



**COMPLEX EVENTS**  
**Руководство пользователя**

**Версия 1.6**

**Москва**  
**2022 г.**

# ОГЛАВЛЕНИЕ

1. ИСТОРИЯ ИЗМЕНЕНИЙ.....	3
2. БЫСТРЫЙ СТАРТ.....	4
3. ВНЕШНИЙ ВИД.....	7
4. СОЗДАНИЕ И РЕДАКТИРОВАНИЕ ПРОГРАММЫ.....	8
4.1 Общие сведения.....	8
4.2 Описание элементов блок-схемы.....	8
4.2.1 Блоки «Начало» и «Конец».....	8
4.2.2 Блок «Действие».....	9
4.2.3 Блок «Условие».....	9
4.3 Создание блок-схемы.....	10
4.4 Описание элементов схемы функциональных блоков.....	11
4.4.1 Константы и переменные.....	11
4.4.2 Функциональные блоки.....	12
4.5 Создание схемы функциональных блоков.....	12
4.6 Элементы описания схемы.....	14
4.6.1 Название и Описание блока.....	14
4.6.2 Текст.....	14
4.6.3 Прямоугольник.....	14
4.7 Отмена и повторение действий (Undo/Redo).....	15
4.8 Автоматическая нумерация функциональных блоков.....	15
4.9 Поиск переменных.....	15
4.10 Работа с файлами.....	16
5. СБОРКА ПРОГРАММЫ.....	17
6. ОТЛАДКА ПРОГРАММЫ.....	18
6.1 Запуск отладки.....	18
6.2 Подключение отладчика к работающей программе.....	18
6.3 Работа в режиме отладки.....	18
6.3.1 Строка состояния программы.....	19
6.3.2 Управление исполнением программы.....	19
6.3.3 Просмотр значений данных схемы.....	19
6.3.4 Время исполнения цикла программы.....	19
6.4 Загрузка и чтение программы без отладки.....	20
7. НАСТРОЙКИ.....	21
7.1 Основные настройки.....	21
7.2 Настройки отладки.....	21
7.3 Цветовая схема.....	21
8. ПРИЛОЖЕНИЯ.....	22
8.1 Сочетания клавиш.....	22
8.2 Список кодов событий Complex Events.....	22
8.3 Библиотека функциональных блоков.....	23
8.3.1 Общие инструкции.....	26
8.3.2 Математические инструкции.....	28
8.3.3 Логические инструкции.....	31
8.3.4 Побитовые инструкции.....	32
8.3.5 Инструкции сравнения.....	35
8.3.6 Инструкции выбора и ограничения.....	36
8.3.7 Триггеры, генераторы, счётчики.....	39
8.3.8 Специальные функции.....	44

8.3.9	Функции доступа к периферийным устройствам .....	52
8.3.10	Функции доступа к цифровым портам .....	56

# 1. ИСТОРИЯ ИЗМЕНЕНИЙ

Версия 1.0 от 14.10.2021:

- Первая версия документа.

Версия 1.1 от 16.11.2021:

- Добавлен раздел «Описание блоков блок-схемы»;
- Добавлен блок «FROM\_FLOAT»;
- Добавлен блок «CALENDAR»;
- В разделе «Цветовая схема» добавлено описание новых функций.

Версия 1.2 от 23.11.2021:

- Добавлено описание режима drag'n'drop для перемещения блоков и функций на схему;
- Добавлен раздел с описанием операций Undo/Redo.

Версия 1.3 от 15.12.2021:

- Добавлен раздел «Элементы описания схемы»;
- В разделе «Настройки отладки» добавлено описание новой настройки;
- Добавлена функция «DELAY»;
- Добавлена функция «NTC\_CRASH\_FILE»;
- Дополнено описание функции «NTC\_ACCEL»;
- Дополнено описание функций «TO\_FLOAT» и «FROM\_FLOAT».

Версия 1.4 от 25.01.2022:

- Исправлено описание функции «TP»;
- Дополнено описание функций «EVENT», «SMS», «CALL», «CAM»;
- Добавлена функция «USER\_SMS»;
- Добавлена функция «RECV\_SMS»;
- В разделе «Список кодов событий Complex Events» удалены неиспользуемые события;
- В разделе «Библиотека функциональных блоков» переименованы блоки из группы «Блоки доступа к периферийным устройствам»;
- Добавлена функция «PWRSAVE».

Версия 1.5 от 07.04.2022:

- Изменено описание в разделе «Общие сведения»;
- Добавлен раздел «Описание элементов схемы функциональных блоков»;
- Обновлено описание функций «TO\_FLOAT», «FROM\_FLOAT», «FLEX», «USER\_PARAM», «USER\_SMS», «OUTPUT»;
- Добавлены функции «APERTURE», «RXD\_GET», «RXD\_CMP», «RXD\_STR2INT», «RXD\_STR2FLOAT», «RXD\_CHECKSUM», «TXD\_INIT», «TXD\_SET», «TXD\_GET», «TXD\_CHECKSUM», «TXD\_GET», «RS\_TRANS», «RS\_SEND», «RS\_RECV».
- Добавлен раздел «Функции доступа к цифровым портам».

Версия 1.6 от 21.06.2022:

- Дополнено описание в разделе "Создание схемы функциональных блоков";
- Добавлен раздел "Автоматическая нумерация функциональных блоков";
- Добавлен раздел "Поиск переменных";
- Обновлено описание функции "FLEX", "RXD\_GET", "TXD\_SET", "TXD\_GET";
- Добавлена функция "INFO", "IMEI", "ICCID", "IMSI", "LOG\_MSG", "MODBUS\_READ", "MODBUS\_WRITE";
- Обновлено стили информационных рамок.

## 2. БЫСТРЫЙ СТАРТ

Напишем простую программу, увеличивающую значение переменной на 1.

Для этого:

1. Подключите устройство с поддержкой Complex Events.
2. Запустите конфигуратор.
3. Создайте новую конфигурацию (нажмите на кнопку *Создание новой конфигурации*).

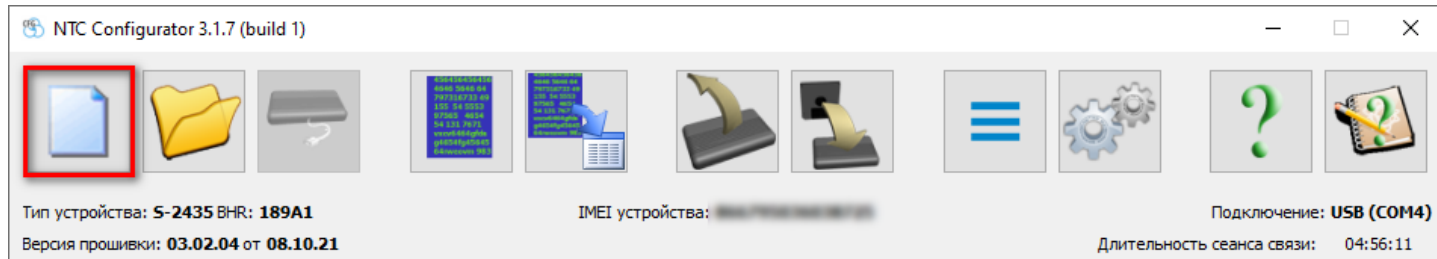


Рисунок 2.1 – Создание новой конфигурации

4. Перейдите на вкладку *Complex Events*.
5. Установите галочку – *Использовать Complex Events* и нажмите на кнопку *Открыть окно Complex Events*.

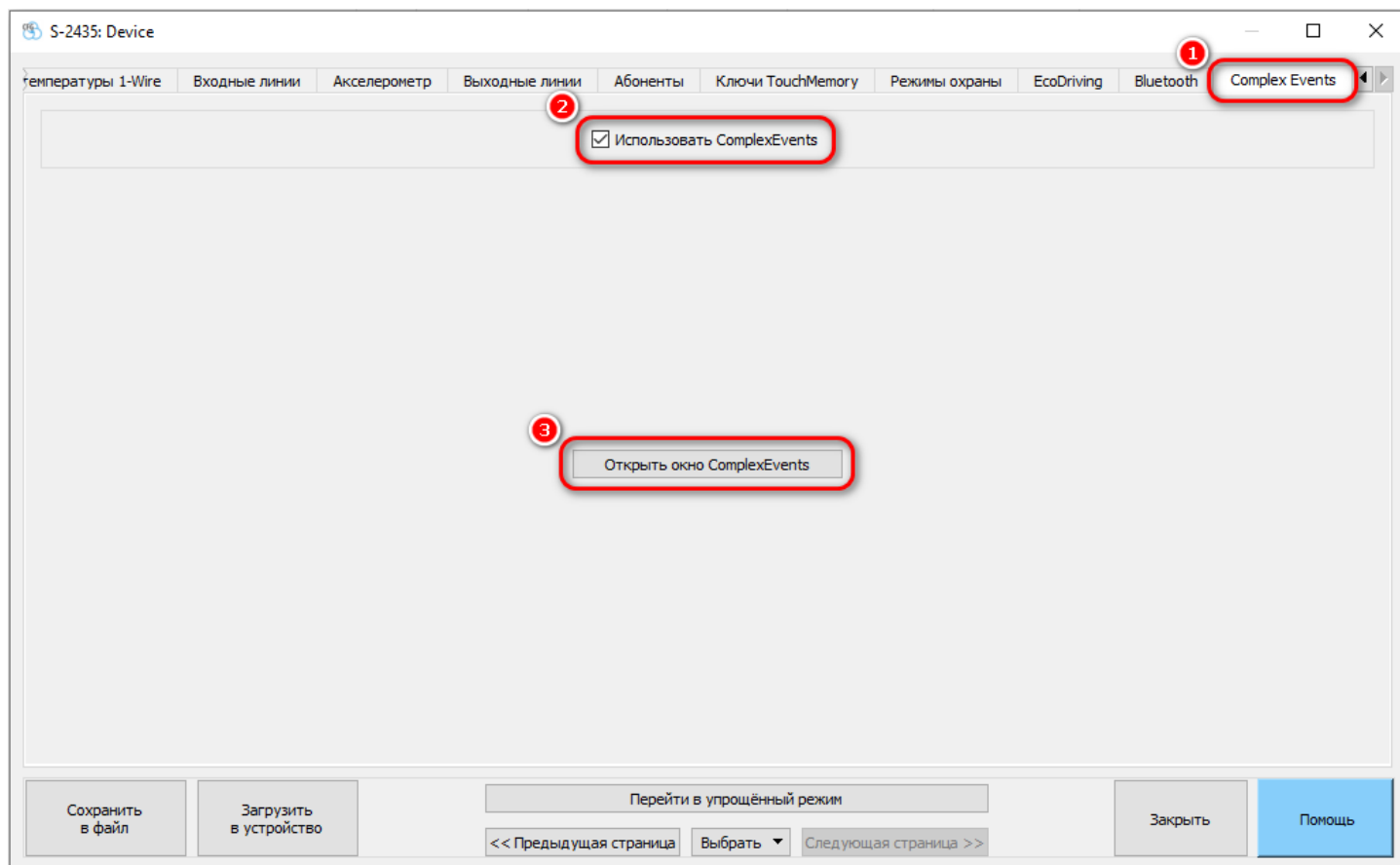



Рисунок 2.2 – Запуск редактора Complex Events

6. Далее в открывшемся окне выберите пункт меню *Файл - Новый* (или нажмите на кнопку  на панели инструментов). В редакторе блок-схемы, в левой части редактора появится простейшая блок-схема.
7. Нажмите на блок *Действие* в левой части редактора. В правой части редактора теперь будет отображаться содержимое выбранного блока *Действие*.

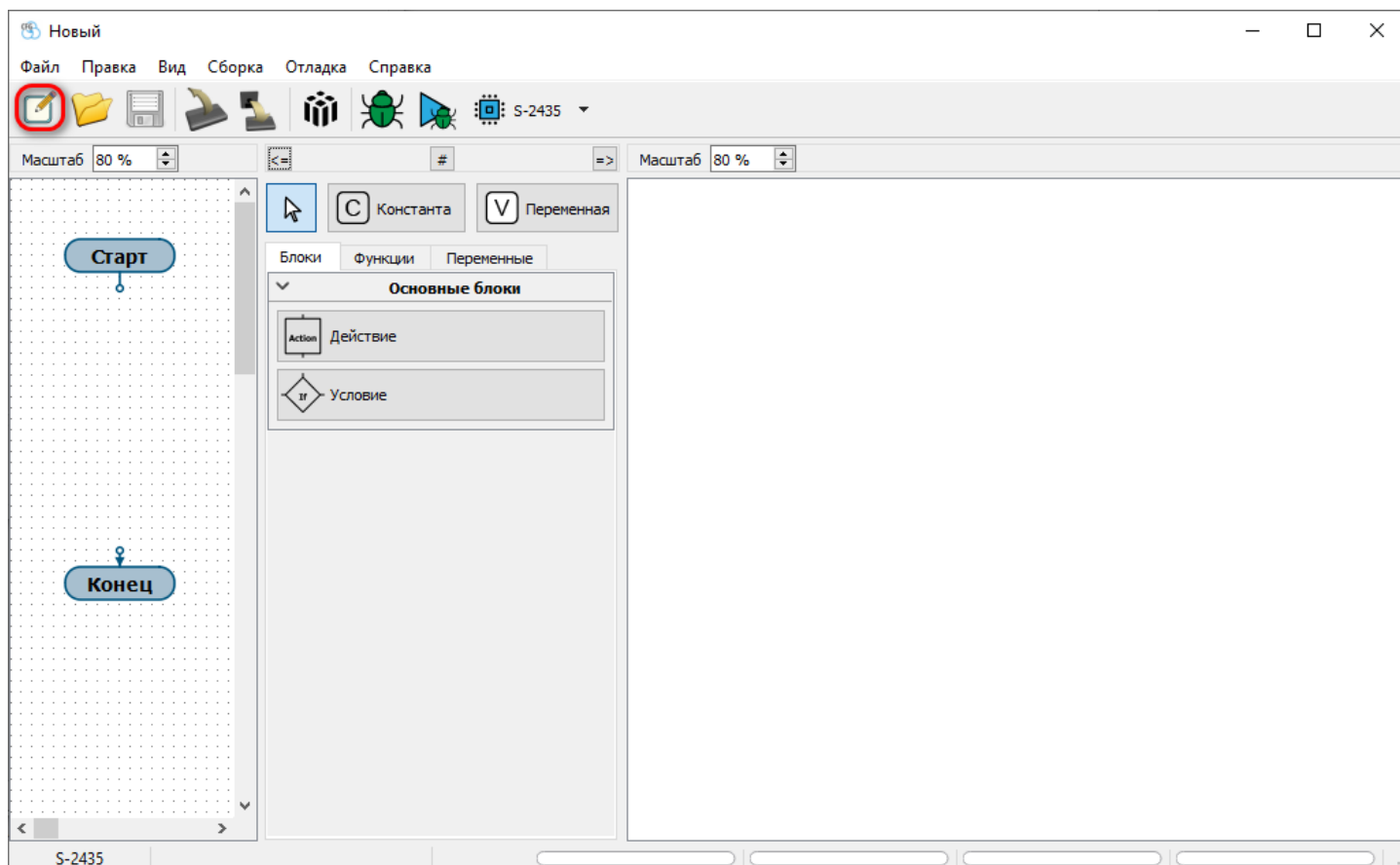


Рисунок 2.3 – Создание новой, простейшей блок-схемы

8. Выберите вкладку **Функции** на панели в средней части редактора.
9. В отобразившемся окне функций нажмите на кнопку **ADD** в группе **Математические операции**. Затем переместите указатель мыши на правую часть редактора и нажмите в любое место. В редакторе появится функция **Сложение**.
10. Нажмите на кнопку **Переменная**, переместите указатель мыши на правую часть редактора и нажмите на свободное место.
11. Нажмите на кнопку **Константа**, переместите указатель мыши на правую часть редактора и нажмите на свободное место.
12. Измените значение константы на 1. Для этого, выполните двойной щелчок по константе и в открывшемся диалоговом окне в поле **Значение** введите число 1.
13. Расположите добавленные элементы (рис 2.4) и соедините их. Чтобы соединить два вывода, нужно нажать левой клавишей мыши на первый вывод, затем нажать на второй вывод.

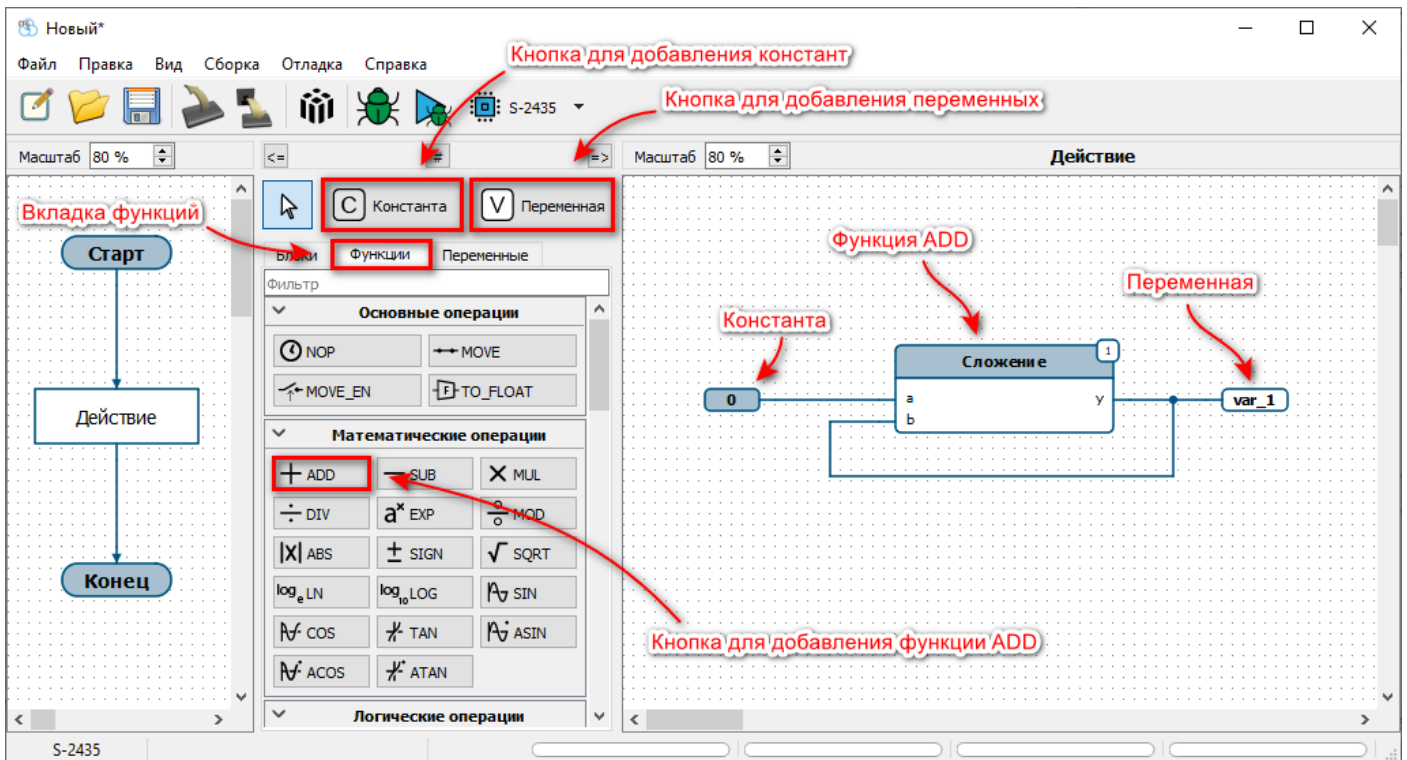







Рисунок 2.4 – Диаграмма инкрементации переменной var\_1 на 1

14. Выберите пункт меню **Сборка – Собрать** (или нажмите на кнопку  на панели инструментов). Если всё сделано правильно, программа соберётся без ошибок.
15. Выберите пункт меню **Отладка – Отладка** (или нажмите на кнопку  на панели инструментов). Если устройство подключено, появится окно с предложением загрузить конфигурацию в устройство, нажмите на кнопку **Да**. Ожидайте несколько секунд, пока устройство не перезагрузится, затем приложение автоматически загрузит программу и войдёт в режим отладки.
16. Нажмите несколько раз на кнопку **Шаг**  (пункт меню **Отладка - Шаг**) и убедитесь, что значение переменной увеличивается на 1. Текущее значение переменной отображается над её выводом.
17. Нажмите на кнопку **Продолжить**  (пункт меню **Отладка - Продолжить**) и убедитесь, что значение переменной увеличивается.
18. Нажмите на кнопку **Завершить отладку**  (пункт меню **Отладка – Завершить отладку**).

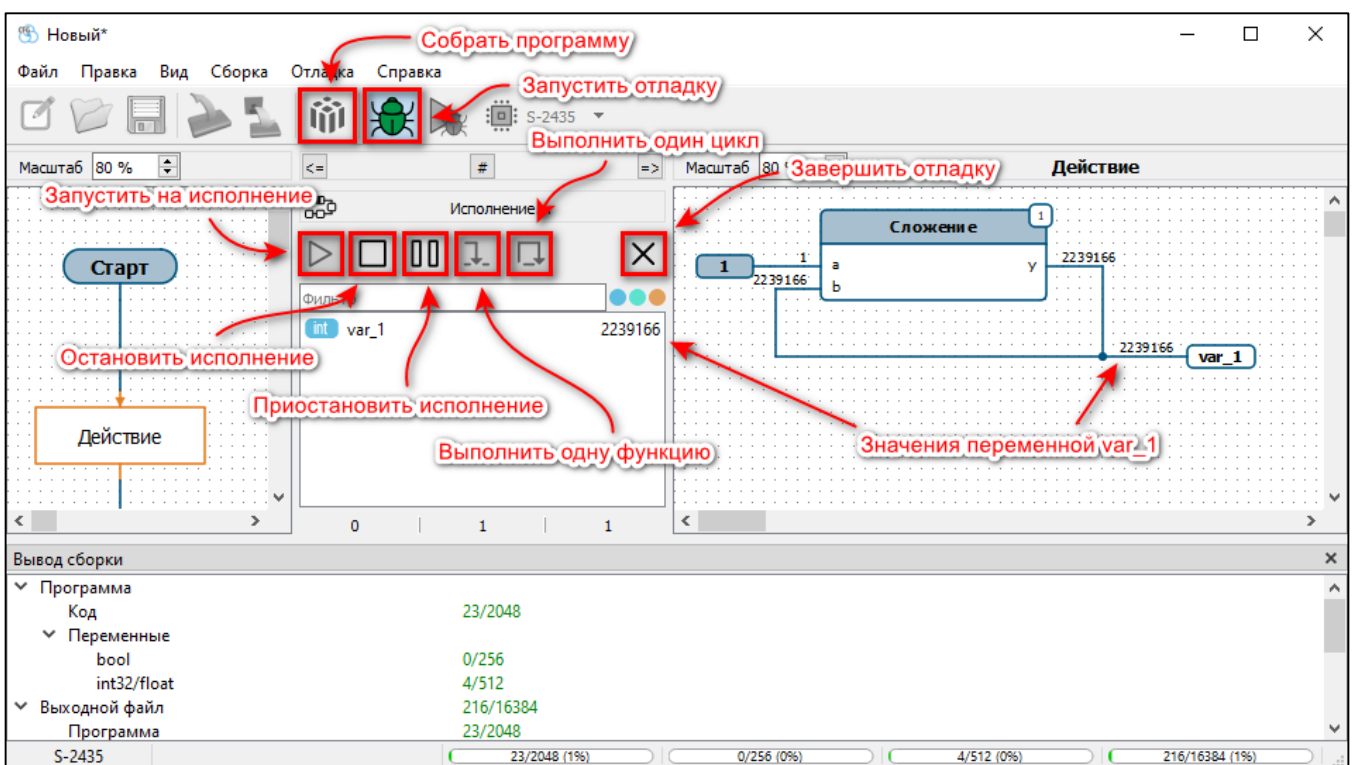


Рисунок 2.5 – Отладка программы

### 3. ВНЕШНИЙ ВИД

Приложение разделено на следующие области: *основное меню*, *панель инструментов*, *статусная панель*, *основная область*.

*Основное меню* содержит опции для манипуляции с приложением.

*Панель инструментов* дублирует наиболее часто используемые пункты меню, может быть скрыта (пункт меню **Вид – Панель инструментов**).

*Статусная панель* отображает информацию о подключённом устройстве и ресурсы необходимые для работы проекта.

*Основная область* приложения включает:

- Редактор блок-схемы проекта.
- Редактор схемы функциональных блоков (функций) для конкретного блока блок-схемы.
- Панель с элементами для создания схемы проекта (между редакторами).
- Дополнительные вкладки – ошибки, вывод сборки, точки останова (отображение управляется через меню **Вид**).

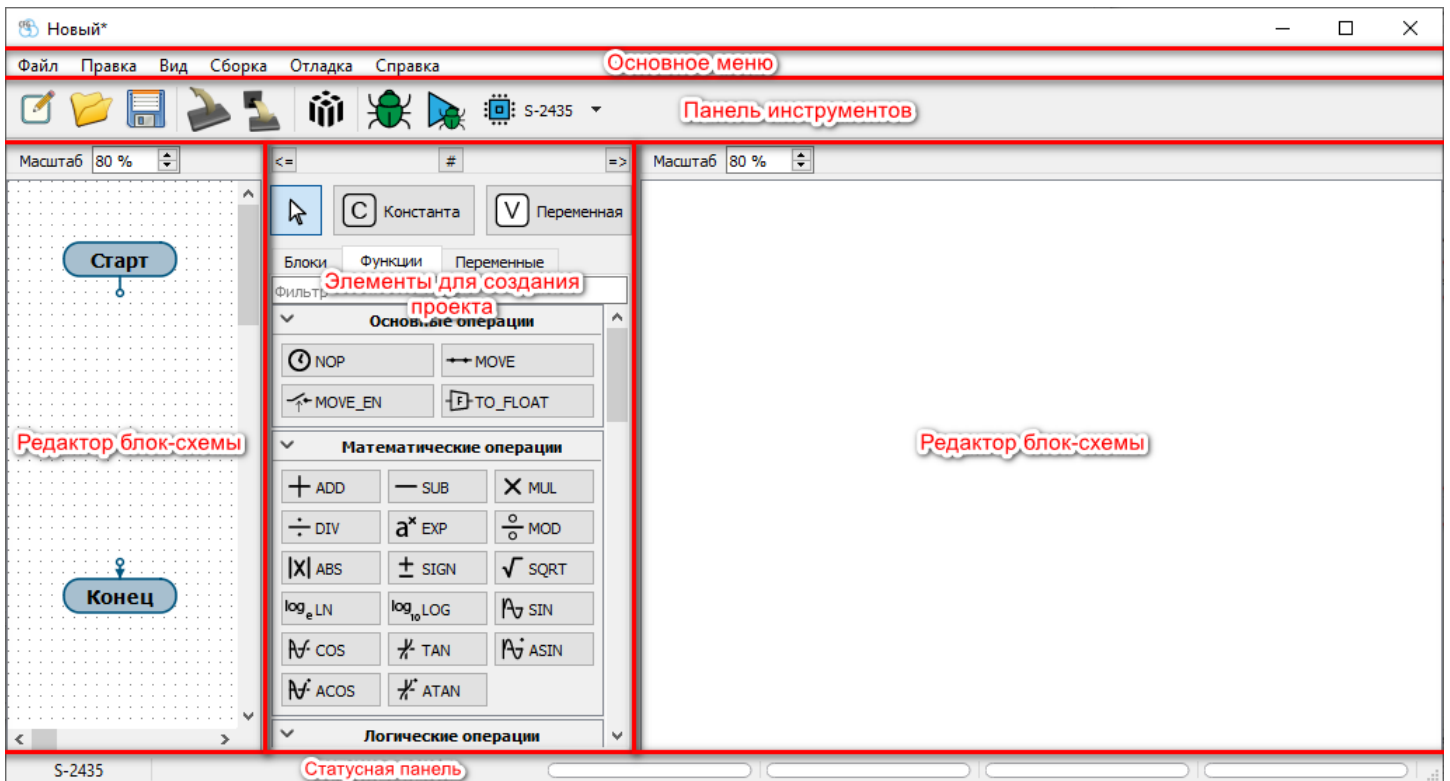


Рисунок 3.1 – Внешний вид редактора



## 4. СОЗДАНИЕ И РЕДАКТИРОВАНИЕ ПРОГРАММЫ

### 4.1 Общие сведения

Схема программы, загружаемой в устройство, составляется с помощью графических элементов.

Сначала, в редакторе в левой части приложения, составляется общая блок-схема. Блок-схема – это общий алгоритм работы программы, который состоит из блоков (*шагов*), соединённых между собой линиями, указывающими направление последовательности исполнения программы. Поддерживаются следующие блоки:

- *Старт* – обозначает начало программы, всегда присутствует на блок-схеме в единственном экземпляре.
- *Конец* – обозначает конец программы, всегда присутствует на блок-схеме в единственном экземпляре.
- *Действие* – блок обработки данных.
- *Условие* – блок обработки данных с условием, позволяет продолжить работу программы в одном из двух направлений. Данный блок позволяет изменять последовательность исполнения программы, для программирования условий и циклов.

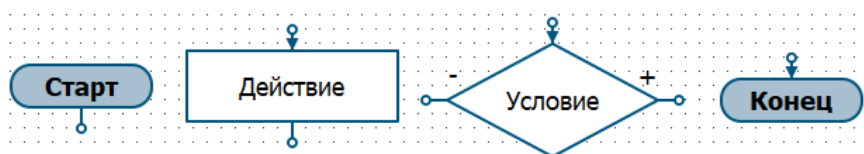


Рисунок 4.1 – Внешний вид блоков *Старт*, *Действие*, *Условие*, *Конец*

Начиная от блока *Старт*, блоки исполняются друг за другом в определённой пользователем последовательности (с помощью линий). Достижение программой блока *Конец* – означает конец обработки данного цикла. Циклы бесконечно выполняются друг за другом, от блока *Старт* до блока *Конец*.

В правой части редактора составляется схема обработки данных для конкретного блока (*Действия* или *Условия*) из левой части. Данная схема состоит из соединённых между собой функциональных блоков (функций), констант и переменных.

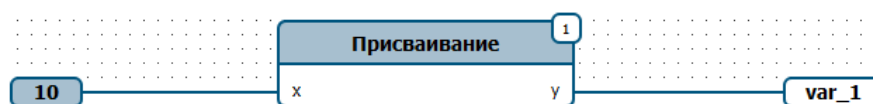


Рисунок 4.2 – Внешний вид элементов *Константа*, *Функция* и *Переменная*

Схема функциональных блоков по своей сути похожа на язык программирования CFC (Continuous Function Chart), который предназначен для программирования ПЛК (программируемых логических контроллеров).

### 4.2 Описание элементов блок-схемы

#### 4.2.1 Блоки «Начало» и «Конец»

Блоки обозначают начало и конец программы. Присутствуют на схеме в единственном экземпляре и не могут быть удалены.

## 4.2.2 Блок «Действие»

Блок используется для описания одной или нескольких функций. У блока есть один вход и один выход, которые позволяют разместить его на блок-схеме и показывают направление движения программы.

После выполнения последней функции блока *Действие* программа переходит к выполнению блока, который подключен к выходу.

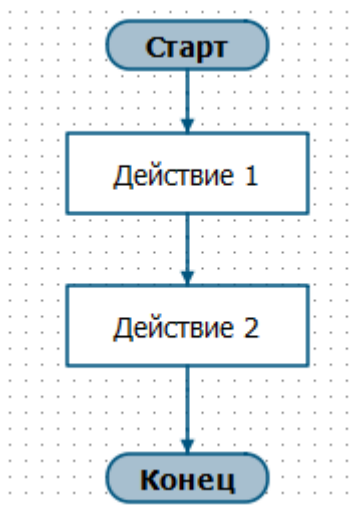


Рисунок 4.3 – Пример включения блока Действие в блок-схему

## 4.2.3 Блок «Условие»

Блок используется для описания ветвления программы в зависимости от заданных пользователем условий. Как и в блоке *Действие* внутри блока *Условие* производится описание одной или нескольких функций. У блока есть один вход и два выхода: «Выход +» и «Выход -», которые позволяют разместить его на блок-схеме и показывают направление движения программы.

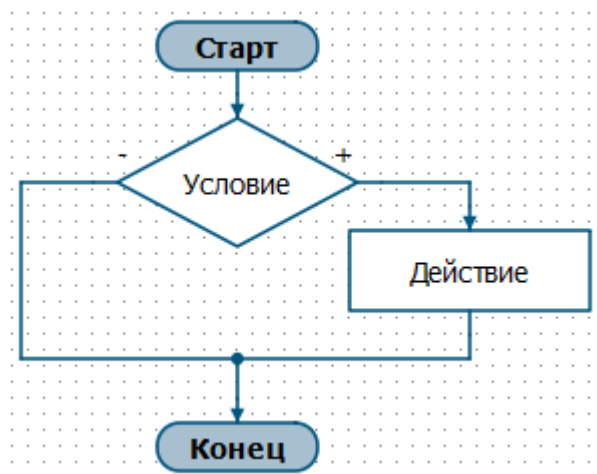


Рисунок 4.4 – Пример включения блока Действие в блок-схему

Отличительной особенностью блока является наличие системной переменной *result*, которая размещена внутри блока (в правой части редактора). Переменную *result* нельзя удалить или скопировать. Для работы блока в переменную *result* должно быть записано значение *True* или *False*.

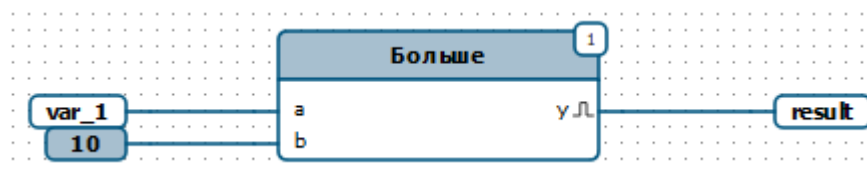


Рисунок 4.5 – Пример подключения переменной result внутри блока Условие

После выполнения последней функции блока *Условие* программа проверяет значение переменной *result* и если значение *True*, то программа переходит к выполнению блока, подключенного к «Выходу +», если значение *False* то программа переходит к выполнению блока, подключенного к «Выходу -».

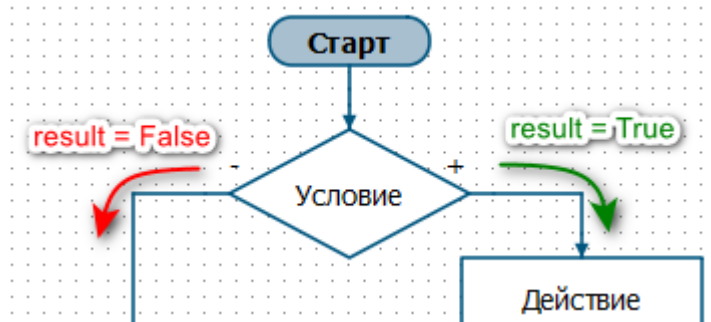


Рисунок 4.6 – Направление исполнения программы в зависимости от значения переменной *result*

### 4.3 Создание блок-схемы

Как было сказано выше, блоки *Старт* и *Конец* присутствуют в программе в единственном экземпляре, их нельзя ни удалить, ни добавить копию одного из них. Можно только менять их положение на блок-схеме.

Блоки *Действие* и *Условие* можно добавлять на блок-схему в необходимом количестве. Для этого перейдите на вкладку *Блоки* на панели в средней части приложения и воспользуйтесь одним из двух способов перемещения блоков на схему:

- Выбор первым нажатием, размещение вторым нажатием.  
Нажмите на нужный блок левой кнопкой мыши в группе *Основные блоки*, затем переместите мышь на редактор блок-схем (в левой части) и повторно нажмите левую кнопку мыши, выбранный блок добавится на схему.
- Перетаскивание (drag'n'drop).  
Наведите указатель мыши на нужный блок в группе *Основные блоки*, затем «захватите» его (зажмите левую кнопку мыши) и переместите мышь на редактор блок-схем (в левой части). «Отпустите» блок (отпустите левую кнопку мыши), выбранный блок добавится на схему.

Чтобы отменить добавление блока нажмите клавишу **Esc** или нажмите на кнопку *Стрелка*. Блокам можно назначить название и описание. Для этого нужно сделать двойной щелчок по блоку или нажать правой клавишей мыши и выбрать пункт меню *Свойства* (дублирует пункт основного меню *Правка - Свойства*). В появившемся диалоговом окне ввести соответствующие параметры. Описание блока отображается при наведении на него мыши.

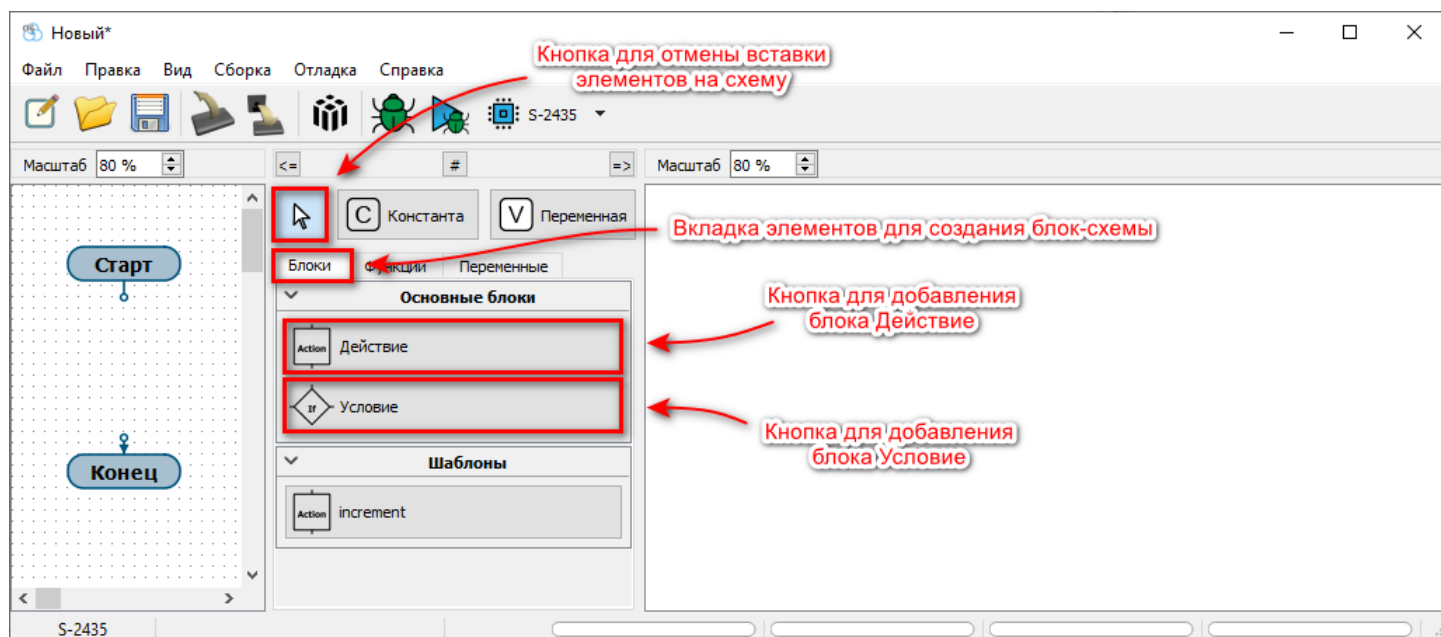


Рисунок 4.7 – Элементы для создания блок-схемы

Очерёдность исполнения блоков определяет пользователь с помощью соединительных линий. Соединённые выходы образуют цепь. Чтобы соединить два вывода, нужно сделать щелчок мышью по первому выводу, при этом появится линия, затем сделать клик по второму выводу. Направление линий можно изменять, делая клики в нужных местах в процессе создания (после клика по первому выводу). Вывод можно подключить к уже существующей цепи, кликнув сначала по выводу, затем по цепи. Для отмены создания цепи нужно нажать клавишу **Esc**. Цепь может содержать несколько выходов и один вход. Случай, когда цепь содержит более одного входа, приведёт к ошибке компиляции. Входы блоков обозначаются стрелочкой. Все выходы блоков блок-схемы должны быть подключены.

Каждый блок, добавленный пользователем, содержит свою схему функциональных блоков (в левой части приложения).

## 4.4 Описание элементов схемы функциональных блоков

### 4.4.1 Константы и переменные

Для того, чтобы пользователь мог задать нужное исходное состояние программы, а также для получения и обработки результатов работы программы, используются элементы *Константа* и *Переменная*.

*Константа* – постоянная величина, значение которой определяется на этапе составления программы и далее в ходе исполнения программы не меняется.

*Переменная* – именованная область памяти, которая используется для записи, чтения и хранения различных значений. Значение переменной определяется при составлении программы и впоследствии может постоянно меняться в ходе исполнения программы.

Характеристики констант и переменных:

Название	Применимость	Описание
Название	Переменная	Текстовое название, которое позволяет обратиться к значению каждой конкретной переменной (прочитать его или изменить). Максимальная длина названия – 16 символов.
Тип	Переменная Константа	<p>От типа зависит область допустимых значений и размер, выделяемый в памяти под константу или переменную.</p> <p><b>«Int32»</b> Целое число от -2147483648 до 2147483647. Занимает в памяти 4 байта.</p> <p><b>«Float»</b> Число с плавающей точкой. Диапазон значений без потери точности для чисел, состоящих не более, чем из 7 значащих цифр. Например, от -9999999 до 9999999 или от -0.999999 до 0.999999. Занимает в памяти 4 байта.</p> <p><b>«Bool»</b> Логический тип, имеющий два значения <i>True</i> или <i>False</i>. Занимает в памяти менее 1 байта.</p> <p><u>Преобразование типов в Complex Events:</u></p> <p>«INT32 к FLOAT» и обратно «FLOAT к INT32»: Переносится только целая часть и знак: int32 «-123» преобразуется во float «-123.0»; float «5.99» преобразуется в int32 «5».</p> <p>«FLOAT/INT32 к BOOL»: <i>True</i> – значения <b>не равные</b> «0» (или «0.0»); <i>False</i> – значения <b>равные</b> «0» (или «0.0»).</p> <p>«BOOL к FLOAT/INT32»: <i>True</i> преобразуется в «1» (или «1.0»); <i>False</i> преобразуется в «0» (или «0.0»).</p>

Значение (отображение)	Переменная Константа	Вид отображения значения числа с типом INT32 при отладке (при вычислениях отображение никак не влияет на результат). « <b>DEC</b> » - Число «26952» в десятичной системе исчисления '26952' « <b>HEX</b> »- Число «26952» в шестнадцатеричной системе исчисления '0x6948' « <b>BIN</b> » - Число «26952» в двоичной системе исчисления '0b0110100101001000' « <b>ASCII</b> » - Число «26952» как текст кодировке ASCII 'Hi'
Значение	Переменная Константа	Значение, которое примет константа или переменная при начале работы программы.
Доступ на запись (во время отладки)	Переменная	Если флаг установлен, то во время работы отладчика пользователь может без остановки программы изменить значение переменной вручную.

#### 4.4.2 Функциональные блоки

*Функциональный блок (функция)* – это блок, который имеет определённое число входов и определённое число выходов. На входы функции поступают данные (например, от других блоков), затем эти данные обрабатываются и формируются данные, которые поступают на выходы данной функции (эти выходы могут быть подключены ко входам других функциональных блоков). Функции исполняются друг за другом. Очередность определяется порядковым номером.

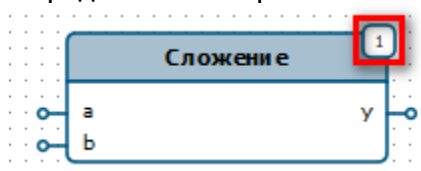


Рисунок 4.8 – Порядковый номер для исполнения функции

Подробное описание каждой функции приведено в разделе «[Библиотека функциональных блоков](#)».

### 4.5 Создание схемы функциональных блоков

При выборе блока на блок-схеме (в левой части), в правой части приложения появляется его функциональная схема. Функциональная схема может содержать функциональные блоки (функции), переменные и константы, соединённые между собой. Линии на схеме функциональных блоков указывают направление передачи данных.

Чтобы добавить функцию на схему, перейдите на вкладку *Функции* на панели в средней части приложения и воспользуйтесь одним из двух способов перемещения блоков на схему:

- Выбор первым нажатием, размещение вторым нажатием.  
Нажмите на нужную функцию левой кнопкой мыши, затем переместите мышь на редактор схемы функциональных блоков (в правой части) и повторно нажмите левую кнопку мыши, выбранная функция добавится на схему.
- Перетаскивание (drag'n'drop).  
Наведите указатель мыши на нужную функцию, затем «захватите» ее (зажмите левую кнопку мыши) и переместите мышь на редактор схемы функциональных блоков (в правой части). «Отпустите» функцию (отпустите левую кнопку мыши), выбранная функция добавится на схему.

Чтобы отменить добавление функции нажмите клавишу **Esc**, или на кнопку **Стрелка**. Константы и переменные добавляются аналогично функциям, для это есть кнопки **Константа** и **Переменная**. При использовании кнопки **Переменная** на схему будет всегда добавляться новая переменная. Чтобы добавить на схему созданную ранее переменную, перейдите на вкладку **Переменные** и выберите нужную из списка. На этой вкладке отображаются все добавленные пользователем переменные. Одну и ту же переменную можно добавлять на разные схемы функциональных блоков.

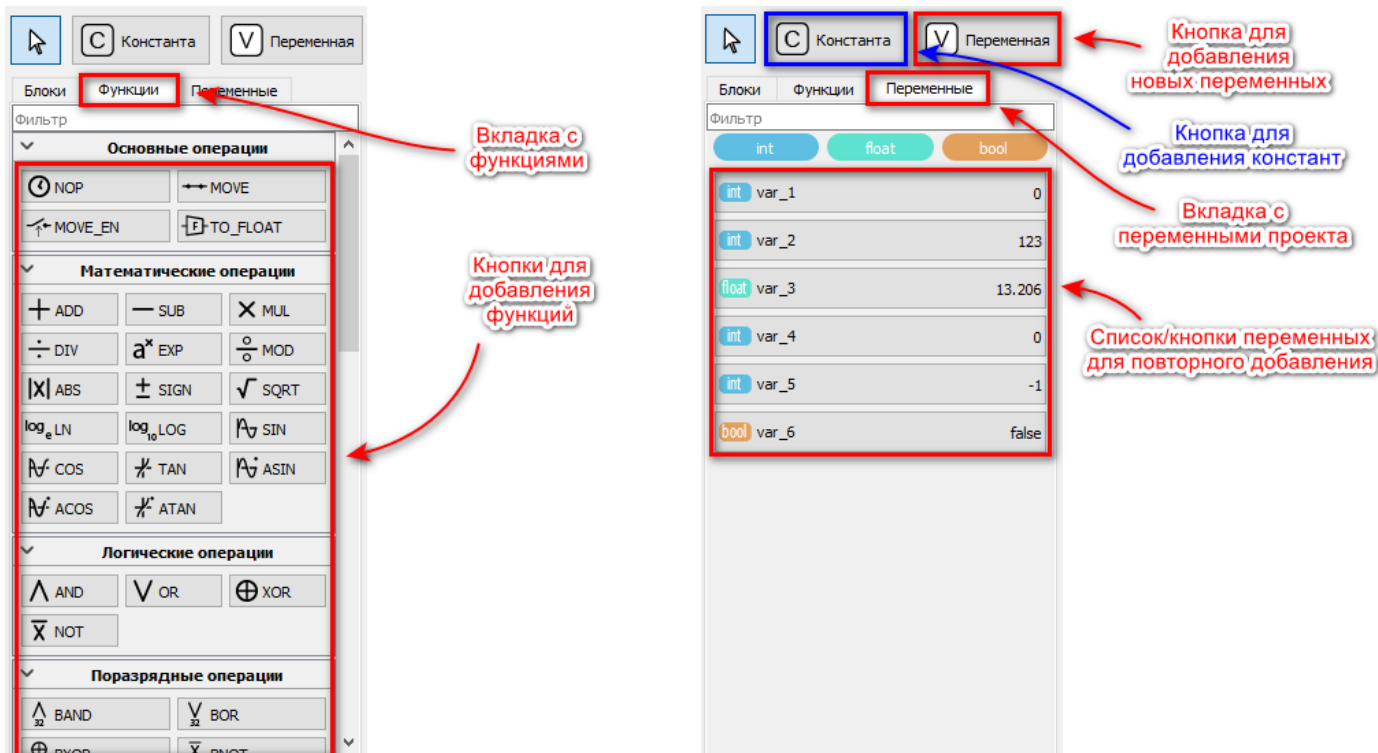
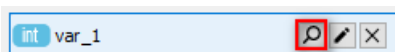


Рисунок 4.9 – Элементы для создания схем функциональных блоков

Чтобы быстро найти переменную на схеме, нужно нажать кнопку **Поиск** для нужной переменной. Подробно об интерфейсе поиска можно прочитать в разделе «[Поиск переменных](#)».



Входы функций всегда располагаются слева, а выходы всегда справа. Соединение выводов элементов функциональной схемы осуществляется, как и на блок-схеме. Соединённые выходы образуют цепь. В цепи может быть только один выход функции, только одна переменная или константа. Если в цепи присутствует выход, то константа к данной цепи не должна быть подключена. Подключать все выходы функций к цепям не обязательно.

Функции исполняются последовательно друг за другом. В правом верхнем углу отображается порядковый номер функции, который определяет очерёдность исполнения. Чем меньше порядковый номер, тем раньше исполняется функция. Изменить порядковый номер можно выполнив двойной клик по функции или нажать правой клавишей мыши и выбрать пункт меню **Свойства**. Затем в появившемся диалоговом окне изменить значение параметра **Очерёдность исполнения**. В данном диалоговом окне можно изменять другие параметры функций, если они для неё предусмотрены.

В редакторе предусмотрен механизм автоматической нумерации функций, подробное описание приведено в разделе «[Автоматическая нумерация функциональных блоков](#)».



## 4.6 Элементы описания схемы

Для улучшения информативности схемы в редакторе предусмотрены следующие механизмы:

- Добавление/изменение *Названия* и *Описания* блока в левой части схемы
- Добавление элементов *Текст* и *Прямоугольник* на левую или правую часть схемы

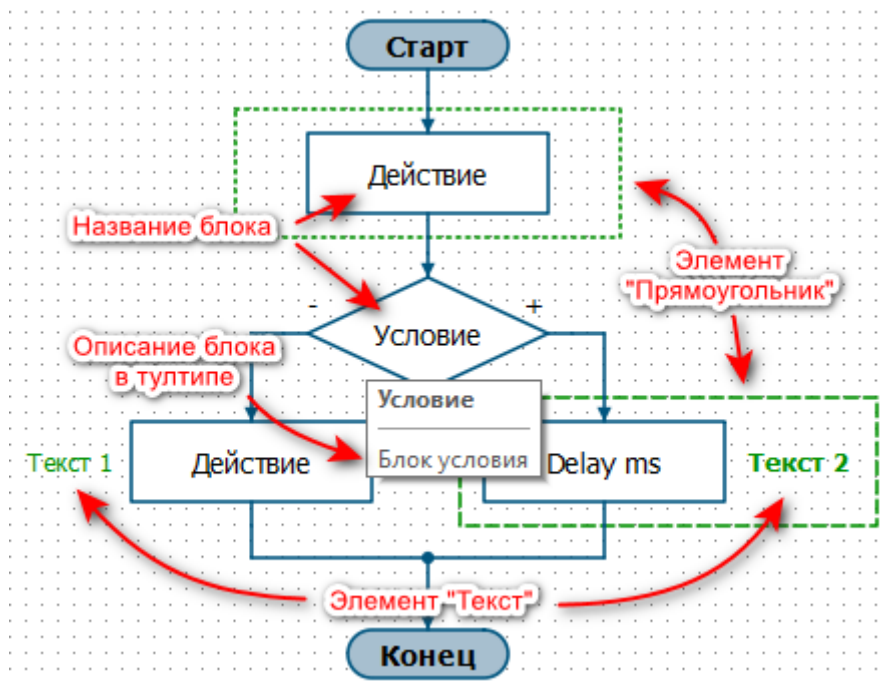


Рисунок 4.10 – Кнопки управления операциями отмены и повторения

### 4.6.1 Название и Описание блока

У каждого блока левой части схемы есть *Название* и *Описание*. *Название* отображается на схеме внутри блока и в заголовке тултипа, который появляется при наведении указателя мыши на блок. *Описание* блока отображается только в тултипе. Для изменения *Названия* или *Описания* нужно нажать ПКМ на блок и в контекстном меню выбрать пункт *Свойства*.

### 4.6.2 Текст

Для размещения надписей на схеме можно использовать элемент *Текст*. Элемент можно разместить как в левой, так и в правой части. Для размещения необходимо выбрать пункт меню *Поместить - Текст*. Для надписей можно настроить следующие параметры:

- Размер шрифта
- Стилль написания (жирный, курсив, подчеркнутый)
- Вертикальное выравнивание (по левому краю, по центру, по правому краю)

Цвет текста определяется глобальной настройкой цветовой схемы (см. [раздел «Цветовая схема»](#)).

### 4.6.3 Прямоугольник

Для размещения рамок на схеме можно использовать элемент *Прямоугольник*. Элемент можно разместить как в левой, так и в правой части. Для размещения необходимо выбрать пункт меню *Поместить - Прямоугольник*. Для рамок можно настроить только тип линии.

Цвет рамок и их толщина определяются глобальной настройкой цветовой схемы (см. [раздел «Цветовая схема»](#)).

## 4.7 Отмена и повторение действий (Undo/Redo)

В редакторе доступны операции отмены и повторения действий.



Рисунок 4.11– Кнопки управления операциями отмены и повторения

В памяти хранятся последние 100 действий пользователя. Программа контролирует все основные манипуляции пользователя: создание/удаление/перемещение блоков, линий, переменных и констант, изменение названий переменных, изменение значений переменных и констант, изменение свойств блоков и функций и т.п. Это значительно облегчает работу в редакторе.

## 4.8 Автоматическая нумерация функциональных блоков

В редакторе предусмотрен механизм автоматической нумерации функциональных блоков (пункт меню **Правка – Пронумеровать функциональные блоки**). Эта функция позволяет быстро пронумеровать блоки в зависимости от их расположения на схеме.

Доступно два алгоритма нумерации:

**Вниз затем вправо** – Нумерация по столбцам сверху вниз, слева направо.

**Вправо затем вниз** – Нумерация по строкам слева направо, сверху вниз.


Механизм нумерации ориентируется только на визуальное расположение элементов схемы и не корректирует свою работу в зависимости от порядка подключения элементов или их функционала.



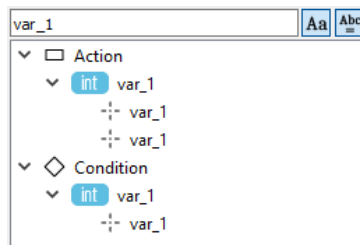
*Механизм автоматической нумерации предназначен для быстрой «черновой» нумерации. После нее рекомендуется проверить результат и внести корректировки вручную.*

## 4.9 Поиск переменных

Для удобства работы с переменными можно воспользоваться интерфейсом поиска. Интерфейс можно вызвать двумя способами:


- Из меню **Вид – Поиск переменных**
- В списке используемых переменных на вкладке **Переменные** нужно нажать кнопку **Поиск** 



В редакторе откроется область со списком точек использования переменных. При двойном нажатии на любое из упоминаний редактор центрирует область просмотра в нужном месте схемы. Если вызвать интерфейс через кнопку **Поиск**, то при открытии интерфейса в строке поиска будет указано название выбранной переменной.





## 4.10 Работа с файлами

При запуске редактора Complex Events, создаётся пустой проект, на блок-схеме присутствуют только блоки *Старт* и *Конец*. Пункт меню **Файл – Новый** (кнопка  на панели инструментов) создаёт новый проект с простейшей блок-схемой.

Созданный проект можно сохранить в файл (пункты меню **Файл – Сохранить**, **Файл – Сохранить как** или кнопка  на панели инструментов), открыть из файла (пункт меню **Файл – Открыть** или кнопка  на панели инструментов). Для быстрого доступа к недавним проектам есть список недавно сохранённых и открытых файлов, он доступен через пункт меню **Файл – Недавние файлы**.

Схемы блоков можно сохранять и открывать в/из файла. Для этого нужно выбрать нужный блок на блок-схеме и воспользоваться пунктами меню **Правка – Импортировать**, **Правка – Экспортировать** (или нажать правой клавишей мыши и выбрать аналогичные пункты меню из списка). Для быстрого доступа блоки можно сохранять в шаблоны (пункт меню **Правка – Отправить в шаблоны**). Сохранённые шаблоны доступны в группе *Шаблоны* на вкладке *Блоки* на панели в средней части приложения.

## 5. СБОРКА ПРОГРАММЫ

Сборка программы вызывается через пункт меню *Сборка – Собрать* (или через кнопку  на панели инструментов). Сборка включает в себя:

- компиляцию проекта (можно вызвать отдельно, пункт меню *Сборка - Скомпилировать*);
- построение выходного файла программы, для загрузки в прибор;
- проверку конфигурации устройства;
- вывод ошибок и предупреждений;
- отображение ресурсов, которые нужны для построения программы.

При компиляции происходит построение программы и выделение необходимых ресурсов. Если программа содержит ошибки, или в устройстве не хватает необходимых ресурсов для построения, то соответствующие сообщения добавляются на вкладку *Проблемы* (открывается автоматически).

При построении выходного файла формируется файл, загружаемый в устройство, в него входит программа, исполняемая интерпретатором Complex Events и исходный файл проекта. Если размер полученного файла превышает допустимый, то на вкладку *Проблемы* добавляются соответствующие сообщения.

Для корректной работы программы в конфигурации устройства должна быть включена поддержка Complex Events, и если программа использует функции, работающие с периферией устройства, то эта периферия должна быть соответствующим образом настроена. Если в конфигурации присутствуют некорректные настройки, соответствующие сообщения добавляются на вкладку *Проблемы*.

На вкладке *Проблемы* отображаются сообщения об ошибках и предупреждениях. Доступ к вкладке осуществляется через пункт меню *Вид - Проблемы*. При двойном нажатии левой клавишей мыши на сообщении, приложение показывает проблемный элемент: показывает в графическом редакторе, отображает нужную вкладку конфигурации устройства и т.д.

На вкладке *Вывод сборки* отображаются потребляемые программой ресурсы. Доступ к вкладке осуществляется через пункт меню *Вид – Вывод сборки*.

Ресурсы программы:

- Код программы – 2048 байт
- Переменные
  - bool** - 256 шт
  - int / float** - 512 байт (по 4 байта на одну переменную)
- Общий размер файла, загружаемого в устройство – 16384 байт.

Информация об используемых ресурсах располагается в правой части статусной панели:

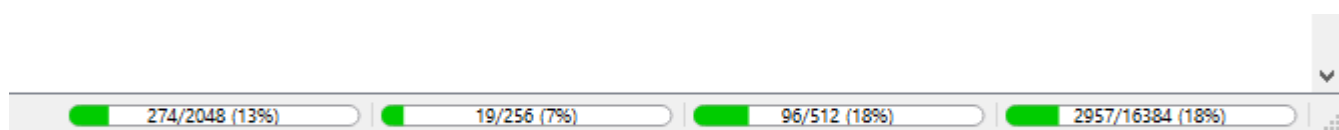



Рисунок 5.1– Ресурсы программы

## 6. ОТЛАДКА ПРОГРАММЫ


Все отладочные опции доступны при подключённом устройстве.

### 6.1 Запуск отладки

Для отладки программы на устройстве выберите пункт меню **Отладка – Начать отладку** или нажмите кнопку  на панели инструментов. После этого приложение выполнит следующие действия:

- Выполнит сборку проекта. Если в результате сборки будут сообщения об ошибках, отладка прервётся.
- Если в результате сборки будут сообщения о некорректной конфигурации устройства, будет предложено прервать отладку (если иное не выбрано в настройках).
- Предложит загрузить конфигурацию в устройство (если иное не выбрано в настройках). При согласии будет произведена загрузка конфигурации и ожидание перезагрузки устройства.
- Загрузит программу в устройство.
- Войдёт в режим отладки, с остановкой на первой исполняемой функции.

### 6.2 Подключение отладчика к работающей программе

Для отладки уже работающего устройства выберите пункт меню **Отладка – Подключиться к работающему устройству** или нажмите кнопку  на панели инструментов. После этого приложение выполнит следующие действия:

- Предложит скачать конфигурацию из устройства (если иное не выбрано в настройках).
- Скачает программу из устройства и откроет в редакторе.
- Запустит сборку программы, при наличии сообщений об ошибках подключение прервётся.
- Войдёт в режим отладки, при этом программа будет продолжать исполняться.

### 6.3 Работа в режиме отладки

В режиме отладки приложение запрещает изменять текущую схему. Панель с элементами схемы в средней части заменяется отладочной панелью.

На отладочной панели отображаются: строка состояния программы, кнопки для управления программой, список переменных, информация о времени исполнения цикла программы.

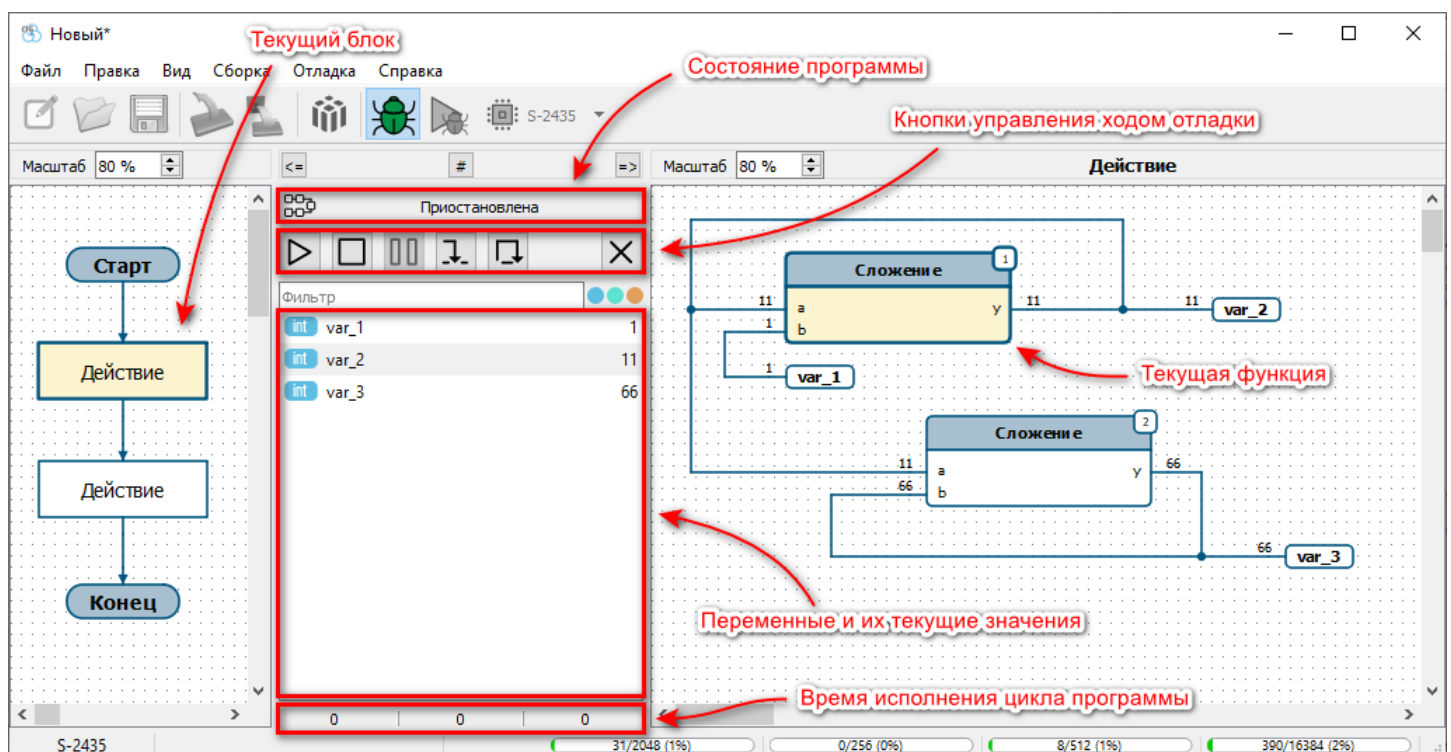


Рисунок 6.1 – Внешний вид редактора в режиме отладки

### 6.3.1 Строка состояния программы








Программа может находиться в следующих состояниях:

- *Нет программы* – в устройство не загружена программа или загружена с ошибкой.
- *Ошибка* – в процессе исполнения программы возникла ошибка.
- *Остановлена* – исполнение программы остановлено. При последующем старте программы произойдёт инициализация переменных и запуск с первой функции.
- *Загрузка* – запись программы в устройство.
- *Приостановлена* – работа программы приостановлена. При возобновлении работы программа продолжит исполнение с текущей функции. В данном режиме текущая функция и её блок подсвечены в окнах редактора.
- *Исполнение* – устройство исполняет программу.

После выхода из режима отладки, устройство запустит или продолжит работу программы (в зависимости от текущего состояния), но только если программа не была в режимах: *Нет программы* или *Ошибка*.

### 6.3.2 Управление исполнением программы

Для управления ходом исполнения программы предусмотрены специальные кнопки под строкой состояния программы (эти кнопки продублированы в меню *Отладка*):

- *Продолжить*  – если программа в состоянии *Остановлена* – запускает программу на исполнение, если программа в состоянии *Приостановлена* – продолжает работу с текущей функции.
- *Стоп*  – останавливает исполнение программы (переводит в состояние *Остановлена*).
- *Пауза*  – приостанавливает исполнение программы (переводит в состояние *Приостановлена*).
- *Шаг*  – исполняет одну функцию и приостанавливается на следующей.
- *Цикл*  – исполняет все функции до тех пор, пока не перейдёт в начало программы, на первой функции приостанавливается.
- *Отправить пользовательскую команду*  – открывает диалог отправки данных в пользовательскую команду. Данная команда отображается на средней панели, только если в программе используется функция **CMD**.
- *Завершить отладку*  – завершает отладку программы, выводит устройство из отладочного режима и переводит редактор в обычный режим.

Для остановки программы перед исполнением конкретной функции в приложении предусмотрены точки останова. Для установки и снятия точки останова, нужно кликнуть правой клавишей мыши по интересующей функции и выбрать пункт меню *Поставить/снять точку останова* (продублирован в меню *Отладка*). Установка точек останова доступна также в режиме редактирования схемы проекта. Устройство физически поддерживает до 8 точек останова. Список текущих точек останова можно просмотреть на вкладке *Точки останова* (открывается через пункт меню *Вид – Точки останова*). Через данную вкладку можно удалять точки останова, выбрав нужные и нажав клавишу **Del**. При двойном клике по точке останова в списке приложение покажет функцию, на которой она установлена.

### 6.3.3 Просмотр значений данных схемы

В режиме отладки приложение на схеме отображает текущие значения на входах и выходах функций (непосредственно над каждым выводом). Данные считываются с устройства с периодом, заданным в настройках приложения.



Под кнопками в средней части программы располагается список используемых переменных с текущими значениями. Данный список можно фильтровать по имени переменной или по типу данных.

### 6.3.4 Время исполнения цикла программы

Программа исполняется циклически. Время одного цикла может изменяться в зависимости от состояния данных программы или от степени загруженности устройства. Для оценки времени исполнения

программы устройство измеряет период цикла. На панели в средней части программы, под списком переменных отображено минимальное, максимальное и усреднённое значение цикла в миллисекундах.

## 6.4 Загрузка и чтение программы без отладки

Редактор позволяет загрузить программу в устройство без входа в режим отладки. Для этого есть пункт меню **Отладка – Записать программу в устройство** (или кнопка  на панели инструментов). Аналогично можно считать программу из устройства и открыть её в редакторе через пункт меню **Отладка – Прочитать программу из устройства** (или кнопку  на панели инструментов).

Важно помнить, что при таком способе загрузки программы редактор не проверяет совместимость конфигурации устройства с загружаемой программой.

## 7. НАСТРОЙКИ

Настройки открываются через пункт меню *Файл – Настройки*. Настройка разделены на следующие группы: *Основные настройки, Настройки отладки, Цветовая схема*.

### 7.1 Основные настройки

В данном окне настраивается интервал автоматического сохранения изменений проекта – поле *Сохранять автоматически*. Доступны следующие значения:

- 10 секунд
- 30 секунд
- 1 минуту
- 5 минут
- 10 минут
- Нет (автоматическое сохранение отключено)

### 7.2 Настройки отладки

В данном окне настраиваются следующие параметры:

- *Период обновления данных* – это временной период в миллисекундах, с которым считывается отладочная информация, если связь с устройством установлена по USB. Минимальное значение 100 мс.
- *Период обновления данных при низкоскоростном соединении* – временной период в миллисекундах, с которым считывается отладочная информация, если связь с устройством установлена по Bluetooth или через сервер RCS. Минимальное значение 1000 мс.
- *Запускать отладку с некорректной конфигурацией устройства* – возможные значения: *Спросить* (по умолчанию), *Нет*, *Да*
- *Загружать конфигурацию в устройство перед отладкой* – возможные значения: *Спросить* (по умолчанию), *Нет*, *Да*
- *Скачать конфигурацию из устройства перед подключением к отладке* – возможные значения: *Спросить* (по умолчанию), *Нет*, *Да*

### 7.3 Цветовая схема

В данном окне настраивается цветовое оформление графических элементов редактора.

Поле *Предустановленные схемы* позволяет выбрать одну из стандартных цветовых схем. Для применения выбранной цветовой схемы необходимо выбрать ее в выпадающем списке и нажать кнопку *Применить*.

Для ручного редактирования цветовых схем редактора можно воспользоваться группой настроек, приведенных ниже.

Параметры в группе *Диаграмма блоков* относятся к интерфейсу, расположенному в левой части редактора. Параметры в группе *Диаграмма функций* относятся к интерфейсу, расположенному в правой части редактора. Параметры в группе *Общее* относятся к общим графическим элементам.

Поле *Для состояния* – определяет состояния, в котором находятся графические элементы. Возможны четыре состояния, первые два общие, следующие два относятся к режиму отладки:

- *Нормальное* – обычное состояние, когда элемент не выбран.
- *Выделен* – когда пользователь выбрал данный элемент, один или несколько.
- *Текущий* – в режиме отладки, программа приостановлена на данном элементе.
- *Текущий выделен* – дополнительно к предыдущему состоянию, элемент выбран.

## 8. ПРИЛОЖЕНИЯ

### 8.1 Сочетания клавиш

<b>Работа с проектом:</b>	
CTRL + N	Создать новый проект
CTRL + O	Открыть проект из файла
CTRL + S	Сохранить проект в файл
<b>Сборка:</b>	
CTRL + B	Собрать проект
CTRL + SHIFT + B	Скомпилировать проект
<b>Отладка:</b>	
F5	Начать отладку, продолжить исполнение
F2	Завершить отладку
F10	Выполнить один цикл
F11	Выполнить одну функцию
F9	Поставить/Снять точку останова

### 8.2 Список кодов событий Complex Events

При работе функции устройство может формировать события со следующими кодами (event\_code):

<b>Код (HEX)</b>	<b>Код (DEC)</b>	<b>Текстовый псевдоним для SMS</b>	<b>Расшифровка</b>
0xA056	41046	CMPLXEVNT_A	Complex Events. Пользовательское событие №1
0xA057	41047	CMPLXEVNT_S	Complex Events. Пользовательское событие №2
0xA058	41048	CMPLXEVNT_F	Complex Events. Пользовательское событие №3
0xA22F	41519	C_CVNT_U	Complex Events. Обновление программы.

## 8.3 Библиотека функциональных блоков

Перечень функциональных блоков:

Название	#	Описание	Количество операндов				Размер, байт	Тип операндов
			IN	OUT	INT	CONST		
<b>Общие</b>								
NOP	3	Нет операции	-	-	-	-	1	-
DELAY	78	Задержка	1	-	-	-	3	int32
MOVE	4	Присваивание	1	1	-	-	5	Любые
MOVE_EN	5	Присваивание по условию	2	1	-	-	7	Любые
<i>TO_FLOAT</i>	<i>6</i>	<i>Преобразовать во float</i>	<i>1</i>	<i>1</i>	-	-	<i>5</i>	<i>int32</i>
<i>FROM_FLOAT</i>	<i>75</i>	<i>Преобразовать из float</i>	<i>1</i>	<i>1</i>	-	-	<i>5</i>	<i>float</i>
<b>Математические операции</b>								
ADD	7	Сложение	2	1	-	-	7	float int32
SUB	8	Вычитание	2	1	-	-	7	float int32
MUL	9	Умножение	2	1	-	-	7	float int32
DIV	10	Деление	2	1	-	-	7	float int32
EXP	11	Возведение в степень	2	1	-	-	7	float int32
MOD	12	Остаток от деления	1	1	-	-	5	float int32
ABS	13	Абсолютное значение	1	1	-	-	5	float int32
SIGN	14	Выделение знака	1	1	-	-	5	float int32
SQRT	15	Квадратный корень	1	1	-	-	5	float
LN	16	Натуральный логарифм	1	1	-	-	5	float
LOG	17	Десятичный логарифм	1	1	-	-	5	float
SIN	18	Синус	1	1	-	-	5	float
COS	19	Косинус	1	1	-	-	5	float
TAN	20	Тангенс	1	1	-	-	5	float
ASIN	21	Арксинус	1	1	-	-	5	float
ACOS	22	Арккосинус	1	1	-	-	5	float
ATAN	23	Арктангенс	1	1	-	-	5	float
<b>Логические операции</b>								
AND	24	Логическое И	2	1	-	-	7	bool
OR	25	Логическое ИЛИ	2	1	-	-	7	bool
XOR	26	Логическое исключающее ИЛИ	2	1	-	-	7	bool
NOT	27	Логическое НЕ	1	1	-	-	5	bool
<b>Побитовые операции</b>								
BAND	28	Побитное И	2	1	-	-	7	int32
BOR	29	Побитное ИЛИ	2	1	-	-	7	int32
BXOR	30	Побитное исключающее ИЛИ	2	1	-	-	7	int32
BNOT	31	Побитное НЕ	1	1	-	-	5	int32
BSHL	32	Битовый сдвиг влево	2	1	-	-	7	int32
BSHR	33	Битовый сдвиг вправо	2	1	-	-	7	int32
CODER	34	Кодер	N	1	-	-	4+2*(N+1)	int32
DECODER	35	Декодер	1	N	-	-	4+2*(N+1)	int32
<b>Сравнение</b>								
EQ	36	Равно	2	1	-	-	7	float int32
NE	37	Не равно	2	1	-	-	7	float int32
GT	38	Больше	2	1	-	-	7	float int32
GE	39	Больше или равно	2	1	-	-	7	float int32
<b>Выбор и ограничение</b>								
SEL	40	Выбор значения	3	1	-	-	9	float int32
MAX	41	Максимальное значение	2	1	-	-	7	float int32
MIN	42	Минимальное значение	2	1	-	-	7	float int32
LIMIT	43	Ограничение	3	1	-	-	7	float int32
MUX	44	Мультиплексор	1+N	1	-	-	4+2*(N+2)	float int32
DMUX	45	Демультимплексор	2	N	-	-	4+2*(N+2)	float int32
APPERTURE	94	Фиксация изменений	2	1	1	-	9	float int32



<b>Триггеры, генераторы, счётчики</b>								
SR	46	Триггер с доминантой включения	2	1	-	-	7	bool
RS	47	Триггер с доминантой выключения	2	1	-	-	7	bool
TT	48	T-триггер	1	1	1	-	7	bool
TP	49	Генератор импульса	2	2	2	-	11	bool
BLINK	50	Генератор импульсов	3	3	1	-	15	bool
TON	51	Таймер с задержкой включения	2	2	2	-	13	bool
TOFF	52	Таймер с задержкой выключения	2	2	2	-	13	bool
RISING	53	Детектор переднего фронта	1	1	1	-	7	bool
FALLING	54	Детектор заднего фронта	1	1	1	-	7	bool
CNT	55	Счётчик	5	3	2	-	21	bool
RAND	56	Генератор случайных чисел	-	1	-	-	3	int32
PWM	57	ШИМ генератор	2	2	1	-	11	int32
<b>Специальные функции</b>								
EVENT	58	Генератор событий	2	-	1	2	9	int32
CMD	59	Команда от устройства	-	6	-	-	13	int32
FLEX	60	Считывание значения из FLEX таблицы	-	1	-	3	6	int32
USER_PARAM	61	Запись значения в пользовательский параметр	2	-	-	1	6	int32
SMS	62	Отправить СМС	1	1	1	2+N	9+N	bool
USER_SMS	79	Отправить нестандартное СМС	1+N	1	1	M+L	7+2·N+M+L	bool
RECV_SMS	80	Получено СМС	0	1	0	1+N+M	4+N+M	bool
CALL	63	Сделать звонок	1	1	1	2	9	bool
CAM	64	Сделать снимок	1	1	1	-	7	bool
GEOZONE	65	Геозона	5	1	1	1	16	float int32
CALENDAR	76	Календарь	2	7	-	-	19	int32
INFO	95	Об устройстве	-	2	-	-	5	int32
IMEI	96	IMEI модема	-	2	-	-	5	int32
ICCID	97	ICCID SIM карты	1	2	-	-	7	int32
IMSI	98	IMSI SIM карты	1	2	-	-	7	int32
LOG_MSG	106	Отправить сообщение в лог	1+N	0	1	M	5+2·N+M	bool
<b>Функции доступа к периферийным устройствам</b>								
INPUT	66	Вход	1	2	-	1	8	int32
OUTPUT	67	Выход	1	-	1	1	6	int32
HYGRO	68	Гигрометр	-	2	-	1	6	float
ACCEL	69	Акселерометр	-	9	-	-	19	int32
ECODRIVE	70	EcoDrive	-	9	-	-	19	int32
ONEWIRE_KEY	71	OneWire ключ	-	3	-	-	7	int32
RFID	72	RFID	-	5	-	-	11	int32
TACHOGRAPH	73	Tachograph driver	-	7	-	1	16	int32
GUARD	74	Режим охраны	1	2	1	1	9	float int32
CRASH_FILE	77	Формирование файла ДТП	2	3	2	-	15	bool
PWRSAVE	81	Управление энергосбережением	6	-	-	-	13	bool
<b>Функции доступа к цифровым портам</b>								
<i>RXD_GET</i>	82	<i>Прочитать значение из RXD буфера</i>	2	1	-	1	8	<i>float int32</i>

RXD_CMP	83	Поиск данных в RXD буфере	1	1	-	1+N	6+N	int32
RXD_STR2INT	84	Преобразовать строку из RXD буфера в целое число	1	1	-	-	5	int32
RXD_STR2FLOAT	85	Преобразовать строку из RXD буфера в число с плавающей точкой	1	1	-	-	5	float
RXD_CHECKSUM	86	Проверка контрольной суммы в RXD буфере	3	1	-	2	11	bool
TXD_INIT	87	Инициализация TXD буфера	1	-	-	1+N	4+N	bool
<del>TXD_SET</del>	<del>88</del>	<del>Запись значения в TXD буфер</del>	<del>4</del>	<del>-</del>	<del>-</del>	<del>1</del>	<del>10</del>	<del>float int32</del>
TXD_CHECKSUM	89	Записать контрольную сумму в TXD буфер	4	-	-	2	11	bool
TXD_GET	90	Прочитать значение из TXD буфера	2	1	-	1	8	float int32
RS_TRANS	91	Запрос/ответ через последовательный порт	3	3	-	2	15	bool
RS_SEND	92	Отправить данные в последовательный порт	2	1	-	1	8	bool
RS_RECV	93	Принять данные из последовательного порта	2	3	-	2	14	bool
RXD_GET	107	Прочитать значение из RXD буфера	2	N	-	1	7+2·N	float int32
TXD_SET	108	Запись значения в TXD буфер	3+N	-	-	1	9+2·N	float int32
MODBUS_READ	109	Чтение данных по протоколу Modbus RTU	1	2+N	1	7	9+2·N + 10	float int32  bool
MODBUS_WRITE	110	Запись данных по протоколу Modbus RTU	1+N	2	1	7	9+2·N + 10	float int32  bool

## 8.3.1 Общие инструкции

### 8.3.1.1 NOP - нет операции

Инструкция ничего не делает и не имеет входов, выходов.

### 8.3.1.2 DELAY - задержка

	Сигнатура	Тип	Описание
<b>Входы</b>	period	int32	Длительность задержки в мс.



Блок задерживает работу Complex Events на время, заданное входом period. Поэтому при работе программы и при отладке нельзя увидеть текущее оставшееся время задержки. Но если поставить отладчик на паузу в момент исполнения задержки, то он подсветит нужный блок «DELAY» на схеме.

### 8.3.1.3 MOVE - присваивание

	Сигнатура	Тип	Описание
<b>Входы</b>	x	float, int32, bool	Операнд на входе. Значение на входе x копируется в значение на выходе y
<b>Выходы</b>	y	float, int32, bool	Операнд на выходе



Тип блока определяется, типом значения на входе x

### 8.3.1.4 MOVE\_EN - присваивание по условию

	Сигнатура	Тип	Описание
<b>Входы</b>	x	float, int32, bool	Операнд на входе
	enable	bool	Условие копирования. Значение на входе x копируется в значение на выходе y, если enable = true, в противном случае y не изменяется.
<b>Выходы</b>	y	float, int32, bool	Операнд на выходе



Тип блока определяется, типом значения на входе x

### 8.3.1.5 (устарело) *TO\_FLOAT* – преобразовать *int32(IEEE754)* во *float*



Функция скрыта, начиная с версии редактора v3.3.0.

	Сигнатура	Тип	Описание
<b>Входы</b>	x	int32	Операнд на входе
<b>Выходы</b>	y	float	Операнд на выходе

Функция интерпретирует целочисленное значение, пришедшее на вход **x** как число с плавающей точкой, записанное по стандарту IEEE754 и переводит его в более легкое для восприятия и вычислений представление.

INT32		FLOAT
<b>1095977927</b>	=	<b>13.206</b>

### 8.3.1.6 (устарело) *FROM\_FLOAT* – преобразовать *float* в *int32(IEEE754)*



Функция скрыта, начиная с версии редактора v3.3.0.

	Сигнатура	Тип	Описание
<b>Входы</b>	x	float	Операнд на входе
<b>Выходы</b>	y	int32	Операнд на выходе

Функция переводит число с плавающей точкой, пришедшее на вход **x** в целочисленный формат записи по стандарту IEEE754.

FLOAT		INT32
<b>13.206</b>	=	<b>1095977927</b>

## 8.3.2 Математические инструкции

### 8.3.2.1 ADD – сложение

	Сигнатура	Тип	Описание
Входы	<i>a</i>	float, int32	Слагаемое 1
	<i>b</i>	float, int32	Слагаемое 2
Выходы	<i>y</i>	float, int32	Сумма

$$y = a + b$$



Тип блока определяется, типом значения на входе **a**

### 8.3.2.2 SUB – вычитание

	Сигнатура	Тип	Описание
Входы	<i>a</i>	float, int32	Уменьшаемое
	<i>b</i>	float, int32	Вычитаемое
Выходы	<i>y</i>	float, int32	Разность

$$y = a - b$$



Тип блока определяется, типом значения на входе **b**

### 8.3.2.3 MUL – умножение

	Сигнатура	Тип	Описание
Входы	<i>a</i>	float, int32	Множитель 1
	<i>b</i>	float, int32	Множитель 2
Выходы	<i>y</i>	float, int32	Произведение

$$y = a \cdot b$$



Тип блока определяется, типом значения на входе **a**

### 8.3.2.4 DIV – деление

	Сигнатура	Тип	Описание
Входы	<i>a</i>	float, int32	Делимое
	<i>b</i>	float, int32	Делитель
Выходы	<i>y</i>	float, int32	Частное

$$y = \frac{a}{b}$$



Тип блока определяется, типом значения на входе **a**

### 8.3.2.5 EXP – возведение в степень

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Значение
	$b$	float, int32	Степень
Выходы	$y$	float, int32	Результат

$$y = a^b$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.2.6 MOD – модуль числа (остаток от деления)

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Делимое
	$b$	float, int32	Делитель
Выходы	$y$	float, int32	Результат

$$y = a \% b$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.2.7 ABS – абсолютное значение

	Сигнатура	Тип	Описание
Входы	$x$	float, int32	Входной операнд
Выходы	$y$	float, int32	Результат

$$y = |x|$$



Тип блока определяется, типом значения на входе  $x$

### 8.3.2.8 SIGN – выделение знака

	Сигнатура	Тип	Описание
Входы	$x$	float, int32	Входной операнд
Выходы	$y$	float, int32	Результат

$$y = \begin{cases} x > 0, & 1 \\ x = 0, & 0 \\ x < 0, & -1 \end{cases}$$



Тип блока определяется, типом значения на входе  $x$

### 8.3.2.9 SQRT – квадратный корень

	Сигнатура	Тип	Описание
Входы	$x$	float	Входной операнд
Выходы	$y$	float	Результат

$$y = \sqrt{x}$$

### 8.3.2.10 LN – натуральный логарифм

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \ln x$$

### 8.3.2.11 LOG – десятичный логарифм

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \log x$$

### 8.3.2.12 SIN – синус

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \sin(x)$$

### 8.3.2.13 COS – косинус

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \cos(x)$$

### 8.3.2.14 TAN – тангенс

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \operatorname{tg}(x)$$

### 8.3.2.15 ASIN – арксинус

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \operatorname{asin}(x)$$

### 8.3.2.16 ACOS – арккосинус

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \operatorname{acos}(x)$$

### 8.3.2.17 ATAN – арктангенс

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float	Входной операнд
<b>Выходы</b>	$y$	float	Результат

$$y = \operatorname{atg}(x)$$

### 8.3.3 Логические инструкции

#### 8.3.3.1 AND – логическое И

	Сигнатура	Тип	Описание
Входы	$a$	bool	Операнд 1
	$b$	bool	Операнд 2
Выходы	$y$	bool	Конъюнкция

$$y = a \wedge b$$

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

#### 8.3.3.2 OR – логическое ИЛИ

	Сигнатура	Тип	Описание
Входы	$a$	bool	Операнд 1
	$b$	bool	Операнд 2
Выходы	$y$	bool	Дизъюнкция

$$y = a \vee b$$

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

#### 8.3.3.3 XOR – логическое исключающее ИЛИ

	Сигнатура	Тип	Описание
Входы	$a$	bool	Операнд 1
	$b$	bool	Операнд 2
Выходы	$y$	bool	Строгая дизъюнкция

$$y = a \oplus b$$

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

#### 8.3.3.4 NOT – логическое НЕ

	Сигнатура	Тип	Описание
Входы	$x$	bool	Входной операнд
Выходы	$y$	bool	Инверсия

$$y = \bar{x}$$

x	y
0	1
1	0



## 8.3.4 Побитовые инструкции

### 8.3.4.1 BAND – побитное И

	Сигнатура	Тип	Описание
Входы	<i>a</i>	int32	Операнд 1
	<i>b</i>	int32	Операнд 2
Выходы	<i>y</i>	int32	Побитная конъюнкция

$$y = a \bigwedge b$$

Операнд	Значение (DEC)	Значение (HEX)	Значение (BIN)							
			Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
<b>a</b>	150	0x96	1	0	0	1	0	1	1	0
<b>b</b>	85	0x55	0	1	0	1	0	1	0	1
<b>y</b>	20	0x14	0	0	0	1	0	1	0	0

### 8.3.4.2 BOR – побитное ИЛИ

	Сигнатура	Тип	Описание
Входы	<i>a</i>	int32	Операнд 1
	<i>b</i>	int32	Операнд 2
Выходы	<i>y</i>	int32	Побитная дизъюнкция

$$y = a \bigvee b$$

Операнд	Значение (DEC)	Значение (HEX)	Значение (BIN)							
			Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
<b>a</b>	150	0x96	1	0	0	1	0	1	1	0
<b>b</b>	85	0x55	0	1	0	1	0	1	0	1
<b>y</b>	215	0xD7	1	1	0	1	0	1	1	1

### 8.3.4.3 BXOR – побитное исключающее ИЛИ

	Сигнатура	Тип	Описание
Входы	<i>a</i>	int32	Операнд 1
	<i>b</i>	int32	Операнд 2
Выходы	<i>y</i>	int32	Побитная строгая дизъюнкция

$$y = a \oplus b$$

Операнд	Значение (DEC)	Значение (HEX)	Значение (BIN)							
			Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
<b>a</b>	150	0x96	1	0	0	1	0	1	1	0
<b>b</b>	85	0x55	0	1	0	1	0	1	0	1
<b>y</b>	193	0xC3	1	1	0	0	0	0	1	1

### 8.3.4.4 BNOT – побитное НЕ

	Сигнатура	Тип	Описание
Входы	<i>x</i>	int32	Входной операнд
Выходы	<i>y</i>	int32	Побитная инверсия

$$y = \bar{x}$$

Операнд	Значение (DEC)	Значение (HEX)	Значение (BIN)							
			Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
<b>x</b>	150	0x96	1	0	0	1	0	1	1	0
<b>y</b>	105	0x69	0	1	1	0	1	0	0	1

### 8.3.4.5 BSHL – битовый сдвиг влево

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	int32	Сдвигаемое
	$n$	int32	Величина сдвига (количество сдвигаемых бит)
<b>Выходы</b>	$y$	int32	Результат сдвига

$$y = x \ll n$$

0 0 0    0 0 0 1 0 0 1 0 1 1 0

X = 150 (dec) = 0x96 (hex)

$$Y = X \ll 2$$

0 0 0    0 1 0 0 1 0 1 1 0 0 0

Y = 600 (dec) = 0x258 (hex)

### 8.3.4.6 BSHR – битовый сдвиг вправо

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	int32	Сдвигаемое
	$n$	int32	Величина сдвига (количество сдвигаемых бит)
<b>Выходы</b>	$y$	int32	Результат сдвига

$$y = x \gg n$$

0 0 0    0 0 0 1 0 0 1 0 1 1 0

X = 150 (dec) = 0x96 (hex)

$$Y = X \gg 2$$

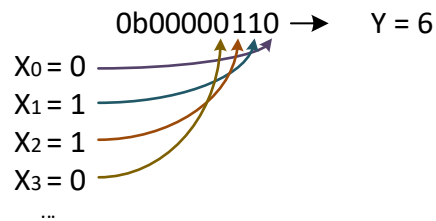
0 0 0    0 0 0 0 0 1 0 0 1 0 1 1 0

Y = 37 (dec) = 0x25 (hex)

### 8.3.4.7 CODER – кодер

	Сигнатура	Тип	Описание
<b>Входы</b>	$x_0$	bool	Разряд 0
	$x_1$	bool	Разряд 1
	...		
	$x_{N-1}$	bool	Разряд $N-1$
<b>Выходы</b>	$y$	int32	Поразрядная сумма

$$y = \sum_{i=0}^{N-1} x_i \ll i$$

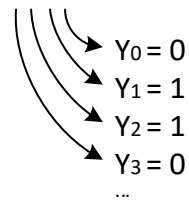


### 8.3.4.8 DECODER – декодер

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	int32	Входное значение
<b>Выходы</b>	$y_0$	bool	Разряд 0
	$y_1$	bool	Разряд 1
	...		
	$y_{N-1}$	bool	Разряд $N-1$

$$y_i = (x \wedge (1 \ll i)) \gg i, \quad i \in 0..N-1$$

$$X = 6 \rightarrow 0b000000110$$



## 8.3.5 Инструкции сравнения

### 8.3.5.1 EQ – равно

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
Выходы	$y$	bool	Результат $true$ , если $a = b$

$$y = \begin{cases} a = b, & true \\ & false \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.5.2 NE – не равно

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
Выходы	$y$	bool	Результат $true$ , если $a \neq b$

$$y = \begin{cases} a \neq b, & true \\ & false \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.5.3 GT – больше

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
Выходы	$y$	bool	Результат $true$ , если $a > b$

$$y = \begin{cases} a > b, & true \\ & false \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.5.4 GE – больше или равно

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
Выходы	$y$	bool	Результат $true$ , если $a \geq b$

$$y = \begin{cases} a \geq b, & true \\ & false \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

## 8.3.6 Инструкции выбора и ограничения

### 8.3.6.1 SEL – выбор значения

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
	$n$	bool	Операнд выбора. Если $n$ равно «1», то на выход будет передано значение входа $b$ . Иначе на выход будет передано значение входа $a$ .
Выходы	$y$	float, int32	Результат

$$y = \begin{cases} n = false, & a \\ & b \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.6.2 MAX – максимальное значение

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
Выходы	$y$	float, int32	Максимальное значение

$$y = \begin{cases} a > b, & a \\ & b \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.6.3 MIN – минимальное значение

	Сигнатура	Тип	Описание
Входы	$a$	float, int32	Операнд 1
	$b$	float, int32	Операнд 2
Выходы	$y$	float, int32	Минимальное значение

$$y = \begin{cases} a < b, & a \\ & b \end{cases}$$



Тип блока определяется, типом значения на входе  $a$

### 8.3.6.4 LIMIT – ограничение

	Сигнатура	Тип	Описание
Входы	$x$	float, int32	Входной операнд
	$max$	float, int32	Максимум
	$min$	float, int32	Минимум
Выходы	$y$	float, int32	Если $x$ меньше $min$ то на выходе будет установлено $min$ . Если $x$ больше $max$ то на выходе будет установлено $max$ . Иначе на выходе будет установлено значение $x$ .

$$y = \begin{cases} x > max, & max \\ x < min, & min \\ & x \end{cases}$$



Тип блока определяется, типом значения на входе  $x$

### 8.3.6.5 MUX – мультиплексор

	Сигнатура	Тип	Описание
Входы	$x_0$	float, int32	Вход 0
	$x_1$	float, int32	Вход 1
	...		
	$x_{N-1}$	float, int32	Вход $N-1$
	$k$	int32	Номер входа, значение которого будет передано на выход.
Выходы	$y$	float, int32	Выход принимает значение одного из входов.

$$y = x_k, \quad k \in 1 \dots N - 1$$



Тип блока определяется, типом значения на входе  $x$

### 8.3.6.6 DMUX – демультимплексор

	Сигнатура	Тип	Описание
Входы	$x$	float, int32	Вход. Значение, которое будет передано на один из выходов.
	$k$	int32	Номер выхода, на который будет передано входное значение.
Выходы	$y_0$	float, int32	Выход 0
	$y_1$	float, int32	Выход 1
	...		
	$y_{N-1}$	float, int32	Выход $N-1$

$$y_i = \begin{cases} x, & i = k \\ 0, & \end{cases}, \quad k \in 0 \dots N - 1$$

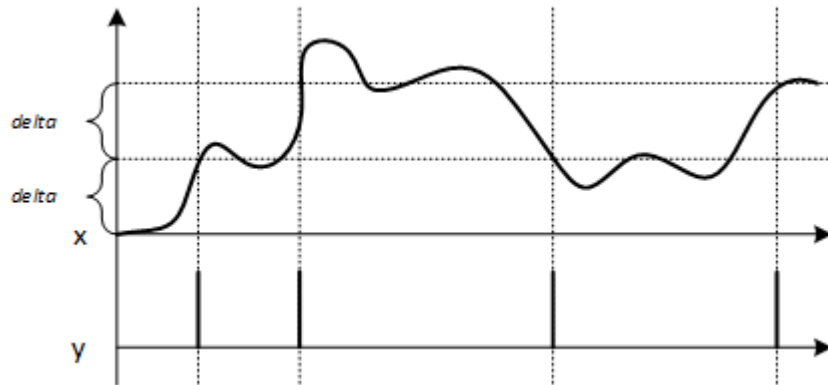


Тип блока определяется, типом значения на входе  $x$

### 8.3.6.7 APPERTURE – фиксация изменений

	Сигнатура	Тип	Описание
<b>Входы</b>	$x$	float, int32	Входной операнд
	$delta$	float, int32	Величина, при изменении на которую выход $y$ устанавливается в <i>true</i>
<b>Выходы</b>	$y$	bool	Результат
<b>Внутренние</b>	$x\_old$	float, int32	Значение $x$ , при предыдущей фиксации

$$y = \begin{cases} |x - x\_old| \geq delta, & true \\ false & \end{cases}$$



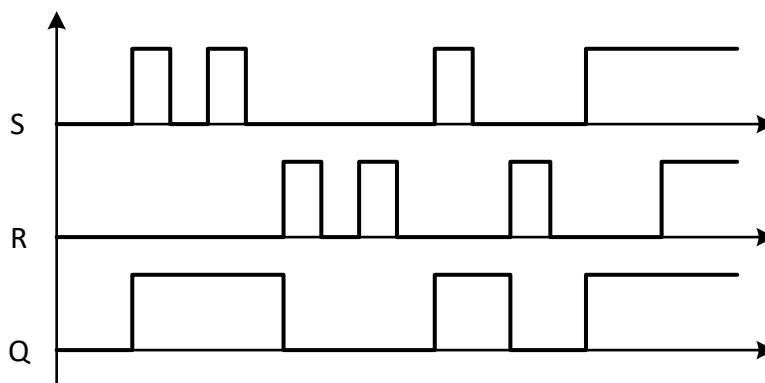
Тип блока определяется, типом значения на входе  $x$

### 8.3.7 Триггеры, генераторы, счётчики

#### 8.3.7.1 SR – триггер с доминантой включения

	Сигнатура	Тип	Описание
<b>Входы</b>	$S$	bool	Установка выхода. Когда на вход $S$ приходит «1», выход $Q$ устанавливается в «1». Вход $S$ «доминантный», т.е. если на входах $S$ и $R$ установлены «1», то выход $Q$ будет установлен в «1».
	$R$	bool	Сброс выхода. Когда на вход $R$ приходит «1», выход $Q$ устанавливается в «0».
<b>Выходы</b>	$Q$	bool	Выход

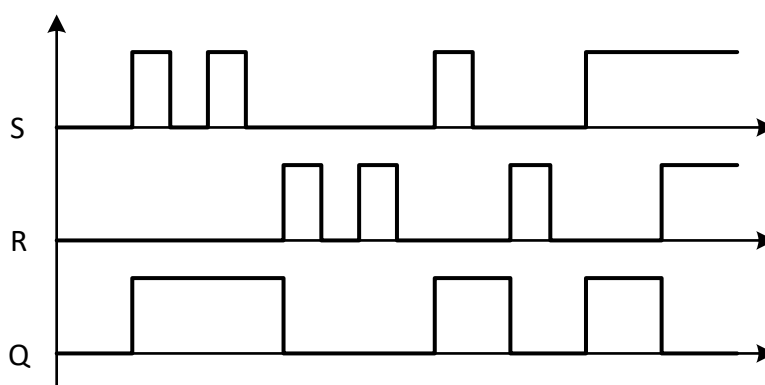
$$Q = (\bar{R} \wedge Q) \vee S$$



#### 8.3.7.2 RS – триггер с доминантой выключения

	Сигнатура	Тип	Описание
<b>Входы</b>	$S$	bool	Установка выхода. Когда на вход $S$ приходит «1», выход $Q$ устанавливается в «1».
	$R$	bool	Сброс выхода. Когда на вход $R$ приходит «1», выход $Q$ устанавливается в «0». Вход $R$ «доминантный», т.е. если на входах $S$ и $R$ установлены «1», то выход $Q$ будет установлен в «0».
<b>Выходы</b>	$Q$	bool	Выход

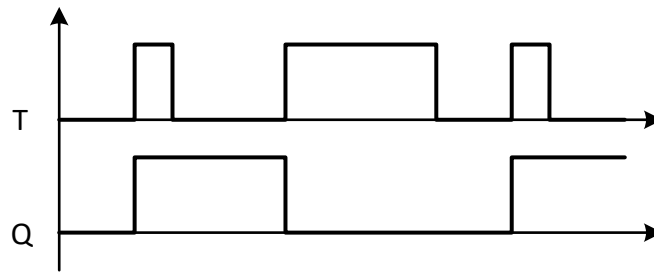
$$Q = \bar{R} \wedge (Q \vee S)$$





### 8.3.7.3 ТТ – Т-триггер

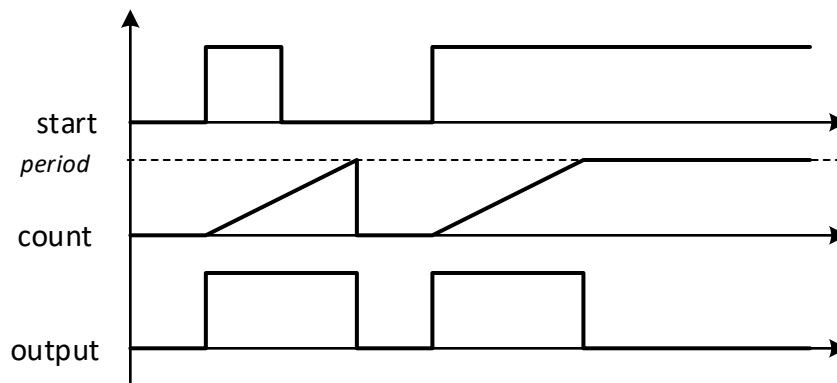
	Сигнатура	Тип	Описание
<b>Входы</b>	<i>T</i>	bool	Вход. Когда на вход <i>T</i> приходит перепад от «0» к «1», выход <i>Q</i> инвертируется.
<b>Выходы</b>	<i>Q</i>	bool	Выход
<b>Внутренние</b>	<i>old</i>	bool	Значение <i>T</i> на предыдущем такте



### 8.3.7.4 ТР – генератор импульса

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>start</i>	bool	Запуск (передний фронт)
	<i>period</i>	int32	Длительность импульса в мс
<b>Выходы</b>	<i>output</i>	bool	Выход. Устанавливается в 1, входом <i>start</i> . Сбрасывается в 0, при достижении счётчика <i>count</i> значения входа <i>period</i> (период).
	<i>count</i>	int32	Значение счётчика в мс. Запускается входом <i>start</i> .
<b>Внутренние</b>	<i>tick</i>	int32	Начало отсчёта счётчика <i>count</i>
	<i>old</i>	bool	Значение <i>start</i> на предыдущем такте

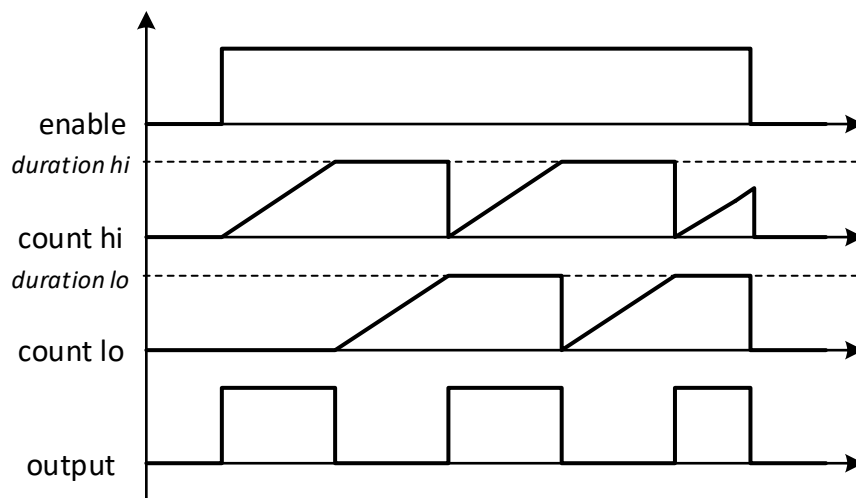
Когда на вход *start* поступает «1», на выходе *output* возникает «1» и запускается внутренний счётчик *count*. Когда счётчик достигнет значения *period*, счёт останавливается и выход *output* обнуляется.



### 8.3.7.5 BLINK – генератор импульсов

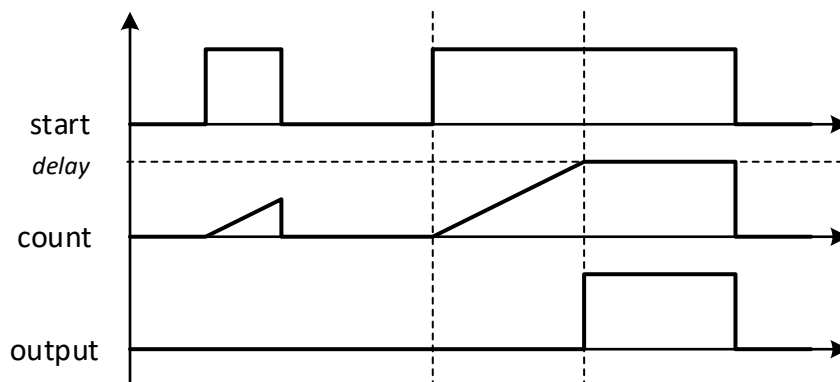
	Сигнатура	Тип	Описание
Входы	<i>enable</i>	bool	Разрешающий сигнал
	<i>duration hi</i>	int32	Длительность состояний логической «1» в мс
	<i>duration lo</i>	int32	Длительность состояний логического «0» в мс
Выходы	<i>output</i>	bool	Выход. Находится в состоянии логической «1», при счёте счётчика <i>count hi</i> , и в состоянии логического «0» при счёте счётчика <i>count lo</i> .
	<i>count hi</i>	int32	Счётчик состояние логической «1» в мс. Сбрасывается одновременно со сбросом счётчика <i>count lo</i> .
	<i>count lo</i>	int32	Счётчик состояние логического «0» в мс. Сбрасывается при достижении значения <i>duration lo</i>
Внутренние	<i>TCK</i>	int32	Начало отсчёта счётчиков <i>count hi</i> и <i>count lo</i>

Когда на входе *enable* логический «0», генератор выключен и все выходы равны нулю. Когда на входе *enable* логическая «1», генератор работает, счётчики *count hi* и *count lo* считают поочередно от нуля до значений *duration hi*, *duration lo* соответственно. Когда считает счётчик *count hi*, выход *output* равен «1», когда считает *count lo*, выход *output* равен «0».



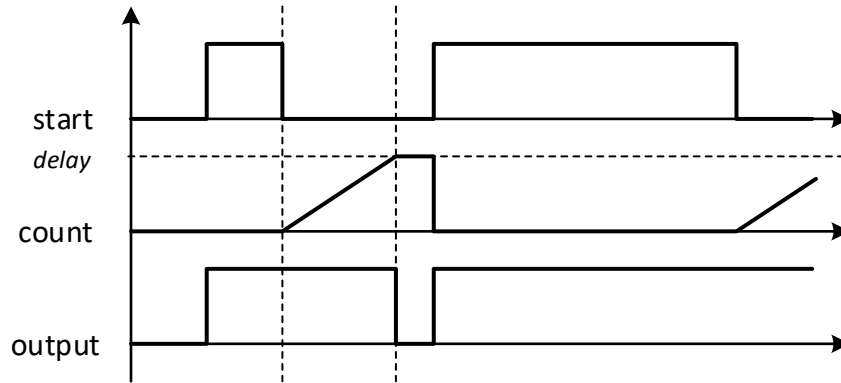
### 8.3.7.6 TON – таймер с задержкой включения

	Сигнатура	Тип	Описание
Входы	<i>start</i>	bool	Запуск
	<i>delay</i>	int32	Длительность включения в мс
Выходы	<i>output</i>	bool	Устанавливается в 1, при достижении счётчика <i>count</i> значения входа <i>delay</i> .
	<i>count</i>	int32	Значение счётчика в мс. Запускается входом <i>start</i> .
Внутренние	<i>old</i>	bool	<i>start</i> на предыдущем такте
	<i>TCK</i>	int32	Начало отсчёта счётчика <i>count</i>



### 8.3.7.7 TOFF – таймер с задержкой выключения

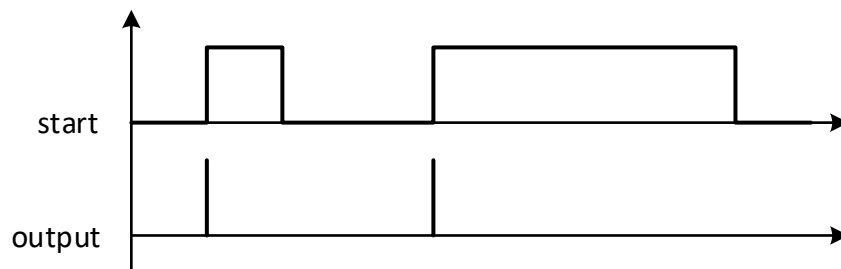
	Сигнатура	Тип	Описание
Входы	<i>start</i>	bool	Запуск
	<i>delay</i>	int32	Длительность выключения в мс
Выходы	<i>output</i>	bool	Устанавливается в 1, входом <i>start</i> . Сбрасывается в 0, при достижении счётчика <i>CNT</i> значения входа <i>delay</i> .
	<i>count</i>	int32	Значение счётчика в мс. Запускается при изменении входа <i>start</i> от 1 к 0.
Внутренние	<i>old</i>	bool	<i>start</i> на предыдущем такте
	<i>TCK</i>	int32	Начало отсчёта счётчика <i>CNT</i>



### 8.3.7.8 RISING – детектор переднего фронта

	Сигнатура	Тип	Описание
Входы	<i>input</i>	bool	Когда на входе <i>input</i> происходит переход от «0» к «1» на один цикл на выходе <i>output</i> устанавливается логическая «1».
Выходы	<i>output</i>	bool	Выход. Устанавливается в 1, при изменении <i>input</i> от 0 к 1.
Внутренние	<i>old</i>	bool	Значение <i>input</i> на предыдущем такте

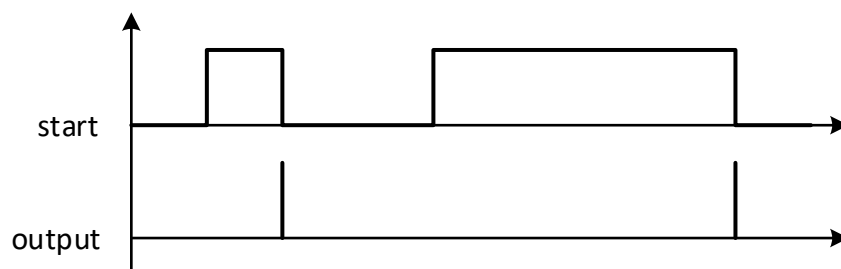
$$output = input \wedge \overline{old}$$



### 8.3.7.9 FALLING – детектор заднего фронта

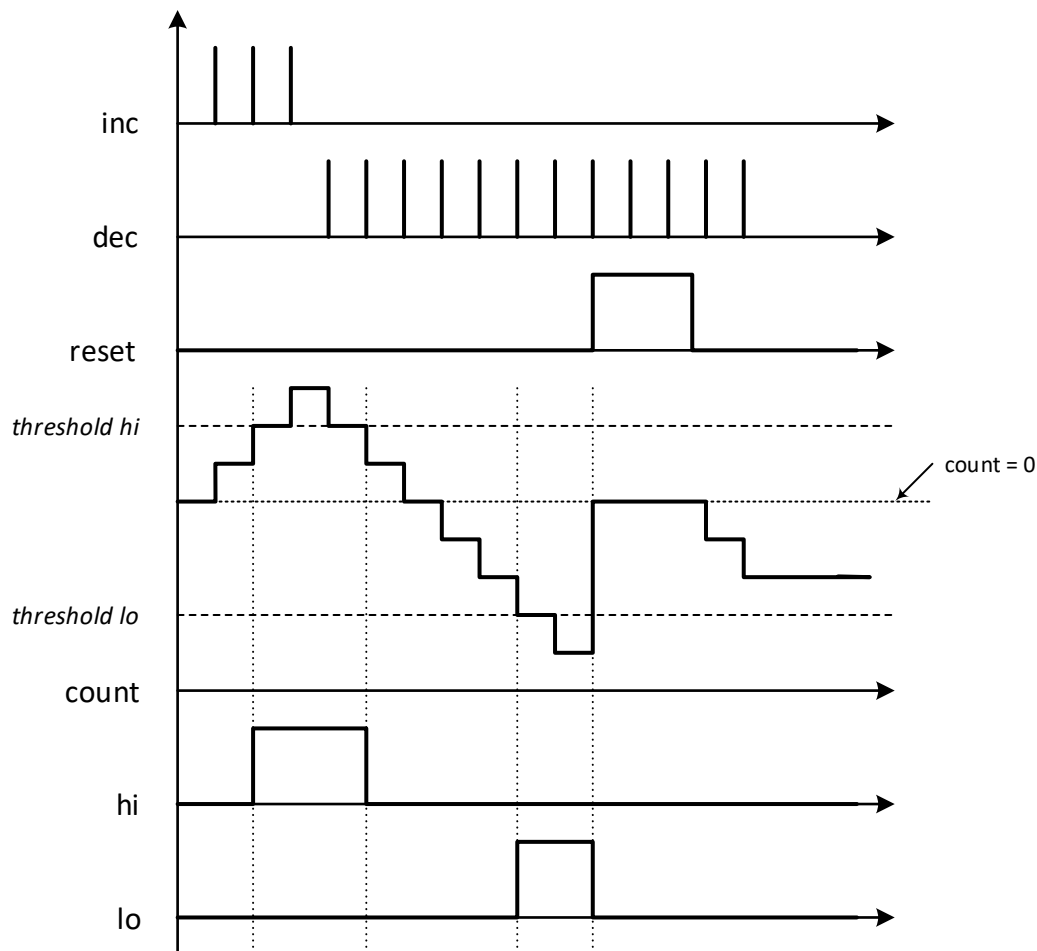
	Сигнатура	Тип	Описание
Входы	<i>input</i>	bool	Когда на входе <i>input</i> происходит переход от «0» к «1» на один цикл программы на выходе <i>output</i> устанавливается логическая «1».
Выходы	<i>output</i>	bool	Выход. Устанавливается в 1, при изменении <i>input</i> от 1 к 0.
Внутренние	<i>old</i>	bool	Значение <i>input</i> на предыдущем такте

$$output = old \wedge \overline{input}$$



### 8.3.7.10 CNT – счётчик

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>inc</i>	bool	Инкрементирующий вход. Когда на входе <i>inc</i> происходит переход от «0» к «1» счётчик <i>count</i> увеличивает своё значение на 1.
	<i>dec</i>	bool	Декрементирующий вход. Когда на входе <i>dec</i> происходит переход от «0» к «1» счётчик <i>count</i> уменьшает своё значение на 1.
	<i>reset</i>	bool	Сброс счётчика <i>count</i> . Когда вход <i>reset</i> равен «1», счётчик <i>count</i> сбрасывается в ноль.
	<i>threshold hi</i>	int32	Верхнее пороговое значение. При достижении счётчика <i>count</i> значения <i>threshold hi</i> выход <i>hi</i> устанавливается в логическую «1».
	<i>threshold lo</i>	Int32	Нижнее пороговое значение. При уменьшении значения счётчика <i>count</i> меньше значения <i>threshold lo</i> выход <i>lo</i> устанавливается в логическую «1».
<b>Выходы</b>	<i>count</i>	int32	Значение счётчика
	<i>hi</i>	bool	Значение счётчика $count \geq threshold\ hi$
	<i>lo</i>	bool	Значение счётчика $count \leq threshold\ lo$
<b>Внутренние</b>	<i>inc old</i>	bool	Значение <i>inc</i> на предыдущем такте
	<i>dec old</i>	bool	Значение <i>dec</i> на предыдущем такте

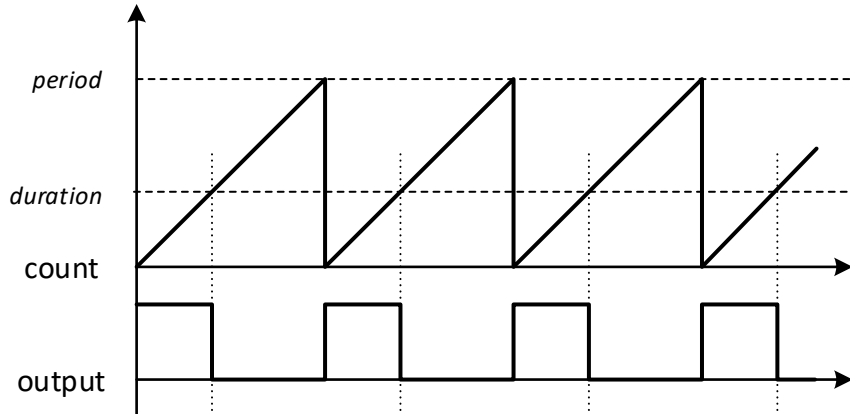


### 8.3.7.11 RAND – генератор случайный чисел

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>output</i>	int32	Псевдослучайное число

### 8.3.7.12 PWM – ШИМ генератор

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>duration</i>	int32	Длительность импульса ШИМ в мс
	<i>period</i>	int32	Период ШИМ в мс
<b>Выходы</b>	<i>output</i>	bool	Выход. Равен 1 когда счётчик <i>count</i> больше или равен длительности импульса <i>duration</i> .
	<i>count</i>	int32	Счётчик ШИМ в мс. Считает от 0 до <i>period-1</i>
<b>Внутренние</b>	<i>tick</i>	int32	Начало отсчёта счётчика <i>count</i>



## 8.3.8 Специальные функции

### 8.3.8.1 EVENT – генератор событий

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>generate</i>	bool	Сигнал генерации события.
	<i>force</i>	bool	Внеочередное событие. Если параметр <i>force</i> равен <i>true</i> , то событие будет отправлено на сервер вне очереди, в противном случае событие обдует отправлено в порядке общей очередности.
<b>Настройки</b>	<i>index</i>	uint8	Номер события. Доступны 3 кода события, которые будут подставлены в №2 поле ( <i>event_code</i> ) протокола FLEX: <b>CE_EVT_1</b> – Событие №41046; <b>CE_EVT_2</b> – Событие №41047; <b>CE_EVT_3</b> – Событие №41048.
	<i>format</i>	uint8	Формат пакета (Функция в разработке)
<b>Внутренние</b>	<i>old</i>	bool	Значение сигнала генерации события, на предыдущем цикле



Если на вход **generate** подключена константа и ее значение *True*, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается сформировать событие.

Если на вход **generate** подключена переменная или выход другой функции, то срабатывание происходит при переходе от *False* к *True*.

### 8.3.8.2 CMD – команда от устройства

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>active</i>	bool	Сигнал о приходе команды
	<i>param1</i>	int32	Параметр 1
	<i>param2</i>	int32	Параметр 2
	<i>param3</i>	int32	Параметр 3
	<i>param4</i>	int32	Параметр 4
	<i>param5</i>	int32	Параметр 5

Для получения параметров от пользователя или системы мониторинга предусмотрена команда, которую устройство может принять по каналам USB, Bluetooth, SMS, Internet.

При получении команды устройство на один цикл работы блока устанавливает на выходе *active* значение «1» (впоследствии оно будет сброшено в «0»).

Выходы *paramX* принимают значения последней принятой команды. (выходы сбрасываются в «0» при первом запуске модуля Complex Events или при получении значения «0» в команде).

Формат команды:

<b>Запрос</b>	*!CEVT<s><param1>[,<param2>,<param3>,<param4>,<param5>] Пример: *!CEVT 120,300 // Допускается не дописывать последние значения *!CEVT 10,,,,200 // Для пропуска промежуточных нужно оставить запятые	
<b>Ответ</b>	*@CEVT	
<b>Канал обмена</b>	Internet, USB, Bluetooth, SMS	
<b>Обозначение</b>	<b>Расшифровка</b>	<b>Формат данных</b>
<s>	Разделитель – пробел (0x20)	char
<param1>	Значение, устанавливаемое на выходе param1. Текстовое значение преобразуется в число I32. Пустое значение воспринимается как 0.	char[]
<param2>	Аналогично параметру <param1>, но для param2	char[]
<param3>	Аналогично параметру <param1>, но для param3	char[]
<param4>	Аналогично параметру <param1>, но для param4	char[]
<param5>	Аналогично параметру <param1>, но для param5	char[]

### 8.3.8.3 FLEX – считывание значения из FLEX таблицы



Функция обновлялась. Текущая реализация используется с версии редактора v3.4.1

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>value</i>	int32	Значение, возвращаемое блоком
<b>Настройки</b>	<i>index</i>	uint8	Номер поля FLEX, из которого необходимо получить значение
	<i>offset</i>	uint8	Смещение в байтах от начала поля (некоторые поля содержат несколько десятков байт)
	<i>type</i>	uint8	Тип параметра для чтения: <b>uint8</b> – однобайтовое число без знака; <b>int8</b> – однобайтовое число со знаком; <b>uint16</b> – двухбайтовое без знака; <b>int16</b> – двухбайтовое со знаком; <b>int32/float</b> – четырехбайтовое со знаком / вещественное.

Логика работы функции зависит от типа данных:

- Если на выход **value** подключена **переменная** с типом **FLOAT** и параметр **type = int32/float**, то функция читает данные из памяти по стандарту IEEE754. Такой способ необходимо использовать для параметров FLEX, которые хранятся в формате Float (Например, параметр "скорость")

- Иначе функция читает данные как число INT32. Такой способ необходимо использовать для параметров FLEX, которые хранятся в любом формате кроме Float.

Конвертация выполняется автоматически при помощи функций **FROM FLOAT** и **TO FLOAT**.

### 8.3.8.4 USER\_PARAM – запись значения в пользовательский параметр

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>value</i>	int32/float	Значение, которое должно быть записано в соответствующий пользовательский параметр
	<i>enable</i>	bool	Условие записи. Значение на входе <i>value</i> записывается, если <i>enable = true</i> , в противном случае значение не записывается.
<b>Настройки</b>	<i>index</i>	uint8	Индекс пользовательского параметра, в который будет произведена запись

Для работы функции, в конфигурации устройства должна быть настроена передача соответствующего пользовательского параметра. Нужно сначала поместить блок на схему, потом (перед компиляцией) внести изменения в конфигурацию.



Конфигурация > Настройка протокола:

..> выбрать «FLEX3.0»

..> Пользовательские параметры > Назначить параметры «Пользовательский параметр СЕх».

Логика работы функции зависит от типа данных:

- Если на вход **value** подключена **переменная** с типом **FLOAT**, то функция записывает данные в память по стандарту IEEE754. Такой способ необходимо использовать для параметров, которые будут прочитаны сервером в формате Float (Например, так следует записать число 12.016). Для передачи такого значения на сервер необходимо использовать пользовательский параметр размером 4 байта.

- Иначе функция записывает данные как целое число. Такой способ необходимо использовать для параметров, которые будут прочитаны сервером в форматах Int или Uint (Например, так следует записать число 43605). Для передачи на сервер можно использовать пользовательский параметр любого размера.

Конвертация выполняется автоматически при помощи функций **FROM FLOAT** и **TO FLOAT**.

### 8.3.8.5 SMS – отправить СМС

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>start</i>	bool	Сигнал отправки SMS.
<b>Выход</b>	<i>active</i>	bool	Выполнение. На выходе появляется <i>true</i> пока устройство не выполнит предыдущую попытку отправки SMS.
<b>Настройки</b>	<i>user</i>	uint8	Номер абонента в памяти устройства.
	<i>type</i>	uint8	Тип сообщения
	<i>message</i>	string	Пользовательская строка, которая будет добавления в сообщение, до 32 символов (ASCII строка, только латинские символы). НЕ используется, если <i>type</i> = «Стандартное SMS».
<b>Внутренние</b>	<i>old</i>	bool	Значение <i>start</i> на предыдущем цикле



Если на вход **start** подключена константа и ее значение *True*, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается отправить сообщение.

Если на вход **start** подключена переменная или выход другой функции, то срабатывание происходит при переходе от *False* к *True*.

### 8.3.8.6 USER\_SMS – отправить нестандартное СМС

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>start</i>	bool	Сигнал отправки SMS.
	<i>value<sub>0</sub></i>	int32/float	Аргумент 0
	<i>value<sub>1</sub></i>	int32/float	Аргумент 1
	...		
	<i>value<sub>N-1</sub></i>	int32/float	Аргумент <i>N-1</i>
<b>Выход</b>	<i>active</i>	bool	Выполнение. На выходе появляется <i>true</i> пока устройство не выполнит текущую попытку отправки SMS.
<b>Настройки</b>	<i>user</i>	string	Строка с произвольным номером телефона или с номером абонента из конфигурации
	<i>message</i>	string	Текст сообщения. В тело сообщения могут быть добавлены аргументы. Пример: Напряжение = {0} В, Температура = {1} *С
<b>Внутренние</b>	<i>old</i>	bool	Значение <i>start</i> на предыдущем цикле



Если на вход **start** подключена константа и ее значение *True*, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается отправить сообщение.

Если на вход **start** подключена переменная или выход другой функции, то срабатывание происходит при переходе от *False* к *True*.

### 8.3.8.7 RECV\_SMS – индикатор получения СМС

	Сигнатура	Тип	Описание
<b>Выход</b>	<i>active</i>	bool	Сигнал о получении СМС, подошедшей по шаблону <i>message</i> с проверкой по условиям <i>flags</i> . На выходе на один цикл исполнения программы появляется <i>true</i> .
<b>Настройки</b>	<i>phone</i>	string	Строка с произвольным номером телефона
	<i>message</i>	string	Текст шаблона (до 16 символов)
	<i>flags</i>	uint8	Параметры проверки



### 8.3.8.8 CALL – сделать звонок

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>start</i>	bool	Сигнал вызова.
<b>Выход</b>	<i>active</i>	bool	Выполнение. На выходе появляется <i>true</i> пока устройство не выполнит предыдущую попытку дозвона.
<b>Настройки</b>	<i>user</i>	uint8	Номер абонента в памяти устройства
	<i>type</i>	uint8	Тип дозвона
<b>Внутренние</b>	<i>old</i>	bool	Значение <i>start</i> на предыдущем цикле



Если на вход **start** подключена константа и ее значение *True*, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается отправить сообщение.

Если на вход **start** подключена переменная или выход другой функции, то срабатывание происходит при переходе от *False* к *True*.

### 8.3.8.9 CAM – сделать снимок

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>start</i>	bool	Сигнал сделать снимок.
<b>Выход</b>	<i>active</i>	bool	Выполнение. На выходе появляется <i>true</i> пока устройство формирует и сохраняет снимок.
<b>Внутренние</b>	<i>old</i>	bool	Значение <i>start</i> на предыдущем цикле



Для работы функции, в конфигурации устройства должна быть настроена работа с фотокамерой.

Конфигурация > RS-232/RS-485 > Использовать как > «Камера».

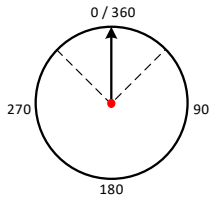


Если на вход **start** подключена константа и ее значение *True*, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается отправить сообщение.

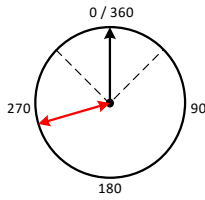
Если на вход **start** подключена переменная или выход другой функции, то срабатывание происходит при переходе от *False* к *True*.

### 8.3.8.10 GEOZONE – Геозона

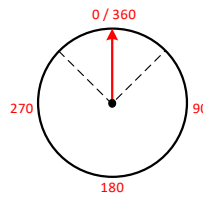
	Сигнатура	Тип	Описание
<b>Вход</b>	<i>latitude</i>	float	Широта центра геозоны (Например: 55,755669)
	<i>longitude</i>	float	Долгота центра геозоны (Например: 37,616802)
	<i>radius</i>	float	Радиус окружности геозоны в метрах
	<i>course</i>	int32	Направление движения (курс) для фиксации входа в геозону
	<i>course delta</i>	int32	Разброс угла направления движения. Если <i>course delta</i> принимает значение 360, то контроль курса для входа в геозону не производится.
<b>Выход</b>	<i>active</i>	bool	Значение <i>true</i> , если объект внутри геозоны и .
<b>Настройки</b>	<i>speed min</i>	int16	Скорость, ниже которой не обновляется <i>current course</i>
<b>Внутренние</b>	<i>current course</i>	int32	Текущий курс



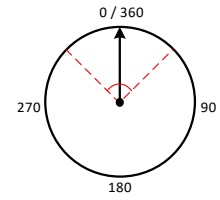
*latitude, longitude*



*radius*



*course*



*course delta*

### 8.3.8.11 CALENDAR – Календарь

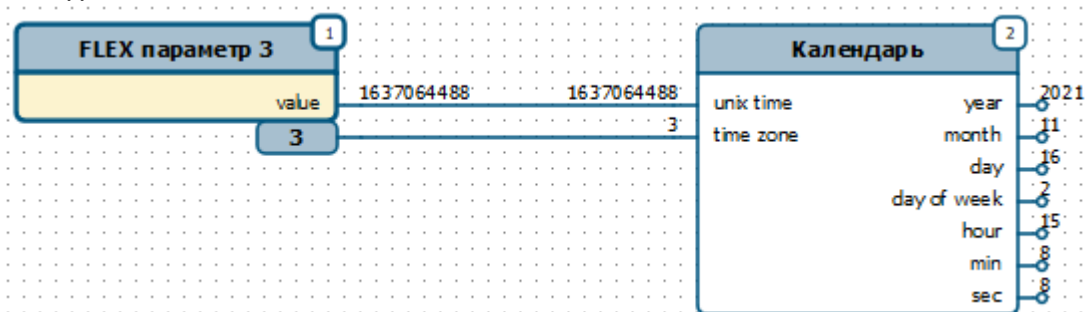
	Сигнатура	Тип	Описание
<b>Вход</b>	<i>UNIX time</i>	int32	Время в формате UNIX-time.
	<i>timezone</i>	int32	Временная зона. Целое число от -12 до 12.
<b>Выход</b>	<i>year</i>	int32	Год
	<i>month</i>	int32	Номер месяца Целое число от 1 до 12. Например: 1 –Январь и т.д.
	<i>day</i>	int32	День месяца. Целое число от 1 до 31.
	<i>day of week</i>	int32	День недели. Целое число от 1 до 7. Например: 1 – Понедельник и т.д.
	<i>hour</i>	int32	Час. Целое число от 0 до 24.
	<i>min</i>	int32	Минута. Целое число от 0 до 59.
	<i>sec</i>	int32	Секунда. Целое число от 0 до 59.



Время формате UNIX-time это целое число, представляющее собой количество секунд, прошедших с 00:00:00 01.01.1970г.

Блок преобразует время в формате UNIX-time с учетом часового пояса в более удобные для использования, отдельные параметры год, месяц день и другие.

Преобразовать текущее время устройства нужно создать блок FLEX для получения поля №3 [time] и подключить на вход *UNIX time*.



### 8.3.8.12 INFO – Информация об устройстве

	Сигнатура	Тип	Описание
Выход	<i>model</i>	int32	Цифровое обозначение модели устройства.
	<i>version</i>	int32	Версия прошивки устройства, представленная как целое число, где младшие 2 цифры расположены в первом байте, средние 2 цифры во втором и старшие 2 цифры в третьем байте.

Например, устройство S-2435 с прошивкой v03.02.31:

model = 2435

version = 197151 (0x0003021F)

### 8.3.8.13 IMEI – IMEI модема

	Сигнатура	Тип	Описание
Выход	<i>digits 8..0</i>	int32	Число, представляющее младшие 9 цифр IMEI.
	<i>digits 14..9</i>	int32	Число, представляющее старшие 6 цифр IMEI.

Например, IMEI 866795030518573 будет представлен так:

digits 8..0 = 30518573

digits 14..9 = 866795



В примере для **digits 8..0** записано не 030518573, а 30518573. Крайние нули слева не отображаются при выводе числовых значений.

### 8.3.8.14 ICCID – ICCID SIM карты

	Сигнатура	Тип	Описание
Вход	<i>SIM index</i>	bool	Номер слота SIM карты: «0» - внешний; «1» - внутренний.
Выход	<i>digits 8..0</i>	int32	Число, представляющее младшие 9 цифр ICCID.
	<i>digits 16..9</i>	int32	Число, представляющее следующие 8 цифр ICCID.

Например, ICCID 8970199201010570553 будет представлен так:

digits 8..0 = 10570553

digits 16..9 = 70199201



В примере для **digits 8..0** записано не 010570553, а 10570553. Крайние нули слева не отображаются при выводе числовых значений.





Длина ICCID номера обычно от 19 до 20 цифр. Функция позволяет получить только младшие 17 цифр. Старшие 2 цифры, для любых SIM карт по стандарту ISO/IEC 7812-1 должны иметь значение '89'.

### 8.3.8.15 IMSI – IMSI SIM карты

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>SIM index</i>	bool	Номер слота SIM карты: «0» - внешний; «1» - внутренний.
<b>Выход</b>	<i>digits 8..0</i>	int32	Число, представляющее младшие 9 цифр IMSI.
	<i>digits 14..9</i>	int32	Число, представляющее старшие 6 цифр IMSI.

Например, IMSI 250991039698855 будет представлен так:  
 digits 8..0 = 39698855  
 digits 14..9 = 250991


 В примере для **digits 8..0** записано не 039698855, а 39698855. Крайние нули слева не отображаются при выводе числовых значений.


 Первые три цифры IMSI это MCC (код страны, например, 250 – Россия). За ним следует две или три цифры MNC (код мобильной сети, например, 99 – Билайн). Все последующие цифры – идентификатор пользователя MSIN.

### 8.3.8.16 LOG\_MSG – отправить сообщение в пользовательский лог

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>send</i>	bool	Сигнал отправки сообщения.
	<i>value<sub>0</sub></i>	int32/float	Аргумент 0
	<i>value<sub>1</sub></i>	int32/float	Аргумент 1
	...		
	<i>value<sub>N-1</sub></i>	int32/float	Аргумент N-1
	<i>message</i>	string	Текст сообщения. В тело сообщения могут быть добавлены аргументы. Пример: Напряжение = {0} В, Температура = {1} *С
<b>Внутренние</b>	<i>old</i>	bool	Значение <i>start</i> на предыдущем цикле

Функция выполняет вывод произвольного текста с аргументами в окно пользовательских логов программы NTC Configurator.

 Для просмотра логов нужно в главном окне программы NTC Configurator перейти «Дополнительно» > «Показать окно логов» > установить флаг «Complex Events».

 Если на вход **send** подключена константа и ее значение True, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается отправить сообщение.  
 Если на вход **send** подключена переменная или выход другой функции, то срабатывание происходит при переходе от False к True.

## 8.3.9 Функции доступа к периферийным устройствам

### 8.3.9.1 INPUT – входная линия

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>reset</i>	bool	Сброс счётчика (если входная линия настроена как «счётчик импульсов»)
<b>Выходы</b>	<i>voltage</i>	int32	Напряжение (проходит через небольшую фильтрацию алгоритмами устройства)
	<i>value</i>	int32	Тип значения зависит от настройки входной линии: <b>«Дискретная»</b> – состояние сработки «1» или «0»; <b>«Аналоговая»</b> – напряжение в мВ (без фильтрации); <b>«Частотная»</b> – частота в Гц; <b>«Счетная»</b> – количество посчитанных импульсов.
<b>Настройки</b>	<i>index</i>	uint8	Номер входной линии устройства



Для работы функции, в конфигурации устройства соответствующая входная линия не должна быть отключена.  
 Конфигурация > Входные линии > Использовать как > любое значение кроме «Не используется».

### 8.3.9.2 OUTPUT – выход

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>value</i>	int32	Состояние выходной линии, которое необходимо установить. Логика зависит от настройки линии. <b>«Общего назначения»:</b> «1» - включить (замкнуть на массу) «0» - выключить. <b>«Зуммер» (только OUT_1):</b> Частота (Гц), которую необходимо генерировать на выходе.
<b>Настройки</b>	<i>index</i>	uint8	Номер выходной линии



Для работы функции, в конфигурации устройства соответствующая выходная линия должна быть настроена определенным образом.  
 Конфигурация > Выходные линии > Использовать как > «Общего назначения».  
 Для линии OUT\_1 допустима настройка «Зуммер».



Блок работает в режиме «давящего сигнала». При каждом исполнении блок пытается установить состояние выходной линии, которое задано входным значением.

### 8.3.9.3 HYGRO – гигрометр

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>temperature</i>	float	Температура, °C
	<i>humidity</i>	float	Влажность, %
<b>Настройки</b>	<i>Index</i>	uint8	Номер датчика температуры/влажности для отображения

### 8.3.9.4 ACCEL – акселерометр

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>x</i>	int32	Текущее ускорение по оси акселерометра X
	<i>y</i>	int32	Текущее ускорение по оси акселерометра Y
	<i>z</i>	int32	Текущее ускорение по оси акселерометра Z
	<i>acc sqrt</i>	int32	Квадратный корень из суммы квадратов ускорений по каждой оси
	<i>int sqrt</i>	int32	--
	<i>angle</i>	int32	Угол наклона относительно местной (временной) вертикали
	<i>pitch</i>	int32	Угол тангажа: наклон вперед < 0 наклон назад > 0
	<i>roll</i>	int32	Угол крена: крен влево < 0 крен вправо > 0
	<i>calibrated</i>	bool	Состояние калибровки акселерометра ( <i>true</i> - откалиброван)

### 8.3.9.5 ECODRIVE – EcoDrive

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>speed</i>	int32	Текущее значение скорости
	<i>boost</i>	int32	Текущее значение ускорения (после калибровки)
	<i>retard</i>	int32	Текущее значение торможения (после калибровки)
	<i>drift_right</i>	int32	Текущее значение ускорения вправо (после калибровки)
	<i>drift_left</i>	int32	Текущее значение ускорения влево (после калибровки)
	<i>jump</i>	int32	Текущее значение вертикального ускорения (после калибровки)
	<i>belt</i>	int32	--
	<i>light</i>	int32	--
	<i>prm</i>	int32	--



Для работы функции, в конфигурации устройства должна быть настроена работа с модулем контроля качества вождения.  
Конфигурация > EcoDriving > «Включить контроль качества вождения».

### 8.3.9.6 ONEWIRE\_KEY – Информация о текущем ключе/карте на интерфейсах 1Wire или RS-xxx (ближнего действия)

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>lo</i>	int32	Младшие 4 байта кода
	<i>hi</i>	int32	Старшие 4 байта кода
	<i>valid</i>	bool	Код находится в списке доверенных кодов устройства

### 8.3.9.7 RFID – Информация о текущей метке RFID на интерфейсе RS-xxx (беспроводная, дальнего действия)

	Сигнатура	Тип	Описание
<b>Выходы</b>	<i>lo</i>	int32	Младшие 4 байта кода
	<i>hi</i>	int32	Старшие 4 байта кода
	<i>pwr</i>	int32	Мощность сигнала
	<i>type</i>	int32	--
	<i>valid</i>	bool	Код находится в списке доверенных кодов устройства



Для работы функции, в конфигурации устройства должна быть настроена работа со считывателями RFID меток.  
Конфигурация > RS-232/RS-485 > Устройство X > «Считыватель меток RFID».

### 8.3.9.8 TACHOGRAPH – Tachograph driver

	Сигнатура	Тип	Описание
Выходы	<i>code0_3</i>	int32	0 .. 3 байты кода карты
	<i>code4_7</i>	int32	4 .. 7 байты кода карты
	<i>code8_11</i>	int32	8 .. 11 байты кода карты
	<i>code12_15</i>	int32	12 .. 15 байты кода карты
	<i>state</i>	int32	Состояние водителя
	<i>type</i>	int32	--
	<i>active</i>	bool	--
Настройки	<i>index</i>	uint8	Номер водителя (1-й или 2-ой)



Для работы функции, в конфигурации устройства должна быть настроена работа с тахографом.

Конфигурация > RS-232/RS-485 > Устройство X > «Тахограф».

### 8.3.9.9 GUARD – Режим охраны

	Сигнатура	Тип	Описание
Входы	<i>enable</i>	bool	Постановка/снятие с охраны: «0» – наблюдение «1» – охрана
Выходы	<i>mode</i>	int32	Текущий режим работы: «0» – наблюдение «1» – охрана
	<i>error</i>	int32	Код ошибки при смене режима охраны: «1» – выключен режим охраны в конфигурации устройства; «2» – не истёк таймаут запрета на смену режима; «3» – включён режим: не переходить в режим охраны при включённом зажигании; «4» – устройство уже в данном режиме; «5» – включен режим: не переходить в охрану, если сработал один из охранных датчиков
Настройки	<i>type</i>	uint8	Тип смены режима работы: «По уровню» - при каждом исполнении блок <u>устанавливает</u> режим работы согласно значению входа; «По восходящему фронту» - при каждом исполнении блок <u>переключает</u> режим работы не противоположный если состояние входа изменилось с «0» на «1».
Внутренние	<i>old</i>	bool	Значение <i>enable</i> на предыдущем цикле



Для работы функции, в конфигурации устройства должна быть настроена работа с охранными функциями.


Конфигурация > Режим охраны > «Использовать режим охраны»




Если тип смены режима работы задан «По уровню», то блок работает в режиме «давящего сигнала». При каждом исполнении блок пытается установить состояние режима, которое задано входным значением.

### 8.3.9.10 CRASH\_FILE – Формирование файла ДТП

	Сигнатура	Тип	Описание
Входы	<i>generate</i>	bool	По восходящему фронту сформировать файл ДТП
	<i>unlock</i>	bool	По восходящему фронту снять блокировку от перезаписи
Выходы	<i>active</i>	bool	Файл ДТП формируется. Значение <i>true</i> устанавливается при начале формирования файла, значение <i>false</i> устанавливается когда формирование файла завершено.
	<i>time</i>	int32	Время создания файла в формате UNIX ( <i>0</i> – файл ДТП отсутствует)
	<i>locked</i>	bool	Устанавливается <i>true</i> если файл защищён от перезаписи
Внутренние	<i>generate_old</i>	bool	Значение <i>generate</i> на предыдущем цикле
	<i>unlock_old</i>	bool	Значение <i>unlock</i> на предыдущем цикле

 Для работы функции, в конфигурации устройства должна быть настроена фиксация ДТП. Конфигурация > Акселерометр > Фиксация ДТП > «Включить фиксацию ДТП ...»

 Если на вход **generate** подключена константа и ее значение *True*, то функция работает в режиме «давящего сигнала». При каждом исполнении функция пытается сделать фотоснимок.  
Если на вход **generate** подключена переменная или выход другой функции, то срабатывание происходит при переходе от *False* к *True*.

### 8.3.9.11 PWRSAVE – Управление энергосбережением

	Сигнатура	Тип	Описание
Входы	<i>gsm off</i>	bool	Отключить питание GSM модуля. Если <i>true</i> , то устройство закроет все установленные интернет соединения и отключит питание GSM модуля. Если <i>false</i> , то работа GSM модуля разрешена.
	<i>gnss off</i>	bool	Отключить питание GNSS модуля Если <i>true</i> , то устройство отключит питание навигационного модуля. Если <i>false</i> , то работа навигационного модуля разрешена.
	<i>battery off</i>	bool	Отключить заряд батареи Если <i>true</i> , то устройство отключит зарядку встроенной аккумуляторной батареи (но продолжит питаться от нее). Если <i>false</i> , то зарядка встроенной аккумуляторной батареи производится в штатном режиме.
	<i>periph off</i>	bool	Отключить перефирию. Если <i>true</i> , то устройство отключит питание цифровых интерфейсов, для которых есть возможность отключения. Если <i>false</i> , то работа цифровых интерфейсов разрешена.
	<i>events off</i>	bool	Запрет формирования событий чёрного ящика. Если <i>true</i> , то устройство запретит формирование/запись событий чёрный ящик. Если <i>false</i> , то формирование/запись событий производится в штатном режиме в соответствии с конфигурацией.
Скрытые	<i>sleep</i>	bool	Войти в режим пониженного энергопотребления (ножка предусмотрена для будущего функционала)

 Для работы функции, в конфигурации устройства должен быть настроен режим энергосбережения.

Конфигурация > Системные настройки:  
.. > Включить «Использовать режим энергосбережения»  
.. > Выбрать «... управляется функцией Complex Events»



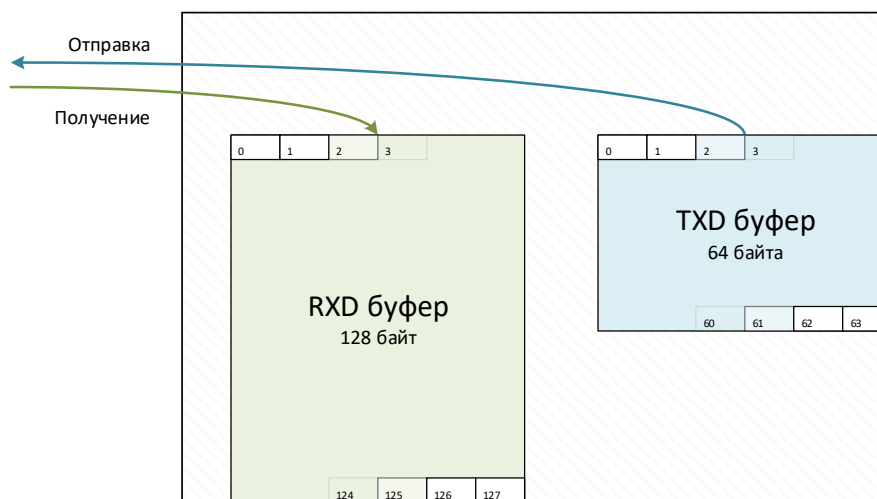
### 8.3.10 Функции доступа к цифровым портам

При работе со всеми цифровыми портами используются два буфера для приема и отправки данных: **RXD** (буфер для приема) и **TXD** (буфер для отправки).

Размеры буферов фиксированы:

- RXD буфер – 128 байт;
- TXD буфер – 64 байта.

В редакторе при отладке буферы изображены в виде массива байт, индексированных с 0 до (размер\_буфера - 1).



Процедуру отправки данных можно разделить на несколько основных этапов:

- Запись данных в TXD буфер;
- Отправка данных из TXD буфера через интерфейс.

#### 8.3.10.1 RS\_SEND – Отправить данные в последовательный порт

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>start</i>	bool	Если <i>true</i> , функция совершает попытку отправки данных.
	<i>send size</i>	int32	Количество байт для отправки через интерфейс.
<b>Выходы</b>	<i>state</i>	int32	Состояние передатчика: «0» – нет активности; «1» – отправка данных; «-1» – интерфейс недоступен (не настроен).
<b>Настройки</b>	<i>port</i>	uint8	Выбор цифрового интерфейса.

Функция выполняет отставку данных через последовательный интерфейс *port*. Для отправки берутся данные из TXD буфера с позиции 0 до (*send\_size* - 1).

**i** Для работы функции, в конфигурации устройства должен быть настроен соответствующий интерфейс.  
 Конфигурация > RS-232/RS-485 > Устройство 1 > «Complex Events (асинхронный режим)».

Процедуру приема данных можно разделить на несколько основных этапов:

- Прием данных из интерфейса в RXD буфер;
- Чтение данных из RXD буфера.

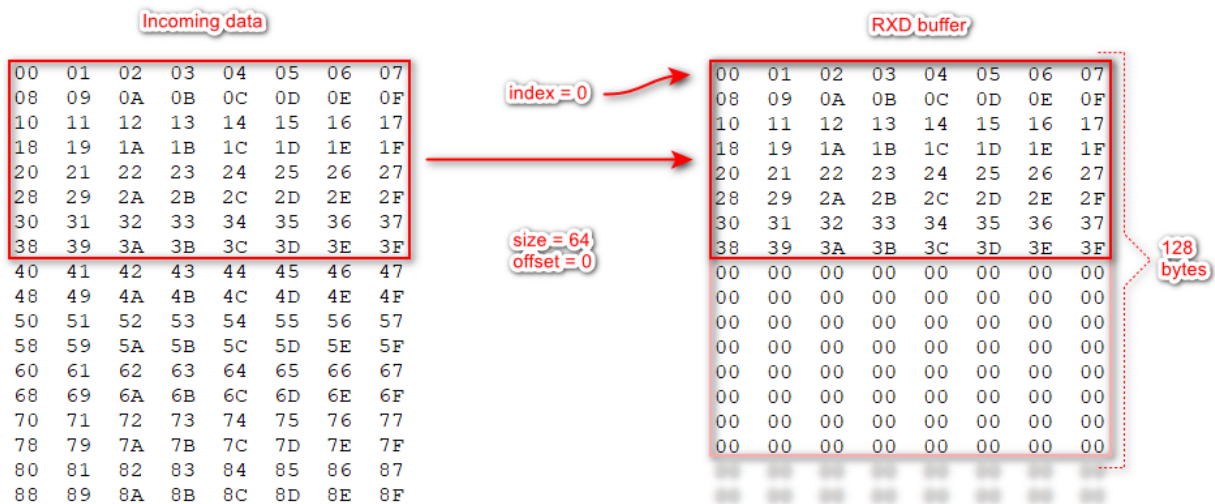
Если при отправке данных процедура крайне проста, то при работе с приемом данных следует учитывать важную особенность обработки данных – устройство способно принять неограниченно большой объем данных, но RXD буфер позволяет хранить не более 128 байт. При этом за один такт исполнения функции приема устройство помещает из интерфейса в RXD буфер не более 64 байт данных.

Если устройство будет выполнять прием данных, объем которых превышает 128 байт, то произойдет переполнение RXD буфера. При переполнении, RXD буфер хранит только последние 128 байт принятых данных.

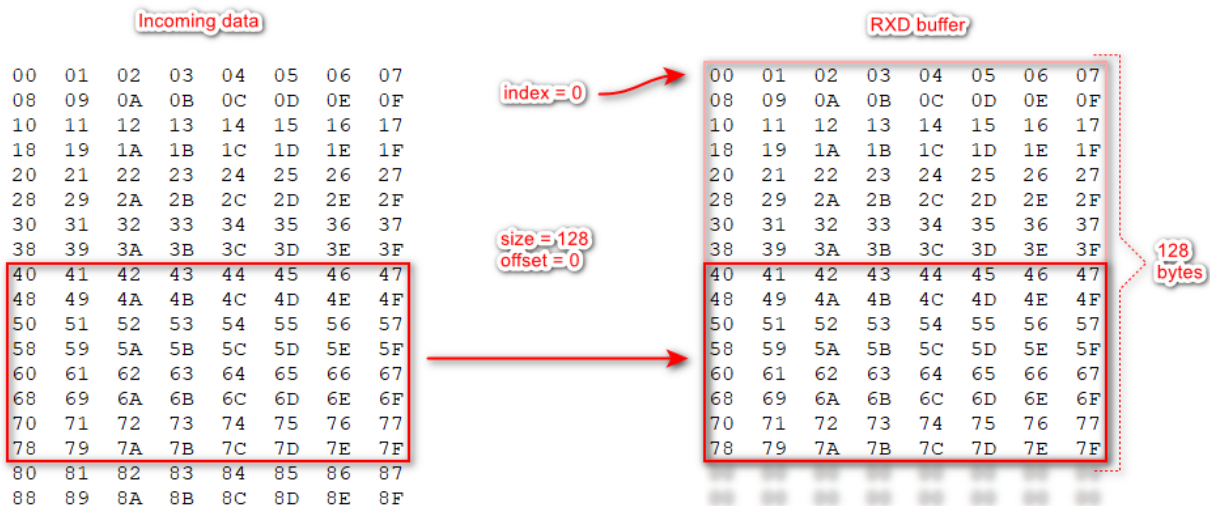
Поэтому, если необходимо обработать данные, объем которых превышает 128 байт следует составить программу таким образом, что после каждого такта работы функции приема данных из интерфейса, выполняется обработка текущего содержимого RXD буфера. Этот подход позволит за несколько итераций обработать весь необходимый объем данных.

Ниже наглядно изображен процесс приема данных, объем которых немного превышает объем RXD буфера:

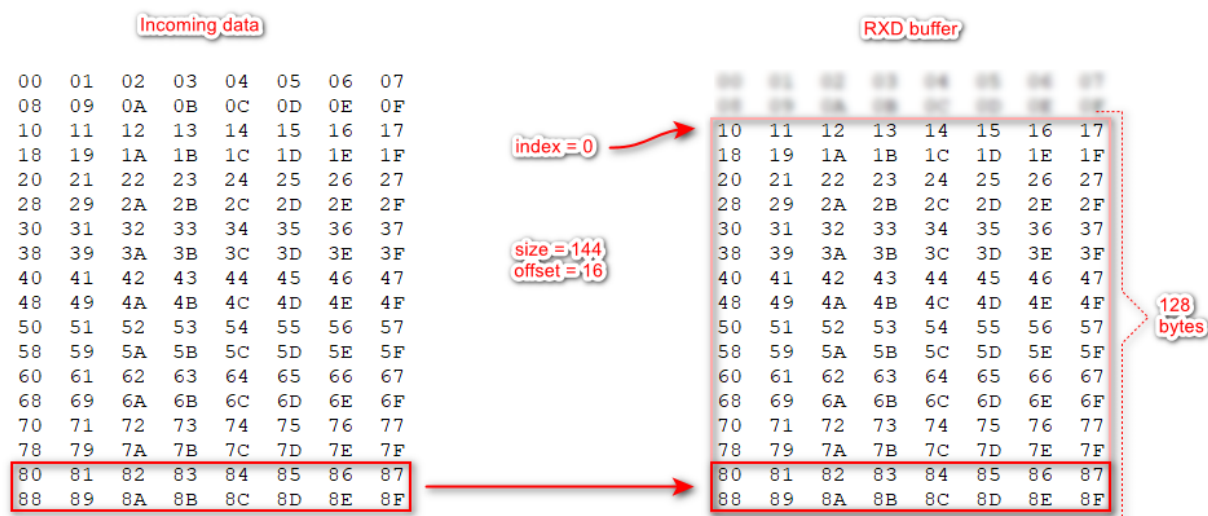
1. Функция приема данных обнаруживает новый входящий поток данных. На первом такте функция принимает 64 байта, увеличивает счетчик принятых данных *size* на 64 и помещает данные в RXD буфер, начиная с индекса 0.



2. На следующем такте функция принимает еще 64 байта, увеличивает счетчик принятых данных *size* на 64 и помещает данные в RXD буфер, начиная с индекса 64.



3. На следующем такте функция принимает оставшиеся данные ( $X$  байт), увеличивает счетчик принятых данных  $size$  на  $X$ . Стирает первые  $X$  байт в RXD буфере. Смещает содержимое RXD буфера на  $X$  байт «влево» (т.е. байт по индексу  $X-1$  теперь будет располагаться по индексу 0). Увеличивает счетчик переполнения  $offset$  на  $X$ . Помещает данные в RXD буфер, начиная с индекса  $(128-X)$ .



### 8.3.10.2 RS\_RECV – Принять данные из последовательного порта

	Сигнатура	Тип	Описание
Входы	<i>enabled</i>	bool	Если <i>true</i> , функция ожидает входящие данные от интерфейса.
	<i>reset</i>	Bool	Если <i>true</i> , функция очистит RXD буфер (все байты будут установлены в 0x00) и следующие данные будут записываться начиная с индекса 0.
Выходы	<i>state</i>	int32	Состояние приемника: «0» - прием отключен; «1» - ожидание данных; «2» - прием данных; «3» - данные приняты; «-1» - интерфейс недоступен (не настроен);
	<i>size</i>	int32	Размер полученного массива данных. Полученные данные сразу помещаются в RXD буфер.
	<i>offset</i>	int32	Количество потерянных данных из-за переполнения RXD буфера (если принято более 128 байт). В буфере всегда находятся последние 128 байт полученных данных.
Настройки	<i>port</i>	uint8	Цифровой интерфейс, которым управляет функция. Если выбранный интерфейс не настроен, то функция сформирует ошибку <i>state = -1</i> .
	<i>timeout</i>	uint16	Время после приема последнего байта, по истечении которого считается, что прием данных завершен <i>state = 3</i> . Следующие данные будут считаться новыми и будут записаны в RXD буфер с индекса 0.

Функция выполняет прием данных через последовательный интерфейс *port*. Когда функция фиксирует начало передачи данных ( $state = 1$ ), то первые полученные байты копируются в RXD буфер, начиная с индекса 0. За один такт работы, функция способна получить из интерфейса 64 байта. Если количество входящих данных больше 64 байт, то процесс получения будет выполнен за несколько тактов ( $state = 2$ ), при этом оставшиеся данные будут дописываться в RXD буфер начиная с индекса 64. Функция зафиксирует конец приема данных ( $state = 3$ ), если после приема последнего байта истек *timeout*. Следующие данные будут считаться новыми и будут записаны в RXD буфер с индекса 0.

**i** Для работы функции, в конфигурации устройства должен быть настроен соответствующий интерфейс.  
 Конфигурация > RS-232/RS-485 > Устройство 1 > «Complex Events (асинхронный режим)».

В качестве частного случая обмена данными предусмотрена функция для выполнения транзакции типа «запрос/ответ». Эту процедуру можно разделить на несколько основных этапов:

- Запись данных в TXD буфер;
- Отправка данных из буфера TXD через интерфейс;
- Прием данных из интерфейса в RXD буфер;
- Чтение данных из RXD буфера.

### 8.3.10.3 RS\_TRANS – Запрос/ответ через последовательный порт

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>start</i>	bool	Если <i>true</i> , функция пытается начать транзакцию.
	<i>send size</i>	int32	Размер массива данных из TXD буфера, для отправки.
	<i>require size</i>	int32	Размер ожидаемого ответа.
<b>Выходы</b>	<i>ended</i>	bool	Сигнал завершения транзакции. Сигнал <u>не</u> устанавливается, если интерфейс не настроен ( <i>state = -1</i> ).
	<i>state</i>	int32	Состояние транзакции: «0» - нет активности; «1» - ожидание доступа к интерфейсу; «2» - доступ к интерфейсу получен; «3» - транзакция в процессе; «4» - транзакция завершена успешно; «-1» - интерфейс недоступен (не настроен); «-2» - истек таймаут ожидания ответа; «-3» - неизвестная ошибка.
	<i>recv size</i>	int32	Размер полученного массива данных. Полученные данные сразу помещаются в RXD буфер.
<b>Настройки</b>	<i>port</i>	uint8	Цифровой интерфейс, которым управляет функция. Если выбранный интерфейс не настроен, то функция сформирует ошибку <i>state = -1</i> .
	<i>timeout</i>	uint16	Время, в течение которого после отправки данных функция ожидает ответ. Если за отведенное время получено количество байт $\leq$ <i>require size</i> , то транзакция завершается с ошибкой <i>state = -2</i> .

Функция выполняет отправку данных через последовательный интерфейс *port*. Для отправки берутся данные из TXD буфера в диапазоне от 0 до (*send size - 1*). Далее функция ожидает ответ в течение времени *timeout* или пока в RXD буфер не поступят данные длиной  $\geq$  *require size*.



Для работы функции, в конфигурации устройства должен быть настроен соответствующий интерфейс.  
 Конфигурация > RS-232/RS-485 > Устройство X > «Complex Events (транзакция)».

Для работы с RXD и TXD буферами используется набор функций, которые позволяют выполнить основные операции чтения/записи и преобразования данных.

#### 8.3.10.4 RXD\_GET – Прочитать значение из RXD буфера



Функция обновлялась. Текущая реализация используется с версии редактора v3.4.1

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>index</i>	int32	Позиция в буфере RXD, начиная с которой необходимо выполнить чтение. Самый первый элемент буфера имеет индекс 0.
	<i>size</i>	int32	Количество байт, которые необходимо прочитать из буфера RXD в каждый выход <i>valueX</i> . Допустимые значения от 1 до 4.
<b>Выходы</b>	<i>value<sub>0</sub></i>	int32/float	Прочитанное значение 0
	<i>value<sub>1</sub></i>	int32/float	Прочитанное значение 1
	...		
	<i>value<sub>N-1</sub></i>	int32/float	Прочитанное значение <i>N-1</i>
<b>Настройки</b>	<i>N</i>	uint8	Количество выходов <i>value</i>
	<i>endian</i>	uint8	Порядок байт, который будет использован при копировании элементов буфера на выход <i>value</i> . Например RXD = [01,02,03,04,05,...], <i>index</i> = 0, <i>size</i> = 4: «Младшим вперед» <i>value</i> = 0x04030201. «Старшим вперед» <i>value</i> = 0x01020304. «Старшим вперед (2 байта)» <i>value</i> = 0x03040102.
	<i>sign</i>	bool	Если флаг установлен, то функция будет воспринимать прочитанные данные как отрицательное число, если старший бит равен «1».

Функция выполняет последовательное чтение буфера RXD для каждого выхода *valueX*. Чтение начинается с индекса *index*. Функция читает *size* байт и передает их на выход *valueX*. Затем индекс чтения смещается на *size*, после чего производится чтение для следующего выхода *valueX*. В результате из буфера будет прочитан диапазон байт от *index* до  $(index+(size*N)-1)$ .

Логика работы функции зависит от типа данных:



- Если к выходу *valueX* подключена переменная с типом **FLOAT** и вход **size = 4**, то функция читает данные из буфера по стандарту IEEE754. Такой способ необходимо использовать для значений, которые хранятся в формате *Float* (например, значение 12.6).

- Иначе функция читает данные как **INT32**.

Конвертация выполняется автоматически при помощи функций **FROM FLOAT** и **TO FLOAT**.

### 8.3.10.5 RXD\_CMP – Поиск данных в RXD буфере

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>index</i>	int32	Позиция в буфере RXD, начиная с которой будет произведен поиск. Самый первый элемент буфера имеет индекс 0.
<b>Выход</b>	<i>result</i>	int32	Результат поиска: «≥0» - Данные <u>найжены</u> , индекс элемента буфера, следующего сразу <u>после</u> найденной последовательности данных. «-1» - Данные <u>не найжены</u> .
<b>Настройки</b>	<i>data</i>	bin	Последовательность для поиска в RXD буфере. Задается в HEX "3120322033" или ASCII "1 2 3".
<b>Внутренние</b>	<i>size</i>	uint8	Размер поля <i>data</i> .

Пример:

Если в RXD = [01,02,03,04,05,06...], *index* = 0, *data* = [0203], то *value* = 3

Если в RXD = [01,02,03,04,05,06...], *index* = 2, *data* = [0203], то *value* = -1

Если в RXD = [01,02,03,04,05,06...], *index* = 0, *data* = [3322], то *value* = -1

### 8.3.10.6 RXD\_STR2INT – Преобразовать строку из RXD буфера в целое число

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>index</i>	int32	Позиция в RXD буфере, на которой расположено INT значение.
<b>Выход</b>	<i>value</i>	int32	Прочитанное значение. Если значение не прочитано, то <i>value</i> = 0

Начиная с позиции *index* функция пытается прочитать INT значение, хранящееся как строка в кодировке ASCII.

Пример:

Буфер RXD = [7a,67,2d,32,2e,36,66...]. В кодировке ASCII это строка "zg-2.6f".

Если *index* = 2, то *value* = -2

Если *index* = 3, то *value* = 2

Если *index* = 4, то *value* = 0

### 8.3.10.7 RXD\_STR2FLOAT – Преобразовать строку из RXD буфера в число с плавающей точкой

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>index</i>	int32	Позиция в RXD буфере, на которой расположено FLOAT значение.
<b>Выход</b>	<i>value</i>	int32	Прочитанное значение. Если значение не прочитано, то <i>value</i> = 0

Начиная с позиции *index* функция пытается прочитать FLOAT значение, хранящееся как строка в кодировке ASCII.

Пример:

Буфер RXD = [7a,67,2d,32,2e,36,66...]. В кодировке ASCII это строка "zg-2.6f".

Если *index* = 2, то *value* = -2.6

Если *index* = 3, то *value* = 2.6

Если *index* = 4, то *value* = 0

### 8.3.10.8 RXD\_CHECKSUM – Проверка контрольной суммы в RXD буфере

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>index</i>	int32	Позиция в RXD буфере, начиная с которой производится расчет.
	<i>size</i>	int32	Длина массива данных для расчета CRC.
	<i>valued index</i>	int32	Позиция в RXD буфере, на которой расположено значение, с которым будет сравниваться рассчитанное CRC.
<b>Выход</b>	<i>valid</i>	bool	Результат проверки CRC.
<b>Настройки</b>	<i>type</i>	uint8	Алгоритм расчета CRC: <b>«CRC-16 (Modbus)»</b> Стандартный алгоритм CRC-16 Modbus. <b>«CRC-8 (Maxim/Dallas)»</b> Стандартный алгоритм CRC-8 Maxim/Dallas. <b>«XOR (8 bits)»</b> Последовательная операция XOR. <b>«Сумма (8 bits)»</b> Последовательное сложение элементов.
	<i>options</i>	uint8	<b>«Порядок байт»</b> - Порядок байт, при сравнении CRC (если расчетное CRC = 0x0201). <b>«Младшим вперед»</b> Будет использовано значение 0x0102. <b>«Старшим вперед»</b> Будет использовано значение 0x0201. <b>«Инвертировать»</b> Если флаг установлен, то перед сравнением CRC будет побитово инвертировано. Например, если было 0x0201, то будет 0xfdfе. <b>«Добавить 1»</b> Если флаг установлен, то перед сравнением CRC будет увеличено на 1. Например, если было 0x0201, то будет 0x0202.

Функция выполняет расчет CRC по RXD буферу, начиная с *index* до (*index+size-1*). Рассчитанное CRC сравнивается со значением, хранящемся в RXD буфере начиная с позиции *value index*.



Операции "Порядок байт", "Инвертировать", "Добавить 1" выполняются после расчета CRC по очереди в порядке перечисления и влияют на финальное значение, используемое при сравнении.



### 8.3.10.9 TXD\_INIT – Инициализация TXD буфера

	Сигнатура	Тип	Описание
<b>Вход</b>	<i>enable</i>	bool	Если <i>true</i> , то выполняется инициализация TXD буфера данными пользователя.
<b>Настройки</b>	<i>data</i>	bin	Последовательность для инициализации TXD буфера. Задается в HEX "3120322033" или ASCII "1 2 3".
<b>Внутренние</b>	<i>size</i>	uint8	Количество байт, которые будут записаны в TXD буфер

Функция заполняет TXD буфер данными, введенными пользователем, начиная с 0 индекса. Если длина последовательности пользователя меньше, чем длина буфера, то оставшиеся ячейки заполняются 0x00.

### 8.3.10.10 TXD\_SET – Запись значения в TXD буфер



Функция обновлялась. Текущая реализация используется с версии редактора v3.4.1

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>enable</i>	bool	Если <i>true</i> , то выполняется запись значения в буфер.
	<i>index</i>	int32	Позиция в TXD буфере, начиная с которой производится запись.
	<i>size</i>	int32	Количество байт, которые будут взяты с входа <i>valueX</i> и записаны в буфер (от 1 до 4 байт).
	<i>value<sub>0</sub></i>	int32/float	Значение 0, которое необходимо записать в буфер.
	<i>value<sub>1</sub></i>	int32/float	Значение 1
	...		
	<i>value<sub>N-1</sub></i>	int32/float	Значение <i>N-1</i>
<b>Настройки</b>	<i>N</i>	uint8	Количество входов <i>value</i>
	<i>endian</i>	uint8	Порядок байт, который будет использован при записи в буфер. Например TXD = [01,02,03,04,05,...], <i>index</i> = 1, <i>size</i> = 4, <i>value</i> = 0x44332211: <b>«Младшим вперед»</b> После записи TXD = [01,11,22,33,44,...] <b>«Старшим вперед»</b> После записи TXD = [01,44,33,22,11,...] <b>«Старшим вперед (2 байта)»</b> После записи TXD = [01,22,11,44,33,...]

Функция выполняет последовательную запись значений *valueX* размером от 1 до 4 байт в TXD буфер начиная с позиции *index*. В отличие от функции инициализации, эта функция затрагивает только байты в диапазоне от *index* до  $(index+(size*N)-1)$ .



Логика работы функции зависит от типа данных:

- Если ко входу **value** подключена **переменная** с типом **FLOAT** и вход **size = 4**, то функция записывает данные по стандарту IEEE754. Такой способ необходимо использовать для значений, которые хранятся в формате *Float* (например, значение 12.016).

- Иначе функция записывает данные как целое число. Такой способ необходимо использовать для значений в форматах *Int* или *Uint* (Например, так следует записать число 43605).

Конвертация выполняется автоматически при помощи функций [FROM FLOAT](#) и [TO FLOAT](#).



### 8.3.10.11 TXD\_CHECKSUM – Записать контрольную сумму в TXD буфер

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>enable</i>	bool	Если <i>true</i> , функция выполняет расчет CRC.
	<i>index</i>	int32	Позиция в TXD буфере, начиная с которой производится расчет.
	<i>size</i>	int32	Длина массива данных для расчета CRC.
	<i>value index</i>	int32	Позиция в TXD буфере, начиная с которой будет записано рассчитанное CRC.
<b>Настройки</b>	<i>type</i>	uint8	Алгоритм расчета CRC: <b>«CRC-16 (Modbus)»</b> Стандартный алгоритм CRC-16 Modbus. <b>«CRC-8 (Maxim/Dallas)»</b> Стандартный алгоритм CRC-8 Maxim/Dallas. <b>«XOR (8 bits)»</b> Последовательная операция XOR. <b>«Сумма (8 bits)»</b> Последовательное сложение элементов.
	<i>options</i>	uint8	<b>«Порядок байт»</b> - Порядок байт, перед записью CRC (если расчетное CRC = 0x0201). <b>«Младшим вперед»</b> Будет записано значение 0x0102. <b>«Старшим вперед»</b> Будет записано значение 0x0201. <b>«Инвертировать»</b> Если флаг установлен, то перед записью CRC будет побитово инвертировано. Например, если было 0x0201, то будет 0xfdfе. <b>«Добавить 1»</b> Если флаг установлен, то перед записью CRC будет увеличено на 1. Например, если было 0x0201, то будет 0x0202.

Функция выполняет расчет CRC по TXD буферу, начиная с *index* до (*index+size-1*). Рассчитанное CRC записывается в TXD буфер начиная с позиции *value index*.



Операции "Порядок байт", "Инвертировать", "Добавить 1" выполняются после расчета CRC по очереди в порядке перечисления и влияют на финальное значение, используемое при записи.

### 8.3.10.12 TXD\_GET – Прочитать значение из TXD буфера



Функция обновлялась. Текущая реализация используется с версии редактора v3.4.1

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>index</i>	int32	Позиция в буфере TXD, начиная с которой необходимо выполнить чтение. Самый первый элемент буфера имеет индекс 0.
	<i>size</i>	int32	Количество байт, которые необходимо прочитать из буфера TXD. Допустимые значения от 1 до 4.
<b>Выходы</b>	<i>value</i>	int32   float	Прочитанное значение.
<b>Настройки</b>	<i>endian</i>	uint8	Порядок байт, который будет использован при копировании элементов буфера на выход <i>value</i> . Например RXD = [01,02,03,04,05,...], <i>index</i> = 0, <i>size</i> = 4: «Младшим вперед» <i>value</i> = 0x04030201. «Старшим вперед» <i>value</i> = 0x01020304. «Старшим вперед (2 байта)» <i>value</i> = 0x03040102.
	<i>sign</i>	bool	Если флаг установлен, то функция будет воспринимать прочитанные данные как отрицательное число, если старший бит равен «1».



Логика работы функции зависит от типа данных:

- Если к выводу **value** подключена **переменная** с типом **FLOAT** и вход **size = 4**, то функция читает данные из буфера по стандарту IEEE754. Такой способ необходимо использовать для значений, которые хранятся в формате *Float* (например, значение 12.6).

- Иначе функция читает данные как **INT32**.

Конвертация выполняется автоматически при помощи функций **FROM FLOAT** и **TO FLOAT**.

Для удобства приема и отправки данных по протоколу ModBus предусмотрены специальные функции MODBUS\_READ и MODBUS\_WRITE, которые фактически являются доработанными вариантами RS\_TRANS. Процесс обмена данными значительно упрощается относительно универсальных функций обмена данными, т.к. функция сама составляет запрос/команду, сама контролирует получение ответа, и сама производит разбор данных.

### 8.3.10.13 MODBUS\_READ – Чтение данных по протоколу Modbus RTU

	Сигнатура	Тип	Описание
<b>Входы</b>	<i>enable</i>	bool	Отправка запросов разрешена
<b>Выходы</b>	<i>valid</i>	bool	<i>True</i> , если на последний запрос был получен корректный ответ и значения на выходах <i>valueX</i> актуальны
	<i>state</i>	int32	Состояние: «0» – не активен «1» – ожидание доступа к интерфейсу «2» – доступ к интерфейсу получен «3» – транзакция в процессе «4» – транзакция завершена успешно «-1» – интерфейс недоступен (не настроен) «-2» – истек таймаут ожидания ответа «-3» – неизвестная ошибка
	<i>value<sub>0</sub></i>	int32/float/bool	Последнее прочитанное значение 0.
	<i>value<sub>1</sub></i>	int32/float/bool	Последнее прочитанное значение 1
	...		
	<i>value<sub>N-1</sub></i>	int32/float/bool	Последнее прочитанное значение <i>N-1</i>
<b>Настройки</b>	<i>N</i>	uint8	Количество выходов <i>value</i>
	<i>port</i>	uint8	Цифровой интерфейс, которым управляет функция. Если выбранный интерфейс не настроен, то функция сформирует ошибку <i>state = -1</i> .
	<i>period</i>	uint16	Период повторной отправки запроса если на входе <i>enable</i> удерживается значение <i>true</i> . Повтор выполняется как в случае ошибки, так и в случае успешного завершения транзакции.
	<i>timeout</i>	uint16	Время, в течение которого после отправки данных функция ожидает ответ. Если корректный ответ не получен, то транзакция завершается с ошибкой <i>state = -2</i> .
	<i>function</i>	uint8	Функция ModBus, с помощью которой считываются данные
	<i>number</i>	uint8	Сетевой номер, опрашиваемого датчика
	<i>address</i>	uint16	Адрес запрашиваемых данных
	<i>type</i>	uint8	Тип параметра для чтения: <b>uint8</b> – однобайтовое число без знака; <b>int8</b> – однобайтовое число со знаком; <b>uint16</b> – двухбайтовое без знака; <b>int16</b> – двухбайтовое со знаком; <b>int32/float</b> – четырехбайтовое со знаком / вещественное.
<i>endian</i>	uint8	Порядок байт, который будет использован при копировании полученных данных на выход <i>value</i> . Например, данные = [01,02,03,04], <i>type</i> = int32: <b>«Младшим вперед»</b> <i>value</i> = 0x04030201. <b>«Старшим вперед»</b> <i>value</i> = 0x01020304. <b>«Старшим вперед (2 байта)»</b> <i>value</i> = 0x03040102.	
<b>Внутренние</b>	<i>count</i>	int32	Внутренний счётчик таймаута

Например, настроим функцию так:

Параметр	Значение
<i>N</i>	3
<i>port</i>	RS-485
<i>period</i>	1000 ms
<i>timeout</i>	100 ms
<i>function</i>	(03) Чтение регистров ввода
<i>number</i>	17
<i>address</i>	107 (0x6B)
<i>type</i>	int16
<i>endian</i>	Старшим вперед

Примеры сформированного запроса и ожидаемого ответа:

Запрос		Ответ	
Значение (HEX)	Название поля ModBus	Значение (HEX)	Название поля ModBus
11	Сетевой номер датчика	11	Сетевой номер датчика
03	Функция ModBus	03	Функция ModBus
00	Адрес первого регистра (Hi байт)	06	Количество байт данных
6B	Адрес первого регистра (Lo байт)	AE	Значение регистра 0x006B (Hi байт)
00	Количество регистров (Hi байт)	41	Значение регистра 0x006B (Lo байт)
03	Количество регистров (Lo байт)	56	Значение регистра 0x006C (Hi байт)
76	CRC (Hi байт)	52	Значение регистра 0x006C (Lo байт)
87	CRC (Lo байт)	43	Значение регистра 0x006D (Hi байт)
		40	Значение регистра 0x006D (Lo байт)
		49	CRC (Hi байт)
		AD	CRC (Lo байт)

Устройство сформирует запрос и попытается выполнить его отправку через интерфейс RS-485. После отправки, устройство в течение 100 ms будет ожидать ответ.

После получения данных устройство проверит формат пакета на соответствие протоколу ModBus, проверит ожидаемую функцию и контрольную сумму. Если все проверки пройдены, то на выходах будут следующие значения:

*valid* = true

*value0* = 0xAE41

*value1* = 0x5652

*value2* = 0x4340

Если ответ за отведенное время не получен, то на выходах *valueX* останутся предыдущие значения, на выход *valid* примет значение false.

Если на входе *enable* останется значение true, то через 1000 ms после начала предыдущей транзакции функция повторит отправку запроса и разбор ответа.



При работе функции используются универсальные буферы [RXD](#) и [TXD](#)



Для работы функции, в конфигурации устройства должен быть настроен соответствующий интерфейс.

Конфигурация > RS-232/RS-485 > Устройство X > «Complex Events (транзакция)».



Логика работы функции зависит от типа данных:

- Если к выходу **value** подключена **переменная** с типом **FLOAT** и **type = int32/float**, то функция читает данные из буфера по стандарту IEEE754. Такой способ необходимо использовать для значений, которые хранятся в формате Float (например, значение 12.6).

- Иначе функция читает данные как INT32.

Конвертация выполняется автоматически при помощи функций [FROM FLOAT](#) и [TO FLOAT](#).

### 8.3.10.14 MODBUS\_WRITE – Запись данных по протоколу Modbus RTU

	Сигнатура	Тип	Описание
Входы	<i>enable</i>	bool	Отправка команд разрешена
	<i>value<sub>0</sub></i>	int32/float/bool	Записываемое значение 0.
	<i>value<sub>1</sub></i>	int32/float/bool	Записываемое значение 1
	...		
	<i>value<sub>N-1</sub></i>	int32/float/bool	Записываемое значение <i>N-1</i>
Выходы	<i>actual</i>	bool	<i>True</i> , если на последнюю команду был получен корректный ответ и значения на входах <i>valueX</i> успешно записаны
	<i>state</i>	int32	Состояние: «0» – не активен «1» – ожидание доступа к интерфейсу «2» – доступ к интерфейсу получен «3» – транзакция в процессе «4» – транзакция завершена успешно «-1» – интерфейс недоступен (не настроен) «-2» – истек таймаут ожидания ответа «-3» – неизвестная ошибка
Настройки	<i>N</i>	uint8	Количество входов <i>value</i>
	<i>port</i>	uint8	Цифровой интерфейс, которым управляет функция. Если выбранный интерфейс не настроен, то функция сформирует ошибку <i>state = -1</i> .
	<i>period</i>	uint16	Период повторной отправки команды если на входе <i>enable</i> удерживается значение <i>true</i> . Повтор выполняется как в случае ошибки, так и в случае успешного завершения транзакции.
	<i>timeout</i>	uint16	Время, в течение которого после отправки данных функция ожидает ответ. Если корректный ответ не получен, то транзакция завершается с ошибкой <i>state = -2</i> .
	<i>function</i>	uint8	Функция ModBus, с помощью которой записываются данные
	<i>number</i>	uint8	Сетевой номер, опрашиваемого датчика
	<i>address</i>	uint16	Адрес для записи данных
	<i>type</i>	uint8	Тип параметра для записи: <b>uint8</b> – однобайтовое число без знака; <b>int8</b> – однобайтовое число со знаком; <b>uint16</b> – двухбайтовое без знака; <b>int16</b> – двухбайтовое со знаком; <b>int32/float</b> – четырехбайтовое со знаком / вещественное.
<i>endian</i>	uint8	Порядок байт, который будет использован при копировании значений со входов <i>valueX</i> в тело команды. Например, значение = 0x01020304, <i>type</i> = int32/float: <b>«Младшим вперед»</b> <i>TXD</i> = [04,03,02,01] <b>«Старшим вперед»</b> <i>TXD</i> = [01,02,03,04] <b>«Старшим вперед (2 байта)»</b> <i>TXD</i> = [03,04,01,02]	
Внутренние	<i>count</i>	int32	Внутренний счётчик таймаута

Например, настроим функцию так:

Параметр	Значение	Вход	Значение
<i>N</i>	1	<i>value0</i>	3
<i>port</i>	RS-485		
<i>period</i>	1000 ms		
<i>timeout</i>	100 ms		
<i>function</i>	(06) Запись одного регистра хранения		
<i>number</i>	17		
<i>address</i>	1 (0x01)		
<i>type</i>	uint8		
<i>endian</i>	Старшим вперед		

Примеры сформированной команды и ожидаемого ответа:

Команда		Ответ	
Значение (HEX)	Название поля ModBus	Значение (HEX)	Название поля ModBus
11	Сетевой номер датчика	11	Сетевой номер датчика
06	Функция ModBus	06	Функция ModBus
00	Адрес первого регистра (Hi байт)	00	Адрес первого регистра (Hi байт)
01	Адрес первого регистра (Lo байт)	01	Адрес первого регистра (Lo байт)
00	Устанавливаемое значение (Hi байт)	00	Установленное значение (Hi байт)
03	Устанавливаемое значение (Lo байт)	03	Установленное значение (Lo байт)
76	CRC (Hi байт)	76	CRC (Hi байт)
87	CRC (Lo байт)	87	CRC (Lo байт)



При отправке команды на установку одного регистра хранения в ответ ожидается эхо

Устройство сформирует команду и попытается выполнить ее отправку через интерфейс RS-485. После отправки, устройство в течение 100 ms будет ожидать ответ.

После получения данных устройство проверит формат пакета на соответствие протоколу ModBus, проверит ожидаемую функцию и контрольную сумму. Если все проверки пройдены, то на выходе *actual* будет установлено значение true.

Если ответ за отведенное время не получен, то выход *actual* примет значение false.

Если на входе *enable* останется значение true, то через 1000 ms после начала предыдущей транзакции функция повторит отправку команды и разбор ответа.



При работе функции используются универсальные буферы *RXD* и *TXD*



Для работы функции, в конфигурации устройства должен быть настроен соответствующий интерфейс.

Конфигурация > RS-232/RS-485 > Устройство X > «Complex Events (транзакция)».



Логика работы функции зависит от типа данных:

- Если ко входу **value** подключена **переменная** с типом **FLOAT** и **type = int32/float**, то функция записывает данные по стандарту IEEE754. Такой способ необходимо использовать для значений, которые хранятся в формате Float (например, значение 12.016).

- Иначе функция записывает данные как целое число. Такой способ необходимо использовать для значений в форматах *Int* или *Uint* (Например, так следует записать число 43605).

Конвертация выполняется автоматически при помощи функций **FROM FLOAT** и **TO FLOAT**.