ПРОГРАММИРОВАНИЕ

СОДЕРЖАНИЕ

ПРОГРАММИРОВАНИЕ

ОСНОВНЫЕ ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

IEC 61131-3 – раздел международного стандарта МЭК 61131 (также существует соответствующий европейский стандарт EN 61131), описывающий языки программирования для программируемых логических контроллеров.

Целевое устройство (система) – аппаратное средство с определённой архитектурой процессора, на котором могут исполняться различные исполняемые файлы, обращающиеся с помощью него к модулям УСО.

Прикладная программа (исполняемый файл) для целевого устройства – скомпилированный и скомпонованный файл (*.bin, *.hex, *.so), который будет выполняться на целевом устройстве.

Проект – совокупность программных модулей (программ, функциональных блоков, функций), плагинов внешних модулей каналов В/В, ресурсов, пользовательских типов данных, сборка (компиляция и компоновка) которых, представляет собой прикладную программу для целевого устройства. Каждый проект сохраняется в отдельном файле.

Ресурс – элемент, отвечающий за конфигурацию проекта: глобальные переменные и экземпляры проекта, связываемыми с программными модулями типа «Программа» и задачами.

Программный модуль – элемент, представляющий собой функцию, функциональный блок или программу. Каждый программный модуль состоит из раздела объявлений и кода. Для написания всего кода программного используется только один из языков программирования стандарта IEC 61131-3.

Программа – программный модуль, представляющий собой единицу исполнения, как правило, связывается (ассоциируется) с задачей.

Экземпляр – представляет собой программу, как единицу исполнения, связанную (ассоциированную) с определённой задачей. Так же, как экземпляр, рассматриваются переменные, определённые в программных модулях: программа и функциональный блок.

Задача – элемент представляющий время и приоритет выполнения программного модуля типа «Программа» в рамках экземпляра проекта.

Функция – программный модуль, который возвращает только единственное значение, которое может состоять из одного и нескольких элементов (если это битовое поле или структура).

Функциональный блок – программный модуль, который принимает и возвращает произвольное число значений, а так же позволяет сохранять своё состояние (подобно классу в различных объектно-ориентированных языках). В отличие от функции функциональный блок не формирует возвращаемое значение.

ПРОГРАММИРОВАНИЕ

ОСНОВНЫЕ ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ (продолжение)

Переменная – область памяти, в которой находятся данные, с которыми оперирует программный модуль.

Пользовательский тип данных – тип данных, добавленный в проект и представляющий собой: псевдоним существующего типа, поддиапазон существующего типа, перечисление, массив или структуру.

POU (Program Organization Unit) – программный компонент, к которому относятся: функции, функциональные блоки, программы.

ПРОГРАММИРОВАНИЕ

РЕГИСТРЫ ДАННЫХ

Рабочие данные (регистры) в ПЛК располагаются в ОЗУ:

- Инициируются при запуске системы
  - значения «по-умолчанию»
  - считываются из энергонезависимой памяти (если предусмотрено)
- Обновляются в процессе работы системы
- Сбрасываются при перезапуске или выключении системы
  - предварительно сохраняются в энергонезависимой памяти (если предусмотрено)

Адресация регистров зависит от способа обращения к данным (протокола):

- ***сетевая адресация***
- ***локальная адресация***

Для доступа к регистрам по сети с помощью сетевых протоколов (например, ModBus RTU, ModBus TCP) принята числовая адресация — число в диапазоне 0 … 65535. Кроме того, в данном случае к адресации регистров добавляется адресация целевого ПЛК (например, IP-адрес, номер сетевого порта, адрес целевого устройства).

Для доступа к регистрам непосредственно из пользовательской программы (локальный доступ), обычно, принята более сложная адресация — буквенно-цифровая, например:

`[`**`код Зоны`** `памяти][`**`код Типа`** `данных][`**`Адрес`**`]`

где, Адрес — набор числовых кодов, разделенных символом точка.

Адрес			
код Группы регистров	код Канала В/В	код Подгруппы регистров	код Параметра

Поставщик информационной системы должен предоставить описание регистров:

- Карта адресов регистров (по группам):
  - локальный доступ
    - адрес
  - сетевой доступ (протокол)
    - имя таблицы с кодами поддерживаемых функций чтения/записи
    - адрес
    - порядок следования байт
    - описание
- Список применяемых кодов Зон памяти
- Список применяемых кодов Типов данных
- Список (сводный) применяемых Групп регистров

ПРОГРАММИРОВАНИЕ

РЕГИСТРЫ ДАННЫХ (продолжение)

Список применяемых кодов Зон памяти

Код зоны	Тип зоны	Описание
I	входы	Значения физических каналов ввода
M	память	Значения программных переменных и констант (коды, признаки, настройки, уставки, вычисления)
Q	выходы	Значения физических каналов вывода

Список применяемых кодов Типов данных

Код типа	Тип данных		Размер	
	IEC	C	байт	бит
X	BOOL	uint8_t	1	8
B	BYTE, USINT SINT	uint8_t int8_t	1	8
W	WORD, UINT INT	uint16_t int16_t	2	16
D	DWORD, UDINT DINT REAL	uint32_t int32_t float	4	32
L	LWORD, ULINT LINT LREAL	uint64_t int64_t double	48	64

ПРОГРАММИРОВАНИЕ

ТИПЫ ДАННЫХ БАЗОВЫЕ

Стандарт IEC 61131-3 поддерживает весь необходимый набор типов данных, аналогичных классическим языкам программирования.

Код типа	**Тип данных**		**Размер**		**Диапазон значений** *(формат)*
	IEC	**C**	**байт**	**бит**	
X	BOOL	uint8_t	1	8	FALSE, TRUE
B	BYTE, USINT SINT	uint8_t int8_t	1	8	0 … 255 -128 … 127
W	WORD, UINT INT	uint16_t int16_t	2	16	0 … 65535 -32768 … 32767
D	DWORD, UDINT DINT REAL	uint32_t int32_t float	4	32	0 … 4294967295 -2147483648 … 2147483647 $3.4\ x10^{-38}$ … $3.4\ x10^{38}$
L	LWORD, ULINT LINT LREAL	uint64_t int64_t double	48	64	0 … $(2^{64}-1)$ $-(2^{63}-1)$ … $(2^{63}-1)$ $1.7\ x10^{-308}$ ... $1.7\ x10^{308}$

Битовых полей нет, но ряд сред разработки в реализации языков программирования поддерживают явное побитовое обращение вида:

```
(* пример установки 3-го бита переменной a *)
a.3:= 1;
```

ПРОГРАММИРОВАНИЕ

ТИПЫ ДАННЫХ СПЕЦИАЛЬНЫЕ

Также стандарт IEC-61131-3 предоставляет специальные типы данных:

Тип данных	Формат значения	Описание
STRING	`'string'` `''`	***Строка*** *Данные обрамляются одинарными кавычками* *Пустая строка — две одинарные кавычки* *Доступна специальная библиотека функций*

TIME	`T#72h59m59s999ms`
	`T#120h`
	`T#4320m`
	`T#60000s`
	`T#5000ms`
		Время (длительность)
		T# - префикс
		h — часы
		m — минуты
		s — секунды
		ms — миллисекунды
		Каждая литера предваряется числом в диапазоне от 0 до 65535
		Если число перед литерой равно 0, то его вместе с самой литерой можно удалить
		Доступна специальная библиотека функций

DATE	`D#2021-12-31`	***Календарная дата*** D# - префикс 1-я группа (4 числа) — год 2-я группа (2 числа) — месяц (1 … 12) 3-я группа (2 числа) — день (1 … 31) *Формат цельный (0-е литеры удалять нельзя)* *Доступна специальная библиотека функций*
TIME_OF_DAY	`TOD#23:59:59`	***Время суток*** TOD# - префикс 1-я группа (2 числа) — часы (0 … 23) 2-я группа (2 числа) — минуты (0 … 59) 3-я группа (2 числа) — секунды (0 … 59) *Формат цельный (0-е литеры удалять нельзя)* *Доступна специальная библиотека функций*
DATE_AND_TIME	`DT#2021-12-31-23:59:59`	***Момент времени*** DT# - префикс *Значение - комплекс из D и TOD* *Формат цельный (0-е литеры удалять нельзя)*

ПРОГРАММИРОВАНИЕ

ТИПЫ ДАННЫХ ОБОБЩЕННЫЕ

Стандарт IEC 61131-3 поддерживает обобщенные типы данных (для универсальности), которые доступны только для переменных В/В, определяемых в POU:

- ***ANY***
  - ***ANY_NUM***
    - ANY_INT
      INT, SINT, DINT LINT
      UINT, USINT, UDINT ULINT
  - ***ANY_REAL***
    REAL, LREAL
- ***ANY_DATE***
  DATE
  DATE_AND_TIME
- ***ANY_TIME***
  TIME
  TIME_OF_DAY

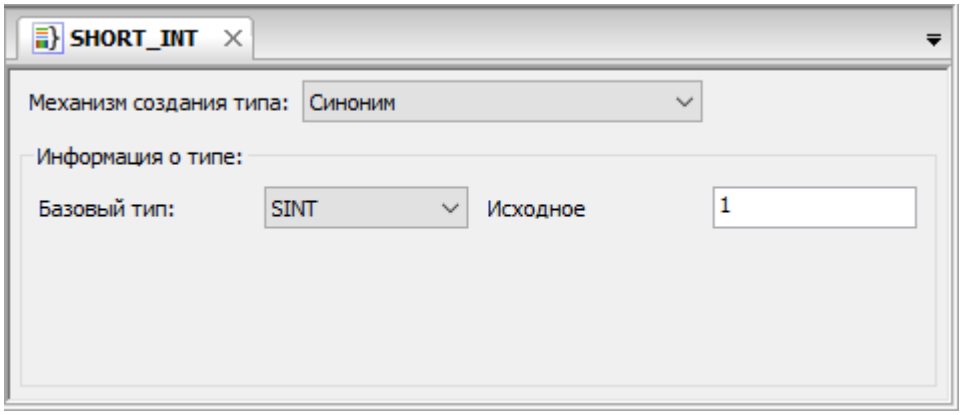
ПРОГРАММИРОВАНИЕ

ТИПЫ ДАННЫХ ПОЛЬЗОВАТЕЛЬСКИЕ

Ряд сред разработки предоставляют возможности определения (структурированных) пользовательских типов данных:

Синоним

- Указывается имя нового типа (синонима)
- Выбирается базовый (наследуемый) тип данных
- Указывается начальное значение
  *аналогия **typedef** в языке Си*

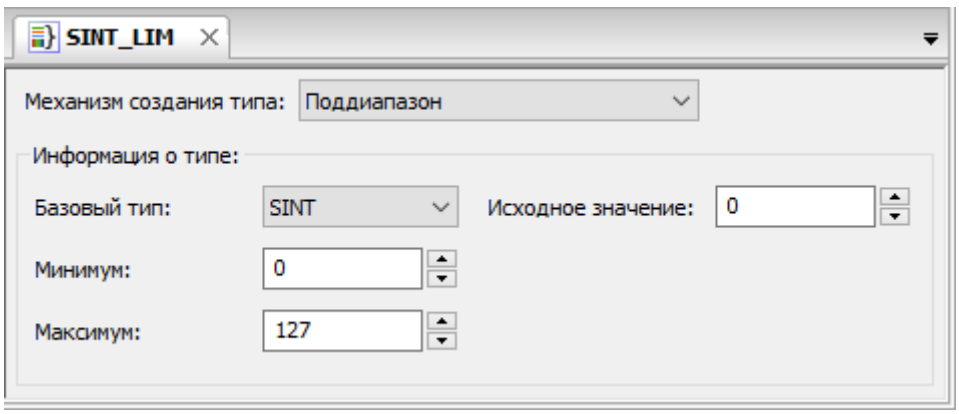


определение синонима SHORT_INT для базового типа SINT

Ряд сред разработки предоставляют возможности определения (структурированных) пользовательских типов данных:

Поддиапазон

- Указывается имя нового типа (поддиапазона)
- Выбирается базовый (наследуемый) тип данных
  только для целочисленных
- Указывается диапазон возможных значений
  - минимум
  - максимум
- Указывается начальное значение



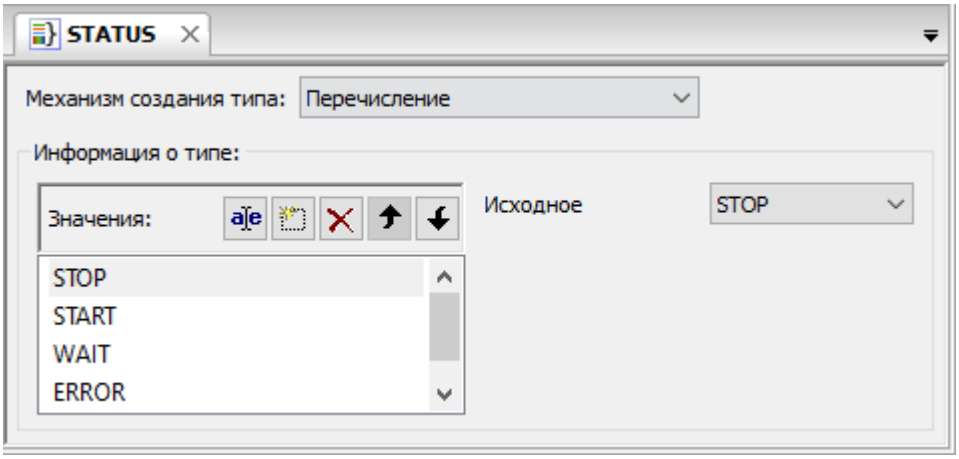
определение поддиапазона SINT_LIM для базового типа SINT

ПРОГРАММИРОВАНИЕ

ТИПЫ ДАННЫХ ПОЛЬЗОВАТЕЛЬСКИЕ (продолжение)

Перечисление

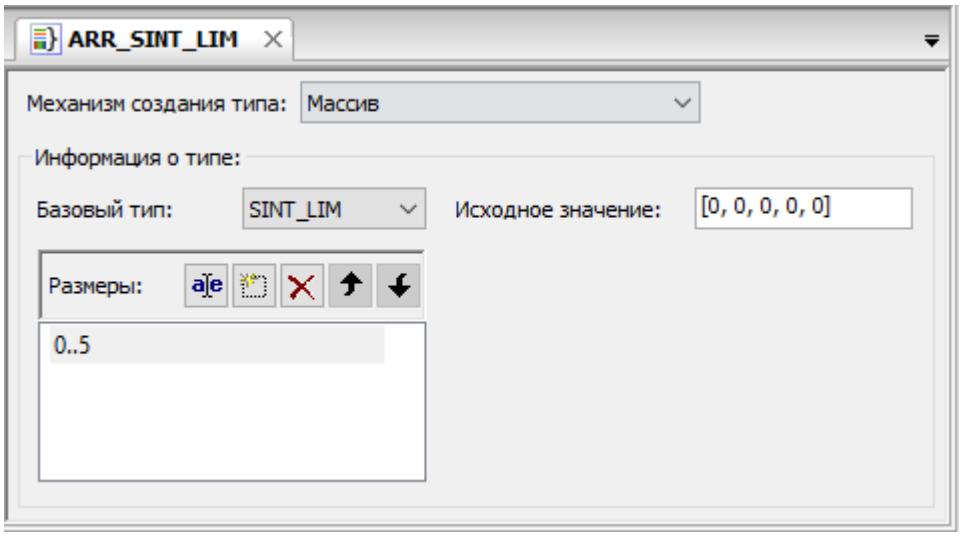
- Указывается имя нового типа (перечисления)
- Определяется список именнованных значений
  каждому значению списка (скрыто и автоматически) присваивается уникальный целочисленный код (0 … n)
- Указывается начальное значение
  *аналогия **enum** в языке Си*



определение перечисления STATUS

Массив

- Указывается имя нового типа (массива)
- Задается тип
  базовый, либо ранее созданный пользовательский
- Задается размерность массива
  начало..конец
- Указывается начальное значение
  [0-значение, 1-значение, ..., n-значение]



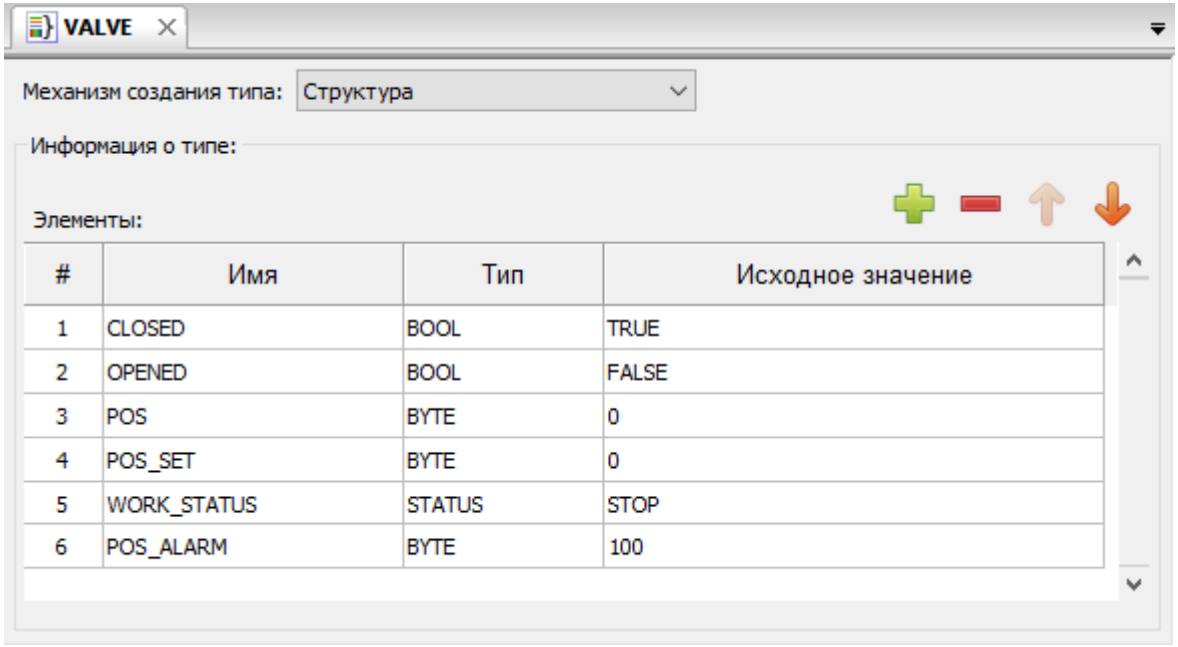
определение массива ARR_SINT_LIM

ПРОГРАММИРОВАНИЕ

ТИПЫ ДАННЫХ ПОЛЬЗОВАТЕЛЬСКИЕ (продолжение)

Структура

- Указывается имя нового типа (структуры)
- Определяется таблица полей структуры
  каждое поле имеет свое Имя, Тип и Начальное значение
  *аналогия **struct** в языке Си*



#	Имя	Тип	Исходное значение
1	CLOSED	BOOL	TRUE
2	OPENED	BOOL	FALSE
3	POS	BYTE	0
4	POS_SET	BYTE	0
5	WORK_STATUS	STATUS	STOP
6	POS_ALARM	BYTE	100

определение структуры VALVE

ПРОГРАММИРОВАНИЕ

ЯЗЫКИ ПРОГРАММИРОВАНИЯ IEC 61131-3

Международный стандарт IEC 61131-3 описывает языки программирования для программируемых логических контроллеров (ПЛК).

Первая редакция стандарта вышла в 1993 году.
Вторая редакция — 2003.
Третья редакция — 2012.

Ключевые изменения последней редакции:

- Определены элементарные типы данных
- Добавлен массив с измеряемой длиной
- Добавлена инициализация переменных
- Возможность вызова функций без возврата результата
- Определены объектно-ориентированные функциональные блоки (FB)
- Поддержка пространства имен
- Язык «IL» (Список инструкций) считается устаревшим

Стандарт определяет пять (5) языков программирования:

1) ST (Structured Text)
2) IL (Instruction List)
3) LD (Ladder Diagram)
4) FBD (Function Block Diagram)
5) SFC (Sequential Function Chart)

Для описания некоторых конструкций языка используются фигурные и квадратные скобки. Считается, что:

- Выражение в ***< фигурных скобках >*** может использоваться ноль и более раз подряд;
- Выражение в ***[квадратных скобках]*** не обязательно к использованию.

При написании программ возможно использование различных библиотечных (стандартных, системных, аппаратнозависимых) и пользовательских функций и/или функциональных блоков.

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ

Общие сведения

- ***ST***
- ***Текстовый язык***
- Основные принципы — ***язык Pascal***
- Удобен для программ, включающих числовой анализ или сложные алгоритмы
- Основные элементы:
  - ***ключевые слова***
    в верхнем регистре
  - ***символы различных операторов***
  - ***символьно-числовые имена переменных и констант***
    имя должно начинаться с буквенного символа
  - ***значения***
    базовые, специальные и пользовательские типы данных
  - *символы пробела и табуляции не влияют на синтаксис (допустимы)*
  - *символы переноса строк не влияют на синтаксис в блочных конструкциях*
- Порядок выполнения справа налево, сверху вниз

Программа представляется в виде текста, по синтаксису схожего с языком Pascal. Нотация близка специалистам по прикладному программированию.

Язык предоставляет следующие конструкции:

- Арифметические операции
- Логические (побитовые) операции
- Операции сравнения
- Операция присвоения
- Конструкция IF – ELSEIF – ELSE
- Цикл FOR
- Цикл WHILE
- Цикл REPAET UNTIL
- Конструкция CASE
- Комментарий

Арифметические операции

- **+** сложение
- **-** вычитание
- ***** умножение
- **/** деление
- **mod** остаток от целочисленного деления

Приоритет операций в выражениях определен (см. таблицу к п.Присвоение). Чем выше приоритет, тем раньше исполняется операция.

При записи арифметических выражений допустимо использование скобок для указания порядка вычислений.

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Логические (побитовые) операции

- ***OR*** - логическое (побитовое) сложение
- ***AND*** - логическое (побитовое) умножение
- ***XOR*** - логическое (побитовое) «исключающее ИЛИ»
- ***NOT*** - логическое (побитовое) отрицание

Операции сравнения

- **=** – сравнение на «равенство»
- **<>** – сравнение на «неравенство»
- **>** – сравнение на «больше»
- **>=** – сравнение на «не меньше»
- **<** – сравнение на «меньше»
- **<=** – сравнение на «не больше»

В качестве результата сравнения всегда используется значение типа BOOL.

Рекомендуется как можно меньше использовать сравнения: >= или <=, так как они являются более затратными.

Присвоение

Для обозначения присвоения используется парный знак «:=» (двоеточие равно).

В правой и левой части выражения должны быть операнды одного типа (автоматического приведения типов не предусмотрено).

В левой части выражения (принимающая сторона) может быть использована только переменная.

Правая часть может содержать выражение или константу.

Правая часть должна завершаться символом «;» (точка с запятой).

```
Variable:= Value;
```

Операция	**Приоритет**
Сравнение	1
Сложение, вычитание	2
Умножение, деление	3
OR	4
AND, XOR	5
NOT	6
Унарный минус (-x)	7
Вызов функции	8

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Конструкция IF — ELSEIF - ELSE

Служит для выполнения выражений в случае выполнения ряда условий.

```
IF <Boolean Expression> THEN
  <Statement List>
[ELSEIF <Boolean Expression> THEN
  <Statement List>]
[ELSE
  <Statement List>]
END_IF;
```

Где,

<Boolean Expression> - логическое условие, состоящее из логических операций и/или операций сравнения; для удобства восприятия синтаксиса, условие может быть обрамлено круглыми скобками.

<Statement List> - список выражений и/или конструкций, которые выполняются при соблюдении условия.

Внутри конструкции могут находиться любые выражения и/или конструкции, в том числе, и еще от одного до нескольких (вложенных) IF.

Пример:

```
IF Var > 10 THEN
  IF Var < Var2 + 1 THEN
    Var:= 10;
  ELSE
    Var:= 0;
  END_IF;
END_IF;
```

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Цикл FOR

Служит для задания цикла с фиксированным количеством шагов (итераций).

```
FOR <Control Variable>:= <Expression1> TO <Expression2> [BY <Expression3>] DO
  <Statement List>
END_FOR;
```

Где,

<Control Variable> - переменная приращения цикла [INT]

<Expression1> - стартовое значение переменной приращения цикла [INT]

<Expression2> - конечное значение переменной приращения цикла [INT]

<Expression3> - шаг приращения переменной цикла [INT]:

= положительное число для инкремента переменной цикла

= отрицательное число для декремента переменной цикла

= 1, если шаг не задан

<Statement List> - список выражений и/или конструкций, которые выполняются на каждой итерации

Цикл стартует с <Control Variable>:= <Expression1>

1) Выполняется список выражений и/или конструкций
2) Значение <Control Variable> автоматически увеличивается на шаг <Expression3>

Цикл повторяется до тех пор, пока <Control Variable> не достигнет значения <Expression2>

Цикл можно прервать досрочно, используя внутри конструкции оператор ***EXIT***.

Внутри конструкции могут находиться любые выражения и/или конструкции, в том числе, и еще от одного до нескольких (вложенных) FOR.

Пример:

```
FOR i:= 1 TO 10 BY 2 DO
  k:= k*2;
  IF (k > 20) THEN
    EXIT;
  END_IF;
END_FOR;
```

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Цикл WHILE

Служит для задания цикла с предусловием.

```
WHILE <Boolean Expression> DO
  <Statement List>
END_WHILE;
```

Где,

<Boolean Expression> - логическое условие (предусловие), состоящее из логических операций и/или операций сравнения; для удобства восприятия синтаксиса, условие может быть обрамлено круглыми скобками.

<Statement List> - список выражений и/или конструкций, которые выполняются на каждой итерации.

Цикл стартует, если выполняется логическое условие (на момент старта).

1) Выполняется список выражений и/или конструкций
2) Проверяется логическое условие

Цикл повторяется до тех пор, пока предусловие возвращает TRUE.

Цикл можно прервать досрочно, используя внутри конструкции оператор ***EXIT***.

Внутри конструкции могут находиться любые выражения и/или конструкции, в том числе, и еще от одного до нескольких (вложенных) WHILE.

Пример:

```
k:= 10;

WHILE (k > 0) DO
  i:= i + k;
  k:= k – 1;
  IF (i > 99) THEN
    EXIT;
  END_IF;
END_FOR;
```

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Цикл REPEAT UNTIL

Служит для задания цикла с постусловием. Аналог DO WHILE языка Си.

```
REPEAT
  <Statement List>
UNTIL <Boolean Expression>;
END_REPEAT;
```

Где,

<Boolean Expression> - логическое условие (постусловие), состоящее из логических операций и/или операций сравнения; для удобства восприятия синтаксиса, условие может быть обрамлено круглыми скобками.

<Statement List> - список выражений и/или конструкций, которые выполняются на каждой итерации.

Цикл стартует всегда (один цикл будет выполнен в любом случае).

1) Выполняется список выражений и/или конструкций
2) Проверяется логическое условие

Цикл повторяется до тех пор, пока постусловие возвращает TRUE.

Цикл можно прервать досрочно, используя внутри конструкции оператор ***EXIT***.

Внутри конструкции могут находиться любые выражения и/или конструкции, в том числе, и еще от одного до нескольких (вложенных) REPEAT UNTIL.

Пример:

```
k:= 10;

REPEAT
  i:= i + k;
  k:= k – 1;
  IF (i > 99) THEN
    EXIT;
  END_IF;
UNTIL (k = 0)
END_REPEAT;
```

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Конструкция CASE

Служит для задания выбора из диапазона значений. Аналог SWITCH CASE языка Си.

```
CASE <Expression> OF
  CASE_ELEMENT {CASE_ELEMENT}:
    <Statement List>
  [ELSE <Statement List>]
END_CASE;
```

Где,

<Expression> - значение, которое проверяется на вхождение в заданный диапазон.

CASE_ELEMENT — набор диапазонов значений, который может включать в себя одно число, и/или несколько чисел, перечисленных через запятую, и/или диапазон чисел (начало..конец); наборы значений в пределах одного описания CASE_ELEMENT разделяется запятой (,); в конце описания набора ставится двоеточие (:).

<Statement List> - список выражений и/или конструкций, которые выполняются при совпадении <Expression> с CASE_ELEMENT.

При задании CASE_ELEMENT необходимо соблюдать следующие правила:

- наборы значений в пределах всей конструкции не должны пересекаться
- при указании диапазона значений начало диапазона должно быть меньше его конца

Внутри конструкции могут находиться любые выражения и/или конструкции, в том числе, и еще от одного до нескольких (вложенных) CASE.

Пример:

```
CASE k OF
  1:
    k := k * 10;

  2..5:
    k := k * 5;
    I := 0;

  6, 9..20:
    k := k – 1;

  ELSE
    k := 0;
    i := 1;
END_CASE;
```

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Неправильная запись	Правильная запись
01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 5, 9..20: 08: k := k - 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;	01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 6, 9..20: 08: k := k - 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;
Диапазоны в строках 04 и 07 пересекаются	

Неправильная запись	Правильная запись
01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 6, 20..9: 08: k := k - 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;	01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 6, 9..20: 08: k := k - 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;
В строке 07 диапазон значений задан неправильно	

ПРОГРАММИРОВАНИЕ

STRUCTURED TEXT / СТРУКТУРИРОВАННЫЙ ТЕКСТ (продолжение)

Комментарий

В языке ST доступны однострочные комментарии, которые задаются в виде специально оформленной строки:

```
(* Текст комментария: строка 1 *)
(* Текст комментария: строка 2 *)
(* Текст комментария: строка 3 *)
```

ПРОГРАММИРОВАНИЕ

INSTRUCTION LIST / СПИСОК ИНСТРУКЦИЙ

Общие сведения

- ***IL***
- ***Текстовый язык***
- Основные принципы — ***язык Assembler***
- Основные элементы:
  - ***ключевые слова***
    в верхнем регистре
  - ***переходы по меткам***
  - ***аккумулятор***
  - ***символьно-числовые имена переменных и констант***
    имя должно начинаться с буквенного символа
  - ***значения***
    базовые, специальные и пользовательские типы данных
  - *символы пробела и табуляции не влияют на синтаксис (допустимы)*
  - *символы переноса строк не влияют на синтаксис в блочных конструкциях*
- Порядок выполнения справа налево, сверху вниз

Программа представляется в виде списка инструкций процессора (по факту к конкретной архитектуре процессора привязки здесь нет).
Нотация близка специалистам по низкоуровневому программированию.

Данный язык не удобен для реализации сложных алгоритмов с большим количеством ветвлений.

Основа языка, как и в случае с Assembler, это переходы по меткам и аккумулятор. В аккумулятор загружается значение переменной, а дальнейшее выполнение алгоритма представляет собой извлечение значения из аккумулятора и совершение над ним определенного рода действий.

В третьей редакции стандарта IEC 61131-3 язык IL помечен как необязательный (соответственно, он может не поддерживаться средой разработки). Поэтому, подробно рассматривать особенности синтаксиса этого языка здесь не будем.

пример программы на языке IL

В примере выше реализовано выражение, эквивалентное выражению на языке ST:

```
C:= A AND NOT B;
```

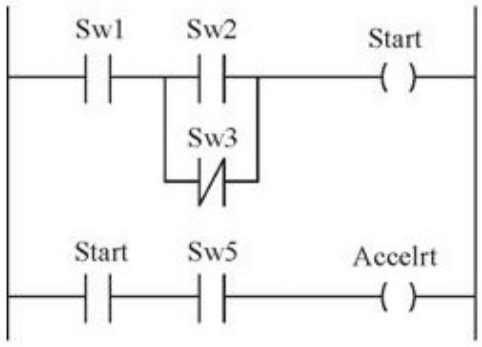
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ

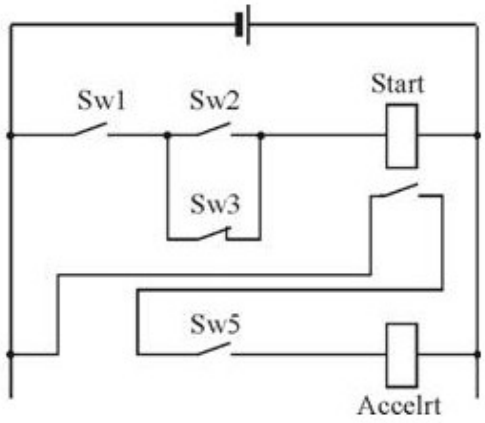
Общие сведения

- ***LD***
- ***Графический язык***
- Основные принципы - ***релейно-контактные схемы***
- Основные элементы:
  - ***шины питания***
  - ***контакты***
  - ***обмотки (катушки) реле***
  - ***горизонтальные и вертикальные перемычки***
  - ***связанные переменные типа BOOL***
- Поддерживается возможность использования различных функциональных блоков

Программа представляется в виде электрического потока.
Нотация близка специалистам по электронике.



программа на языке LD



эквивалентная электрическая схема

Схемы, реализованные на данном языке, называются многоступенчатыми. Они представляют собой набор горизонтальных цепей, напоминающих ступеньки лестницы, соединяющих вертикальные шины питания.

Объекты языка программирования LD обеспечивают средства для структурирования программного модуля в некоторое количество контактов и катушек.

Порядок обработки объектов LD-секции:

- сверху вниз
  контакты связанные с функциональными блоками обрабатываются в порядке следования вводов/выводов соответсвтующего блока

Слева и справа схема ограничена вертикальными линиями — шинами питания.
Значение левой шины всегда TRUE (условно можно обозначить «+ Питания»).

Между шинами питания расположены цепи, образованные контактами и катушками реле (по аналогии с обычными электронными цепями).

Слева любая цепь начинается набором контактов.
Справа любая цепь заканчивается набором катушек.

ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

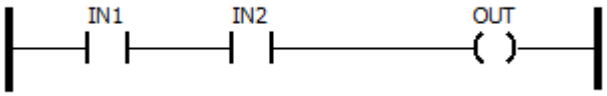
Каждому контакту и катушке привязана логическая переменная (типа BOOL). Каждая логическая переменная, в свою очередь, может быть привязана к определенному каналу В/В целевой системы (***к контактам привязывают каналы ввода, к катушкам — каналы вывода***).

Контакты в зависимости от состояния (TRUE или FALSE) пропускают или не пропускают через себя «ток» от левой шины питания.

Катушки принимают соответствующий сигнал (TRUE или FALSE) от левой схемы и замыкаются на правую шину питания. Если катушка получает значение TRUE, то логическая цепь считаетая замкнутой (истинной).

Последовательное или параллельное соединение контактов и/или катушек реализует различные логические операции (И, ИЛИ).

Пример:



Пример представляет собой реализацию логического выражения, которое можно записать на языке ST следующим образом:

```
OUT:= (IN1 AND IN2);
```

Язык позволяет:

- выполнять последовательное соединение контактов
- выполнять параллельное соединение контактов
- применять нормально разомкнутые или замкнутые контакты
- использовать переключаемые контакты
- записывать комментарии
- использовать переходы
- включать в схему все возможные элементы языка функционльных диаграмм (FBD)
- управлять работой функциональных блоков по входам «EN/ENO»

Контакт

Это LD-элемент, который передает состояние горизонтальной связи левой стороны к горизонтальной связи на правой стороне (результат булевой операции AND).

Состояние левой связи и функция самого контакта не изменяют переменную, связанную с этим контактом.

Контакт имеет следующие модификаторы:

- Нормальный (нормально-разомкнутое состояние)
- Инвесный (нормально-замкнутое состояние)
- Нарастающий фронт
- Спадающий фронт

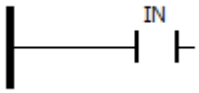
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

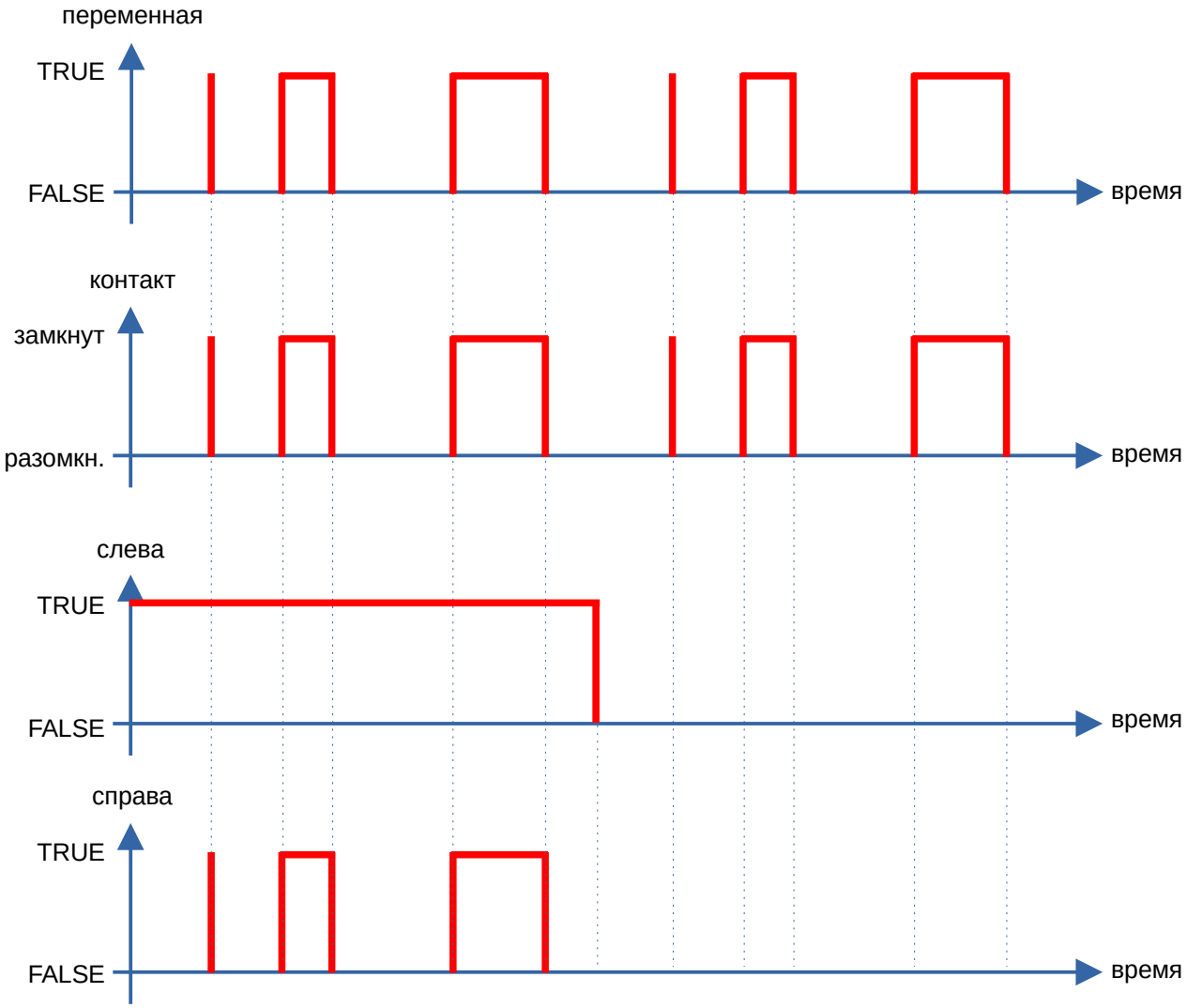
Нормальный контакт

Связанная переменная (VAR) управляет контактом через функцию-модификатор:

- VAR = TRUE
  - контакт замкнут
    - левый сигнал проходит
    - справа = слева
- VAR = FALSE
  - контакт разомкнут
    - левый сигнал не проходит
    - справа = FALSE



нормальный контакт



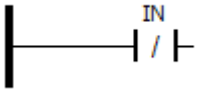
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

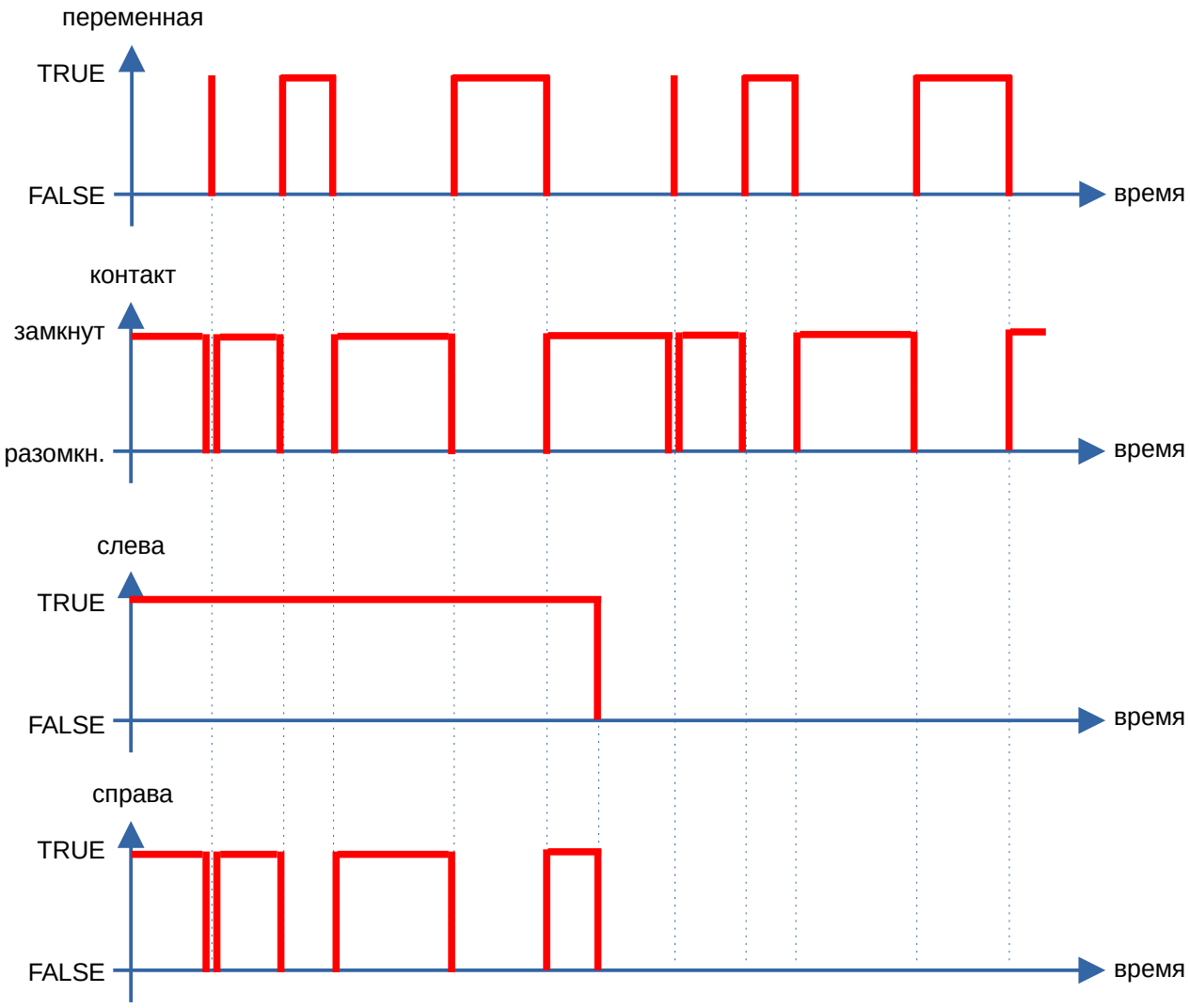
Инверсный контакт

Связанная переменная (VAR) управляет контактом через функцию-модификатор:

- VAR = TRUE
  - контакт разомкнут
    - левый сигнал не проходит
    - справа = FALSE
- VAR = FALSE
  - контакт замкнут
    - левый сигнал проходит
    - справа = слева



инверсный контакт



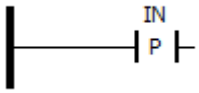
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

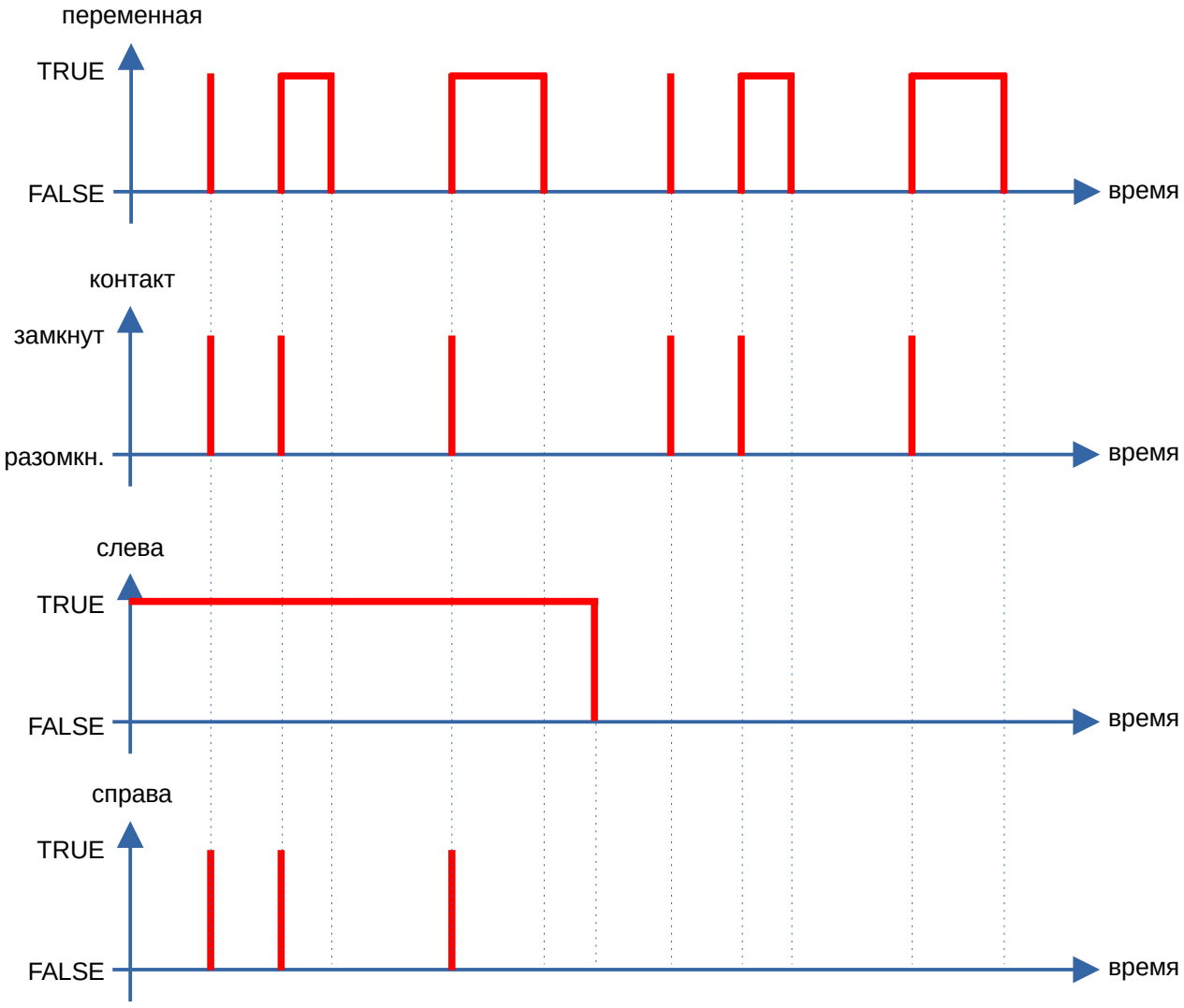
Контакт с обнаружением нарастающего фронта

Связанная переменная (VAR) управляет контактом через функцию-модификатор:

- VAR = TRUE
  - контакт замкнут только в момент перехода FALSE › TRUE
    (в пределах одного цикла работы программы)
    левый сигнал проходит
    справа = слева
- VAR = FALSE
  - контакт разомкнут
    левый сигнал не проходит
    справа = FALSE



контакт с обнаружением нарастающего фронта



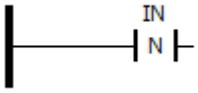
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

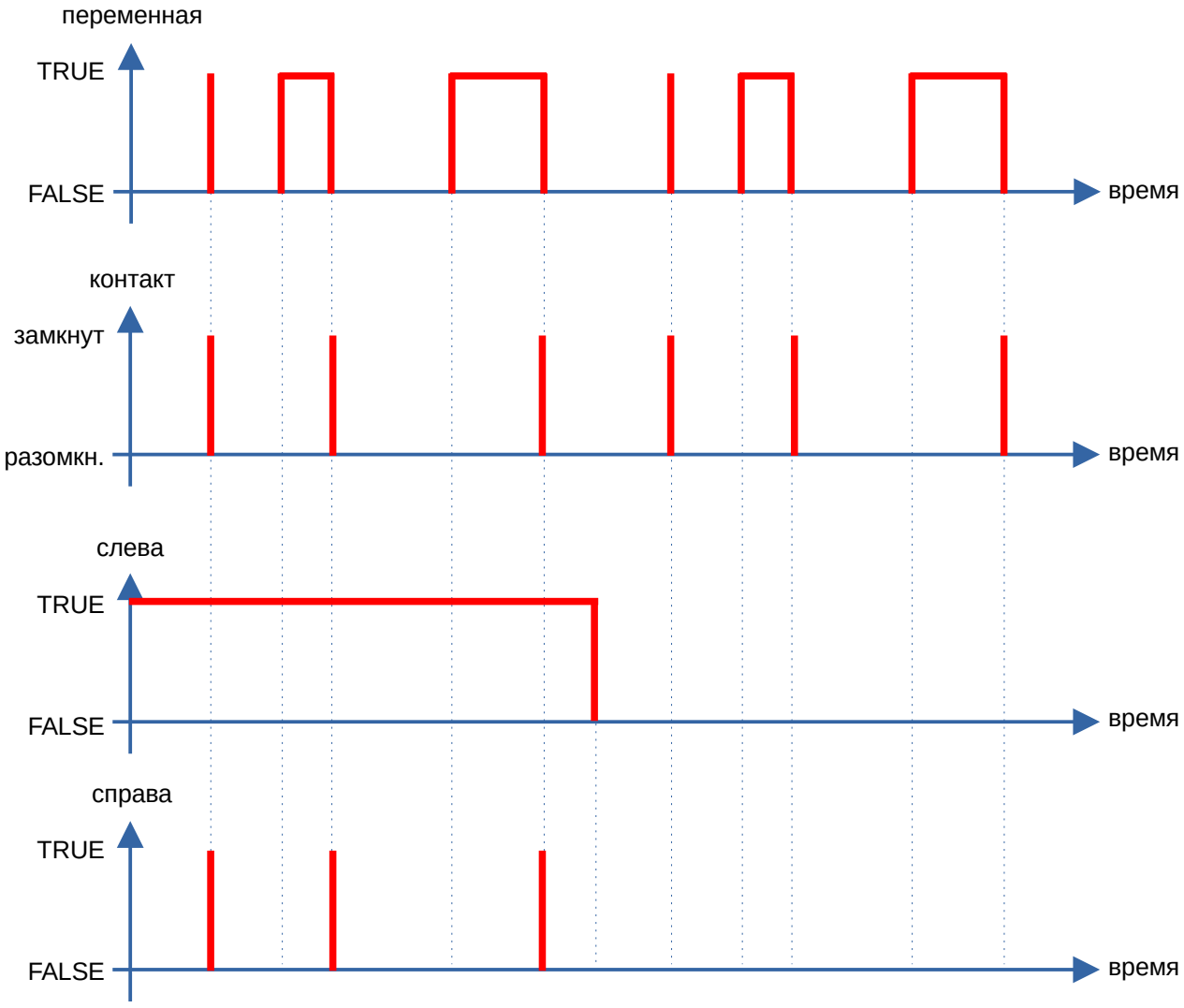
Контакт с обнаружением спадающего фронта

Связанная переменная (VAR) управляет контактом через функцию-модификатор:

- VAR = TRUE
  - контакт замкнут только в момент перехода TRUE › FALSE
    (в пределах одного цикла работы программы)
    левый сигнал проходит
    справа = слева
- VAR = FALSE
  - контакт разомкнут
    левый сигнал не проходит
    справа = FALSE



контакт с обнаружением спадающего фронта



ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

Катушка

Это LD-элемент, который принимает состояние горизонтальной связи левой стороны и передает ее к неизменной горизонтальной связи на правой стороне.

За катушкой всегда должна располагаться правая (конечная) шина.

Состояние левой связи и функция катушки изменяют значение переменной, связанной с этой катушкой.

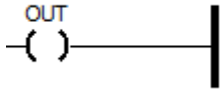
Катушка имеет следующие модификаторы:

- Нормальный (нормально-разомкнутое состояние)
- Инвесный (нормально-замкнутое состояние)
- Установка
- Сброс
- Нарастающий фронт
- Спадающий фронт

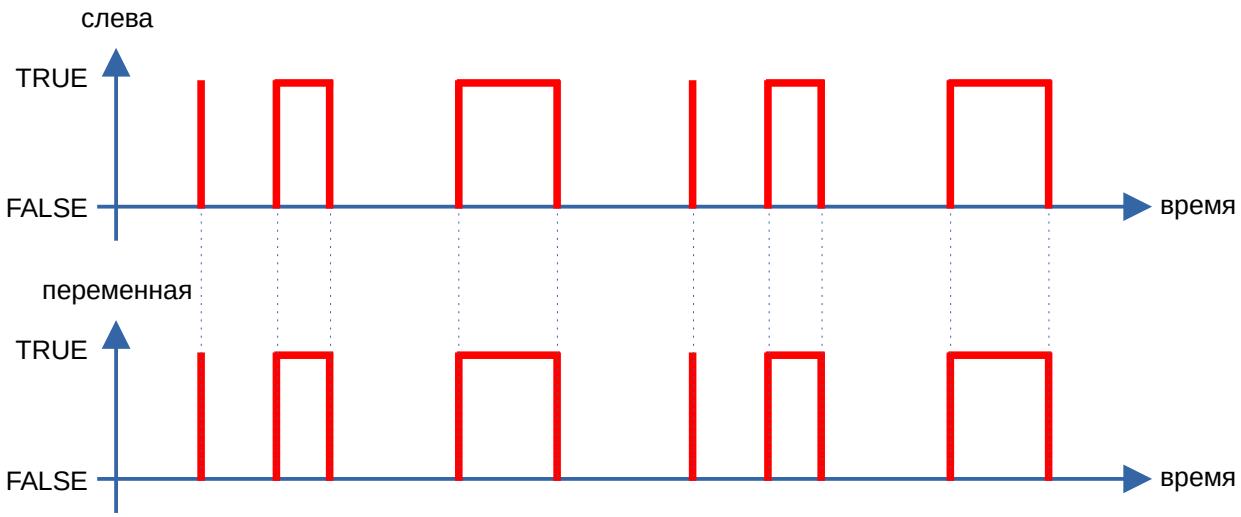
Нормальная катушка

Левый сигнал изменяет связанную переменную катушки (VAR) через функцию-модификатор:

- VAR:= слева



нормальная катушка



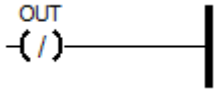
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

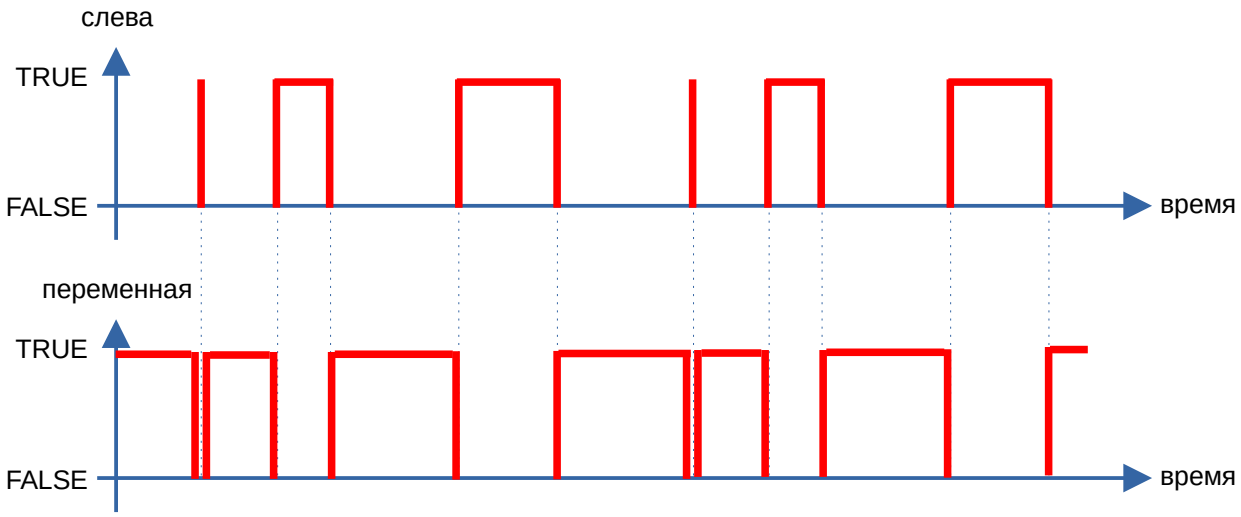
Инверсная катушка

Левый сигнал изменяет связанную переменную катушки (VAR) через функцию-модификатор:

- VAR:= NOT(слева)



инверсная катушка



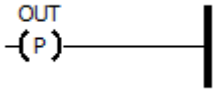
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

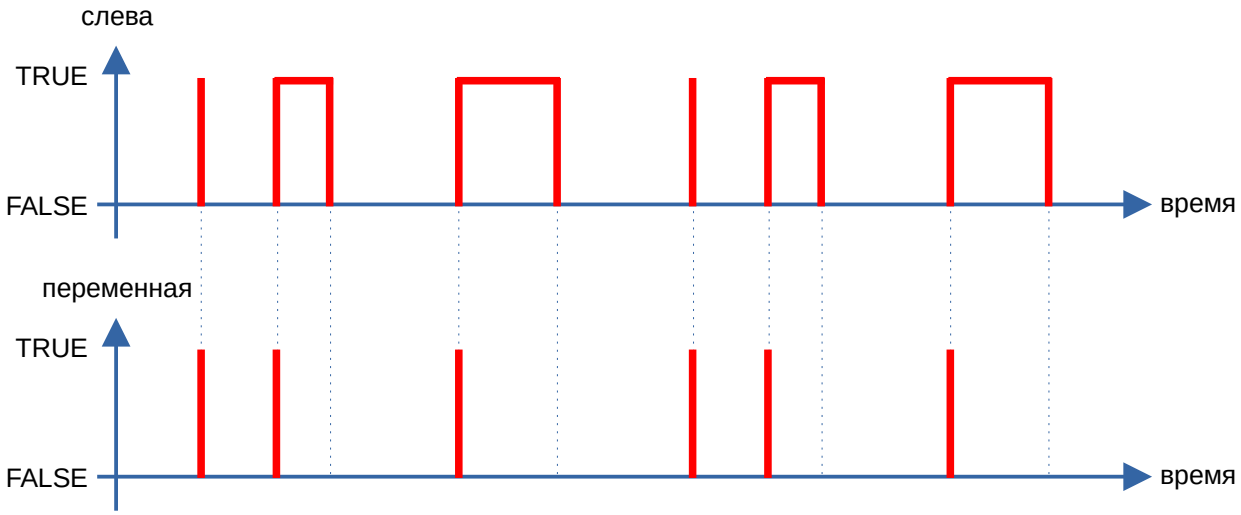
Катушка с обнаружением нарастающего фронта

Левый сигнал изменяет связанную переменную катушки (VAR) через функцию-модификатор:

- В момент перехода левого сигнала с FALSE › TRUE
  (в пределах одного цикла работы программы)
  - VAR:= TRUE
- В остальных случаях
  - VAR:= FALSE



катушка с обнаружением нарастающего фронта



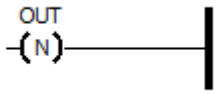
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

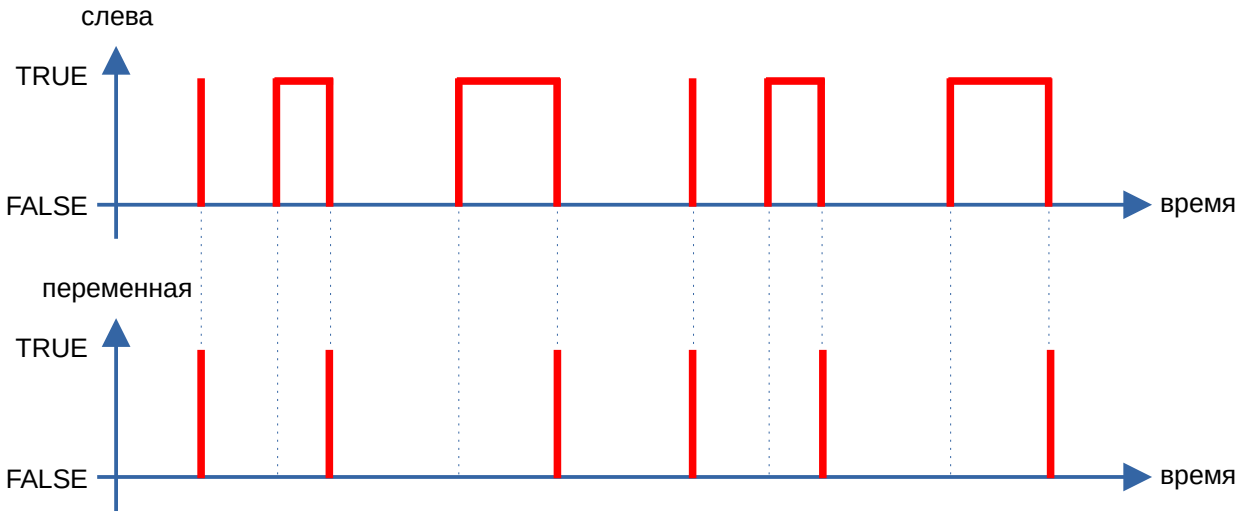
Катушка с обнаружением спадающего фронта

Левый сигнал изменяет связанную переменную катушки (VAR) через функцию-модификатор:

- В момент перехода левого сигнала с TRUE › FALSE
  (в пределах одного цикла работы программы)
  - VAR:= TRUE
- В остальных случаях
  - VAR:= FALSE



катушка с обнаружением спадающего фронта



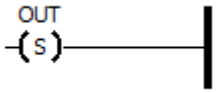
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

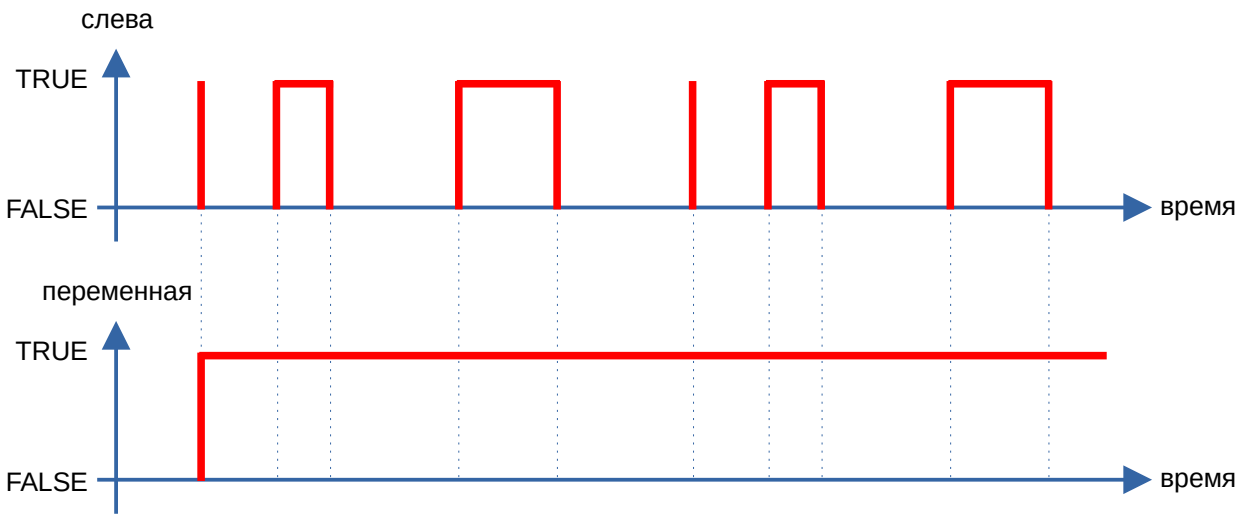
Катушка установки

Левый сигнал изменяет связанную переменную катушки (VAR) через функцию-модификатор:

- При сброшенном флаге захвата связанной переменной
  и при переходе левого сигнала с TRUE › FALSE
  (в пределах одного цикла работы программы)
  - VAR:= TRUE
  - устанавливается флаг захвата связанной переменной
    (дальнейшие изменения левого сигнала на связанную переменную не влияют;
    для сброса переменной в FALSE использовать катушку сброса)



катушка установки



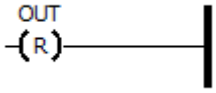
ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

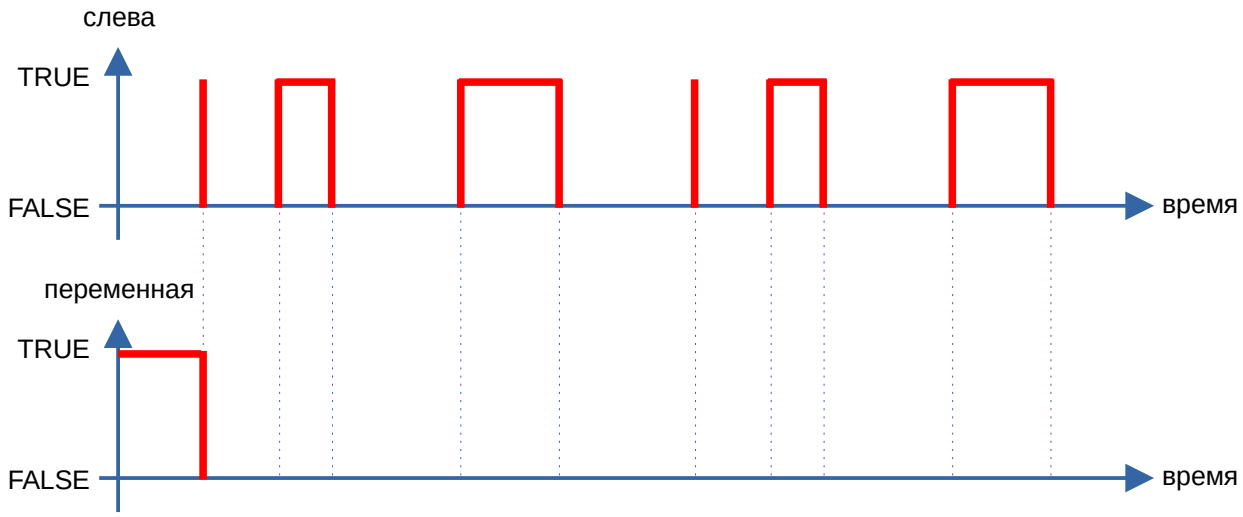
Катушка сброса

Левый сигнал изменяет связанную переменную катушки (VAR) через функцию-модификатор:

- При переходе левого сигнала с TRUE › FALSE
  (в пределах одного цикла работы программы)
  - VAR:= FALSE
  - сбрасывается флаг захвата связанной переменной



катушка сброса

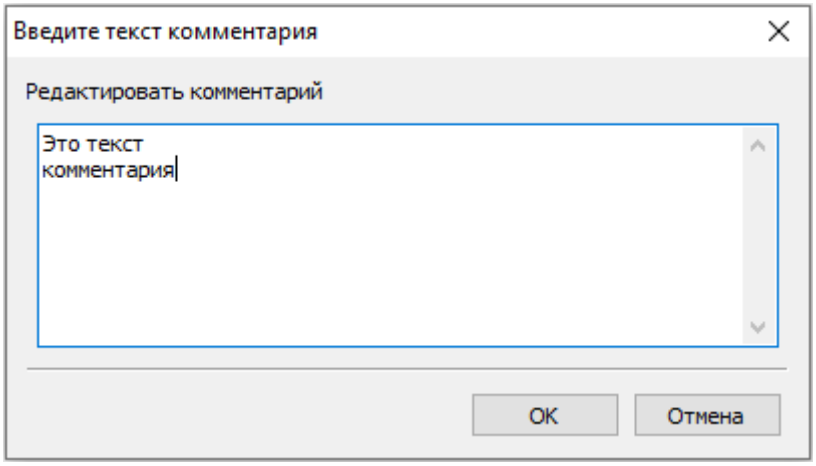


ПРОГРАММИРОВАНИЕ

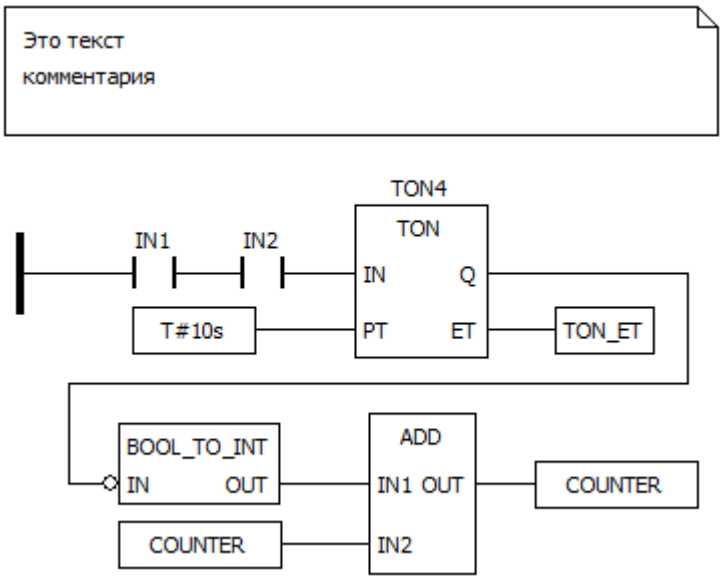
LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

Комментарий

В языке LD доступны многострочные комментарии, которые редактируются в специальном «Редакторе комментариев», а в самом коде отображаются в виде специально оформленного блока.



редактор комментария



блок комментария в исходном коде редактора LD

ПРОГРАММИРОВАНИЕ

LADDER DIAGRAM / РЕЛЕЙНО-КОНТАКТНЫЕ СХЕМЫ (продолжение)

Переход

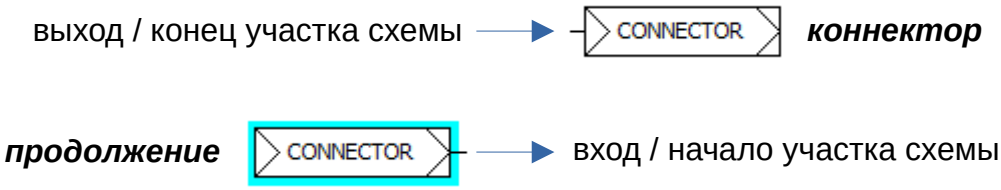
Для передачи значения из одного участка схемы на другой без прямого соединения выхода и входа в языке LD используются элементы «CONNECTION» (соединения).

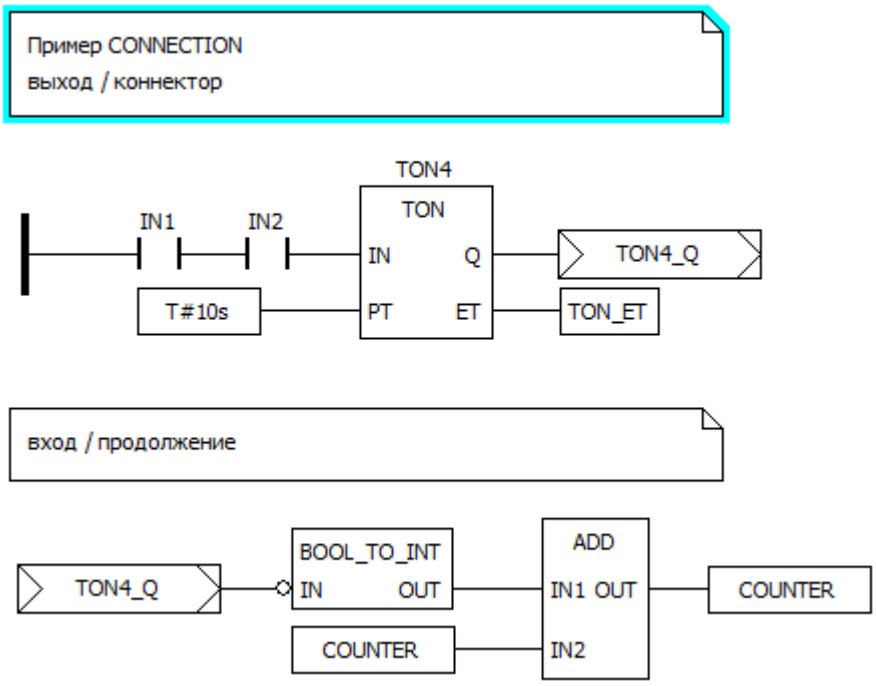
Соединение состоит из пары элементов:

- ***Коннектор*** (выход / конец одного участка схемы)
- ***Продолжение*** (вход / начало другого участка схемы)

«Коннектор» — исходная точка (выход потока).
«Продолжение» — конечная точка (вход).

Имена элементов «Коннектор» и «Продолжение» должны быть одинаковые.





ПРОГРАММИРОВАНИЕ

FUNCTION BLOCK DIAGRAM / ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ

Общие сведения

- ***FBD***
- ***Графический язык***
- Основные принципы — ***объектно-ориентированные блоки***
- Основные элементы:
  - ***переменные***
  - ***блоки функций***
  - ***соединения***

Программа представляется в виде потоковой диаграммы связанных между собой переменных и блоков функций.

Нотация близка специалистам-технологам.

Язык позволяет использовать мощные алгоритмы простым вызовом функционального блока (например, алгоритмы ПИД-регулятора, регулятора нечеткой логики, меню графического экрана — реализуются в виде единичных блоков).

FBD является более эффективным для представления структурной информации, чем язык LD. Программа на языке FBD не поддерживает использования нотации языка LD. В то же время, программа на языкыке LD поддерживает нотацию FBD в полной мере, что расширяет возможности языка LD. Поэтому, ***для расширения возможностей реализации логики рекомендуется создавать программы на языке LD***.

Блоки бывают следующих видов (определяется при создании блока):

- ***Функциональный блок***
- ***Блок функции***

Язык позволяет:

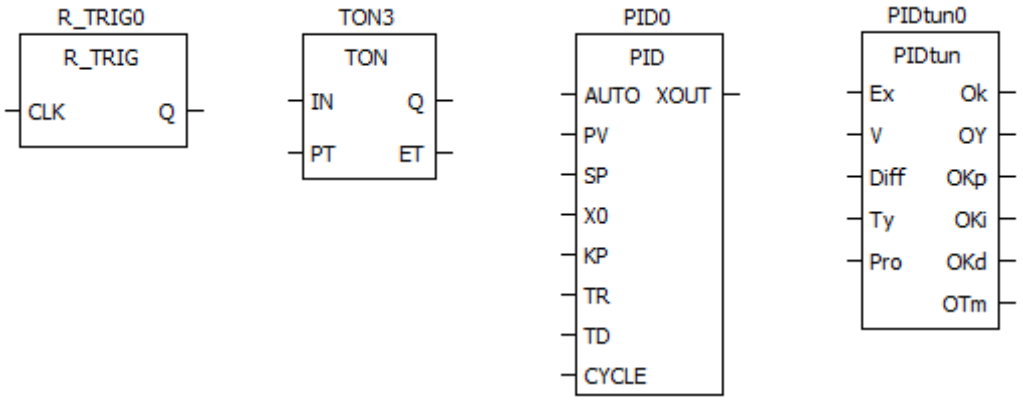
- включать в диаграмму переменные
- включать в диаграмму функциональные блоки и блоки функций
- выполнять последовательное и параллельное соединение переменных и блоков
- применять к контактам блоков различные модификаторы
- записывать комментарии
- использовать переходы
- управлять работой блоков по входам «EN/ENO»

ПРОГРАММИРОВАНИЕ

FUNCTION BLOCK DIAGRAM / ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ

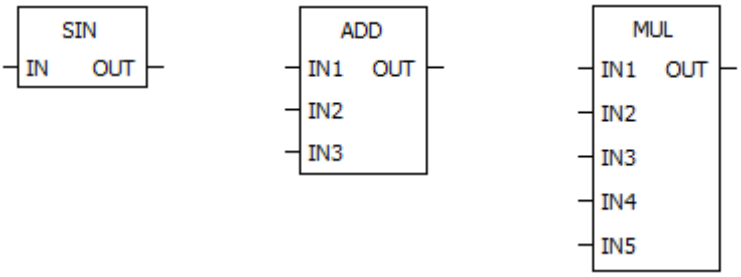
Функциональный блок

- Аналог класса
- При использовании создается индивидуальный экземпляр
  - экземпляр добавляется в таблицу локальных переменных
  - над блоком размещается имя экземпляра
- Имеет любое количество входов и выходов
- Локальные переменные блока сохраняют свои значения между его вызовами



Блок функции

- Аналог функции
- При использовании не создается индивидуальный экземпляр
- Имеет любое количество входов и один выход
- Локальные переменные блока не сохраняются свои значения между его вызовами



Контакты блока классифицируются:

- по направлению: вход (слева), выход (справа)
- по типу данных

С входами и выходами блока связываются переменные или константы проекта или входы и выходы соседних блоков.

ПРОГРАММИРОВАНИЕ

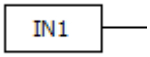
FUNCTION BLOCK DIAGRAM / ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ

Переменные

В нотации FBD переменные бывают следующих классов:

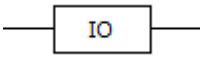
- Вход
- Вход/Выход
- Выход

Переменная класса «Вход»:



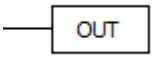
- только передает свое значение
- соединяется:
  - со входом функции или функционального блока
  - с переменной класса «Вход/Выход» или «Выход»

Переменная класса «Вход/Выход»:

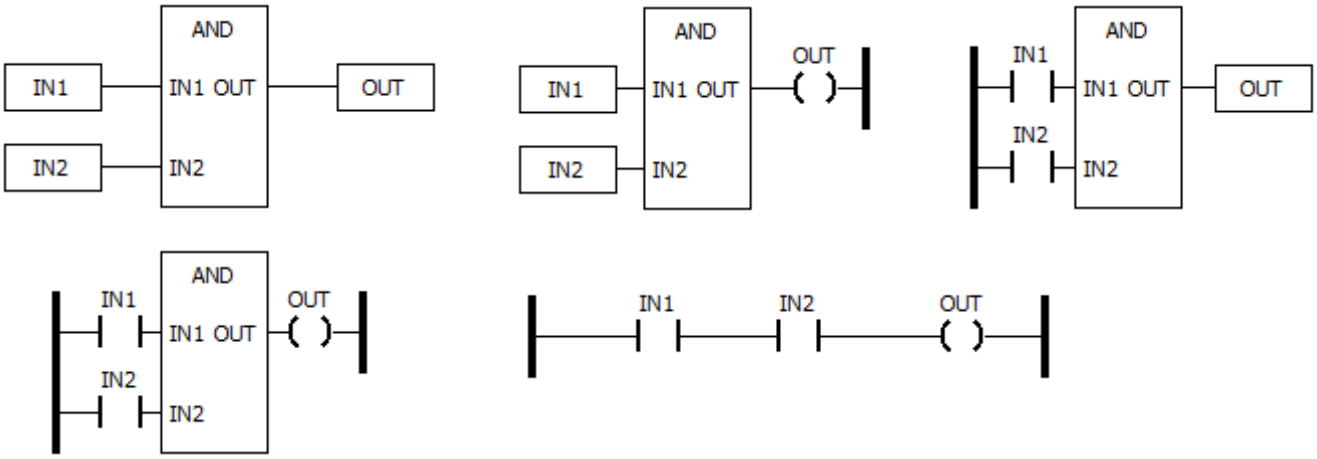


- принимает значение и передает далее по схеме
- соединяется:
  - со входом функции или функционального блока
  - с выходом функции или функционального блока
  - с переменной класса «Вход», «Вход/Выход» или «Выход»

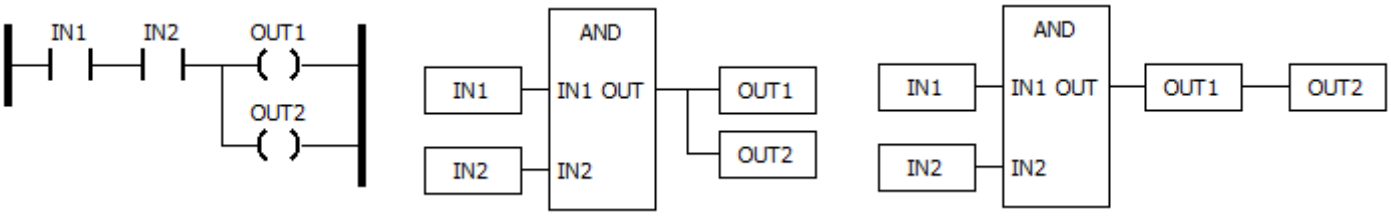
Переменная класса «Выход»:



- только принимает значение
- соединяется:
  - с выходом функции или функционального блока
  - с переменной класса «Вход» или «Вход/Выход»



пример реализации выражения OUT:= IN1 AND IN2 различными способами



пример реализации выражения OUT1:= OUT2:= IN1 AND IN2; различными способами

ПРОГРАММИРОВАНИЕ

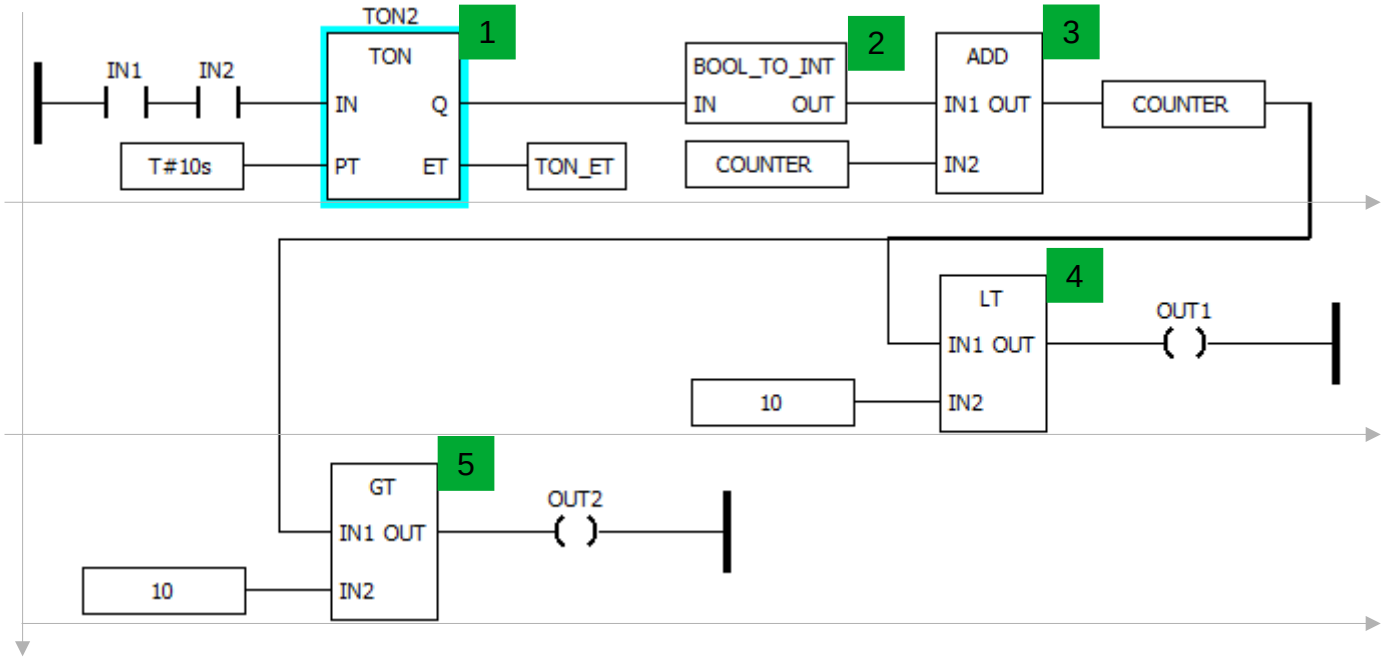
FUNCTION BLOCK DIAGRAM / ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ

Порядок исполнения

- Слева направо
- Сверху вниз

Порядок следования блоков можно изменить (в свойствах блока).

Некоторые редакторы языков FBD/LD позволяют включить/выключить визуальное отображение порядка исполнения блоков (в виде номеров над блоками).



ПРОГРАММИРОВАНИЕ

FUNCTION BLOCK DIAGRAM / ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ

Дополнительные параметры

Все функциональные блоки и блоки функций могут быть вызваны с дополнительными (необязательными) формальными параметрами

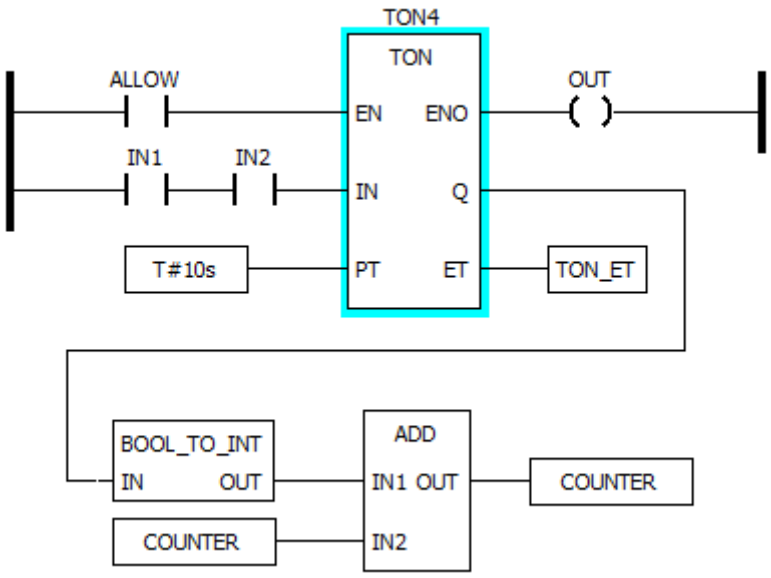
- EN
  - входной контакт
  - тип данных: BOOL
  - значение «по-умолчанию»: TRUE
- ENO
  - выходной контакт
  - тип данных: BOOL

Включение/выключение отображения дополнительных контактов настраивается в свойствах блока.

Если на вход «EN» блока подано значение «TRUE», то алгоритм, определяемый блоком, будет выполнен. После выполнения алгоритма без ошибок на выход «ENO» автоматически выдается значение «TRUE». Остальные выходы блока принимают значение в соответствие с алгоритмом. Если во время выполнения алгоритма блока возникает ошибка, то на выход «ENO» выдается «FALSE».

Если на вход «EN» блока подано значение «FALSE», то алгоритм, определяемый блоком, не будет выполнен. В этом случае на выход «ENO» автоматически выдается значение «FALSE». Остальные выходы блока принимают значения «по-умолчанию», которые были учтены при разработке самого блока.

Поведение блока будет одинаково как в случае его вызова с «EN = TRUE», так и при его вызове без параметров «EN/ENO» (если отключены в свойствах блока).



таймер TON4 запустится только, когда
ALLOW = TRUE, IN1 = TRUE, IN2 = TRUE

ПРОГРАММИРОВАНИЕ

FUNCTION BLOCK DIAGRAM / ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ

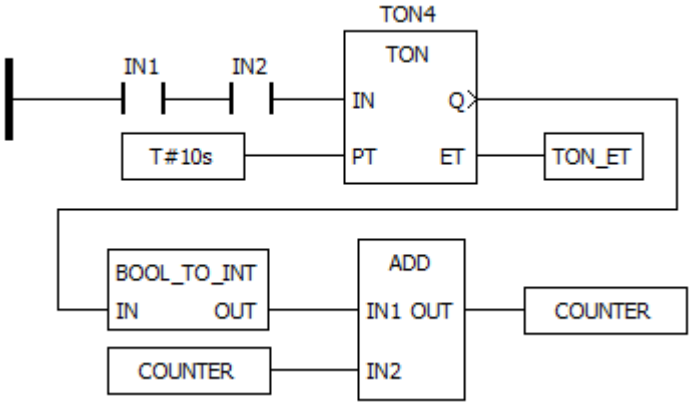
Модификатор контакта

Некоторые редакторы языка FBD позволяют для входных и выходных контактов блока назначить модификатор:

- Инверсия (○)
- Детектор нарастающего фронта (< или /)
- Детектор спадающего фронта (> или \ )

Модификатор назначается только для контактов типа «BOOL».

Функции модификаторов полностью соответствуют аналогичным функциям контактов и катушек языка LD.



По завершении счета таймер TON4 формирует выход Q = TRUE и
удерживает это состояние до тех пор, пока не сбросится вход IN (= FALSE).
Пока Q = TRUE, то с каждым циклом программы инкрементируется счетчик COUNTER.
Установив модификатор «Спадающий фронт» на выход Q, счетчик COUNTER будет
инкрементироваться каждый раз при сборсе таймера.

Комментарий

В языке FBD доступны многострочные комментарии, которые редактируются и отображаются в коде как и в редакторе языка LD.

Переход

Для передачи значения из одного участка схемы на другой без прямого соединения выхода и входа в языке FBD используются элементы «CONNECTION» (соединения), такие же как и в языке LD.

ПРОГРАММИРОВАНИЕ

SEQUENTIAL FUNCTION CHART / ФУНКЦИОНАЛЬНЫЕ ДИАГРАММЫ

Общие сведения

- ***SFC***
- ***Графический язык***
- Основные принципы — ***диаграмма состояний процесса***
- Основные элементы:
  - ***шаги***
  - ***переходы***
  - ***действия***
  - ***прыжки***
  - ***связи типа «Дивергения» и «Конвергенция»***

Программа представляется в виде потоковой диаграммы связанных между собой шагов и действий.

Нотация близка специалистам-технологам.

В каждом цикле работы управляющей программы выполняются Действия активного на данный момент Шага.

В конце каждого шага проверяется условие Перехода к следующему Шагу. Если условие перехода истанное, то следующий Шаг становится активным, но связанное с ним действие будет выполнено уже на следующем цикле работы управляющей программы.

Язык позволяет:

- включать в диаграмму переменные
- включать в диаграмму функциональные блоки и блоки функций
- описывать блоки действий
- выполнять последовательное и параллельное соединение переменных и блоков
- применять к контактам блоков различные модификаторы
- записывать комментарии
- использовать переходы
- управлять работой блоков по входам «EN/ENO»

состояния, в которых выполняются определенные действия, одновременно могут быть активны несколько состояний, одно из состояний является начальным;

переходы из состояния в состояние, для каждого перехода задаются логическое условие перехода к следующему шагу

альтернативное ветвление алгоритма, когда из текущего состояния возможны переходы к нескольким состояниям, при этом каждому переходу соответствует своё логическое условие и при выполнении алгоритма производится только один из альтернативных переходов

параллельное ветвление, в отличие от альтернативного имеет общее условие перехода на несколько параллельно работающих веток

переход к заданному состоянию

остановка процесса

ПРОГРАММИРОВАНИЕ

SEQUENTIAL FUNCTION CHART / ФУНКЦИОНАЛЬНЫЕ ДИАГРАММЫ

Общие сведения

- ***SFC***
- ***Графический язык***
- Основные принципы — ***диаграмма состояний процесса***
- Основные элементы:
  - ***шаги***
  - ***переходы***
  - ***действия***
  - ***прыжки***
  - ***связи типа «Дивергения» и «Конвергенция»***

Программа представляется в виде потоковой диаграммы связанных между собой шагов и действий.

Нотация близка специалистам-технологам.

В каждом цикле работы управляющей программы выполняются Действия активного на данный момент Шага.

В конце каждого шага проверяется условие Перехода к следующему Шагу. Если условие перехода истанное, то следующий Шаг становится активным, но связанное с ним действие будет выполнено уже на следующем цикле работы управляющей программы.

Язык позволяет:

- включать в диаграмму переменные
- включать в диаграмму функциональные блоки и блоки функций
- описывать блоки действий
- выполнять последовательное и параллельное соединение переменных и блоков
- применять к контактам блоков различные модификаторы
- записывать комментарии
- использовать переходы
- управлять работой блоков по входам «EN/ENO»

состояния, в которых выполняются определенные действия, одновременно могут быть активны несколько состояний, одно из состояний является начальным;

переходы из состояния в состояние, для каждого перехода задаются логическое условие перехода к следующему шагу

альтернативное ветвление алгоритма, когда из текущего состояния возможны переходы к нескольким состояниям, при этом каждому переходу соответствует своё логическое условие и при выполнении алгоритма производится только один из альтернативных переходов

параллельное ветвление, в отличие от альтернативного имеет общее условие перехода на несколько параллельно работающих веток

переход к заданному состоянию

остановка процесса