ТАЙМЕРЫ И СЧЁТЧИКИ НА STRUCTURED TEXT В CODESYS V3.5

50 готовых решений для АСУ ТП

Учебно-практическое пособие

Исправленная и дополненная редакция

Telegram: https://t.me/plcmasters

Электронные учебные пособия:

https://electricalschool.info/e-textbooks.html#!digiseller/articles/156405

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

2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 7

Как работать с пособием 8

Префиксы имен переменных 8

ГЛАВА 1. НИЗКОУРОВНЕВАЯ МЕХАНИКА ВРЕМЕНИ И СЧЕТА В CODESYS V3.5 10

1.1. Модель времени в ПЛК: аппаратные прерывания, тики ОС и джиттер 10

1.2. Системные функции и типы данных времени: TIME, LTIME, TIME() и часы RTC 11

1.3. Проблема переполнения системного таймера (49,7 суток) и безопасная разностная арифметика 12

1.4. Анатомический разбор стандартных блоков библиотеки Standard.library (TON, TOF, TP, CTU, CTUD) 13

ГЛАВА 2. МОДИФИЦИРОВАННЫЕ И НАДЕЖНЫЕ ТАЙМЕРЫ (РЕШЕНИЯ 1–8) 16

РЕШЕНИЕ 1. Паузируемый таймер задержки включения (FB_PausableTON) 16

РЕШЕНИЕ 2. Перезапускаемый таймер с динамической сменой уставки (FB_DynamicTON) 19

РЕШЕНИЕ 3. Энергонезависимый реманентный таймер (FB_RetentiveTON) 21

РЕШЕНИЕ 4. Двухпороговый таймер с предупредительной сигнализацией (FB_DualThresholdTON) 25

РЕШЕНИЕ 5. Программный фильтр подавления дребезга контактов (FB_DebounceFilter) 28

РЕШЕНИЕ 6. Таймер с защитой от кратковременных провалов сигнала (FB_GlitchFilterTON) 30

РЕШЕНИЕ 7. Симметричный таймер задержки включения и отключения (FB_TONOF) 33

РЕШЕНИЕ 8. Сторожевой таймер с автоматическим перезапуском (FB_WatchdogSupervisor) 35

Контрольные вопросы к главам 1 и 2 ..... **Ошибка! Закладка не определена.**

ГЛАВА 3. ИМПУЛЬСНЫЕ ГЕНЕРАТОРЫ, МОДУЛЯТОРЫ И ШИМ (РЕШЕНИЯ 9–16)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 9. Асимметричный мультивибратор (FB_BlinkerAsymmetric) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 10. Программный регулятор широтно-импульсной модуляции (FB_SoftwarePWM)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 11. Одновибратор с перезапуском (FB_RetriggerablePulse) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 12. Пачковый генератор фиксированного числа импульсов (FB_PulseBurst)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 13. Тактовый генератор сетки частот (FB_ClockGrid) .. **Ошибка! Закладка не определена.**

РЕШЕНИЕ 14. Генератор импульсов псевдослучайной длительности (FB_RandomPulseGenerator)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 15. Преобразователь аналогового значения в частоту (FB_AnalogToFrequency)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 16. Генератор периодического строба со сдвигом фазы (FB_SyncPulseMaster) ..... **Ошибка! Закладка не определена.**

ГЛАВА 4. ПРОМЫШЛЕННЫЕ СЧЕТЧИКИ, ДЕТЕКТОРЫ И ФАЗОВЫЙ УЧЕТ (РЕШЕНИЯ 17–24) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 17. 64-битный реверсивный промышленный счетчик (FB_CounterLINT)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 18. Кольцевой счетчик-распределитель шагов (FB_RingCounter) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 19. Программный квадратурный декодер энкодера (FB_QuadratureDecoder) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 20. Селективный счетчик по длительности импульса (FB_PulseLengthFilterCounter) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 21. Пакетный дозирующий счетчик (FB_BatchCounter)**Ошибка! Закладка не определена.**

РЕШЕНИЕ 22. Счетчик изделий с адаптивной компенсацией двойного счета (FB_DoubleDetectionBlocker)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 23. Реверсивный счетчик заполнения накопителя (FB_OccupancyTracker) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 24. Тахометрический счетчик скорости вращения (FB_PulseRateSpeedEstimator) ..... **Ошибка! Закладка не определена.**

Контрольные вопросы к главам 3 и 4 .... **Ошибка! Закладка не определена.**

ГЛАВА 5. ХРОНОМЕТРИЯ ОБОРУДОВАНИЯ, МОТОЧАСЫ И УЧЕТ РЕСУРСА (РЕШЕНИЯ 25–32)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 25. Счетчик наработки моточасов с каскадным накоплением (FB_HourCounter)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 26. Двухканальный счетчик циклов и коммутационного ресурса (FB_SwitchCycleCounter) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 27. Сервисный таймер межсервисного интервала (FB_MaintenanceDueTimer)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 28. Трехзонный хронометр баланса времени для OEE (FB_OeeTimeTracker)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 29. Интегральный тепловой таймер затяжного пуска (I^2t) (FB_MotorOverloadIntegralTimer)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 30. Адаптивный таймер эквивалентной термической выдержки (FB_CumulativeThermalDwell)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 31. Метрологический интервальный таймер калибровки датчиков (FB_CalibrationIntervalManager)**Ошибка! Закладка не определена.**

РЕШЕНИЕ 32. Многотарифный хронометр учета времени суток (FB_TimeOfUseTracker)..... **Ошибка! Закладка не определена.**

ГЛАВА 6. СЛОЖНЫЕ ЦИКЛОВЫЕ ТАЙМЕРЫ И МНОГОСТАДИЙНЫЕ ИНТЕРВАЛЫ (РЕШЕНИЯ 33–41)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 33. Программный кулачковый контроллер времени (FB_ElectronicCamTimer) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 34. Рецептурный многостадийный профилер времени (FB_MultiStageRampTimer)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 35. Волновой таймер группового пуска агрегатов (FB_StaggeredStartTimer) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 36. Таймер контроля технологического окна (FB_WindowComparatorTimer) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 37. Циклический контроллер импульсной регенерации рукавных фильтров (FB_FilterPulseController)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 38. Автомат светофорного регулирования технологического переезда (FB_TrafficLightController)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 39. Программный таймер предварительной продувки газовой горелки (FB_BurnerPurgeTimer)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 40. Двухфакторный таймер регенерации фильтров водоподготовки (FB_WaterSoftenerRegenTimer)**Ошибка! Закладка не определена.**

РЕШЕНИЕ 41. Временной супервизор скорости потока и замятия конвейера (FB_ConveyorJamDetector)..... **Ошибка! Закладка не определена.**

Контрольные вопросы к главам 5 и 6 .... **Ошибка! Закладка не определена.**

ГЛАВА 7. ВЫСОКОТОЧНЫЕ, СОБЫТИЙНЫЕ И СЕТЕВЫЕ ИЗМЕРЕНИЯ (РЕШЕНИЯ 42–50)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 42. Прецизионный микросекундный секундомер (FB_MicrosecondStopwatch)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 43. Измеритель времени механического отклика приводов (FB_ActuatorResponseBenchmark)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 44. Скользящий счетчик производительности за минутное окно (FB_SlidingWindowThroughput)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 45. Метрологический интерполятор расхода по импульсам (FB_FlowMeterPulseTotalizer)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 46. Хронологический фиксатор аварий с меткой времени RTC (FB_RtcEventLogger) ..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 47. Программный супервизор сетевого протокола связи (FB_NetworkHeartbeatSupervisor)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 48. Измеритель фазового сдвига и рассинхронизации валов (FB_PhaseShiftDetector)..... **Ошибка! Закладка не определена.**

РЕШЕНИЕ 49. Адаптивный таймер задержки с компенсацией скорости конвейера (FB_SpeedCompensatedDelay)**Ошибка! Закладка не определена.**

РЕШЕНИЕ 50. Универсальный комбинированный метрологический модуль учета ресурса (Master Metric Core)..... **Ошибка! Закладка не определена.**

Контрольные вопросы к главе 7..... **Ошибка! Закладка не определена.**

ЗАКЛЮЧЕНИЕ ..... **Ошибка! Закладка не определена.**

ПРИЛОЖЕНИЯ..... **Ошибка! Закладка не определена.**

ПРИЛОЖЕНИЕ А. ТАБЛИЦА ТИПОВ ДАННЫХ ВРЕМЕНИ В IEC 61131-3..... **Ошибка! Закладка не определена.**

ПРИЛОЖЕНИЕ Б. БЕЗОПАСНАЯ РАЗНОСТНАЯ АРИФМЕТИКА ПРИ ПЕРЕХОДЕ ЧЕРЕЗ 49,7 СУТОК..... **Ошибка! Закладка не определена.**

ПРИЛОЖЕНИЕ В. АЛФАВИТНЫЙ УКАЗАТЕЛЬ 50 ФУНКЦИОНАЛЬНЫХ БЛОКОВ..... **Ошибка! Закладка не определена.**

ВВЕДЕНИЕ

Дискретные технологические процессы в современной промышленной автоматизации непрерывно опираются на две базовые физические категории: время протекания процессов и количество свершившихся дискретных событий. Включение технологических насосов с задержкой для предотвращения гидроударов, генерация импульсов широтно-импульсной модуляции для управления твердотельными реле нагревателей, фильтрация дребезга механических контактов, подсчет упаковок готовой продукции на высокоскоростных конвейерах и учет моточасов для систем предиктивного обслуживания (Predictive Maintenance) — все эти задачи требуют надежного программного аппарата.

Стандарт IEC 61131-3 предоставляет инженеру минимальный базовый набор функциональных блоков времени и счета, входящих в стандартную библиотеку `Standard.library`: таймеры `TON`, `TOF`, `TP` и счетчики `CTU`, `CTD`, `CTUD`. Однако на реальном промышленном объекте стандартная реализация этих блоков быстро вскрывает существенные ограничения:

- Стандартный таймер `TON` мгновенно сбрасывает накопленный интервал времени при кратковременном прерывании разрешающего сигнала, что делает его неприменимым в циклических процессах с паузами.
- Стандартные таймеры не сохраняют состояние в энергонезависимой памяти (`RETAIN` / `PERSISTENT`), приводя к обнулению технологических выдержек при просадках напряжения питания шкафа управления.
- Вызов таймера внутри условных операторов `IF-THEN` или веток `CASE` без строгого понимания механизма фиксации фронтов приводит к зависаниям алгоритмов и пропуску шагов.
- Стандартные счетчики имеют ограниченную 16-битную разрядность (по стандарту IEC 61131-3 — `INT`, до 32 767; в библиотеке Standard.library CODESYS — `WORD`, до 65 535), которая на скоростных линиях фасовки и штамповки исчерпывается за десятки минут – единицы часов работы.

Настоящее учебное пособие призвано связать теорию стандарта IEC 61131-3 с практическими требованиями к прикладному программному обеспечению ПЛК. Пособие содержит 50 учебных инженерных решений на языке Structured Text (ST) для среды CODESYS V3.5. Код написан на подмножестве ST, общем для сред на базе CODESYS (в том числе EcoStruxure Machine Expert); совместимость с конкретной версией среды, с TwinCAT 3 и с библиотеками производителя ПЛК требует проверки при переносе проекта.

Каждое решение построено по единой схеме: технологическое назначение и постановка задачи → спецификация интерфейса → программная реализация на ST → разбор алгоритмической логики и (при необходимости) замечания по применению и ограничениям.

Как работать с пособием

1. Изучите главу 1: модель времени ПЛК, типы `TIME`/`LTIME` и разностную арифметику. На этих приемах построены почти все решения.
2. Для каждого решения создайте в проекте CODESYS объект POU типа «Функциональный блок» (язык ST). Раздел объявлений (`VAR_INPUT` … `END_VAR`) вводится в верхнюю часть редактора, тело — в нижнюю. В редакторе CODESYS строка `END_FUNCTION_BLOCK` не пишется — это нормально; она нужна только при экспорте в текстовый формат PLCopen/IEC.
3. Вызывайте экземпляр блока в программе `PLC_PRG` безусловно — один раз за цикл задачи. Вызов внутри `IF` или ветки `CASE` нарушает расчет приращений времени и обработку фронтов.
4. Проверяйте блок в режиме симуляции (Online → Simulation) или на учебном стенде: задавайте входы через окно мониторинга, наблюдайте выходы, используйте трассировку (Trace) для временных диаграмм.
5. Ответьте на контрольные вопросы в конце глав и выполните предложенные доработки.

Префиксы имен переменных

В пособии используется венгерская нотация, распространенная в проектах CODESYS: `x` — `BOOL`, `t` — `TIME`, `lt` — `LTIME`, `i`/`ui` — `INT`/`UINT`, `di`/`udi` — `DINT`/`UDINT`, `li`/`uli` — `LINT`/`ULINT`, `usi` — `USINT`, `r` — `REAL`, `dt` — `DATE_AND_TIME`, `tod` — `TIME_OF_DAY`, `a` — массив, `e` — перечисление, `fb` — экземпляр функционального блока, `ST_` — структура.

> Важно. Решения пособия носят учебный характер. Функции безопасности (продувка топки и розжиг горелки, защита двигателя от перегрузки, аварийные остановы, блокировки ограждений) на реальных объектах выполняются сертифицированными средствами (контроллеры и реле безопасности, автоматы горения, устройства защиты двигателей) в соответствии с действующими ТНПА. Программные блоки пособия их не заменяют и могут выполнять только вспомогательные, диагностические и информационные функции. Перед применением на объекте любой блок

> необходимо адаптировать, протестировать и согласовать с проектной документацией.

ГЛАВА 1. НИЗКОУРОВНЕВАЯ МЕХАНИКА ВРЕМЕНИ И СЧЕТА В CODESYS V3.5

1.1. Модель времени в ПЛК: аппаратные прерывания, тики ОС и джиттер

Для грамотного использования таймеров на языке ST разработчик обязан понимать, как формируется временная шкала внутри процессорного модуля программируемого логического контроллера. В отличие от персональных компьютеров общего назначения под управлением ОС Windows или Linux, промышленный контроллер функционирует под управлением операционной системы жесткого реального времени (RTOS: VxWorks, QNX, Linux RT Preempt или специализированные микроядра).

В основе измерения времени лежит аппаратный таймер микропроцессора (Hardware Timer / PIT / RTC), который генерирует периодические прерывания высокой частоты (High Resolution Ticks). Планировщик задач (Task Scheduler) CODESYS v3.5 использует эти аппаратные тики для отсчета базового системного времени и управления запуском пользовательских задач (Tasks).

Когда в конфигурации задач (Task Configuration) инженером задается циклическая задача с периодом T_{task}=10 мс, планировщик настраивает аппаратный таймер на генерацию прерывания каждые 10 миллисекунд. Процесс выполнения задачи состоит из следующих стадий:

1. Аппаратное прерывание таймера переводит задачу в состояние готовности (Ready).
2. Диспетчер задач сохраняет контекст текущей низкоприоритетной задачи и передает управление высокоприоритетной циклической задаче (Context Switch). Время между прерыванием и фактическим началом выполнения первой инструкции пользовательского кода называется джиттером (Jitter — фазовое дрожание). Величина джиттера зависит от аппаратной платформы, ОС и загрузки процессора; ее фактическое значение для своего контроллера можно увидеть на вкладке мониторинга конфигурации задач CODESYS.
3. Происходит считывание физических дискретных и аналоговых входов в область образов входов `%I`.
4. Выполняется прикладной код пользователя на языке ST от первой до последней инструкции.
5. Вычисленные значения записываются в область образов выходов `%Q`.

Из этой механики вытекает фундаментальное правило промышленной автоматизации: **дискретность любого программного таймера в ПЛК**

ограничена периодом скана задачи, в которой вызывается данный таймер. Если задача вызывается с периодом 20 мс, программный таймер с уставкой 5 мс физически не сможет зафиксировать интервал в 5 мс: он сработает только на следующем скане задачи, то есть через 20 мс. Программное измерение интервалов, меньших периода задачи, требует применения аппаратных модулей счета (Fast I/O) и прерываний по событиям (External Event Tasks).

1.2. Системные функции и типы данных времени: TIME, LTIME, TIME() и часы RTC

Стандарт IEC 61131-3 и среда CODESYS v3.5 определяют несколько ключевых типов данных для представления времени:

1. **Тип TIME:** 32-битное значение (внутренне обрабатывается как `DWORD`/`UDINT`), выражающее относительное время в миллисекундах. Диапазон значений — от `T#0ms` до `T#49d17h2m47s295ms` (49 суток 17 часов 2 минуты 47 секунд 295 миллисекунд). Литералы времени записываются в формате `T#10d4h3m20s50ms` или `TIME#1h30m`.
2. **Тип LTIME (время высокого разрешения):** 64-битное значение (внутренне обрабатывается как `LWORD`/`ULINT`) с разрешением 1 нс. Диапазон — от `LTIME#0ns` до `LTIME#213503d23h34m33s709ms551us615ns` (≈ 584 года). Литералы: `LTIME#1000us`, `LTIME#2h15m30s500us200ns`.
3. **Типы календарного времени (DATE, TIME_OF_DAY / TOD, DATE_AND_TIME / DT):** используются для привязки к суточному и календарному времени. В CODESYS `DATE` и `DT` хранят 32-битное число секунд от 01.01.1970 00:00:00 (предельная дата — 07.02.2106), а `TOD` — число миллисекунд от начала суток.

Оператор `TIME()` возвращает время в миллисекундах, прошедшее с момента запуска системы, в формате `TIME`. Аргументов не требует:

```
tCurrentTimestamp := TIME();
```

Для измерений высокого разрешения в CODESYS рекомендуется оператор `LTIME()`: он возвращает монотонно возрастающее время в наносекундах (тип `LTIME`). Прежний способ — функция `SysTimeGetUs` библиотеки SysTimeCore: она возвращает код ошибки (`RTS_IEC_RESULT`), а значение микросекунд записывает в параметр `pUsTime` типа `SYSTIME`:

```
ltNow   := LTIME();                                    // ltNow
: LTIME, наносекунды
eResult := SysTimeGetUs(pUsTime := ulSysMicros);  //
ulSysMicros : SYSTIME (ULINT), мкс
```

Для чтения часов реального времени (RTC) используются функции библиотеки SysTimeRtc (например, `SysTimeRtcHighResGet` — миллисекунды от 01.01.1970 по UTC) или библиотеки производителя ПЛК. Перечень доступных функций зависит от платформы и требует уточнения по документации контроллера.

1.3. Проблема переполнения системного таймера (49,7 суток) и безопасная разностная арифметика

Поскольку тип `TIME` представляет собой 32-битное число миллисекунд, максимальное значение, которое он может принять:

$$2^{32} - 1 = 4\,294\,967\,295 \text{ мс} \approx 49{,}71 \text{ суток}$$

В момент достижения этого предела значение системного таймера сбрасывается в `0` и отсчет начинается заново. Если программист использует некорректную логику сравнения меток времени, примерно через 49,7 суток непрерывной работы возможны зависание логики или ложное срабатывание.

Рассмотрим ошибочный алгоритм вычисления задержки:

```
// КРАЙНЕ ОПАСНО: Ошибочная проверка уставки через прямое
сложение
IF xStartTrigger THEN
    tDeadline := TIME() + T#10S; // Фиксация момента
окончания
END_IF;

IF TIME() >= tDeadline THEN
    xOutput := TRUE; // Логика зависнет при переходе через
49.7 суток!
END_IF;
```

Если контроллер непрерывно проработал 49 суток 17 часов 2 минуты 40 секунд, значение `TIME()` близко к максимуму. Сложение `TIME() + T#10S` переполнит 32-битное беззнаковое число и вернет маленькое значение (около `T#3S`). Условие `TIME() >= tDeadline` станет истинным немедленно, на следующем скане, — выход сработает без выдержки. При другом соотношении моментов (переполняется уже сам `TIME()`, а метка `tDeadline` осталась большой) условие, наоборот, может не выполняться почти 49,7 суток.

Надежным методом является **разностная арифметика по модулю** 2^{32}. В языке ST вычитание двух меток времени вычисляется корректно даже при переполнении счетчика:

```
// БЕЗОПАСНО (для интервалов короче 49,7 суток): проверка
через разность текущего и стартового времени
IF xStartTrigger AND NOT xRunning THEN
    tStart := TIME(); // Фиксируем точку старта
    xRunning := TRUE;
END_IF;

// Разность (TIME() - tStart) на беззнаковом уровне
компилятора вычисляется
// корректно даже в момент перехода через ноль:
IF xRunning AND ((TIME() - tStart) >= tTargetDelay) THEN
    xOutput := TRUE;
    xRunning := FALSE;
END_IF;
```

Благодаря свойствам беззнакового вычитания по модулю 2^{32} разность `(TIME() - tStart)` равна фактически прошедшему времени и при переходе счетчика через ноль.

> Ограничения метода: 1) измеряемый интервал должен быть меньше 2^{32} мс (≈ 49,7 суток) — для более длинных интервалов используют `LTIME()` или накопление в целочисленных счетчиках секунд и часов (решения 25, 27); 2) после перезапуска ПЛК отсчет `TIME()` начинается заново, поэтому сохраненные до отключения метки времени недействительны (см. решение 3).

1.4. Анатомический разбор стандартных блоков библиотеки Standard.library (TON, TOF, TP, CTU, CTUD)

Для осознанного использования стандартных таймеров разберем их внутреннее программное устройство.

Таймер задержки включения (TON)

Функциональный блок `TON` отслеживает переход входного дискретного сигнала `IN` из `FALSE` в `TRUE`. Упрощенная учебная модель его работы (это не исходный код библиотеки) выглядит так:

```
FUNCTION_BLOCK TON_Internal_Explanation
VAR_INPUT
    IN : BOOL; // Разрешающий сигнал
    PT : TIME; // Уставка времени задержки (Preset Time)
```

```
END_VAR
VAR_OUTPUT
    Q  : BOOL; // Выход срабатывания
    ET : TIME; // Текущее прошедшее время (Elapsed Time)
END_VAR
VAR
    tStartTime : TIME;
    xPrevIN    : BOOL;
END_VAR

// Фиксация переднего фронта сигнала IN
IF IN AND NOT xPrevIN THEN
    tStartTime := TIME();
    ET := T#0S;
    Q := FALSE;
ELSIF IN THEN
    // Контроль накопления времени
    IF (TIME() - tStartTime) >= PT THEN
        ET := PT;
        Q := TRUE;
    ELSE
        ET := TIME() - tStartTime;
        Q := FALSE;
    END_IF;
ELSE
    // Мгновенный сброс при снятии сигнала IN
    ET := T#0S;
    Q := FALSE;
END_IF;

xPrevIN := IN; // Сохранение предыстории входа
```

Отсюда очевидна анатомическая уязвимость стандартного TON: как только IN становится равным FALSE, все внутренние переменные ET и Q немедленно обнуляются. Таймер не может запомнить, что он уже отсчитал 8 из 10 секунд, если вход кратковременно пропал на один цикл задачи.

Таймер задержки выключения (TOF)

Таймер TOF удерживает выход Q во включенном состоянии при снятии сигнала IN еще в течение времени PT. Отсчет начинается по заднему фронту (IN перешел из TRUE в FALSE).

Импульсный таймер (TP — Monostable Pulse)

Таймер `TP` генерирует импульс фиксированной длительности `PT` по переднему фронту `IN`. Ключевая особенность: состояние входа `IN` во время отсчета импульса полностью игнорируется. Длительность выходного сигнала `Q` невозможно прервать или продлить стандартными средствами.

Счетчики CTU, CTD, CTUD

Счетчики стандарта IEC работают по передним фронтам сигналов счета:

- `CTU` (Count Up): наращивает переменную `CV` (Current Value) на единицу по каждому фронту `CU`. Выход `Q := (CV >= PV)`.
- `CTD` (Count Down): уменьшает `CV` по фронту `CD` до достижения нуля.
- `CTUD` (Count Up/Down): реверсивный счетчик с раздельными входами счета вверх `CU` и вниз `CD`.

Главное ограничение стандартных счетчиков — 16-битная разрядность `CV` и `PV`. Стандарт IEC 61131-3 задает для них тип `INT` (до 32 767), а в библиотеке Standard.library CODESYS используется тип `WORD` (0…65 535). Поведение счетчика на границе диапазона зависит от реализации и должно проверяться по документации используемой библиотеки. Для больших значений в CODESYS есть библиотека Standard64 (блоки `LCTU`, `LCTD`, `LCTUD` и 64-битные таймеры `LTON`, `LTOF`, `LTP`); собственный 64-битный счетчик рассмотрен в решении 17.

ГЛАВА 2. МОДИФИЦИРОВАННЫЕ И НАДЕЖНЫЕ ТАЙМЕРЫ (РЕШЕНИЯ 1–8)

РЕШЕНИЕ 1. Паузируемый таймер задержки включения (FB_PausableTON)

Технологическое назначение и постановка задачи

В реальных циклических процессах (термообработка, перемешивание в реакторах, вакуумная сушка) технологический цикл периодически требуется ставить на паузу по внешним причинам (открытие защитного ограждения оператором, ожидание набора температуры, временная просадка давления). Стандартный блок `TON` в такой ситуации применять нельзя: при отключении входа `IN` он полностью сбрасывает накопленный интервал, заставляя установку начинать процесс заново.

Блок `FB_PausableTON` сохраняет накопленное прошедшее время при взводе входа `xPause`. При снятии сигнала `xPause` таймер продолжает отсчет интервала именно с той миллисекунды, на которой был приостановлен.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xIN	BOOL	Общий разрешающий сигнал работы таймера
	xPause	BOOL	Сигнал технологической паузы (заморозка счета)
	xReset	BOOL	Асинхронный безусловный сброс таймера
	tPT	TIME	Целевая уставка времени (Preset Time)
VAR_OUTPUT	xQ	BOOL	Выход срабатывания (время истекло)
	tET	TIME	Фактическое накопленное время (Elapsed Time)
	tRemaining	TIME	Оставшееся время до срабатывания выхода

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_PausableTON
VAR_INPUT
    xIN         : BOOL; // Команда включения
технологического процесса
```

```
    xPause      : BOOL; // Флаг заморозки таймера (Hold / Pause)
    xReset      : BOOL; // Асинхронный сброс таймера в исходное состояние
    tPT         : TIME; // Заданная уставка выдержки времени
END_VAR
VAR_OUTPUT
    xQ          : BOOL; // Сигнал истечения времени задержки
    tET         : TIME; // Накопленное эффективное время работы
    tRemaining  : TIME; // Оставшееся время до окончания технологической фазы
END_VAR
VAR
    xPrevIN     : BOOL; // История состояния входа xIN для детектора фронта
    xPrevPause  : BOOL; // История входа паузы
    tLastTime   : TIME; // Метка системного времени на предыдущем цикле вызова
    tAccumTime  : TIME; // Накопленная база времени в миллисекундах
END_VAR

// 1. АСИНХРОННЫЙ БЕЗУСЛОВНЫЙ СБРОС
IF xReset OR NOT xIN THEN
    tAccumTime  := T#0MS;
    tET         := T#0MS;
    tRemaining  := tPT;
    xQ          := FALSE;
    xPrevIN     := xIN;
    xPrevPause  := xPause;
    tLastTime   := TIME();
    RETURN;
END_IF;

// 2. ОБРАБОТКА ПЕРВИЧНОГО ПУСКА (ПЕРЕДНИЙ ФРОНТ СИГНАЛА xIN)
IF xIN AND NOT xPrevIN THEN
    tAccumTime  := T#0MS;
    tLastTime   := TIME();
    xQ          := FALSE;
END_IF;
```

```
// 3. РАСЧЕТ И ИНТЕГРИРОВАНИЕ ВРЕМЕНИ С УЧЕТОМ СТАТУСА
ПАУЗЫ
// Если таймер включен и сигнал паузы НЕ активен —
выполняем накопление дельты времени
IF xIN AND NOT xPause AND NOT xQ THEN
    // Безопасное инкрементирование дельты времени между
вызовами задачи ПЛК:
    tAccumTime := tAccumTime + (TIME() - tLastTime);
END_IF;

// Системная метка времени обновляется всегда для
корректного вычисления разности
tLastTime := TIME();

// 4. ОГРАНИЧЕНИЕ И СРАВНЕНИЕ С ЦЕЛЕВОЙ УСТАВКОЙ
IF tAccumTime >= tPT THEN
    tET        := tPT;
    tRemaining := T#0MS;
    xQ         := TRUE;
ELSE
    tET        := tAccumTime;
    tRemaining := tPT - tAccumTime;
    xQ         := FALSE;
END_IF;

// Сохранение предыстории для детекторов
xPrevIN    := xIN;
xPrevPause := xPause;
```

Разбор алгоритмической логики

- Принудительный сброс выходов при пропадании сигнала `xIN` или взводе `xReset`. При этом выход `tRemaining` явно инициализируется полной уставкой `tPT`.
- Математическая основа паузируемого счета: `tAccumTime := tAccumTime + (TIME() - tLastTime)`. Вместо прямой привязки к абсолютной точке старта блок накапливает дифференциальные приращения времени Δt между последовательными вызовами функционального блока.
- Инструкция `tLastTime := TIME()` выполняется вне условий, что гарантирует: в момент снятия сигнала `xPause` значение Δt будет вычислено строго за один рабочий цикл контроллера, исключая скачкообразное прибавление времени, проведенного в паузе.

РЕШЕНИЕ 2. Перезапускаемый таймер с динамической сменой уставки (FB_DynamicTON)

Технологическое назначение и постановка задачи

В системах автоматического дозирования, упаковочных автоматах и термокамерах величина задержки `tPT` часто корректируется алгоритмами верхнего уровня или оператором с панели HMI прямо в процессе отсчета (например, оператор уменьшил время запайки пакета с 4.0 до 2.5 секунд).

Поведение стандартного `TON` при изменении `PT` во время отсчета в стандарте явно не оговорено и может различаться в разных реализациях (в CODESYS `ET` сравнивается с текущим значением `PT` на каждом вызове). Кроме того, стандартный `TON` нельзя перезапустить без снятия `IN`, и он не формирует строб окончания выдержки. Блок `FB_DynamicTON` задает эти функции явно: пересчитывает условие срабатывания на каждом скане, допускает перезапуск командой `xRestart` и выдает однократный строб `xTargetReached`.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xIN	BOOL	Разрешающий сигнал отсчета
	tPT	TIME	Динамическая уставка времени
	xRestart	BOOL	Команда принудительного перезапуска счета без снятия xIN
VAR_OUTPUT	xQ	BOOL	Выход срабатывания
	tET	TIME	Текущее прошедшее время
	xTargetReached	BOOL	Однократный импульсный флаг достижения уставки

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_DynamicTON
VAR_INPUT
    xIN             : BOOL; // Входной сигнал управления
    tPT             : TIME; // Динамическая переменная
уставки времени
    xRestart        : BOOL; // Импульс немедленного сброса
и перезапуска отсчета
```

```
END_VAR
VAR_OUTPUT
    xQ               : BOOL; // Статический флаг
срабатывания таймера
    tET              : TIME; // Фактическое накопленное
время
    xTargetReached  : BOOL; // Импульсный сигнал (строб на
1 скан) при достижении уставки
END_VAR
VAR
    xPrevIN          : BOOL; // Флаг фиксации фронта входа
xIN
    xPrevRestart     : BOOL; // Флаг фиксации фронта входа
xRestart
    xPrevQ           : BOOL; // История выхода xQ для
генерации импульса xTargetReached
    tStartTimestamp : TIME; // Метка времени старта отсчета
END_VAR

// 1. АНАЛИЗ ФРОНТОВ ВКЛЮЧЕНИЯ И ПЕРЕЗАПУСКА
IF (xIN AND NOT xPrevIN) OR (xIN AND xRestart AND NOT
xPrevRestart) THEN
    tStartTimestamp := TIME(); // Перезапуск отсчета
времени
    xQ               := FALSE;
    tET              := T#0MS;
END_IF;

// 2. ВЫЧИСЛЕНИЕ ПРОШЕДШЕГО ВРЕМЕНИ И ДИНАМИЧЕСКИЙ КОНТРОЛЬ
УСТАВКИ
IF xIN THEN
    // Время вычисляется только до срабатывания: после него
выход фиксируется,
    // поэтому переполнение TIME() при длительном удержании
xIN не сбросит xQ
    IF NOT xQ THEN
        tET := TIME() - tStartTimestamp; // Безопасный
разностный метод
    END_IF;

    // Динамическая верификация: если текущая уставка tPT
уменьшилась
    // и стала меньше или равна накопленному времени —
немедленно формируем выход
```

```
    IF xQ OR (tET >= tPT) THEN
        xQ  := TRUE;
        tET := tPT; // Ограничиваем отображаемое время
значением текущей уставки
    END_IF;
ELSE
    // При отключении разрешающего сигнала — полный сброс
    tET := T#0MS;
    xQ  := FALSE;
END_IF;

// 3. ГЕНЕРАЦИЯ ОДНОКРАТНОГО СТРОБА ЗАВЕРШЕНИЯ (R_TRIG ДЛЯ
ВЫХОДА Q)
xTargetReached := xQ AND NOT xPrevQ;

// 4. ОБНОВЛЕНИЕ ПЕРЕМЕННЫХ ИСТОРИИ
xPrevIN      := xIN;
xPrevRestart := xRestart;
xPrevQ       := xQ;
```

Разбор алгоритмической логики

- Условие объединяет классический фронт `xIN` и команду принудительного перезапуска `xRestart`. Это позволяет обнулить таймер без отключения технологического механизма.
- Проверка `IF tET >= tPT` выполняется непрерывно на каждом скане. Если таймер отсчитал 3 секунды при уставке 5 секунд, и уставка с HMI динамически изменилась на 2.5 секунды, блок на том же самом скане выставит `xQ := TRUE`.
- Встроенный детектор импульса формирует `xTargetReached := TRUE` ровно на один рабочий цикл ПЛК в момент срабатывания, исключая необходимость внешнего вызова функционального блока `R_TRIG`.
- После срабатывания выход `xQ` фиксируется до снятия `xIN` или команды `xRestart`. Поэтому увеличение уставки после срабатывания уже не сбрасывает выход, а удержание `xIN` дольше 49,7 суток не приводит к ложному сбросу `xQ` из-за переполнения `TIME()`.

РЕШЕНИЕ 3. Энергонезависимый реманентный таймер (FB_RetentiveTON)

Технологическое назначение и постановка задачи

На технологических установках с длительными производственными циклами (автоклавы, печи карбонизации, системы биологической очистки, где

технологические фазы длятся от нескольких часов до десятков суток) кратковременное исчезновение электропитания шкафа управления не должно приводить к аннулированию технологического процесса.

Стандартные таймеры хранят переменные во временной оперативной памяти (SRAM), которая обнуляется при выключении ПЛК. Блок `FB_RetentiveTON` спроектирован для размещения в реманентной энергонезависимой памяти (`RETAIN` / `PERSISTENT`), восстанавливает накопленный интервал после рестарта контроллера и продолжает технологический процесс.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xIN	BOOL	Разрешение технологического процесса
	xClear	BOOL	Полный сброс энергонезависимого времени (команда новой партии)
	tPT	TIME	Заданное время технологического цикла
VAR_OUTPUT	xQ	BOOL	Флаг завершения выдержки
	tET	TIME	Накопленное энергонезависимое время процесса
	udiPowerFailCount	UDINT	Счетчик количества пропаданий питания за цикл

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_RetentiveTON
VAR_INPUT
    xIN                 : BOOL;  // Команда
технологического процесса
    xClear              : BOOL;  // Принудительный сброс
накопленного времени (сброс партии)
    tPT                 : TIME;  // Рецептурная уставка
времени цикла
END_VAR
VAR_OUTPUT
    xQ                  : BOOL;  // Выход завершения
технологической выдержки
```

```
    tET                 : TIME;  // Текущее сохраненное
время
    udiPowerFailCount   : UDINT; // Диагностический счетчик
зафиксированных рестартов ПЛК
END_VAR
VAR RETAIN
    tStoredET           : TIME;  // Накопленное время,
сохраняемое при отключении питания
    udiCyclesInterrupted: UDINT; // Сохраняемый счетчик
сбоев
END_VAR
VAR
    xFirstScanDone      : BOOL;  // Системный маркер
первого скана контроллера
    tLastSysTime        : TIME;  // Метка времени
предыдущего скана задачи
END_VAR

// 1. ОБРАБОТКА ХОЛОДНОГО / ГОРЯЧЕГО РЕСТАРТА ПЛК (ПЕРВЫЙ
ЦИКЛ ПОСЛЕ ВКЛЮЧЕНИЯ)
IF NOT xFirstScanDone THEN
    tLastSysTime        := TIME();
    xFirstScanDone      := TRUE;
    // Если питание восстановилось при активном сигнале
процесса — фиксируем факт сбоя
    IF xIN AND (tStoredET > T#0MS) AND (tStoredET < tPT)
THEN
        udiCyclesInterrupted := udiCyclesInterrupted + 1;
    END_IF;
END_IF;

// 2. БЕЗУСЛОВНЫЙ СБРОС ЭНЕРГОНЕЗАВИСИМЫХ ДАННЫХ
IF xClear THEN
    tStoredET            := T#0MS;
    tET                  := T#0MS;
    udiCyclesInterrupted := 0;
    xQ                   := FALSE;
    tLastSysTime         := TIME();
    RETURN;
END_IF;

// 3. ИНТЕГРИРОВАНИЕ ВРЕМЕНИ В ЭНЕРГОНЕЗАВИСИМУЮ ПЕРЕМЕННУЮ
IF xIN AND NOT xQ THEN
```

```
    // Накапливаем фактическое время работы между сканами
контроллера
    tStoredET := tStoredET + (TIME() - tLastSysTime);
END_IF;

// Системная временная метка обновляется непрерывно
tLastSysTime := TIME();

// 4. ФОРМИРОВАНИЕ ВЫХОДОВ
IF tStoredET >= tPT THEN
    tET := tPT;
    xQ  := TRUE;
ELSE
    tET := tStoredET;
    xQ  := FALSE;
END_IF;

// Передача диагностической информации
udiPowerFailCount := udiCyclesInterrupted;
```

Разбор алгоритмической логики

- **Секция VAR RETAIN:** переменные `tStoredET` и `udiCyclesInterrupted` размещаются в энергонезависимой памяти контроллера (NVRAM, MRAM, FRAM или Flash с буферным питанием). Сохранность данных при пропадании питания 24 В обеспечивается, только если платформа поддерживает реманентную память: тип и объем памяти уточняются по документации ПЛК.
- Если локальная переменная ФБ объявлена как `VAR RETAIN`, CODESYS размещает в реманентной области весь экземпляр блока, но реманентными считаются только объявленные `RETAIN`-переменные. Поэтому после рестарта `xFirstScanDone` снова равен `FALSE`, и первый цикл обрабатывается корректно. Это существенно: после перезапуска `TIME()` начинает отсчет заново, и старая метка `tLastSysTime` недействительна — ее обязательно нужно переинициализировать.
- Переменные `RETAIN` сохраняются при отключении питания и «теплом» сбросе, но теряются при «холодном» сбросе, сбросе в исходное состояние и, как правило, при загрузке измененного приложения. Если значения должны пережить загрузку новой версии программы, используют `PERSISTENT`-переменные (список Persistent Variables).

- Блок первого цикла инициализации `IF NOT xFirstScanDone THEN`. Выравнивает базовое системное время `tLastSysTime` и регистрирует факт перезапуска контроллера во время активного процесса.
- Приращение `tStoredET := tStoredET + (TIME() - tLastSysTime)` суммирует интервалы только тогда, когда разрешающий сигнал `xIN` активен. При выключении контроллера суммирование останавливается; после включения отсчет продолжается с сохраненной контрольной точки.

РЕШЕНИЕ 4. Двухпороговый таймер с предупредительной сигнализацией (FB_DualThresholdTON)

Технологическое назначение и постановка задачи

Требования безопасности машин и правила по охране труда предусматривают, что перед автоматическим пуском опасного технологического оборудования (прокатные станы, конвейеры транспортировки бревен, дробильные установки) персонал должен быть заблаговременно оповещен звуковым и световым сигналом. Конкретные требования к виду и длительности сигнала требуют уточнения по действующим ТНПА и проектной документации.

Блок `FB_DualThresholdTON` формирует предупредительный сигнал за фиксированное время `tWarningLeadTime` до переключения основного силового выхода `xMainOutput`.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xRun	BOOL	Команда на запуск технологического цикла
	tTotalDelay	TIME	Полная технологическая выдержка времени
	tWarningLeadTime	TIME	Длительность предупредительной сигнализации до пуска
VAR_OUTPUT	xWarningSignal	BOOL	Сигнал включения сирены / световой колонны
	xMainOutput	BOOL	Основной силовой выход включения привода
	tTimeToWarning	TIME	Время, оставшееся до включения звукового предупреждения

Направление	Идентификатор	Тип данных	Описание сигнала
	tTimeToStart	TIME	Время, оставшееся до пуска силового привода

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_DualThresholdTON
VAR_INPUT
    xRun                : BOOL; // Команда пуска установки
    tTotalDelay         : TIME; // Общее время задержки до
пуска привода
    tWarningLeadTime    : TIME; // Опережение
предупредительного сигнала (например, за 5 секунд)
END_VAR
VAR_OUTPUT
    xWarningSignal      : BOOL; // Выход включения
светозвуковой сирены
    xMainOutput         : BOOL; // Выход главного
контактора привода
    tTimeToWarning      : TIME; // Хронометраж: сколько
времени осталось до сирены
    tTimeToStart        : TIME; // Хронометраж: сколько
времени осталось до пуска
END_VAR
VAR
    xPrevRun            : BOOL; // Детектор фронта команды
пуска
    tStartTimestamp     : TIME; // Метка системного времени
старта
    tElapsed            : TIME; // Фактическое прошедшее
время
    tWarningThreshold   : TIME; // Расчетная точка времени
активации сирены
END_VAR

// 1. ИНИЦИАЛИЗАЦИЯ И ДЕТЕКЦИЯ СТАРТА
IF xRun AND NOT xPrevRun THEN
    tStartTimestamp := TIME();
    tElapsed        := T#0MS;
    xWarningSignal  := FALSE;
    xMainOutput     := FALSE;
END_IF;
```

```
// 2. ВАЛИДАЦИЯ УСТАВОК ВРЕМЕНИ
// Сирена не может звучать дольше, чем общее время задержки
IF tWarningLeadTime < tTotalDelay THEN
    tWarningThreshold := tTotalDelay - tWarningLeadTime;
ELSE
    tWarningThreshold := T#0MS; // Сирена включится сразу
при нажатии кнопки пуска
END_IF;

// 3. ОСНОВНОЙ РАСЧЕТ ВРЕМЕНИ И СРАВНЕНИЕ С ПОРОГАМИ
IF xRun THEN
    tElapsed := TIME() - tStartTimestamp;

    // Фаза 1: Проверка порога включения предупредительной
сигнализации
    IF tElapsed >= tWarningThreshold AND tElapsed <
tTotalDelay THEN
        xWarningSignal := TRUE;
    ELSE
        xWarningSignal := FALSE;
    END_IF;

    // Фаза 2: Проверка порога пуска главного привода
    IF tElapsed >= tTotalDelay THEN
        xMainOutput    := TRUE;
        xWarningSignal := FALSE; // Снятие сирены после
успешного пуска
        tElapsed       := tTotalDelay;
    ELSE
        xMainOutput    := FALSE;
    END_IF;

    // Расчет оставшихся интервалов времени для
визуализации на HMI
    IF tElapsed < tWarningThreshold THEN
        tTimeToWarning := tWarningThreshold - tElapsed;
    ELSE
        tTimeToWarning := T#0MS;
    END_IF;

    tTimeToStart := tTotalDelay - tElapsed;
ELSE
    // При снятии команды пуска — мгновенное отключение
всех сигналов
```

```
    xWarningSignal := FALSE;
    xMainOutput    := FALSE;
    tTimeToWarning := tWarningThreshold;
    tTimeToStart   := tTotalDelay;
    tElapsed       := T#0MS;
END_IF;

xPrevRun := xRun;
```

Разбор алгоритмической логики

- Защита от некорректного ввода уставок. Точка включения сирены рассчитывается как `tWarningThreshold := tTotalDelay - tWarningLeadTime`. Если оператор ввел длительность сирены 10 секунд при общей задержке 6 секунд, расчет автоматически корректируется на нулевую точку (`T#0MS`), исключая отрицательные интервалы.
- Гашение предупредительной сирены `xWarningSignal := FALSE` строго синхронизировано со включением главного привода `xMainOutput := TRUE`, предотвращая лишнюю шумовую нагрузку на цех.

РЕШЕНИЕ 5. Программный фильтр подавления дребезга контактов (FB_DebounceFilter)

Технологическое назначение и постановка задачи

При замыкании и размыкании механических контактов (кнопки на пультах, электромеханические концевые выключатели, реле давления, поплавковые герконовые датчики) в течение первых 2–20 миллисекунд наблюдается физический дребезг контактов — серия хаотических микросекундных размыканий и замыканий. Если завести такой сигнал на скоростной счетный вход или в конечный автомат без фильтрации, контроллер зарегистрирует ложные многократные нажатия.

Блок `FB_DebounceFilter` реализует раздельную программную фильтрацию переднего и заднего фронтов, гарантируя появление сигнала `xFilteredOut` только при его непрерывной стабильности.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xRawIn	BOOL	Неотфильтрованный физический дискретный сигнал со входа ПЛК

Направление	Идентификатор	Тип данных	Описание сигнала
	tFilterOn	TIME	Время стабильности для подтверждения включения (например, 20 мс)
	tFilterOff	TIME	Время стабильности для подтверждения отключения (например, 30 мс)
VAR_OUTPUT	xFilteredOut	BOOL	Отфильтрованный надежный дискретный сигнал
	xPulseOn	BOOL	Одиночный строб фронта включения
	xPulseOff	BOOL	Одиночный строб фронта выключения

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_DebounceFilter
VAR_INPUT
    xRawIn          : BOOL; // Сырой дискретный вход с физического модуля
    tFilterOn       : TIME := T#20MS; // Время фильтрации переднего фронта
    tFilterOff      : TIME := T#20MS; // Время фильтрации заднего фронта
END_VAR
VAR_OUTPUT
    xFilteredOut    : BOOL; // Валидированный стабильный выход
    xPulseOn        : BOOL; // Однократный импульс по подтвержденному включению
    xPulseOff       : BOOL; // Однократный импульс по подтвержденному выключению
END_VAR
VAR
    fbTimerOn       : TON;  // Стандартный таймер фильтрации включения
    fbTimerOff      : TON;  // Стандартный таймер фильтрации отключения
    xPrevFiltered   : BOOL; // Предыдущее состояние фильтрованного выхода
END_VAR

// 1. ФИЛЬТРАЦИЯ ПЕРЕДНЕГО ФРОНТА (ВКЛЮЧЕНИЕ)
```

```
// Таймер запустится, если вход замкнут, а отфильтрованный
выход все еще равен FALSE
fbTimerOn(IN := (xRawIn AND NOT xFilteredOut), PT :=
tFilterOn);

IF fbTimerOn.Q THEN
    xFilteredOut := TRUE; // Физический вход подтвердил
непрерывную стабильность
END_IF;

// 2. ФИЛЬТРАЦИЯ ЗАДНЕГО ФРОНТА (ОТКЛЮЧЕНИЕ)
// Таймер запустится, если вход разомкнут, а
отфильтрованный выход все еще равен TRUE
fbTimerOff(IN := (NOT xRawIn AND xFilteredOut), PT :=
tFilterOff);

IF fbTimerOff.Q THEN
    xFilteredOut := FALSE; // Размыкание подтверждено
END_IF;

// 3. ФОРМИРОВАНИЕ СТРОБ-ИМПУЛЬСОВ ДЛЯ ВНЕШНИХ АВТОМАТОВ
xPulseOn  := xFilteredOut AND NOT xPrevFiltered;
xPulseOff := NOT xFilteredOut AND xPrevFiltered;

// 4. СОХРАНЕНИЕ ПРЕДЫСТОРИИ
xPrevFiltered := xFilteredOut;
```

Разбор алгоритмической логики

- Вызов `fbTimerOn(IN := (xRawIn AND NOT xFilteredOut))` автоматически сбрасывает таймер, если сигнал `xRawIn` исчезнет хотя бы на один миллисекундный скан ПЛК. Внутренний счетчик `TON` начнет отсчет заново. Это гарантирует подавление любого высокочастотного дребезга.
- Раздельные уставки `tFilterOn` и `tFilterOff` позволяют гибко настраивать датчики: например, быстрое обнаружение приближения детали индуктивным датчиком (5 мс) и медленное подтверждение освобождения зоны (50 мс для защиты от провисания кабелей).

РЕШЕНИЕ 6. Таймер с защитой от кратковременных провалов сигнала (FB_GlitchFilterTON)

Технологическое назначение и постановка задачи

В промышленных сетях и полевых кабельных трассах большой протяженности дискретные датчики часто подвергаются наводкам от мощных

частотно-регулируемых электроприводов (ПЧ). Это вызывает появление кратковременных провалов сигнала (Glitch) длительностью 1–3 миллисекунды.

Если такой сигнал управляет классическим таймером TON, одиночный провал сбрасывает весь накопленный интервал технологической выдержки. Блок FB_GlitchFilterTON обеспечивает продолжение отсчета, если кратковременное пропадание сигнала не превысило заданный порог устойчивости tGlitchTolerance.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xSignal	BOOL	Входной дискретный сигнал
	tTargetDelay	TIME	Основная уставка задержки включения
	tGlitchTolerance	TIME	Максимально допустимая длительность помехи (например, 10 мс)
VAR_OUTPUT	xOutput	BOOL	Выход срабатывания таймера
	tAccumulatedTime	TIME	Общее накопленное время сигнала
	udiGlitchesSuppressed	UDINT	Счетчик подавленных помех для сервисной службы

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_GlitchFilterTON
VAR_INPUT
    xSignal             : BOOL; // Входной технологический
сигнал
    tTargetDelay        : TIME; // Основная задержка
включения
    tGlitchTolerance    : TIME := T#10MS; // Максимальная
допустимая длительность провала
END_VAR
VAR_OUTPUT
    xOutput             : BOOL; // Выходной сигнал
    tAccumulatedTime    : TIME; // Накопленное время
    udiGlitchesSuppressed : UDINT; // Диагностика:
количество проигнорированных сбоев
```

```
END_VAR
VAR
    fbGlitchTimer       : TON;  // Таймер фиксации
недопустимой длительности провала
    tStartTimestamp     : TIME; // Метка старта процесса
    xRunning            : BOOL; // Флаг активности отсчета
    xPrevSignal         : BOOL; // История входа
END_VAR

// 0. ПОДСЧЕТ ПОДАВЛЕННЫХ ПОМЕХ: сигнал вернулся, а отсчет
не был прерван
IF xSignal AND NOT xPrevSignal AND xRunning THEN
    udiGlitchesSuppressed := udiGlitchesSuppressed + 1;
END_IF;

// 1. ФИКСАЦИЯ ПЕРВИЧНОГО ПУСКА
IF xSignal AND NOT xRunning THEN
    tStartTimestamp := TIME();
    xRunning        := TRUE;
    xOutput         := FALSE;
END_IF;

// 2. МОНИТОРИНГ И ПОДАВЛЕНИЕ ПРОВАЛА СИГНАЛА (GLITCH)
// Таймер измеряет время нахождения сигнала в состоянии
FALSE при активном процессе
fbGlitchTimer(IN := (NOT xSignal AND xRunning), PT :=
tGlitchTolerance);

IF fbGlitchTimer.Q THEN
    // Провал длился дольше допустимого предела: это не
помеха, а реальное отключение!
    xRunning        := FALSE;
    xOutput         := FALSE;
    tAccumulatedTime:= T#0MS;
END_IF;

// 3. РАСЧЕТ И ВЫДАЧА ОСНОВНОГО ВРЕМЕНИ
IF xRunning AND xSignal THEN
    tAccumulatedTime := TIME() - tStartTimestamp;
    IF tAccumulatedTime >= tTargetDelay THEN
        xOutput := TRUE;
    END_IF;
END_IF;
```

```
xPrevSignal := xSignal;
```

Разбор алгоритмической логики

- В исходной редакции счетчик помех находился внутри ветки `ELSIF NOT xSignal …` и проверял условие `xSignal AND NOT xPrevSignal`, которое в этой ветке никогда не выполняется. Счетчик всегда оставался равным нулю. Подсчет перенесен в секцию 0 и выполняется по возврату сигнала, если отсчет не был прерван.
- Время провала, не превысившего `tGlitchTolerance`, входит в `tAccumulatedTime`: отсчет ведется от момента первичного пуска. Если время провала нужно исключать, используйте принцип накопления приращений из решения 1.
- Порог `tGlitchTolerance` должен быть больше периода задачи: провал короче одного цикла ПЛК программно вообще не обнаруживается.

РЕШЕНИЕ 7. Симметричный таймер задержки включения и отключения (FB_TONOF)

Технологическое назначение и постановка задачи

Для управления вентиляторами охлаждения шкафов, насосами подачи смазки и индикаторными контурами требуется одновременная реализация двух функций: включение только после стабильного присутствия запроса в течение времени `tDelayOn` и удержание механизма во включенном состоянии еще в течение времени `tDelayOff` после снятия запроса (для охлаждения ТЭНов или удаления газов).

Блок `FB_TONOF` объединяет функции блоков `TON` и `TOF` в едином оптимизированном коде без лишних накладных расходов.

Спецификация интерфейса

Направление	Идентификатор	Тип данных	Описание сигнала
VAR_INPUT	xInput	BOOL	Запрос на включение агрегата
	tDelayOn	TIME	Уставка времени задержки включения
	tDelayOff	TIME	Уставка времени задержки отключения (выбег)
VAR_OUTPUT	xOutput	BOOL	Результирующий выход управления пускателем
	tActiveET	TIME	Текущее время активного переходного процесса

Программная реализация на Structured Text

```
FUNCTION_BLOCK FB_TONOF
VAR_INPUT
    xInput      : BOOL; // Входной сигнал управления
    tDelayOn    : TIME; // Задержка включения
    tDelayOff   : TIME; // Задержка отключения
END_VAR
VAR_OUTPUT
    xOutput     : BOOL; // Итоговый выходной сигнал
    tActiveET   : TIME; // Индикация отсчитываемого
интервала
END_VAR
VAR
    fbTimerOn   : TON;  // Внутренний таймер включения
    fbTimerOff  : TOF;  // Внутренний таймер отключения
END_VAR

// Каскадный вызов внутренних стандартизированных
экземпляров
fbTimerOn(IN := xInput, PT := tDelayOn);
fbTimerOff(IN := fbTimerOn.Q, PT := tDelayOff);

// Результирующий сигнал формируется выходом хвостового
таймера TOF
xOutput := fbTimerOff.Q;

// Формирование диагностического времени переходного
процесса
IF xInput AND NOT xOutput THEN
    tActiveET := fbTimerOn.ET;  // Отображаем накопление
времени до пуска
ELSIF NOT xInput AND xOutput THEN
    tActiveET := fbTimerOff.ET; // Отображаем время выбега
до останова
ELSE
    tActiveET := T#0MS;
END_IF;
```

Разбор алгоритмической логики

- Блок последовательно соединяет `TON` и `TOF`: выход включается, если запрос держится не меньше `tDelayOn`, и отключается через `tDelayOff` после снятия запроса (после срабатывания `TON`).

- Если запрос пропадает во время задержки включения, `TON` сбрасывается и выход не включается — короткие запросы фильтруются.

РЕШЕНИЕ 8. Сторожевой таймер с автоматическим перезапуском (FB_WatchdogSupervisor)

Технологическое назначение и постановка задачи

В распределенных АСУ ТП центральный контроллер непрерывно обменивается пакетами данных с удаленными станциями ввода-вывода, панелями оператора HMI или весовыми терминалами по протоколам Modbus TCP / EtherCAT. Для гарантированного обнаружения зависания периферийного узла или обрыва линии связи используется механизм передачи периодического сигнала жизни (Lifebit / Heartbeat).

Блок `FB_WatchdogSupervisor` ожидает периодического изменения входного сигнала `xHeartbeat` в течение интервала `tWatchdogTimeout`. Если за заданное время сигнал жизни не изменил свое логическое состояние, блок формирует аварийный сигнал `xCommTimeoutAlarm` для безопасного останова оборудования.