpeedLow  := FALSE;
   SpeedHigh := FALSE;
END_IF;

Q_SpeedLow  := SpeedLow;
Q_SpeedHigh := SpeedHigh;
```

```
END_PROGRAM
```

Разбор

- Приоритет высокой скорости реализован через порядок IF/ELSIF.
- Явно обнуляем второй выход при выборе первого, чтобы гарантировать отсутствия «гона».

20. Три состояния: стоп/пуск/реверс

Задача

У двигателя есть три режима:

- 0 — Стоп
- 1 — Пуск вперёд
- 2 — Пуск назад

Выбор задаётся переменной Mode : INT.
Нужно включить соответствующие выходы Q_Fwd и Q_Rev, не допуская одновременного включения.

Программа ST

```
PROGRAM Example20_ForwardReverse
VAR
   Mode : INT;  (* 0=Stop, 1=Forward, 2=Reverse *)
END_VAR
```

```
CASE Mode OF
   0:  (* Stop *)
      Q_Fwd := FALSE;
      Q_Rev := FALSE;
   1:  (* Forward *)
      Q_Fwd := TRUE;
      Q_Rev := FALSE;
   2:  (* Reverse *)
      Q_Fwd := FALSE;
      Q_Rev := TRUE;
ELSE
      (* Неверное значение — всё выключаем *)
      Q_Fwd := FALSE;
      Q_Rev := FALSE;
END_CASE;
END_PROGRAM
```

Разбор

- `CASE` аккуратно описывает взаимоисключающие режимы.
- В ветке `ELSE` полезно явно отключить всё для защиты от некорректных значений.

21. Простой конечный автомат мигания в три шага

Задача

Реализовать последовательность:

1. 1 с — выход Q_Lamp включён.
2. 1 с — выключен.
3. 2 с — снова включён.

Затем цикл повторяется.

Программа ST

```
PROGRAM Example21_BlinkFSM
VAR
  State     : INT := 0;
  StepTimer  : TON;
END_VAR

CASE State OF
  0:  (* Шаг 1: 1с включено *)
    Q_Lamp := TRUE;
    StepTimer(IN := TRUE, PT := T#1s);
    IF StepTimer.Q THEN
      StepTimer(IN := FALSE);
      State := 1;
    END_IF;

  1:  (* Шаг 2: 1с выключено *)
    Q_Lamp := FALSE;
    StepTimer(IN := TRUE, PT := T#1s);
    IF StepTimer.Q THEN
      StepTimer(IN := FALSE);
      State := 2;
    END_IF;
```

```
  2:  (* Шаг 3: 2с включено *)
      Q_Lamp := TRUE;
      StepTimer(IN := TRUE, PT := T#2s);
      IF StepTimer.Q THEN
         StepTimer(IN := FALSE);
         State := 0;
      END_IF;
END_CASE;
END_PROGRAM
```

Разбор

- State кодирует текущий шаг: 0, 1, 2.
- Для каждого шага используем один и тот же таймер StepTimer с разными PT.
- Это простейший пример конечного автомата (state machine) в ST.

22. Наработка оборудования (счётчик времени работы)

Задача

Считать суммарное время работы двигателя Q_Motor в секундах в переменной RunTime.
Предположим, что цикл ПЛК — 100 мс.

Программа ST

```
PROGRAM Example22_RunTimeCounter
VAR
   RunTime : DINT := 0;   (* секунды *)
   Acc     : INT  := 0;   (* накопление миллисекунд *)
END_VAR

IF Q_Motor THEN
   Acc := Acc + 100;      (* 100 мс за цикл *)
   IF Acc >= 1000 THEN
      RunTime := RunTime + 1;
      Acc := Acc - 1000;
   END_IF;
END_IF;

Q_RunTime := RunTime;      (* можно вывести на HMI *)
END_PROGRAM
```

Разбор

- Каждый цикл, пока мотор работает, в `Acc` добавляем 100 мс.
- Как только наберётся ≥1000 мс, увеличиваем секунды и вычитаем тысячу.

23. Вентиляция по температуре (простой двупороговый регулятор)

Задача

Вентилятор Q_Fan включается, когда температура Temp превышает 30 °C, и выключается, когда опускается ниже 25 °C (гистерезис).

Программа ST

```
PROGRAM Example23_FanByTemp
VAR
   Temp     : REAL;
   FanOn    : BOOL := FALSE;
   OnTemp   : REAL := 30.0;
   OffTemp  : REAL := 25.0;
END_VAR

IF Temp >= OnTemp THEN
   FanOn := TRUE;
ELSIF Temp <= OffTemp THEN
   FanOn := FALSE;
END_IF;

Q_Fan := FanOn;
END_PROGRAM
```

Разбор

- Та же идея гистерезиса, только по температуре.
- «Мёртвая зона» между 25 и 30 °C исключает частые переключения вентилятора.

24. Среднее значение по 4 датчикам

Задача

Есть четыре датчика температуры T1..T4.

Нужно вычислить среднюю температуру T_Avg.

Программа ST

```
PROGRAM Example24_Average4
VAR
   T1, T2, T3, T4 : REAL;
   T_Avg        : REAL;
END_VAR

T_Avg := (T1 + T2 + T3 + T4) / 4.0;

Q_T_Avg := T_Avg;
END_PROGRAM
```

Разбор

- Простейшее арифметическое среднее.
- В реальной системе полезно дополнительно проверять датчики на достоверность (нет ли обрывов и т.п.).

25. Контроль «залипания» датчика

Задача

Датчик I_Sensor должен периодически изменять своё состояние (например, каждая проходящая деталь).
Если в течение 10 секунд его состояние не меняется, считаем, что он «залип» и устанавливаем флаг SensorStuck.

Программа ST

```
PROGRAM Example25_StuckSensor
VAR
   LastState   : BOOL := FALSE;
   WatchTimer  : TON;
   SensorStuck : BOOL := FALSE;
END_VAR

IF I_Sensor <> LastState THEN
   (* Состояние изменилось — сбрасываем таймер *)
   WatchTimer(IN := FALSE);
   WatchTimer(IN := TRUE);
   SensorStuck := FALSE;
   LastState := I_Sensor;
ELSE
   (* Состояние не меняется — считаем время *)
   WatchTimer(IN := TRUE, PT := T#10s);
   IF WatchTimer.Q THEN
      SensorStuck := TRUE;
   END_IF;
END_IF;

Q_SensorStuck := SensorStuck;
```

```
END_PROGRAM
```

Разбор

- При каждом изменении входа перезапускаем таймер и сбрасываем флаг.
- Если состояние долго не меняется, таймер доходит до 10 с, и SensorStuck становится TRUE.

26. Взаимная блокировка двух двигателей

Задача

Два двигателя Q_M1 и Q_M2 не должны работать одновременно.

- Команда на M1: I_M1Cmd, на M2: I_M2Cmd.
- Если обе команды TRUE, включаем только M1 (приоритет).

Программа ST

```
PROGRAM Example26_MutualInterlock
VAR
  M1_Run : BOOL := FALSE;
  M2_Run : BOOL := FALSE;
END_VAR

IF I_M1Cmd THEN
  M1_Run := TRUE;
ELSE
  M1_Run := FALSE;
```

```
END_IF;

IF I_M2Cmd AND NOT M1_Run THEN
   M2_Run := TRUE;
ELSE
   M2_Run := FALSE;
END_IF;

Q_M1 := M1_Run;
Q_M2 := M2_Run;
END_PROGRAM
```

Разбор

- Сначала определяем состояние M1.
- M2 может включиться только если есть команда и M1 не работает.

27. Чередование двух насосов (lead/lag)

Задача

Два насоса должны работать по очереди:

- При первом запросе запускается насос 1, при следующем — насос 2 и так далее.
- Запрос на запуск — фронт сигнала `I_StartCycle`.

Программа ST

```
PROGRAM Example27_AlternatingPumps
VAR
   LastStart   : BOOL := FALSE;
   UsePump1    : BOOL := TRUE;
END_VAR

IF I_StartCycle AND NOT LastStart THEN
   (* Фронт запроса — переключаем ведущий насос *)
   UsePump1 := NOT UsePump1;
END_IF;

LastStart := I_StartCycle;

(* Управление насосами *)
Q_Pump1 := UsePump1 AND I_Enable;
Q_Pump2 := (NOT UsePump1) AND I_Enable;
END_PROGRAM
```

Разбор

- При каждом новом «цикле» меняем, какой насос считается ведущим.
- Оба насоса управляются одним разрешающим сигналом `I_Enable`.

28. Блокировка двигателя открытой дверью

Задача

Мотор Q_Motor может быть включён только если дверь шкафа закрыта (I_DoorClosed = TRUE).
Если дверь открывается во время работы, нужно немедленно остановить мотор.

Программа ST

```
PROGRAM Example28_DoorInterlock
VAR
   MotorRun : BOOL := FALSE;
END_VAR

IF NOT I_DoorClosed THEN
   MotorRun := FALSE;
ELSIF I_Stop THEN
   MotorRun := FALSE;
ELSIF I_Start THEN
   MotorRun := TRUE;
END_IF;

Q_Motor := MotorRun;
END_PROGRAM
```

Разбор

- Проверка двери стоит первой — это жёсткая блокировка.
- При открытии дверь сразу отключает MotorRun.

29. Простое меню параметров (3 пункта)

Задача

Есть 3 уставки P1, P2, P3.

Кнопками I_Next и I_Prev выбираем активную уставку (номер 0,1,2), отображаемую на HMI.

Программа ST

```
PROGRAM Example29_SimpleMenu
VAR
   P1, P2, P3 : REAL;
   Index     : INT := 0;     (* 0..2 *)
   LastNext   : BOOL := FALSE;
   LastPrev   : BOOL := FALSE;
END_VAR

(* Обработка кнопки Next *)
IF I_Next AND NOT LastNext THEN
   Index := Index + 1;
   IF Index > 2 THEN
      Index := 0;
   END_IF;
END_IF;
LastNext := I_Next;

(* Обработка кнопки Prev *)
IF I_Prev AND NOT LastPrev THEN
   Index := Index - 1;
```

```
  IF Index < 0 THEN
    Index := 2;
  END_IF;
END_IF;
LastPrev := I_Prev;

Q_SelectedIndex := Index;
END_PROGRAM
```

Разбор

- Кольцевое меню: 0→1→2→0 и обратно.
- Используем детекторы фронта для кнопок, чтобы индекс менялся только один раз при нажатии.

30. Сердцебиение («heartbeat») для диагностики

Задача

Сгенерировать бит `Q_Heartbeat`, который меняет состояние раз в секунду. Этот бит может использоваться SCADA, чтобы проверять, что ПЛК жив.

Программа ST

```
PROGRAM Example30_Heartbeat
VAR
  HB_Timer : TON;
  HB_Bit   : BOOL := FALSE;
END_VAR
```

```
HB_Timer(IN := TRUE, PT := T#1s);

IF HB_Timer.Q THEN
   HB_Bit := NOT HB_Bit;
   HB_Timer(IN := FALSE);
   HB_Timer(IN := TRUE);
END_IF;

Q_Heartbeat := HB_Bit;
END_PROGRAM
```

Разбор

- Аналог «мигалки» с периодом 2 с (1 с в одном состоянии, 1 с в другом).
- SCADA может контролировать, что `Q_Heartbeat` меняется, например, хотя бы раз в N секунд.

31. Логирование максимального значения

Задача

Отслеживать максимальное значение температуры `Temp` за время работы и сохранять его в `TempMax`.

Программа ST

```
PROGRAM Example31_MaxValue
VAR
```

```
  Temp    : REAL;
  TempMax : REAL := -1000.0;   (* заведомо ниже возможных значений *)
END_VAR

IF Temp > TempMax THEN
  TempMax := Temp;
END_IF;

Q_TempMax := TempMax;
END_PROGRAM
```

Разбор

- Инициализируем TempMax очень малым значением.
- При каждом цикле обновляем максимум, если текущее значение больше.

32. Выбор большего из двух сигналов

Задача

Даны два задания на скорость Setpoint1 и Setpoint2.
Нужно выбрать большее из них для управления: SetpointMax.

Программа ST

```
PROGRAM Example32_MaxOfTwo
VAR
  Setpoint1, Setpoint2 : REAL;
```

```
  SetpointMax        : REAL;
END_VAR

IF Setpoint1 >= Setpoint2 THEN
  SetpointMax := Setpoint1;
ELSE
  SetpointMax := Setpoint2;
END_IF;

Q_SetpointMax := SetpointMax;
END_PROGRAM
```

Разбор

- Простая конструкция IF/ELSE.
- Можно обобщить на массивы и циклы для N сигналов.

33. Программный TON без стандартного FB

Задача

Реализовать поведение таймера включения TON вручную:

- Вход `In`, параметр `Pt : TIME`.
- Выходы `Q : BOOL`, `Et : TIME`.

Программа ST

```
PROGRAM Example33_SoftwareTON
VAR
   In    : BOOL;
   Pt    : TIME := T#3s;
   Q     : BOOL := FALSE;
   Et    : TIME := T#0s;
   Cycle  : TIME := T#100ms;   (* длительность цикла *)
END_VAR

IF In THEN
   IF Et < Pt THEN
      Et := Et + Cycle;
      Q := FALSE;
   ELSE
      Q := TRUE;
   END_IF;
ELSE
   Et := T#0s;
   Q  := FALSE;
END_IF;

Q_TON_Q  := Q;
Q_TON_ET := Et;
END_PROGRAM
```

Разбор

- `Et` накапливает время в шаге `Cycle`.

- При выключении входа таймер сбрасывается.
- Это упрощённая модель, но хорошо помогает понять, как работает TON.

34. Расходомер: импульсы → объём

Задача

Датчик выдаёт один импульс `I_Pulse` на 10 литров жидкости.

Нужно посчитать суммарный объём `Volume` в литрах.

Программа ST

```
PROGRAM Example34_FlowMeter
VAR
   LastPulse : BOOL := FALSE;
   Volume    : REAL := 0.0;
END_VAR

IF I_Pulse AND NOT LastPulse THEN
   Volume := Volume + 10.0;
END_IF;

LastPulse := I_Pulse;

Q_Volume := Volume;
END_PROGRAM
```

Разбор

- На каждый фронт импульса добавляем 10 л.
- В реальности коэффициент может быть параметром, а не жёстко прошитым.

35. Однократное срабатывание при удержании кнопки

Задача

При нажатии и удержании кнопки `I_Button` должно происходить только одно действие — например, сброс счётчика.
Не важно, как долго кнопка удерживается.

Программа ST

```
PROGRAM Example35_OneShot
VAR
   LastButton : BOOL := FALSE;
   DoAction   : BOOL := FALSE;
END_VAR

DoAction := I_Button AND NOT LastButton;

IF DoAction THEN
   (* Здесь выполняем однократное действие, например: *)
   Counter := 0;
END_IF;

LastButton := I_Button;
```

```
END_PROGRAM
```

Разбор

- Это ещё один вариант детектора фронта.
- Задача — показать его использование для «одиночного» действия.

36. Двухкнопочный пуск

Задача

Двигатель Q_Motor можно запустить только при одновременном нажатии двух кнопок I_Start1 и I_Start2 (условный safety).
Кнопки должны быть нажаты в пределах 500 мс друг от друга.

Программа ST

```
PROGRAM Example36_TwoHandStart
VAR
   Start1Time : TIME := T#0s;
   Start2Time : TIME := T#0s;
   Window     : TIME := T#500ms;
END_VAR

IF I_Start1 THEN
   Start1Time := TIME();
END_IF;

IF I_Start2 THEN
```

```
   Start2Time := TIME();
END_IF;

IF (TIME() - Start1Time <= Window) AND
  (TIME() - Start2Time <= Window) THEN
   Q_Motor := TRUE;
ELSE
   Q_Motor := FALSE;
END_IF;
END_PROGRAM
```

Разбор

- Используем разность времени для проверки «окна» 500 мс.
- В реальной системе двухручный пуск реализуется аппаратно, но пример полезен для логики.

37. Сторожевой контроль собственной задачи

Задача

Проверять, что часть программы `DoWork()` выполняется не дольше 50 мс. Если дольше — устанавливать флаг `Overtime`.

Программа ST

```
PROGRAM Example37_Watchdog
VAR
   T_Start  : TIME;
```

```
  T_End    : TIME;
  Overtime : BOOL := FALSE;
END_VAR

T_Start := TIME();

DoWork();  (* условная тяжёлая функция *)

T_End := TIME();

IF (T_End - T_Start) > T#50ms THEN
  Overtime := TRUE;
ELSE
  Overtime := FALSE;
END_IF;

Q_Overtime := Overtime;
END_PROGRAM
```

Разбор

- Это программный «локальный» watchdog.
- Фактическая функция получения времени зависит от платформы.

38. Освещение по времени суток

Задача

Освещение Q_Light должно включаться с 18:00 до 06:00 по встроенным часам ПЛК.

Программа ST

```
PROGRAM Example38_LightByTime
VAR
   Hour : INT;
END_VAR

(* Предполагаем, что Hour обновляется системной функцией чтения часов ПЛК *)
Hour := PLC_HOUR();   (* абстрактная функция *)

IF (Hour >= 18) OR (Hour < 6) THEN
   Q_Light := TRUE;
ELSE
   Q_Light := FALSE;
END_IF;
END_PROGRAM
```

Разбор

- Условие «вечер или ночь» реализовано как >=18 ИЛИ <6.
- Реальные функции часов зависят от конкретного контроллера.

39. Поиск сработавшего датчика в массиве

Задача

Есть массив из 8 дискретных датчиков Sensors[0..7].

Нужно:

- Определить, есть ли хотя бы один сработавший.
- Найти индекс первого сработавшего и сохранить в FirstIndex (или -1, если нет ни одного).

Программа ST

```
PROGRAM Example39_FindFirstSensor
VAR
  Sensors    : ARRAY[0..7] OF BOOL;
  Found      : BOOL := FALSE;
  FirstIndex : INT := -1;
  i          : INT;
END_VAR

Found := FALSE;
FirstIndex := -1;

FOR i := 0 TO 7 DO
  IF Sensors[i] THEN
    Found := TRUE;
    FirstIndex := i;
    EXIT;     (* выходим из цикла *)
  END_IF;
END_FOR;
```

```
Q_Found     := Found;
Q_FirstIndex := FirstIndex;
END_PROGRAM
```

Разбор

- EXIT немедленно прекращает цикл после нахождения первого TRUE.
- Важно перед циклом сбросить Found и FirstIndex.

40. Подсчёт количества сработавших датчиков

Задача

Тот же массив Sensors[0..7].
Нужно подсчитать, сколько из них находятся в состоянии TRUE, и сохранить в Count.

Программа ST

```
PROGRAM Example40_CountSensors
VAR
   Sensors : ARRAY[0..7] OF BOOL;
   Count   : INT := 0;
   i       : INT;
END_VAR

Count := 0;

FOR i := 0 TO 7 DO
```

```
  IF Sensors[i] THEN
    Count := Count + 1;
  END_IF;
END_FOR;

Q_Count := Count;
END_PROGRAM
```

Разбор

- Наглядный пример использования цикла FOR по массиву.
- Тот же приём используется для любых «суммирований» по коллекции сигналов.

41. Приоритет команд: авария > ручной > авто

Задача

У нас три источника команды `Cmd`:

1. Аварийная блокировка `I_Fault` (приоритет максимальный — всегда стоп).
2. Ручная команда `I_ManualCmd`.
3. Автоматическая команда `AutoCmd`.

На выход `Q_Cmd` должно попадать решение с высшим приоритетом.

Программа ST

```
PROGRAM Example41_CommandPriority
VAR
  AutoCmd : BOOL;
  Cmd     : BOOL := FALSE;
END_VAR

IF I_Fault THEN
  Cmd := FALSE;             (* авария = стоп *)
ELSIF I_ManualCmd THEN
  Cmd := TRUE;              (* ручной режим *)
ELSE
  Cmd := AutoCmd;           (* автоматическая команда *)
END_IF;

Q_Cmd := Cmd;
END_PROGRAM
```

Разбор

- Приоритет реализуется естественным порядком IF/ELSIF.
- Легко расширить до большего числа уровней.

42. Выбор источника задания (local/remote)

Задача

Есть два задания скорости: локальное SpLocal и удалённое SpRemote. Переключатель I_Remote выбирает, какое из них использовать на выходе SpOut.

Программа ST

```
PROGRAM Example42_SetpointSource
VAR
  SpLocal, SpRemote : REAL;
  SpOut           : REAL;
END_VAR

IF I_Remote THEN
  SpOut := SpRemote;
ELSE
  SpOut := SpLocal;
END_IF;

Q_SpOut := SpOut;
END_PROGRAM
```

Разбор

- Простейший мультиплексор двух сигналов.
- Типичный блок в системах с «местным/дистанционным» управлением.

43. Простейший P-регулятор

Задача

Реализовать пропорциональный регулятор:

- Заданное значение Setpoint, измеренное Process.
- Ошибка Error = Setpoint - Process.
- Управляющее воздействие Out = Kp * Error.
- Ограничить Out диапазоном 0..100.

Программа ST

```
PROGRAM Example43_PController
VAR
   Setpoint, Process : REAL;
   Error, Out       : REAL;
   Kp               : REAL := 2.0;
END_VAR

Error := Setpoint - Process;
Out := Kp * Error;

(* Ограничение 0..100 *)
IF Out > 100.0 THEN
   Out := 100.0;
ELSIF Out < 0.0 THEN
   Out := 0.0;
END_IF;

Q_Out := Out;
```

```
END_PROGRAM
```

Разбор

- Это «голый» P-регулятор без интегральной/дифференциальной части.
- Всё равно полезный шаблон для обучения.

44. Ограничение скорости изменения задания (ramp)

Задача

Нужно плавно изменять `Current` к `Target`:

- За один цикл изменение не должно превышать `Step` (например, 1 единицу).

Программа ST

```
PROGRAM Example44_Ramp
VAR
   Current : REAL := 0.0;
   Target  : REAL;
   Step    : REAL := 1.0;
END_VAR

IF Current < Target THEN
   Current := Current + Step;
   IF Current > Target THEN
      Current := Target;
```

```
  END_IF;
ELSIF Current > Target THEN
  Current := Current - Step;
  IF Current < Target THEN
    Current := Target;
  END_IF;
END_IF;

Q_Current := Current;
END_PROGRAM
```

Разбор

- Классический «ramp» — сглаживание резких скачков задания.
- Если смена Target большая, Current догоняет его постепенно.

45. Переменная с сохранением после выключения (RETAIN)

Задача

Счётчик произведённой продукции TotalCount должен сохраняться даже при выключении ПЛК.
Используем ключевое слово RETAIN (для поддерживаемых ПЛК).

Программа ST

```
PROGRAM Example45_RetainCounter
VAR RETAIN
  TotalCount : DINT := 0;
```

```
END_VAR

IF I_Pulse AND NOT LastPulse THEN
   TotalCount := TotalCount + 1;
END_IF;

LastPulse := I_Pulse;

Q_TotalCount := TotalCount;
END_PROGRAM
```

Разбор

- Секция VAR RETAIN указывает, что данные должны сохраняться в энергонезависимой памяти.
- Конкретная реализация и ограничения зависят от ПЛК, но идея одинакова.

46. Преобразование типов INT ↔ REAL

Задача

Показать безопасное преобразование типов:

- Целое значение АЦП AI_Raw : INT → напряжение U : REAL (0–10 В).
- Потом округлить U до целых десятых в U_Tenths : INT.

Программа ST

```
PROGRAM Example46_TypeCast
VAR
   AI_Raw    : INT;   (* 0..27648 *)
   U         : REAL;
   U_Tenths  : INT;
END_VAR

U := REAL(AI_Raw) * 10.0 / 27648.0;

(* умножаем на 10 и округляем *)
U_Tenths := INT( U * 10.0 + 0.5 );

Q_U_Volts     := U;
Q_U_TenthsInt  := U_Tenths;
END_PROGRAM
```

Разбор

- REAL(...) и INT(...) — стандартные приведения типов.
- Для округления добавляем 0.5 и отбрасываем дробную часть.

47. Функция «между двумя порогами»

Задача

Реализовать функцию Between, возвращающую TRUE, если X лежит в диапазоне [Min, Max] включительно.

Программа ST

```
FUNCTION Between : BOOL
VAR_INPUT
   X   : REAL;
   Min : REAL;
   Max : REAL;
END_VAR

IF (X >= Min) AND (X <= Max) THEN
   Between := TRUE;
ELSE
   Between := FALSE;
END_IF;
END_FUNCTION
```

Разбор

- Это пример отдельной функции (FUNCTION), а не PROGRAM.
- Очень удобна для читаемости кода: `IF Between(Temp, 20.0, 30.0) THEN ....`

48. Структура «Устройство» и массив устройств

Задача

Создать структурированный тип `Device` с полями:

- `Enabled : BOOL`
- `Fault : BOOL`

- Cmd : BOOL

И массив из 4 устройств. Управлять ими в цикле.

Определение типа и массива

```
TYPE Device :
STRUCT
   Enabled : BOOL;
   Fault   : BOOL;
   Cmd     : BOOL;
END_STRUCT
END_TYPE


PROGRAM Example48_DeviceArray
VAR
   Devices : ARRAY[0..3] OF Device;
   i       : INT;
END_VAR

FOR i := 0 TO 3 DO
   IF Devices[i].Enabled AND NOT Devices[i].Fault THEN
      Q_Devices_Cmd[i] := Devices[i].Cmd;
   ELSE
      Q_Devices_Cmd[i] := FALSE;
   END_IF;
END_FOR;
END_PROGRAM
```

Разбор

- Появляется объектное мышление: каждое устройство — структура.
- В цикле легко пройтись по всем, не копируя один и тот же код 4 раза.

49. Шаблон функционального блока «Пуск двигателя»

Задача

Создать функциональный блок `FB_StartMotor`:

Входы:

- `Start`, `Stop`, `Emergency`

Выход:

- `Q_Motor`

Код FB

```
FUNCTION_BLOCK FB_StartMotor
VAR_INPUT
  Start    : BOOL;
  Stop     : BOOL;
  Emergency: BOOL;
END_VAR
VAR_OUTPUT
  Q_Motor  : BOOL;
END_VAR
VAR
  MotorRun : BOOL := FALSE;
```

```
END_VAR

IF Emergency THEN
   MotorRun := FALSE;
ELSIF Stop THEN
   MotorRun := FALSE;
ELSIF Start THEN
   MotorRun := TRUE;
END_IF;

Q_Motor := MotorRun;
END_FUNCTION_BLOCK
```

Пример использования

```
PROGRAM Example49_UseFB
VAR
   M1, M2 : FB_StartMotor;
END_VAR

M1(Start := I_M1_Start, Stop := I_M1_Stop, Emergency := I_Emergency);
M2(Start := I_M2_Start, Stop := I_M2_Stop, Emergency := I_Emergency);

Q_M1 := M1.Q_Motor;
Q_M2 := M2.Q_Motor;
END_PROGRAM
```

Разбор

- Это обобщение примера 3–4 в виде переиспользуемого блока.
- Один и тот же FB можно вызывать для множества двигателей.

50. Аварийный стоп с запоминанием до квитирования

Задача

Реализовать логику:

- При срабатывании аварийного сигнала `I_Fault` блокируем работу и устанавливаем `FaultLatched = TRUE`.
- Даже если `I_Fault` исчез, блокировка сохраняется, пока оператор не нажмёт `I_ResetFault`.
- Все двигатели `Q_AllMotors` должны быть выключены, пока `FaultLatched = TRUE`.

Программа ST

```
PROGRAM Example50_LatchedFault
VAR
   FaultLatched : BOOL := FALSE;
   LastReset    : BOOL := FALSE;
END_VAR

(* Латч аварии *)
IF I_Fault THEN
   FaultLatched := TRUE;
END_IF;
```

```
(* Сброс по кнопке, если сама авария уже ушла *)
IF I_ResetFault AND NOT LastReset THEN
   IF NOT I_Fault THEN
      FaultLatched := FALSE;
   END_IF;
END_IF;

LastReset := I_ResetFault;

(* Выходы двигателей — только если нет залоченной аварии *)
IF FaultLatched THEN
   Q_Motor1 := FALSE;
   Q_Motor2 := FALSE;
   Q_Motor3 := FALSE;
ELSE
   Q_Motor1 := Cmd_M1;
   Q_Motor2 := Cmd_M2;
   Q_Motor3 := Cmd_M3;
END_IF;
END_PROGRAM
```

Разбор

- `FaultLatched` — «запомненная» авария.
- Сброс возможен только при исчезнувшей физической аварии и фронте `I_ResetFault`.
- Типичный паттерн для безопасной остановки оборудования.

Заключение

В этих 50 примерах вы увидели:

- Базовый синтаксис ST (IF, CASE, циклы, функции, FB).
- Работа с таймерами, счётчиками, аналоговыми сигналами и структурами данных.
- Типичные приёмы: антидребезг, детекторы фронтов, гистерезис, конечные автоматы, приоритеты команд, retain-переменные.

На практике такие фрагменты редко используются «как есть» — но почти в каждом реальном проекте вы найдёте их вариации.
Следующий шаг — взять несколько своих задач из реального объекта и попробовать описать их в стиле этих примеров, постепенно переходя от простых PROGRAM к аккуратной архитектуре из функциональных блоков и структур.

30 задач для самостоятельного выполнения

Ниже 30 условий задач для самостоятельного решения на языке ST для ПЛК по стандарту IEC 61131-3.

Условия задач

1. **Пуск-стоп мотора с самоподхватом**
   Реализуйте логику пуска и останова двигателя двумя кнопками Start и Stop с самоподхватом: после отпускания Start двигатель остаётся включён, пока не

нажали Stop. Добавьте вход Emergency, который всегда немедленно отключает двигатель.

2. **Мигание лампы с настраиваемым периодом**
   Сделайте программу, в которой лампа мигает с периодом, задаваемым переменной BlinkPeriod : TIME. При изменении BlinkPeriod во время работы частота мигания должна плавно перестраиваться без перезапуска ПЛК.

3. **Антидребезг для двух кнопок**
   Напишите блок антидребезга для двух кнопок Btn1 и Btn2, возвращающий отфильтрованные сигналы Btn1Debounced, Btn2Debounced. Время стабилизации в миллисекундах задаётся параметром DebounceTime.

4. **Импульс по фронту сигнала**
   Реализуйте генерацию выходного импульса фиксированной длительности (например, 200 мс) при каждом фронте входа Trigger. Длительность импульса должна задаваться параметром PulseTime.

5. **Счётчик деталей с обнулением по смене**
   Создайте программу, которая считает детали по датчику PartSensor (фронт 0→1). При сигнале ShiftReset счётчик обнуляется. Дополнительно заведите счётчик «всего за сутки», который не обнуляется по смене.

6. **Задержка включения и задержка отключения**
   Реализуйте выход Q_Delayed, который включается через OnDelay после прихода Cmd и выключается через OffDelay после снятия Cmd. Оба параметра должны быть типа TIME и задаваться извне.

7. **Гистерезис по уровню бака**
   По уровню Level в процентах (0–100 %) управляйте насосом так: включаем

при уровне выше HighLevel, выключаем при уровне ниже LowLevel. Значения порогов должны быть настраиваемыми переменными.

8. **Два режима работы насоса: ручной и автоматический**
   Реализуйте переключатель режимов ModeAuto : BOOL. В автоматическом режиме насос управляется по уровню (как в задаче 7), в ручном — напрямую по входу ManualCmd. При аварии Fault насос всегда выключен.

9. **Секундомер наработки оборудования**
   Напишите логику, считающую суммарное время работы двигателя MotorOn в секундах в переменной RunTimeSec. Время должно накапливаться только когда двигатель включён и продолжать считаться после пауз.

10. **Латч аварии с квитированием**
    Реализуйте переменную FaultLatched, которая становится TRUE при любой входной аварии Fault и остаётся TRUE до нажатия ResetFault. Сброс возможен только если Fault = FALSE.

11. **Две скорости двигателя с приоритетом высокой**
    У двигателя есть команды SlowCmd и FastCmd. Нужно включать только одну скорость: при активной FastCmd всегда должна быть выбрана высокая скорость, даже если SlowCmd = TRUE.

12. **Реверс двигателя с запретом одновременного включения**
    Реализуйте управление двигателем вперёд/назад: входы CmdFwd, CmdRev, выходы Q_Fwd, Q_Rev. Исключите ситуацию, когда оба выхода TRUE одновременно, и введите режим «Стоп» по сигналу Stop.

13. **Простой конечный автомат цикла мойки**
    Опишите логику из трёх шагов:

a. Наполнение (выход Fill), 2) Мойка (Wash), 3) Слив (Drain). Каждый шаг длится заданное время, после чего автомат переходит к следующему, затем цикл повторяется.

14. **Фильтрация аналогового сигнала скользящим средним**

    Есть аналоговый вход AI_Raw : INT. Реализуйте фильтр — скользящее среднее по N последним измерениям (N задаётся параметром). Выведите отфильтрованное значение в AI_Filtered.

15. **Контроль «залипания» дискретного датчика**

    Датчик I_Sensor должен периодически менять своё состояние. Если он не меняется более MaxStuckTime : TIME, установить флаг SensorStuck. При первом изменении после этого флаг сбрасывается.

16. **Контроль минимального и максимального значения температуры**

    По входу Temp (REAL) отслеживайте минимальное и максимальное значения за время работы. Реализуйте переменные TempMin и TempMax, которые обновляются при появлении новых экстремумов.

17. **Определение первого сработавшего датчика в массиве**

    Дано: массив Sensors[0..7] : BOOL. Задача: найти индекс первого сработавшего датчика и записать в FirstIndex (или -1, если ни один не активен). Предусмотрите возможность расширения на другой размер массива.

18. **Подсчёт количества активных сигналов в массиве**

    Для того же массива Sensors[0..7] вычислите количество входов со значением TRUE и сохраните результат в ActiveCount. Используйте цикл, а не прямое сложение по каждому элементу.

19. **Простой P-регулятор с ограничением выхода**

    Реализуйте пропорциональный регулятор: Error = Setpoint - Process, Out = Kp * Error. Добавьте ограничение OutMin и OutMax. Все переменные типа REAL, Kp, OutMin и OutMax должны быть настраиваемыми.

20. **Ограничение скорости изменения задания (ramp)**

    Сделайте блок, плавно изменяющий текущее значение Current к целевому Target с максимальным шагом MaxStep за один цикл. Используйте его, чтобы сгладить резкие скачки задания скорости.

21. **Двухручный пуск с временным окном**

    Реализуйте «двухручный пуск»: две кнопки Button1, Button2 должны быть нажаты обе в интервале не более Window : TIME друг от друга. Если условие выполняется — включить Q_Start, иначе держать FALSE.

22. **Переключение активного насоса (lead/lag)**

    Есть два насоса Pump1 и Pump2. При каждом новом старте системы (фронт StartCycle) нужно менять «ведущий» насос: в первый цикл работает Pump1, в следующий — Pump2, далее снова Pump1 и т.д.

23. **Меню выбора уставки с кнопками «вперёд/назад»**

    Реализуйте логику, которая по кнопкам Next и Prev переключает номер активной уставки Index в диапазоне 0..3 по кругу. Индекс должен изменяться только по фронту нажатия кнопки, а не при удержании.

24. **Имитация трафик-лайта для одной дороги**

    Опишите цикл работы светофора: зелёный → жёлтый → красный с заданными временами каждого сигнала. Используйте внутреннюю переменную State (INT) и таймер для перехода между состояниями.

25. **Сердцебиение для SCADA (heartbeat)**

    Создайте бит Heartbeat, который регулярно меняет своё состояние (TRUE/FALSE) раз в заданный период HBPeriod : TIME. Этот бит должен позволять верхнему уровню контролировать «жив ли» контроллер.

26. **Контроль двери шкафа с блокировкой мотора**

    Если дверь шкафа DoorClosed = FALSE, все моторы (M1, M2, M3) должны быть немедленно выключены и оставаться выключенными, пока дверь не закроется. При закрытой двери моторы управляются своими командами.

27. **Логирование времени последнего события**

    Реализуйте переменную LastEventTime : TIME, в которую записывается момент времени при каждом нажатии кнопки EventButton. Используйте стандартную функцию получения системного времени ПЛК (абстрактно).

28. **Режим «местный/дистанционный» для одного агрегата**

    Для агрегата с командами LocalCmd и RemoteCmd реализуйте выбор источника по сигналу RemoteMode. В местном режиме учитывается только LocalCmd, в дистанционном — только RemoteCmd. Добавьте общий аварийный стоп.

29. **Простая цифровая фильтрация по принципу «мёртвой зоны»**

    Есть аналоговый сигнал Value и предыдущее отфильтрованное значение Filtered. Нужно обновлять Filtered только если изменение |Value - Filtered| больше порога Deadband. Используйте тип REAL.

30. **FB «Старт двигателя» как переиспользуемый блок**

    Сформулируйте интерфейс функционального блока FB_StartMotor с входами Start, Stop, Emergency и выходом Q_Motor. В условии пропишите требуемое

поведение (самоподхват, приоритет аварии), а реализацию напишите самостоятельно по аналогии с простыми задачами.
