ИНТЕГРИРОВАННЫЙ СЕРВОПРИВОД СПШ10

Язык программирования

Версия 4.2

ЗАО «Сервотехника», 2012 год

ОГЛАВЛЕНИЕ

СПШ10 – это интегрированный сервопривод на базе гибридного шагового электродвигателя, в котором используется бесшаговое (векторное) управление на основе адаптированного специально для шаговых двигателей алгоритма.

Данный документ описывает язык программирования сервопривода SML и примеры программ для решения типовых задач.

Внимание. Данное описание действительно для версии программного обеспечения МотоМастер 1.13.0 и выше, и версии программного обеспечения сервопривода СПШ 16.4 и выше, версии привода СПС 128.0 и выше.

ЗАО «Сервотехника» не возлагает на себя обязанность оповещать пользователей о появлении обновлений комплекта документации и программного обеспечения. Пожалуйста, следите за новостями на сайте компании www.servotechnica.ru.

Особые указания по пользованию руководством

Отдельные указания имеют следующее значение:

ОПАСНОСТЬ:
Означает, что непринятие соответствующих мер предосторожности подвергает опасности жизнь и здоровье пользователя.

ПРИМЕЧАНИЕ:
Указывает на то, что неправильное обращение может привести к неправильной работе устройства. Однако опасностей для здоровья пользователя или риска повреждения аппаратуры или иного имущества не имеется.
Кроме того, примечания такого рода могут обращать внимание пользователя на возможность иной настройки параметра, наличие иной функции или возможность применения дополнительных или расширительных устройств.

Программируемый логический контроллер

В системе управления сервоприводов, выпускаемых компанией ЗАО «Сервотехника», реализован программируемый логический контроллер (ПЛК).

ПЛК предназначен для исполнения прикладных программ, созданных пользователем. Основная задача ПЛК – это предоставить возможность автономно выполнять вспомогательные операции самим приводом без использования контроллера верхнего уровня.

К наиболее часто используемым задачам, решаемым с помощью встроенного ПЛК, относятся:

Выход в нулевую позицию по позиционному выключателю.

Аварийный останов привода в случае выхода из рабочей зоны действия.

Управление режимом работы привода.

Включение/выключение внешнего оборудования по определенным событиям.

С помощью ПЛК можно решать и более комплексные задачи:

Автономная работа нескольких приводов, объединенных в локальную сеть с помощью шины CAN, с целью совместного позиционирования.

Разработка программ для внутреннего ПЛК выполняется на языке SML (Servo Motor Language), разработанного компанией ЗАО «Сервотехника» специально для сервоприводов.

Разработка и отладка программ может выполняться в программе МотоМастер©, для этого в нее включен текстовый редактор, расположенный в окне «Контроль». Подробное описание интерфейса программы МотоМастер©, приведен в документе «I - руководство пользователя .pdf».

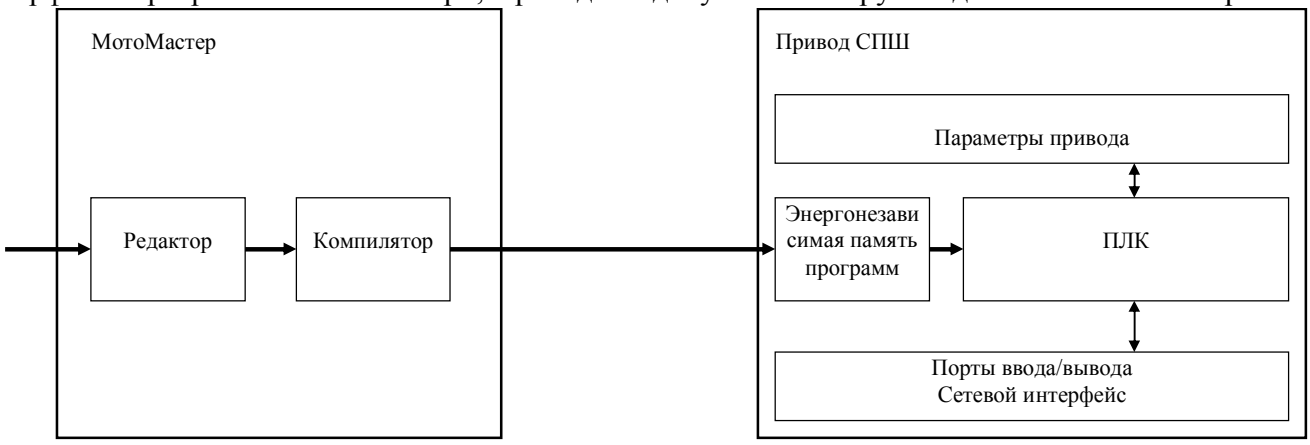

Рис. 1. Взаимодействие МотоМастер и привода в процессе программирования ПЛК.

С помощью программы МотоМастер© пользователь разрабатывает программное обеспечение. Далее при записи программы в привод компилятор, встроенный в МотоМастер©, проверяет синтаксис программы и в случае корректного написания выполняет компиляцию и запись программы в энергонезависимую память программ привода. Далее пользователь может выполнить запуск программы на выполнение. При задании режима запуска программы по событию включение, ПЛК автоматически загружает указанную программу из памяти и, таким образом, готов работать автономно.

В энергонезависимой памяти системы управления привода может храниться до 8 пользовательских программ. При этом в процессе отработки при необходимости ПЛК может переходить от выполнения одной программы к другой.

ПЛК привода работает в фоновом режиме и прерывается такими задачами реального времени как расчет контуров тока, скорости, позиции, прием данных по интерфейсам и пр. Поэтому частота обработки программы зависит от загруженности процессора.

При стандартных настройках привода время выполнения одной команды программы в ПЛК составляет в среднем 50 мкс. Временные характеристики ПЛК в конкретном приложении привода можно проанализировать с помощью параметров dd15 (Время выполнения одной команды интерпретатором), dd16 (Время выполнения программы интерпретатором). Параметры можно анализировать как в режиме реального времени, для оценки средних величин, так и в виде графиков, используя для этого осциллограф.

После разработки программы рекомендуется сохранять программу на диск компьютера с целью возможности восстановления программ после потенциальной потери данных энергонезависимой памяти. Такая ситуация возможна при выходе двигателя из строя, при импорте параметров привода.

В текущей версии редактора программ Мотомастер© существует ряд ограничений:

- не допускаются пустые строки в программе, в т. ч. и последняя;
- не допускаются лишние пробелы и символы табуляции внутри команд;
- все команды чувствительны к регистру;
- размер каждой программы (в объектном коде) не может превышать 256 слов.

Принятые обозначения

При описании языка программирования SML используются ряд символических обозначений, описание которых приведено в данной главе.

<Константа>

Целочисленное знаковое число.

<Константа IQ12>

Дробное число в формате IQ12. Подробнее см. пункт «***Формат IQ12***».

Все константы могут быть введены в десятичном или шестнадцатеричном форматах.

Формат представления десятичных чисел:

1
15
256

Формат представления шестнадцатеричных чисел:

0x1
0x0F
0xFF

Формат представления дробных чисел:

123.1
0.0012
-0.1

ПРИМЕЧАНИЕ:
Для разделения целой и дробной частью допускается использовать только точку.

Максимальный диапазон констант от -2^{31} до $2^{31}-1$.

Пример использования констант:

```
X=10
IF(Z=-10)
  Y=0x0A
ENDIF
```

<Переменная>

Символическое обозначение пользовательских переменных. При написании программ пользователю доступны 3 переменных общего назначения

X, Y, Z

Переменная содержит 32х битное знаковое значение.

Все переменные не имеют определенного типа. Число может содержать как целочисленное значение, так и значение, хранящееся в формате IQ12.

Все переменные обнуляются при загрузке программы на выполнение.

<Параметр>

Символическое обозначение параметра привода. В программе пользователя ПЛК доступны все параметры привода, которые представлены в дереве конфигурации МотоМастера©. Переменные доступны как на чтение, так и на запись. Запрет на запись имеется для переменных, имеющих атрибут только на чтение (см. документ II - описание параметров.pdf).

Работу с параметрами необходимо выполнять в соответствие с их форматом. Значения некоторых параметров привода хранятся в формате с фиксированной точкой, например, параметр dd1 «Ток в обмотке 1» хранится в формате IQ12 и при значении параметра равном 1.5 в ПЛК будет считано 1.5*(4096)=6144. Формат хранения параметров описан в документе «II - описание параметров.pdf».

Обращение к переменным выполняется по имени, указанному в дереве конфигурации МотоМастера©. Например:

```
ip0=4
X=ip1
Z=up0+X
Z=up1|0x00000001
```

<Операнд>

Используется для обозначения одного любой записи из списка: *<Константа>, <Переменная>*.

<*Операция*>

*Данный символ обозначает а*рифметическую или логическую операцию.

Операция	Описание
+	Сумма
-	Вычитание
/	Деление
*	Умножение
&	Логическое побитовое ‘И’
|	Логическое побитовое ‘ИЛИ

ПРИМЕЧАНИЕ:
Данные операции относятся только к целочисленным значениям. Для чисел в формате IQ12 используются специальные функции.

Запись типа
<*Переменная*> =<*Переменная*> <*Операция*> <*Переменная*>

которая может встречаться по тексту, может означать, например, одно из следующих выражений:

X=Z/2
Z=Y*X
Y=Y&X

<*Выражение*>

Используется в данном руководстве для обозначения произвольного сочетания переменных, констант и операций:

Запись типа
X =<*Выражение*>

которая может встречаться по тексту, может означать, например, одно из следующих выражений:

X=Y
X=Y&0x0F
X=X&Y
X=up0&Y
up1=Y/10
X=15

ОГРАНИЧЕНИЯ: В выражениях может использоваться не более одной операции: запись типа X=X&Y&Z недопустима.
В выражениях может использоваться не более одного параметра: записи типа X=up0+up1, up0=up1 не допустимы.
Если в выражении присутствует операция, то выражение может начинаться только с переменной или параметра: запись типа X=0/Z недопустима.

<*Условие*>

Используется в условных операциях, циклических операциях, операциях ожидания и пр.

<*Выражение*> = <*Переменная*>	Условие истинно, если <*Выражение*> равно <*Переменная*>
<*Выражение*> < <*Переменная*>	Условие истинно, если <*Выражение*> меньше <*Переменная*>
<*Выражение*> > <*Переменная*>	Условие истинно, если <*Выражение*> больше <*Переменная*>
<*Выражение*> *!=* <*Переменная*>	Условие истинно, если <*Выражение*> неравно <*Переменная*>

<*Выражение*> = <*Константа*>	Условие истинно, если <*Выражение*> равно <*Константе*>
<*Выражение*> < <*Константа*>	Условие истинно, если <*Выражение*> меньше <*Константе*>
<*Выражение*> > <*Константа*>	Условие истинно, если <*Выражение*> больше <*Константе*>
<*Выражение*> *!=* <*Константа*>	Условие истинно, если <*Выражение*> неравно <*Константе*>

Примеры условий:
(X=1)
(up0&0x0F=X)
(X+Y=Z)

ОГРАНИЧЕНИЯ: В качестве операций в условии не может использоваться * и / .

Команды

Команды перехода и останова

Синтаксис команды завершения выполнения текущей программы ПЛК:

HALT

остановить выполнение программы ПЛК, при этом все установленные программой параметры (задания по положению и скорости, значения портов выхода и др.) сохраняются при дальнейшей работе.

Синтаксис команды повторного выполнения текущей программы ПЛК:

REPEAT

повторить выполнение основного тела программы с первой строки, при этом все установленные программой параметры (задания по положению и скорости, значения портов выхода) сохраняются.

Синтаксис команды перехода на выполнение программы ПЛК:

PROGRAM <Константа>

где ***<Константа>*** = 0..7 — перейти на выполнение программы, хранящейся в банке ***<Константа>***. Значения переменных X, Y и Z при этом обнуляются.

ПРИМЕЧАНИЕ:
Для корректного завершения работы ПЛК все программы должны завершаться одной из этих команд.

Временная задержка

Для паузы в выполнении программы используется команда:

D=<Выражение>

приостановить выполнение программы на заданное количество миллисекунд. Максимальный временной интервал, который может сформировать данная функция, составляет 32767 мс.

Пример использования временной задержки

Прог. 1.

X=1000	
D=up0+X	Сформировать задержку, время которой представляет собой сумму содержимого параметра up0 и переменной X
Z=Z+1	Увеличить значение переменной Z на 1
D=1	Задержка 1 мс.
REPEAT	Повторить программу сначала

Установка режима движения

Управление позицией

Вариант 1. Синтаксис команды:

P=<Выражение>

установить задание контура позиции.

Скорость движения будет ограничена параметром vp8 (Максимальная скорость вращения).

Вариант 2. Синтаксис команды:

P=<Операнд>,W=<Операнд>

установить задание контура позиции и ограничить скорость.

Команда изменяет значение параметра vp8.

Как для варианта 1, так и для варианта 2 ускорение будет зависеть от параметра vp9 (Динамический режим), по следующему правилу:

a. vp9=Режим с максимальной динамикой
   Привод выполняет разгон до номинальной скорости с максимально возможным ускорением.
b. vp9=Режим плавного разгона/торможения
   Привод выполняет разгон до номинальной скорости с ускорением, заданным параметров vp5 (Ускорение).

Вариант 3. Синтаксис команды:

P=<Операнд>,W=<Операнд>,A=<Операнд>

установить задание контура позиции, ограничить скорость и ограничить ускорение. При этом ускорение будет ограничено только если контур скорости работает в режиме плавного разгона.

Команда изменяет значение параметров vp5, vp8.

При выполнении команд управления позицией автоматически замыкается контур позиции.

ПРИМЕЧАНИЕ:
Задание позиции выполняется асинхронно, т.е. программа продолжает свое выполнение, не дожидаясь окончания перемещения.
ПРИМЕЧАНИЕ:
При выполнении данной команды ПЛК автоматически замыкает контур позиции, поэтому привод выполнит задание независимо от состояния параметра pp5 (Состояние контура позиции).

Управление скоростью

Вариант 1. Синтаксис команды:

W=<Выражение>

Установить задание скорости. При этом ускорение будет задаваться как описано в п. «Управление позицией» настоящего документа.
При выполнении команды автоматически замыкается контур скорости и размыкается контур позиции.

Вариант 2. Синтаксис команды:

W=<Операнд>,A=<Операнд>

установить задание скорости и задать требуемое ускорение. При этом скорость будет ограничена значением параметра vp8. Команда изменяет значение параметра vp5. Ускорение будет ограничено только если контур скорости работает в режиме плавного разгона.

ПРИМЕЧАНИЕ:
Программа продолжает выполнение, не дожидаясь установления заданной скорости.
ПРИМЕЧАНИЕ:
При выполнении данной команды ПЛК автоматически размыкает контур позиции и замыкает контур скорости, поэтому привод выполнит задание независимо от состояния параметров pp5 (Состояние контура позиции) и vp7(Состояние контура скорости).

Управление ускорением

Синтаксис команды: ***A=<Выражение>***

установить ускорение. Команда изменяет значение параметра vp5.

ПРИМЕЧАНИЕ:
Не используйте команды управления положением, если контур положения разомкнут и вал двигателя вращается. Это приведет к резким переходным процессам по скорости, т.к. команды управления позицией автоматически замыкают контур позиции.
Не используйте команды управления скоростью, если контур положения замкнут и вал двигателя вращается. Это приведет к резким переходным процессам по скорости, т.к. команды управления позицией автоматически размыкают контур позиции.

Примеры программ управления движением

Прог. 2.

X=1000	Инициализация переменной
Y=100	Инициализация переменной
W=X,A=Y	Задание скорости вращения и ускорения
X=X-100	
WAIT(W>X)	Ожидание выхода на требуемую скорость
W=0	Прекратить вращение
WAIT(W<5)	Ожидание завершения вращения
REPEAT	Повторить программу сначала

Результат отработки программы Прог. 2:

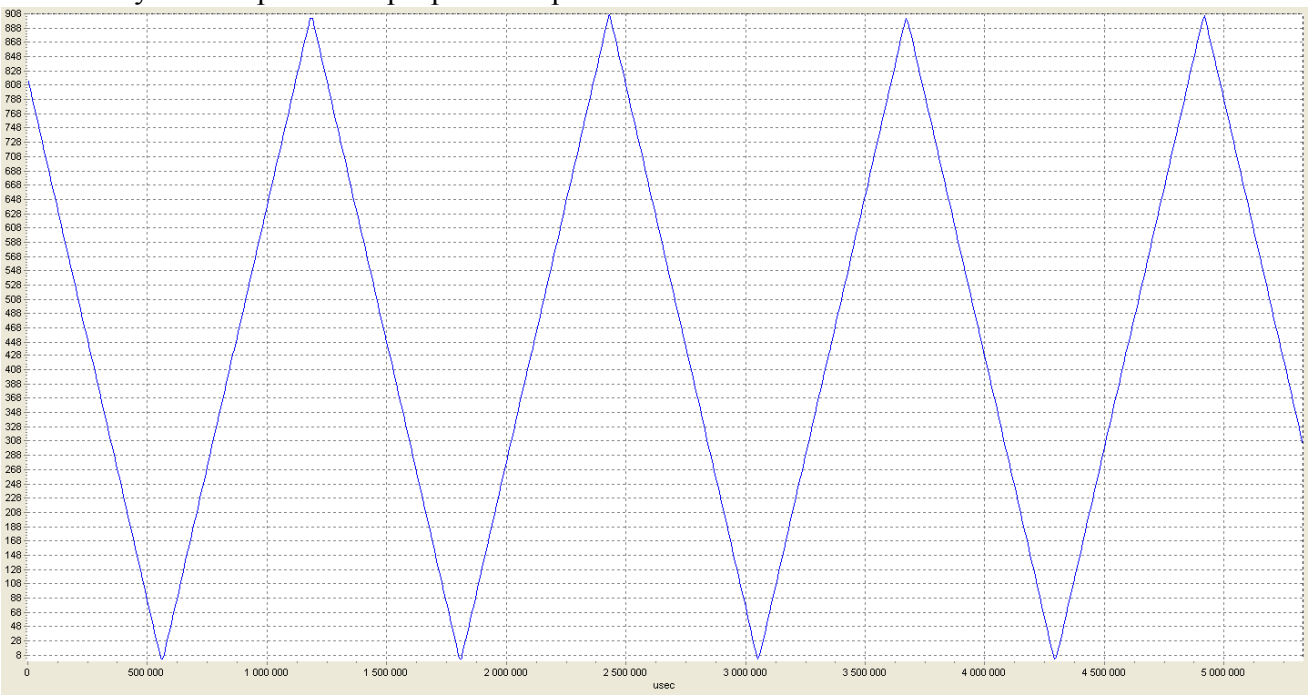

Рис. 2. График скорости в масштабе 1:1 (синий).

Прог. 3.

P=100000,W=1000,A=10	Задать позицию с ограничением скорости и ускорения
WAIT(W>500)	Ожидание достижения определенной скорости
P=100000,W=1000,A=50	Изменение ограничений скорости и ускорения
WAIT(P>50000)	Ожидание достижения промежуточной позиции
P=100000,W=1500,A=100	Изменение ограничений скорости и ускорения
WAIT(P=100000)	Ожидание завершения движения
P=0,W=2000,A=200	Возврат в исходную позицию
D=2000	Ожидание завершения цикла
REPEAT	Повторить программу сначала

Результат отработки программы Прог. 3:

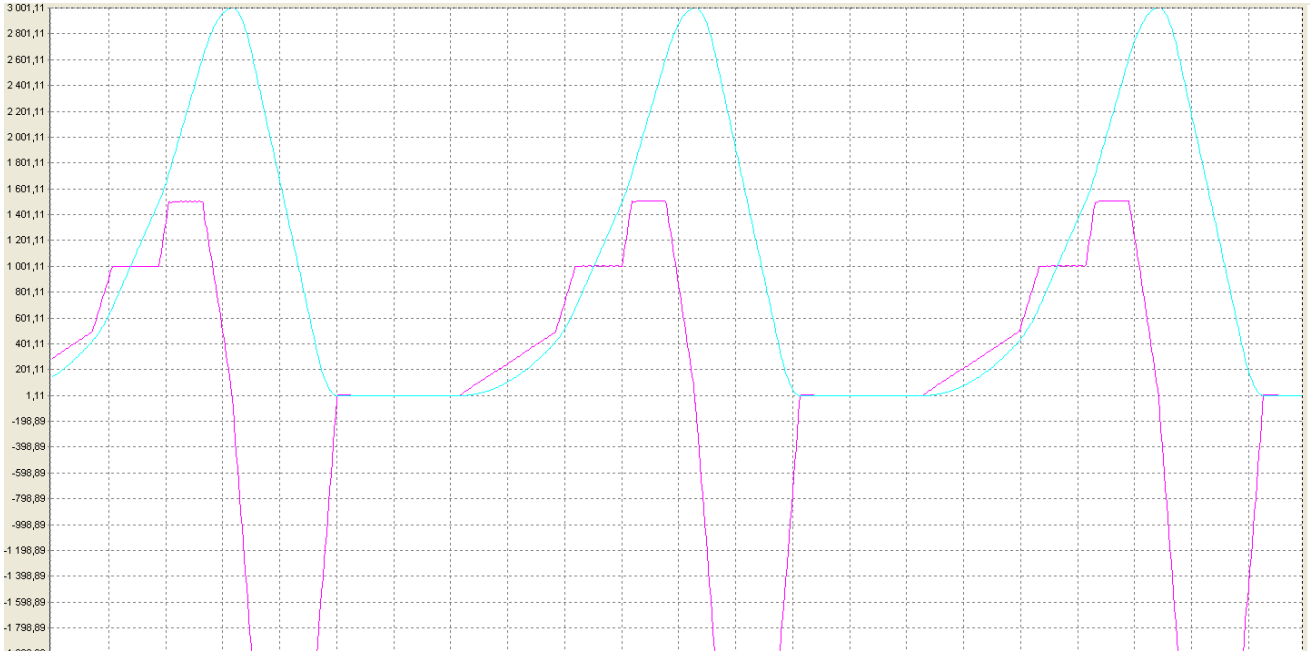

Рис. 3. График скорости в масштабе 1:1 (красный), график позиции 1:35 (зеленый).

Прог. 4.

X=W+200	Инициализация переменной параметром текущей скорости вращения + 200
W=X	Установка задания контура скорости
D=300	Задержка
IF(X>2000)	Если скорость превысила 2000 об/мин
X=P	то запомним текущую позицию
W=0	Остановим привод
WAIT(W=0)	Дождемся останова
P=X	Установим задание контура скорости (выполним возврат в запомненную позицию)
HALT	Останов программы
ENDIF	Конец условного оператора
REPEAT	Повторить программу сначала

Результат отработки программы Прог. 4.

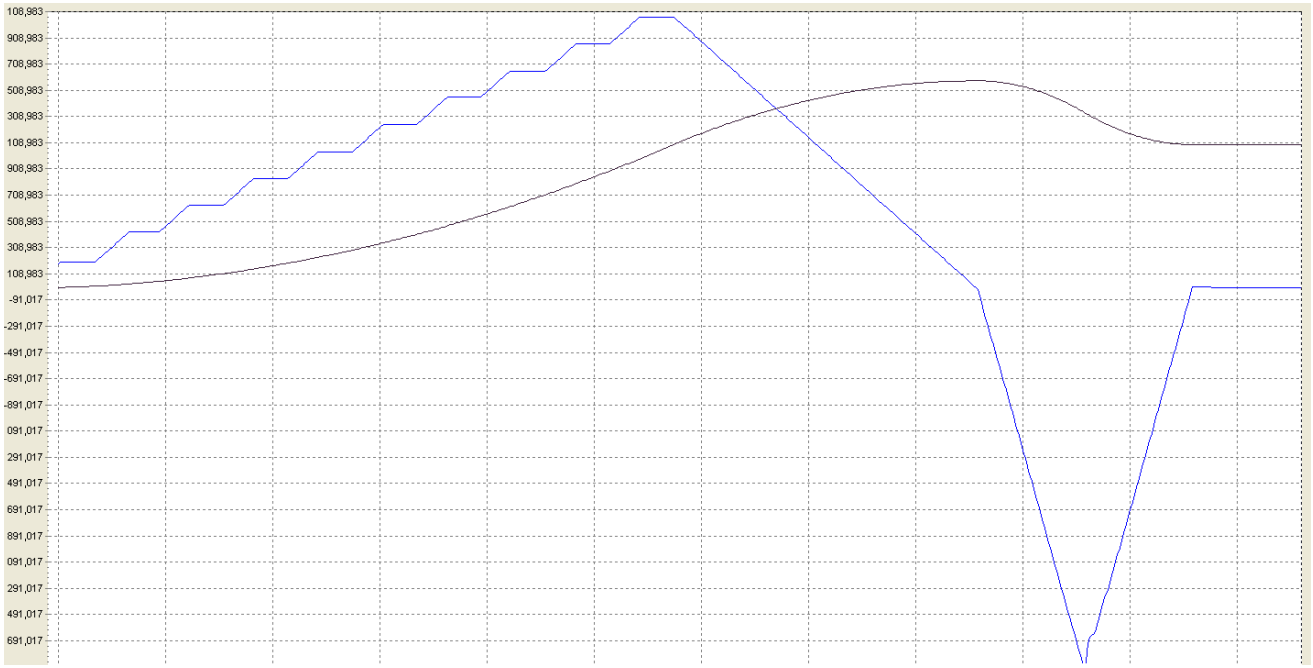

Рис. 4. График скорости в масштабе 1:1 (синий), график позиции 1:200 (черный).

Управление портами ввода-вывода

Программам пользователя доступны цифровые порты ввода-вывода, выведенные на разъем привода DB15-M и описанного в таблице «Разъем дискретного входа/выхода» в документе «I - руководство пользователя.pdf».

Синтаксис команды установки цифровых выходов:

P_OUT.n=<Константа>

, где n может принимать значения 0 или 1.

Синтаксис чтения портов ввода:

<Переменная>=PORT

При этом в переменную ***<Переменная>*** будут считаны 4 младших бита данных начиная с 0го в соответствие с номером портов ввода разъема DB15-M, описание которого приведено в таблице «Разъем дискретного входа/выхода» в документе «I - руководство пользователя.pdf».

Операнд ***PORT*** может также использоваться в условных операциях.

Порты ввода могут использоваться индивидуально в условных операторах, при этом синтаксис обращения к порту ввода показан ниже:

P_IN.n

, где n может принимать значения 0, 1, 2 или 3.

Для исключения эффекта «дребезга контактов» (влияния шумов) на обработку входных сигналов драйвер портов ввода после изменения сигнала на входе каждого порта не опрашивает порт в течение 3 мс, что позволяет эффективно бороться с переходными процессами на портах при переключении между логическими состояниями. Если данного алгоритма недостаточно для исключения шума на портах ввода, то можно установить внешний фильтр, например, в виде RC-цепи или выполнить фильтрацию сигнала программно.

ПРИМЕЧАНИЕ:
P_IN.n и P_OUT.n не могут быть использованы в выражениях.

Примеры программ работы с портами ввода-вывода

Прог. 5.

X=PORT&2	Считать значение порта 1 в переменную X
IF(X!=0)	Если значение порта не равно нулю, то
P_OUT.0=0	Выставить порт вывода 0 в состояние логического нуля.
HALT	Остановить выполнение программы
ENDIF	
P_OUT.0=1	Иначе выставить значение порта вывода 0 в состояние логической единицы
IF(P_IN.0==0)	Если значение порта 0 не равно нулю, то
P_OUT.1=1	Выставим значение порта вывода 1 в состояние логической единицы
WAIT(P_IN.0=1)	Выполним цикл ожидания возврата порта ввода 0 в состояние логической единицы и
P_OUT.1=0	Обнулим порт вывода 0
ENDIF	
REPEAT	Повторить программу сначала

Функция ABS

Функция ABS может использоваться только в условных операциях для получения значения числа без учета его знака. Синтаксис функции:

ABS(<Выражение>) = <Операнд>

ABS(<Выражение>) > < Операнд >

ABS(<Выражение>) < < Операнд >

ABS(<Выражение>) != < Операнд >

Пример использования функции ABS

Прог. 6.

IF(ABS(dd2)>5)	Обнулить переменную Z, если абсолютное значение переменной dd2 больше 5
Z=0	

```
ENDIF
X=500
WAIT(ABS(W)>X)        Ожидание превышения текущей скорости вращения значения переменной X
REPEAT                Повторить программу сначала
```

Условные операции и циклы ожидания

Условные операции предназначены для выполнения определенной части кода программы только при соблюдении определенных условий. Синтаксис команды:

IF(<Условие>)

<Действия>

ENDIF

В качестве действий могут выступать любые команды, в том числе и вложенные условные операции.

Допускается использование вложенных условий.

Предусмотрена также разновидность условного оператора, синтаксис которой приведен ниже:

IF(<Условие>)

<Действия 1>

ELSE

<Действия 2>

ENDIF

Отличие данной конструкции заключается в том, что если <Условие> верно, то выполняются <Действия 1>, иначе <Действия 2>.

Команда WAIT предназначены для прерывания программы до наступления определенных событий описанных в условии цикла ожидания. Синтаксис команды:

WAIT(<Условие>)

ВНИМАНИЕ! Не следует использовать команду WAIT в обработчиках событий, описание которых приведено в п. «События».

Пример использования функции условных операций и операции ожидания

Прог. 7.

```
IF(P_IN.0=0)          Если значение порта 0 установлено в 0, то
 IF(dd11=0)           ---------Вложенный цикл- если параметр dd11=0, то
  P_OUT.0=0           ------------------Выставить значение порта 0 в значение 0
 HALT                 ------------------и приостановить программу
 ELSE                 ---------Иначе
```

```
  P_OUT.0=1          ------------------Выставить значение порта 0 в значение 1
 ENDIF               ---------
ELSE                 Иначе
 WAIT(P=X)           ---------Ожидание достижение позиции значения переменной X
 X=X+4000            ---------
 P=X                 ---------Установить новое значение контура позиции
ENDIF
REPEAT               Повторить программу сначала
```

Циклические операции

Данная конструкция предназначена для выполнения циклических операций при соблюдении указанного условия. Синтаксис команды:

WHILE(<Условие>)

<Действия>

ENDWHILE

Допускается использование вложенных циклов.

Пример использования циклических операций

Прог. 8.

```
X=10
WHILE(X>0)           Цикл приема 10 импульсов с порта 0
 WAIT(P_IN.0=1)
 WAIT(P_IN.0=0)
 X=X-1
ENDWHILE
P=P+4000             Выполнить перемещения на 4000 импульсов
D=1000
REPEAT               Повторить программу сначала
```

Прог. 9.

```
X=P/4000             Выделить целое количество оборотов вала двигателя
IF(X<0)              Если позиция отрицательная, то
 P_OUT.0=0           Выставить направление - отрицательное
 WHILE(X<0)          Цикл выдачи импульсов, равных количеству оборотов[1]
  P_OUT.1=1
  D=500
  P_OUT.1=0
  D=500
  X=X+1
 ENDWHILE
ELSE                 Иначе
 P_OUT.0=1           Выставить направление - положительное
```

```
 WHILE(X>0)                Цикл выдачи импульсов, равных количеству оборотов[1]
  P_OUT.1=1
  D=500
  P_OUT.1=0
  D=500
  X=X-1
 ENDWHILE
ENDIF
HALT                       Прервать программу
```

[1] – пример составлен для привода с разрешением энкодера 4000 импульсов на оборот.

События

Помимо синхронной обработки различных событий, такие как опрос портов (например, Прог. 8), в языке программирования SML предусмотрена возможность асинхронной обработки различных событий по аналогии с обработчиками прерывания в языках ассемблер и Си. Для добавления события необходимо его зарегистрировать в начале программы с указанием условий, при котором активизируется его обработчик.

Каждое событие может находиться в активном или неактивном состоянии. Если событие активизировано, то обработчик запустится при срабатывании условия. В неактивном состоянии обработчик события не вызывается.

Синтаксис регистрации события, неактивного по умолчанию:

EVENT <Номер> <Условие>

Синтаксис регистрации события, активного по умолчанию:

EVENT <Номер> + <Условие>

Также доступно специальное событие START, срабатывающее только перед запуском программы на выполнение. Синтаксис регистрации:

EVENT <Номер> + START

, где ***<Номер>*** – порядковый номер события.

ОГРАНИЧЕНИЯ

1. Нумерация событий должна идти подряд начиная с 0.
2. В условиях событий нельзя использовать параметры привода.

Помимо активации события при декларации, предусмотрены команды явной активации и деактивации событий, которые могут быть использованы как в теле основной программы, так и в теле обработчика событий. Синтаксис команды активации события:

+EVENT <Номер>

Синтаксис команды деактивации события:

-EVENT <Номер>

При выполнении условия «Условие», ПЛК прерывает выполнение основной программы и переходит к обработчику события. Все процедуры обработки событий располагаются непосредственно за основным телом программы. Синтаксис обработчика событий:

ON_EVENT <Номер>

<Операции>

ОГРАНИЧЕНИЯ

1. Перед обработчиком первого события обязательно должна присутствовать команда REPEAT, HALT или PROGRAM.
2. В обработчиках событий нельзя использовать команду задержки D=<мс>.

Непосредственно перед началом выполнения обработчика события произошедшее событие автоматически переводится в неактивное состояние.

При одновременном срабатывании нескольких условий событий, сначала будет выполнен обработчик события с меньшим номером.

Если при выполнении обработчика события выполнено условие другого события, то его обработчик будет вызван только если условие сохранится после окончания выполнения обработчика текущего события.

ВНИМАНИЕ! Обработчик события по смыслу близок к обработчику прерывания в процессоре, поэтому избегайте использования в обработчиках потенциально опасных действий таких как циклы ожидания (WAIT), циклические операции (WHILE). Данные операции могут привести к длительному или бесконечному циклу обработки события, что повлияет как на работу ПЛК, так и на работу других модулей, таких как модуль связи с ПК и др.

Пример использования событий

Прог. 10.

EVENT 0 + START	Декларация события 0, которое активируется один раз непосредственно перед запуском программы на выполнение
WHILE(X>0)	Цикл последовательного перемещения на 4000 импульсов X раз
X=X-1	
P=P+4000	
D=1000	
ENDWHILE	
HALT	Завершить выполнение программы
ON_EVENT 0	Обработчик события 0
X=up0	Выполнить инициализацию переменной X, значением, находящимся в параметре up0

Прог. 11.

EVENT 0 + P_IN.0=1	Событие 0 — нажата кнопка
EVENT 1 + P_IN.0=0	Событие 1 — отпущена кнопка
REPEAT	Основная программа – представляет собой пустой цикл ожидания событий
ON_EVENT 0	Обработчик события 0 - когда кнопка нажата
P_OUT.0=0	Снять сигнал готовности
W=0,A=1000	Выполнить быстрый останов
WAIT(ABS(W)<10)	Ожидание завершения вращения
+EVENT 1	Разрешить реакцию на отпускание кнопки
ON_EVENT 1	Обработчик события 1 - когда кнопка отпущена
A=200	Восстановить обычное ускорение
P_OUT.0=1	Установить сигнал готовности
+EVENT 0	Разрешить реакцию на следующее нажатие

Прог. 12.

EVENT 0 ABS(W)<100	Событие 0 — скорость ниже заданной. Событие по умолчанию не активное
W=200	Задание скорости
WAIT(W>100)	Ожидание выхода на заданное значение
+EVENT 0	Активизируем событие 0
REPEAT	Основная программа – представляет собой пустой цикл ожидания событий
ON_EVENT 0	Обработчик события 0 – низкая скорость вращения
P_OUT.0=1	Выставить аварийный сигнал
W=0,A=1000	Выполнить быстрый останов
WAIT(W<2)	Ожидание завершения вращения
$drv_status=1	Установить статус привода «Останов по команде»
cp7=0	Выключить генерацию ШИМ сигнала

Обмен сообщениями по шине CAN

В языке программирования ПЛК предусмотрена возможность обмена сообщениями по цифровой шине CAN между двумя и более приводами. Обмен по данной шине осуществляется в виде передачи сообщений строго определенного формата между конкретными узлами одной сети или в виде широковещательных сообщений. Для этих целей предусмотрены функции передачи и приема.

Синтаксис функции асинхронной передачи сообщений по шине CAN:

SEND(addr, cmd, par)

послать устройству с адресом addr сообщение с командой cmd и данными со значением параметра par. При этом посылка сообщения производится асинхронно, т. е. программа продолжит выполнение сразу после выполнения команды SEND, не дожидаясь завершения ее передачи.

addr=<Константа> – адрес удаленного узла от 0 до 7. Адрес 7 – широковещательное сообщение, которое будет принято всеми устройствами в сети.

cmd=<Константа> – пользовательская команда в диапазоне от 21 до 31.

par=<Операнд> - 32-битные данные (Константа или переменная общего назначения)

Данная функция приема может использоваться в основной программе для синхронного приема.

Синтаксис функции синхронного приема сообщений по шине CAN:

GET(addr, cmd, par)

ожидать от устройства с адресом addr сообщения с командой cmd, записать данные пришедшего сообщения в параметр Par.

addr=<Константа> – адрес удаленного узла от 0 до 7. Адрес 7 – прием данных от любого привода в сети.

cmd=<Константа> – пользовательская команда в диапазоне от 21 до 31.

par=<Переменная> - 32-битные данные (прием параметра в одну из переменных общего назначения).

Синтаксис функции асинхронного приема сообщений по шине CAN:

CAN_RECV(addr,cmd,par)

Значение параметров аналогично описанию функции GET.

Данная функция предназначена для использования в событиях для использования в поле <Условие>.

Пример использования шины CAN

Прог. 13. Работа конвейера с синхронизацией исполнительных устройств по шине CAN. Программа конвейера.

```
X=P+40000          Вычислить следующую позицию конвейера
P=X                Выполнить перемещение
WAIT(P=X)          Ожидание завершения перемещения
SEND(2,21,X)       Послать команду «конвейер передвинут»
GET(2,22,Y)        Ожидание команды «операция выполнена»
REPEAT             Повторить программу сначала
```

Прог. 14. Работа конвейера с синхронизацией исполнительных устройств по шине CAN. Программа ведомого привода.

```
GET(1,21,X)          Ожидание команды «Конвейер передвинут»
P=10000,W=500,A=100  Подвести инструмент
WAIT(P=10000)        Ожидание завершения перемещения.
P_OUT.1=1            Включить исполнительное оборудование
D=3000               Ожидание 3 сек
P_OUT.1=0            Выключить исполнительное оборудование
P=0,W=500,A=100      Отвести исполнительное оборудование
WAIT(P=0)            Ожидание завершения перемещения.
SEND(1,22,X)         Послать команду «операция выполнена»
REPEAT               Повторить программу сначала
```

Прог. 15. Синхронизация работы нескольких приводов по шине CAN. Программа ведущего.

```
EVENT 0 + START              Событие 0 – Начало программы
EVENT 1 + CAN_RECV(2,21,Y)   Событие 1 - Принято сообщение о завершении цикла привода 2
EVENT 2 + CAN_RECV(3,21,Y)   Событие 2 - Принято сообщение о завершении цикла привода 3
IF(X=7)                      Если все приводы закончили движение, то
 SEND(2,22,0)                Послать команду начала движения приводу 2
 SEND(3,22,0)                Послать команду начала движения приводу 3
 X=0                         Сбросить состояния готовности
 IF(Z=0)                     Вычислить следующее положение
  Z=10000
 ELSE
  Z=0
 ENDIF
 P=Z,W=500                   Переместиться в следующую позицию
 WAIT(P=Z)
 X=X|1                       Установить флаг завершения цикла привода 1
ENDIF
REPEAT
```

```
ON_EVENT 0
X=7                          Начать цикл при запуске программы
ON_EVENT 1
X=X|2                        Установить флаг завершения цикла привода 2
ON_EVENT 2
X=X|4                        установить флаг завершения цикла привода 3
```

Прог. 16. Синхронизация работы нескольких приводов по шине CAN. Программа ведомых приводов с адресами 2, 3.

```
GET(1,22,X)                  Ожидание команды начала движения
IF(Z=0)                      Вычислить следующую позицию
  Z=10000
ELSE
  Z=0
ENDIF
P=Z,W=200                    Переместиться в следующую позицию
WAIT(P=Z)
SEND(1,21,0)                 Отправить сообщение о завершении цикла
REPEAT
```

Передача сообщений в шину USB

Синтаксис функции передачи сообщений по шине USB:

#send_usb(<Операнд>)

Передача при этом будет выполнена по протоколу

Значение	**Описание**
0x01	SOH. Маркер начала пакета.
0x06	PARM_VAL_CMD. Команда – ответ привода на запрос значения параметра.
0x07	Группа пользовательских параметров.
0x08	Адрес переменной X, в группе.
<Операнд>	Значение операнда в виде 4х байт, начиная с младшего байта.
0x03	ETX. Маркер конца пакета.

При все байты операнда равные 0x01, 0x03, 0x1A будут обрабатываться механизмом прозрачности и передаваться в виде двух байт. С подробностями работы протокола можно ознакомиться в документе «***ECG. Руководство по эксплуатации.doc***».

Инициализация текущей позиции привода

Для инициализации текущей позиции привода используется следующая команда:

PCUR= <Выражение>

При выполнении данной команды выполняется одновременно инициализация текущей позиции привода и задания контура позиции, при этом движения вала двигателя не происходит.

ПРИМЕЧАНИЕ:
Не используйте команду установки текущей позиции при вращении вала двигателя.

Данная команда наиболее часто используется во время выхода в нулевую позицию, при работе привода в составе станка.

Прог. 17. Выход в нулевую позицию.

W=-350	Двигаться к левому концевому датчику
WAIT(P_IN.0=1)	Ожидание срабатывание концевого выключателя
W=30	Двигаться в обратном направлении с малой скоростью
WAIT(P_IN.0=0)	пока не разомкнется выключатель
P=P,A=5000	Максимально быстро захватить текущую позицию
D=500	Ожидание завершения позиционирования
PCUR=0	Объявляем текущую позицию за нуль
PROGRAM 1	Переход к основной программе

Усредненное значение момента

Для отслеживание усредненного значения момента, в языке программирования SMLиспользуется параметр I. Привод использует замкнутый принцип регулирования, поэтому в режиме позиционирования или в режиме управления скоростью привод автоматически корректирует развиваемый момент, чтобы преодолеть противодействующий момент и выполнить задание с требуемыми характеристиками. Поэтому параметр I может свидетельствовать о режиме работы привода.

Параметр I доступен только на чтение.

Параметр I может использоваться в выражениях и условных операциях. Формат параметра IQ12 см. п. Принятые обозначения.

Прог. 18.

W=W+100	Увеличить заданную скорость вращения на 100 об/мин
D=500	Ожидание завершения переходного процесса
X=I+100	Считать значение усредненного тока с некоторым корректирующим коэффициентом
IF(X>500)	При превышении током значения 0.1А
HALT	Приостановить выполнение программы
ENDIF	
REPEAT	

Наиболее часто данный параметр используется для выхода в нулевую позицию при использовании вместо концевого выключателя механического упора.

Прог. 19. Поиск нулевой позиции по упору.

W=-150	Задание контура скорости
WAIT(ABS(I)>800)	Усредненное развиваемый при токе примерно 0.2А
W=0,A =5000	Выполнить быстрый останов
D=500	Ожидание завершения останова
PCUR=0	Объявляем текущую позицию за нуль

PROGRAM 1 Переход к основной программе

Поиск z-метки

При использовании относительных датчиков, таких как квадратурные энкодеры, привод при включении не в состоянии выставить определенный угол, что требуется в некоторых приложениях, поэтому в большинство энкодеров присутствует дополнительный сигнал Z, который встречается один раз за оборот вала двигателя. Это позволяет приводу найти с высокой точностью нулевое позиции вала и относительно него выставить требуемый угол.

Кроме того, использование данной метки позволяет выполнить выход в нулевую позицию при работе приводов в составе станков, линейных модулей, с высокой точностью.

Для выполнения подобных задач в язык программирования включена функция:

#rst_pos_i(<Константа>)

, где ***<Константа>*** принимает значение 0, если при поиске z-метки требуется вращение против часовой стрелке или 1 - по часовой стрелке.

Наличие z-метки для приводов СПШ является опцией. При выполнении функции #rst_pos_i в приводе, у которого установлен энкодер без Z метки (см. параметр st5) выполнение программы будет остановлено и параметр ip11 будет установлен в значение «Не удалось выполнить поиск Z метки».

Прог. 20. Поиск нулевой позиции с использованием Z-метки.

```
W=-150                  Задание контура скорости
WAIT(P_IN.0=1)          Ожидание срабатывание концевого выключателя
W=0,A =5000             Выполнить быстрый останов
D=500                   Ожидание завершения останова
W=30                    Задание контура скорости в обратном направлении
WAIT(P_IN.0=0)          Ожидание выключения концевого выключателя
W=0,A=5000              Выполнить быстрый останов
PCUR=0                  Объявляем текущую позицию за нуль
#rst_pos_i(1)           Выполнить поиск Z метки
IF(P>4000)              вал повернулся больше чем на 1 оборот - ошибка [1]
  HALT                  Остановить программу
ENDIF
PCUR=0
PROGRAM 1               Перейти на основную программу
```

[1] – пример составлен для привода с разрешением энкодера 4000 импульсов на оборот.

Установка ограничений позиции

При работе привода в рамках исполнительной системы, как правило, существуют ограничение угла поворота вала двигателя. В языке программирования предусмотрены функции установки программных ограничителей позиции при движении в положительную и отрицательную сторону. Синтаксис функций:

 # p_limit_r(<Операнд>)

 # p_limit_l (<Операнд>)

При обнаружении выхода угла поворота вала за пределы отрезка [p_limit_l;p_limit_r], система управления выполнит останов с максимально возможным ускорением, а затем медленно вернется в допустимые пределы.

При установке ограничителей в значение -2^{31} их действие отключается.

Функции не записывают введенные программно ограничения по позиции в энергонезависимую память программ, поэтому ограничения сбрасываются после перезапуска привода.

Прог. 21. Пример программы использования программных ограничителей.

```
#p_limit_l(-2147483647)          Выключить левый ограничитель
#p_limit_r(2147483647)           Выключить правый ограничитель
X=2                              Инициализация счетчика цикла
WHILE(X>0)                       Цикл периодического движения
 P=8000
 D=1500
 P=-8000
 D=1500
 X=X-1
ENDWHILE
P=0
D=1500
#p_limit_l(-4000)                Установить левый ограничитель
#p_limit_r(4000)                 Установить правый ограничитель
X=2
WHILE(X>0)
 P=8000
 D=1500
 P=-8000
 D=1500
 X=X-1
ENDWHILE
HALT                             Прекратить выполнение программы
```

Результат выполнения программы Прог. 21 представлен на Рис. 5.

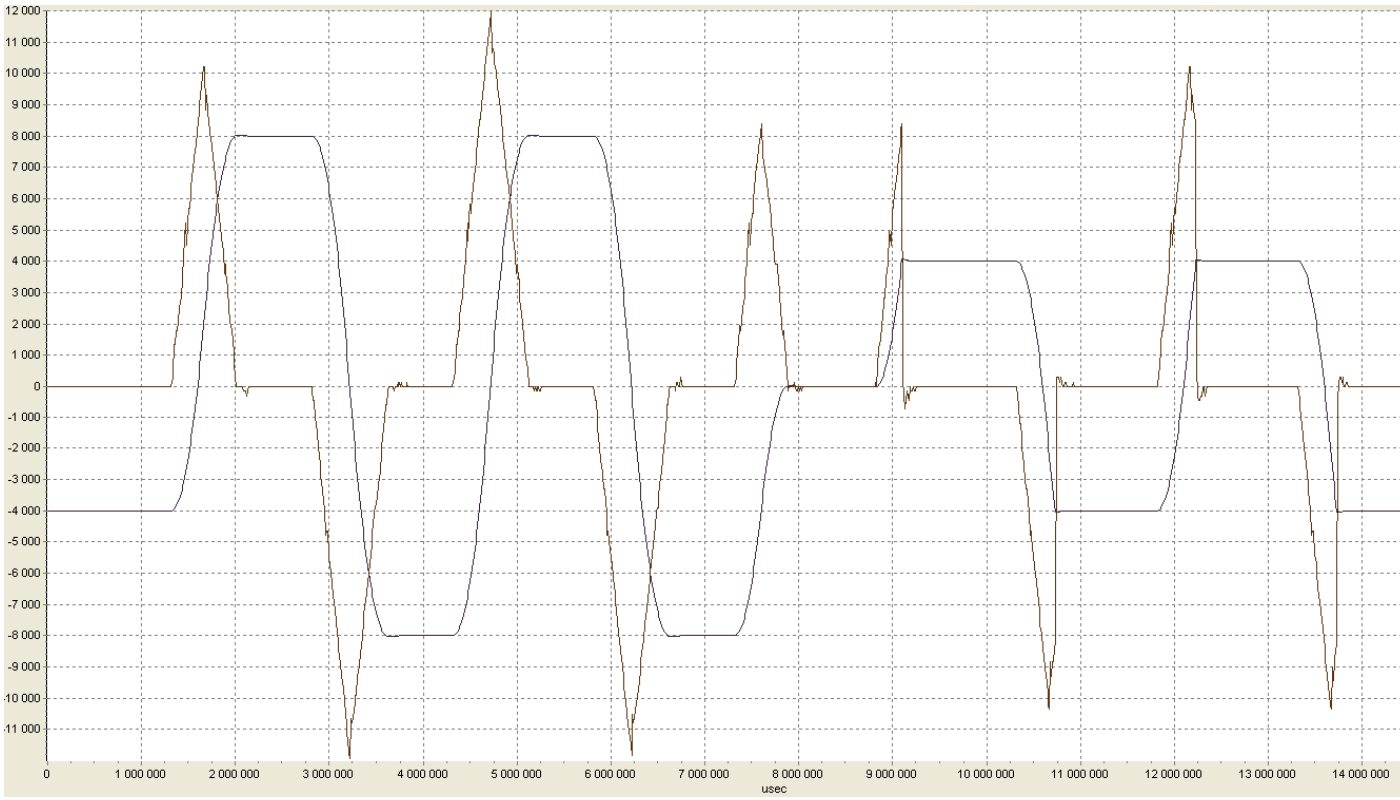

Рис. 5. Результат отработки программы Прог. 21. График позиции в масштабе 1:1 (синий), график скорости 20:1 (коричневый).

Статус привода

В языке программирования предусмотрена переменная, с помощью которой осуществляется доступ к статусу привода. Данная переменная может использоваться как в выражениях, так и в условиях:

$drv_status = <Выражение>

При установке привода в состояние 1, привод переводится в состояние «Останов». При этом генерация управляющего сигнала двигателя прекращается.

Установка статуса в состояние 0 возобновляет генерацию управляющего сигнала.

ПРИМЕЧАНИЕ:
При установке $drv_status в 0 производится процедура поиска фазы двигателя.

Прог. 22. Пример программы управления статусом.

```
EVENT 0 + P_IN.0=1          Событие — нажата кнопка аварийного останова
EVENT 1 P_IN.0=0            Событие — отпущена кнопка аварийного останова
IF($drv_status!=0)          Если привод по каким-либо причинам перешел в аварийное состояние, то
  P_OUT.0=0                   выводим статус на порт вывода
ELSE
```

```
  P_OUT.0=1
ENDIF
REPEAT
ON_EVENT 0
$drv_status=1                  Аварийный останов — снять питание с двигателей
+EVENT 1                       Разрешить включать привод при отпускании кнопки
ON_EVENT 1
$drv_status=0                  Кнопка аварийного останова отпущена — включить привод
+EVENT 0                       Разрешить реакцию на нажатие кнопки
```

Работа со счетчиком Step/Dir

Система управления привода имеет аппаратный счетчик импульсов, который подключен к портам IN0, IN1 (см. Руководство пользователя.pdf). Содержимое счетчика доступно программам пользователя как на чтение, так и на запись, посредством переменной:

$SD_counter=<Выражение>

Прог. 23. Пример программы управления статусом.

```
IF(ip0!=2)                     Если интерфейс Step/Dir не установлен, то
  IF($SD_counter!=0)           Обнуляем счетчик импульсов
    $SD_counter=0
  ENDIF
  IF(P!=0)                     Обнуляем позицию
    P=0
   WAIT(P=0)
  ENDIF
  ip0=2                        Устанавливаем интерфейс управления Step\Dir
ENDIF
HALT
```

Чтение значений аналоговых входов

Значения аналоговых входов 0 и 1 доступны в ПЛК через переменные

$analog_inputA

и

$analog_inputB

соответственно.

Переменные доступны только на чтение.

При этом оцифрованные значение аналогового сигнала на входе порта предварительно обрабатывается для удобства дальнейшей обработки. В данной версии системы управления используется 12 битный АЦП, поэтому после оцифровки сигнал может принимать значение в диапазоне от 0 до 4096, что соответствует аналоговому сигналу от -10В до +10В.

Далее сигнал преобразуется в соответствие со схемой, приведенной на Рис. 6.

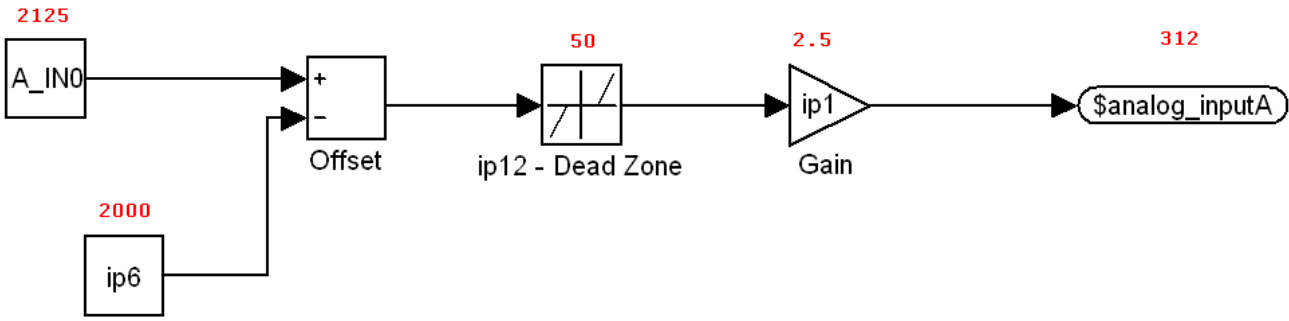

Рис. 6. Схема преобразование аналогового сигнала.

На рисунке Рис. 6 приведен также пример преобразования данных (выделенный красным цветом).

Прог. 24. Пример программы работы с аналоговыми сигналами.

```
X=$analog_inputA-$analog_inputB
IF(ABS(X)<1000)
W=X
ENDIF
REPEAT
```

Значения $analog_inputA и $analog_inputB учитывают коэффициент веса, смещение и мертвую зону аналоговых входов.

Установка порта аварийного останова

ПЛК отрабатывает программу 1 команду за ~50мкс в фоновом режиме (в режиме нереального времени), что не позволяет гарантировать временной интервал, в течение которого будет обработано то или иное событие, возникшее в приводе. Однако существуют события, связанные с безопасностью работы привода и системы в целом, при наступлении которых время реакции должно быть строго определено и заранее известно. К таким событиям относится выключение привода через порт ввода. Для таких событий в ПЛК СПШ предусмотрена функция #crash_port.

#crash_port(<Константа>),

где ***<Константа>,***в данном случае, формируется с помощью следующего выражения:

(Номер порта) | (Уровень << 4).

где ***Номер порта*** – десятичное число от 0 до 3, определяющее какой порт является входом сигнала аварийного останова, ***Уровень*** – принимает значение 0 или 1 и соответствует уровню, при котором срабатывает механизм аварийного останова.

Для отключения функции по аварийному входу введите команду:

#crash_port(15),

Время срабатывания аварийного останова в данном случае гарантированно не превысит 50мкс.

Например, если аварийный сигнал подключен к порту 1, а Уровень срабатывания 1, то сформированная константа будет равна (0x01) | (0x01<<4) = (1)|(0x10)=0x11.

Использование массива энергонезависимой памяти данных

В состав системы управления привода входит энергонезависимая память данных, часть из которой выделена для хранения данных общего назначения. Размер массива 6300 32-битных эле-

ментов. Массив доступен программам пользователя на чтение и запись. Для доступа к массиву используются переменные:

$eeprom_mas

Данная переменная указывает на текущий элемент массива. При обращении к данной переменной на чтение или запись индекс массива автоматически увеличивается на 1 и переменная $eeprom_mas начинает указывать на следующий элемент массива. При достижении конца массива индекс обнуляется, и индекс перемещается на первый элемент.

Для доступа к индексу массива энергонезависимой памяти общего назначения используется переменная:

$eeprom_mas_i

которая доступна на чтении и запись. При попытке установить индекс в недопустимое область памяти его содержимое обнуляется.

ПРИМЕЧАНИЕ:
Не выполняйте команды вида $eeprom_mas=$eeprom_mas+1, результат такого класса команд не определен.

Примеры использования массива энергонезависимой памяти данных

Прог. 25. Пример формирования произвольного профиля движения.

Данный пример представляет собой универсальный алгоритм отработки профиля любой сложности и произвольного размера. При этом формирование профиля может выполняться, например, в ПК и записываться в привод через Моторастер (меню Файл\Импорт EEPROM).

Пример можно дополнить функциями обмена сообщениями по шине CAN между приводами. При использовании подобной синхронизации можно обеспечить N мерное совместное движение, например, портала.

```
$eeprom_mas_i=0          Установить индекс на первый элемент массива
X=$eeprom_mas            Считать количество точек в профиле
WHILE($eeprom_mas_i<X)   Пока не кончится профиль
 Y=$eeprom_mas           Считать положение следующей точки
 Z=$eeprom_mas           Считать скорость на следующем участке
 P=Y,W=Z                 Отработать следующий участок
 WAIT(P=Y)               Ожидание завершения отработки
ENDWHILE
HALT
```

Например, при записи таблицы Табл. 1, отработанный профиль будет выглядеть как показано на Рис. 7.

Табл. 1. Профиль движения в формате пользователя.

```
12

4000
100
12000
200
40000
500
80000
```

1500
40000
2500
0
1000

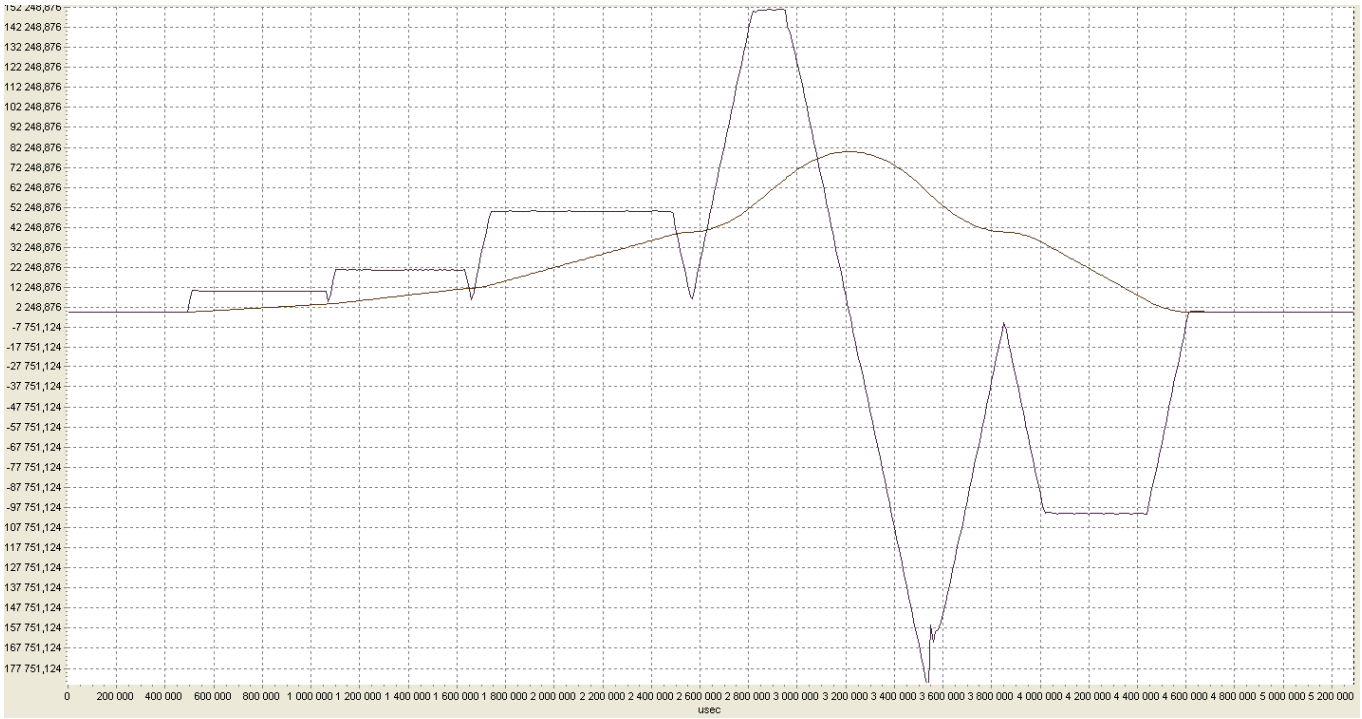

Рис. 7. Результат отработки профиля,приведенного в Табл. 1. График позиции в масштабе 1:1 (коричневый), график скорости 100:1 (фиолетовый).

Прог. 26. Пример программы формирования профиля движения в режиме обучения.

```
IF(X=0)
  $eeprom_mas_i=0          Инициализация индекса массива
ENDIF
WAIT(P_IN.0=0)             Ожидание сигнала захвата и сохранения текущей позиции. В процессе ожидания предполагается, что оператор устанавливает требуемую позицию привода.
$eeprom_mas=P              Захват и сохранение текущей позиции
WAIT(P_IN.0=1)             Ожидание снятия сигнала программирования
D=100
IF(X>5)                    Всего нужно запомнить 5 позиций
  HALT                       Завершить формирование профиля
ENDIF
X=X+1
IF(Y=0)                    Два условных оператора используются для индикации завершения процесса сохранения позиции.
  Y=1
  P_OUT.0=1
  D=100
ENDIF
```

```
IF(Y=1)
  Y=0
  P_OUT.0=0
ENDIF
REPEAT
```

Формат IQ12

В ПЛК привода СПШ поддерживаются операции с дробными числами. При этом используется формат дробного числа с фиксированной точкой IQ12.

Дробные числа представлены в виде знакового 32х битного значения.

Дробная часть числа в формате IQ12 занимает младшие 12 бит.

Например, дробное число 2.5 в формате IQ12 будет выглядеть как 10240.

Для хранения значений дробных чисел могут использоваться любые 32х битные переменные, которые доступные в программе, например, переменные общего назначения X, Y и Z.

ПРИМЕЧАНИЕ:
Компилятор ПЛК не следит за типами переменных. Пользователь должен быть внимательным при использовании разных форматов чисел, хранящихся в переменных.

Функции преобразования форматов iq, vartoiq, vartoint

Функция преобразования дробного числа, представленного в виде константы, в формат IQ12:

iq12(<Константа>)

ВНИМАНИЕ! Функция iq12 используется без решетки, в отличии от остальных функций.

Функция преобразования дробного числа, хранящегося в переменной, в формат IQ12:

#vartoiq(<Переменная>)

Функция преобразования дробного числа, хранящегося в переменной в формате IQ12, в целое число:

 # vartoint(<Переменная>)

Прог. 27. Пример программы преобразования форматов.

X=iq(1.5)	Преобразовать дробное число 1.5 в формат IQ12 и записать результат в переменную общего назначения X
Y=10	Проинициализировать переменную Y целым числом 10
Y=#vartoiq(Y)	Преобразовать переменную Y из целочисленного в дробное в формате IQ12
Z=#vartoint(Y)	Преобразовать переменную Y из дробного IQ12 в целое и записать результат в переменную Z.
HALT	

Результат после выполнения программы будет значение переменных:

X=6144
Y=40960
Z=10

Выделение дробной части frac

Функция выделение дробной части числа, хранящегося в переменной в формате IQ12:

<Переменная> = #vartoint(<Переменная>)

Например, после выполнения следующего фрагмента программы:
X=iq12(1.5)
Y=#frac(X)
Переменная Y будет содержать значение 0.5.

Функции для выполнения арифметических операций над дробными числами

Функции умножения и деления двух чисел в формате IQ12:

<Переменная> = #mpy (<Переменная>, <Переменная>)

<Переменная> = #div (<Переменная>, <Переменная>)

Прог. 28. Пример программы работы с функцией mpy.

```
X=iq(1.5)
Y=$analog_inputA        Считывание аналогового входа
Y=#vartoiq(Y)           Переводим в IQ12
Y=# mpy(Y, X)           Масштабирование переменной Y
Y=#vartoint(Y)          Преобразовать переменную Y из дробного IQ12 в целое.
HALT
```

Тригонометрические функции

<Переменная> = #sin (<Переменная>)

<Переменная> = #cos (<Переменная>)

Функции возвращают значение в формате IQ12 в диапазоне от 0.0 до 1.0.

<Переменная> = #atan (<Переменная 1>,<Переменная 2>)

Функция возвращают значение в формате IQ12 в диапазоне от 0.0 (0.0 радиан) до 1.0 (2π радиан).

<Переменная 1> - Y
<Переменная 2> - X

Arctan(Y/X)

Прог. 29. Пример программы вычисления sin.

```
P=#sin(X)
```

```
X=X+1
REPEAT
```

Результат выполнения программы представлен на Рис. 8.

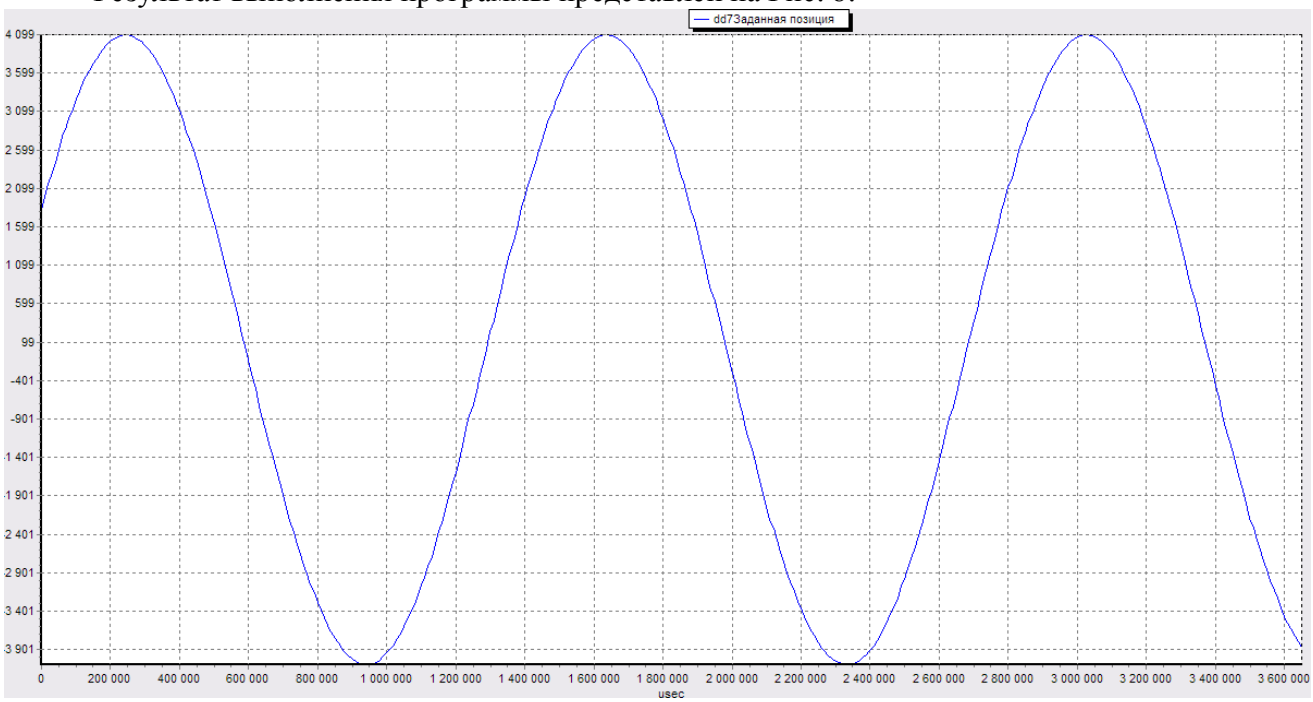

Рис. 8. Результат функции sin.

Прог. 30. Пример программы вычисления arctan.

```
Y=10
Y=#vartoiq(Y)
P=#atan(X,Y)
X=X+1
REPEAT
```

Результат выполнения программы представлен на Рис. 9.

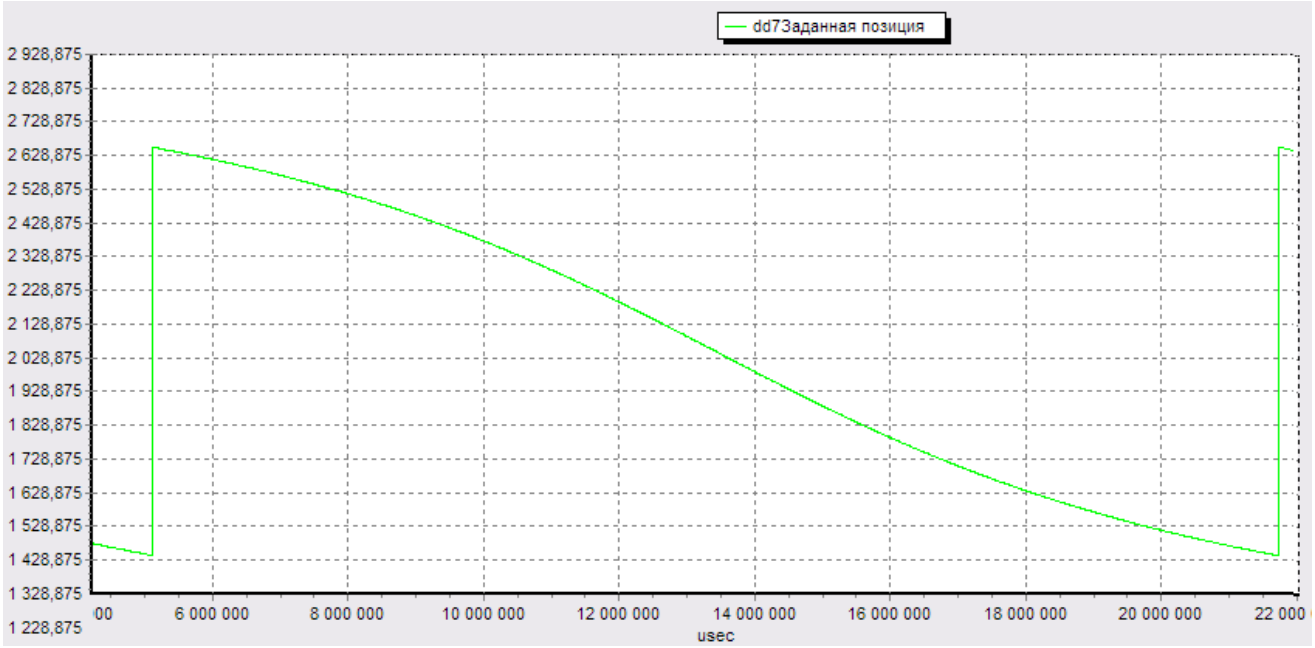

Рис. 9. Результат функции atan.

Извлечение квадратного корня sqrt, mag

<Переменная> = #sqrt(<Переменная>)

<Переменная> = #mag(<Переменная>,<Переменная>)

Функция sqrt выполняет извлечение квадратного корня переменной в формате IQ12.

Функция mag выполняет извлечение квадратного корня суммы переменных, предварительно возведя их в квадрат.

Функции работы со стеком push и pop

В ПЛК привода реализован стек для хранения 32х битных переменных. Размер стека рассчитан на 8 переменных. Схема работы стека FILO (First input last output).

Функция помещения в вершину стека:

#push(<Переменная>)

Функция извлечения переменной из вершины стека:

#pop(<Переменная>)

При переполнении стека программа будет остановлена и ПЛК привода будет переведено в состояние ошибка.

Прог. 31. Пример программы работы со стеком.

```
#push(X)                Помещение переменных в стек
#push(Y)
Выполнение различных
операций с использованием пе-
ременных X и Y
#pop(Y)                 Извлечение переменных из стека
#pop(X)
HALT
```

ПРИМЕЧАНИЕ:
Необходимо помнить о порядке извлечения переменных. Правильный порядок извлечения приведен в Прог. 31.

Технологический ПИД регулятор

Синтаксис переменных доступа к ПИД регулятору:

$pid_ref=<Переменная> - установить задание регулятора

$pid_fb=<Переменная> - установить значение обратной связи регулятора

<Переменная>=$pid_out – считывание результата расчета регулятора

Перед использованием ПИД регулятора необходимо выполнить его настройку с помощью Программы Мотомстер во вкладке Конфигурация. Параметры регулятора находятся в группе параметров pd.

Структура ПИД регулятора представлена на Рис. 10.

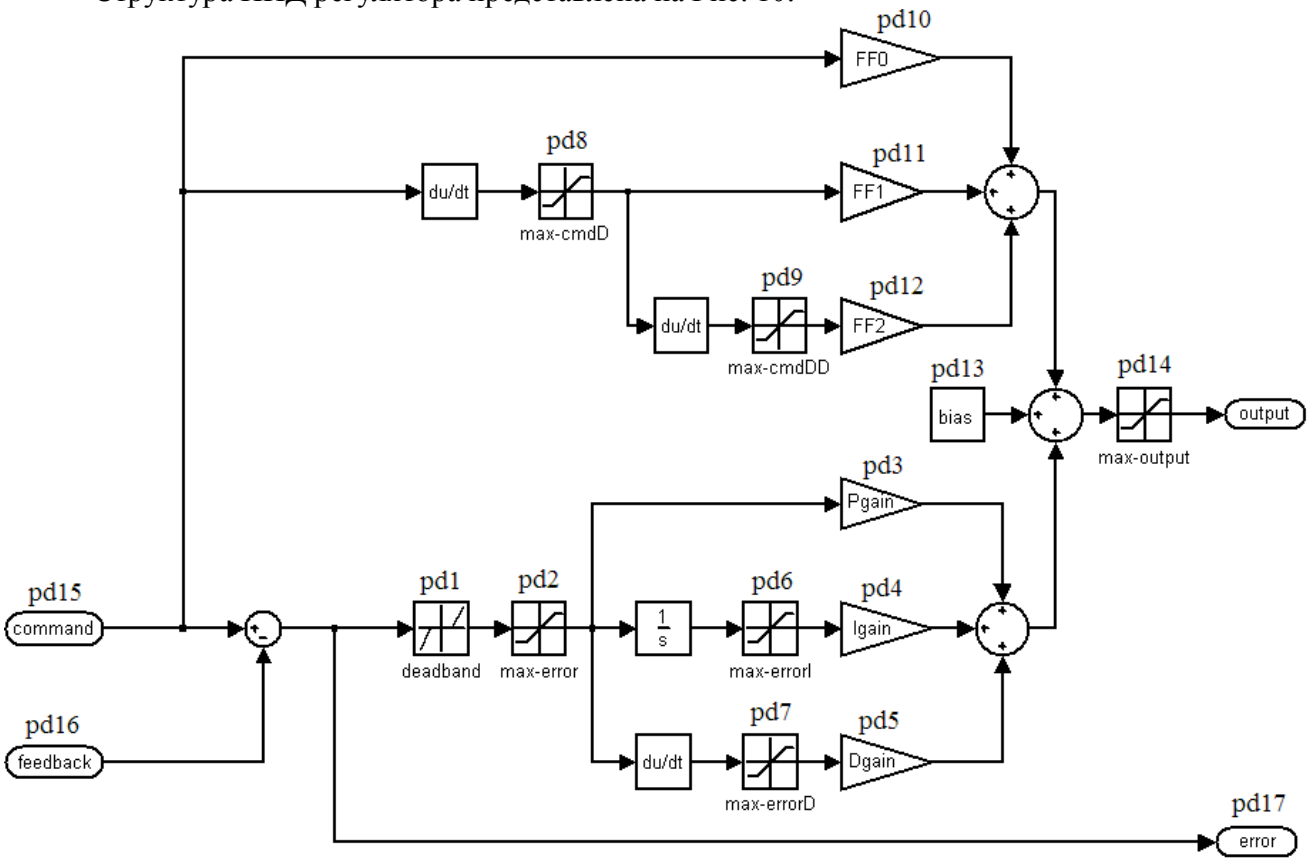

Рис. 10. Структура технологического ПИД регулятора.

Запуск функции расчета технологического ПИД регулятора, для обеспечения режима реального времени, осуществляется в обработчике прерывания контура тока. Таким образом, частота дискретизации ПИД регулятора равна частоте дискретизации контура тока.

Прог. 32. Пример программы использования технологического ПИД регулятора.

```
$pid_ref=iq(500)                Задание желаемой позиции
```

```
X=iq(10)          Инициализация коэффициента масштабирования
Y=$analog_inputA  Считывание аналогового входа, например, с датчика расстояния.
Y=#vartoiq(Y)     Переводим в IQ12
Y=# mpy(Y, X)     Масштабирование переменной Y
$pid_fb=Y         Установка ОС
Y=$pid_out        Захват результата расчета ПИД регулятора
P=#vartoint(Y)    Преобразовать переменную Y из дробного IQ12 в целое и уста-
                  новка задания позиции.

REPEAT
```

Функция rst_prog

Under construction

Сводная таблица переменных языка программирования

Параметр	Мин.	Макс.	Чтение	Запись	Описание	Примечание
P	-2^31	2^31-1	X	X	Положение	Чтение — текущей зиции Запись — задание позиции
W	-5000	5000	X	X	Скорость	Чтение — текущая скорость Запись — задание скорости или ограничение скорости (при установке в составе команды P)
A	0	32767		X	Ускорение	Только в режиме плавного разгона
X	-2^31	2^31-1	X	X	Переменная общего назначения	
Y	-2^31	2^31-1	X	X	Переменная общего назначения	
Z	-2^31	2^31-1	X	X	Переменная общего назначения	
PORT	0	15	X		Цифровой вход	
P_IN.n	0	1	X		Бит цифрового входа	Только проверка на 0 или 1
P_OUT.n	0	1	X	X	Бит цифрового выхода	Только проверка на 0 или 1 и установка в 0 или 1
D	0	32767		X	Задержка	
PCUR	-2^31	2^31-1		X	Текущее положение	
I	-cp3	cp3	X		Текущий момент	

$eeprom_mas	-2^31	2^31-1	X	X	Текущий элемент массива энергонезависимой памяти данных	Указатель при чтении или записи смещается на следующий элемент
$eeprom_mas_i	0	6300	X	X	Индекс массива энергонезависимой памяти данных	Изменяется при чтении $eeprom_mas
$analog_input_A	-32000	32000	X		Аналоговый вход 0	
$analog_input_B	-32000	32000	X		Аналоговый вход 1	
$SD_counter	-2^31-1	2^31-1	X	X	Счетчик Step/Dir	
$drv_status	0	9	X	X	Статус привода	Можно записать только 0 или 1
$pid_ref	-2^31	2^31-1		X	Задание технологического ПИД регулятора	
$pid_fb	-2^31	2^31-1		X	ОС технологического ПИД регулятора	
$pid_out	-2^31	2^31-1	X		Выход технологического ПИД регулятора	

Сводная таблица переменных функций программирования

Функция	**Возврат**	**Параметр 1**	**Параметр 2**	**Описание**
p_limit_r		Позиция ограничения		Установка ограничения положительного
p_limit_l		Позиция ограничения		Установка ограничения отрицательного
rst_pos_i		0 - против часовой, 1 - по часовой		Поиск z метки
crash_port		Биты 0 и 1 - номер порта, Бит 2 - уровень активного сигнала		Активизировать порт аварийного останова
send_usb		Параметр 32 бита		

rst_prog				Перезапуск текущей программы
sin		Переменная 32 бита (IQ12)		Синус угла
cos		Переменная 32 бита (IQ12)		Косинус угла
atan		Переменная 32 бита (IQ12)	Переменная 32 бита (IQ12)	Арктангенс(Y, X)
sqrt		Переменная 32 бита (IQ12)		Извлечение квадратного корня
vartoiq		Переменная 32 бита (IQ12)		Функция преобразования дробного числа, хранящегося в переменной, в формат IQ12
vartoint		Переменная 32 бита (IQ12)		
frac		Переменная 32 бита (IQ12)		Выделение дробной части числа
mag		32х битная переменная	32х битная переменная	Извлечение квадратного корня из квадратов параметров
mpy	Результата умножения (IQ12)	Множитель 1 (IQ12)	Множитель 2 (IQ12)	Умножение дробных чисел
div	Результата деления (IQ12)	Числитель (IQ12)	Знаменатель(IQ12)	Деление дробных чисел
push		Переменная 32 бита		Помещение значения переменной в вершину пользовательского стека
pop		Переменная 32 бита		Извлечение значения переменной из вершины пользовательского стека