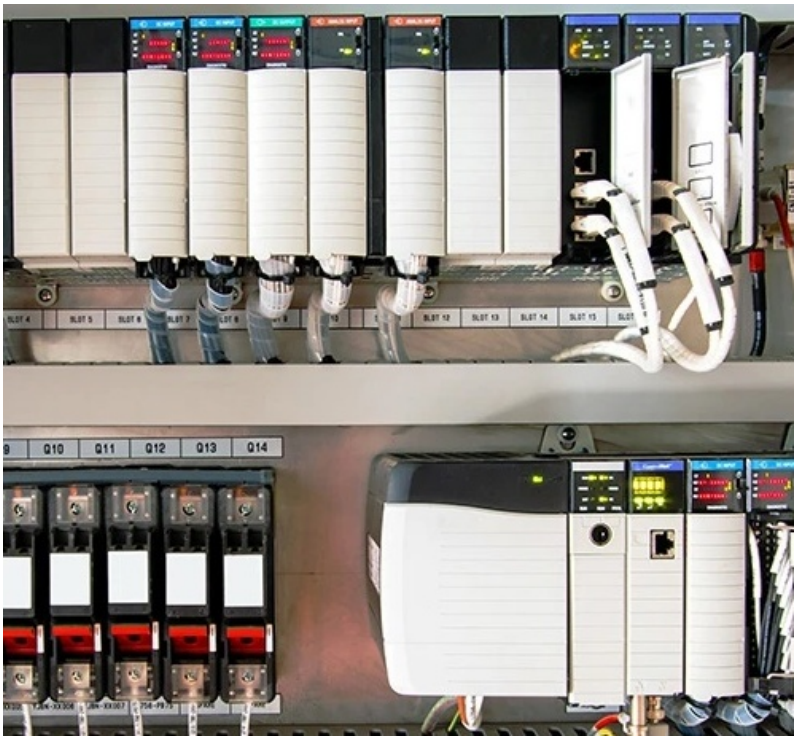

Практическое программирование ПЛК: решенные задачи на Structured Text (ST)

V1.0

Повный Андрей Владимирович, преподаватель Филиала БГТУ «Гомельский государственный политехнический колледж»

Автоматика и робототехника - https://t.me/club_automate

Программируемые контроллеры, автоматизация - https://vk.com/club_plc

Школа для электрика - https://t.me/electricalschool

Гомель, 2025

Введение

Современные системы автоматизации и управления технологическими процессами требуют четкого понимания принципов работы динамических звеньев и регуляторов. Программируемые логические контроллеры (ПЛК) играют ключевую роль в реализации алгоритмов управления, а язык **Structured Text (ST)**, входящий в стандарт **IEC 61131-3**, является одним из наиболее мощных инструментов для разработки сложных регуляторов и фильтров.

Данный задачник предназначен для **инженеров, программистов ПЛК и студентов**, изучающих автоматизацию и управление. В нем представлена **постепенно усложняющаяся подборка задач** — от базовых динамических звеньев до полноценных ПИД-регуляторов и их комбинаций.

Цели задачника:

✓ **Практическое освоение ST** – от простых арифметических операций до сложных алгоритмов.

✓ **Понимание динамических систем** – моделирование звеньев, анализ переходных процессов.

✓ **Разработка регуляторов** – от пропорциональных (П) до интегро-дифференцирующих (ИД) структур.

✓ **Применение в реальных задачах** – управление двигателями, температурой, уровнем и другими процессами.

Структура задачника:

1. **Базовые динамические звенья** – апериодические, интегрирующие, дифференцирующие.
2. **Комбинированные системы** – колебательные звенья, фильтры, соединения звеньев.
3. **Регуляторы** – П, ПИ, ПИД, защита от перегрузок.
4. **Практические приложения** – управление резервуарами, температурой, скоростью.

Каждая задача включает:

- **Теоретическую справку** (передаточная функция, назначение).
- **Готовый код на ST** с комментариями.
- **Рекомендации по настройке** и применению.

Этот задачник поможет не только научиться программировать регуляторы, но и **глубже понять теорию автоматического управления** через практику.

Рекомендуемый порядок изучения: линейный, от простого к сложному, но задачи можно выполнять и выборочно, в зависимости от уровня подготовки.

P.S. Для лучшего усвоения материала рекомендуется экспериментировать с параметрами в симуляторах (например, CODESYS, TIA Portal, MATLAB/Simulink).

Список задач:

Базовые элементы:

1. **Усилительное звено** (W(s)=K)
   Простейший статический элемент
2. **Звено запаздывания**
   Реализация временной задержки
3. **Апериодическое звено первого порядка**
   Базовый элемент, простейшая динамическая система
4. **Интегрирующее звено**
   Основа для понимания накопления
5. **Реально-дифференцирующее звено**
   Введение в дифференцирование с фильтрацией
6. **Апериодическое звено второго порядка**
   Развитие темы динамических звеньев
7. **Колебательное звено**
   Сложный случай динамики с осцилляциями
8. **Стандартный ПИ-регулятор**
   Комбинация пропорционального и интегрального
9. **Стандартный ПИД-регулятор**
   Полная версия с дифференциальной составляющей
10. **Интегро-дифференцирующее звено**
    Комплексный элемент как заключительная задача

Дополнительные задачи для задачника:

Комбинированные системы:

11. **Последовательное соединение звеньев**
    Каскад из 2-3 разных звеньев
12. **Параллельное соединение звеньев**
    Суммирование выходов разных звеньев

Специальные элементы:

13. **Релейный элемент с гистерезисом**
    Реализация нелинейного элемента
14. **Лимитер с плавным ограничением**
    Saturation с smooth-переходом
15. **Фильтр низких частот (ФНЧ)**
    Вариант апериодического звена с настройкой по частоте

Практические приложения:

16. **Управление двигателем с ПИД-регулятором**
    Полная система с обратной связью

17. **Температурный регулятор с защитой**
    Практическая реализация с защитами

18. **Балансировка двух резервуаров**
    Система с двумя взаимосвязанными регуляторами

Дополнительно:

19. **Генератор сигналов (синус, меандр, пила)**
    Для тестирования других звеньев

20. **Адаптивный ПИД-регулятор**
    С автоматической подстройкой параметров

21. **Генератор управляющего воздействия**
    Реализация ПИД без объекта управления

22. **Фильтр скользящего среднего**
    Альтернативный метод фильтрации

23. **Взвешенный медианный фильтр**
    Для обработки зашумленных сигналов

Лучшая книга по ST на русском языке:

Изучаем Structured Text МЭК 61131-3

Задача 1. Разработать приложение на языке ST, реализующее усилительное звено (W(s)=K)

Усилительное звено (пропорциональное звено) - это простейший тип динамического звена с передаточной функцией:

$$W(s) = K$$

где:

- **K** - коэффициент усиления;
- **s** - оператор Лапласа.

Свойства:

- Не изменяет форму сигнала;
- Усиливает/ослабляет входной сигнал в **K** раз;
- Не имеет динамических характеристик (нет переходных процессов);
- Фазовая характеристика: 0° на всех частотах.

Полная реализация на языке ST

```
FUNCTION_BLOCK AMPLIFYING_ELEMENT
VAR_INPUT
   IN: REAL;                 // Входной сигнал
   ENABLE: BOOL := TRUE;       // Активация блока
   K: REAL := 1.0;             // Коэффициент усиления (по умолчанию 1)
   MANUAL: BOOL := FALSE;      // Ручной режим
   MANUAL_VALUE: REAL := 0.0;  // Значение выхода в ручном режиме
END_VAR

VAR_OUTPUT
   OUT: REAL;                // Выходной сигнал
END_VAR

METHOD CALCULATE: REAL
VAR_INPUT
   INPUT_VAL: REAL;
```

```
END_VAR
BEGIN
   // Простейшая формула усилительного звена
   RETURN K * INPUT_VAL;
END_METHOD

METHOD UPDATE
BEGIN
   IF NOT ENABLE THEN
      OUT := 0.0;            // Блок отключен - выход 0
   ELSIF MANUAL THEN
      OUT := MANUAL_VALUE;    // Ручной режим управления
   ELSE
      OUT := CALCULATE(IN);   // Автоматический режим
   END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
   UPDATE();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
   SignalAmplifier: AMPLIFYING_ELEMENT;
   RawSignal: REAL := 10.0;    // Входной сигнал
   AmplifiedSignal: REAL;      // Усиленный сигнал
END_VAR

// Инициализация усилителя
```

```
SignalAmplifier(
   K := 2.5,       // Установка коэффициента усиления
   ENABLE := TRUE, // Активация блока
   MANUAL := FALSE // Автоматический режим
);

// Основной цикл обработки
SignalAmplifier.IN := RawSignal; // Подаем входной сигнал
SignalAmplifier();               // Вызываем блок усиления

// Получаем выходное значение
AmplifiedSignal := SignalAmplifier.OUT;
// Теперь AmplifiedSignal = 25.0 (10.0 * 2.5)
```

Дополнительные модификации

1. Версия с ограничением выхода

```
FUNCTION_BLOCK AMPLIFYING_ELEMENT_LIMITED
VAR_INPUT
   IN: REAL;
   K: REAL := 1.0;
   MAX_OUT: REAL := 100.0;  // Верхний предел
   MIN_OUT: REAL := -100.0; // Нижний предел
END_VAR

VAR_OUTPUT
   OUT: REAL;
   CLIPPED: BOOL := FALSE; // Флаг ограничения
END_VAR

BEGIN
   OUT := K * IN;
```

```
    // Ограничение выходного значения
    IF OUT > MAX_OUT THEN
        OUT := MAX_OUT;
        CLIPPED := TRUE;
    ELSIF OUT < MIN_OUT THEN
        OUT := MIN_OUT;
        CLIPPED := TRUE;
    ELSE
        CLIPPED := FALSE;
    END_IF;
END_FUNCTION_BLOCK
```

2. Версия с плавным изменением коэффициента

```
FUNCTION_BLOCK AMPLIFYING_ELEMENT_SMOOTH
VAR_INPUT
    IN: REAL;
    K_TARGET: REAL := 1.0;     // Целевой коэффициент
    K_RATE: REAL := 0.1;       // Скорость изменения (ед./сек)
    TS: REAL := 0.1;           // Время дискретизации
END_VAR

VAR_OUTPUT
    OUT: REAL;
END_VAR

VAR
    K_CURRENT: REAL := 1.0;    // Текущий коэффициент
END_VAR

BEGIN
```

```
    // Плавное изменение коэффициента
    IF K_CURRENT < K_TARGET THEN
        K_CURRENT := K_CURRENT + K_RATE * TS;
        IF K_CURRENT > K_TARGET THEN
            K_CURRENT := K_TARGET;
        END_IF;
    ELSIF K_CURRENT > K_TARGET THEN
        K_CURRENT := K_CURRENT - K_RATE * TS;
        IF K_CURRENT < K_TARGET THEN
            K_CURRENT := K_TARGET;
        END_IF;
    END_IF;
    OUT := K_CURRENT * IN;
END_FUNCTION_BLOCK
```

Типовые применения в автоматизации

1. **Коррекция сигналов датчиков**

```
// Пример: преобразование сигнала датчика давления (0-10V → 0-16bar)
PressureScaling(K := 1.6); // 10V * 1.6 = 16bar
```

2. **Управление мощностью исполнительных механизмов**

```
// Регулировка скорости двигателя
MotorSpeedControl(K := 0.8); // 80% от максимальной скорости
```

3. **Балансировка сигналов в сложных системах**

```
// Согласование сигналов между подсистемами
SignalBalancer(K := 0.75);
```

4. **Тестовые режимы оборудования**

```
// Режим тестирования с пониженной мощностью
TestModeAmplifier(K := 0.5); // 50% от номинала
```

Рекомендации по применению

1. **Защита от переполнения**: Всегда учитывайте возможный диапазон входных значений;

2. **Динамическое изменение K**: Реализуйте плавное изменение коэффициента при необходимости;
3. **Комбинирование с другими блоками**: Часто используется вместе с фильтрами и ограничителями;
4. **Калибровка оборудования**: Идеально подходит для задач калибровки датчиков.

Отличия от других звеньев

Характеристика	Усилительное звено	Интегрирующее	Дифференцирующее
Изменение сигнала	Только амплитуда	Накопление	Скорость изменения
Фазовый сдвиг	Нет	-90°	+90°
Реакция на скачок	Мгновенная	Постепенная	Импульсная
Основное применение	Масштабирование	Накопление	Выявление изменений

Этот базовый элемент является строительным блоком для более сложных систем управления и часто используется как составная часть ПИД-регуляторов и других алгоритмов автоматизации.

Задача 2. Разработать приложение на языке ST, реализующее звено запаздывания

Звено запаздывания описывается передаточной функцией:

$$W(s) = e^{(-\tau s)}$$

где:

- **τ** - время запаздывания [сек];
- **s** - оператор Лапласа.

Основные свойства:

- Выходной сигнал повторяет входной с задержкой на время τ;
- Не изменяет амплитуду сигнала;

- Вызывает фазовый сдвиг, пропорциональный частоте;
- Критически важно для моделирования реальных промышленных процессов.

Полная реализация на языке ST

1. Базовая версия с кольцевым буфером

```
FUNCTION_BLOCK TIME_DELAY
VAR_INPUT
   IN: REAL;                // Входной сигнал
   ENABLE: BOOL := TRUE;       // Активация блока
   DELAY_TIME: REAL := 1.0;    // Время запаздывания [сек]
   TS: REAL := 0.1;            // Время дискретизации [сек]
   RESET: BOOL := FALSE;       // Сброс буфера
END_VAR

VAR_OUTPUT
   OUT: REAL;               // Выходной сигнал
   BUFFER_FULL: BOOL := FALSE; // Флаг заполненности буфера
END_VAR

VAR
   BUFFER: ARRAY[0..1000] OF REAL; // Кольцевой буфер
   WRITE_PTR: INT := 0;        // Указатель записи
   READ_PTR: INT := 0;         // Указатель чтения
   BUFFER_SIZE: INT := 0;      // Фактический размер буфера
   INIT_DONE: BOOL := FALSE;   // Флаг инициализации
END_VAR

METHOD INIT_BUFFER
// Инициализация буфера
VAR
   i: INT;
```

```
END_VAR
BEGIN
   BUFFER_SIZE := MIN(DELAY_TIME / TS, 1000);
   IF BUFFER_SIZE < 1 THEN
      BUFFER_SIZE := 1;
   END_IF;

   FOR i := 0 TO BUFFER_SIZE-1 DO
      BUFFER[i] := IN; // Заполняем текущим значением
   END_FOR;

   WRITE_PTR := 0;
   READ_PTR := 0;
   INIT_DONE := TRUE;
END_METHOD

METHOD UPDATE
// Обновление буфера и выходного значения
BEGIN
   IF RESET OR NOT INIT_DONE THEN
      INIT_BUFFER();
      BUFFER_FULL := FALSE;
   ELSIF ENABLE THEN
      // Запись нового значения
      BUFFER[WRITE_PTR] := IN;
      WRITE_PTR := (WRITE_PTR + 1) MOD BUFFER_SIZE;

      // Чтение задержанного значения
      OUT := BUFFER[READ_PTR];
      READ_PTR := (READ_PTR + 1) MOD BUFFER_SIZE;
```

```
        // Установка флага заполненности
        IF WRITE_PTR = READ_PTR THEN
            BUFFER_FULL := TRUE;
        END_IF;
    ELSE
        OUT := IN; // В обход задержки при отключении
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UPDATE();
END_FUNCTION_BLOCK
```

2. Оптимизированная версия с динамическим буфером

```
FUNCTION_BLOCK TIME_DELAY_DYNAMIC
VAR_INPUT
    IN: REAL;
    DELAY_TIME: REAL := 1.0;
    TS: REAL := 0.1;
    MAX_DELAY: REAL := 60.0; // Максимальная задержка [сек]
END_VAR

VAR_OUTPUT
    OUT: REAL;
END_VAR

VAR
    BUFFER: POINTER TO ARRAY OF REAL;
    BUFFER_SIZE: UINT;
    WRITE_IDX: UINT := 0;
```

```
    INITIALIZED: BOOL := FALSE;
END_VAR

METHOD INIT: BOOL
// Динамическая инициализация буфера
VAR
    status: BOOL := TRUE;
END_VAR
BEGIN
    IF DELAY_TIME <= 0.0 OR TS <= 0.0 THEN
        status := FALSE;
    ELSE
        BUFFER_SIZE := MIN(DELAY_TIME/TS, MAX_DELAY/TS);
        MEMSET(BUFFER, 0, BUFFER_SIZE*SIZEOF(REAL));
        WRITE_IDX := 0;
        INITIALIZED := TRUE;
    END_IF;
    RETURN status;
END_METHOD

METHOD UPDATE_DELAY
VAR
    read_idx: UINT;
    elements: UINT;
END_VAR
BEGIN
    IF NOT INITIALIZED THEN
        IF NOT INIT() THEN
            OUT := IN;
            RETURN;
        END_IF;
```

```
    END_IF;

    // Запись нового значения
    BUFFER[WRITE_IDX] := IN;

    // Расчет индекса чтения
    elements := BUFFER_SIZE;
    read_idx := (WRITE_IDX + 1) MOD elements;

    // Чтение задержанного значения
    OUT := BUFFER[read_idx];

    // Обновление индекса записи
    WRITE_IDX := read_idx;
END_METHOD

BEGIN
    UPDATE_DELAY();
END_FUNCTION_BLOCK
```

Примеры использования

1. Базовое применение

```
PROGRAM MAIN
VAR
    ProcessDelay: TIME_DELAY;
    SensorValue: REAL;
    DelayedValue: REAL;
END_VAR

ProcessDelay(
    DELAY_TIME := 5.0,   // Задержка 5 секунд
```

```
    TS := 0.1           // Период обновления 100 мс
);

// Основной цикл
ProcessDelay.IN := SensorValue;
ProcessDelay();
DelayedValue := ProcessDelay.OUT;
```

2. Применение в системе управления

```
FUNCTION_BLOCK TEMPERATURE_CONTROLLER
VAR_INPUT
    TempInput: REAL;
    DelayEnabled: BOOL;
END_VAR

VAR
    TempDelay: TIME_DELAY;
    PID: PID_CONTROLLER;
END_VAR

TempDelay(
    DELAY_TIME := 2.5,
    TS := 0.1,
    ENABLE := DelayEnabled
);

TempDelay.IN := TempInput;
TempDelay();

PID(
    ProcessValue := TempDelay.OUT,
```

```
    // ... другие параметры ПИД
);
```

Особенности реализации

1. **Кольцевой буфер** - эффективная структура данных для реализации задержки.
2. **Динамическая настройка**:
    - Время задержки может изменяться в runtime;
    - Автоматический расчет размера буфера.
3. **Защитные механизмы**:
    - Проверка корректности параметров;
    - Ограничение максимальной задержки;
    - Обработка случая нулевой задержки.
4. **Эффективность**:
    - Минимальные вычислительные затраты;
    - Постоянное время выполнения независимо от длины задержки.

Рекомендации по применению

1. **Выбор времени дискретизации**:
    - Должно соответствовать периоду вызова блока;
    - Влияет на точность и размер буфера.
2. **Память ПЛК**:
    - Для больших задержек требуется значительная память;
    - Максимальный размер буфера следует ограничивать.
3. **Компенсация запаздывания**:
    - В системах управления может потребоваться специальная компенсация;
    - Метод Смита для компенсации транспортного запаздывания.
4. **Комбинирование с другими блоками**:
    - Часто используется вместе с фильтрами и регуляторами;
    - Может применяться для синхронизации сигналов.

Сравнение методов реализации

Метод	**Плюсы**	**Минусы**
Кольцевой буфер	Простота, стабильность	Фиксированный размер
Динамический буфер	Гибкость, экономия памяти	Сложность реализации
Линейная интерполяция	Более плавный выход	Вычислительная нагрузка
FIFO-очередь	Точное время задержки	Требовательность к памяти

Данная реализация обеспечивает баланс между точностью, производительностью и потреблением памяти, что делает ее идеальной для промышленных применений.

Задача 3. Разработать приложение на языке ST, реализующее апериодическое звено первого порядка

Апериодическое звено первого порядка (инерционное звено) описывается дифференциальным уравнением:

$$T * dy/dt + y = K * x$$

где:

- T - постоянная времени;
- K - коэффициент усиления;
- x - входной сигнал;
- y - выходной сигнал.

Реализация функционального блока

```
FUNCTION_BLOCK FirstOrderLag
VAR_INPUT
    Input: REAL;            // Входной сигнал
    K: REAL := 1.0;         // Коэффициент усиления (по умолчанию 1)
    T: REAL := 1.0;         // Постоянная времени (сек)
    Ts: REAL := 0.1;        // Время дискретизации (период вызова, сек)
```

```
    Reset: BOOL := FALSE;    // Сброс состояния (выход = 0)
END_VAR

VAR_OUTPUT
    Output: REAL;            // Выходной сигнал
END_VAR

VAR
    PrevOutput: REAL := 0.0; // Предыдущее значение выхода
END_VAR

METHOD Calculate: REAL
VAR_INPUT
    InputValue: REAL;
END_VAR
BEGIN
    // Дискретная реализация апериодического звена 1-го порядка
    // Используем метод Эйлера для численного интегрирования
    RETURN PrevOutput + (Ts / T) * (K * InputValue - PrevOutput);
END_METHOD

METHOD UpdateOutput
BEGIN
    IF Reset THEN
        Output := 0.0;
        PrevOutput := 0.0;
    ELSE
        Output := Calculate(Input);
        PrevOutput := Output;
    END_IF;
END_METHOD
```

```
// Основной код функционального блока
BEGIN
    UpdateOutput();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
    Filter: FirstOrderLag;
    RawInput: REAL;          // "Сырой" входной сигнал (например, с датчика)
    FilteredValue: REAL;     // Отфильтрованное значение
END_VAR

// Инициализация параметров фильтра
Filter(
    K := 1.0,       // Коэффициент усиления
    T := 2.0,       // Постоянная времени 2 секунды
    Ts := 0.1       // Период вызова 100 мс
);

// Основной цикл обработки
Filter.Input := RawInput; // Передаем входное значение
Filter();                 // Вызываем фильтр

// Получаем отфильтрованное значение
FilteredValue := Filter.Output;
```

Особенности реализации

1. **Дискретная реализация**: Используется метод Эйлера для численного интегрирования дифференциального уравнения.
2. **Настраиваемые параметры**:

- K - коэффициент усиления;
- T - постоянная времени (чем больше T, тем более инерционное звено);
- Ts - период дискретизации (должен соответствовать реальному периоду вызова блока).

3. **Функция сброса**: При установке Reset в TRUE выходное значение сбрасывается в 0.
4. **Простота вычислений**: Алгоритм требует минимальных вычислительных ресурсов, что важно для ПЛК.

Теория

Апериодическое звено первого порядка имеет передаточную функцию:

W(s) = K / (T*s + 1)

где s - оператор Лапласа.

Временные характеристики:

- При подаче на вход ступенчатого сигнала выход достигает 63% установившегося значения за время T;
- Установившееся значение равно K * Input.

Эта реализация может использоваться для:

- Фильтрации сигналов датчиков;
- Моделирования динамических процессов;
- Создания "мягких" включений/выключений;
- Аппроксимации более сложных динамических систем.

Задача 4. Разработать приложение на языке ST, реализующее интегрирующее звено

Интегрирующее звено описывается передаточной функцией:

$$W(s) = K/(T*s)$$

где:

- K - коэффициент усиления;
- T - постоянная времени интегрирования;
- s - оператор Лапласа.

Полная реализация функционального блока

```
FUNCTION_BLOCK INTEGRATING_ELEMENT
VAR_INPUT
   // Основные входные параметры
   IN: REAL;                // Входной сигнал
   ENABLE: BOOL := TRUE;       // Активация блока
   K: REAL := 1.0;             // Коэффициент усиления (по умолчанию 1)
   TI: REAL := 1.0;            // Постоянная времени интегрирования [сек]
   TS: REAL := 0.1;            // Время дискретизации [сек]

   // Дополнительные параметры
   RESET: BOOL := FALSE;       // Сброс интегратора (выход = 0)
   MANUAL: BOOL := FALSE;      // Ручной режим
   MANUAL_VALUE: REAL := 0.0;  // Значение выхода в ручном режиме
   MAX_OUT: REAL := 1000.0;    // Максимальное значение выхода
   MIN_OUT: REAL := -1000.0;   // Минимальное значение выхода
END_VAR

VAR_OUTPUT
   OUT: REAL;                  // Выходной сигнал
   ERROR: BOOL := FALSE;       // Флаг ошибки (выход за пределы)
END_VAR

VAR
   // Внутренние переменные
   INTEGRAL: REAL := 0.0;      // Накопленное значение интеграла
   PREV_IN: REAL := 0.0;       // Предыдущее значение входа
   FIRST_SCAN: BOOL := TRUE;   // Флаг первого скана
END_VAR

METHOD CALCULATE: REAL
```

```
VAR_INPUT
   INPUT_VAL: REAL;
END_VAR
VAR
   NEW_OUTPUT: REAL;
END_VAR
BEGIN
   // Метод трапеций для численного интегрирования
   NEW_OUTPUT := INTEGRAL + K * (INPUT_VAL + PREV_IN) * TS / (2.0 * TI);

   // Проверка на переполнение
   IF NEW_OUTPUT > MAX_OUT THEN
      NEW_OUTPUT := MAX_OUT;
      ERROR := TRUE;
   ELSIF NEW_OUTPUT < MIN_OUT THEN
      NEW_OUTPUT := MIN_OUT;
      ERROR := TRUE;
   ELSE
      ERROR := FALSE;
   END_IF;

   RETURN NEW_OUTPUT;
END_METHOD

METHOD UPDATE
BEGIN
   IF RESET OR FIRST_SCAN THEN
      // Сброс всех состояний
      INTEGRAL := 0.0;
      PREV_IN := 0.0;
      OUT := 0.0;
```

```
        ERROR := FALSE;
        FIRST_SCAN := FALSE;
    ELSIF NOT ENABLE THEN
        // Блок отключен - выход не изменяется
        ERROR := FALSE;
    ELSIF MANUAL THEN
        // Ручной режим
        OUT := MANUAL_VALUE;
        INTEGRAL := MANUAL_VALUE; // Для плавного перехода в автоматический
        ERROR := FALSE;
    ELSE
        // Автоматический режим - вычисление интеграла
        OUT := CALCULATE(IN);
        INTEGRAL := OUT;
        PREV_IN := IN;
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UPDATE();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
    FlowIntegrator: INTEGRATING_ELEMENT;
    FlowRate: REAL;           // Текущий расход [м3/ч]
    TotalVolume: REAL;        // Накопленный объем [м3]
    bResetTotal: BOOL;        // Кнопка сброса объема
END_VAR
```

```
// Инициализация интегратора расхода
FlowIntegrator(
    K := 1.0,             // Коэффициент усиления
    TI := 3600.0,         // Интегрирование за 1 час (3600 сек)
    TS := 1.0,            // Период вызова 1 сек
    MAX_OUT := 999999.0,  // Максимальный объем
    MIN_OUT := 0.0,       // Минимальный объем
    RESET := bResetTotal  // Сброс по команде
);

// Основной цикл расчета
FlowIntegrator.IN := FlowRate;  // Подаем текущий расход
FlowIntegrator();               // Вызываем интегратор

// Получаем накопленный объем
TotalVolume := FlowIntegrator.OUT;
```

Дополнительные особенности реализации

1. **Антивиндап защита**: Ограничение выходного значения для предотвращения переполнения.
2. **Гибкое управление**:
    - Ручной/автоматический режим;
    - Возможность сброса;
    - Отключение блока.
3. **Диагностика**: Флаг ошибки при достижении предельных значений.
4. **Точность интегрирования**: Использование метода трапеций для минимизации ошибки дискретизации.
5. **Универсальность**: Может использоваться для различных приложений:
    - Накопление количества продукции;
    - Расчет общего времени работы;
    - Интегральная составляющая в регуляторах;

- Преобразование скорости в положение.

Для настройки:

- Установите K=1 и TI в желаемый период интегрирования;
- Настройте TS равным периоду вызова блока;
- Установите MAX_OUT/MIN_OUT согласно требованиям процесса.

Задача 5. Разработать приложение на языке ST, реализующее реально-дифференцирующее звено

Реально-дифференцирующее звено описывается передаточной функцией:

$$W(s) = K*T*s / (T*s + 1)$$

где:

- K - коэффициент усиления;
- T - постоянная времени дифференцирования;
- s - оператор Лапласа.

Полная реализация функционального блока

```
FUNCTION_BLOCK REAL_DIFFERENTIATING_ELEMENT
VAR_INPUT
  // Основные входные параметры
  IN: REAL;                 // Входной сигнал
  ENABLE: BOOL := TRUE;     // Активация блока
  K: REAL := 1.0;           // Коэффициент усиления (по умолчанию 1)
  TD: REAL := 1.0;          // Постоянная времени дифференцирования [сек]
  TS: REAL := 0.1;          // Время дискретизации [сек]

  // Дополнительные параметры
  RESET: BOOL := FALSE;     // Сброс состояния (выход = 0)
  MANUAL: BOOL := FALSE;    // Ручной режим
  MANUAL_VALUE: REAL := 0.0;  // Значение выхода в ручном режиме
  MAX_OUT: REAL := 1000.0;    // Максимальное значение выхода
```

```
    MIN_OUT: REAL := -1000.0;   // Минимальное значение выхода
END_VAR

VAR_OUTPUT
    OUT: REAL;                  // Выходной сигнал
    ERROR: BOOL := FALSE;       // Флаг ошибки (выход за пределы)
END_VAR

VAR
    // Внутренние переменные
    PREV_IN: REAL := 0.0;       // Предыдущее значение входа
    PREV_OUT: REAL := 0.0;      // Предыдущее значение выхода
    FIRST_SCAN: BOOL := TRUE;   // Флаг первого скана
END_VAR

METHOD CALCULATE: REAL
VAR_INPUT
    INPUT_VAL: REAL;
END_VAR
BEGIN
    // Дискретная реализация реального дифференцирующего звена
    // Используется метод обратной разности (Backward Euler)
    RETURN (K * TD * (INPUT_VAL - PREV_IN) + TS * PREV_OUT) / (TD + TS);
END_METHOD

METHOD UPDATE
VAR
    NEW_OUTPUT: REAL;
END_VAR
BEGIN
    IF RESET OR FIRST_SCAN THEN
```

```
    // Сброс всех состояний
    PREV_IN := 0.0;
    PREV_OUT := 0.0;
    OUT := 0.0;
    ERROR := FALSE;
    FIRST_SCAN := FALSE;
ELSIF NOT ENABLE THEN
    // Блок отключен - выход не изменяется
    ERROR := FALSE;
ELSIF MANUAL THEN
    // Ручной режим
    OUT := MANUAL_VALUE;
    PREV_OUT := MANUAL_VALUE; // Для плавного перехода в автоматический
    ERROR := FALSE;
ELSE
    // Автоматический режим - вычисление дифференциала
    NEW_OUTPUT := CALCULATE(IN);

    // Проверка на переполнение
    IF NEW_OUTPUT > MAX_OUT THEN
        NEW_OUTPUT := MAX_OUT;
        ERROR := TRUE;
    ELSIF NEW_OUTPUT < MIN_OUT THEN
        NEW_OUTPUT := MIN_OUT;
        ERROR := TRUE;
    ELSE
        ERROR := FALSE;
    END_IF;

    // Обновление состояний
    OUT := NEW_OUTPUT;
```

```
        PREV_OUT := NEW_OUTPUT;
        PREV_IN := IN;
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UPDATE();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

st

```
PROGRAM MAIN
VAR
    TempDifferentiator: REAL_DIFFERENTIATING_ELEMENT;
    Temperature: REAL;          // Текущая температура [°C]
    TempRate: REAL;             // Скорость изменения температуры [°C/мин]
    bResetRate: BOOL;           // Кнопка сброса
END_VAR

// Инициализация дифференциатора температуры
TempDifferentiator(
    K := 1.0,              // Коэффициент усиления
    TD := 10.0,            // Постоянная времени 10 сек
    TS := 1.0,             // Период вызова 1 сек
    MAX_OUT := 10.0,       // Максимальная скорость
    MIN_OUT := -10.0,      // Минимальная скорость
    RESET := bResetRate    // Сброс по команде
);

// Основной цикл расчета
TempDifferentiator.IN := Temperature;  // Подаем текущую температуру
```

TempDifferentiator(); // Вызываем дифференциатор

// Получаем скорость изменения температуры (в °C/сек)

TempRate := TempDifferentiator.OUT * 60.0; // Преобразуем в °C/мин

Дополнительные особенности реализации

1. **Фильтрация высокочастотных шумов**: В отличие от идеального дифференциатора, реальный дифференциатор имеет фильтрующие свойства благодаря наличию постоянной времени TD.
2. **Гибкое управление**:
    - Ручной/автоматический режим
    - Возможность сброса
    - Отключение блока
3. **Диагностика**: Флаг ошибки при достижении предельных значений.
4. **Точность дифференцирования**: Используется метод обратной разности для устойчивой дискретной реализации.
5. **Универсальность**: Может использоваться для различных приложений:
    - Определение скорости изменения параметров;
    - Прогнозирование трендов;
    - Д-составляющая в ПИД-регуляторах;
    - Обнаружение резких изменений сигналов.
6. **Настройка**:
    - Установите K=1 и TD в желаемую постоянную времени;
    - Настройте TS равным периоду вызова блока;
    - Установите MAX_OUT/MIN_OUT согласно требованиям процесса.
7. **Особенности работы**:
    - Чем меньше TD, тем ближе звено к идеальному дифференциатору;
    - Чем больше TD, тем сильнее фильтрующий эффект;
    - При TD=0 звено становится идеальным дифференциатором (не рекомендуется из-за усиления шумов).

Задача 6. Разработать приложение на языке ST, реализующее апериодическое звено второго порядка

Апериодическое звено второго порядка описывается передаточной функцией:

$$W(s) = K / (T1*s + 1)(T2*s + 1)$$

где:

- K - коэффициент усиления;
- T1, T2 - постоянные времени;
- s - оператор Лапласа.

Реализация функционального блока

```
FUNCTION_BLOCK SecondOrderLag
VAR_INPUT
    Input: REAL;            // Входной сигнал
    K: REAL := 1.0;         // Коэффициент усиления
    T1: REAL := 1.0;        // Первая постоянная времени (сек)
    T2: REAL := 1.0;        // Вторая постоянная времени (сек)
    Ts: REAL := 0.1;        // Время дискретизации (период вызова, сек)
    Reset: BOOL := FALSE;   // Сброс состояния (выход = 0)
END_VAR

VAR_OUTPUT
    Output: REAL;           // Выходной сигнал
END_VAR

VAR
    // Состояния для дискретной реализации
    x1: REAL := 0.0;        // Первое состояние (промежуточное значение)
    x2: REAL := 0.0;        // Второе состояние (выход)
    PrevInput: REAL := 0.0; // Предыдущее значение входа
END_VAR
```

```
METHOD CalculateOutput: REAL
VAR_INPUT
    InputValue: REAL;
END_VAR
VAR
    a0, a1, a2, b1, b2: REAL;
    NewOutput: REAL;
END_VAR
BEGIN
    // Коэффициенты дискретной модели (билинейное преобразование)
    a0 := 4.0 * T1 * T2 + 2.0 * (T1 + T2) * Ts + Ts * Ts;
    a1 := 2.0 * Ts * Ts - 8.0 * T1 * T2;
    a2 := 4.0 * T1 * T2 - 2.0 * (T1 + T2) * Ts + Ts * Ts;
    b1 := 2.0 * K * Ts * Ts;
    b2 := K * Ts * Ts;

    // Расчет нового выходного значения
    NewOutput := (b1 * InputValue + b2 * PrevInput - a1 * x1 - a2 * x2) / a0;

    // Обновление состояний
    PrevInput := InputValue;
    x2 := x1;
    x1 := NewOutput;

    RETURN NewOutput;
END_METHOD

METHOD UpdateOutput
BEGIN
    IF Reset THEN
        // Сброс всех состояний
```

```
        x1 := 0.0;
        x2 := 0.0;
        PrevInput := 0.0;
        Output := 0.0;
    ELSE
        // Расчет нового значения
        Output := CalculateOutput(Input);
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UpdateOutput();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
    Filter: SecondOrderLag;
    RawInput: REAL;          // Нефильтрованный входной сигнал
    FilteredValue: REAL;     // Отфильтрованное значение
END_VAR

// Инициализация фильтра
Filter(
    K := 1.0,       // Коэффициент усиления
    T1 := 2.0,      // Первая постоянная времени
    T2 := 0.5,      // Вторая постоянная времени
    Ts := 0.1       // Период вызова 100 мс
);
```

```
// Основной цикл обработки
Filter.Input := RawInput; // Передаем входное значение
Filter();                 // Вызываем фильтр

// Получаем отфильтрованное значение
FilteredValue := Filter.Output;
```

Особенности реализации

1. **Дискретная реализация**: Использовано билинейное (Tustin) преобразование для перехода от непрерывной модели к дискретной.
2. **Два состояния**: Реализация через пространство состояний для точного соответствия дифференциальному уравнению второго порядка.
3. **Настраиваемые параметры**:
    - K - коэффициент усиления;
    - T1, T2 - постоянные времени;
    - Ts - период дискретизации.
4. **Функция сброса**: При установке Reset в TRUE все внутренние состояния сбрасываются в 0.
5. **Стабильность**: Алгоритм устойчив при правильном выборе параметров (T1, T2 > 0).

Теория и применение

Апериодическое звено второго порядка:

- Имеет более крутой спад АЧХ по сравнению с звеном первого порядка
- Может использоваться для:
    - Фильтрации сигналов с двумя разными постоянными времени;
    - Моделирования сложных инерционных объектов;
    - Создания плавных траекторий изменения сигналов;
    - Аппроксимации более сложных динамических систем.

При T1 = T2 характеристика становится критически демпфированной (без выброса).

Для настройки:

1. T1 определяет основную инерционность;
2. T2 добавляет дополнительное запаздывание;
3. K устанавливает статическое усиление.

Задача 7. Разработать приложение на языке ST, реализующее колебательное звено

Колебательное звено описывается передаточной функцией:

$$W(s) = K / (T^2*s^2 + 2*\xi*T*s + 1)$$

где:

- K - коэффициент усиления;
- T - постоянная времени;
- ξ (дзета) - коэффициент демпфирования (ξ < 1 - колебания);
- s - оператор Лапласа.

Реализация функционального блока

```
FUNCTION_BLOCK OscillatoryElement
VAR_INPUT
    Input: REAL;            // Входной сигнал
    K: REAL := 1.0;         // Коэффициент усиления
    T: REAL := 1.0;         // Постоянная времени (сек)
    Zeta: REAL := 0.5;      // Коэффициент демпфирования (0 < ξ < 1)
    Ts: REAL := 0.1;        // Время дискретизации (период вызова, сек)
    Reset: BOOL := FALSE;   // Сброс состояния (выход = 0)
END_VAR

VAR_OUTPUT
    Output: REAL;           // Выходной сигнал
    OutputDerivative: REAL; // Производная выходного сигнала (опционально)
END_VAR

VAR
    // Состояния для дискретной реализации
    x1: REAL := 0.0;        // Первое состояние (выход)
    x2: REAL := 0.0;        // Второе состояние (производная выхода)
    PrevInput: REAL := 0.0; // Предыдущее значение входа
```

```
    FirstPass: BOOL := TRUE; // Флаг первого прохода
END_VAR

METHOD CalculateOutput
VAR
    a0, a1, a2, b0: REAL;
    NewOutput: REAL;
    NewDerivative: REAL;
BEGIN
    // Коэффициенты дискретной модели (билинейное преобразование)
    a0 := 4.0*T*T + 4.0*Zeta*T*Ts + Ts*Ts;
    a1 := 2.0*Ts*Ts - 8.0*T*T;
    a2 := 4.0*T*T - 4.0*Zeta*T*Ts + Ts*Ts;
    b0 := K*Ts*Ts;

    // Расчет новых значений состояний
    NewOutput := (b0*(Input + PrevInput) - a1*x1 - a2*x2) / a0;
    NewDerivative := (2.0/Ts)*(NewOutput - x1) - x2;

    // Обновление состояний
    PrevInput := Input;
    x2 := NewDerivative;
    x1 := NewOutput;

    // Установка выходных значений
    Output := NewOutput;
    OutputDerivative := NewDerivative;
END_METHOD

METHOD UpdateOutput
BEGIN
```

```
    IF Reset OR FirstPass THEN
        // Сброс всех состояний
        x1 := 0.0;
        x2 := 0.0;
        PrevInput := 0.0;
        Output := 0.0;
        OutputDerivative := 0.0;
        FirstPass := FALSE;
    ELSE
        // Расчет нового значения
        CalculateOutput();
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UpdateOutput();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
    Oscillator: OscillatoryElement;
    InputSignal: REAL;
    FilteredOutput: REAL;
    OutputDerivative: REAL;
END_VAR

// Инициализация колебательного звена
Oscillator(
    K := 1.0,       // Коэффициент усиления
```

```
    T := 2.0,       // Постоянная времени 2 секунды
    Zeta := 0.3,    // Коэффициент демпфирования (0.3 - выраженные колебания)
    Ts := 0.05      // Период вызова 50 мс
);

// Основной цикл обработки
Oscillator.Input := InputSignal; // Передаем входное значение
Oscillator();                    // Вызываем блок

// Получаем выходные значения
FilteredOutput := Oscillator.Output;
OutputDerivative := Oscillator.OutputDerivative;
```

Особенности реализации

1. **Дискретная реализация**: Использовано билинейное преобразование для перехода от непрерывной модели к дискретной.
2. **Два состояния**: Реализация через пространство состояний (выход и его производная).
3. **Настраиваемые параметры**:
    - K - коэффициент усиления;
    - T - постоянная времени (определяет частоту колебаний);
    - Zeta - коэффициент демпфирования (0 < Zeta < 1 - колебательный режим);
    - Ts - период дискретизации.
4. **Дополнительный выход**: Производная выходного сигнала для анализа динамики.
5. **Функция сброса**: При установке Reset в TRUE все внутренние состояния сбрасываются.

Теория и применение

Колебательное звено:

- При $\xi < 1$ проявляет колебательные свойства
- Частота колебаний: $\omega = 1/T$
- Период колебаний: $T0 = 2\pi T/\sqrt{(1-\xi^2)}$
- Логарифмический декремент затухания: $\Lambda = 2\pi\xi/\sqrt{(1-\xi^2)}$

Применение:

- Моделирование механических систем с упругостью и трением;
- Фильтрация сигналов с резонансными свойствами;
- Создание генераторов тестовых сигналов;
- Анализ устойчивости систем управления.

Для получения различных режимов:

- $\xi = 0$ - незатухающие колебания;
- $0 < \xi < 1$ - затухающие колебания;
- $\xi \geq 1$ - апериодический режим (реализован в предыдущих блоках).

Задача 8. Разработать приложение на языке ST, реализующее ПИ-регулятор

```
FUNCTION_BLOCK PI_Controller
VAR_INPUT
   // Входные параметры
   Setpoint: REAL;          // Заданное значение (уставка)
   ProcessValue: REAL;      // Текущее значение процесса
   Kp: REAL := 1.0;         // Коэффициент пропорциональной составляющей
   Ti: REAL := 10.0;        // Постоянная времени интегрирования (сек)
   Ts: REAL := 0.1;         // Время дискретизации (период вызова блока, сек)
   ManualMode: BOOL := FALSE; // Ручной режим
   ManualOutput: REAL := 0.0; // Значение выхода в ручном режиме
   Reset: BOOL := FALSE;    // Сброс интегральной составляющей
   MaxOutput: REAL := 100.0; // Максимальное значение выхода
   MinOutput: REAL := 0.0;   // Минимальное значение выхода
END_VAR

VAR_OUTPUT
   Output: REAL;            // Выходное значение регулятора
   Error: REAL;             // Текущая ошибка (Setpoint - ProcessValue)
END_VAR
```

```
VAR
    Integral: REAL := 0.0;   // Интегральная составляющая
    PrevError: REAL := 0.0;  // Предыдущее значение ошибки
END_VAR

METHOD Calculate: REAL
VAR_INPUT
    Setpoint: REAL;
    ProcessValue: REAL;
END_VAR

VAR
    CurrentError: REAL;
    Proportional: REAL;
END_VAR

BEGIN
    // Вычисление текущей ошибки
    CurrentError := Setpoint - ProcessValue;
    Error := CurrentError;

    // Пропорциональная составляющая
    Proportional := Kp * CurrentError;

    // Интегральная составляющая (метод трапеций)
    IF NOT Reset AND Ti > 0 THEN
        Integral := Integral + Kp * (CurrentError + PrevError) * Ts / (2.0 * Ti);

        // Ограничение интегральной составляющей для предотвращения windup
        IF Integral > MaxOutput THEN
```

```
            Integral := MaxOutput;
        ELSIF Integral < MinOutput THEN
            Integral := MinOutput;
        END_IF;
    ELSE
        Integral := 0.0;
    END_IF;

    // Запоминаем ошибку для следующего цикла
    PrevError := CurrentError;

    // Суммируем составляющие
    RETURN Proportional + Integral;
END_METHOD

METHOD UpdateOutput
BEGIN
    IF ManualMode THEN
        Output := ManualOutput;
        // Сброс интегральной составляющей при ручном режиме
        Integral := 0.0;
        PrevError := 0.0;
    ELSE
        Output := Calculate(Setpoint, ProcessValue);

        // Ограничение выходного значения
        IF Output > MaxOutput THEN
            Output := MaxOutput;
        ELSIF Output < MinOutput THEN
            Output := MinOutput;
        END_IF;
```

```
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UpdateOutput();
END_FUNCTION_BLOCK
```

Использование ПИ-регулятора в программе ПЛК

Пример вызова функционального блока в основной программе:

```
PROGRAM MAIN
VAR
    TemperatureController: PI_Controller;
    SetpointTemp: REAL := 50.0;
    CurrentTemp: REAL;
    HeaterOutput: REAL;
END_VAR

// Инициализация параметров регулятора
TemperatureController(
    Kp := 2.5,          // Коэффициент усиления
    Ti := 5.0,          // Постоянная времени интегрирования (сек)
    Ts := 0.1,          // Период вызова (100 мс)
    MaxOutput := 100.0,  // Максимальная мощность нагревателя
    MinOutput := 0.0     // Минимальная мощность нагревателя
);

// Основной цикл управления
TemperatureController.Setpoint := SetpointTemp;
TemperatureController.ProcessValue := CurrentTemp; // Текущая температура с датчика
```

```
// Вызов ПИ-регулятора
TemperatureController();

// Использование выходного значения
HeaterOutput := TemperatureController.Output;
```

Особенности реализации

1. **Антивиндап**: Реализовано ограничение интегральной составляющей для предотвращения "накрутки" (windup effect).
2. **Ручной режим**: Поддержка ручного управления выходным значением.
3. **Метод трапеций**: Для вычисления интегральной составляющей используется метод трапеций, который точнее простого прямоугольного метода.
4. **Ограничение выхода**: Выходное значение ограничено заданными пределами.
5. **Сброс интегратора**: Реализована возможность принудительного сброса интегральной составляющей.
6. **Дискретность**: Учитывается время дискретизации Ts для корректного расчета интегральной составляющей.

Для настройки регулятора необходимо подобрать коэффициенты Kp (пропорциональный) и Ti (интегральный) в соответствии с динамикой конкретного процесса.

Задача 9. Разработать приложение на языке ST, реализующее стандартный ПИД регулятор

Ниже представлена полная реализация ПИД-регулятора с дополнительными функциями для промышленных ПЛК.

```
FUNCTION_BLOCK PID_Controller
VAR_INPUT
  // Основные входы
  Setpoint: REAL;          // Заданное значение (уставка)
  ProcessValue: REAL;      // Текущее значение процесса
  Enable: BOOL := TRUE;    // Активация регулятора

  // Параметры настройки
```

```
    Kp: REAL := 1.0;         // Коэффициент пропорциональной составляющей
    Ti: REAL := 0.0;         // Постоянная времени интегрирования (сек, 0 - отключить)
    Td: REAL := 0.0;         // Постоянная времени дифференцирования (сек, 0 - отключить)
    Ts: REAL := 0.1;         // Время дискретизации (период вызова блока, сек)

    // Ограничения и режимы
    ManualMode: BOOL := FALSE; // Ручной режим
    ManualOutput: REAL := 0.0; // Значение выхода в ручном режиме
    MaxOutput: REAL := 100.0;  // Максимальное значение выхода
    MinOutput: REAL := 0.0;    // Минимальное значение выхода
    MaxOutputStep: REAL := 10.0; // Максимальное изменение выхода за один цикл

    // Дополнительные параметры
    DeadBand: REAL := 0.0;    // Зона нечувствительности
    Reset: BOOL := FALSE;     // Сброс внутренних состояний
END_VAR

VAR_OUTPUT
    Output: REAL;            // Выходное значение регулятора
    Error: REAL;             // Текущая ошибка (Setpoint - ProcessValue)
    P_Term: REAL;            // Пропорциональная составляющая (для диагностики)
    I_Term: REAL;            // Интегральная составляющая (для диагностики)
    D_Term: REAL;            // Дифференциальная составляющая (для диагностики)
END_VAR

VAR
    // Внутренние переменные состояния
    Integral: REAL := 0.0;   // Интегральная составляющая
    PrevError: REAL := 0.0;  // Предыдущее значение ошибки
    PrevProcessValue: REAL := 0.0; // Предыдущее значение процесса
    PrevOutput: REAL := 0.0; // Предыдущее значение выхода
```

```
    FirstPass: BOOL := TRUE; // Флаг первого прохода
END_VAR

METHOD CalculatePID: REAL
VAR_INPUT
    Setpoint: REAL;
    ProcessValue: REAL;
END_VAR
VAR
    CurrentError: REAL;
    Derivative: REAL;
    OutputValue: REAL;
    DeltaOutput: REAL;
END_VAR
BEGIN
    // Вычисление текущей ошибки с учетом зоны нечувствительности
    CurrentError := Setpoint - ProcessValue;
    Error := CurrentError;

    // Пропорциональная составляющая
    P_Term := Kp * CurrentError;

    // Интегральная составляющая (метод трапеций)
    IF Ti > 0 AND NOT FirstPass THEN
        Integral := Integral + Kp * (CurrentError + PrevError) * Ts / (2.0 * Ti);

        // Ограничение интегральной составляющей для предотвращения windup
        IF Integral > (MaxOutput - P_Term) THEN
            Integral := MaxOutput - P_Term;
        ELSIF Integral < (MinOutput - P_Term) THEN
            Integral := MinOutput - P_Term;
```

```
    END_IF;
END_IF;
I_Term := Integral;

// Дифференциальная составляющая (по измерению)
IF Td > 0 AND NOT FirstPass THEN
    Derivative := -Kp * Td * (ProcessValue - PrevProcessValue) / Ts;
ELSE
    Derivative := 0.0;
END_IF;
D_Term := Derivative;

// Суммируем все составляющие
OutputValue := P_Term + I_Term + D_Term;

// Ограничение скорости изменения выхода
IF NOT FirstPass AND MaxOutputStep > 0 THEN
    DeltaOutput := OutputValue - PrevOutput;
    IF DeltaOutput > MaxOutputStep THEN
        OutputValue := PrevOutput + MaxOutputStep;
    ELSIF DeltaOutput < -MaxOutputStep THEN
        OutputValue := PrevOutput - MaxOutputStep;
    END_IF;
END_IF;

// Общие ограничения выхода
IF OutputValue > MaxOutput THEN
    OutputValue := MaxOutput;
ELSIF OutputValue < MinOutput THEN
    OutputValue := MinOutput;
END_IF;
```

```
    RETURN OutputValue;
END_METHOD

METHOD UpdateOutput
BEGIN
    IF Reset OR FirstPass THEN
        // Сброс внутренних состояний
        Integral := 0.0;
        PrevError := 0.0;
        PrevProcessValue := ProcessValue;
        PrevOutput := 0.0;
        Output := 0.0;
        FirstPass := FALSE;
    ELSIF NOT Enable THEN
        // Режим отключен - выход 0
        Output := 0.0;
        Integral := 0.0;
    ELSIF ManualMode THEN
        // Ручной режим
        Output := ManualOutput;
        // Поддержание соответствия для плавного перехода в автоматический
        PrevOutput := ManualOutput;
    ELSE
        // Автоматический режим - расчет ПИД
        Output := CalculatePID(Setpoint, ProcessValue);

        // Сохраняем состояния для следующего цикла
        PrevError := Error;
        PrevProcessValue := ProcessValue;
        PrevOutput := Output;
```

```
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UpdateOutput();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
    TemperaturePID: PID_Controller;
    SetpointTemp: REAL := 75.0;
    CurrentTemp: REAL;
    HeaterOutput: REAL;
END_VAR

// Инициализация ПИД-регулятора температуры
TemperaturePID(
    Kp := 2.5,          // Пропорциональный коэффициент
    Ti := 120.0,        // Интегральное время (2 минуты)
    Td := 30.0,         // Дифференциальное время (30 сек)
    Ts := 0.2,          // Период вызова 200 мс
    MaxOutput := 100.0, // Максимальная мощность нагревателя (100%)
    MinOutput := 0.0,   // Минимальная мощность нагревателя (0%)
    MaxOutputStep := 5.0 // Максимальное изменение мощности за цикл (5%)
);

// Основной цикл управления
TemperaturePID.Setpoint := SetpointTemp;
TemperaturePID.ProcessValue := CurrentTemp; // Текущая температура с датчика
```

```
// Вызов ПИД-регулятора
TemperaturePID();

// Использование выходного значения
HeaterOutput := TemperaturePID.Output;
```

Особенности реализации

1. **Полная структура ПИД**:
    - Пропорциональная составляющая (P);
    - Интегральная составляющая (I) с антивиндапом;
    - Дифференциальная составляющая (D) по измерению (более устойчивая, чем по ошибке).
2. **Дополнительные функции**:
    - Ручной/автоматический режим;
    - Ограничение выходного сигнала;
    - Ограничение скорости изменения выхода;
    - Зона нечувствительности (DeadBand);
    - Сброс внутренних состояний.
3. **Защитные механизмы**:
    - Обработка первого вызова;
    - Плавные переходы между режимами.
4. **Диагностика**:
    - Выходы отдельных составляющих (P_Term, I_Term, D_Term);
    - Текущая ошибка регулирования.
5. **Оптимизация для ПЛК**:
    - Минимальные вычисления в каждом цикле и учет времени дискретизации Ts.

Для настройки регулятора рекомендуется:

1. Сначала установить Ti=0 и Td=0, настроить Kp;
2. Затем настроить Ti для устранения статической ошибки;
3. И только потом добавлять Td для улучшения динамики.

Задача 10. Разработать приложение на языке ST, реализующее интегро-дифференцирующее звено

Интегро-дифференцирующее звено описывается передаточной функцией:

$$W(s) = K * (Td*s + 1 + 1/(Ti*s))$$

где:

- K - коэффициент усиления;
- Td - постоянная времени дифференцирования;
- Ti - постоянная времени интегрирования;
- s - оператор Лапласа.

Полная реализация функционального блока

```
FUNCTION_BLOCK INTEGRO_DIFF_ELEMENT
VAR_INPUT
   // Основные входные параметры
   IN: REAL;                // Входной сигнал
   ENABLE: BOOL := TRUE;       // Активация блока
   K: REAL := 1.0;             // Коэффициент усиления
   TI: REAL := 10.0;           // Постоянная времени интегрирования [сек]
   TD: REAL := 1.0;            // Постоянная времени дифференцирования [сек]
   TS: REAL := 0.1;            // Время дискретизации [сек]

   // Дополнительные параметры
   RESET: BOOL := FALSE;       // Сброс внутренних состояний
   MANUAL: BOOL := FALSE;      // Ручной режим
   MANUAL_VALUE: REAL := 0.0;  // Значение выхода в ручном режиме
   MAX_OUT: REAL := 1000.0;    // Максимальное значение выхода
   MIN_OUT: REAL := -1000.0;   // Минимальное значение выхода
END_VAR

VAR_OUTPUT
   OUT: REAL;                  // Выходной сигнал
```

```
    ERROR: BOOL := FALSE;       // Флаг ошибки (выход за пределы)
END_VAR

VAR
    // Внутренние переменные состояния
    INTEGRAL: REAL := 0.0;      // Накопленное интегральное значение
    PREV_IN: REAL := 0.0;       // Предыдущее значение входа
    PREV_OUT: REAL := 0.0;      // Предыдущее значение выхода
    FIRST_SCAN: BOOL := TRUE;   // Флаг первого скана
END_VAR

METHOD CALCULATE: REAL
VAR_INPUT
    INPUT_VAL: REAL;
END_VAR
VAR
    DIFF_TERM: REAL;            // Дифференциальная составляющая
    INTEGRAL_TERM: REAL;        // Интегральная составляющая
    NEW_OUTPUT: REAL;
END_VAR
BEGIN
    // Дифференциальная составляющая (метод обратных разностей)
    DIFF_TERM := TD * (INPUT_VAL - PREV_IN) / TS;

    // Интегральная составляющая (метод трапеций)
    IF TI > 0 THEN
        INTEGRAL_TERM := INTEGRAL + (INPUT_VAL + PREV_IN) * TS / (2.0 * TI);
    ELSE
        INTEGRAL_TERM := 0.0;
    END_IF;
```

```
    // Суммируем все составляющие
    NEW_OUTPUT := K * (DIFF_TERM + INPUT_VAL + INTEGRAL_TERM);

    // Проверка на переполнение
    IF NEW_OUTPUT > MAX_OUT THEN
        NEW_OUTPUT := MAX_OUT;
        ERROR := TRUE;
    ELSIF NEW_OUTPUT < MIN_OUT THEN
        NEW_OUTPUT := MIN_OUT;
        ERROR := TRUE;
    ELSE
        ERROR := FALSE;
    END_IF;

    RETURN NEW_OUTPUT;
END_METHOD

METHOD UPDATE
BEGIN
    IF RESET OR FIRST_SCAN THEN
        // Сброс всех состояний
        INTEGRAL := 0.0;
        PREV_IN := 0.0;
        PREV_OUT := 0.0;
        OUT := 0.0;
        ERROR := FALSE;
        FIRST_SCAN := FALSE;
    ELSIF NOT ENABLE THEN
        // Блок отключен - выход не изменяется
        ERROR := FALSE;
    ELSIF MANUAL THEN
```

```
        // Ручной режим
        OUT := MANUAL_VALUE;
        INTEGRAL := MANUAL_VALUE / K; // Для плавного перехода
        PREV_IN := MANUAL_VALUE / K;
        ERROR := FALSE;
    ELSE
        // Автоматический режим - вычисление
        OUT := CALCULATE(IN);

        // Обновление состояний
        IF TI > 0 THEN
            INTEGRAL := INTEGRAL + (IN + PREV_IN) * TS / (2.0 * TI);
        END_IF;
        PREV_IN := IN;
        PREV_OUT := OUT;
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    UPDATE();
END_FUNCTION_BLOCK
```

Пример использования в программе ПЛК

```
PROGRAM MAIN
VAR
    ProcessFilter: INTEGRO_DIFF_ELEMENT;
    RawInput: REAL;             // Сырой входной сигнал
    FilteredOutput: REAL;       // Обработанный сигнал
    bResetFilter: BOOL;         // Кнопка сброса фильтра
END_VAR
```

```
// Инициализация интегро-дифференцирующего звена
ProcessFilter(
    K := 1.5,            // Коэффициент усиления
    TI := 5.0,           // Постоянная времени интегрирования
    TD := 0.5,           // Постоянная времени дифференцирования
    TS := 0.1,           // Период вызова 100 мс
    MAX_OUT := 100.0,    // Максимальный выход
    MIN_OUT := -100.0,   // Минимальный выход
    RESET := bResetFilter  // Сброс по команде
);

// Основной цикл обработки
ProcessFilter.IN := RawInput;  // Подаем входной сигнал
ProcessFilter();               // Вызываем блок обработки

// Получаем обработанный сигнал
FilteredOutput := ProcessFilter.OUT;
```

Особенности реализации

1. **Комбинированная обработка**:
    - Интегральная составляющая (метод трапеций);
    - Дифференциальная составляющая (метод обратных разностей);
    - Пропорциональная составляющая.
2. **Защитные механизмы**:
    - Ограничение выходного значения;
    - Защита от переполнения интегратора;
    - Сброс внутренних состояний.
3. **Гибкое управление**:
    - Ручной/автоматический режим;
    - Возможность отключения блока;

- Настройка параметров в реальном времени.

4. **Дискретная реализация**:
   - Учет времени дискретизации TS;
   - Корректная работа при изменении параметров.
5. **Применение**:
   - Фильтрация сигналов с выделением трендов;
   - Предварительная обработка сигналов для регуляторов;
   - Компенсация динамических характеристик систем;
   - Анализ скорости изменения процессов.

Для настройки:

1. Начните с K=1, TI=0 (только дифференцирование);
2. Затем добавьте интегральную составляющую, установив TI;
3. Подберите TD для желаемой реакции на быстрые изменения сигнала.

Задача 11. Разработать приложение на языке ST, реализующее последовательное соединение звеньев

Последовательное соединение звеньев - это каскадное подключение динамических элементов, где выход предыдущего звена является входом следующего. Общая передаточная функция:

$$W(s) = W_1(s) * W_2(s) * ... * W_n(s)$$

Полная реализация на языке ST

1. Универсальный блок для соединения любых звеньев

```
FUNCTION_BLOCK SERIAL_LINK
VAR_INPUT
   IN: REAL;                  // Входной сигнал
   ENABLE: BOOL := TRUE;          // Активация блока
   RESET: BOOL := FALSE;          // Сброс всех звеньев
END_VAR

VAR_OUTPUT
```

```
    OUT: REAL;                  // Выходной сигнал
END_VAR

VAR
    // Встроенные блоки звеньев (пример для 3 звеньев)
    BLOCK1: AMPLIFYING_ELEMENT;     // Усилительное звено
    BLOCK2: FIRST_ORDER_LAG;        // Апериодическое звено 1-го порядка
    BLOCK3: INTEGRATING_ELEMENT;    // Интегрирующее звено
    INITIALIZED: BOOL := FALSE;
END_VAR

METHOD INIT
// Инициализация параметров звеньев
BEGIN
    // Настройка усилительного звена
    BLOCK1(
        K := 2.5,
        ENABLE := TRUE,
        MANUAL := FALSE
    );

    // Настройка апериодического звена
    BLOCK2(
        K := 1.0,
        T := 2.0,
        TS := 0.1,
        RESET := FALSE
    );

    // Настройка интегрирующего звена
    BLOCK3(
```

```
        K := 1.0,
        TI := 5.0,
        TS := 0.1,
        RESET := FALSE
    );

    INITIALIZED := TRUE;
END_METHOD

METHOD UPDATE
// Последовательное обновление всех звеньев
BEGIN
    IF NOT INITIALIZED OR RESET THEN
        INIT();
    END_IF;

    IF ENABLE THEN
        // Последовательное соединение
        BLOCK1.IN := IN;
        BLOCK1();

        BLOCK2.IN := BLOCK1.OUT;
        BLOCK2();

        BLOCK3.IN := BLOCK2.OUT;
        BLOCK3();

        OUT := BLOCK3.OUT;
    ELSE
        OUT := IN; // Байпас при отключении
    END_IF;
```

```
END_METHOD

BEGIN
   UPDATE();
END_FUNCTION_BLOCK
```

2. Гибкая версия с массивом звеньев

```
FUNCTION_BLOCK FLEX_SERIAL_LINK
VAR_INPUT
   IN: REAL;
   ENABLE: BOOL := TRUE;
   RESET: BOOL := FALSE;
END_VAR

VAR_OUTPUT
   OUT: REAL;
END_VAR

VAR
   BLOCKS: ARRAY[0..9] OF GENERIC_BLOCK; // Массив звеньев
   BLOCK_COUNT: INT := 3;              // Фактическое количество
   INIT_DONE: BOOL := FALSE;
END_VAR

METHOD CONFIGURE
// Конфигурация цепочки звеньев
BEGIN
   // Пример конфигурации:
   // 1. Усилительное звено
   BLOCKS[0] := AMPLIFYING_ELEMENT(
      K := 1.5,
```

```
        ENABLE := TRUE
    );

    // 2. Апериодическое звено
    BLOCKS[1] := FIRST_ORDER_LAG(
        K := 1.0,
        T := 1.5,
        TS := 0.1
    );

    // 3. Интегрирующее звено
    BLOCKS[2] := INTEGRATING_ELEMENT(
        K := 0.8,
        TI := 10.0,
        TS := 0.1
    );

    BLOCK_COUNT := 3;
    INIT_DONE := TRUE;
END_METHOD

METHOD PROCESS_SIGNAL: REAL
VAR_INPUT
    input_val: REAL;
END_VAR
VAR
    i: INT;
    temp_val: REAL;
END_VAR
BEGIN
    temp_val := input_val;
```

```
    FOR i := 0 TO BLOCK_COUNT-1 DO
        CASE TYPE(BLOCKS[i]) OF
            AMPLIFYING_ELEMENT:
                BLOCKS[i].AsAMPLIFYING_ELEMENT.IN := temp_val;
                BLOCKS[i].AsAMPLIFYING_ELEMENT();
                temp_val := BLOCKS[i].AsAMPLIFYING_ELEMENT.OUT;

            FIRST_ORDER_LAG:
                BLOCKS[i].AsFIRST_ORDER_LAG.IN := temp_val;
                BLOCKS[i].AsFIRST_ORDER_LAG();
                temp_val := BLOCKS[i].AsFIRST_ORDER_LAG.OUT;

            // Добавьте обработку других типов звеньев...
        END_CASE;
    END_FOR;

    RETURN temp_val;
END_METHOD

BEGIN
    IF NOT INIT_DONE OR RESET THEN
        CONFIGURE();
    END_IF;

    IF ENABLE THEN
        OUT := PROCESS_SIGNAL(IN);
    ELSE
        OUT := IN;
    END_IF;
END_FUNCTION_BLOCK
```

Примеры использования

1. Простая цепочка из двух звеньев

```
PROGRAM MAIN
VAR
   MyFilter: SERIAL_LINK;
   InputSignal: REAL := 10.0;
   FilteredSignal: REAL;
END_VAR

// Конфигурация (может быть выполнена в переменных)
MyFilter.BLOCK1.K := 2.0;       // Усилитель 2x
MyFilter.BLOCK2.T := 1.0;       // Постоянная времени 1 сек

// Основной цикл обработки
MyFilter.IN := InputSignal;
MyFilter();
FilteredSignal := MyFilter.OUT;
```

2. Сложная система управления

```
FUNCTION_BLOCK PROCESS_CONTROLLER
VAR_INPUT
   Setpoint: REAL;
   ProcessValue: REAL;
END_VAR

VAR
   InputFilter: FLEX_SERIAL_LINK;
   MainPID: PID_CONTROLLER;
   OutputLimiter: LIMITER_BLOCK;
END_VAR
```

```
// Конфигурация входного фильтра
InputFilter.BLOCKS[0] := FIRST_ORDER_LAG(T := 2.0);
InputFilter.BLOCKS[1] := MOVING_AVERAGE(WINDOW := 5);
InputFilter.BLOCK_COUNT := 2;

// Основной цикл
InputFilter.IN := ProcessValue;
InputFilter();

MainPID(
   Setpoint := Setpoint,
   ProcessValue := InputFilter.OUT
);

OutputLimiter(
   IN := MainPID.OUT,
   MIN := 0.0,
   MAX := 100.0
);
```

Ключевые особенности реализации

1. **Модульность**:
    - Каждое звено реализовано как независимый функциональный блок;
    - Легко добавлять или удалять звенья.
2. **Гибкость конфигурации**:
    - Параметры каждого звена настраиваются индивидуально;
    - Возможность динамического изменения структуры.
3. Э**ффективность**:
    - Минимальные накладные расходы на соединение;
    - Оптимальное использование ресурсов ПЛК.

4. **Масштабируемость**:
    - Поддержка произвольного количества звеньев;
    - Возможность создания библиотеки типовых конфигураций.

Рекомендации по применению

1. **Порядок соединения**:
    - Фильтры → Преобразователи → Регуляторы → Исполнительные устройства;
    - Учитывайте фазовые сдвиги при комбинации разных звеньев.
2. **Отладка**:
    - Анализируйте сигналы между звеньями;
    - Используйте временное отключение звеньев для диагностики.
3. **Оптимизация**:
    - Для критичных по времени систем ограничьте количество звеньев;
    - Используйте упрощенные модели где это возможно.
4. **Безопасность**:
    - Добавляйте ограничители между звеньями;
    - Реализуйте механизмы аварийного обхода (bypass).

Дополнительные возможности

1. **Адаптивные цепи**:

```
IF condition THEN
   BLOCK2.T := 5.0; // Изменение параметров в runtime
END_IF;
```

2. **Переконфигурация**:

```
METHOD CHANGE_CONFIG
VAR
   new_config: ARRAY[0..9] OF GENERIC_BLOCK;
END_VAR
```

3. **Диагностика**:

```
METHOD GET_INTERNAL_SIGNAL: REAL
VAR_INPUT
   stage: INT; // Номер звена
```

```
END_VAR
```

Такая реализация позволяет создавать сложные системы обработки сигналов с понятной структурой и легкой настройкой.

Задача 12. Разработать приложение на языке ST, реализующее параллельное соединение звеньев

Параллельное соединение звеньев предполагает подачу одного входного сигнала на несколько звеньев одновременно, с последующим суммированием их выходных сигналов. Общая передаточная функция:

$$W(s) = W_1(s) + W_2(s) + ... + W_n(s)$$

Полная реализация на языке ST

1. Базовый вариант с фиксированным числом звеньев

```
FUNCTION_BLOCK PARALLEL_BLOCKS
VAR_INPUT
   IN: REAL;                  // Общий входной сигнал
   ENABLE: BOOL := TRUE;      // Активация блока
   RESET: BOOL := FALSE;      // Сброс всех звеньев
END_VAR

VAR_OUTPUT
   OUT: REAL;                 // Суммарный выход
   STATUS: INT := 0;          // Статус выполнения
END_VAR

VAR
   // Параллельные звенья (пример для 3 звеньев)
   BLOCK1: FIRST_ORDER_LAG;       // Апериодическое звено
   BLOCK2: INTEGRATING_ELEMENT;   // Интегратор
   BLOCK3: AMPLIFYING_ELEMENT;    // Усилитель

   // Коэффициенты ветвей
   K1: REAL := 1.0;
```

```
    K2: REAL := 1.0;
    K3: REAL := 1.0;

    INIT_DONE: BOOL := FALSE;
END_VAR

METHOD INITIALIZE
// Инициализация всех звеньев
BEGIN
    BLOCK1(
        K := 1.0,
        T := 2.0,
        TS := 0.1,
        RESET := RESET
    );

    BLOCK2(
        K := 1.0,
        TI := 5.0,
        TS := 0.1,
        RESET := RESET
    );

    BLOCK3(
        K := 1.0,
        ENABLE := TRUE,
        RESET := RESET
    );

    INIT_DONE := TRUE;
    STATUS := 1;
```

```
END_METHOD

METHOD PROCESS
// Параллельная обработка и суммирование
VAR
   out1, out2, out3: REAL;
END_VAR
BEGIN
   // Параллельное выполнение всех звеньев
   BLOCK1.IN := IN;
   BLOCK1();
   out1 := BLOCK1.OUT * K1;

   BLOCK2.IN := IN;
   BLOCK2();
   out2 := BLOCK2.OUT * K2;

   BLOCK3.IN := IN;
   BLOCK3();
   out3 := BLOCK3.OUT * K3;

   // Суммирование результатов
   OUT := out1 + out2 + out3;
   STATUS := 2;
END_METHOD

BEGIN
   IF NOT INIT_DONE OR RESET THEN
      INITIALIZE();
   END_IF;
```

```
    IF ENABLE THEN
        PROCESS();
    ELSE
        OUT := 0.0;
        STATUS := 0;
    END_IF;
END_FUNCTION_BLOCK
```

2. Расширенная версия с динамической конфигурацией

```
FUNCTION_BLOCK DYNAMIC_PARALLEL
VAR_INPUT
    IN: REAL;
    ENABLE: BOOL := TRUE;
    RESET: BOOL := FALSE;
END_VAR

VAR_OUTPUT
    OUT: REAL;
    ACTIVE_BLOCKS: INT := 0;
END_VAR

VAR
    BLOCKS: ARRAY[0..9] OF GENERIC_BLOCK;
    GAINS: ARRAY[0..9] OF REAL := [1.0, 1.0, 1.0, 0, 0, 0, 0, 0, 0, 0];
    BLOCK_COUNT: INT := 3;
    INITIALIZED: BOOL := FALSE;
END_VAR

METHOD CONFIGURE
// Настройка конфигурации параллельных ветвей
BEGIN
```

```
    // Пример конфигурации:
    BLOCKS[0] := FIRST_ORDER_LAG(T := 1.5);
    BLOCKS[1] := INTEGRATING_ELEMENT(TI := 3.0);
    BLOCKS[2] := AMPLIFYING_ELEMENT(K := 2.0);

    GAINS[0] := 0.8;
    GAINS[1] := 1.2;
    GAINS[2] := 0.5;

    BLOCK_COUNT := 3;
    INITIALIZED := TRUE;
END_METHOD

METHOD UPDATE
// Обновление и суммирование выходов
VAR
    i: INT;
    sum: REAL := 0.0;
    active: INT := 0;
END_VAR
BEGIN
    sum := 0.0;
    active := 0;

    FOR i := 0 TO BLOCK_COUNT-1 DO
        CASE TYPE(BLOCKS[i]) OF
            FIRST_ORDER_LAG:
                BLOCKS[i].AsFIRST_ORDER_LAG.IN := IN;
                BLOCKS[i].AsFIRST_ORDER_LAG();
                sum := sum + BLOCKS[i].AsFIRST_ORDER_LAG.OUT * GAINS[i];
                active := active + 1;
```

```
            INTEGRATING_ELEMENT:
                BLOCKS[i].AsINTEGRATING_ELEMENT.IN := IN;
                BLOCKS[i].AsINTEGRATING_ELEMENT();
                sum := sum + BLOCKS[i].AsINTEGRATING_ELEMENT.OUT * GAINS[i];
                active := active + 1;

            // Добавьте другие типы звеньев...
        END_CASE;
    END_FOR;

    OUT := sum;
    ACTIVE_BLOCKS := active;
END_METHOD

BEGIN
    IF NOT INITIALIZED OR RESET THEN
        CONFIGURE();
    END_IF;

    IF ENABLE THEN
        UPDATE();
    ELSE
        OUT := 0.0;
        ACTIVE_BLOCKS := 0;
    END_IF;
END_FUNCTION_BLOCK
```

Примеры использования

1. Фильтрация сигнала с разными характеристиками

```
PROGRAM MAIN
```

```
VAR
   SignalProcessor: PARALLEL_BLOCKS;
   RawInput: REAL;
   ProcessedOutput: REAL;
END_VAR

// Настройка параметров
SignalProcessor.BLOCK1.T := 0.5;    // Быстрый фильтр
SignalProcessor.BLOCK2.TI := 10.0;  // Медленный интегратор
SignalProcessor.K3 := 0.3;          // Вес усилителя

// Основной цикл
SignalProcessor.IN := RawInput;
SignalProcessor();
ProcessedOutput := SignalProcessor.OUT;
```

2. Адаптивная система управления

```
FUNCTION_BLOCK ADAPTIVE_CONTROLLER
VAR_INPUT
   ProcessValue: REAL;
   Mode: INT; // 0-медленный, 1-быстрый, 2-комбинированный
END_VAR

VAR
   ParallelPath: DYNAMIC_PARALLEL;
   ControlOutput: REAL;
END_VAR

// Конфигурация в зависимости от режима
CASE Mode OF
   0: // Медленный режим
```

```
        ParallelPath.GAINS[0] := 0.1; // Слабая фильтрация
        ParallelPath.GAINS[1] := 0.9; // Преобладает интегратор

    1: // Быстрый режим
        ParallelPath.GAINS[0] := 0.9;
        ParallelPath.GAINS[1] := 0.1;

    2: // Сбалансированный режим
        ParallelPath.GAINS[0] := 0.5;
        ParallelPath.GAINS[1] := 0.5;
END_CASE;

ParallelPath.IN := ProcessValue;
ParallelPath();
ControlOutput := ParallelPath.OUT;
```

Ключевые особенности реализации

1. **Параллельное выполнение**:
    - Все звенья обрабатывают входной сигнал одновременно;
    - Минимальная задержка между входом и выходом.
2. **Гибкое взвешивание**:
    - Индивидуальные коэффициенты для каждой ветви;
    - Возможность динамического изменения весов.
3. **Модульность архитектуры**:
    - Легко добавлять или отключать ветви обработки;
    - Независимая настройка каждого звена.
4. **Диагностика**:
    - Контроль количества активных ветвей;
    - Возможность мониторинга промежуточных результатов.

Рекомендации по применению

1. **Компенсация фазовых сдвигов**:

- При соединении разнотипных звеньев учитывайте их фазовые характеристики;
- Используйте дополнительные корректирующие звенья при необходимости.

2. **Оптимизация производительности**:
   - Для критичных по времени систем ограничьте количество параллельных ветвей;
   - Самые быстрые звенья размещайте первыми в массиве.
3. **Безопасность**:
   - Реализуйте ограничение суммарного выхода;
   - Добавьте проверку на переполнение при суммировании.
4. **Отладка**:
   - Временное отключение ветвей для анализа вклада каждой;
   - Визуализация промежуточных результатов.

Типовые сферы применения

1. **Многодиапазонная обработка сигналов**:
   - Параллельные фильтры для разных частотных диапазонов;
   - Анализ сигналов с разными динамическими характеристиками.
2. **Составные регуляторы**:
   - Комбинация быстрых и медленных каналов регулирования;
   - Адаптивные системы управления.
3. **Резервирование систем**:
   - Параллельное выполнение альтернативных алгоритмов;
   - Выбор лучшего результата или их усреднение.
4. **Системы мониторинга**:
   - Одновременный анализ сигнала разными методами;
   - Комплексная диагностика процессов.

Данная реализация обеспечивает высокую гибкость при построении сложных систем обработки сигналов с возможностью адаптации к различным требованиям технологических процессов.

Задача 13. Разработать приложение на языке ST, реализующее релейный элемент с гистерезисом

Релейный элемент с гистерезисом (пороговый элемент с памятью) реализует нелинейную характеристику управления. Основные параметры:

- **Уставка включения (Setpoint_ON)** - значение, при котором выход переключается в ON;
- **Уставка выключения (Setpoint_OFF)** - значение, при котором выход переключается в OFF;
- **Гистерезис (Hysteresis)** - зона нечувствительности между переключениями.

Полная реализация на языке ST

1. Базовая версия релейного элемента

```
FUNCTION_BLOCK RELAY_HYSTERESIS
VAR_INPUT
   IN: REAL;                  // Входной сигнал
   ENABLE: BOOL := TRUE;          // Активация блока
   Setpoint_ON: REAL := 50.0;      // Порог включения
   Setpoint_OFF: REAL := 30.0;     // Порог выключения
   ManualMode: BOOL := FALSE;      // Ручной режим
   ManualValue: BOOL := FALSE;     // Ручное значение выхода
   Reset: BOOL := FALSE;          // Сброс состояния
END_VAR

VAR_OUTPUT
   OUT: BOOL;                 // Выходное значение
   STATE: INT := 0;              // Текущее состояние (0-OFF, 1-ON)
END_VAR

VAR
   LastState: BOOL := FALSE;       // Предыдущее состояние
END_VAR
```

```
METHOD UPDATE_STATE
// Обновление состояния реле
BEGIN
    IF Reset THEN
        OUT := FALSE;
        LastState := FALSE;
        STATE := 0;
        RETURN;
    END_IF;

    IF ManualMode THEN
        OUT := ManualValue;
        STATE := INT(ManualValue);
        RETURN;
    END_IF;

    IF IN >= Setpoint_ON THEN
        OUT := TRUE;
        LastState := TRUE;
        STATE := 1;
    ELSIF IN <= Setpoint_OFF THEN
        OUT := FALSE;
        LastState := FALSE;
        STATE := 0;
    ELSE
        // Зона гистерезиса - сохраняем предыдущее состояние
        OUT := LastState;
        STATE := INT(LastState);
    END_IF;
END_METHOD
```

```
BEGIN
   IF ENABLE THEN
      UPDATE_STATE();
   ELSE
      OUT := FALSE;
      STATE := 0;
   END_IF;
END_FUNCTION_BLOCK
```

2. Расширенная версия с дополнительными функциями

```
FUNCTION_BLOCK ADVANCED_HYSTERESIS_RELAY
VAR_INPUT
   IN: REAL;                    // Входной сигнал
   ENABLE: BOOL := TRUE;        // Активация блока
   Setpoint: REAL := 40.0;      // Центральная уставка
   Hysteresis: REAL := 5.0;     // Ширина гистерезиса
   Inverted: BOOL := FALSE;     // Инвертирование выхода
   ManualMode: BOOL := FALSE;   // Ручной режим
   ManualValue: BOOL := FALSE;  // Ручное значение
   Reset: BOOL := FALSE;        // Сброс состояния
END_VAR

VAR_OUTPUT
   OUT: BOOL;                   // Выходное значение
   STATE: INT := 0;             // Текущее состояние
   Setpoint_ON: REAL;           // Автоматически вычисляемый порог включения
   Setpoint_OFF: REAL;          // Автоматически вычисляемый порог выключения
END_VAR

VAR
   LastState: BOOL := FALSE;    // Предыдущее состояние
```

```
    Initialized: BOOL := FALSE;     // Флаг инициализации
END_VAR

METHOD CALCULATE_THRESHOLDS
// Вычисление порогов с учетом гистерезиса
BEGIN
    Setpoint_ON := Setpoint + Hysteresis/2;
    Setpoint_OFF := Setpoint - Hysteresis/2;
END_METHOD

METHOD UPDATE_STATE
// Обновление состояния реле
BEGIN
    IF NOT Initialized OR Reset THEN
        CALCULATE_THRESHOLDS();
        LastState := FALSE;
        Initialized := TRUE;
    END_IF;

    IF ManualMode THEN
        OUT := ManualValue XOR Inverted;
        STATE := INT(ManualValue);
        RETURN;
    END_IF;

    IF IN >= Setpoint_ON THEN
        LastState := TRUE;
    ELSIF IN <= Setpoint_OFF THEN
        LastState := FALSE;
    END_IF;
```

```
    OUT := LastState XOR Inverted;
    STATE := INT(LastState);
END_METHOD

BEGIN
    IF ENABLE THEN
        UPDATE_STATE();
    ELSE
        OUT := FALSE XOR Inverted;
        STATE := 0;
    END_IF;
END_FUNCTION_BLOCK
```

Примеры использования

1. Управление температурой

```
PROGRAM TEMP_CONTROL
VAR
    TempRelay: RELAY_HYSTERESIS;
    Temperature: REAL;
    HeaterOn: BOOL;
END_VAR

// Настройка параметров
TempRelay(
    Setpoint_ON := 75.0,    // Включаем нагрев при 75°C
    Setpoint_OFF := 70.0,   // Выключаем при 70°C
    ENABLE := TRUE
);

// Основной цикл
TempRelay.IN := Temperature;
```

```
TempRelay();
HeaterOn := TempRelay.OUT;
```

2. Система контроля уровня с инверсией

```
PROGRAM LEVEL_CONTROL
VAR
   LevelSwitch: ADVANCED_HYSTERESIS_RELAY;
   Level: REAL;
   PumpOn: BOOL;
END_VAR

// Конфигурация
LevelSwitch(
   Setpoint := 50.0,       // Средний уровень
   Hysteresis := 10.0,     // Ширина зоны гистерезиса
   Inverted := TRUE,       // Инверсия выхода
   ENABLE := TRUE
);

// Рабочий цикл
LevelSwitch.IN := Level;
LevelSwitch();
PumpOn := LevelSwitch.OUT; // TRUE когда уровень < 45, FALSE когда > 55
```

Дополнительные модификации

1. Релейный элемент с задержкой переключения

```
FUNCTION_BLOCK DELAYED_HYSTERESIS_RELAY
VAR_INPUT
   IN: REAL;
   Setpoint_ON: REAL := 50.0;
   Setpoint_OFF: REAL := 30.0;
```

```
    Delay_ON: TIME := T#5s;     // Задержка включения
    Delay_OFF: TIME := T#3s;    // Задержка выключения
END_VAR

VAR_OUTPUT
    OUT: BOOL;
END_VAR

VAR
    LastState: BOOL;
    Timer_ON: TON;
    Timer_OFF: TON;
END_VAR

BEGIN
    IF IN >= Setpoint_ON THEN
        Timer_ON(IN := TRUE, PT := Delay_ON);
        IF Timer_ON.Q THEN
            OUT := TRUE;
            LastState := TRUE;
        END_IF;
        Timer_OFF(IN := FALSE);
    ELSIF IN <= Setpoint_OFF THEN
        Timer_OFF(IN := TRUE, PT := Delay_OFF);
        IF Timer_OFF.Q THEN
            OUT := FALSE;
            LastState := FALSE;
        END_IF;
        Timer_ON(IN := FALSE);
    ELSE
        OUT := LastState;
```

```
    Timer_ON(IN := FALSE);
    Timer_OFF(IN := FALSE);
  END_IF;
END_FUNCTION_BLOCK
```

Ключевые особенности реализации

1. **Гибкость настройки**:
    - Возможность задания отдельных порогов ON/OFF;
    - Либо центральной уставки с гистерезисом;
    - Поддержка инверсии выхода.
2. **Защитные функции**:
    - Ручной режим управления;
    - Принудительный сброс состояния;
    - Блокировка работы (ENABLE).
3. **Диагностика**:
    - Выход текущего состояния;
    - Отображение расчетных порогов.
4. **Дополнительные возможности**:
    - Задержки переключения;
    - Адаптивные пороги;
    - Расширенная логика работы.

Рекомендации по применению

1. **Выбор параметров**:
    - Гистерезис должен превышать уровень шумов сигнала;
    - Для температурных процессов типичный гистерезис 2-5°C;
    - Для давлений 0.2-1 бар в зависимости от системы.
2. **Защита от дребезга**:
    - При работе с дискретными датчиками добавьте фильтрацию;
    - Используйте задержки переключения как в модифицированной версии.
3. **Интеграция в системы**:

- Комбинируйте с таймерами для минимального времени работы;
- Используйте как элемент защиты в каскадных системах.

4. **Отладка**:
   - Визуализируйте переключения на графиках;
   - Мониторьте текущие пороги срабатывания.

Релейный элемент с гистерезисом является важным компонентом в системах автоматизации, обеспечивая устойчивое переключение исполнительных механизмов и защиту от частых срабатываний при колебаниях сигнала около уставки.

Задача 14. Разработать приложение на языке ST, реализующее лимитер с плавным ограничением

Лимитер с плавным ограничением (smooth limiter) обеспечивает:

- Ограничение выходного сигнала по верхнему и нижнему уровням;
- Плавный переход при достижении границ (в отличие от жесткого ограничения);
- Защиту исполнительных механизмов от резких изменений сигнала.

Полная реализация на языке ST

1. Базовая версия лимитера

```
FUNCTION_BLOCK SMOOTH_LIMITER
VAR_INPUT
   IN: REAL;                 // Входной сигнал
   ENABLE: BOOL := TRUE;       // Активация блока
   MAX_LIM: REAL := 100.0;     // Верхний предел
   MIN_LIM: REAL := 0.0;       // Нижний предел
   SMOOTH_FACTOR: REAL := 0.1; // Коэффициент плавности (0..1)
   RESET: BOOL := FALSE;       // Сброс состояния
END_VAR

VAR_OUTPUT
   OUT: REAL;                // Выходной сигнал
   LIMITED: BOOL := FALSE;     // Флаг ограничения
```

```
END_VAR

VAR
    LastOutput: REAL := 0.0;    // Предыдущее значение выхода
    Initialized: BOOL := FALSE; // Флаг инициализации
END_VAR

METHOD APPLY_LIMIT: REAL
VAR_INPUT
    input_val: REAL;
END_VAR
BEGIN
    // Плавное ограничение
    IF input_val > MAX_LIM THEN
        RETURN LastOutput + (MAX_LIM - LastOutput) * SMOOTH_FACTOR;
    ELSIF input_val < MIN_LIM THEN
        RETURN LastOutput + (MIN_LIM - LastOutput) * SMOOTH_FACTOR;
    ELSE
        RETURN input_val;
    END_IF;
END_METHOD

METHOD UPDATE
BEGIN
    IF RESET OR NOT Initialized THEN
        LastOutput := IN;
        LIMITED := FALSE;
        Initialized := TRUE;
    END_IF;

    IF ENABLE THEN
```

```
        OUT := APPLY_LIMIT(IN);
        LIMITED := (OUT <> IN);
        LastOutput := OUT;
    ELSE
        OUT := IN;
        LIMITED := FALSE;
    END_IF;
END_METHOD

BEGIN
    UPDATE();
END_FUNCTION_BLOCK
```

2. Расширенная версия с динамическими параметрами

```
FUNCTION_BLOCK ADVANCED_SMOOTH_LIMITER
VAR_INPUT
    IN: REAL;                  // Входной сигнал
    ENABLE: BOOL := TRUE;      // Активация блока
    MAX_LIM: REAL := 100.0;    // Верхний предел
    MIN_LIM: REAL := 0.0;      // Нижний предел
    RISE_TIME: TIME := T#2s;   // Время выхода на максимум
    FALL_TIME: TIME := T#2s;   // Время возврата к минимуму
    TS: REAL := 0.1;           // Время дискретизации [сек]
    RESET: BOOL := FALSE;      // Сброс состояния
END_VAR

VAR_OUTPUT
    OUT: REAL;                 // Выходной сигнал
    AT_MAX: BOOL := FALSE;     // Флаг верхнего предела
    AT_MIN: BOOL := FALSE;     // Флаг нижнего предела
END_VAR
```

```
VAR
    CurrentValue: REAL := 0.0;
    RiseRate: REAL;
    FallRate: REAL;
    Initialized: BOOL := FALSE;
END_VAR

METHOD CALCULATE_RATES
// Расчет скоростей изменения
BEGIN
    IF RISE_TIME > T#0s THEN
        RiseRate := (MAX_LIM - MIN_LIM) / (TIME_TO_REAL(RISE_TIME)/TS);
    ELSE
        RiseRate := 0.0;
    END_IF;

    IF FALL_TIME > T#0s THEN
        FallRate := (MAX_LIM - MIN_LIM) / (TIME_TO_REAL(FALL_TIME)/TS);
    ELSE
        FallRate := 0.0;
    END_IF;
END_METHOD

METHOD UPDATE_VALUE
BEGIN
    IF IN > CurrentValue THEN
        // Рост значения
        CurrentValue := CurrentValue + RiseRate;
        IF CurrentValue > IN THEN CurrentValue := IN; END_IF;
    ELSIF IN < CurrentValue THEN
```

```
        // Уменьшение значения
        CurrentValue := CurrentValue - FallRate;
        IF CurrentValue < IN THEN CurrentValue := IN; END_IF;
    END_IF;

    // Применение жестких границ
    IF CurrentValue > MAX_LIM THEN CurrentValue := MAX_LIM; END_IF;
    IF CurrentValue < MIN_LIM THEN CurrentValue := MIN_LIM; END_IF;

    OUT := CurrentValue;
    AT_MAX := (CurrentValue >= MAX_LIM);
    AT_MIN := (CurrentValue <= MIN_LIM);
END_METHOD

BEGIN
    IF RESET OR NOT Initialized THEN
        CurrentValue := IN;
        CALCULATE_RATES();
        Initialized := TRUE;
    END_IF;

    IF ENABLE THEN
        UPDATE_VALUE();
    ELSE
        OUT := IN;
        AT_MAX := FALSE;
        AT_MIN := FALSE;
    END_IF;
END_FUNCTION_BLOCK
```

Примеры использования

1. Ограничение управляющего сигнала

```
PROGRAM MOTOR_CONTROL
VAR
   SpeedLimiter: SMOOTH_LIMITER;
   TargetSpeed: REAL;
   ActualSpeed: REAL;
END_VAR

// Настройка лимитера
SpeedLimiter(
   MAX_LIM := 3000.0,      // Максимальные обороты
   MIN_LIM := 0.0,         // Минимальные обороты
   SMOOTH_FACTOR := 0.05,  // Плавность ограничения
   ENABLE := TRUE
);

// Основной цикл управления
SpeedLimiter.IN := TargetSpeed;
SpeedLimiter();
ActualSpeed := SpeedLimiter.OUT;
```

2. Защита температурного режима

```
PROGRAM TEMP_PROTECTION
VAR
   TempLimiter: ADVANCED_SMOOTH_LIMITER;
   RequiredTemp: REAL;
   AllowedTemp: REAL;
END_VAR

// Конфигурация лимитера
TempLimiter(
```

```
    MAX_LIM := 120.0,       // Максимальная температура
    MIN_LIM := 20.0,        // Минимальная температура
    RISE_TIME := T#30s,     // Время разогрева
    FALL_TIME := T#60s,     // Время охлаждения
    TS := 0.2               // Период обновления
);

// Рабочий цикл
TempLimiter.IN := RequiredTemp;
TempLimiter();
AllowedTemp := TempLimiter.OUT;
```

Ключевые особенности реализации

1. **Гибкость ограничения**:
    - Плавное приближение к границам;
    - Раздельная настройка скорости роста и спада;
    - Возможность динамического изменения параметров.
2. **Защитные функции**:
    - Жесткое ограничение при выходе за границы;
    - Индикация достижения пределов;
    - Блокировка работы.
3. **Диагностика**:
    - Флаги достижения границ;
    - Возможность мониторинга текущего состояния.
4. **Оптимизация**:
    - Минимальные вычислительные затраты;
    - Учет времени дискретизации.

Рекомендации по применению

1. **Выбор параметров**:
    - Для систем с инерцией (температура) используйте большие времена плавности;

- В быстродействующих системах (скорость) уменьшайте коэффициент плавности.

2. **Интеграция в системы**:
   - Размещайте лимитер после регулятора, но перед исполнительным механизмом;
   - Комбинируйте с другими блоками защиты.
3. **Отладка**:
   - Начинайте с жесткого ограничения (SMOOTH_FACTOR = 1);
   - Постепенно уменьшайте коэффициент до достижения нужной плавности.
4. **Безопасность**:
   - Всегда устанавливайте разумные пределы;
   - Реализуйте дополнительные защитные алгоритмы.

Дополнительные модификации

1. Лимитер с адаптивными пределами

```
FUNCTION_BLOCK ADAPTIVE_LIMITER
VAR_INPUT
   IN: REAL;
   DYNAMIC_MAX: REAL;  // Динамический верхний предел
   DYNAMIC_MIN: REAL;   // Динамический нижний предел
   RATE_LIMIT: REAL;    // Максимальная скорость изменения
END_VAR

VAR_OUTPUT
   OUT: REAL;
END_VAR

VAR
   LastValue: REAL;
END_VAR

BEGIN
```

```
    // Ограничение скорости изменения
    IF (IN - LastValue) > RATE_LIMIT THEN
        OUT := LastValue + RATE_LIMIT;
    ELSIF (IN - LastValue) < -RATE_LIMIT THEN
        OUT := LastValue - RATE_LIMIT;
    ELSE
        OUT := IN;
    END_IF;
    // Применение динамических границ
    IF OUT > DYNAMIC_MAX THEN OUT := DYNAMIC_MAX; END_IF;
    IF OUT < DYNAMIC_MIN THEN OUT := DYNAMIC_MIN; END_IF;
    LastValue := OUT;
END_FUNCTION_BLOCK
```

Данная реализация лимитера с плавным ограничением обеспечивает безопасное управление технологическими параметрами, предотвращая резкие изменения сигналов и защищая оборудование от перегрузок.

Задача 15. Разработать приложение на языке ST фильтр низких частот (ФНЧ)

Фильтр низких частот (ФНЧ) пропускает низкочастотные составляющие сигнала и подавляет высокочастотные. Основные параметры:

- **Частота среза (Fc)** - граничная частота в герцах
- **Постоянная времени (T)** - $T = 1/(2\pi Fc)$
- **Коэффициент сглаживания (α)** - определяет степень фильтрации

Полная реализация на языке ST

1. Базовый вариант (экспоненциальное сглаживание)

```
FUNCTION_BLOCK LOW_PASS_FILTER
VAR_INPUT
    IN: REAL;                // Входной сигнал
    ENABLE: BOOL := TRUE;    // Активация блока
    FC: REAL := 1.0;         // Частота среза [Гц]
```

```
    TS: REAL := 0.1;            // Время дискретизации [сек]
    RESET: BOOL := FALSE;       // Сброс состояния
END_VAR

VAR_OUTPUT
    OUT: REAL;                  // Отфильтрованный сигнал
END_VAR

VAR
    Alpha: REAL;                // Коэффициент сглаживания
    PrevOut: REAL := 0.0;       // Предыдущее выходное значение
    Initialized: BOOL := FALSE; // Флаг инициализации
END_VAR

METHOD CALCULATE_ALPHA
// Расчет коэффициента сглаживания
BEGIN
    IF FC <= 0 OR TS <= 0 THEN
        Alpha := 1.0; // Без фильтрации при некорректных параметрах
    ELSE
        Alpha := TS / (TS + 1.0/(2.0*3.14159*FC));
    END_IF;
END_METHOD

METHOD UPDATE_FILTER
// Обновление фильтра
BEGIN
    OUT := Alpha * IN + (1 - Alpha) * PrevOut;
    PrevOut := OUT;
END_METHOD
```

```
BEGIN
    IF RESET OR NOT Initialized THEN
        CALCULATE_ALPHA();
        PrevOut := IN;
        OUT := IN;
        Initialized := TRUE;
    END_IF;

    IF ENABLE THEN
        CALCULATE_ALPHA(); // Для динамического изменения Fc
        UPDATE_FILTER();
    ELSE
        OUT := IN; // Байпас при отключении
    END_IF;
END_FUNCTION_BLOCK
```

2. Усовершенствованный вариант (Биквадратный ФНЧ)

```
FUNCTION_BLOCK BIQUAD_LPF
VAR_INPUT
    IN: REAL;                // Входной сигнал
    ENABLE: BOOL := TRUE;       // Активация блока
    FC: REAL := 1.0;            // Частота среза [Гц]
    Q: REAL := 0.707;           // Добротность (0.5-1.0)
    FS: REAL := 10.0;           // Частота дискретизации [Гц]
    RESET: BOOL := FALSE;       // Сброс состояния
END_VAR

VAR_OUTPUT
    OUT: REAL;                // Отфильтрованный сигнал
END_VAR
```

```
VAR
    // Коэффициенты фильтра
    A0, A1, A2: REAL;
    B1, B2: REAL;

    // Состояния фильтра
    X1, X2: REAL := 0.0;       // Предыдущие входы
    Y1, Y2: REAL := 0.0;       // Предыдущие выходы

    Initialized: BOOL := FALSE;
END_VAR

METHOD CALCULATE_COEFFS
// Расчет коэффициентов биквадратного фильтра
VAR
    omega, sn, cs, alpha: REAL;
    norm: REAL;
END_VAR
BEGIN
    omega := 2.0 * 3.14159 * FC / FS;
    sn := SIN(omega);
    cs := COS(omega);
    alpha := sn / (2.0 * Q);

    norm := 1.0 / (1.0 + alpha);

    A0 := (1.0 - cs) / 2.0 * norm;
    A1 := (1.0 - cs) * norm;
    A2 := A0;
    B1 := 2.0 * cs * norm;
    B2 := (1.0 - alpha) * norm;
```

```
END_METHOD

METHOD UPDATE_FILTER
// Применение фильтра
BEGIN
    OUT := A0*IN + A1*X1 + A2*X2 - B1*Y1 - B2*Y2;

    // Обновление состояний
    X2 := X1;
    X1 := IN;
    Y2 := Y1;
    Y1 := OUT;
END_METHOD

BEGIN
    IF RESET OR NOT Initialized THEN
        CALCULATE_COEFFS();
        X1 := 0.0; X2 := 0.0;
        Y1 := 0.0; Y2 := 0.0;
        OUT := IN;
        Initialized := TRUE;
    END_IF;

    IF ENABLE THEN
        CALCULATE_COEFFS(); // Для динамического изменения параметров
        UPDATE_FILTER();
    ELSE
        OUT := IN;
    END_IF;
END_FUNCTION_BLOCK
```

Примеры использования

1. Фильтрация сигнала датчика

```
PROGRAM SENSOR_FILTERING
VAR
   TempFilter: LOW_PASS_FILTER;
   RawTemp: REAL;          // Сырое значение с датчика
   FilteredTemp: REAL;     // Отфильтрованное значение
END_VAR

// Настройка фильтра (частота среза 0.5 Гц)
TempFilter(
   FC := 0.5,      // Частота среза 0.5 Гц
   TS := 0.1,      // Период обновления 100 мс
   ENABLE := TRUE
);

// Основной цикл обработки
TempFilter.IN := RawTemp;
TempFilter();
FilteredTemp := TempFilter.OUT;
```

2. Продвинутая фильтрация в системе управления

```
PROGRAM MOTION_CONTROL
VAR
   SpeedFilter: BIQUAD_LPF;
   MeasuredSpeed: REAL;
   FilteredSpeed: REAL;
END_VAR

// Настройка биквадратного фильтра
SpeedFilter(
```

```
    FC := 10.0,     // Частота среза 10 Гц
    Q := 0.8,       // Добротность
    FS := 100.0,    // Частота дискретизации 100 Гц
    ENABLE := TRUE
);

// Рабочий цикл
SpeedFilter.IN := MeasuredSpeed;
SpeedFilter();
FilteredSpeed := SpeedFilter.OUT;
```

Ключевые особенности реализации

1. **Гибкость настройки**:
    - Возможность изменения частоты среза;
    - Поддержка динамического обновления параметров;
    - Различные алгоритмы для разных требований.
2. **Защитные функции**:
    - Проверка корректности параметров;
    - Байпас при отключении;
    - Сброс состояния фильтра.
3. **Оптимизация**:
    - Минимальные вычислительные затраты (базовая версия);
    - Высокое качество фильтрации (биквадратная версия).
4. **Практическое применение**:
    - Подавление шумов датчиков;
    - Сглаживание управляющих сигналов;
    - Предварительная обработка сигналов.

Рекомендации по выбору и настройке

1. **Для простых применений**:
    - Используйте базовый вариант;

- Начните с Fc = 1 Гц, регулируйте по результатам.

2. **Для точных систем**:
   - Применяйте биквадратный фильтр;
   - Оптимальная добротность Q = 0.707 (Баттерворт).
3. **Параметры дискретизации**:
   - TS должно соответствовать реальному периоду вызова;
   - FS должна быть как минимум в 2 раза выше Fc.
4. **Отладка**:
   - Анализируйте АЧХ фильтра;
   - Проверяйте фазовые сдвиги в вашей системе;
   - Сравнивайте фильтрованные и нефильтрованные сигналы.

Дополнительные модификации

1. Адаптивный ФНЧ с автоматической настройкой

```
FUNCTION_BLOCK ADAPTIVE_LPF
VAR_INPUT
   IN: REAL;
   NOISE_LEVEL: REAL; // Оценка уровня шума
   MIN_FC: REAL := 0.1; // Минимальная частота среза
   MAX_FC: REAL := 10.0; // Максимальная частота среза
   TS: REAL := 0.1;
END_VAR

VAR_OUTPUT
   OUT: REAL;
   CURRENT_FC: REAL;
END_VAR

VAR
   Filter: LOW_PASS_FILTER;
END_VAR
```

```
BEGIN
   // Адаптивная настройка частоты среза
   CURRENT_FC := MAX(MIN_FC, MIN(MAX_FC, 1.0/NOISE_LEVEL));

   Filter(
      FC := CURRENT_FC,
      TS := TS,
      IN := IN,
      ENABLE := TRUE
   );

   OUT := Filter.OUT;
END_FUNCTION_BLOCK
```

Данная реализация фильтра низких частот обеспечивает эффективное подавление высокочастотных помех при сохранении полезного сигнала, что делает его незаменимым инструментом в системах промышленной автоматизации.

Задача 16. Разработать приложение на языке ST, реализующее управление двигателем с ПИД-регулятором

```
FUNCTION_BLOCK MOTOR_PID_CONTROL
VAR_INPUT
   // Сигналы управления
   Enable: BOOL := FALSE;          // Общее разрешение работы
   Start: BOOL := FALSE;           // Команда пуска
   Stop: BOOL := FALSE;            // Команда останова
   EmergencyStop: BOOL := FALSE;   // Аварийный останов

   // Задание управления
   Setpoint: REAL := 0.0;          // Уставка (об/мин или другая величина)
   MaxSpeed: REAL := 3000.0;       // Максимальная скорость
```

```
    // Обратная связь
    ActualSpeed: REAL := 0.0;        // Текущая скорость с датчика
    SpeedValid: BOOL := FALSE;       // Валидность сигнала скорости

    // Параметры ПИД-регулятора
    Kp: REAL := 1.0;                 // Пропорциональный коэффициент
    Ti: REAL := 0.0;                 // Интегральное время (0 - отключить I)
    Td: REAL := 0.0;                 // Дифференциальное время (0 - отключить D)
    Ts: REAL := 0.1;                 // Время дискретизации [сек]

    // Ограничения
    MaxOutput: REAL := 100.0;        // Максимальный выход регулятора (%)
    MinOutput: REAL := 0.0;          // Минимальный выход регулятора (%)
    MaxOutputStep: REAL := 5.0;      // Макс. изменение выхода за цикл (%)
END_VAR

VAR_OUTPUT
    // Сигналы на исполнительные устройства
    Output: REAL := 0.0;             // Выходной сигнал управления (0-100%)
    Direction: INT := 0;             // Направление (1-вперед, -1-назад, 0-стоп)
    Ready: BOOL := FALSE;            // Флаг готовности системы
    Running: BOOL := FALSE;          // Флаг работы двигателя
    Alarm: WORD := 0;                // Коды аварий

    // Диагностика
    ControlError: REAL := 0.0;       // Текущая ошибка регулирования
    P_Term: REAL := 0.0;             // Пропорциональная составляющая
    I_Term: REAL := 0.0;             // Интегральная составляющая
    D_Term: REAL := 0.0;             // Дифференциальная составляющая
END_VAR
```

```
VAR
    // Внутренние переменные ПИД-регулятора
    PID: PID_Controller;            // Функциональный блок ПИД-регулятора
    SpeedFilter: LOW_PASS_FILTER;   // Фильтр сигнала скорости

    // Управляющие переменные
    ActiveSetpoint: REAL := 0.0;    // Текущая активная уставка
    RampGenerator: RAMP_GEN;        // Генератор линейного изменения уставки

    // Защитные переменные
    StartTimer: TON;                // Таймер плавного пуска
    StopTimer: TON;                 // Таймер плавного останова
    WatchdogTimer: TON;             // Контроль сигнала скорости
END_VAR

METHOD INITIALIZE
// Инициализация системы
BEGIN
    // Настройка ПИД-регулятора
    PID(
        Kp := Kp,
        Ti := Ti,
        Td := Td,
        Ts := Ts,
        MaxOutput := MaxOutput,
        MinOutput := MinOutput,
        MaxOutputStep := MaxOutputStep
    );

    // Настройка фильтра скорости
```

```
    SpeedFilter(
        FC := 5.0,   // Частота среза 5 Гц
        TS := Ts
    );

    // Настройка генератора рампы
    RampGenerator(
        RampTime := 10.0, // Время нарастания 10 сек
        Ts := Ts
    );

    // Сброс таймеров
    StartTimer(IN := FALSE);
    StopTimer(IN := FALSE);
    WatchdogTimer(IN := FALSE);

    Ready := TRUE;
    Alarm := 0;
END_METHOD

METHOD UPDATE_CONTROL
// Основной алгоритм управления
BEGIN
    // Фильтрация сигнала скорости
    SpeedFilter.IN := ActualSpeed;
    SpeedFilter();

    // Контроль валидности сигнала скорости
    WatchdogTimer(
        IN := SpeedValid,
        PT := T#2s
```

```
);

IF WatchdogTimer.Q THEN
    Alarm.0 := TRUE; // Потеря сигнала скорости
    EmergencyStop := TRUE;
ELSE
    Alarm.0 := FALSE;
END_IF;

// Обработка команд управления
IF EmergencyStop THEN
    Running := FALSE;
    ActiveSetpoint := 0.0;
    Output := 0.0;
    Direction := 0;
ELSIF Start AND NOT Running THEN
    // Плавный пуск
    RampGenerator.StartValue := 0.0;
    RampGenerator.EndValue := Setpoint;
    RampGenerator.Start := TRUE;
    Running := TRUE;
ELSIF Stop AND Running THEN
    // Плавный останов
    RampGenerator.StartValue := ActiveSetpoint;
    RampGenerator.EndValue := 0.0;
    RampGenerator.Start := TRUE;
    Running := FALSE;
END_IF;

// Генерация уставки
RampGenerator();
```

```
    ActiveSetpoint := RampGenerator.Out;

    // ПИД-регулирование
    IF Running THEN
        PID.Setpoint := ActiveSetpoint;
        PID.ProcessValue := SpeedFilter.OUT;
        PID();

        Output := PID.Output;
        Direction := INT(SIGN(Setpoint));

        // Сохранение составляющих для диагностики
        ControlError := PID.Error;
        P_Term := PID.P_Term;
        I_Term := PID.I_Term;
        D_Term := PID.D_Term;
    ELSE
        Output := 0.0;
        Direction := 0;
        PID.Reset := TRUE;
    END_IF;
END_METHOD

// Основной цикл управления
BEGIN
    IF NOT Enable THEN
        INITIALIZE();
        Ready := FALSE;
    ELSE
        UPDATE_CONTROL();
    END_IF;
```

```
END_FUNCTION_BLOCK
```

Дополнительные необходимые блоки

1. ПИД-регулятор (используемый в системе)

```
FUNCTION_BLOCK PID_Controller
VAR_INPUT
    Setpoint: REAL;
    ProcessValue: REAL;
    Kp: REAL := 1.0;
    Ti: REAL := 0.0;
    Td: REAL := 0.0;
    Ts: REAL := 0.1;
    MaxOutput: REAL := 100.0;
    MinOutput: REAL := 0.0;
    MaxOutputStep: REAL := 5.0;
    Reset: BOOL := FALSE;
END_VAR

VAR_OUTPUT
    Output: REAL;
    Error: REAL;
    P_Term: REAL;
    I_Term: REAL;
    D_Term: REAL;
END_VAR

VAR
    PrevError: REAL := 0.0;
    Integral: REAL := 0.0;
    PrevProcessValue: REAL := 0.0;
    PrevOutput: REAL := 0.0;
```

```
END_VAR

BEGIN
    Error := Setpoint - ProcessValue;

    // Пропорциональная составляющая
    P_Term := Kp * Error;

    // Интегральная составляющая
    IF Ti > 0 THEN
        Integral := Integral + Kp * Error * Ts / Ti;

        // Антивиндап
        IF Integral > (MaxOutput - P_Term) THEN
            Integral := MaxOutput - P_Term;
        ELSIF Integral < (MinOutput - P_Term) THEN
            Integral := MinOutput - P_Term;
        END_IF;
    ELSE
        Integral := 0.0;
    END_IF;
    I_Term := Integral;

    // Дифференциальная составляющая (по измерению)
    IF Td > 0 THEN
        D_Term := -Kp * Td * (ProcessValue - PrevProcessValue) / Ts;
    ELSE
        D_Term := 0.0;
    END_IF;

    // Суммирование составляющих
```

```
    Output := P_Term + I_Term + D_Term;

    // Ограничение скорости изменения
    IF (Output - PrevOutput) > MaxOutputStep THEN
        Output := PrevOutput + MaxOutputStep;
    ELSIF (Output - PrevOutput) < -MaxOutputStep THEN
        Output := PrevOutput - MaxOutputStep;
    END_IF;

    // Общие ограничения
    IF Output > MaxOutput THEN
        Output := MaxOutput;
    ELSIF Output < MinOutput THEN
        Output := MinOutput;
    END_IF;

    // Сохранение состояний
    PrevError := Error;
    PrevProcessValue := ProcessValue;
    PrevOutput := Output;

    IF Reset THEN
        Integral := 0.0;
        PrevError := 0.0;
        PrevProcessValue := ProcessValue;
        Output := 0.0;
    END_IF;
END_FUNCTION_BLOCK
```

2. Генератор линейного изменения (рампы)

```
FUNCTION_BLOCK RAMP_GEN
```

```
VAR_INPUT
    StartValue: REAL := 0.0;     // Начальное значение
    EndValue: REAL := 100.0;     // Конечное значение
    RampTime: REAL := 10.0;      // Время изменения [сек]
    Ts: REAL := 0.1;             // Время дискретизации [сек]
    Start: BOOL := FALSE;        // Команда начала изменения
    Reset: BOOL := FALSE;        // Сброс состояния
END_VAR

VAR_OUTPUT
    Out: REAL;                   // Текущее выходное значение
    Busy: BOOL := FALSE;         // Флаг выполнения
END_VAR

VAR
    CurrentValue: REAL := 0.0;
    Step: REAL := 0.0;
    Active: BOOL := FALSE;
END_VAR

BEGIN
    IF Reset THEN
        CurrentValue := StartValue;
        Out := StartValue;
        Busy := FALSE;
        Active := FALSE;
    ELSIF Start AND NOT Active THEN
        Step := (EndValue - StartValue) * Ts / RampTime;
        CurrentValue := StartValue;
        Active := TRUE;
        Busy := TRUE;
```

```
    END_IF;

    IF Active THEN
        IF ABS(EndValue - CurrentValue) > ABS(Step) THEN
            CurrentValue := CurrentValue + Step;
        ELSE
            CurrentValue := EndValue;
            Active := FALSE;
            Busy := FALSE;
        END_IF;
    END_IF;

    Out := CurrentValue;
END_FUNCTION_BLOCK
```

Пример использования системы

```
PROGRAM MAIN
VAR
    Motor1: MOTOR_PID_CONTROL;
    SpeedSetpoint: REAL := 1500.0;
    ActualSpeed: REAL;
    SpeedFeedbackOK: BOOL;
    ControlEnable: BOOL;
END_VAR

// Инициализация параметров управления
Motor1(
    MaxSpeed := 3000.0,
    Kp := 2.5,
    Ti := 0.5,
    Td := 0.1,
```

```
    Ts := 0.1
);

// Основной цикл управления
Motor1(
    Enable := ControlEnable,
    Setpoint := SpeedSetpoint,
    ActualSpeed := ActualSpeed,
    SpeedValid := SpeedFeedbackOK
);

// Где-то в логике управления:
IF StartButton THEN
    Motor1.Start := TRUE;
END_IF;

IF StopButton THEN
    Motor1.Stop := TRUE;
END_IF;

IF EmergencyCondition THEN
    Motor1.EmergencyStop := TRUE;
END_IF;
```

Ключевые особенности системы

1. **Полнофункциональное управление**:
    - Плавный пуск/останов;
    - Аварийная остановка;
    - Защита от потери обратной связи.
2. **Гибкая настройка**:
    - Регулируемые параметры ПИД-регулятора;

- Настройка времени рампы;
- Ограничения выходных сигналов.

3. **Диагностика**:
   - Мониторинг составляющих ПИД;
   - Коды аварий;
   - Статусы работы.
4. **Безопасность**:
   - Контроль валидности сигналов;
   - Ограничение скорости изменения;
   - Защита от перегрузок.

Рекомендации по настройке

1. **Настройка ПИД-регулятора**:
   - Сначала установите Ti=0 и Td=0, настройте Kp;
   - Затем добавьте интегральную составляющую (Ti);
   - В последнюю очередь - дифференциальную (Td).
2. **Параметры рампы**:
   - Время нарастания должно соответствовать инерции системы;
   - Для двигателей обычно 5-30 секунд.
3. **Фильтрация сигнала**:
   - Частота среза фильтра должна быть выше частоты регулирования;
   - Типичные значения 5-20 Гц для двигателей.
4. **Защитные функции**:
   - Установите разумные ограничения MaxOutput;
   - Настройте MaxOutputStep для предотвращения рывков.

Данная реализация обеспечивает надежное и гибкое управление двигателем с защитой от аварийных ситуаций и широкими возможностями диагностики.

Задача 17. Температурный регулятор с защитой на языке ST

Полная реализация температурного регулятора

```
FUNCTION_BLOCK TEMP_CONTROLLER
VAR_INPUT
   // Сигналы управления
   Enable: BOOL := FALSE;          // Общее разрешение работы
   Setpoint: REAL := 50.0;         // Уставка температуры (°C)
   ManualMode: BOOL := FALSE;      // Ручной режим
   ManualOutput: REAL := 0.0;      // Ручное значение выхода (%)

   // Обратная связь
   ActualTemp: REAL := 20.0;       // Текущая температура (°C)
   TempValid: BOOL := FALSE;       // Валидность сигнала температуры

   // Параметры ПИД-регулятора
   Kp: REAL := 5.0;                // Пропорциональный коэффициент
   Ti: REAL := 120.0;              // Интегральное время (сек)
   Td: REAL := 30.0;               // Дифференциальное время (сек)
   Ts: REAL := 1.0;                // Время дискретизации (сек)

   // Настройки защиты
   MaxTemp: REAL := 120.0;         // Максимальная температура (°C)
   TempHysteresis: REAL := 5.0;    // Гистерезис защиты (°C)
   MaxOutput: REAL := 100.0;       // Максимальный выход (%)
   MinOutput: REAL := 0.0;         // Минимальный выход (%)
   OutputRateLimit: REAL := 2.0;   // Макс. изменение выхода за цикл (%)

   // Сигналы безопасности
   OverrideCooling: BOOL := FALSE;  // Принудительное охлаждение
   ResetProtection: BOOL := FALSE;  // Сброс защит
```

```
END_VAR

VAR_OUTPUT
    // Управляющие сигналы
    HeaterOutput: REAL := 0.0;       // Управление нагревателем (%)
    CoolerOutput: REAL := 0.0;       // Управление охладителем (%)

    // Статус и диагностика
    Ready: BOOL := FALSE;            // Флаг готовности
    Running: BOOL := FALSE;          // Флаг работы
    Alarm: WORD := 0;                // Коды аварий (битовое поле)
    ControlError: REAL := 0.0;       // Текущая ошибка регулирования

    // Составляющие ПИД
    P_Term: REAL := 0.0;
    I_Term: REAL := 0.0;
    D_Term: REAL := 0.0;
END_VAR

VAR
    // Основные компоненты
    PID: PID_CONTROLLER;             // ПИД-регулятор
    TempFilter: LOW_PASS_FILTER;     // Фильтр температуры
    OutputLimiter: RATE_LIMITER;     // Ограничитель скорости изменения

    // Защитные механизмы
    OverTempTimer: TON;              // Таймер перегрева
    SensorWatchdog: TON;             // Контроль датчика
    CoolDownActive: BOOL := FALSE;   // Флаг активного охлаждения

    // Внутренние переменные
```

```
    FilteredTemp: REAL := 20.0;
    SafeSetpoint: REAL := 0.0;
    LastOutput: REAL := 0.0;
END_VAR

METHOD INITIALIZE
// Инициализация системы
BEGIN
    // Настройка ПИД-регулятора
    PID(
        Kp := Kp,
        Ti := Ti,
        Td := Td,
        Ts := Ts,
        MaxOutput := MaxOutput,
        MinOutput := MinOutput,
        MaxOutputStep := OutputRateLimit
    );

    // Настройка фильтра температуры
    TempFilter(
        FC := 0.1,  // Частота среза 0.1 Гц
        TS := Ts
    );

    // Настройка ограничителя
    OutputLimiter(
        MaxRate := OutputRateLimit,
        Ts := Ts
    );
```

```
    // Сброс таймеров
    OverTempTimer(IN := FALSE);
    SensorWatchdog(IN := FALSE);

    Ready := TRUE;
    Alarm := 0;
END_METHOD

METHOD UPDATE_PROTECTION
// Обновление защитных функций
BEGIN
    // Контроль валидности температуры
    SensorWatchdog(
        IN := TempValid,
        PT := T#10s
    );

    IF SensorWatchdog.Q THEN
        Alarm.0 := 1; // Ошибка датчика
        Running := FALSE;
    ELSE
        Alarm.0 := 0;
    END_IF;

    // Защита от перегрева
    IF FilteredTemp > MaxTemp THEN
        OverTempTimer(
            IN := TRUE,
            PT := T#30s  // Задержка перед аварией
        );
```

```
        IF OverTempTimer.Q THEN
            Alarm.1 := 1; // Перегрев
            Running := FALSE;
            CoolDownActive := TRUE;
        END_IF;
    ELSIF FilteredTemp < (MaxTemp - TempHysteresis) THEN
        OverTempTimer(IN := FALSE);
        Alarm.1 := 0;
        CoolDownActive := FALSE;
    END_IF;

    // Принудительное охлаждение
    IF OverrideCooling THEN
        CoolDownActive := TRUE;
        Running := FALSE;
    END_IF;

    // Сброс защит
    IF ResetProtection THEN
        Alarm := 0;
        CoolDownActive := FALSE;
        OverTempTimer(IN := FALSE);
    END_IF;
END_METHOD

METHOD UPDATE_CONTROL
// Основной алгоритм управления
BEGIN
    // Фильтрация температуры
    TempFilter.IN := ActualTemp;
    TempFilter();
```

```
FilteredTemp := TempFilter.OUT;

// Расчет безопасной уставки
SafeSetpoint := LIMIT(Setpoint, MinOutput, MaxTemp - 5.0);

IF Running AND NOT CoolDownActive THEN
    // Нормальный режим работы
    PID.Setpoint := SafeSetpoint;
    PID.ProcessValue := FilteredTemp;
    PID();

    // Ограничение скорости изменения
    OutputLimiter.IN := PID.Output;
    OutputLimiter();
    LastOutput := OutputLimiter.OUT;

    // Сохранение составляющих для диагностики
    ControlError := PID.Error;
    P_Term := PID.P_Term;
    I_Term := PID.I_Term;
    D_Term := PID.D_Term;
ELSE
    // Режим охлаждения или останов
    PID.Reset := TRUE;
    LastOutput := 0.0;
END_IF;

// Управление выходами
IF CoolDownActive THEN
    HeaterOutput := 0.0;
    CoolerOutput := 100.0; // Максимальное охлаждение
```

```
    ELSE
        HeaterOutput := LastOutput;
        CoolerOutput := 0.0;
    END_IF;

    // Ручной режим
    IF ManualMode THEN
        HeaterOutput := ManualOutput;
        CoolerOutput := 0.0;
    END_IF;
END_METHOD

// Основной цикл управления
BEGIN
    IF NOT Enable THEN
        INITIALIZE();
        Ready := FALSE;
    ELSE
        UPDATE_PROTECTION();
        UPDATE_CONTROL();
    END_IF;
END_FUNCTION_BLOCK
```

Дополнительные необходимые блоки

1. ПИД-регулятор (аналогичный предыдущему примеру)

2. Фильтр низких частот

```
FUNCTION_BLOCK LOW_PASS_FILTER
VAR_INPUT
    IN: REAL;
    FC: REAL := 1.0;  // Частота среза (Гц)
    TS: REAL := 1.0;  // Время дискретизации (сек)
```

```
END_VAR

VAR_OUTPUT
   OUT: REAL;
END_VAR

VAR
   Alpha: REAL;
   LastOut: REAL;
END_VAR

BEGIN
   Alpha := TS / (TS + 1.0/(2.0*3.14159*FC));
   OUT := Alpha * IN + (1 - Alpha) * LastOut;
   LastOut := OUT;
END_FUNCTION_BLOCK
```

3. Ограничитель скорости изменения

```
FUNCTION_BLOCK RATE_LIMITER
VAR_INPUT
   IN: REAL;
   MaxRate: REAL := 1.0; // Макс. скорость (%/сек)
   Ts: REAL := 1.0;      // Время дискретизации (сек)
END_VAR

VAR_OUTPUT
   OUT: REAL;
END_VAR

VAR
   LastValue: REAL;
```

```
    MaxStep: REAL;
END_VAR

BEGIN
    MaxStep := MaxRate * Ts;

    IF (IN - LastValue) > MaxStep THEN
        OUT := LastValue + MaxStep;
    ELSIF (IN - LastValue) < -MaxStep THEN
        OUT := LastValue - MaxStep;
    ELSE
        OUT := IN;
    END_IF;
    LastValue := OUT;
END_FUNCTION_BLOCK
```

Пример использования

```
PROGRAM MAIN
VAR
    OvenController: TEMP_CONTROLLER;
    TempSensor: REAL;
    TempSensorOK: BOOL;
    SetTemperature: REAL := 80.0;
    SystemEnable: BOOL;
END_VAR

// Инициализация контроллера
OvenController(
    MaxTemp := 150.0,
    TempHysteresis := 10.0,
    Kp := 8.0,
```

```
    Ti := 180.0,
    Td := 40.0
);

// Основной цикл управления
OvenController(
    Enable := SystemEnable,
    Setpoint := SetTemperature,
    ActualTemp := TempSensor,
    TempValid := TempSensorOK
);

// Внешняя логика управления
IF StartButton THEN
    OvenController.Running := TRUE;
END_IF;

IF EmergencyButton THEN
    OvenController.OverrideCooling := TRUE;
END_IF;
```

Ключевые особенности системы

1. **Комплексная защита**:
    - Контроль перегрева с гистерезисом;
    - Мониторинг исправности датчика;
    - Принудительное охлаждение;
    - Ограничение скорости изменения мощности.
2. **Гибкое управление**:
    - Автоматический и ручной режимы;
    - Настройка параметров ПИД-регулятора;
    - Адаптивная фильтрация сигнала.

3. **Диагностика**:
    - Детальный мониторинг работы ПИД-регулятора;
    - Коды аварий с битовой маской;
    - Контроль всех критических параметров.
4. **Безопасность**:
    - Автоматическое ограничение уставки;
    - Защита от "разгона" интегральной составляющей;
    - Плавное изменение выходных сигналов.

Рекомендации по настройке

1. **Параметры ПИД-регулятора**:
    - Для термических процессов обычно Ti >> Td;
    - Начните с Kp = 5-10% на °C ошибки;
    - Время интегрирования 2-5 минут (120-300 сек).
2. **Настройка защиты**:
    - MaxTemp должна быть ниже опасного уровня;
    - Гистерезис 5-10% от MaxTemp;
    - Время реакции на перегрев 20-60 сек.
3. **Фильтрация сигнала**:
    - Частота среза 0.1-0.5 Гц для температур;
    - Увеличивайте при медленных процессах.
4. **Ограничения**:
    - OutputRateLimit 1-5%/сек для нагревателей;
    - Уменьшайте для инерционных систем.

Данная реализация обеспечивает надежное и безопасное управление температурой с комплексной защитой и широкими возможностями диагностики.

Задача 18. Разработать приложение на языке ST, реализующее балансировку двух резервуаров

Полная система управления

```
FUNCTION_BLOCK TANK_BALANCER
VAR_INPUT
   // Сигналы управления
   Enable: BOOL := FALSE;          // Общее разрешение работы
   AutoMode: BOOL := TRUE;         // Автоматический режим балансировки
   ManualValveCmd: REAL := 0.0;    // Ручное управление клапаном (-100..100%)

   // Обратная связь
   Level1: REAL := 0.0;            // Уровень в резервуаре 1 (%)
   Level2: REAL := 0.0;            // Уровень в резервуаре 2 (%)
   LevelsValid: BOOL := FALSE;     // Валидность сигналов уровня

   // Параметры системы
   BalanceSetpoint: REAL := 50.0;  // Уставка балансировки (%)
   MaxFlowRate: REAL := 10.0;      // Макс. скорость перекачки (%/сек)
   DeadBand: REAL := 2.0;          // Зона нечувствительности (%)
   Kp: REAL := 2.0;                // Коэффициент П-регулятора
   Ts: REAL := 1.0;                // Время дискретизации (сек)

   // Защитные параметры
   MaxLevel: REAL := 95.0;         // Максимальный уровень (%)
   MinLevel: REAL := 5.0;          // Минимальный уровень (%)
   EmergencyStop: BOOL := FALSE;   // Аварийный останов
END_VAR

VAR_OUTPUT
   // Управляющие сигналы
   ValveOpen: REAL := 0.0;         // Открытие клапана на перекачку (0..100%)
```

```
    ValveDirection: INT := 0;        // Направление (1: 1→2, -1: 2→1, 0: стоп)
    PumpSpeed: REAL := 0.0;          // Скорость насоса (%)

    // Статус и диагностика
    Ready: BOOL := FALSE;            // Флаг готовности
    Balanced: BOOL := FALSE;         // Флаг достижения баланса
    Alarm: WORD := 0;                // Коды аварий
    LevelDifference: REAL := 0.0;    // Текущая разница уровней
END_VAR

VAR
    // Компоненты системы
    BalancePID: PID_CONTROLLER;      // ПИД-регулятор балансировки
    FlowLimiter: RATE_LIMITER;       // Ограничитель скорости потока
    LevelFilter1: LOW_PASS_FILTER;   // Фильтр уровня 1
    LevelFilter2: LOW_PASS_FILTER;   // Фильтр уровня 2

    // Защитные механизмы
    LevelWatchdog: TON;              // Контроль сигналов уровня
    HighLevelTimer: TON;             // Таймер высокого уровня
    LowLevelTimer: TON;              // Таймер низкого уровня

    // Внутренние переменные
    FilteredLevel1: REAL := 0.0;
    FilteredLevel2: REAL := 0.0;
    SafeValveCmd: REAL := 0.0;
    LastDirection: INT := 0;
END_VAR

METHOD INITIALIZE
// Инициализация системы
```

```
BEGIN
    // Настройка ПИД-регулятора (используем только P-составляющую)
    BalancePID(
        Kp := Kp,
        Ti := 0.0,  // Отключаем интегральную составляющую
        Td := 0.0,  // Отключаем дифференциальную
        Ts := Ts,
        MaxOutput := 100.0,
        MinOutput := -100.0,
        MaxOutputStep := MaxFlowRate
    );

    // Настройка фильтров уровня
    LevelFilter1(FC := 0.05, TS := Ts);
    LevelFilter2(FC := 0.05, TS := Ts);

    // Настройка ограничителя потока
    FlowLimiter(MaxRate := MaxFlowRate, Ts := Ts);

    // Сброс таймеров
    LevelWatchdog(IN := FALSE);
    HighLevelTimer(IN := FALSE);
    LowLevelTimer(IN := FALSE);

    Ready := TRUE;
    Alarm := 0;
END_METHOD

METHOD UPDATE_PROTECTION
// Обновление защитных функций
BEGIN
```

```
// Контроль валидности уровней
LevelWatchdog(
    IN := LevelsValid,
    PT := T#15s
);

IF LevelWatchdog.Q THEN
    Alarm.0 := 1; // Ошибка датчиков уровня
    AutoMode := FALSE;
ELSE
    Alarm.0 := 0;
END_IF;

// Защита от переполнения
IF (FilteredLevel1 > MaxLevel) OR (FilteredLevel2 > MaxLevel) THEN
    HighLevelTimer(
        IN := TRUE,
        PT := T#30s
    );

    IF HighLevelTimer.Q THEN
        Alarm.1 := 1; // Авария высокого уровня
        AutoMode := FALSE;
    END_IF;
ELSE
    HighLevelTimer(IN := FALSE);
    Alarm.1 := 0;
END_IF;

// Защита от опустошения
IF (FilteredLevel1 < MinLevel) OR (FilteredLevel2 < MinLevel) THEN
```

```
        LowLevelTimer(
            IN := TRUE,
            PT := T#30s
        );

        IF LowLevelTimer.Q THEN
            Alarm.2 := 1; // Авария низкого уровня
            AutoMode := FALSE;
        END_IF;
    ELSE
        LowLevelTimer(IN := FALSE);
        Alarm.2 := 0;
    END_IF;

    // Аварийный останов
    IF EmergencyStop THEN
        AutoMode := FALSE;
    END_IF;
END_METHOD

METHOD UPDATE_BALANCE_CONTROL
// Алгоритм балансировки
BEGIN
    // Расчет разницы уровней
    LevelDifference := FilteredLevel1 - FilteredLevel2;

    // Определение состояния баланса
    Balanced := ABS(LevelDifference) <= DeadBand;

    IF AutoMode AND NOT Balanced THEN
        // Режим автоматической балансировки
```

```
    BalancePID.Setpoint := BalanceSetpoint;
    BalancePID.ProcessValue := FilteredLevel1;
    BalancePID();

    // Ограничение скорости изменения
    FlowLimiter.IN := BalancePID.Output;
    FlowLimiter();
    SafeValveCmd := FlowLimiter.OUT;

    // Определение направления
    IF SafeValveCmd > 0 THEN
        ValveDirection := 1; // 1→2
        LastDirection := 1;
    ELSIF SafeValveCmd < 0 THEN
        ValveDirection := -1; // 2→1
        LastDirection := -1;
    ELSE
        ValveDirection := 0;
    END_IF;

    // Управление клапаном и насосом
    ValveOpen := ABS(SafeValveCmd);
    PumpSpeed := LIMIT(ABS(SafeValveCmd)*0.8, 0.0, 80.0); // Насос на 80% от клапана
ELSE
    // Ручной режим или баланс достигнут
    BalancePID.Reset := TRUE;

    IF ManualMode THEN
        // Ручное управление
        SafeValveCmd := ManualValveCmd;
        ValveOpen := ABS(SafeValveCmd);
```

```
            PumpSpeed := ABS(SafeValveCmd)*0.8;

            IF SafeValveCmd > 0 THEN
                ValveDirection := 1;
            ELSIF SafeValveCmd < 0 THEN
                ValveDirection := -1;
            ELSE
                ValveDirection := 0;
            END_IF;
        ELSE
            // Автоматический режим, баланс достигнут
            ValveOpen := 0.0;
            PumpSpeed := 0.0;
            ValveDirection := 0;
        END_IF;
    END_IF;
END_METHOD

// Основной цикл управления
BEGIN
    IF NOT Enable THEN
        INITIALIZE();
        Ready := FALSE;
    ELSE
        // Фильтрация уровней
        LevelFilter1.IN := Level1;
        LevelFilter1();
        FilteredLevel1 := LevelFilter1.OUT;

        LevelFilter2.IN := Level2;
        LevelFilter2();
```

```
        FilteredLevel2 := LevelFilter2.OUT;

        UPDATE_PROTECTION();
        UPDATE_BALANCE_CONTROL();
    END_IF;
END_FUNCTION_BLOCK
```

Дополнительные необходимые блоки

1. ПИД-регулятор (аналогичный предыдущим примерам)

2. Фильтр низких частот (аналогичный предыдущим примерам)

3. Ограничитель скорости изменения (аналогичный предыдущим примерам)

Пример использования

```
PROGRAM MAIN
VAR
    TankSystem: TANK_BALANCER;
    LevelTank1: REAL;
    LevelTank2: REAL;
    LevelsValid: BOOL;
    SystemEnable: BOOL;
END_VAR

// Инициализация системы
TankSystem(
    BalanceSetpoint := 50.0,  // Балансировка на 50%
    MaxFlowRate := 5.0,       // Макс скорость перекачки 5%/сек
    DeadBand := 1.5,          // Зона нечувствительности 1.5%
    Kp := 1.5,                // Коэффициент усиления
    MaxLevel := 90.0,         // Максимальный уровень
    MinLevel := 10.0          // Минимальный уровень
);
```

```
// Основной цикл управления
TankSystem(
    Enable := SystemEnable,
    Level1 := LevelTank1,
    Level2 := LevelTank2,
    LevelsValid := LevelsValid
);

// Внешняя логика управления
IF StartBalancing THEN
    TankSystem.AutoMode := TRUE;
END_IF;

IF EmergencySituation THEN
    TankSystem.EmergencyStop := TRUE;
END_IF;
```

Ключевые особенности системы

1. **Интеллектуальная балансировка**:
    - П-регулятор для плавного управления;
    - Зона нечувствительности для предотвращения колебаний;
    - Ограничение скорости перекачки.
2. **Комплексная защита**:
    - Контроль достоверности сигналов;
    - Защита от переполнения и опустошения;
    - Аварийный останов.
3. **Гибкое управление**:
    - Автоматический и ручной режимы;
    - Настройка параметров балансировки;
    - Возможность внешнего управления.

4. **Диагностика**:
    - Мониторинг разницы уровней;
    - Индикация состояния баланса;
    - Коды аварий.

Рекомендации по настройке

1. **Параметры балансировки**:
    - Начните с Kp = 1-3% на % разницы уровней;
    - DeadBand устанавливайте 1-3% от шкалы;
    - MaxFlowRate выбирайте исходя из производительности насоса.
2. **Защитные уставки**:
    - MaxLevel на 5-10% ниже физического максимума;
    - MinLevel на 5-10% выше физического минимума.
3. **Фильтрация сигналов**:
    - Для больших резервуаров используйте FC = 0,02-0,1 Гц;
    - Для малых - FC = 0,1-0,5 Гц.
4. **Интеграция**:
    - Согласуйте скорость насоса и клапана
    - Добавьте задержки при изменении направления

Данная реализация обеспечивает надежную балансировку уровней в двух резервуарах с защитой от аварийных ситуаций и гибкими возможностями настройки.

Задача 19. Генератор сигналов (синус, меандр, пила) на языке ST

Полная реализация функционального блока

```
FUNCTION_BLOCK SIGNAL_GENERATOR
VAR_INPUT
  // Управляющие параметры
  Enable: BOOL := TRUE;          // Активация генератора
  WaveType: INT := 0;            // Тип сигнала (0-синус, 1-меандр, 2-пила)
  Frequency: REAL := 1.0;        // Частота сигнала [Гц]
  Amplitude: REAL := 1.0;        // Амплитуда сигнала
  Offset: REAL := 0.0;           // Постоянное смещение
```

```
    Phase: REAL := 0.0;             // Начальная фаза [град]
    Reset: BOOL := FALSE;           // Сброс генератора

    // Параметры меандра
    DutyCycle: REAL := 50.0;        // Коэффициент заполнения [%]

    // Параметры дискретизации
    Ts: REAL := 0.01;               // Время дискретизации [сек]
END_VAR

VAR_OUTPUT
    Output: REAL := 0.0;            // Выходной сигнал
    PhaseOut: REAL := 0.0;          // Текущая фаза [град]
END_VAR

VAR
    // Внутренние переменные
    TimeCounter: REAL := 0.0;       // Счетчик времени
    RadPhase: REAL := 0.0;          // Фаза в радианах
    LastOutput: REAL := 0.0;        // Предыдущее значение
    Initialized: BOOL := FALSE;     // Флаг инициализации
END_VAR

METHOD CALCULATE_PHASE
// Расчет текущей фазы
BEGIN
    TimeCounter := TimeCounter + Ts;

    // Нормализация фазы (0-360 градусов)
    PhaseOut := (Phase + TimeCounter * Frequency * 360.0) MOD 360.0;
    RadPhase := PhaseOut * 3.1415926535 / 180.0;
```

```
END_METHOD

METHOD GENERATE_SINE: REAL
// Генерация синусоидального сигнала
BEGIN
   RETURN Amplitude * SIN(RadPhase) + Offset;
END_METHOD

METHOD GENERATE_SQUARE: REAL
// Генерация прямоугольного сигнала (меандра)
VAR
   normPhase: REAL;
BEGIN
   normPhase := PhaseOut MOD 360.0;

   IF normPhase < (360.0 * DutyCycle / 100.0) THEN
      RETURN Amplitude + Offset;
   ELSE
      RETURN -Amplitude + Offset;
   END_IF;
END_METHOD

METHOD GENERATE_SAWTOOTH: REAL
// Генерация пилообразного сигнала
VAR
   normPhase: REAL;
BEGIN
   normPhase := PhaseOut MOD 360.0;
   RETURN ((normPhase / 360.0) * 2.0 * Amplitude - Amplitude) + Offset;
END_METHOD
```

```
METHOD UPDATE_OUTPUT
// Обновление выходного сигнала
BEGIN
    CASE WaveType OF
        0: // Синусоида
            Output := GENERATE_SINE();

        1: // Меандр
            Output := GENERATE_SQUARE();

        2: // Пила
            Output := GENERATE_SAWTOOTH();

        ELSE
            Output := 0.0;
    END_CASE;
END_METHOD

BEGIN
    IF Reset OR NOT Initialized THEN
        TimeCounter := 0.0;
        PhaseOut := Phase;
        Initialized := TRUE;
    END_IF;

    IF Enable THEN
        CALCULATE_PHASE();
        UPDATE_OUTPUT();
    ELSE
        Output := 0.0;
        PhaseOut := 0.0;
```

```
    END_IF;
END_FUNCTION_BLOCK
```

Примеры использования

1. Генерация тестовых сигналов

```
PROGRAM MAIN
VAR
    TestGenerator: SIGNAL_GENERATOR;
    SineWave: REAL;
    SquareWave: REAL;
    Sawtooth: REAL;
END_VAR

// Генератор синусоиды 1 Гц
TestGenerator(
    WaveType := 0,       // Синус
    Frequency := 1.0,    // 1 Гц
    Amplitude := 5.0,     // ±5 В
    Offset := 2.5,       // Смещение 2.5 В
    Ts := 0.01           // Период 10 мс
);
TestGenerator();
SineWave := TestGenerator.Output;

// Генератор меандра 2 Гц
TestGenerator(
    WaveType := 1,       // Меандр
    Frequency := 2.0,    // 2 Гц
    DutyCycle := 30.0,    // 30% заполнения
    Amplitude := 10.0,    // ±10 В
```

```
    Ts := 0.01
);
TestGenerator();
SquareWave := TestGenerator.Output;

// Генератор пилы 0.5 Гц
TestGenerator(
    WaveType := 2,       // Пила
    Frequency := 0.5,    // 0.5 Гц
    Amplitude := 3.0,     // ±3 В
    Ts := 0.01
);
TestGenerator();
Sawtooth := TestGenerator.Output;
```

2. Использование в системе управления

```
FUNCTION_BLOCK TEST_CONTROLLER
VAR_INPUT
    EnableTest: BOOL;
    TestFreq: REAL := 0.1;
END_VAR

VAR
    TestSignal: SIGNAL_GENERATOR;
    ProcessInput: REAL;
END_VAR

TestSignal(
    Enable := EnableTest,
    WaveType := 0,       // Синусоида
    Frequency := TestFreq,
```

```
    Amplitude := 20.0,    // ±20 единиц
    Offset := 50.0,      // Центр в 50
    Ts := 0.1
);
TestSignal();

// Подача тестового сигнала на процесс
IF EnableTest THEN
    ProcessInput := TestSignal.Output;
ELSE
    ProcessInput := 50.0; // Номинальное значение
END_IF;
```

Дополнительные модификации

1. Генератор с плавным переключением форм сигнала

```
FUNCTION_BLOCK ADVANCED_SIGNAL_GEN
EXTENDS SIGNAL_GENERATOR
VAR_INPUT
    TransitionTime: REAL := 1.0; // Время перехода между формами [сек]
END_VAR

VAR
    TransitionCounter: REAL := 0.0;
    OldWaveType: INT := 0;
    OldOutput: REAL := 0.0;
END_VAR

METHOD SMOOTH_TRANSITION: REAL
VAR
    blendFactor: REAL;
BEGIN
```

```
    IF WaveType <> OldWaveType THEN
        TransitionCounter := 0.0;
        OldWaveType := WaveType;
        OldOutput := LastOutput;
    END_IF;

    IF TransitionCounter < TransitionTime THEN
        TransitionCounter := TransitionCounter + Ts;
        blendFactor := TransitionCounter / TransitionTime;

        CASE OldWaveType OF
            0: OldOutput := GENERATE_SINE();
            1: OldOutput := GENERATE_SQUARE();
            2: OldOutput := GENERATE_SAWTOOTH();
        END_CASE;

        CASE WaveType OF
            0: RETURN OldOutput*(1-blendFactor) + GENERATE_SINE()*blendFactor;
            1: RETURN OldOutput*(1-blendFactor) + GENERATE_SQUARE()*blendFactor;
            2: RETURN OldOutput*(1-blendFactor) + GENERATE_SAWTOOTH()*blendFactor;
        END_CASE;
    ELSE
        RETURN Output;
    END_IF;
END_METHOD

BEGIN
    IF Enable THEN
        CALCULATE_PHASE();
        UPDATE_OUTPUT();
        Output := SMOOTH_TRANSITION();
```

```
        LastOutput := Output;
    ELSE
        Output := 0.0;
        PhaseOut := 0.0;
    END_IF;
END_FUNCTION_BLOCK
```

2. Генератор с дополнительными формами сигналов

```
FUNCTION_BLOCK EXTENDED_SIGNAL_GEN
EXTENDS SIGNAL_GENERATOR
VAR_INPUT
    WaveType: INT := 0; // 0-синус, 1-меандр, 2-пила, 3-треугольник, 4-шум
    NoiseLevel: REAL := 0.1; // Уровень шума для режима 4
END_VAR

METHOD GENERATE_TRIANGLE: REAL
// Генерация треугольного сигнала
VAR
    normPhase: REAL;
BEGIN
    normPhase := PhaseOut MOD 360.0;

    IF normPhase < 180.0 THEN
        RETURN ((normPhase / 180.0) * 2.0 * Amplitude - Amplitude) + Offset;
    ELSE
        RETURN ((360.0 - normPhase) / 180.0 * 2.0 * Amplitude - Amplitude) + Offset;
    END_IF;
END_METHOD

METHOD GENERATE_NOISE: REAL
// Генерация сигнала с шумом
```

```
BEGIN
   RETURN (RAND() * 2.0 - 1.0) * NoiseLevel * Amplitude + Offset;
END_METHOD

METHOD UPDATE_OUTPUT
// Обновленный метод с дополнительными формами
BEGIN
   CASE WaveType OF
      0: Output := GENERATE_SINE();
      1: Output := GENERATE_SQUARE();
      2: Output := GENERATE_SAWTOOTH();
      3: Output := GENERATE_TRIANGLE();
      4: Output := GENERATE_NOISE();
      ELSE Output := 0.0;
   END_CASE;
END_METHOD
END_FUNCTION_BLOCK
```

Ключевые особенности реализации

1. **Гибкость генерации**:
    - Поддержка основных форм сигналов;
    - Настройка амплитуды, смещения и частоты;
    - Контроль фазы сигнала.
2. **Точность работы**:
    - Учет времени дискретизации Ts;
    - Корректная обработка фазы;
    - Плавное изменение параметров.
3. **Дополнительные функции**:
    - Коэффициент заполнения для меандра;
    - Сброс генератора;
    - Выход текущей фазы.

4. **Оптимизация**:
    - Минимальные вычислительные затраты;
    - Отсутствие накопления ошибки;
    - Стабильность при длительной работе.

Рекомендации по применению

1. **Выбор параметров**:
    - Для тестирования систем выбирайте частоту 0.1-10 Гц;
    - Амплитуду устанавливайте в 50-80% от рабочего диапазона;
    - Время дискретизации должно быть в 10-100 раз меньше периода сигнала.
2. **Использование в тестовых системах**:
    - Проверка АЧХ и ФЧХ систем;
    - Тестирование регуляторов;
    - Калибровка измерительных каналов.
3. **Интеграция**:
    - Для плавного старта устанавливайте начальную фазу;
    - Используйте сброс при изменении критических параметров;
    - Комбинируйте с другими блоками обработки сигналов.
4. **Отладка**:
    - Контролируйте выходную фазу для синхронизации;
    - Проверяйте крайние значения параметров;
    - Визуализируйте сигналы в системах SCADA.

Данная реализация генератора сигналов предоставляет удобный инструмент для тестирования и настройки систем автоматизации с широкими возможностями конфигурации.

Задача 20. Разработать приложение на языке ST, реализующее адаптивный ПИД-регулятор с автоматической подстройкой параметров

Полная реализация на языке ST

```
FUNCTION_BLOCK ADAPTIVE_PID
VAR_INPUT
    // Основные входы
    Setpoint: REAL;              // Уставка
    ProcessValue: REAL;          // Текущее значение процесса
    Enable: BOOL := TRUE;        // Активация регулятора
    Reset: BOOL := FALSE;        // Сброс регулятора

    // Базовые параметры ПИД
    BaseKp: REAL := 1.0;         // Базовый пропорциональный коэффициент
    BaseTi: REAL := 10.0;        // Базовое интегральное время (сек)
    BaseTd: REAL := 1.0;         // Базовое дифференциальное время (сек)

    // Параметры адаптации
    AdaptInterval: TIME := T#30s;   // Интервал адаптации
    MinKp: REAL := 0.1;          // Минимальное значение Kp
    MaxKp: REAL := 10.0;         // Максимальное значение Kp
    AdaptSensitivity: REAL := 0.2;  // Чувствительность адаптации (0..1)

    // Ограничения
    MaxOutput: REAL := 100.0;    // Максимальный выход
    MinOutput: REAL := 0.0;      // Минимальный выход
    OutputRateLimit: REAL := 5.0;   // Ограничение скорости изменения выхода
    Ts: REAL := 0.1;             // Время дискретизации [сек]
END_VAR

VAR_OUTPUT
    Output: REAL;                // Выходной сигнал
```

```
    Status: INT := 0;              // Статус работы (0-инициализация, 1-работа, 2-адаптация)
    CurrentKp: REAL := 1.0;        // Текущий Kp
    CurrentTi: REAL := 10.0;       // Текущее Ti
    CurrentTd: REAL := 1.0;        // Текущее Td
    Performance: REAL := 0.0;      // Показатель качества регулирования
END_VAR

VAR
    // Основной ПИД-регулятор
    PID: PID_CONTROLLER;

    // Переменные адаптации
    AdaptTimer: TON;
    LastError: REAL := 0.0;
    ErrorIntegral: REAL := 0.0;
    LastAdaptTime: TIME;

    // Анализ процесса
    OscillationCounter: INT := 0;
    LastCrossingTime: TIME;
    ProcessGain: REAL := 1.0;

    // Системные переменные
    FirstPass: BOOL := TRUE;
    InternalReset: BOOL := FALSE;
END_VAR

METHOD INITIALIZE
// Инициализация регулятора
BEGIN
    PID(
```

```
        Kp := BaseKp,
        Ti := BaseTi,
        Td := BaseTd,
        Ts := Ts,
        MaxOutput := MaxOutput,
        MinOutput := MinOutput,
        MaxOutputStep := OutputRateLimit,
        Reset := TRUE
    );

    CurrentKp := BaseKp;
    CurrentTi := BaseTi;
    CurrentTd := BaseTd;

    AdaptTimer(IN := FALSE);
    ErrorIntegral := 0.0;
    OscillationCounter := 0;
    FirstPass := FALSE;
    Status := 1;
END_METHOD

METHOD CALCULATE_PERFORMANCE
// Расчет показателя качества регулирования
VAR_INPUT
    Error: REAL;
END_VAR
BEGIN
    // Интеграл абсолютной ошибки (IAE)
    ErrorIntegral := ErrorIntegral + ABS(Error) * Ts;

    // Детектирование колебаний
```

```
    IF (LastError * Error < 0) AND (T#1s <= (CURRENT_TIME - LastCrossingTime)) THEN
        OscillationCounter := OscillationCounter + 1;
        LastCrossingTime := CURRENT_TIME;
    END_IF;

    LastError := Error;

    // Комплексный показатель качества
    Performance := ErrorIntegral / (1.0 + OscillationCounter);
END_METHOD

METHOD ADAPT_PARAMETERS
// Адаптация параметров ПИД
VAR
    Error: REAL;
    DeltaKp: REAL;
BEGIN
    Error := Setpoint - ProcessValue;

    // Правило адаптации (упрощенный градиентный спуск)
    DeltaKp := AdaptSensitivity * Error * (ProcessValue / Setpoint);

    // Корректировка Kp
    CurrentKp := LIMIT(BaseKp * (1.0 + DeltaKp), MinKp, MaxKp);

    // Корректировка Ti и Td (сохраняя соотношения)
    CurrentTi := BaseTi * (BaseKp / CurrentKp);
    CurrentTd := BaseTd * (CurrentKp / BaseKp);

    // Применение новых параметров
    PID.Kp := CurrentKp;
```

```
    PID.Ti := CurrentTi;
    PID.Td := CurrentTd;

    Status := 2; // Режим адаптации
END_METHOD

// Основной цикл управления
BEGIN
    IF Reset OR FirstPass THEN
        INITIALIZE();
        InternalReset := FALSE;
    END_IF;

    IF Enable THEN
        // Нормальная работа ПИД
        PID.Setpoint := Setpoint;
        PID.ProcessValue := ProcessValue;
        PID();
        Output := PID.Output;

        // Мониторинг качества регулирования
        CALCULATE_PERFORMANCE(Setpoint - ProcessValue);

        // Периодическая адаптация
        AdaptTimer(
            IN := TRUE,
            PT := AdaptInterval
        );

        IF AdaptTimer.Q THEN
            ADAPT_PARAMETERS();
```

```
            AdaptTimer(IN := FALSE);
        ELSE
            Status := 1; // Нормальный режим
        END_IF;
    ELSE
        Output := 0.0;
        Status := 0;
    END_IF;
END_FUNCTION_BLOCK
```

Дополнительные необходимые блоки

1. Базовый ПИД-регулятор (используется внутри адаптивного)

```
FUNCTION_BLOCK PID_CONTROLLER
VAR_INPUT
    Setpoint: REAL;
    ProcessValue: REAL;
    Kp: REAL := 1.0;
    Ti: REAL := 0.0;
    Td: REAL := 0.0;
    Ts: REAL := 0.1;
    MaxOutput: REAL := 100.0;
    MinOutput: REAL := 0.0;
    MaxOutputStep: REAL := 1.0;
    Reset: BOOL := FALSE;
END_VAR

VAR_OUTPUT
    Output: REAL;
    Error: REAL;
    P_Term: REAL;
    I_Term: REAL;
```

```
    D_Term: REAL;
END_VAR

VAR
    Integral: REAL := 0.0;
    PrevError: REAL := 0.0;
    PrevProcessValue: REAL := 0.0;
    PrevOutput: REAL := 0.0;
END_VAR

BEGIN
    Error := Setpoint - ProcessValue;

    // Пропорциональная составляющая
    P_Term := Kp * Error;

    // Интегральная составляющая
    IF Ti > 0 THEN
        Integral := Integral + Kp * Error * Ts / Ti;
        // Антивиндап
        IF Integral > (MaxOutput - P_Term) THEN
            Integral := MaxOutput - P_Term;
        ELSIF Integral < (MinOutput - P_Term) THEN
            Integral := MinOutput - P_Term;
        END_IF;
    ELSE
        Integral := 0.0;
    END_IF;
    I_Term := Integral;

    // Дифференциальная составляющая (по измерению)
```

```
IF Td > 0 THEN
    D_Term := -Kp * Td * (ProcessValue - PrevProcessValue) / Ts;
ELSE
    D_Term := 0.0;
END_IF;

// Суммирование составляющих
Output := P_Term + I_Term + D_Term;

// Ограничение скорости изменения
IF (Output - PrevOutput) > MaxOutputStep THEN
    Output := PrevOutput + MaxOutputStep;
ELSIF (Output - PrevOutput) < -MaxOutputStep THEN
    Output := PrevOutput - MaxOutputStep;
END_IF;

// Общие ограничения
IF Output > MaxOutput THEN
    Output := MaxOutput;
ELSIF Output < MinOutput THEN
    Output := MinOutput;
END_IF;

// Сохранение состояний
PrevError := Error;
PrevProcessValue := ProcessValue;
PrevOutput := Output;

IF Reset THEN
    Integral := 0.0;
    PrevError := 0.0;
```

```
        PrevProcessValue := ProcessValue;
        Output := 0.0;
    END_IF;
END_FUNCTION_BLOCK
```

Пример использования

```
PROGRAM MAIN
VAR
    TemperatureControl: ADAPTIVE_PID;
    SetpointTemp: REAL := 50.0;
    ActualTemp: REAL;
    HeaterOutput: REAL;
END_VAR

// Инициализация адаптивного ПИД
TemperatureControl(
    BaseKp := 2.5,
    BaseTi := 120.0,
    BaseTd := 30.0,
    AdaptInterval := T#5m,    // Адаптация каждые 5 минут
    MaxOutput := 100.0,
    Ts := 0.5
);

// Основной цикл управления
TemperatureControl(
    Setpoint := SetpointTemp,
    ProcessValue := ActualTemp,
    Enable := TRUE
);
```

HeaterOutput := TemperatureControl.Output;

Ключевые особенности реализации

1. **Механизм адаптации**:
    - Градиентный метод подстройки коэффициентов;
    - Учет интеграла ошибки и колебаний;
    - Периодическая коррекция параметров.
2. **Защитные функции**:
    - Ограничения на диапазон параметров;
    - Контроль скорости изменения выхода;
    - Защита от перерегулирования.
3. **Диагностика**:
    - Выход текущих параметров;
    - Показатель качества регулирования;
    - Статус работы регулятора.
4. **Гибкость**:
    - Возможность задания базовых параметров;
    - Настройка чувствительности адаптации;
    - Регулировка интервала адаптации.

Алгоритм работы адаптации

1. **Оценка качества регулирования**:
    - Расчет интеграла абсолютной ошибки (IAE);
    - Подсчет количества перерегулирований;
    - Комплексный показатель Performance = IAE / (1 + Oscillations).
2. **Коррекция параметров**:
    - Kp изменяется пропорционально ошибке;
    - Ti и Td корректируются для сохранения соотношений;
    - Плавное изменение параметров без скачков.
3. **Периодичность адаптации**:
    - Настройка интервала AdaptInterval;

- Защита от слишком частой адаптации;
- Возможность ручного сброса.

Рекомендации по настройке

1. **Начальные параметры**:
   - Установите BaseKp, BaseTi, BaseTd как для обычного ПИД;
   - Начните с AdaptInterval = 5-10 периодов установления.
2. **Ограничения**:
   - MinKp = 0.1 * BaseKp;
   - MaxKp = 10 * BaseKp;
   - OutputRateLimit = 1-5% от диапазона в секунду.
3. **Адаптация**:
   - AdaptSensitivity = 0.1-0.3 для плавной адаптации;
   - Увеличивайте для быстрых процессов;
   - Уменьшайте для инерционных систем.
4. **Мониторинг**:
   - Контролируйте показатель Performance;
   - Анализируйте изменения CurrentKp, CurrentTi, CurrentTd;
   - Визуализируйте процесс регулирования.

Данная реализация адаптивного ПИД-регулятора обеспечивает автоматическую подстройку под изменяющиеся параметры процесса, сохраняя устойчивость регулирования в различных рабочих условиях.

Задача 21. Разработать приложение на языке ST, реализующее генератор управляющего воздействия без объекта управления

Полная реализация функционального блока

```
FUNCTION_BLOCK PID_SIGNAL_GENERATOR
VAR_INPUT
    // Управляющие параметры
    Enable: BOOL := TRUE;          // Активация генератора
    Mode: INT := 0;                // Режим работы (0-пошаговый, 1-синусоидальный, 2-пилообразный)
    Reset: BOOL := FALSE;          // Сброс генератора

    // Параметры ПИД-регулятора
    Kp: REAL := 1.0;               // Пропорциональный коэффициент
    Ti: REAL := 0.0;               // Интегральное время (0 - отключить)
    Td: REAL := 0.0;               // Дифференциальное время (0 - отключить)

    // Параметры тестового сигнала
    Setpoint: REAL := 50.0;        // Уставка
    Amplitude: REAL := 10.0;       // Амплитуда возмущений
    Frequency: REAL := 0.1;        // Частота тестового сигнала [Гц]
    StepTime: TIME := T#10s;       // Время шага для пошагового режима

    // Ограничения
    MaxOutput: REAL := 100.0;      // Максимальный выход
    MinOutput: REAL := 0.0;        // Минимальный выход
    Ts: REAL := 0.1;               // Время дискретизации [сек]
END_VAR

VAR_OUTPUT
    Output: REAL := 0.0;           // Выходное управляющее воздействие
    SimulatedPV: REAL := 0.0;      // Имитация Process Value
```

```
    Error: REAL := 0.0;             // Текущая ошибка
    P_Term: REAL := 0.0;            // Пропорциональная составляющая
    I_Term: REAL := 0.0;            // Интегральная составляющая
    D_Term: REAL := 0.0;            // Дифференциальная составляющая
END_VAR

VAR
    // Внутренний ПИД-регулятор
    PID: PID_CONTROLLER;

    // Генерация тестового сигнала
    TimeCounter: REAL := 0.0;
    LastStepTime: TIME;
    StepPhase: INT := 0;
    TestSignal: REAL := 0.0;

    // Системные переменные
    FirstPass: BOOL := TRUE;
END_VAR

METHOD GENERATE_TEST_SIGNAL: REAL
// Генерация тестового сигнала для Process Value
BEGIN
    TimeCounter := TimeCounter + Ts;

    CASE Mode OF
        0: // Пошаговый режим
            IF (CURRENT_TIME - LastStepTime) >= StepTime THEN
                StepPhase := (StepPhase + 1) MOD 4;
                LastStepTime := CURRENT_TIME;
            END_IF;
```

```
            CASE StepPhase OF
                0: RETURN Setpoint;
                1: RETURN Setpoint + Amplitude;
                2: RETURN Setpoint;
                3: RETURN Setpoint - Amplitude;
            END_CASE;

        1: // Синусоидальный режим
            RETURN Setpoint + Amplitude * SIN(2 * 3.14159 * Frequency * TimeCounter);

        2: // Пилообразный режим
            RETURN Setpoint + Amplitude * (2 * FRAC(Frequency * TimeCounter) - 1);

        ELSE
            RETURN Setpoint;
    END_CASE;
END_METHOD

METHOD UPDATE_PID
// Обновление ПИД-регулятора
BEGIN
    // Генерация тестового сигнала
    SimulatedPV := GENERATE_TEST_SIGNAL();

    // Расчет ПИД
    PID(
        Setpoint := Setpoint,
        ProcessValue := SimulatedPV,
        Kp := Kp,
        Ti := Ti,
```

```
        Td := Td,
        Ts := Ts,
        MaxOutput := MaxOutput,
        MinOutput := MinOutput
    );

    // Получение результатов
    Output := PID.Output;
    Error := PID.Error;
    P_Term := PID.P_Term;
    I_Term := PID.I_Term;
    D_Term := PID.D_Term;
END_METHOD

// Основной цикл управления
BEGIN
    IF Reset OR FirstPass THEN
        // Инициализация
        PID.Reset := TRUE;
        TimeCounter := 0.0;
        LastStepTime := CURRENT_TIME;
        StepPhase := 0;
        FirstPass := FALSE;
    END_IF;

    IF Enable THEN
        UPDATE_PID();
    ELSE
        Output := 0.0;
        SimulatedPV := Setpoint;
        Error := 0.0;
```

```
        P_Term := 0.0;
        I_Term := 0.0;
        D_Term := 0.0;
    END_IF;
END_FUNCTION_BLOCK
```

Пример использования

```
PROGRAM MAIN
VAR
    PID_Generator: PID_SIGNAL_GENERATOR;
    ControlSignal: REAL;
    SimulatedProcess: REAL;
    TestMode: INT := 1;
END_VAR

// Инициализация генератора
PID_Generator(
    Mode := TestMode,        // Синусоидальный режим
    Setpoint := 50.0,        // Уставка 50%
    Amplitude := 15.0,       // Амплитуда возмущений ±15%
    Frequency := 0.05,       // Частота 0.05 Гц (период 20 сек)
    Kp := 2.0,               // Пропорциональный коэффициент
    Ti := 60.0,              // Интегральное время 60 сек
    Td := 5.0,               // Дифференциальное время 5 сек
    Ts := 0.2                // Период обновления 200 мс
);

// Основной цикл
PID_Generator();
ControlSignal := PID_Generator.Output;
SimulatedProcess := PID_Generator.SimulatedPV;
```

```
// Переключение режимов по условию
IF ChangeToStepMode THEN
   PID_Generator.Mode := 0; // Пошаговый режим
   PID_Generator.StepTime := T#15s;
END_IF;
```

Дополнительные модификации

1. Расширенная версия с дополнительными режимами

```
FUNCTION_BLOCK ADVANCED_PID_GENERATOR
EXTENDS PID_SIGNAL_GENERATOR
VAR_INPUT
   Mode: INT := 0; // 0-шаг, 1-синус, 2-пила, 3-шум, 4-прямоугольный
   NoiseLevel: REAL := 0.1; // Уровень шума для режима 3
   DutyCycle: REAL := 50.0; // Коэффициент заполнения для режима 4
END_VAR

METHOD GENERATE_TEST_SIGNAL: REAL
// Расширенная генерация тестового сигнала
VAR
   phase: REAL;
BEGIN
   TimeCounter := TimeCounter + Ts;
   phase := TimeCounter * Frequency;

   CASE Mode OF
      0: // Пошаговый режим (как в базовой версии)
         RETURN Inherited();

      1: // Синусоидальный режим
         RETURN Setpoint + Amplitude * SIN(2 * 3.14159 * phase);
```

```
        2: // Пилообразный режим
           RETURN Setpoint + Amplitude * (2 * FRAC(phase) - 1);

        3: // Режим с шумом
           RETURN Setpoint + Amplitude * (RAND() - 0.5) * 2.0 * NoiseLevel;

        4: // Прямоугольный сигнал
           IF FRAC(phase) < (DutyCycle / 100.0) THEN
              RETURN Setpoint + Amplitude;
           ELSE
              RETURN Setpoint - Amplitude;
           END_IF;

        ELSE
           RETURN Setpoint;
     END_CASE;
END_METHOD
END_FUNCTION_BLOCK
```

Ключевые особенности реализации

1. **Режимы работы**:
    - Пошаговый тест (ступенчатое изменение);
    - Синусоидальное возмущение;
    - Пилообразное возмущение;
    - Случайный шум (в расширенной версии);
    - Прямоугольный сигнал (в расширенной версии).
2. **Полный ПИД-контроллер**:
    - Все три составляющие (P, I, D);
    - Настройка параметров в реальном времени;
    - Ограничение выходного сигнала.

3. **Диагностика**:
    - Выход всех составляющих ПИД;
    - Текущая ошибка регулирования;
    - Имитация Process Value.
4. **Гибкость**:
    - Настройка амплитуды и частоты тестового сигнала;
    - Возможность расширения дополнительными режимами;
    - Простота интеграции в тестовые стенды.

Применение в тестировании систем

1. **Тестирование алгоритмов управления**:

```
// В системе-имитаторе
TestPID(
  Setpoint := PID_Generator.Output,
  ActualValue := PID_Generator.SimulatedPV
);
```

2. **Обучение операторов**:
    - Демонстрация работы ПИД-регулятора;
    - Показ влияния параметров Kp, Ti, Td;
    - Имитация различных рабочих ситуаций.
3. **Калибровка систем**:
    - Проверка реакции на различные возмущения;
    - Оптимизация параметров регуляторов;
    - Тестирование алгоритмов защиты.
4. **Разработка SCADA-интерфейсов**:
    - Создание тестовых сигналов для визуализации;
    - Проверка графиков трендов;
    - Тестирование систем сигнализации.

Данная реализация генератора управляющего воздействия позволяет полноценно тестировать и настраивать системы управления без необходимости подключения реального технологического оборудования.

Задача 22. Фильтр скользящего среднего на языке ST

Полная реализация функционального блока

```
FUNCTION_BLOCK MOVING_AVERAGE
VAR_INPUT
   IN: REAL;                  // Входной сигнал
   ENABLE: BOOL := TRUE;          // Активация фильтра
   WINDOW_SIZE: UINT := 10;       // Размер окна усреднения (1-100)
   RESET: BOOL := FALSE;          // Сброс фильтра
END_VAR

VAR_OUTPUT
   OUT: REAL := 0.0;              // Отфильтрованный сигнал
   BUFFER_FULL: BOOL := FALSE;    // Флаг заполненности буфера
END_VAR

VAR
   BUFFER: ARRAY[0..99] OF REAL;  // Циклический буфер
   SUM: REAL := 0.0;              // Текущая сумма
   INDEX: UINT := 0;              // Текущий индекс
   COUNT: UINT := 0;              // Текущее количество элементов
   INITIALIZED: BOOL := FALSE;    // Флаг инициализации
END_VAR

METHOD INIT_BUFFER
// Инициализация буфера
BEGIN
   SUM := 0.0;
   INDEX := 0;
   COUNT := 0;
```

```
    // Заполнение буфера текущим значением
    FOR i := 0 TO WINDOW_SIZE-1 DO
        BUFFER[i] := IN;
        SUM := SUM + IN;
    END_FOR;

    OUT := IN;
    BUFFER_FULL := FALSE;
    INITIALIZED := TRUE;
END_METHOD

METHOD UPDATE_FILTER
// Обновление фильтра
BEGIN
    // Если размер окна изменился
    IF COUNT > WINDOW_SIZE THEN
        INIT_BUFFER();
        RETURN;
    END_IF;

    // Вычитаем самое старое значение
    IF COUNT >= WINDOW_SIZE THEN
        SUM := SUM - BUFFER[INDEX];
    END_IF;

    // Добавляем новое значение
    BUFFER[INDEX] := IN;
    SUM := SUM + IN;

    // Обновляем индекс
    INDEX := (INDEX + 1) MOD WINDOW_SIZE;
```

```
    // Увеличиваем счетчик до заполнения буфера
    IF COUNT < WINDOW_SIZE THEN
        COUNT := COUNT + 1;
    END_IF;

    // Расчет среднего значения
    IF COUNT > 0 THEN
        OUT := SUM / COUNT;
    ELSE
        OUT := 0.0;
    END_IF;

    // Установка флага заполненности
    BUFFER_FULL := (COUNT >= WINDOW_SIZE);
END_METHOD

// Основной код функционального блока
BEGIN
    IF RESET OR NOT INITIALIZED THEN
        INIT_BUFFER();
    END_IF;

    IF ENABLE THEN
        UPDATE_FILTER();
    ELSE
        OUT := IN; // Байпас при отключении
    END_IF;
END_FUNCTION_BLOCK
```

Пример использования

```
PROGRAM MAIN
VAR
   TempFilter: MOVING_AVERAGE;
   RawTemp: REAL;
   FilteredTemp: REAL;
END_VAR

// Инициализация фильтра с окном 20 значений
TempFilter(
   WINDOW_SIZE := 20,
   ENABLE := TRUE
);

// Основной цикл обработки
TempFilter.IN := RawTemp;
TempFilter();
FilteredTemp := TempFilter.OUT;

// Динамическое изменение размера окна
IF ChangeWindowSize THEN
   TempFilter.WINDOW_SIZE := 30;
END_IF;
```

Дополнительные модификации

1. Версия с взвешенными коэффициентами

```
FUNCTION_BLOCK WEIGHTED_MOVING_AVERAGE
EXTENDS MOVING_AVERAGE
VAR
   WEIGHTS: ARRAY[0..99] OF REAL; // Весовые коэффициенты
END_VAR
```

```
METHOD INIT_WEIGHTS
// Инициализация весов (пример: линейное уменьшение)
VAR
    i: UINT;
    total: REAL := 0.0;
END_VAR
BEGIN
    // Линейно убывающие веса (новые данные важнее)
    FOR i := 0 TO WINDOW_SIZE-1 DO
        WEIGHTS[i] := WINDOW_SIZE - i;
        total := total + WEIGHTS[i];
    END_FOR;

    // Нормализация весов
    FOR i := 0 TO WINDOW_SIZE-1 DO
        WEIGHTS[i] := WEIGHTS[i] / total;
    END_FOR;
END_METHOD

METHOD UPDATE_FILTER
// Обновление фильтра с весами
VAR
    i, pos: UINT;
    weightedSum: REAL := 0.0;
END_VAR
BEGIN
    // Стандартное обновление буфера
    SUPER.UPDATE_FILTER();

    // Расчет взвешенного среднего
    IF COUNT > 0 THEN
```

```
        weightedSum := 0.0;
        FOR i := 0 TO COUNT-1 DO
            pos := (INDEX + WINDOW_SIZE - i - 1) MOD WINDOW_SIZE;
            weightedSum := weightedSum + BUFFER[pos] * WEIGHTS[i];
        END_FOR;
        OUT := weightedSum;
    END_IF;
END_METHOD

BEGIN
    IF RESET OR NOT INITIALIZED THEN
        INIT_WEIGHTS();
        SUPER.INIT_BUFFER();
    END_IF;

    IF ENABLE THEN
        UPDATE_FILTER();
    ELSE
        OUT := IN;
    END_IF;
END_FUNCTION_BLOCK
```

Ключевые особенности реализации

1. **Эффективный алгоритм**:
    - Циклический буфер для хранения значений;
    - Поддержка динамического изменения размера окна;
    - Минимальные вычислительные затраты.
2. **Гибкость**:
    - Работа с переменным размером окна;
    - Возможность расширения для взвешенного среднего;
    - Байпас при отключении фильтра.

3. **Диагностика**:
    - Флаг заполненности буфера;
    - Поддержка сброса состояния.
4. **Оптимизация**:
    - Одно обновление суммы при добавлении нового значения;
    - Нет необходимости пересчитывать всю сумму каждый раз.

Рекомендации по применению

1. **Выбор размера окна**:
    - Для медленных процессов: 20-100 значений;
    - Для быстрых процессов: 5-20 значений;
    - Зависит от частоты дискретизации.
2. **Инициализация**:
    - При запуске заполняйте буфер текущим значением;
    - Учитывайте время заполнения буфера.
3. **Сравнение с другими фильтрами**:
    - Проще, чем ФНЧ, но менее эффективен против высокочастотных помех;
    - Сохраняет фронты сигнала лучше чем экспоненциальное сглаживание.
4. **Реальные применения**:
    - Фильтрация показаний датчиков;
    - Сглаживание управляющих сигналов;
    - Предварительная обработка данных.

Пример для датчика температуры

```
PROGRAM TEMP_MONITOR
VAR
  TempSensors: ARRAY[1..5] OF MOVING_AVERAGE;
  RawTemps: ARRAY[1..5] OF REAL;
  FilteredTemps: ARRAY[1..5] OF REAL;
  i: INT;
END_VAR
```

```
// Инициализация фильтров для каждого датчика
FOR i := 1 TO 5 DO
   TempSensors[i](
      WINDOW_SIZE := 15,
      ENABLE := TRUE
   );
END_FOR;

// Основной цикл обработки
FOR i := 1 TO 5 DO
   TempSensors[i].IN := RawTemps[i];
   TempSensors[i]();
   FilteredTemps[i] := TempSensors[i].OUT;
END_FOR;
```

Данная реализация фильтра скользящего среднего обеспечивает простое и эффективное сглаживание сигналов с минимальными вычислительными затратами, что делает его идеальным выбором для встраиваемых систем и промышленных контроллеров.

Задача 23. Разработать приложение на языке ST, реализующее взвешенный медианный фильтр

Полная реализация функционального блока

```
FUNCTION_BLOCK WEIGHTED_MEDIAN_FILTER
VAR_INPUT
   IN: REAL;                    // Входной сигнал
   ENABLE: BOOL := TRUE;        // Активация фильтра
   WINDOW_SIZE: UINT := 5;      // Размер окна фильтрации (3-15)
   WEIGHTS: ARRAY[1..15] OF UINT := [1,2,3,2,1,0,0,0,0,0,0,0,0,0,0]; // Весовые коэффициенты
   RESET: BOOL := FALSE;        // Сброс фильтра
   Ts: REAL := 0.1;             // Время дискретизации [сек]
END_VAR
```

```
VAR_OUTPUT
   OUT: REAL := 0.0;              // Отфильтрованный сигнал
   BUFFER_FULL: BOOL := FALSE;    // Флаг заполненности буфера
END_VAR

VAR
   BUFFER: ARRAY[0..14] OF REAL;   // Циклический буфер
   SORTED: ARRAY[0..14] OF REAL;   // Буфер для сортировки
   INDEX: UINT := 0;              // Текущий индекс
   COUNT: UINT := 0;               // Текущее количество элементов
   INITIALIZED: BOOL := FALSE;    // Флаг инициализации
END_VAR

METHOD INIT_BUFFER
// Инициализация буфера
VAR
   i: UINT;
END_VAR
BEGIN
   FOR i := 0 TO WINDOW_SIZE-1 DO
      BUFFER[i] := IN;
   END_FOR;

   INDEX := 0;
   COUNT := 0;
   OUT := IN;
   BUFFER_FULL := FALSE;
   INITIALIZED := TRUE;
END_METHOD

METHOD INSERTION_SORT
```

```
// Сортировка вставками для небольшого окна
VAR_INPUT_OUTPUT
   arr: ARRAY[0..14] OF REAL;
END_VAR
VAR
   i, j: INT;
   key: REAL;
END_VAR
BEGIN
   FOR i := 1 TO WINDOW_SIZE-1 DO
      key := arr[i];
      j := i - 1;

      WHILE (j >= 0) AND (arr[j] > key) DO
         arr[j + 1] := arr[j];
         j := j - 1;
      END_WHILE;

      arr[j + 1] := key;
   END_FOR;
END_METHOD

METHOD CALCULATE_MEDIAN: REAL
// Расчет взвешенной медианы
VAR
   i, j, k: UINT;
   totalWeights: UINT := 0;
   halfWeights: UINT := 0;
   currentWeight: UINT := 0;
   effectiveWindow: UINT := 0;
BEGIN
```

```
// Копирование и сортировка данных
FOR i := 0 TO WINDOW_SIZE-1 DO
   SORTED[i] := BUFFER[(INDEX + WINDOW_SIZE - i - 1) MOD WINDOW_SIZE];
END_FOR;

INSERTION_SORT(SORTED);

// Расчет общего веса
FOR i := 1 TO WINDOW_SIZE DO
   totalWeights := totalWeights + WEIGHTS[i];
END_FOR;

halfWeights := totalWeights / 2;

// Поиск медианы с учетом весов
currentWeight := 0;
FOR i := 0 TO WINDOW_SIZE-1 DO
   effectiveWindow := i + 1;
   currentWeight := currentWeight + WEIGHTS[effectiveWindow];

   IF currentWeight > halfWeights THEN
      RETURN SORTED[i];
   ELSIF currentWeight = halfWeights THEN
      // Для четного общего веса - среднее между двумя центральными
      IF i < WINDOW_SIZE-1 THEN
         RETURN (SORTED[i] + SORTED[i+1]) / 2.0;
      ELSE
         RETURN SORTED[i];
      END_IF;
   END_IF;
END_FOR;
```

```
    RETURN 0.0; // На случай ошибки
END_METHOD

METHOD UPDATE_FILTER
// Обновление фильтра
BEGIN
    // Добавление нового значения
    BUFFER[INDEX] := IN;
    INDEX := (INDEX + 1) MOD WINDOW_SIZE;

    // Увеличение счетчика до заполнения буфера
    IF COUNT < WINDOW_SIZE THEN
        COUNT := COUNT + 1;
    END_IF;

    // Расчет медианы
    IF COUNT >= WINDOW_SIZE THEN
        OUT := CALCULATE_MEDIAN();
        BUFFER_FULL := TRUE;
    ELSE
        OUT := IN; // Пока буфер не заполнен, пропускаем сигнал
    END_IF;
END_METHOD

// Основной код функционального блока
BEGIN
    IF RESET OR NOT INITIALIZED OR (WINDOW_SIZE < 3) OR (WINDOW_SIZE > 15)
THEN
        INIT_BUFFER();
    END_IF;
```

```
    IF ENABLE THEN
        UPDATE_FILTER();
    ELSE
        OUT := IN; // Байпас при отключении
    END_IF;
END_FUNCTION_BLOCK
```

Пример использования

```
PROGRAM MAIN
VAR
    SensorFilter: WEIGHTED_MEDIAN_FILTER;
    RawValue: REAL;
    FilteredValue: REAL;
    CustomWeights: ARRAY[1..15] OF UINT := [1,1,2,3,5,0,0,0,0,0,0,0,0,0,0];
END_VAR

// Инициализация фильтра с окном 5 и пользовательскими весами
SensorFilter(
    WINDOW_SIZE := 5,
    WEIGHTS := CustomWeights, // Центральные значения имеют больший вес
    Ts := 0.1
);

// Основной цикл обработки
SensorFilter.IN := RawValue;
SensorFilter();
FilteredValue := SensorFilter.OUT;

// Динамическое изменение параметров
IF ChangeParameters THEN
    SensorFilter.WINDOW_SIZE := 7;
```

```
    SensorFilter.WEIGHTS := [1,1,2,3,5,3,2,0,0,0,0,0,0,0,0];
END_IF;
```

Дополнительные модификации

1. Адаптивный взвешенный медианный фильтр

```
FUNCTION_BLOCK ADAPTIVE_WMF
EXTENDS WEIGHTED_MEDIAN_FILTER
VAR_INPUT
    NOISE_LEVEL: REAL := 0.1; // Оценка уровня шума (0..1)
END_VAR

METHOD UPDATE_WEIGHTS
// Адаптация весов в зависимости от уровня шума
VAR
    i, center: UINT;
    maxWeight: UINT;
END_VAR
BEGIN
    center := WINDOW_SIZE / 2 + 1;
    maxWeight := TRUNC(10.0 * (1.0 - NOISE_LEVEL) + 1.0);

    // Гауссово распределение весов
    FOR i := 1 TO WINDOW_SIZE DO
        WEIGHTS[i] := maxWeight - ABS(INT(i) - INT(center));
        IF WEIGHTS[i] < 1 THEN WEIGHTS[i] := 1; END_IF;
    END_FOR;
END_METHOD

BEGIN
    IF ENABLE THEN
        UPDATE_WEIGHTS();
```

```
    SUPER.UPDATE_FILTER();
  ELSE
    OUT := IN;
  END_IF;
END_FUNCTION_BLOCK
```

Ключевые особенности реализации

1. **Взвешенная медиана**:
    - Учет важности различных точек в окне;
    - Центральные значения имеют больший вес;
    - Эффективное подавление импульсных помех.
2. **Оптимизированный алгоритм**:
    - Сортировка вставками для малых окон;
    - Циклический буфер для хранения данных;
    - Минимальные вычислительные затраты.
3. **Гибкость**:
    - Настройка размера окна (3-15 значений);
    - Пользовательские весовые коэффициенты;
    - Поддержка динамического изменения параметров.
4. **Защитные механизмы**:
    - Проверка корректности размера окна;
    - Байпас при отключении;
    - Флаг заполненности буфера.

Рекомендации по применению

1. **Выбор размера окна**:
    - 3-5 значений для быстрых процессов;
    - 5-9 значений для большинства применений;
    - 9-15 значений для сильных шумов.
2. **Настройка весов**:
    - Симметричное распределение (например, [1,2,3,2,1]);
    - Гауссово распределение для лучшего подавления шума;

- Больший вес центральным значениям.

3. **Сравнение с другими фильтрами**:
   - Лучше обычного медианного против импульсных помех;
   - Сохраняет перепады лучше чем скользящее среднее;
   - Менее требователен к ресурсам чем Калмановский фильтр.
4. **Области применения**:
   - Обработка сигналов датчиков с импульсными помехами;
   - Фильтрация дискретных измерений;
   - Предварительная обработка в системах управления.

Пример для датчика вибрации

```
PROGRAM VIBRATION_MONITOR
VAR
   VibFilter: ADAPTIVE_WMF;
   RawVibration: REAL;
   FilteredVibration: REAL;
   NoiseEstimate: REAL;
END_VAR

// Инициализация адаптивного фильтра
VibFilter(
   WINDOW_SIZE := 7,
   Ts := 0.01
);

// Основной цикл обработки
NoiseEstimate := CALCULATE_NOISE(RawVibration);
VibFilter(
   IN := RawVibration,
   NOISE_LEVEL := NoiseEstimate,
   ENABLE := TRUE
```

);

FilteredVibration := VibFilter.OUT;

Данная реализация взвешенного медианного фильтра обеспечивает эффективное подавление шумов и выбросов при сохранении полезных сигналов, что делает его идеальным выбором для обработки зашумленных измерений в промышленных системах.

Подписывайтесь на нас в соцсетях:

Автоматика и робототехника - https://t.me/club_automate

Промышленная робототехника, роботизация производства - https://t.me/industrial_robot

Школа для электрика - https://t.me/electricalschool

Программируемые контроллеры, автоматизация - https://vk.com/club_plc

Сайт:

Школа для электрика - https://electricalschool.info/

Каталог технических курсов - https://electricalschool.info/skillforge.html