ЦИФРОВЫЕ РЕШЕНИЯ

Библиотека Util

Руководство пользователя

06.2026
версия 1.7

Содержание

1 Цель документа

Библиотека Util содержит функциональные блоки, функции арифметических и логических операций, ПИД-регулятор, генератор импульсов и пр. Предназначена для ускорения и упрощения разработки, за счет использования готовых функций и блоков.

Данный документ содержит описание алгоритмов работы блоков, входные/выходные параметры и типы данных каждого блока, и предназначен для ознакомления с составом библиотеки.

Для полного понимания принципов работы в среде необходимо ознакомиться с руководством пользователя ALTA IDE.

2 Установка библиотеки

Чтобы подключить библиотеку к проекту:

1. Откройте редактор **Менеджер библиотек** одним из способов:

- дважды нажмите ЛКМ на системную папку **Менеджер библиотек** в дереве проекта;
- или нажмите ПКМ на системную папку **Менеджер библиотек** в дереве проекта и выберите в контекстном меню пункт **Открыть**:

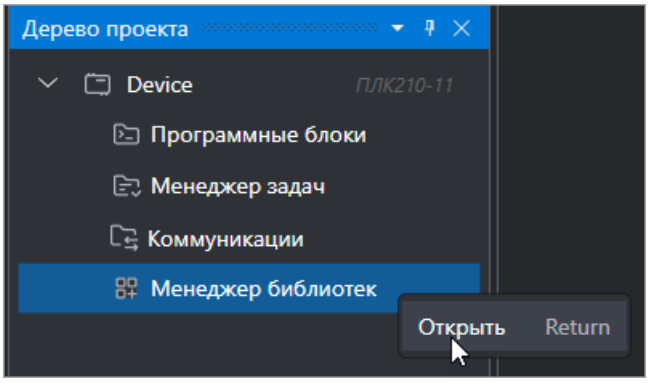

Рисунок 2.1

- В редакторе **Менеджер библиотек** откройте вкладку **Магазин**.
- Выберите библиотеку **Util** и нажмите кнопку **Подключить** на карточке библиотеки.
- Библиотека добавится в проект и отобразится на вкладке **Мои библиотеки** в редакторе **Менеджер библиотек**.

3 Описание библиотеки Util

Библиотека Util содержит:

- мониторы аналоговых сигналов;
- BCD преобразователи;
- битовые/байтовые функции и ФБ;
- регуляторы;
- типы данных;
- манипуляторы функций;
- математические функции;
- сигналы;
- версия библиотеки Util.

3.1 Мониторы аналоговых сигналов

3.1.1 HYSTERESIS

Функциональный блок HYSTERESIS — гистерезис.

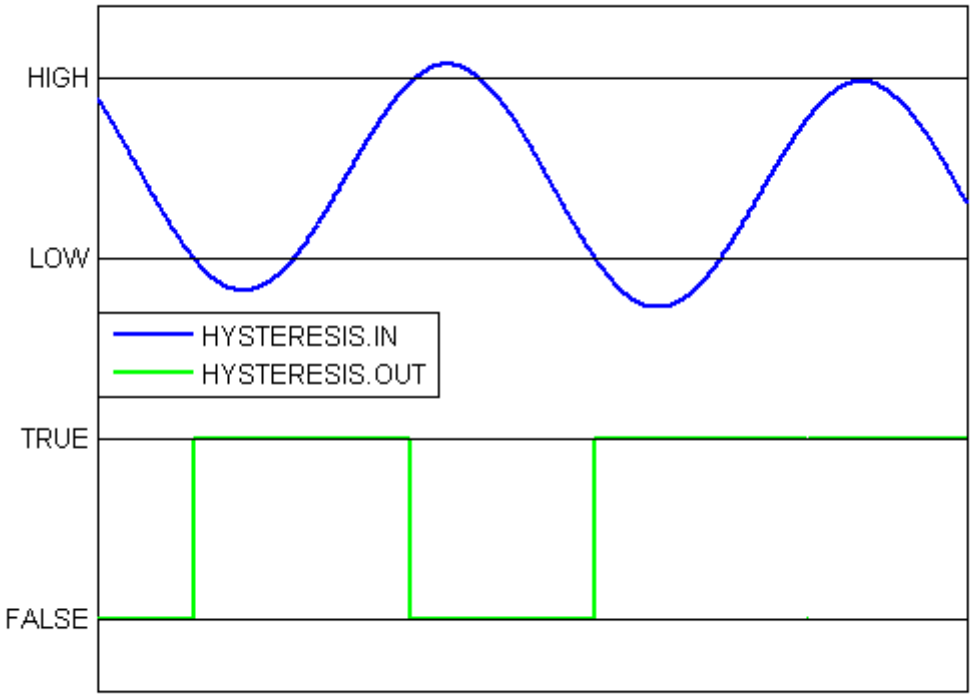

Рисунок 3.1

Имя переменной	Тип данных	Описание
Входные переменные		
IN	INT	Входное значение
HIGH	INT	Верхняя граница
LOW	INT	Нижняя граница
Выходные переменные		
OUT	BOOL	Выход блока

ФБ HYSTERESIS реализует функцию гистерезиса. Выход **OUT** становится TRUE, если входное значение **IN** меньше **LOW**. FALSE, если входное значение **IN** больше **HIGH**.

Пример:

```
VAR
    fbHys:    util.HYSTERESIS;
    aiValue:  INT := 50; //входное аналоговое значение
    alarmHys: BOOL;      //результат ФБ
END_VAR

fbHys (IN := aiValue, LOW := 40, HIGH := 60);
alarmHys := fbHys.OUT;
```

3.1.2 LIMITALARM

Функциональный блок LIMITALARM сигнализирует о нахождении входного значения в диапазоне нижней и верхней границы.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	INT	Входное значение
HIGH	INT	Верхняя граница
LOW	INT	Нижняя граница
Выходные переменные		
O	BOOL	Выход блока TRUE: Входное значение **IN** больше **HIGH**. FALSE: остальное
U	BOOL	Выход блока TRUE: Входное значение **IN** меньше **LOW**. FALSE: остальное
IL	BOOL	Выход блока TRUE: "**O**"; FALSE: "**U**"

Пример:

```
VAR
    fbLim:     util.LIMITALARM;
    aiValue:   INT;        //входное значение
    hiLim:     INT;        //верхняя граница
    loLim:     INT;        //нижняя граница
    alarmHigh: BOOL;       // TRUE, если IN > HIGH (выход O)
    alarmLow:  BOOL;       // TRUE, если IN < LOW (выход U)
    inRange:   BOOL;       // TRUE, если значение в диапазоне (выход IL)
END_VAR

fbLim (IN := aiValue, HIGH := hiLim, LOW := loLim);

alarmHigh := fbLim.O;    // превышение верхнего предела
alarmLow :=  fbLim.U;    // ниже нижнего предела
inRange :=   fbLim.IL;   // в пределах диапазона
```

3.2 BCD преобразователи

BYTE в формате BCD содержит целые числа в диапазоне от 0 до 99. Для каждого десятичного разряда используются четыре бита. Разряд десятков хранится в битах 4–7. Таким образом, формат BCD похож на шестнадцатеричное представление, с той простой разницей, что в BCD-байте могут храниться только значения от 0 до 99, тогда как шестнадцатеричный байт принимает значения от 0 до FF.

Пример:

```
Целое число 51 необходимо преобразовать в формат BCD:
5 в двоичном коде — 0101, 1 в двоичном коде — 0001,в результате получается
BYTE 01010001 в BCD-коде, что соответствует значению 51 = 81.
```

3.2.1 BCD_TO_INT

Функция BCD_TO_INT — преобразует один байт в формате BCD в значение типа INT.

Имя	Тип данных	Описание
Входные переменные		
B	BYTE	Значение в BCD-коде. Если значение не соответствует формату BCD, то выходное значение равно -1.
Выходные переменные		
BCD_TO_INT	INT	Значение типа INT для входного параметра **B**

Пример:

```
i := util.BCD_TO_INT(73);  // результат 49
k := util.BCD_TO_INT(151); // результат 97
l := util.BCD_TO_INT(15);  // выход: -1, т.к. значение не соответствует формату BCD*
```

3.2.2 INT_TO_BCD

Функция INT_TO_BCD — преобразует значение типа INT в байт в формате BCD.

Имя	Тип данных	Описание
Входные переменные		
I	INT	Значение типа INT. Некорректное значение (<0 или >99) приводит к возврату 255
Выходные переменные		
INT_TO_BCD	BYTE	Значение формата BCD для входного параметра **I**

Пример:

```
i := util.INT_TO_BCD(49);   // результат 73
k := util.INT_TO_BCD(97);   // результат 151
l := util.INT_TO_BCD(100);  // Ошибка! Выход: 255
```

3.3 Битовые/байтовые функции и ФБ

3.3.1 EXTRACT

Функция EXTRACT — возвращает бит с номером **N** значения **X**.

Имя	Тип данных	Описание
Входные переменные		
X	DWORD	Входное значение
N	BYTE	Номер извлекаемого бита. Отсчёт начинается с 0
Выходные переменные		
EXTRACT	BOOL	**N**-й бит значения **X**

Пример:

```
FLAG := util.EXTRACT(X:=81, N:=4);   (*Результат: TRUE, так как 81 в двоичном
виде 1010001, поэтому бит "4" равен 1*)
FLAG := util.EXTRACT(X:=33, N:=0);   (*Результат: TRUE, так как 33 в двоичном
виде 100001, поэтому бит "0" равен 1*)
```

3.3.2 PACK

Функция PACK — упаковывает биты B0…B7 в один байт.

Имя	Тип данных	Описание
Входные переменные		
B0	BOOL	Входной бит 0
B1	BOOL	Входной бит 1
B2	BOOL	Входной бит 2
B3	BOOL	Входной бит 3
B4	BOOL	Входной бит 4
B5	BOOL	Входной бит 5
B6	BOOL	Входной бит 6
B7	BOOL	Входной бит 7
Выходные переменные		
PACK	BYTE	Значение, сформированное из входов B0..B7

Пример:

```
VAR
   b0: BOOL := TRUE;    // бит 0
   b1: BOOL := FALSE;   // бит 1
   b2: BOOL := TRUE;    // бит 2
   b3: BOOL := FALSE;   // бит 3
   b4: BOOL := FALSE;   // бит 4
   b5: BOOL := FALSE;   // бит 5
   b6: BOOL := FALSE;   // бит 6
   b7: BOOL := TRUE;    // бит 7

   packed: BYTE;        // упакованный байт
END_VAR

packed := util.PACK(B0 := b0, B1 := b1, B2 := b2, B3 := b3, B4 := b4, B5 := b5,
B6 := b6, B7 := b7);   // packed = 16#85 (1000_0101)
```

3.3.3 PUTBIT

Функция PUTBIT — устанавливает значение указанного бита в переменной типа DWORD.

Имя	Тип данных	Описание
Входные переменные		
X	DWORD	Изменяемое значение
N	BYTE	Позиция изменяемого бита, начиная с 0
B	BOOL	Значение задаваемого бита
Выходные переменные		
PUTBIT	DWORD	Значение с изменённым битом

Пример:

```
Var1 := 38;                          // в двоичном виде: 100110
Var2 := util.PUTBIT(Var1, 4, TRUE);  // Результат: 54 = 2#110110
Var3 := util.PUTBIT(Var1, 1, FALSE); // Результат: 36 = 2#100100
```

3.3.4 UNPACK

Функциональный блок UNPACK — преобразует один байт в 8 бит.

Имя	Тип данных	Описание
Входные переменные		
B	BYTE	Входное значение
Выходные переменные		
B0	BOOL	Выходной бит 0
B1	BOOL	Выходной бит 1
B2	BOOL	Выходной бит 2
B3	BOOL	Выходной бит 3
B4	BOOL	Выходной бит 4
B5	BOOL	Выходной бит 5
B6	BOOL	Выходной бит 6
B7	BOOL	Выходной бит 7

Пример:

```
VAR
   fbUnpack: util.UNPACK;
   Value:    BYTE := 16#85;     // 1000_0101
   b0, b1, b2, b3, b4, b5, b6, b7: BOOL;
END_VAR

fbUnpack(B := Value);

b0 := fbUnpack.B0;              // TRUE
b1 := fbUnpack.B1;              // FALSE
b7 := fbUnpack.B7;              // TRUE
```

3.4 Регуляторы

3.4.1 PD

Функциональный блок PD — PD регулятор.

Имя переменной	Тип данных	Описание
Входные переменные		
ACTUAL	REAL	Фактическое значение, измеренная величина
SET_POINT	REAL	Заданное значение, уставка
KP	REAL	Коэффициент пропорциональности **P**
TV	REAL	Постоянная времени дифференцирования **D** [сек]. Если задана как 0, то работает как P-регулятор
Y_MANUAL	REAL	**Y** устанавливается в это значение при **MANUAL** = TRUE

Имя переменной	Тип данных	Описание
Y_OFFSET	REAL	Смещение (offset) регулирующего воздействия
Y_MIN	REAL	Минимальное значение регулирующего воздействия. Если ограничение не требуется, должно быть установлено в 0
Y_MAX	REAL	Максимальное значение регулирующего воздействия. Если ограничение не требуется, должно быть установлено в 0
MANUAL	BOOL	TRUE: ручной режим, **Y** не изменяется регулятором; FALSE: **Y** формируется регулятором
RESET	BOOL	TRUE: устанавливает **Y** в значение **Y_OFFSET**
Выходные переменные		
Y	REAL	Управляющее воздействие
LIMITS_ACTIVE	BOOL	TRUE: **Y** превысило заданные пределы **Y_MIN**, **Y_MAX** и ограничено этими значениями

Представляет PD-регулятор.

PD-регулятор непрерывно вычисляет значение ошибки **e (t)** как разность между заданным значением (уставкой) и измеренной величиной. PD-регулятор формирует корректирующее воздействие на основе пропорциональной и дифференциальной составляющих (обычно обозначаемых **P** и **D** соответственно), что и определяет тип регулятора.

Параметры **Y_OFFSET**, **Y_MIN** и **Y_MAX** используются для преобразования регулирующего воздействия y в заданный диапазон. **MANUAL** может использоваться для переключения на ручной режим, **RESET** — для сброса регулятора. В нормальном режиме работы (**MANUAL**= FALSE и **RESET** = FALSE и **LIMITS_ACTIVE** = FALSE) регулятор вычисляет ошибку **e (t)** как разность **SET_POINT** — **ACTUAL**, формирует производную по времени $\frac{\delta e}{\delta t}$ и сохраняет эти значения внутренне.

Выход, то есть управляющее воздействие **Y**, вычисляется следующим образом:

$$Y = KP \cdot (e + TV\frac{\delta e}{\delta t}) + Y_{OFFSET}$$

Таким образом, помимо пропорциональной составляющей (**P**-части), на **Y** влияет и текущая скорость изменения ошибки (**D**-часть). Дополнительно **Y** ограничивается диапазоном, заданным **Y_MIN** и **Y_MAX**. Если **Y** выходит за эти пределы, **LIMITS_ACTIVE** устанавливается в TRUE. Если ограничение регулирующего воздействия не требуется, то **Y_MIN** и **Y_MAX** должны быть установлены в 0. При **MANUAL** = TRUE в **Y** записывается значение **Y_MANUAL**. PD-регулятор можно легко преобразовать в P-регулятор, установив **TV** = 0.

Для получения дополнительной информации см.: PID.

Пример:

```
VAR
    fbPD:     util.PD;
    actual:   REAL := 45.0;   // измеренное значение
    setPoint: REAL := 50.0;   // уставка
    yOut:     REAL;           // выход регулятора
    limits:   BOOL;           // признак ограничения
END_VAR

fbPD(
ACTUAL    := actual,
SET_POINT := setPoint,
KP        := 2.0,
TV        := 12.0,
Y_OFFSET  := 0.0,
Y_MIN     := 0.0,         // нижнее ограничение выхода
Y_MAX     := 100.0,       // верхнее ограничение выхода
MANUAL    := FALSE,
RESET     := FALSE
);

yOut      := fbPD.Y;
limits    := fbPD.LIMITS_ACTIVE;
```

3.4.2 PID

Функциональный блок PID — ПИД-регулятор.

Имя переменной	Тип данных	Описание
Входные переменные		
ACTUAL	REAL	Фактическое значение, измеренная величина
SET_POINT	REAL	Заданное значение, уставка
KP	REAL	Коэффициент пропорциональности **P**
TN	REAL	Время интегрирования I [сек]. **TN** > 0, иначе регулятор не вычисляется
TV	REAL	Постоянная времени дифференцирования **D** [сек]. Если задана как 0, то работает как P-регулятор
Y_MANUAL	REAL	Y устанавливается в это значение при **MANUAL** = TRUE
Y_OFFSET	REAL	Смещение (offset) регулирующего воздействия
Y_MIN	REAL	Минимальное значение регулирующего воздействия
Y_MAX	REAL	Максимальное значение регулирующего воздействия
MANUAL	BOOL	TRUE: ручной режим, **Y** не изменяется регулятором; FALSE: **Y** формируется регулятором
RESET	BOOL	TRUE: устанавливает **Y** в значение **Y_OFFSET** и сбрасывает интегральную часть
Выходные переменные		
Y	REAL	Управляющее воздействие

LIMITS_ACTIVE	BOOL	TRUE: **Y** превысило заданные пределы **Y_MIN**, **Y_MAX** и ограничено этими значениями
OVERFLOW	BOOL	Переполнение в интегральной части

Представляет ПИД-регулятор

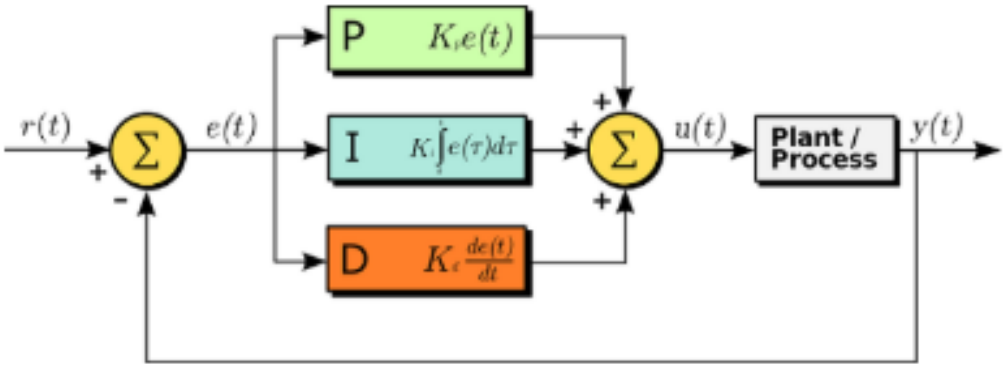

Рисунок 3.2

ПИД-регулятор непрерывно вычисляет значение ошибки **e (t)** как разность между заданным значением (уставкой) и измеренной технологической величиной. ПИД-регулятор формирует корректирующее воздействие на основе пропорциональной, интегральной и дифференциальной составляющих (обычно обозначаемых **P**, **I** и **D** соответственно), что и определяет тип регулятора.

- **P** учитывает текущее значение ошибки. Например, если ошибка велика и положительна, выход регулятора также будет большим и положительным.
- **I** учитывает прошлые значения ошибки. Например, если текущий выход недостаточно велик, интеграл ошибки будет накапливаться со временем, и регулятор отреагирует увеличением воздействия.
- **D** учитывает возможные будущие тенденции ошибки на основе текущей скорости её изменения.

Поскольку PID-регулятор опирается только на измеряемую технологическую величину и не требует знания модели объекта, он широко применим. Настройкой трёх параметров модели PID-регулятор можно адаптировать под требования конкретного процесса. Реакцию регулятора можно описать через скорость отклика на ошибку, степень перерегулирования относительно уставки и выраженность колебаний системы. Использование PID-алгоритма не гарантирует оптимальное управление системой или даже её устойчивость.

Y_OFFSET, **Y_MIN** и **Y_MAX** используются для преобразования регулирующего воздействия **Y** в заданный диапазон. **MANUAL** может использоваться для переключения на ручной режим, **RESET** — для повторной инициализации регулятора. В нормальном режиме работы (**MANUAL** = **RESET** = **LIMITS_ACTIVE** = FALSE) регулятор вычисляет ошибку **e (t)** как разность **SET_POINT** — **ACTUAL**, формирует производную по времени $\frac{\delta e}{\delta t}$ и сохраняет эти значения внутренне.

Выход, то есть управляющее воздействие **Y**, в отличие от PD-регулятора содержит дополнительную интегральную составляющую и вычисляется следующим образом:

$$Y = KP \cdot (e + \frac{1}{TN}\int edt + TV\frac{\delta e}{\delta t}) + Y_{OFFSET}$$

Таким образом, помимо пропорциональной части (**P**), на **Y** влияют и текущая скорость изменения ошибки (**D**), и история ошибки (**I**). PID-регулятор можно легко преобразовать в PI-регулятор, установив **TV** = 0. Из-за наличия интегральной части при некорректной параметризации возможно переполнение, если интеграл ошибки становится слишком большим. Для обеспечения безопасности предусмотрен логический выход **OVERFLOW**, который в этом случае принимает значение TRUE. Обычно это происходит, если система управления неустойчива из-за неверной параметризации. При этом регулятор будет остановлен и сможет быть активирован снова только после повторной инициализации.

ПРИМЕЧАНИЕ
Пока активны ограничения регулирующего воздействия (Y_MIN, Y_MAX), интегральная часть будет подстраиваться так, как если бы история входных значений автоматически влияла на ограниченный выход. Если такое поведение нежелательно, возможен следующий обходной вариант: отключить ограничения в PID-регуляторе (Y_MIN>= Y_MAX) и вместо этого применить оператор LIMIT (стандарт IEC) к выходному значению Y.

ПРИМЕЧАНИЕ
При изменении времени цикла не требуется повторно настраивать параметры регулятора (KP, TN, TV).

Пример:

```
VAR
    fbPID:    util.PID;

    actual:   REAL := 45.0;   // текущее значение
    setPoint: REAL := 50.0;   // уставка

    yOut:     REAL;           // выход регулятора
    limits:   BOOL;           // признак ограничения выхода
    overflow: BOOL;           // признак переполнения интегральной части
END_VAR

fbPID(
ACTUAL    := actual,
SET_POINT := setPoint,
KP        := 2.0,
TN        := 10.0,
TV        := 1.0,
Y_OFFSET  := 0.0,
Y_MIN     := 0.0,
Y_MAX     := 100.0,
MANUAL    := FALSE,
RESET     := FALSE
);

yOut      := fbPID.Y;
limits    := fbPID.LIMITS_ACTIVE;
overflow  := fbPID.OVERFLOW;
```

3.4.3 PID_FIXCYCLE

Функциональный блок PID_FIXCYCLE — ПИД-регулятор, для которого время цикла можно задавать вручную.

Имя переменной	Тип данных	Описание
Входные переменные		
ACTUAL	REAL	Фактическое значение, измеренная величина
SET_POINT	REAL	Заданное значение, уставка
KP	REAL	Коэффициент пропорциональности **P**
TN	REAL	Время интегрирования **I** [сек]. **TN** > 0, иначе регулятор не вычисляется
TV	REAL	Постоянная времени дифференцирования **D** [сек]. Если задана как 0, то работает как P-регулятор
Y_MANUAL	REAL	**Y** устанавливается в это значение при **MANUAL** = TRUE
Y_OFFSET	REAL	Смещение (offset) регулирующего воздействия
Y_MIN	REAL	Минимальное значение регулирующего воздействия
Y_MAX	REAL	Максимальное значение регулирующего воздействия

Имя переменной	Тип данных	Описание
MANUAL	BOOL	TRUE: ручной режим, **Y** не изменяется регулятором; FALSE: **Y** формируется регулятором
RESET	BOOL	TRUE: устанавливает **Y** в значение **Y_OFFSET** и сбрасывает интегральную часть
CYCLE	REAL	Время между двумя вызовами, в секундах
Выходные переменные		
Y	REAL	Управляющее воздействие
LIMITS_ACTIVE	BOOL	TRUE: **Y** превысило заданные пределы **Y_MIN**, **Y_MAX** и ограничено этими значениями
OVERFLOW	BOOL	Переполнение в интегральной части

Представляет PID-регулятор, для которого время цикла можно задавать вручную.

PID-регулятор непрерывно вычисляет значение ошибки **e (t)** как разность между заданным значением (уставкой) и измеренной технологической величиной. PID-регулятор формирует корректирующее воздействие на основе пропорциональной, интегральной и дифференциальной составляющих (обычно обозначаемых **P**, **I** и **D** соответственно), что и определяет тип регулятора.

ПРИМЕЧАНИЕ
Для быстрых задач с фиксированным циклом рекомендуется использовать PID_FIXCYCLE вместо PID, поскольку время цикла задаётся точно, тогда как PID может измерять время цикла лишь с точностью не больше миллисекунд. При очень коротких циклах (1 мс) это может приводить к нестабильной работе (см. PID).

ПРИМЕЧАНИЕ
При изменении времени цикла не требуется повторно настраивать параметры регулятора (KP, TN, TV).

Для получения дополнительной информации см. PID.

Пример:

```
VAR
    fbPID:    util.PID_FIXCYCLE;

    actual:   REAL := 45.0;   // текущее значение
    setPoint: REAL := 50.0;   // уставка
    cycle:    REAL := 0.01;   // время цикла, с (например, 10 мс)
    yOut:     REAL;           // выход регулятора
    limits:   BOOL;           // признак ограничения выхода
    overflow: BOOL;           // признак переполнения интегральной части
END_VAR

fbPID(
ACTUAL    := actual,
SET_POINT := setPoint,
KP        := 2.0,
TN        := 10.0,
TV        := 1.0,
Y_OFFSET  := 0.0,
Y_MIN     := 0.0,
Y_MAX     := 100.0,
MANUAL    := FALSE,
RESET     := FALSE,
CYCLE     := cycle
);

yOut      := fbPID.Y;
limits    := fbPID.LIMITS_ACTIVE;
overflow  := fbPID.OVERFLOW;
```

3.5 Типы данных

3.5.1 POINT

Пользовательский тип данных **POINT** - это структура с двумя полями.

Имя	Тип
X	DINT
Y	DINT

POINT предназначен для хранения пары координат/значений **X** и **Y**, например, “точка” на плоскости, точка графика/характеристики и т.д.

Пример:

```
VAR
   p: util.POINT;   // переменная типа POINT (структура с полями X и Y)
END_VAR

p.X := 1;           // записать значение в поле X
p.Y := 2;           // записать значение в поле Y
```

3.5.2 GEN_MODE

Пользовательский тип данных **GEN_MODE** — перечисление GEN_MODE задаёт режим (тип) генерируемого сигнала.

Имя	Начальное значение
TRIANGLE	0
TRIANGLE_POS	1
SAWTOOTH_RISE	2

Имя	Начальное значение
SAWTOOTH_FALL	3
RECTANGLE	4
SINUS	5
COSINUS	6
SINE	7
COSINE	8

Пример:

см. GEN

3.6 Манипуляторы функций

3.6.1 CHARCURVE

Функциональный блок CHARCURVE — преобразование входного сигнала по характеристической кривой.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	INT	Входной сигнал
N	BYTE	Количество точек, задающих характеристическую кривую: 2 <= **N**<= 11
Входные/выходные переменные		
P	ARRAY[0..10] OF POINT	Массив точек для описания характеристической кривой
Выходные переменные		
OUT	INT	Выходная переменная, содержит преобразованное значение
ERR	BYTE	0: нет ошибки; 1: ошибка в **P**: неверная последовательность (полностью протестировано только если **IN** равно наибольшему значению **X** в **P**); 2: **IN** вне пределов **P**; 4: неверное **N**, количество точек вне допустимого диапазона 2..11

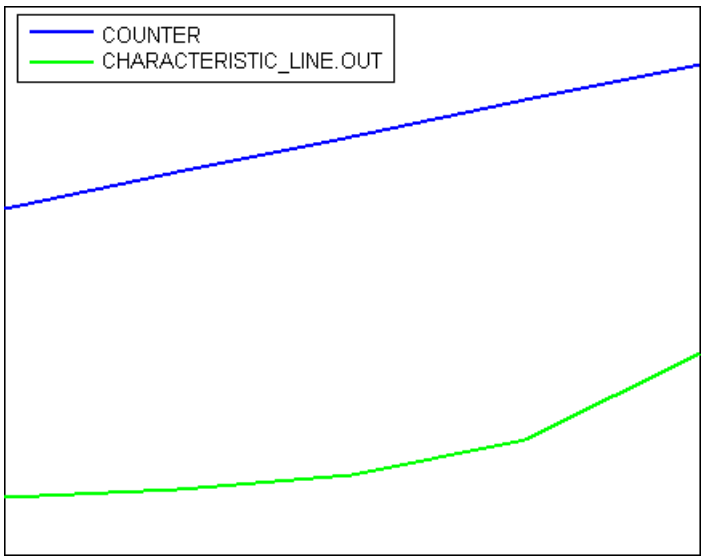

Рисунок 3.3

Характеристическая кривая задаётся массивом POINT-ов, который содержит набор значений **X** и соответствующих им значений **Y**.

Пример:

```
VAR
    CHARACTERISTIC_LINE:    util.CHARCURVE;
    KL: ARRAY[0..10] OF     util.POINT := [
       (X := 0, Y := 0),
       (X := 250, Y := 50),
       (X := 500, Y := 150),
       (X := 750, Y := 400),
       7((X := 1000, Y := 1000))];
    COUNTER:                INT;
END_VAR

COUNTER := COUNTER+10; //подает на CHARCURVE, например, постоянно растущий сигнал
CHARACTERISTIC_LINE(IN := COUNTER, N := 5, P := KL);
```

3.6.2 RAMP_INT

Функциональный блок RAMP_INT — ограничение скорости изменения значения до заданного уровня.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	INT	Входное значение
ASCEND	INT	Ограничение нарастания: максимальное увеличение за интервал **TIMEBASE**. Пример: **ASCEND** := 25 и **TIMEBASE** := T#1s => максимальное увеличение на 25 единиц в секунду
DESCEND	INT	Ограничение спада: максимальное уменьшение за интервал **TIMEBASE** (**DESCEND** > 0). Пример: **DESCEND** := 20 и **TIMEBASE** := T#500ms => максимальное уменьшение на 20 единиц за 500 мс

Имя переменной	Тип данных	Описание
TIMEBASE	TIME	Опорный интервал для **ASCEND**/**DESCEND**. Если **TIMEBASE** = T#0s: **ASCEND**/**DESCEND** задаются на один вызов; иначе: **ASCEND**/**DESCEND** задаются на указанный интервал времени. Если **TIMEBASE** = T#0s, то в этом случае **TIMEBASE** равна времени цикла задачи. Обычно это точно соответствует одному вызову функционального блока RAMP_INT. **ПРИМЕЧАНИЕ** Если TIMEBASE меньше времени цикла задачи, это приведёт к нарушению теоремы отсчётов и может привести к потере информации в выходном сигнале
RESET	BOOL	Сброс функционального блока. TRUE: останавливает внутренние вычисления и повторно инициализирует функциональный блок; последнее вычисленное значение **OUT** сохраняется, чтобы продолжить вычисления с него при следующем запуске. FALSE: выдаёт сглаженный входной сигнал на выход **OUT**
Выходные переменные		
OUT	INT	Значение с ограниченной “рампой”. Значение сохраняется внутренне и используется для расчёта нарастания и спада входного сигнала

Ограничение скорости изменения задаётся максимальным нарастанием **ASCEND** и максимальным спадом **DESCEND**, а также временем **TIMEBASE**, определяющим интервал времени, к которому относятся значения **ASCEND** и **DESCEND**.

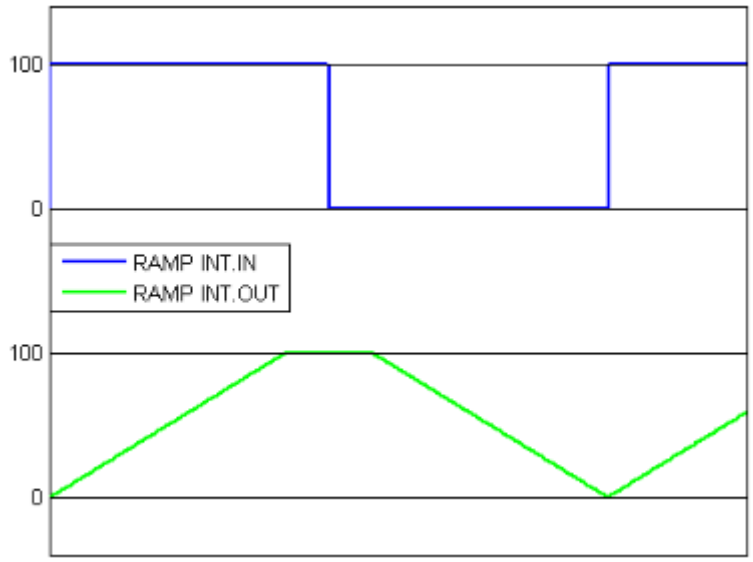

Рисунок 3.4

Пример:

```
VAR
    Ramp:     util.RAMP_INT;
    InValue:  INT;       // входной сигнал (ступенька 0/100)
    OutValue: INT;       // выход RAMP_INT
    Toggle:   BOOL;      // переключение входа
END_VAR

// Пример входного сигнала: переключение между 0 и 100
IF Toggle THEN
InValue := 100;
ELSE
InValue := 0;
END_IF;

Ramp(IN := InValue, ASCEND := 25, DESCEND := 20, TIMEBASE := T#1s, RESET := FALSE);

OutValue := Ramp.OUT;
```

3.6.3 RAMP_REAL

Функциональный блок RAMP_REAL — ограничивает скорость изменения значения до заданного уровня.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	REAL	Входное значение
ASCEND	REAL	Максимальная положительная скорость изменения
DESCEND	REAL	Максимальная отрицательная скорость изменения. Значение не должно быть отрицательным
TIMEBASE	TIME	Опорный интервал для **ASCEND**/**DESCEND**. Если **TIMEBASE** = T#0s: **ASCEND**/**DESCEND** задаются на один вызов; иначе: **ASCEND**/**DESCEND** задаются на указанный интервал времени
RESET	BOOL	Сброс функционального блока
Выходные переменные		
OUT	REAL	Значение с ограниченной скоростью изменения

Функциональный блок аналогичен RAMP_INT, с той разницей, что входы **IN**, **ASCEND**, **DESCEND** и выход **OUT** имеют тип **REAL**. RAMP_REAL требует больше вычислительного времени, но выполняет расчёт точнее, чем RAMP_INT.

ПРИМЕЧАНИЕ
Сброс функционального блока (RESET = TRUE) останавливает вычисления, при этом последнее вычисленное выходное значение OUT сохраняется. Если затем “рампа” снова запускается (RESET = FALSE), расчёт продолжится, начиная с этого последнего выходного значения.

Пример:

```
VAR
    Ramp:     util.RAMP_REAL;
    InValue:  REAL;       // входной сигнал (ступенька 0/100)
    OutValue: REAL;       // выход RAMP_REAL
    Toggle:   BOOL;       // переключение входа
END_VAR

// Пример входного сигнала: переключение между 0 и 100
IF Toggle THEN
InValue := 100;
ELSE
InValue := 0;
END_IF;

Ramp(IN := InValue, ASCEND := 25.5, DESCEND := 20.5, TIMEBASE := T#1s,
RESET := FALSE);

OutValue := Ramp.OUT;
```

3.7 Математические функции

3.7.1 DERIVATIVE

Функциональный блок HYSTERESIS — аппроксимация производной заданного значения по времени.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	REAL	Входное значение
TM	DWORD	Время с последнего вызова, в мс
RESET	BOOL	TRUE: **OUT** устанавливается в 0, а сохранённые значения устанавливаются в текущее входное значение **IN**
Выходные переменные		
OUT	REAL	Производная

Для расчёта записываются четыре последовательных значения, чтобы итоговая производная была максимально точной.

Пример:

```
VAR
    fbDerivative: util.DERIVATIVE;
    rInput:       REAL := 0.0;     // входной сигнал
    dwTime:       DWORD := 100;    // время между вызовами, мс (пример: 100 мс)
    xReset:       BOOL := FALSE;   // сброс
    rOut:         REAL;            // производная (выход)
END_VAR

// Пример изменения входного значения
rInput := rInput+ 1.0;
fbDerivative(IN := rInput, TM := dwTime, RESET := xReset);

rOut := fbDerivative.OUT;
```

3.7.2 INTEGRAL

Функциональный блок INTEGRAL — приближённо вычисляет интеграл по времени.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	REAL	Входное значение
TM	DWORD	Время с последнего вызова, в мс
RESET	BOOL	TRUE: **OUT** устанавливается в 0, а **OVERFLOW** — в FALSE
Выходные переменные		
OUT	REAL	Значение интеграла. Вычисляется суммированием частичных интегралов **IN** * **TM**
OVERFLOW	BOOL	TRUE: значение **OUT** вышло за диапазон переменных типа REAL; функциональный блок заблокирован до повторной инициализации входом **RESET**

Для расчёта записываются четыре последовательных значения, чтобы итоговая производная была максимально точной.

Пример:

```
VAR
    integral_inst: util.INTEGRAL;
    rInput:        REAL := 1.0;   // входное значение (например, постоянный сигнал)
    dwTime:        DWORD := 100;  // время между вызовами, мс (пример: 100 мс)
    xReset:        BOOL := FALSE; // сброс интегратора
    rOut:          REAL;          // значение интеграла
    xOverflow:     BOOL;          // признак переполнения
END_VAR

integral_inst(IN := rInput, TM := dwTime, RESET := xReset);

rOut := integral_inst.OUT;
xOverflow := integral_inst.OVERFLOW;
```

3.7.3 LIN_TRAFO

Функциональный блок LIN_TRAFO — выполняет линейное преобразование.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	REAL	Входное значение
IN_MIN	REAL	Минимальное входное значение
IN_MAX	REAL	Максимальное входное значение
OUT_MIN	REAL	Соответствующее минимальное выходное значение
OUT_MAX	REAL	Соответствующее максимальное выходное значение
Выходные переменные		
OUT	REAL	Выходное значение
ERROR	BOOL	TRUE: **IN_MIN** = **IN_MAX** или **IN** вне диапазона входных значений

Преобразование выполняется с использованием минимального и максимального значений входа для линейной аппроксимации выходного значения.

Пример применения: Датчик температуры выдаёт значения напряжения унифицированного сигнала 0...10 В (вход **IN**). Эти значения необходимо преобразовать в температуру в градусах Цельсия (выход **OUT**). Диапазон входных значений (**B**) задаётся пределами **IN_MIN** = 0 и **IN_MAX** = 10. Диапазон выходных значений (°C) задаётся пределами **OUT_MIN** = -20 и **OUT_MAX** = 40. Таким образом, при входном значении 5 В на выходе получится температура 10 °C.

Пример:

```
VAR
    fbLinTrafo: util.LIN_TRAFO;
    rVolt:      REAL := 5.0;      // Напряжение с датчика (В)
    rTemp:      REAL;             // Температура (°C)
    xError:     BOOL;             // признак ошибки
END_VAR

fbLinTrafo(IN := rVolt, IN_MIN := 0.0, IN_MAX := 10.0, OUT_MIN := -20.0,
OUT_MAX := 40.0);

rTemp := fbLinTrafo.OUT;
xError := fbLinTrafo.ERROR;
```

3.7.4 STATISTICS_INT

Функциональный блок STATISTICS_INT — вычисляет минимальное, максимальное и среднее значение входного сигнала типа данных INT.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	INT	Входное значение
RESET	BOOL	TRUE: **AVG** устанавливается в 0, **MN** и **MX** - в минимально и максимально возможные значения типа INT
Выходные переменные		
MN	INT	Минимальное значение
MX	INT	Максимальное значение
AVG	INT	Среднее значение

Расчёт выполняется во времени и может быть сброшен, чтобы начать новое вычисление/статистику.

Пример:

```
VAR
    fbStat: util.STATISTICS_INT;
    iValue: INT := 0;         // входное значение
    xReset: BOOL := FALSE;    // сброс статистики
    iMin:   INT;              // минимум
    iMax:   INT;              // максимум
    iAvg:   INT;              // среднее
END_VAR

fbStat(IN := iValue, RESET := xReset);

iMin := fbStat.MN
iMax := fbStat.MX
iAvg := fbStat.AVG
```

3.7.5 STATISTICS_REAL

Функциональный блок STATISTICS_REAL — вычисляет минимальное, максимальное и среднее значение входного сигнала типа данных REAL.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	REAL	Входное значение
RESET	BOOL	TRUE: **AVG** устанавливается в 0, **MN** и **MX** - в минимально и максимально возможные значения типа REAL
Выходные переменные		
MN	REAL	Минимальное значение
MX	REAL	Максимальное значение
AVG	REAL	Среднее значение

Расчёт выполняется во времени и может быть сброшен, чтобы начать новое вычисление/статистику.

Пример:

```
VAR
    fbStat: util.STATISTICS_REAL;
    iValue: REAL := 0;         // входное значение
    xReset: BOOL := FALSE;     // сброс статистики
    iMin:   REAL;              // минимум
    iMax:   REAL;              // максимум
    iAvg:   REAL;              // среднее
END_VAR

fbStat(IN := iValue, RESET := xReset);

iMin := fbStat.MN
iMax := fbStat.MX
iAvg := fbStat.AVG
```

3.7.6 VARIANCE

Функциональный блок VARIANCE — вычисляет математическую дисперсию переменной во времени.*

Имя переменной	Тип данных	Описание
Входные переменные		
IN	REAL	Входное значение
RESET	BOOL	Сброс
Выходные переменные		
OUT	REAL	Дисперсия

*Под «во времени» здесь понимается, что при каждом вызове функционального блока значение дисперсии дополняется (накапливается) до тех пор, пока не будет выполнен сброс.

Пример:

```
VAR
    fbVariance: util.VARIANCE;
    rValue:     REAL := 10.0;      // входное значение
    xReset:     BOOL := FALSE;     // сброс расчета
    rVar:       REAL;              // дисперсия
END_VAR

fbVariance(IN := rValue, RESET := xReset);

rVar := fbVariance.OUT;
```

3.8 Сигналы

3.8.1 BLINK

Функциональный блок BLINK — генерирует "мигающий" сигнал (включение и выключение с заданной длительностью).

Имя переменной	Тип данных	Описание
Входные переменные		
ENABLE	BOOL	TRUE: начать мигание FALSE: остановить мигание, при этом **OUT** сохраняет своё значение
TIMELOW	TIME	Время, в течение которого **OUT** = FALSE
TIMEHIGH	TIME	Время, в течение которого **OUT** = TRUE
Выходные переменные		
OUT	BOOL	Выходной сигнал, начинается с FALSE и переключается между TRUE и FALSE с заданной периодичностью.

Пример:

```
VAR
    fbBlink: util.BLINK;
    xEnable: BOOL := TRUE;       // разрешить переключение
    tLow:    TIME := T#500ms;    // время OUT = FALSE
    tHigh:   TIME := T#500ms;    // время OUT = TRUE
    xOut:    BOOL;               // входной сигнал
END_VAR

fbBlink(ENABLE := xEnable, TIMELOW := tLow, TIMEHIGH := tHigh);

xOut := fbBlink.OUT;
```

3.8.2 FREQ_MEASURE

Функциональный блок FREQ_MEASURE — измерение частоты сигнала.

Имя переменной	Тип данных	Описание
Входные переменные		
IN	BOOL	Входной сигнал

Имя переменной	Тип данных	Описание
PERIODS	INT (1..10)	Период - это время между двумя нарастающими фронтами входного сигнала. **OUT** равен средней частоте за заданное число периодов **PERIODS** (количество периодов: минимум 1, максимум 10). Функциональный блок работает только если значение находится в диапазоне 1..10
RESET	BOOL	Сброс измерения
Выходные переменные		
OUT	REAL	Частота [Гц]
VALID	BOOL	FALSE: пока не завершено первое измерение; или если временной интервал между двумя фронтами > 3 * **OUT** (указывает на некорректный входной сигнал)

Пример:

```
VAR
    fbFreq:    util.FREQ_MEASURE;
    xDI:       BOOL;           // дискретный вход (например, датчик импульсов)
    iPeriods: INT := 5;        // усреднение по 5 периодам (1..10)
    xReset:    BOOL := FALSE;  // сброс измерения
    rFreqHz:   REAL;           // измеренная частота, Гц
    xValid:    BOOL;           // признак валидности измерения
END_VAR

fbFreq(IN := xDI, PERIODS := iPeriods, RESET := xReset);

rFreqHz := fbFreq.OUT;
xValid := fbFreq.VALID;
```

3.8.3 GEN

Функциональный блок GEN — генерирует периодические функции заданных типов.

Имя переменной	Тип данных	Описание
Входные переменные		
MODE	GEN_MODE	Доступные типы: • **TRIANGLE**: треугольный сигнал от **-AMPLITUDE** до **+AMPLITUDE**; • **TRIANGLE_POS**: треугольный сигнал от 0 до **+AMPLITUDE**; • **SAWTOOTH_RISE**: пилообразный сигнал, возрастающий от **-AMPLITUDE** до **+AMPLITUDE**; • **SAWTOOTH_FALL**: пилообразный сигнал, убывающий от **+AMPLITUDE** до **-AMPLITUDE**; • **RECTANGLE**: прямоугольный сигнал, переключающийся от **-AMPLITUDE** до **+AMPLITUDE**; • **SINE**: синус; • **COSINE**: косинус
BASE	BOOL	FALSE: период задаётся по количеству вызовов (**CYCLES**); TRUE: период задаётся по времени (**PERIOD**)
PERIOD	TIME	Время периода; используется только при **BASE** = TRUE
CYCLES	INT	Количество вызовов на период; используется только при **BASE** = TRUE
AMPLITUDE	INT	Амплитуда генерируемой функции
RESET	BOOL	TRUE: устанавливает **OUT** в 0
Выходные переменные		
OUT	INT	Сгенерированное значение функции

Генерация может выполняться относительно заданной временной базы или относительно количества вызовов (**BASE**). Пример доступных типов периодических функций приведён на изображении.

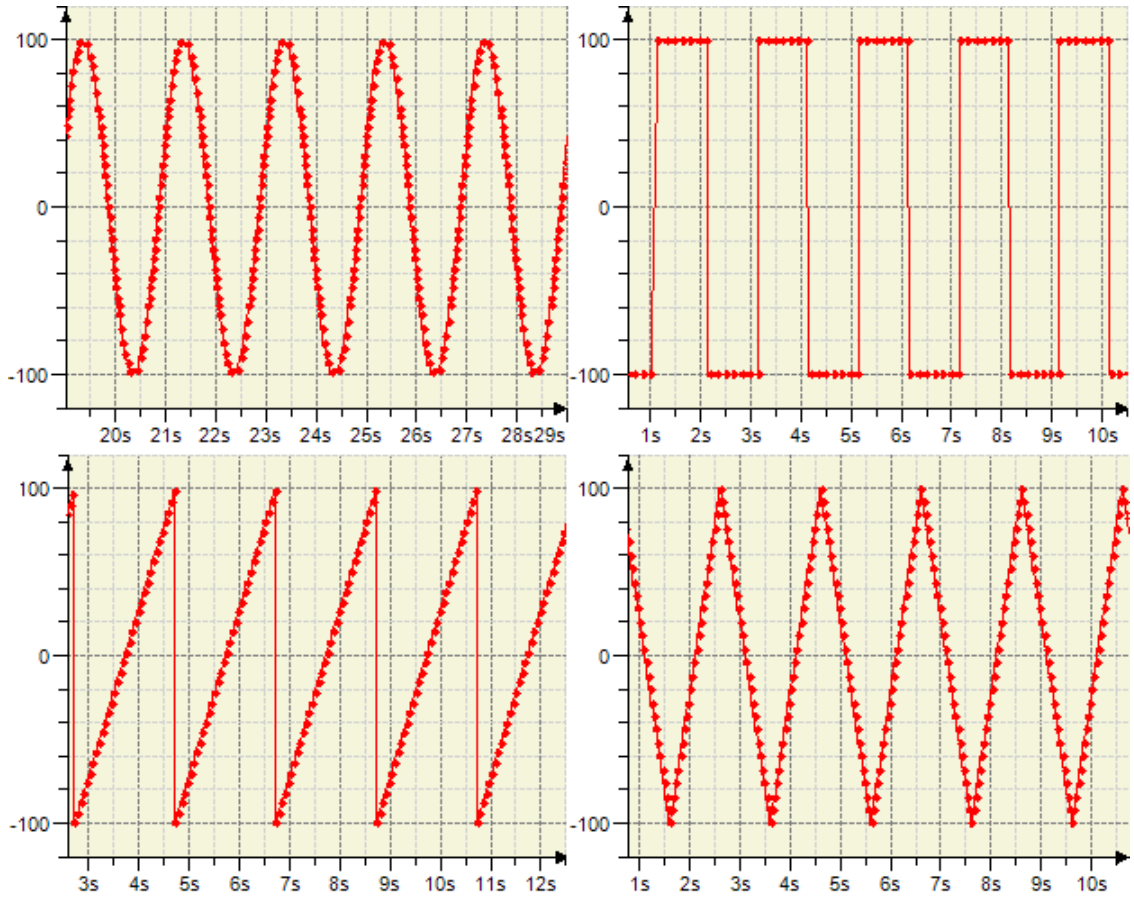

Рисунок 3.5

Пример:

Генерация прямоугольного сигнала заданной частоты и амплитуды

```
VAR
    fbgen:  util.GEN;    (* экземпляр ФБ *)
    signal: INT;         (* выходной сигнал *)
END_VAR

(* Генерация треугольника от -AMPLITUDE до +AMPLITUDE.
Период задан по количеству вызовов: BASE = FALSE, CYCLES = 100 *)

fbgen(
    MODE      := util.TRIANGLE,
    BASE      := FALSE,
    CYCLES    := 100,
    AMPLITUDE := 1000,
    RESET     := FALSE,
    OUT       => signal
);
```

3.9 Версия библиотеки Util

3.9.1 Version_Util

Функция Version_Util позволяет в программе просмотреть версию библиотеки Util.

Пример:

```
VAR
   wVersion: WORD;
END_VAR

wVersion := util.Version_Util(b:=True);
```

ЦИФРОВЫЕ
РЕШЕНИЯ

ООО "Овен Цифровые решения"
Россия, г. Москва, пл. Семёновская, д. 1А, помещ. 3/1
support@owendigital.ru
www.owendigital.ru

рег.:1-RU-157591-1.7