15. Практическое программирование на языке FBD

Данная глава раскрывает некоторые специфические приемы программирования на языке FBD.

Лингвистические языки программирования, такие как **Си**, **Паскаль** и т.п. имеют в своем составе часть операторов, аналогов которых нет в языке FBD, что вызывает определенные трудности при освоении языка у специалистов в области программирования на языках высокого уровня. К этим операторам относятся операторы циклов, операторы ветвления программы типа CASE, системные запросы и т.п. Кроме этого в языке FBD отсутствует механизм прерываний, привычный для классического программирования многозадачных систем.

Рассмотрим, как в системе исполнения реализуется режим многозадачности. Работа системы исполнения имеет циклический характер. Весь контролируемый технологический процесс разбивается на ряд формальных независимых задач, каждая из которых обслуживается отдельной программой. Программы выполняются поочередно с одинаковым приоритетом. Если время одного цикла системы исполнения существенно меньше скорости изменения реального процесса, можно считать, что все программы выполняются параллельно, т.е. одновременно.

Идеология программирования на языке FBD подразумевает, что время исполнения каждой программы должно быть вполне определенным, т.е. детерминированным. Другими словами ни одна программа не имеет права зациклиться на неопределенное время, например на ожидании какого-либо события. Данное правило является “хорошим тоном” в программировании многозадачных процессов и гарантирует, что никакая программа не приостановит исполнение других программ. Такой подход обеспечивает простоту и прозрачность для понимания. Так как нет прерываний, система полностью детерминирована. Задачи вызываются в одной и той же последовательности, что позволяет достаточно просто произвести анализ «наихудшего случая» и вычислить максимальную задержку.

Рассмотрим некоторые приемы программирования на языке FBD.

15.1 Установка функциональных блоков в программе

При установке функциональных блоков необходимо помнить правило, что очередность выполнения блоков в программе: сверху-вниз, слева-направо, а точка привязки - левый верхний угол блока.

Типичной ошибкой при установке блоков является то, что программу на языке FBD отождествляют с электрической схемой, забывая о порядке выполнения блоков. Например, необходимо сложить переменные **A**, **B** (тип FLOAT), привести сумму к типу INTEGER, сравнить с переменной **C** (тип INTEGER) и результат поместить в переменную **D**. В программе на рис.15-1 алгоритм реализован неправильно. Блок EQU будет выполнен последним, т.к. его точка привязки расположена ниже точки привязки блока CMP (порядок выполнения блоков указан цифрами внутри блоков).

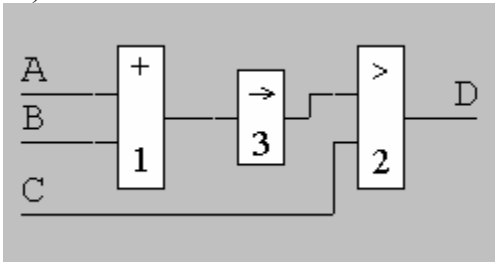
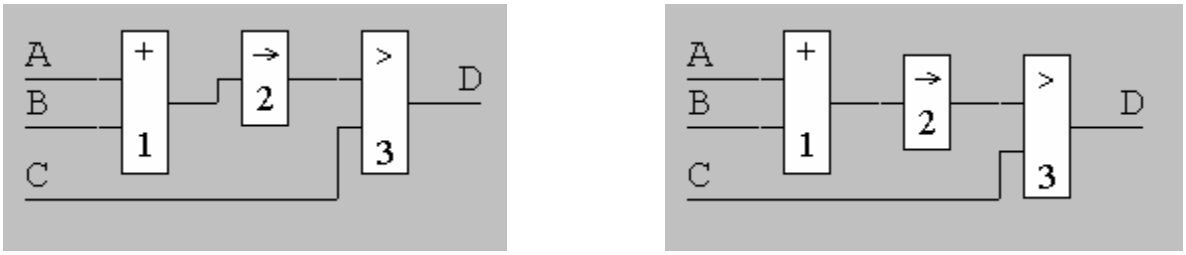


Рис. 15-1 Неправильная установка блоков.

На рис.15-2 приведены примеры правильной реализации алгоритма.



а) б)

Рис. 15-2 Варианты правильной установки блоков.

На рис.15-3 приведен пример программы детектора фронтов логического сигнала. Выход **Q** будет принимать значение TRUE на один цикл системы исполнения, при изменении состояния переменной **IN** на противоположное. То есть на выходе **Q** будет формироваться импульс длительностью в один цикл контроллера при изменении полярности входного сигнала.

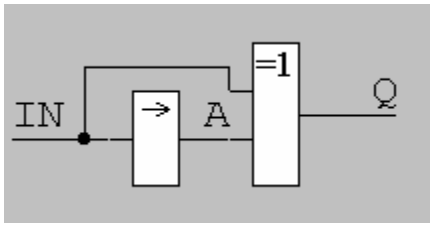


Рис.15-3 Детектор фронта

Подробно рассмотрим работу программы. Начальное значение всех переменных равно FALSE. Первым выполняется блок XOR, так как его точка привязки выше, чем у блока EQU. Значения переменных **IN** и **A** на входа блока XOR равны FALSE, поэтому на выходе **Q** будет FALSE. После выполнения блока EQU значения переменных не меняются. Так будет до тех пор, пока значение переменной **IN** не изменится. Если переменная **IN** станет равна TRUE, то на выходе блока XOR будет TRUE, так как **IN**=TRUE, **A**=FALSE. После блока XOR выполнится блок EQU, который присвоит переменной **A** значение TRUE. В следующем цикле на входах блока XOR обе переменные будут равны TRUE и значение переменной **Q** станет равно FALSE. Аналогично программа будет работать при изменении значения переменной **IN** из состояния TRUE в FALSE.

На рис.15-4 приведена диаграмма работы программы. Следует обратить внимание на то, что если сигнал **IN** меняет свое состояние в каждом цикле, то на выходе **Q** будет состояние TRUE.

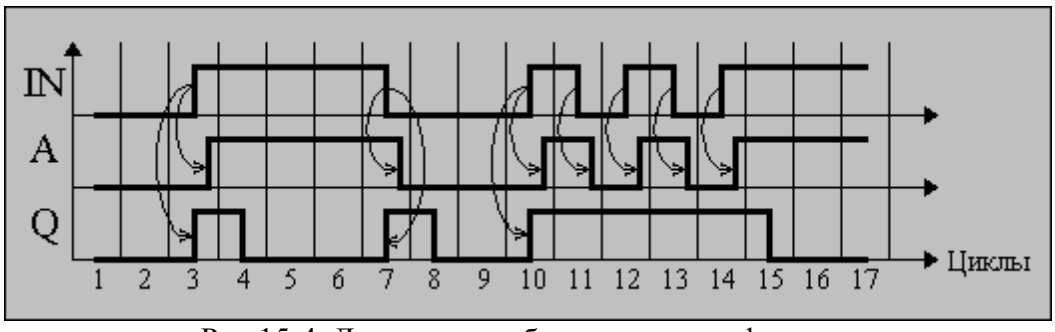


Рис.15-4. Диаграмма работы детектора фронтов

15.2 Секция инициализации

Используя особенности порядка исполнения блоков, можно легко организовать в программе секцию инициализации, т.е. такую секцию, которая выполняется один раз при первом проходе. Пример приведен на рис.15-6.

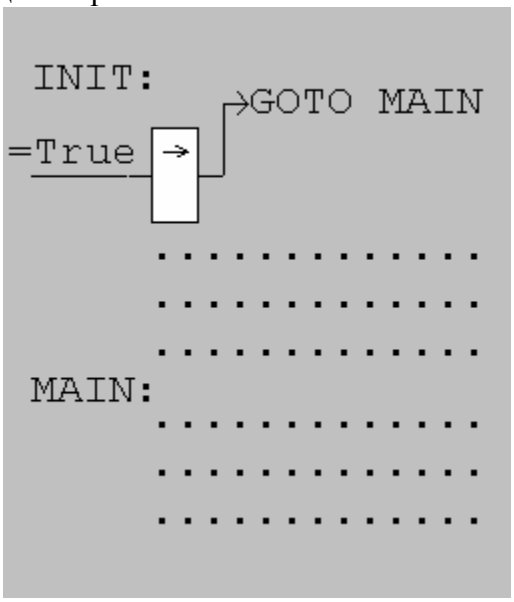


Рис.15-6

Оператор GOTO расположен выше блока EQU, поэтому выполняется первым. При первом проходе программы, значение переменной на входе оператора GOTO равно FALSE (значение по умолчанию), поэтому переход на метку MAIN не выполнится и будет исполнена секция между метками INIT и MAIN. На втором и последующих циклах значение переменной на входе GOTO будет равно TRUE и выполнится переход на метку MAIN (т.е. будет пропущена секция инициализации).

15.3 Ветвление по нескольким условиям

Рассмотрим пример программы в которой выполняются различные процедуры, в зависимости от значения переменной SW, т.е. эквивалент оператора **`Switch`** в Си или **`Case`** в Паскале.

Фрагмент программы на языке Паскаль:

```
case SW of
1: Prog1;
2: Prog2;
3: Prog3;
end;
```

Программа на языке FBD будет выглядеть, как показано на рис.15-7.

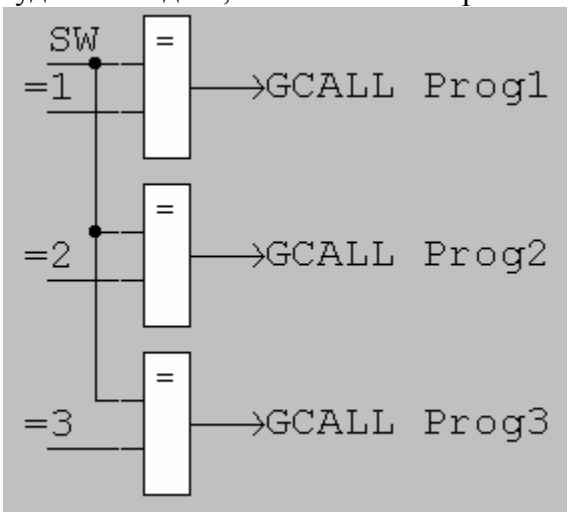


Рис.15-7

Для удобства применения, данный фрагмент программы можно свернуть в библиотечный блок. Тогда программа будет выглядеть следующим образом:

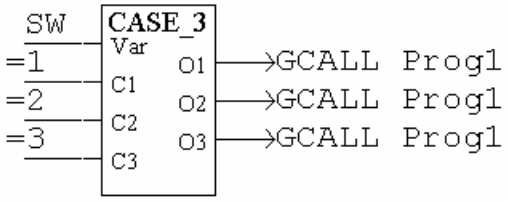


Рис.15-8

Если процедуры расположены внутри данной программы, то вместо GCALL необходимо использовать операторы GOTO, как показано на рис.15-9.

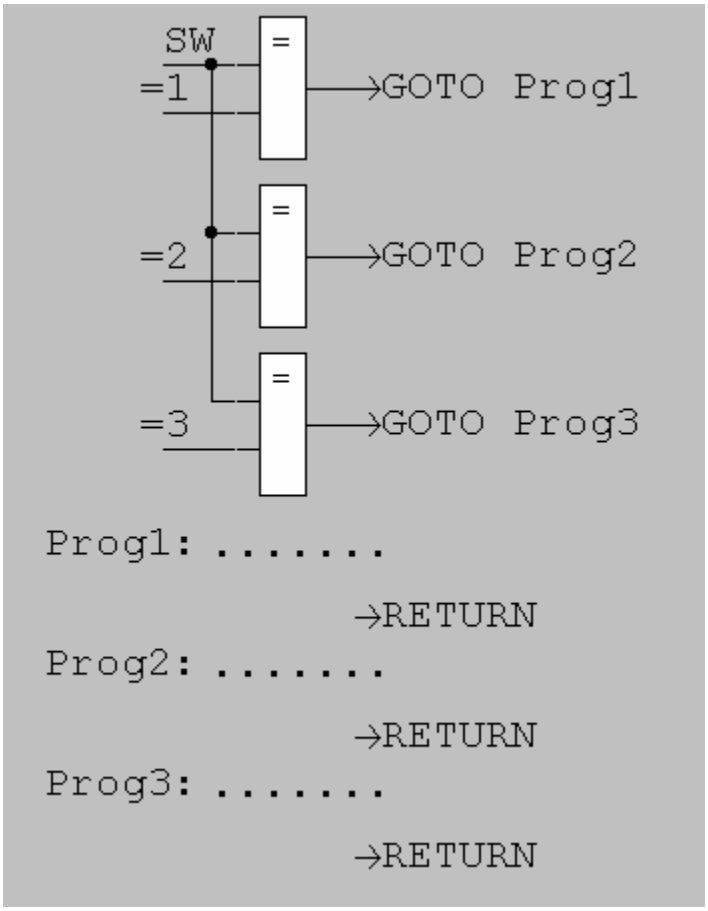


Рис.15-9

15.4 Циклы

Существует два принципиальных подхода к организации циклов:

а) организация “непрозрачного” цикла;

б) организация “прозрачного” цикла.

Во время “прозрачного” цикла в данной программе другие программы системы исполнения также выполняются. Во время “непрозрачного” цикла - ожидают его окончания.

Рассмотрим эти подходы на следующих примерах:

Фрагмент программы на языке СИ.

```
while(i==false)
{  a+=a; //тело цикла
}
```

Цикл выполняется, пока переменная **i** (тип BOOLEAN) имеет значение **false.**

Эквиваленты программы на языке FBD

а) “Непрозрачный” цикл

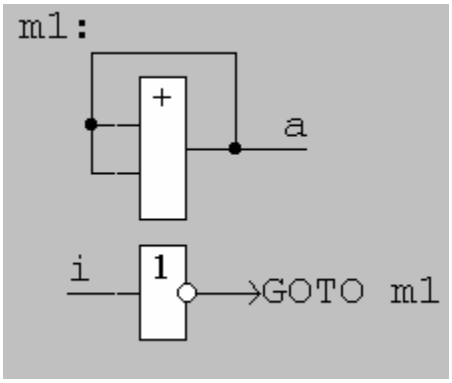


Рис.15-10

Программа будет исполняться непрерывно за один цикл контроллера до тех пор, пока переменная **i** не примет значение **true**. Все остальные программы системы исполнения будут ожидать окончания цикла.

б) “Прозрачный” цикл

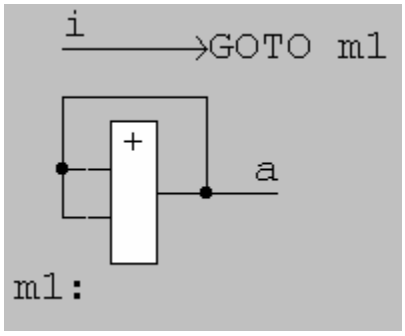


Рис.15-11

Программа будет исполняться один раз в полном цикле системы исполнения, пока переменная **i** имеет значение **false**. Выполнение других программ не зависит от окончания цикла.

Очевидно, что подход реализованный в примере “б”, не мешает работе других программ и является предпочтительным. Напротив, организация “непрозрачных” циклов может существенно снизить быстродействие системы исполнения и привести к потере управления процессом.

15.5 Ожидание события

Задача ожидания события также организуется как “прозрачный” цикл. Условием выполнения программы является наступления события. На рис.15-12 приведен пример программы, ожидающей событие, когда переменная **i** примет значение 5.

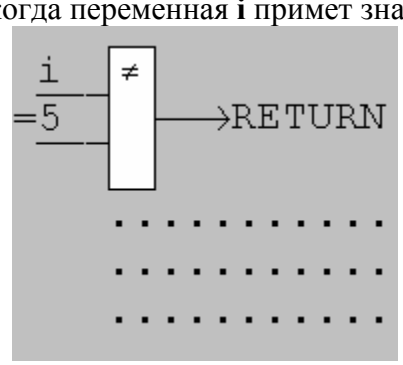


Рис.15-12

Таким же образом можно решить задачу выполнения программы через строго определенные промежутки времени. На рис.15-13 приведен пример программы, которая выполняется один раз в 10 секунд.

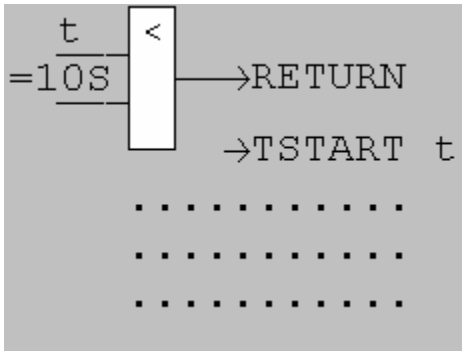
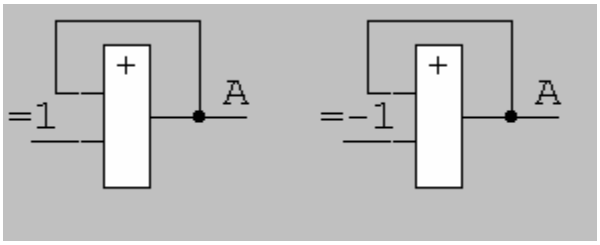


Рис.15-13

Пока таймерная переменная **t** меньше 10S, исполняется оператор RETURN и выполнение тела программы не происходит. При достижении переменной **t** значения 10S оператор RETURN игнорируется и программа выполняется полностью. Обратите внимание, что сразу после блока CMP стоит оператор TSTART. Чтобы обеспечить правильные промежутки времени между выполнениями программы, нужно инициализировать таймерную переменную **t** сразу после блока сравнения, а не в конце тела программы.

15.6 Подсчет событий

Для реализации счетчика событий достаточно блока ADD. На нем можно реализовать схему, ведущую счет как в положительную, так и в отрицательную стороны с произвольным шагом (Рис.15-14).



а) инкремент б) декремент

Рис.15-14

Эти функции можно реализовать и блоком SUB, а также с переменным шагом, если вместо константы применить переменную. Для примера можно посмотреть на реализацию библиотечного блока CPS (Библиотека LIB1-Разное). Значение переменной **CUR** увеличивается на 1 в каждом цикле системы исполнения, а раз в секунду копируется в переменную **CPS**. По величине **CPS** можно судить о производительности системы исполнения.

В примере, приведенном на рис.15-15, переменная **S** увеличивает свое значение на +1, когда переменная **tick** изменит свое состояние из FALSE в TRUE. Переменная **M** увеличивает свое значение на +1, когда значение переменной **S** достигнет значения 60. По умолчанию все переменные равны нулю.

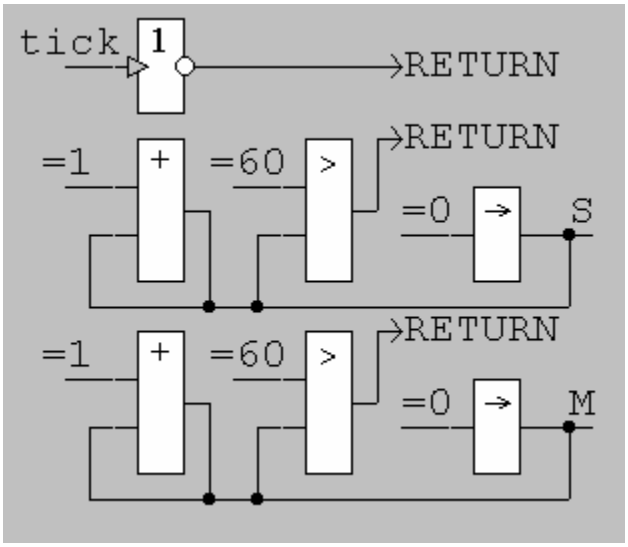
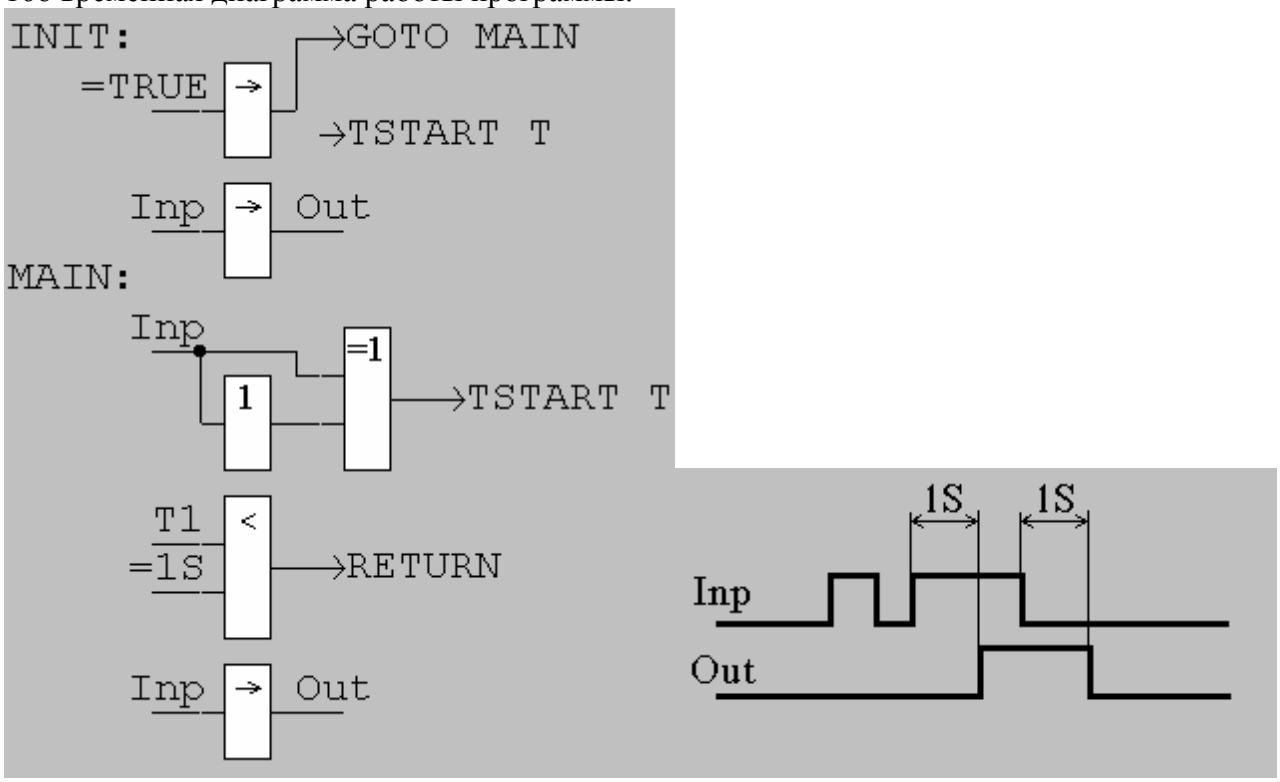


Рис.15-15

15.7 Таймеры

Таймерные переменные являются простым инструментальным средством, позволяющим организовать счет времени, элементы задержки, селекторы длительности и т.п. Таймерную переменную можно сравнить с секундомером, который запускается командой TSTART, а останавливается командой TSTOP. Количество таймерных переменных, как и переменных других типов, ограничено лишь ресурсами системы исполнения.

Рассмотрим несколько примеров использования таймерных переменных. На рис.15-16а приведена программа, в которой реализован селектор импульсов по длительности, а на рис.15-16б временная диаграмма работы программы.



а) б)

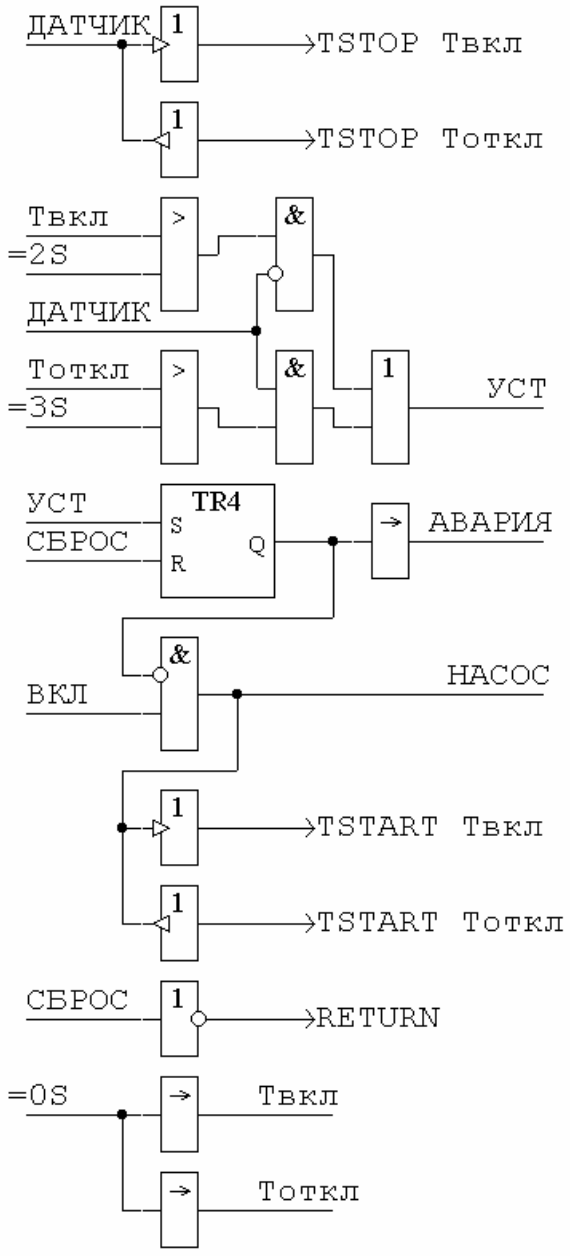
Рис.15-16

Переменная **Out** примет значение переменной **Inp** только в том случае, если переменная **Inp** не изменит свое состояние в течение 1S. В секции INIT происходит начальное зада-

ние значения переменной **Out** (на выход копируется состояние входа) и инициализируется таймерная переменная **T**. Далее, в секции MAIN любое изменение значение **Inp** приводит к перезапуску таймерной переменной **T**. Значение переменной **T** сравнивается с константой =1S. Если **T** больше 1S, то значение переменной **Inp** копируется **в Out**. Таким образом осуществляется “подавление дребезга” сигнала **Inp**. Время селекции можно сделать любым, если вместо константы =1S использовать таймерную переменную или другую константу.

В библиотеке REGULATION элемент HYST реализован по аналогичному алгоритму и предназначен для исключения случайных сбоев от помех или дребезга контактов у дискретных датчиков.

В примере на рис.15-17 приведена программа управления насосом. Таймерные переменные (**Твкл**, **Тоткл**) используются для определения контрольных промежутков времени между сигналом на включение насоса (**НАСОС**) и приходом сигнала подтверждения включения от датчика обратной связи (**ДАТЧИК**). Это блок включен в библиотеку “LIB1-Разное” под названием PUMP.



После подачи команды на включение насоса (**ВКЛ**=TRUE) и если нет аварии (**АВАРИЯ**=FALSE), устанавливается сигнал включения насоса (**НАСОС**=TRUE). Одновременно запускается таймерная переменная **Твкл**, которая измеряет интервал времени между включением насоса и приходом сигнала от датчика обратной связи.

За 2S от датчика должен прийти ответ (**ДАТЧИК**=TRUE), что насос включился и работает нормально. Если за 2S ответ от датчика не пришел, то устанавливается признак аварии (**АВАРИЯ**=TRUE), и сигнал на управление насосом отключается (**НАСОС**=FALSE). Сигнал **АВАРИЯ** является триггерным и сбрасывается сигналом **СБРОС**.

Аналогично схема работает и при отключении насоса. После подачи команды на отключение насоса (**ВКЛ**=FALSE), выключается сигнал управления насосом (**НАСОС**=FALSE) и одновременно запускается таймерная переменная **Тоткл**, которая измеряет интервал времени между выключением насоса и приходом сигнала от датчика обратной связи.

За 3S от датчика должен прийти ответ (**ДАТЧИК**=FALSE), что насос выключился. Если за 3S ответ от датчика не пришел, то устанавливается признак аварии (**АВАРИЯ**=TRUE).

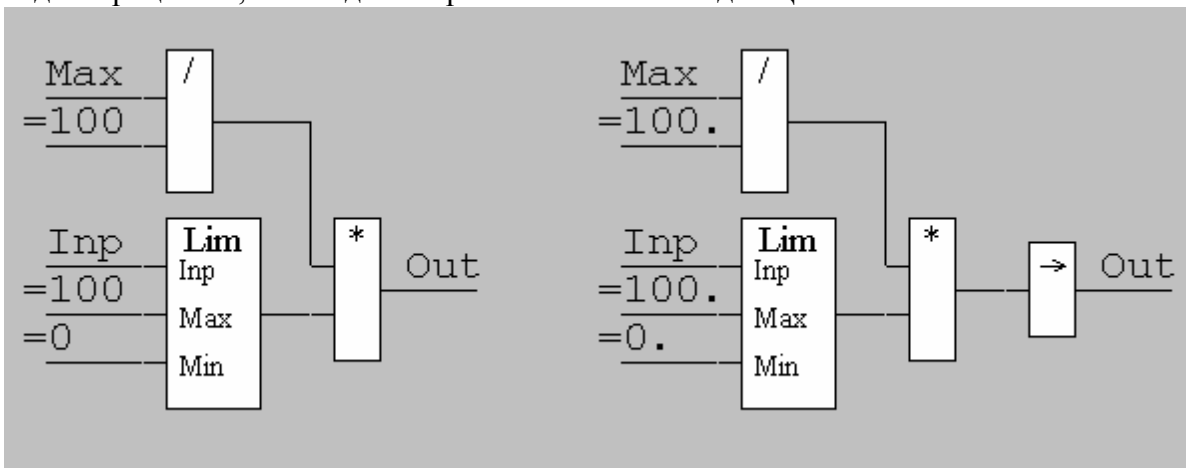
Рис. 15-17

В качестве примеров использования таймерных переменных можно ознакомиться с описанием FBD библиотеки “FREQTIME-Частота и Время” в книге 2 гл. 1.2 настоящего Руководства Пользователя. В ней приведено много примеров различных способов измерений интервалов времени и частоты.

15.8 Нормализаторы и преобразователи сигналов

Задача преобразования значений переменных по определенным законам является типовой в прикладных программах. Преобразование требуется для приведения соответствия измеренного кода с величиной физического параметра, приведения размерности, калибровки измерительных каналов и т.п. Ниже рассмотрены некоторые характерные примеры таких преобразований.

На рис.15-18 приведен пример программы преобразования для управления приводом с цифро-аналоговым преобразователем (ЦАП). Сигнал управления **Inp** выражает положение привода в процентах, а выходная переменная **Out -** в кодах ЦАП.



а) б)

Рис.15-18

Значение переменной **MAX** задает максимальный код ЦАП. На выходе блока DIV будет значение количества кодов ЦАП на 1%. Блок LIM ограничивает входной сигнал диапазоном от 0% до 100%. Значение переменной **Out**, выраженное в кодах ЦАП, будет эквивалентно значению **Inp**, выраженному в процентах.

Пример на рис.15-18а приведен для переменных типа INTEGER. Однако в некоторых случаях целочисленные вычисления могут привести к потере точности из-за погрешности округления. В этом случае все вычисления производятся с переменными типа FLOAT, а затем результат приводится к типу INTEGER (рис 15-18б).

На рис.15-19 приведен пример программы преобразования значения сигнала, которое приходит в кодах АЦП, в физическую величину измеряемую датчиком. В примере приведены данные модуля 5710 Analog I/O Card и датчика температуры AD22100. Датчик выдает напряжение пропорционально температуре. При изменении температуры на +1°C напряжение изменяется на +22.5 мВ. Температуре 0°C соответствует напряжение 1.375В. Диапазон датчика от -50°C до +150°C. Модуль 5710 имеет три коэффициента усиления (x1, x10, x100), диапазон входного напряжения 10В и 12-ти разрядный АЦП.

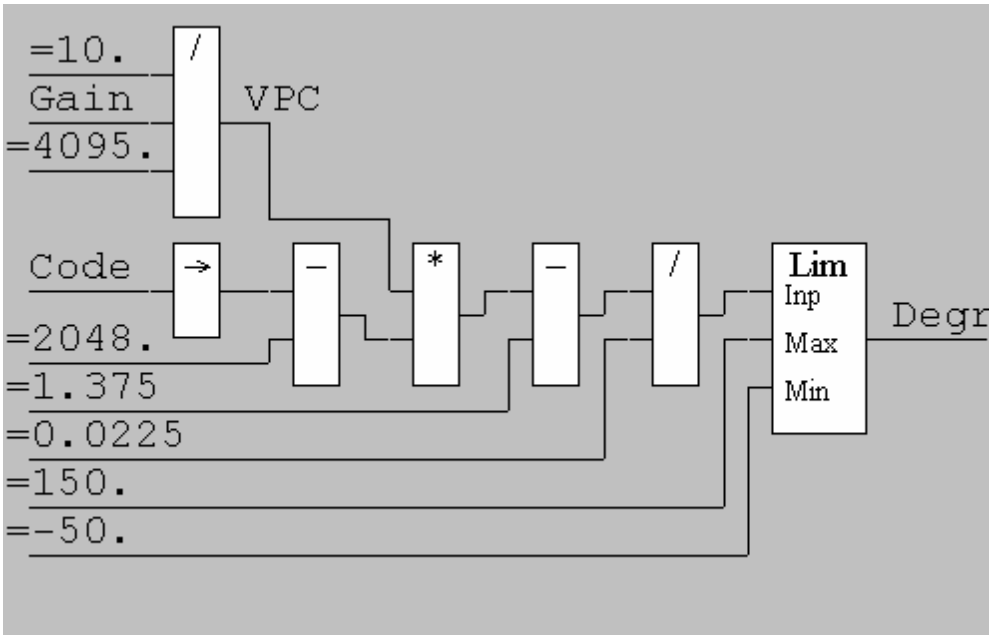


Рис.15-19

Формула преобразования:

```
Degr=((Code-2048)*(MaxV/Gain/MaxC)-1.375)/dC, где
```

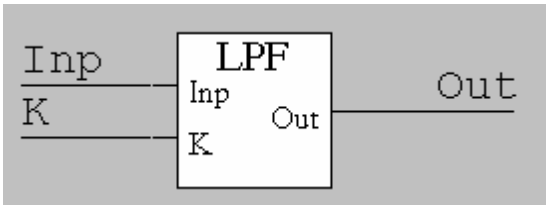
Degr значение измеренной температуры в °C.
Code код сигнала измеренного АЦП.
2048 поправка для смещения диапазона АЦП в пределы от -2048...+2047.
MaxV диапазон входного сигнала АЦП (10 В).
Gain коэффициент усиления модуля 5710 (1, 10 или 100).
MaxC максимальный код АЦП (4095).
1.375 напряжение на датчике при 0°C.
dC приращение напряжение на датчике на 1°C (0.0225 В/°C).

После расчета температуры по приведенной формуле, значение переменной Degr ограничивается диапазоном от -50 до +150, т.е. областью достоверных показаний датчика температуры AD22100.

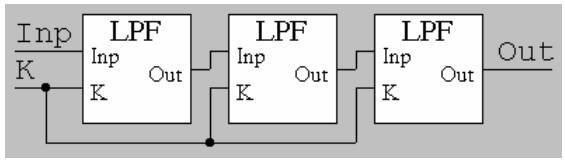
15.9 Фильтрация сигналов

Фильтрация сигналов является из наиболее важных задач, решаемых в большинстве измерительных систем. Реальные сигналы от объектов содержат флюктуации в виде шума, сетевых наводок, случайных выбросов.

Для фильтрации высокочастотной помехи применяется ФНЧ (Рис.15-20а). Значение переменной **K** должно быть в пределах]0,1[и определяет частоту среза фильтра. Чем меньше значение переменной **K**, тем ниже частота среза. Для получения более крутой характеристики, фильтры нужно соединить последовательно (рис.15-20б). Подробное описание блока LPF находится в FBD библиотеке Emulator.



а)



б)

Рис.15-20

Для устранения помехи, имеющей нормальное распределение, в качестве фильтра, можно применить блок AVR, который вычисляет среднее арифметическое выборки. Чем больше размер выборки, тем меньше влияние помехи. Однако, следует иметь ввиду, что с увеличением размера выборки, увеличивается время установления выходного значения фильтра **Out**. Поэтому для быстропеременных процессов размер выборки не следует делать слишком большим (Рис.15-21).

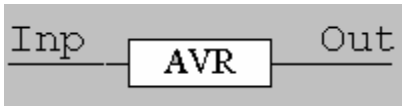


Рис.15-21

Для борьбы со случайной помехой можно использовать различные варианты алгоритмов. Самый простой способ состоит в использовании блока LIM, чтобы исключить из расчетов заведомо ложные показания, когда измеренное значение сигнала выходит за границы диапазона измерения датчика (Рис.15-22).

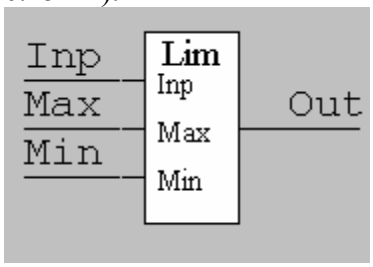


Рис.15-22

Если известен характер поведения измеряемого сигнала, например сигнал не может быстро измениться за короткий промежуток времени, то можно применять различные ограничители скорости нарастания сигналов. На рис 15-23 приведен пример линейного ограничителя. Переменная **Out** стремится к значению переменной **Inp**, но за 1S не может изменить свое значение больше чем на 15. Блок LinLIM находится в FBD библиотеке Regulation-Регуляторы.

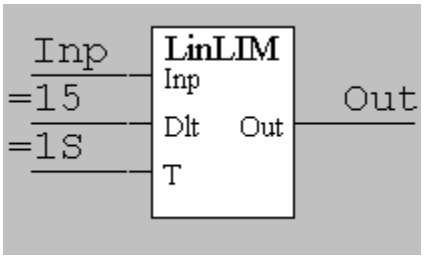


Рис.15-23

На рис 15-24 приведен пример логарифмического ограничителя скорости нарастания сигнала. Переменная **Out** стремится к значению переменной **Inp**, но за 1S не может изменить свое значение больше чем на |**Out**-**Inp**|*0.1. Величина приращения выходного сигнала **Out** определяется динамически, то есть шаг приращения зависит от величины рассогласования входного и выходного значения. Блок LogLIM находится в FBD библиотеке Regulation-Регуляторы.

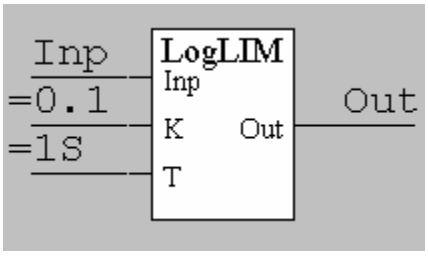


Рис.15-24

Подробно работа блоков LPF, LinLIM и LogLIM описана в книге 2, гл.1.3 Руководства Пользователя.

Для лучшей фильтрации можно применять комбинированные фильтры. Например сначала определить диапазон входного сигнала блоком Lim, потом ограничить скорость изменения блоком LinLIM или LogLIM, а затем усреднить выборку блоком AVR (Рис 15-25).

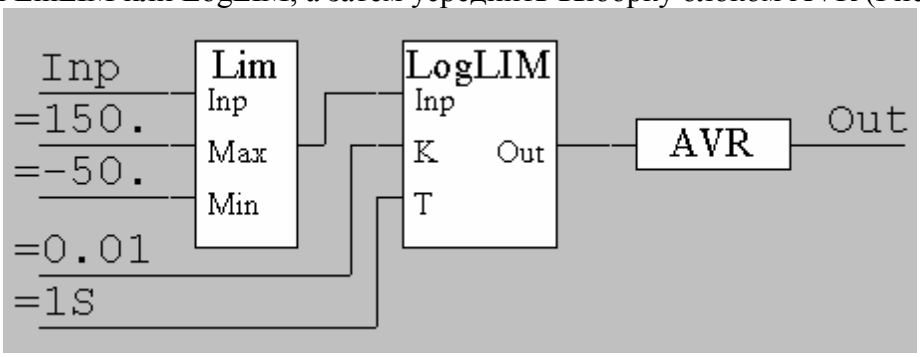


Рис.15-25

15.10 Регуляторы

Назначение регуляторов состоит в том, чтобы устанавливать и поддерживать на заданном уровне W (задающий параметр) определенную физическую величину X (регулируемую величину). Блок-схема простого контура регулирования представлена на рис.15-26

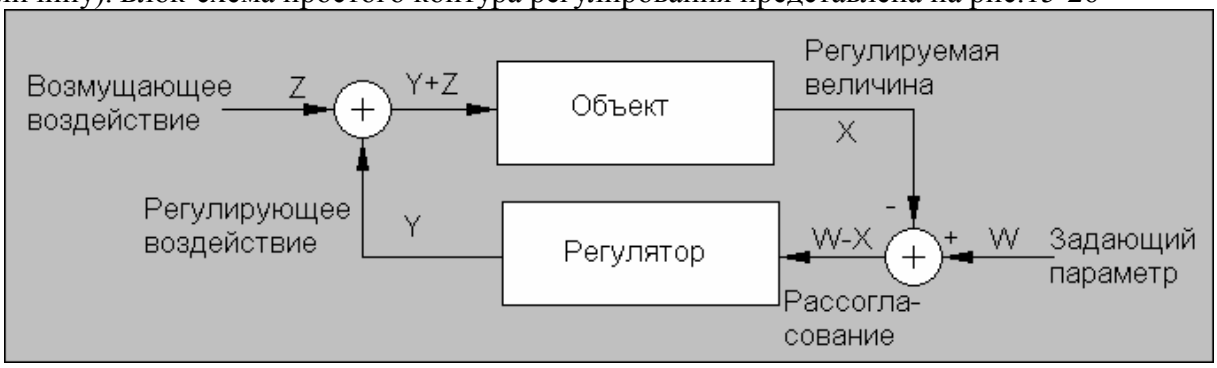


Рис.15-26

Регулятор влияет на регулируемую величину X с помощью регулирующего воздействия Y так, чтобы рассогласование регулирования W-X было возможно меньшим. Воздействующее на объект регулирования возмущение формально можно представить величиной помехи Z, адитивно накладывающейся на заданный параметр. Простейшим примером такого регулятора является П-регулятор (рис.15-27)

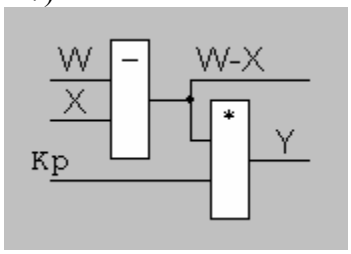


Рис.15-27

Если регулируемая величина X не равна заданному значению W, появляется рассогласование W-X. Благодаря этому регулирующее воздействие Y изменяется на величину (W-X)*Kp. Это изменение компенсирует разность W-X. В установившемся режиме остаточное рассогласование будет меньше при большем коэффициенте пропорциональности регулятора Kp. Однако следует учитывать, что коэффициент усиления в цепи обратной связи не может быть сколь угодно большим, так как сдвиг фазы в контуре регулирования приведет к возник-

новению колебаний. Чтобы обеспечить возможно меньшую погрешность регулирования и нужную переходную характеристику, к П-регулятору добавляют интегратор и дифференциатор, получая ПИ- или ПИД-регулятор. Подробное описание работы регуляторов приведено в книге 2, гл. 1.3 настоящего Руководства Пользователя.

15.11 Моделирование объектов управления

Практическая отладка программы управления сложным объектом является достаточно трудоемким процессом и сопряжена с опасностью непредвиденной реакции реального объекта на управляющие воздействия. Избежать подобной ситуации, а также произвести предварительную отладку программы с подбором управляющих воздействий, позволяет моделирование объекта управления.

Модель объекта управления представляет из себя отдельную программу или набор программ. Эти программы принимают от управляющей программы сигналы управления в виде переменных и эмулируют поведение объекта в виде детерминированной реакции переменных процесса, как заданные функциональные зависимости. Функциональная зависимость поведения переменных процесса определяется разработчиком. Чем точнее и детальнее описана эта функциональная зависимость, тем точнее модель эмулирует реальный процесс. Программа эмуляции объекта вставляется в список программ системы исполнения (на этапе отладки) и работает в цикле с программой регулирования.

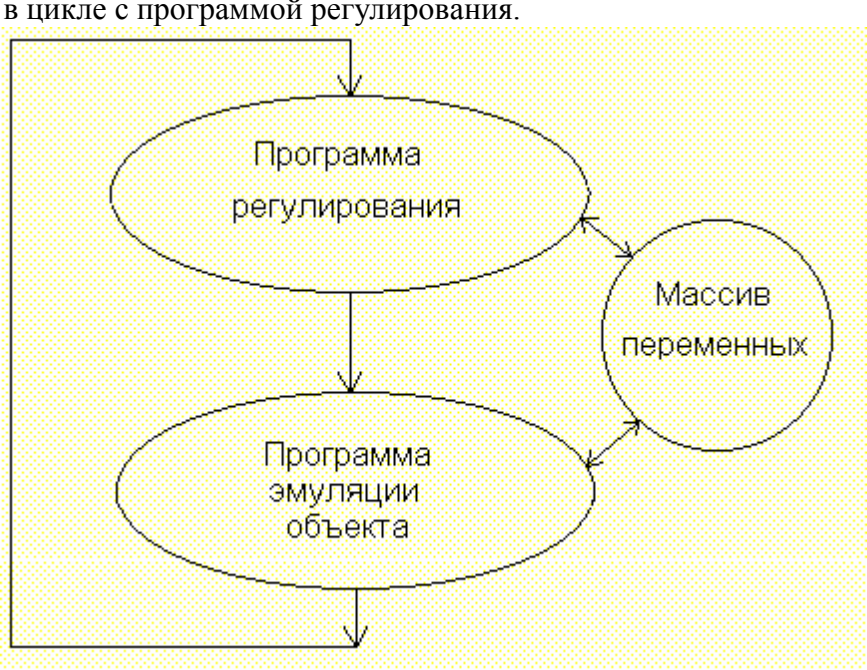


Рис.15- 28. Взаимодействие программ управления и эмуляции объекта

Рассмотрим модель абстрактного процесса в виде резервуара, уровень жидкости в котором (значение параметра регулирования) определяется соотношением объема втекающей и вытекающей из него жидкости.

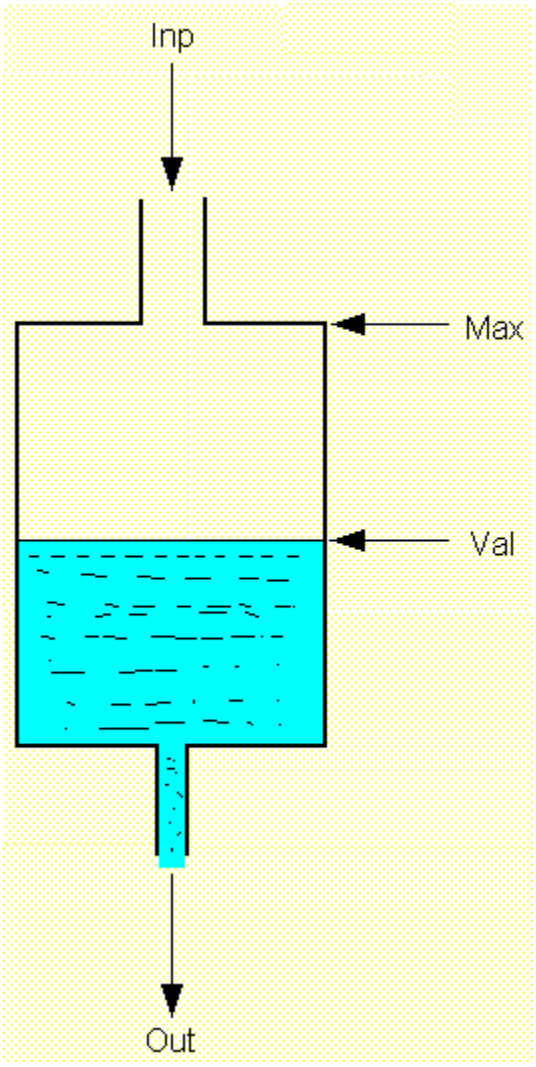


Рис.15- 29

Обозначим количество воды, втекающее в резервуар за единицу времени - **Inp**, а количество воды, вытекающее из резервуара в единицу времени - **Out**. Уровень воды в резервуаре - **Val**, максимально возможный уровень **Max**.
Программа, реализующая данную модель объекта, представлена на рис.15-30.

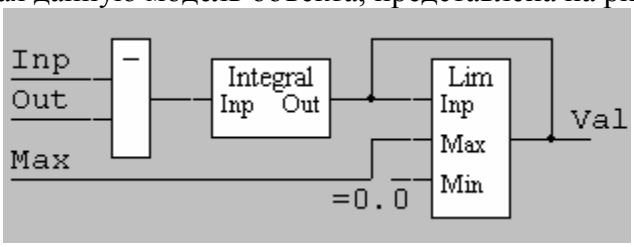


Рис.15-30. Программа-модель резервуара

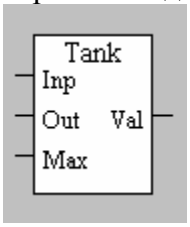


Рис.15-31. Программа-модель резервуара в виде библиотечного блока

Реальные объекты всегда имеют фазовый сдвиг между регулирующим воздействием и реакцией объекта, обусловленный инерционностью физических процессов. Смоделируем этот сдвиг посредством ограничителя скорости нарастания сигнала, имеющего логарифмическую характеристику.

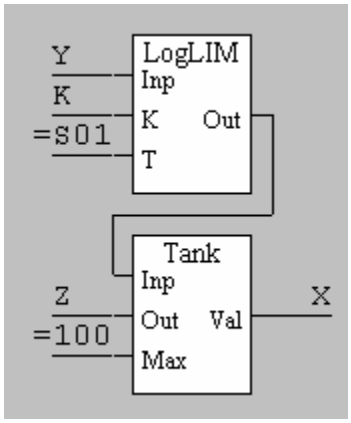


Рис.15-32. Программа модели объекта

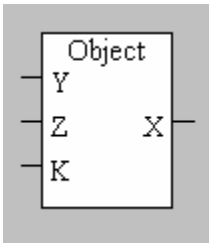


Рис.15-33. Модель объекта в виде библиотечного блока

Таким образом мы получили обобщенную модель объекта, где:

Y - Регулирующее воздействие
Z - Внешнее возмущение
X - Состояние объекта (Обратная связь)
K - Коэффициент реакции объекта

Данная модель с определенной степенью приближения эмулирует такие физические процессы, как изменение температуры объекта, давления, объема и некоторые другие. Так в случае моделирования процесса нагревания, Y - количество тепла, передаваемое объекту, Z - тепловые потери, K- теплоемкость объекта.

Предположим, необходимо исследовать поведение программы стабилизации параметра, представленной на рис.15-34.

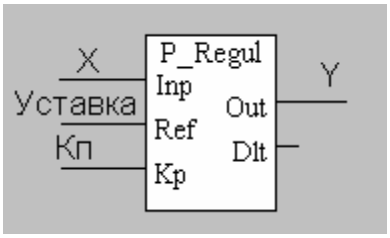


Рис.15-34

Для этого исследуемая программа должна быть замкнута с программой модели через общие глобальные переменные следующим образом (рис.15-35).

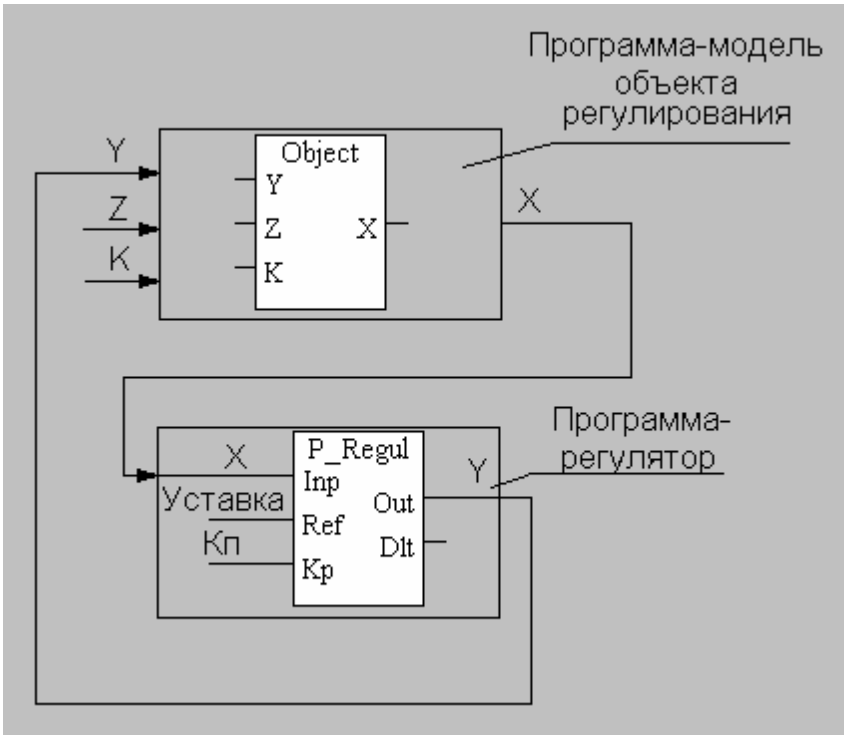


Рис.15-35

Контур регулирования состоит из объекта регулирования и регулятора. На объект воздействует два фактора: управляющее действие регулятора и случайная внешняя помеха, которая вносит искажения в управляющий сигнал. От объекта к регулятору идет обратная связь, характеризующая состояние объекта. Коэффициент K определяет время реакции объекта на управляющее воздействие.

Примеры эмуляторов объектов находятся в библиотеке **Emulator**. В демонстрационном проекте Regul.plc приведен пример программы регулирования, замкнутой с программой эмуляции объекта. Данный проект может быть использован в целях обучения, а также исследования влияния коэффициентов ПИД-регулятора на переходную характеристику объекта регулирования.