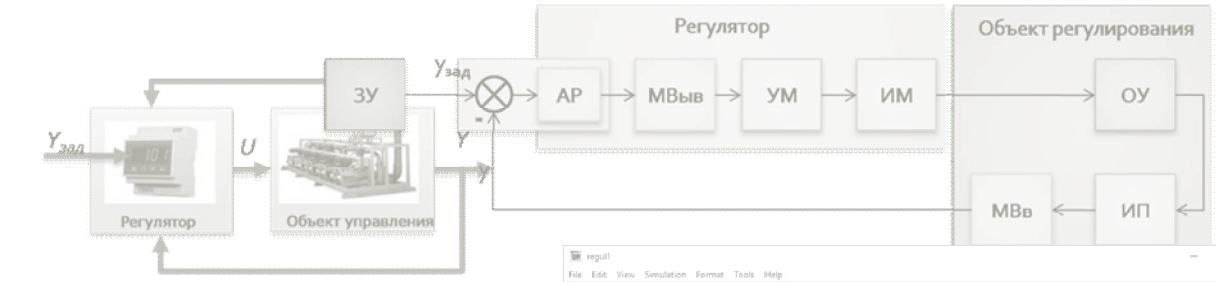
Министерство образования и науки Российской Федерации
Амурский государственный университет



А.Н. Рыбалев

Имитационное моделирование АСУ ТП

Монография

Благовещенск
Издательство АмГУ
2019

УДК 62-5 ББК
32.965я73
Р 93

Рыбалев А.Н.
Имитационное моделирование АСУ ТП / А.Н. Рыбалев. – Благовещенск: Амурский гос. ун-т, 2019. - 408 с.

ISBN 978-5-93493-335-8

Содержание

ВВЕДЕНИЕ

Разработка любой автоматизированной системы управления технологическим процессом (АСУТП) включает в себя решение множества задач. Значительная часть этих задач имеет чисто технический характер: выбор аппаратуры, составление принципиальных и других схем, компоновка шкафов управления и т.д. Ничуть не умаляя важности этих задач, отметим, что, тем не менее, все они являются вторичными по отношению к тем задачам, которые должны быть решены на самом первом этапе проектирования, когда определяется концепция будущей системы.

Что следует понимать под термином «концепция АСУТП»? Ничего сверхъестественного. Это то, что будет делать система управления, как она будет взаимодействовать с объектом управления и с человеком. По нашему мнению, концепция АСУТП проще всего может быть представлена в парадигме «модель-представление-контроллер» (Model-View-Controller, MVC), широко применяемой при построении компьютерных информационных систем. Применительно к АСУТП «модель» – это объект управления, а точнее его модель, «представление» – средства человеко-машинного интерфейса (Human-Machine Interface, HMI), а «контроллер» – это алгоритмы управления, заложенные в программируемые логические контроллеры (ПЛК), промышленные компьютеры и другую аппаратуру управления.

Если создана такого рода концепция (опять-таки модель) будущей системы, то вопросы технической реализации действительно становятся «техническими». Однако эта модель должна быть адекватной в широком смысле этого слова.

Модель объекта управления должна отражать все существенные для системы управления аспекты его поведения. Говоря по-простому, она должна связывать все интересующие нас «выходы» со всеми возможными «входами». Определение этих связей обеспечивается путем анализа процессов, протекающих в объекте и экспериментальными исследованиями. Сразу отметим, что хотя в книге и затронуты некоторые вопросы построения математических моделей объектов управления, эта проблематика не является для нее основной. В любом случае, без какой-либо модели объекта построить систему управления невозможно, и в этом смысле лучше иметь хоть какую-то модель, чем не иметь ее вовсе. С другой стороны, важен вид модели. Очевидно, что для нашей цели единственным приемлемым вариантом является имитационная компьютерная модель, позволяющая вести исследования в реальном масштабе времени и осуществлять обмен информацией с другими компонентами системы – «контроллером» и «системой человеко-машинного интерфейса». В книге все модели объектов управления реализуются в системе имитационного моделирования Simulink Matlab, которая, по мнению автора, является на сегодняшний день наиболее подходящим для этого инструментом.

Алгоритмы управления («контроллер») в принципе также могут быть реализованы блоками Simulink-модели, однако такой подход дает не совсем то, что нам нужно. Алгоритмы должны быть максимально приближены к своей реали-

зации на ПЛК и управляющих машинах. Simulink-модели, очевидно, таким качеством не обладают. Оптимальным видится представление алгоритмов в виде готовых программ, разработанных в универсальных системах класса SoftLogic на языках стандарта МЭК 61131-3 и запущенных на виртуальном контроллере, эмулирующим работу ПЛК на персональном компьютере. Такую возможность, в частности, предоставляет свободно распространяемая система CoDeSys, в составе которой имеется виртуальный контроллер PLC WinNT. Эта система программирования и рассматривается в книге.

Система «представления» или *визуализации технологического процесса* также может быть реализована в Matlab, но как и в случае с «контроллером», лучше разрабатывать ее в специально предназначенных для этого программных средах. Это может быть любая интегрированная среда разработки систем класса SCADA (Supervisory Control And Data Acquisition – диспетчерское управление и сбор данных), программное обеспечение для операторских панелей и т.д. В книге предлагаются решения на базе российской SCADA-системы Trace Mode, инструментальная среда разработки которой распространяется бесплатно.

Теперь о том, для чего, или для кого, это может быть нужно.

На практике разработка имитационной модели АСУ ТП позволит на «нулевом» этапе проектирования и при непосредственном участии заказчика:

– сформировать требования к системе для включения в Техническое задание и другие проектные документы;

– разработать *и отладить* алгоритмы управления и прототипы программного обеспечения АСУ ТП;

– разработать прототипы систем человеко-машинного интерфейса и, возможно, компьютерные тренажеры для обучения оперативного персонала.

В целом разработка имитационной модели позволит заказчику «опробовать» систему еще до ее создания и в дальнейшем эффективно управлять процессом ее проектирования и внедрения.

Важно отметить, что на каждом этапе формирования модели заказчик принимает самое непосредственное участие. В результате:

– заказчик знает, какие функции должна выполнять система и как должно быть организовано ее взаимодействие с оперативным персоналом, что дает возможность грамотно построить процесс взаимодействия с разработчиком;

– разработчик имеет планы щитов и пультов управления (реальных или виртуальных – экранов визуализации и управления), а также прототипы программного обеспечения системы управления (программ ПЛК, операторских панелей, рабочих станций), что существенно упрощает дальнейшее проектирование и внедрение системы.

При *идеальной реализации* концепции заказчик получает именно такую систему управления, на которую он рассчитывал. Исключаются конфликты, связанные с «различным пониманием» требований Технического задания со стороны заказчика и исполнителя. Существенно снижаются затраты на проектирование и внедрение АСУ ТП за счет того, что «консенсус» был достигнут еще на этапе компьютерного моделирования, а программное обеспечение тре-

бует, по большей части, лишь «механического» преобразования. Оперативный персонал на момент внедрения системы уже практически освоил навыки работы с ней, пользуясь компьютерными тренажерами.

В образовательных учреждениях, занимающихся подготовкой специалистов в области промышленной автоматизации, использование имитационных моделей решает целый ряд проблем, связанных с отсутствием *на месте* реальных объектов и аппаратуры управления. Разрабатывая проект АСУ ТП, студент не только выбирает необходимые технические средства и составляет разнообразные схемы, но и занимается моделированием процесса, а также, практически, – созданием программного обеспечения для различных уровней управления. Причем это программное обеспечение действительно *отлаживается* и *апробируется,* пусть даже и только на модели. Хорошо известно, что неотлаженная программа – это вовсе не программа, а всего лишь «декларация о намерениях». Поэтому имитационное моделирование, безусловно, повысит качество подготовки специалистов. А, кроме того, для самого студента получить «живую» программную систему – это еще и очень интересно…

В первой части книги кратко описываются используемые технологии. Основной упор сделан на изложение их сущности и практическим рекомендациям по настройке межпрограммного взаимодействия.

Вторая часть посвящена теоретическим (но не только!) вопросам автоматического регулирования и программно-логического управления. Как это принято, можно было бы разрешить теоретически-подкованным читателям пропустить эту главу, однако автор, нисколько не сомневаясь в этой подкованности, все же настойчиво рекомендует с ней ознакомиться, поскольку материал является в достаточной степени оригинальным. При его изложении предпринята попытка по возможности упростить математические выкладки и больше сосредоточиться на «физике» рассматриваемых процессов. Конечно, совсем без математики не обойдешься, но можно сделать так, чтобы она не сразу «била по голове», а усваивалась постепенно. Поэтому относительно много времени, а точнее, страниц, посвящено вопросам математического описания систем управления и их элементов, причем это описание рассматривается с «физической» точки зрения. С другой стороны, некоторые «фундаментальные» вопросы теории только «затронуты» самым «поверхностным образом». Нет никакого смысла повторять то, что написано в бесчисленном количестве учебников, если только Вы не предлагаете чего-то нового или какой-то новый взгляд на уже существующие вещи.

В третьей части приводится ряд примеров реализации концепции.

Подробно описан процесс разработки имитационных моделей систем регулирования технологического параметра с типовым регулятором. При этом рассматриваются все три уровня таких систем: уровень объекта, точнее, его имитационной модели, уровень контроллера и уровень SCADA-системы. Отдельно разбираются различные варианты технической реализации управляющего воздействия: объект управляется непрерывным сигналом, управляющее воздействие формирует исполнительный механизм постоянной скорости без измерения положения и с его измерением. Затрагиваются вопросы взаимодей-

ствия программного обеспечения контроллера и операторской панели со SCADA-системой, а также вопросы ведения архивов дискретных событий и непрерывных величин.

Продемонстрирован общий подход к построению нетипового линейного программного регулятора для ПЛК с произвольной передаточной функцией. Предложено несколько алгоритмов программной реализации транспортного запаздывания в составе закона регулирования, которые были опробованы в ходе моделирования системы с предиктором Смита.

На примере системы управления процессом дозирования построена достаточно сложная имитационная модель системы программно-логического управления. Ее частью стала модель электрической схемы цепей коммутации с различными блокировками, оформленная как подпрограмма ПЛК на языке LAD. Основная программа для ПЛК построена как конечный автомат. Помимо нее в ПЛК действует также программа архивирования. Оператор SCADA-системы может наблюдать за ходом процесса и просматривать архивы.

На примере системы управления несколькими насосами рассмотрена проблема обеспечения равномерного износа однотипных агрегатов. Предложен универсальный алгоритм и его программная реализация для ПЛК, которые и были опробованы в имитационной системе.

В заключение представлена имитационная модель лабораторной лифтовой установки, реально существующей (и действующей) на кафедре автоматизации производственных процессов и электротехники Амурского государственного университета.

Глава 1. ТЕХНОЛОГИИ

1.1. Межпрограммный обмен

«Изначальная» идея Windows, как и других многозадачных операционных систем, состоит в изоляции процессов (грубо говоря, работающих программ) друг от друга. Процесс ограничен в своем адресном пространстве и не может «заглянуть» в адресное пространство другого процесса. В то же время, взаимодействие процессов часто требуется, и тогда операционная система выступает в качестве посредника. Существует достаточно много механизмов межпрограммного обмена. Простейшим, в некотором смысле, является обмен посредством обычных файлов на диске. Одна программа записывает данные в файл, другая – читает их из него. Очевидно, что скорость такого обмена невелика, и для наших целей такой способ не годится. В нашей работе задействуются технологии *OPC* и *сокетов*.

1.1.1. OPC и COM

Аббревиатура OPC (сейчас – это *Open Platform Communications, взаимодействие открытых платформ*) ранее означала *OLE for Process Control*, *OLE для управления процессами*. OLE расшифровывается как *Object Linking and Embedding – связывание и внедрение объектов* одного приложения в другое. Сейчас эта технология называется ActiveX. С помощью OLE/ActiveX можно, например, вставить в документ MS Word диаграмму MS Excel и редактировать ее средствами Excel, не выходя из Word. Технология OPC в основном занимается передачей данных (значений технологических переменных) из одной программы в другую и поэтому на первый взгляд к OLE/ActiveX не имеет никакого отношения. На самом деле и OLE/ActiveX и OPC являются по сути «надстройками» к базовой технологии Microsoft COM/DCOM.

COM (*Component Object Model*, *объектная модель компонентов*) и DCOM (*Distributed COM, распределенная COM*) – технологии, предназначенные для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно [1]. Разработчики технологии стремились решить две задачи:

1) сделать возможным свободное распространение готовых двоичных компонентов, которые можно было бы подключать к любым разрабатываемым программам, без раскрытия исходных кодов компонентов и без встраивания их откомпилированных кодов в библиотеки средств разработки;

2) сделать так, чтобы разработчикам программного обеспечения эти компоненты представлялись бы в виде объектов классов в стиле объектно-ориентированного программирования.

Разработчик прикладной программы работает с объектами COM-компонентов *почти* также, как он работает объектами известных ему классов. Для того чтобы это стало возможным, реализованы следующие положения:

1) все COM-компоненты имеют уникальные «имена» в операционной системе. Операционная система знает, в каком исполняемом модуле находится объект и поэтому может запустить модуль, его реализующий.

Для именования объектов используются специальные идентификаторы классов, называемые CLSID и идентификаторы их интерфейсов, называемые IID (объект класса может иметь несколько интерфейсов). Это уникальные 128-битовые числа вида

{22F55881-280B-11d0-A8A9-00A0C90C2004}.

Идентификаторы генерируются специальной программой с использованием случайных чисел и, хотя уникальность каждого отдельного идентификатора не гарантируется, общее количество уникальных ключей настолько велико (2^{128} или $3{,}4028\times10^{38}$), что вероятность того, что в мире будут независимо сгенерированы два совпадающих ключа, крайне мала [1, 2].

В системном реестре каждому CLSID поставлены в соответствие имя исполняемого файла (dll – для *inproc-сервера* или exe – для *local-сервера*), а также «человеческое имя» объекта. Причем поддерживаются списки вида:

CLSID – имя

Имя – CLSID.

2) в операционной системе имеется специальная функция, возвращающая адрес COM-объекта по его имени. Эта функция, будучи вызванной прикладной программой, находит в системном реестре информацию об исполняемом модуле, загружает его в память, «заставляет» его создать нужный объект и возвращает адрес этого объекта приложению. Исполняемые модули, реализующие COM-объекты, имеют особый «стандартный вход», посредством которого от операционной системы они получают запросы на создание объектов. Функция API, про которую идет речь, называется CoCreateInstance – создать экземпляр объекта.

3) Приложение, использующее COM-объекты, должно заранее знать, каким образом оно будет взаимодействовать с ним. Программы, построенные по технологии COM, предоставляют пользователю так называемые «объектные модели», описывающие возможности COM-классов. Кроме того, COM-компоненты «снабжаются» т.н. библиотекой типов, двоичными файлами с описанием интерфейсов COM-объекта и их методов на специальном языке IDL. Среды разработки «умеют» читать такие файлы и отображать информацию для разработчиков.

Взаимодействие приложения с COM-объектами происходит по схеме «клиент-сервер». Приложение выступает в качестве клиента, а объекты «живут» внутри сервера. Как уже было упомянуто выше, существует два вида серверов. Так называемые inproc-серверы (in process – в процессе) размещаются в dll-файлах (dynamic link library – динамически подключаемые библиотеки). При обращении к серверу с просьбой создать объект исполняемый код библиотеки загружается непосредственно в адресное пространство приложения и становится, таким образом, напрямую ему доступным. Local-серверы «обитают» в отдельных исполняемых exe-файлах. При запуске сервера исполняемый код запускается в отдельном адресном пространстве, а в адресное пространство приложения загружается специальный агент-посредник proxy/stub DLL, который играет роль представителя объекта у обратившегося к нему клиента. Proxy/stub

DLL переадресует вызовы серверных функций операционной системе, которая (возможно, по сети) доставляет их по назначению и заставляет реальный объект выполнить заданную функцию. Результат затем возвращается приложению-клиенту. Очевидно, что время отклика local-сервера больше чем у inproc-сервера. Может случиться также, что клиент и сервер, даже сосуществуя на одном компьютере, имеют разную архитектуру. Например, сервер – 32-разрядный, а клиент – 64-разрядный. Если тип сервера – local, он запускается в отдельном процессе и дальше за связь между процессами отвечают специальные механизмы операционной системы. В случае inproc-сервера возникает необходимость в загрузке DLL сервера в процесс клиента. Но если процесс 64-разрядный, то 32-разрядная DLL не сможет в нём работать. Поэтому Windows создает суррогатный процесс, в который загружает 32-разрядную DLL и, таким образом, по сути, превращает inproc-сервер в local-сервер.

OPC использует технологию COM/DCOM для решения задачи обмена данными в системах промышленной автоматизации. OPC-клиент (любое приложение, например SCADA-система), вызывая определенные функции объекта OPC-сервера, подписывается на получение определенных данных с определенной частотой. OPC-сервер, получив каким-то образом эти данные (чаще всего, опросив оборудование), вызывает известные функции клиента и вручает ему данные. Таким образом, используются как прямые COM-вызовы (от клиента к серверу), так и обратные (callback, от сервера к клиенту).

OPC-серверы конкретных аппаратных устройств поставляются многими производителями аппаратуры. Связь сервера с аппаратурой может осуществляться через какой-либо физический интерфейс компьютера: последовательный порт, USB, Ethernet, плату расширения с выходом на промышленную сеть и т.д. Для приложения-клиента параметры физического подключения совершенно не интересны, поскольку при настройке соединения с сервером они не задействуются (они используются только при первоначальном конфигурировании самого сервера). Настройка клиента сводится к выбору нужного сервера из списка зарегистрированных в системе, соединения с ним и выбору переменных для чтения и записи из предоставляемого сервером списка переменных. Таким образом, OPC-сервер создает абстракцию аппаратуры, позволяя любому OPC-клиенту записывать и считывать данные с устройства.

Существуют OPC-серверы и не привязанные к конкретной аппаратуре. Например, серверы, обеспечивающие обмен данными с любыми устройствами промышленной сети Modbus. При конфигурировании таких серверов указывают параметры физического подключения компьютера к сети, адреса устройств в сети, номера регистров и ячеек этих устройств, а также функции Modbus, которые будут использоваться для чтения и записи.

Стандарт OPC разрабатывает независимая организация OPC Foundation, членами которой являются практически все известные компании-производители систем промышленной автоматизации.

Стандарт состоит из трех основных спецификаций:

доступ к данным реального времени (Data Access);

обработка тревог и событий (Alarms & Events);

доступ к историческим данным (Historical Data Access).

В общем случае может быть, соответственно, и три вида OPC-серверов, хотя чаще всего все эти функции совмещаются в одном. OPC-серверы физических устройств обычно являются только серверами данных (Data Access Servers), серверы тревог и исторические лишь дополняют функционал серверов данных.

Сервер тревог формирует определенные логические переменные, называемые состояниями (conditions), имея в качестве исходной информации некую переменную (тег), полученную от сервера данных. Состояния изменяют свое значение, если переменная, например, вышла за допустимые границы. Об изменении состояния сервер тревог оповещает клиентов, посылая им событие (тревогу), а клиент возвращает серверу подтверждение, что он тревогу воспринял.

Серверы исторических данных получают от серверов данных параметры в реальном времени и архивируют их, а затем предоставляют эти данные другим приложениям (например, для построения графиков трендов).

Основной спецификацией OPC является доступ к данным реального времени (Data Access). Базовое понятие этой спецификации – элемент данных (Item). Каждый элемент (параметр технологического процесса) имеет значение, время последнего обновления (timestamp) и признак качества, определяющий степень достоверности значения.

Значение может иметь любой скалярный тип (булев, целый, с плавающей точкой и т.п.) или быть строкой.

Время представляется с наносекундной точностью, что в большинстве случаев является избыточным.

Качество – это код, оценивающий достоверность значения параметра – UNCERTAIN, GOOD и BAD (неопределенная, хорошая и плохая). В случае плохой оценки предоставляется расшифровка, например, QUAL_SENSOR_ FAILURE – неисправность датчика.

Элементы данных «помещаются» в *группы* (OPC Group). Группы создаются OPC-сервером по запросу клиента, после чего они наполняются нужными элементами из общего списка элементов сервера. Для группы задается единая для всех его членов частота обновления данных. Клиент может создать для себя на сервере несколько групп, различающихся требуемой частотой обновления. Отсоединение клиента приводит к уничтожению созданных для него групп.

Элементы в группе на самом деле – это клиентские ссылки на некие реальные переменные (теги), находящиеся на сервере или в физическом устройстве.

Для запроса имен тегов служит специальный интерфейс IOPCBrowseServerAddressSpace, с помощью которого сервер описывает клиенту свое «пространство имен». Это пространство организованно в общем случае иерархически, например:

Устройство_1. Модуль_2. Аналоговый Вход_3.

Обмен данными между клиентом и сервером может осуществляться в двух режимах – синхронном и асинхронном.

При *асинхронном варианте* обмена сервер сам оповещает клиента об изменившихся значениях данных, на которые подписался клиент (по возможности с частотой, заданной клиентом при создании группы).

При *синхронном* клиент осуществляет инициативное чтение или запись данных. Запись может быть только синхронной.

1.1.2. Сокеты

Сокет (socket – разъем) – это конечная точка соединения программ в компьютерной сети. Если используются сетевые протоколы TCP/IP, то сокет однозначно идентифицируется IP-адресом компьютера и номером *порта*, закрепленного за приложением. Взаимодействие программ происходит по схеме «клиент – сервер». Обмен данными инициируется исключительно клиентом, поэтому клиент должен «знать» сервер. Сервер о клиентах (до и после обмена!) не знает. Приложения-серверы, как правило, имеют фиксированные номера портов, а приложениям-клиентам номер порта выделяется операционной системой по мере необходимости. У одного сервера может быть несколько клиентов. Механизм сокетов применяется не только для обмена данными между приложениями, запущенными на разных компьютерах и связанных сетью. Программы вполне могут работать и на одной машине. В этом случае используется стандартное доменное имя localhost, означающее буквально локальный хост (этот компьютер). Это имя преобразуется в IP-адрес 127.0.0.1, так называемый сетевой интерфейс «внутренней петли». Пакет с адресом получателя 127.0.0.1 не покидает пределов компьютера, а сразу же отправляется приложению-получателю по указанному номеру порта.

1.2. Элементы имитационной системы и их взаимодействие

1.2.1. Состав и схема взаимодействия программ

Состав и схема взаимодействия программ-элементов имитационной системы показана на рис. 1.1.

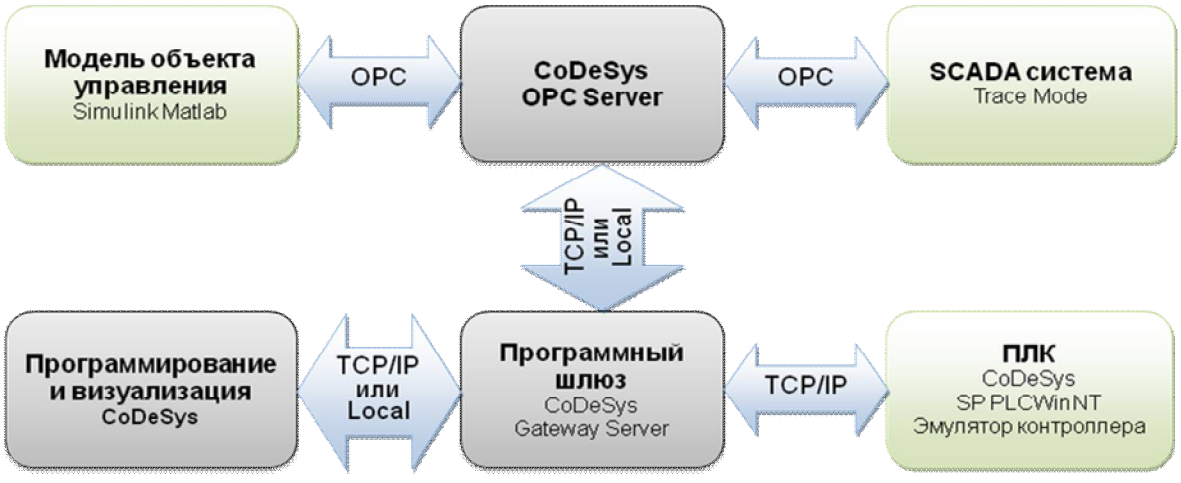


Рис. 1.1. Схема взаимодействия программ.

Объект управления представлен своей имитационной Simulink-моделью. Говоря упрощенно, модель «пересчитывает» входные (управляющие) сигналы в

выходные, несущие информацию о состоянии объекта. Для ввода-вывода используются специальные блоки из пакета OPC Toolbox, наличие которых в Simulink-диаграмме автоматически обеспечивает «работу» модели в реальном времени.

Частота пересчета модели, конечно, не совпадает с частотой обмена по протоколу OPC, – она существенно выше, что позволяет объекту изменять свое состояние и в период между обменами. Однако блоки препятствуют тому, чтобы модельное время изменялось быстрее, чем реальное, как это обычно имеет место в Simulink-моделях. Конечное время расчета в наших системах мы всегда будем устанавливать равным бесконечности (в Matlab за бесконечность «отвечает» константа `inf` – infinity, бесконечность). Таким образом, остановить расчет можно будет только вручную.

Управляющая программа выполняется на программном эмуляторе ПЛК SP PLC WinNT, входящем в состав пакета программ CoDeSys. Необходимость в «отдельном» программном эмуляторе состоит в следующем. Конечно, сама среда программирования CoDeSys дает возможность запуска открытых в ней проектов без подключения к ПЛК в режиме симуляции (Simulation Mode). Однако в данном режиме «общаться» с запущенной программой, т.е. задавать входные и наблюдать выходные переменные, можно только из среды программирования. В этом случае следует забыть об использовании каких бы то ни было внешних программ и всю имитацию АСУ ТП внедрить непосредственно в проект CoDeSys. В принципе такой подход вполне имеет право на существование, поскольку CoDeSys предоставляет полный набор возможностей для его реализации: можно «оформить» программную модель объекта в виде одной или нескольких программных единиц (программ или функциональных блоков), а для имитации человеко-машинного интерфейса задействовать достаточно развитые средства визуализации самой системы CoDeSys. Но, на наш взгляд, такой «бюджетный» вариант имитационной системы все же проигрывает варианту с использованием разных программ по следующим причинам:

1) на языках программирования ПЛК модели более-менее сложных объектов управления составлять неудобно, и, кроме того, теряется возможность управления параметрами пересчета модели (точность, временной шаг и т.д.). Simulink же предоставляет огромное количество «готовых к употреблению» «строительных блоков» для построения модели и достаточно много разных методов численного интегрирования для ее расчета. При этом все настройки доступны для изменения;

2) функционал визуализации CoDeSys при всем его богатстве все же уступает функционалу «полноценных» SCADA-систем. К тому же при переносе полученных решений по визуализации на «целевую платформу» (если это потребуется) неизбежно возникнут определенные трудности. Поэтому лучше с самого начала использовать «настоящую» SCADA-систему, тем более, что все они поддерживают обмен по OPC. Перенаправление информационных потоков (входов/выходов), в отличие, например, от переноса графического интерфейса, – задача вполне себе тривиальная.

Программный эмулятор SP PLC WinNT представляет собой отдельную программу, работающую независимо от среды программирования CoDeSys. Помимо среды программирования (и даже нескольких таких сред, запущенных, например, на разных компьютерах) он может взаимодействовать с любыми другими программами посредством специального программного шлюза CoDeSys Gateway Server, о котором речь пойдет далее. Применяя средства библиотек, специально разработанных для SP PLC WinNT, имеется возможность использовать его и для управления объектами реального мира. Правда, при этом мы ограничены набором физических интерфейсов связи самого компьютера. В частности, существуют библиотеки для организации обмена данными по протоколу Modbus через последовательный порт или USB (с использованием специальных преобразователей RS232/RS485, USB/RS485) и Ethernet (Modbus TCP/IP). В бесплатной версии эмулятор непрерывно может работать не более двух часов, после чего его необходимо перезапустить заново.

В качестве SCADA-системы в книге используется система Trace Mode российской фирмы Adastra. Это одна из самых популярных российских SCADA-систем. Базовая версия среды разработки, которой вполне достаточно для разработки имитационной модели, может быть бесплатно загружена с сайта производителя. Однако, поскольку обмен данными в имитационной системе производится по протоколу OPC, вместо Trace Mode можно использовать любую другую современную SCADA-систему.

Помимо описанных выше *основных компонентов* в состав имитационной модели входят сама среда программирования контроллеров CoDeSys, а также промежуточные звенья: программный шлюз CoDeSys Gateway Server и OPC-сервер CoDeSys OPC Server.

Описание среды программирования CoDeSys не входит в наши задачи, так как она достаточно хорошо представлена в многочисленных публикациях, в частности в [3]. Краткая информация о сервере-шлюзе и OPC-сервере и их настройках приведена ниже.

1.2.2. CoDeSys Gateway Server

Для подключения всех программ к виртуальному контроллеру CoDeSys SP PLCWinNT используется программный шлюз CoDeSys Gateway Server [4]. Gateway Server необходим и для связи со всеми реальными контроллерами, программируемыми с помощью CoDeSys. Как это ясно уже из названия, CoDeSys Gateway Server играет роль сервера, а сама среда программирования – роль клиента. В общем случае одновременно клиентами могут быть несколько программ, которым необходим доступ к одному или нескольким ПЛК. Эти программы могут быть запущены на одном компьютере или на нескольких компьютерах сети, и среди них могут быть и несколько сред программирования CoDeSys. Сам сервер поддерживает различные варианты подключения к контроллерам, в том числе через последовательный порт и Ethernet.

CoDeSys Gateway-сервер, как правило, запускается автоматически при загрузке компьютера и переходит в «ждущий режим». При необходимости он может быть запущен и вручную. О том, что сервер загружен, свидетельствует

появление иконки на панели задач. Щелчком правой кнопки мышки по этой иконке можно вызвать контекстное меню, позволяющее произвести настройку сервера, в частности, ввести парольную защиту, включить трассировку и регистрацию событий.

1.2.3. CoDeSys OPC Server

Для передачи данных CoDeSys OPC-сервер использует так называемые *символьные файлы* [5]. Символьные файлы содержат описания элементов данных (переменных прикладной программы) и создаются системой программирования CoDeSys. Затем они передаются программному шлюзу и контроллеру одновременно с загрузкой программы. В качестве контроллера может использоваться и SP PLC WinNT (рис. 1.2).

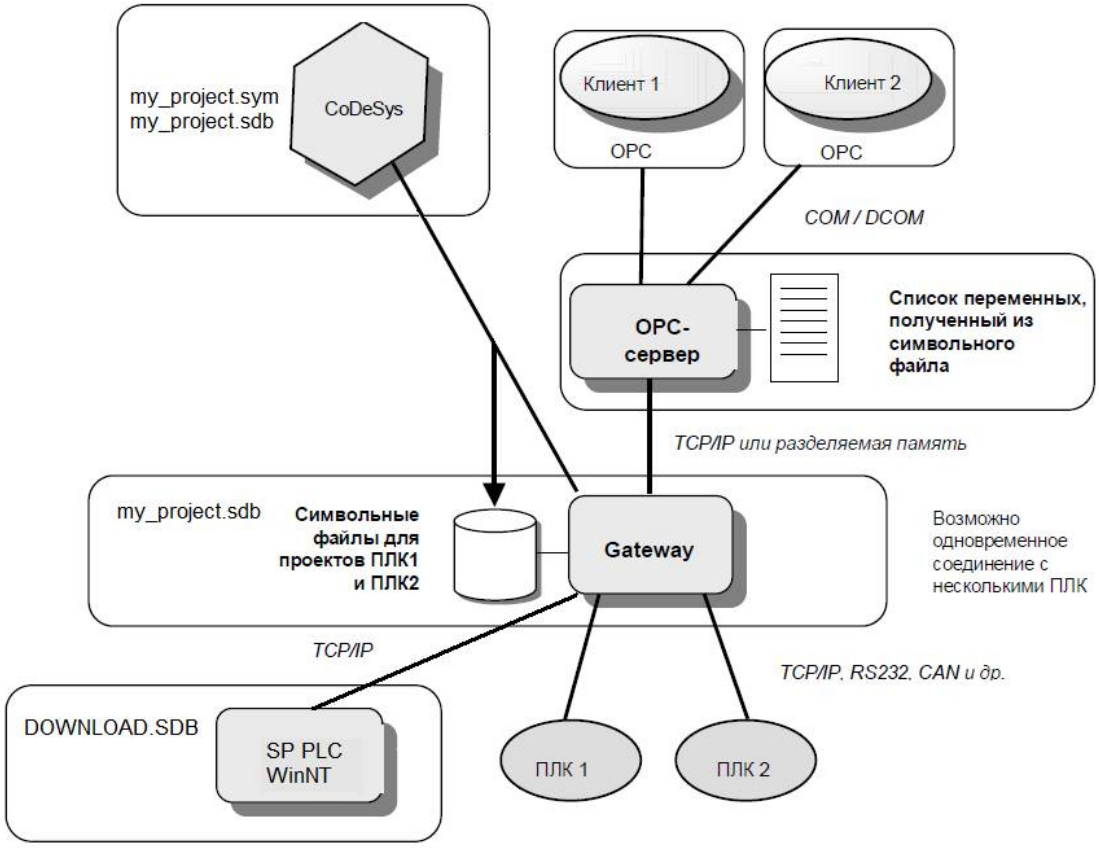


Рис. 1.2. Детали взаимодействия с ПЛК через OPC-сервер.

В папке проекта среда программирования CoDeSys (все детали далее относятся к версии 2.3) создает два файла с одинаковыми именами, совпадающими с именем проекта, и расширениями .sym или .sdb. Файл с расширением .sdb – это бинарная версия текстового файла .sym. Он быстрее считывается программами, поэтому шлюзу и контроллеру передается именно он, тогда как текстовый файл, вероятно, предназначен для человека.

В случае проблем с обменом данными в первую очередь следует проверить наличие и содержание символьных файлов:

1) открыть папку проекта и убедиться, что файлы существуют, а в sym-файле перечислены все переменные, задействованные в обмене. Если файлов нет, то, вероятнее всего, CoDeSys не настроен на их генерацию. Если фай-

лы содержат неполный список переменных, следует в CoDeSys «очистить» и перекомпилировать проект» (меню Project/Clean all и Project/Rebuild all соответственно), после чего вновь подключиться к контроллеру, что приведет к генерации новых файлов и загрузке их в контроллер. Подключение к контроллеру без перекомпиляции проекта не подразумевает этих шагов, – вносятся лишь изменения, сделанные в программе с момента последнего подключения;

2) в папке Gateway Files (ее расположение зависит от операционной системы, возможные варианты: Windows, Program Data) с помощью обыкновенного текстового редактора открыть и просмотреть sdb-файл. Несмотря на бинарный формат файла, имена переменных вполне читаемы. Если файла нет, то возможной причиной является запрет доступа к папке для записи. В этом случае разрешите полный доступ к папке для всех пользователей. Если файл «неполный», выполните действия, описанные выше;

3) в папке Program Files (x86)\3S Software\CoDeSys SP PLCWinNT открыть и просмотреть файл DOWNLOAD.SDB. Порядок действий при обнаружении «неисправностей» описан выше.

OPC-сервер запрашивает у Gateway содержимое символьных файлов и на их основе составляет внутренний список элементов. Текущие значения данных для элементов этого списка становятся доступными клиентам.

Как указано в [5], OPC свободно оперирует со списками до 15 000 элементов и символьными файлами до 1,5 MB. Получить одновременно доступ к переменным сразу нескольких проектов (приложений) можно, если символьные файлы были переданы одному и тому же Gateway.

OPC-сервер может группировать данные как по заказу клиента (частные, private, группы), так и самостоятельно (общие, public, группы). Отдельные группы опрашиваются OPC-сервером поочередно. Данные элементов внутри одной группы читаются одновременно, насколько это позволяет объем коммуникационных буферов ПЛК.

OPC-сервер автоматически запускается операционной системой, как только клиент устанавливает соединение. Если OPC-сервер был предварительно запущен вручную, клиент будет автоматически соединен с работающим OPC-сервером. Как только все клиенты разрывают свои соединения с сервером, он автоматически останавливается. Во время работы OPC-сервера на панели задач отображается соответствующая иконка.

1.2.4. Настройка соединения

Подробно последовательность действий, осуществляемых при подключении контроллеров системы CoDeSys через OPC-сервер к компьютеру, описана в [6]. Ниже перечислены шаги, которые необходимо сделать для организации обмена с виртуальным контроллером SP PLCWinNT V2.4.

1. Создать программу (проект) в среде CoDeSys с целевой платформой 3S CoDeSys SP PLCWinNT V2.4. При настройке целевой платформы следует установить параметр *Download symbol file* вкладки *General* (рис. 1.3).

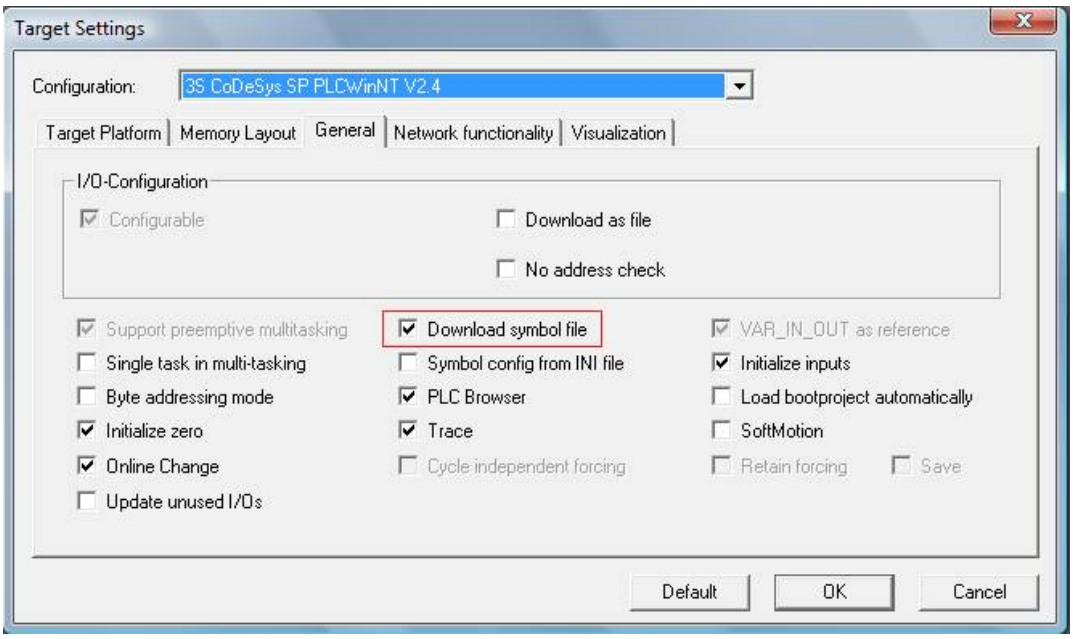


Рис. 1.3. Настройка целевой платформы.

2. Объявить переменные для обмена по OPC. В общем случае такие переменные могут быть объявлены в любой из программ, однако для упрощения доступа лучше все их сделать глобальными.

3. Написать программу PLC_PRG. В программе должен быть, по меньшей мере, один оператор, поскольку без этого она не компилируется.

4. Сохранить проект под осмысленным именем в отдельную папку.

5. Запустить PLCWinNT (*Пуск* → *Все программы* → *3S Software* → *CoDeSys SP PLCWinNT* → *CoDeSys SP PLCWinNT V2.4*) (рис. 1.4), установить связь и загрузить программу в «контроллер».

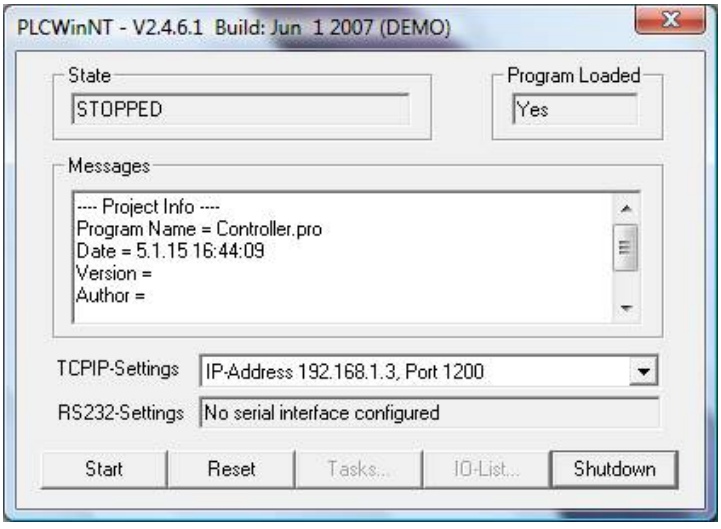


Рис. 1.4. Окно программы PLCWinNT

В случае, если связь с контроллером установить не удалось, следует настроить коммуникационные параметры в меню Online (рис. 1.5).

6. Отключить CoDeSys от PLCWinNT и перейти в Опции (*Options*) в меню *Project*. Выбрать *Symbol Configuration* и установить галочку *Dump symbol entries* (рис. 1.6).

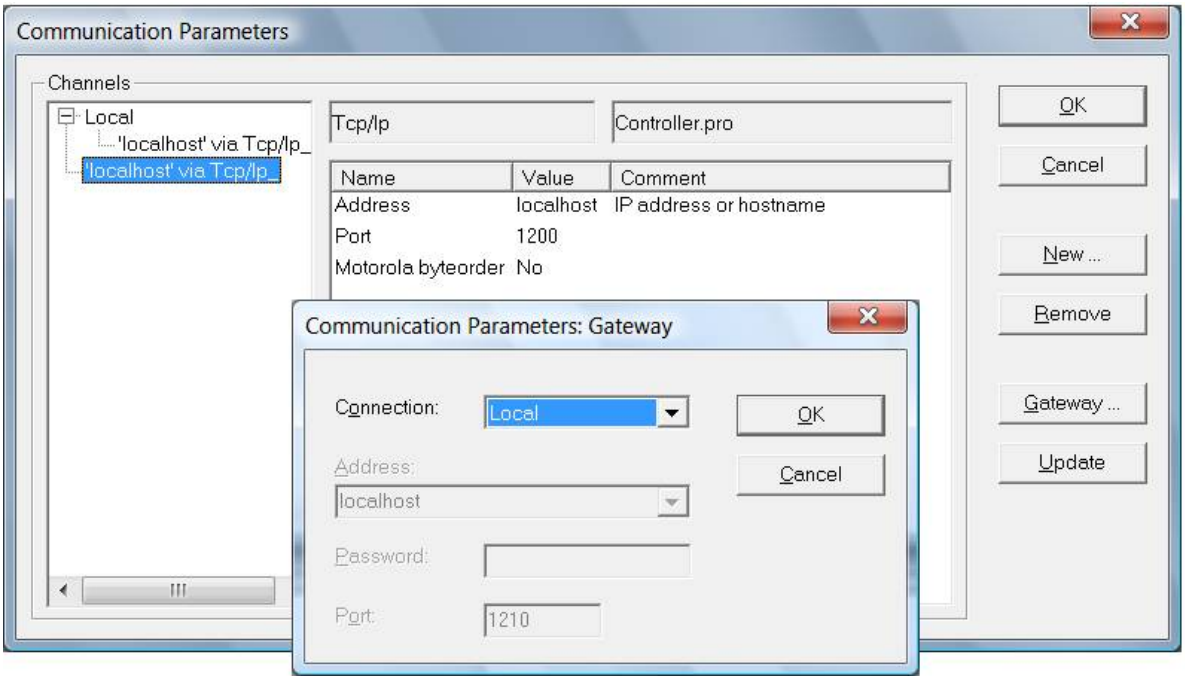


Рис. 1.5. Настройка коммутационных параметров.

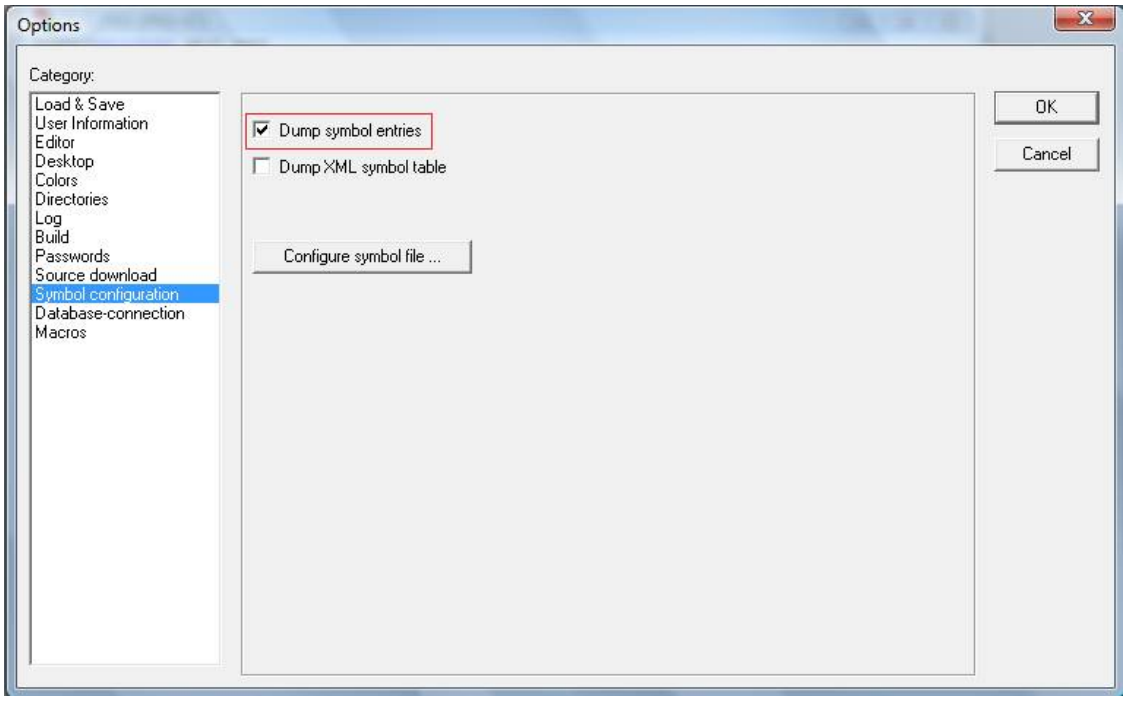


Рис. 1.6. Установка опций.

7. Сконфигурировать «символьный файл», выбрав переменные для обмена по OPC, рис. 1.7.

8. Настроить параметры OPC-сервера. Для этого необходимо запустить конфигуратор OPC-сервера (*Пуск → Все программы → 3S Software → Communication → CoDeSys OPC Configurator*). В окне конфигуратора требуется добавить PLC (*Append PLC*) и настроить соединение (*Connection*). По существу достаточно выбрать из списка соединение, настроенное в проекте CoDeSys (рис. 1.8).

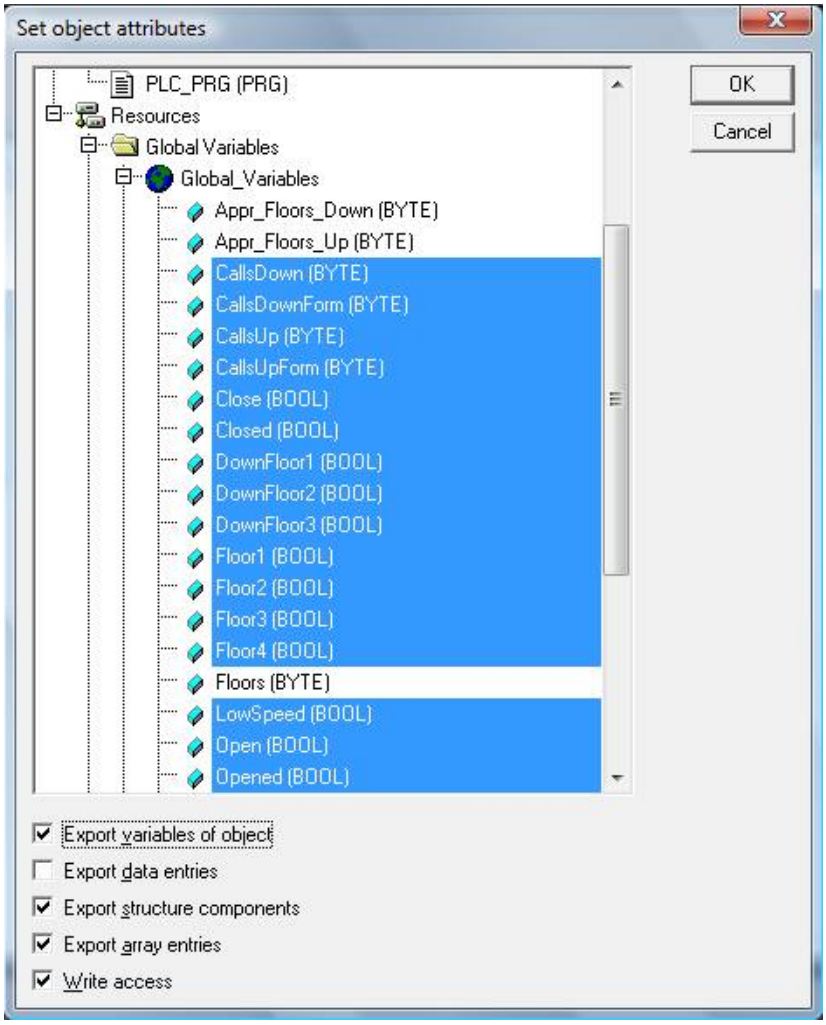


Рис. 1.7. Выбор переменных для обмена по OPC.

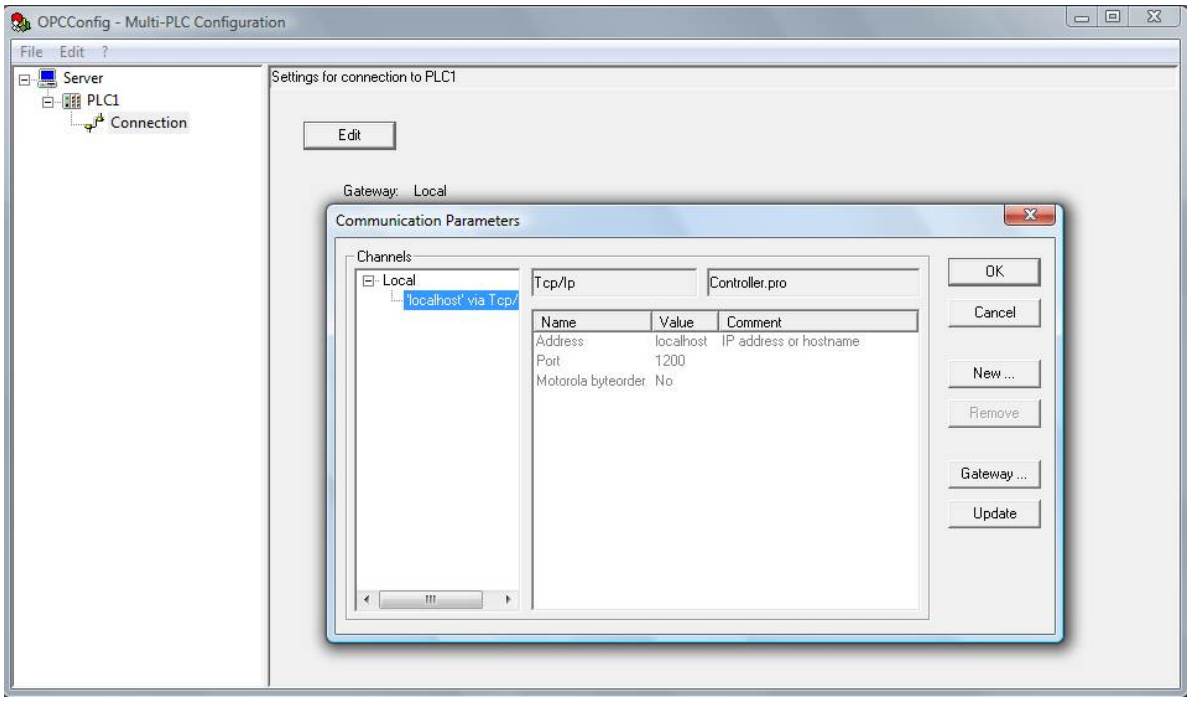


Рис. 1.8. Настройка OPC сервера.

9. «Перестроить» программу (проект) CoDeSys, включив в нее все изменения, сделанные после загрузки в контроллер. Для этого требуется вызвать команду *Clean all* из меню *Project*, затем команду *Rebuild all* из того же меню. CoDeSys перекомпилирует программу и перезагрузит проект при следующем подключении к ПЛК.

10. Подключиться к PLCWinNT, согласившись на перезагрузку проекта. С помощью любой программы OPC-клиента, проверить доступность OPC-сервера CoDeSys и переменных для обмена. Рекомендации по решению возможных проблем приведены в предыдущем пункте.

2. НЕМНОГО ТЕОРИИ

2.1. Управление

Изложение теории управления не является главной задачей данной книги. Однако совсем уж обойти теоретические вопросы не представляется возможным. Далеко не все заинтересованные в основном материале читатели изучали когда-то соответствующие курсы, а те, кто изучал, мог все и позабыть (а кто-то вообще «не знал и забыл»). Без знания теоретических основ такие читатели будут испытывать некоторые трудности при работе с материалом.

В этой главе предпринята попытка изложить только самое необходимое, при этом сделать это максимально доступным языком.

Кратко рассмотрена терминология систем управления, знание которой просто обязательно для всех кто имеет с ними дело.

Изложены основные понятия математического описания объектов и систем регулирования. В отличие от многих книг по теории управления основной упор сделан на физической сущности процессов, происходящих в системах. Привлекаемый математический аппарат подчинен «физике» процессов.

Начать разговор об *управлении*, безусловно, нужно с того, *чем* мы управляем, или, по-другому, с *объекта управления*. Как и во многих других процессах, в процессе управления участвуют объект и субъект. Субъект оказывает воздействие на объект с целью добиться от него желаемого (для субъекта) поведения.

Основные принципы управления одинаковы для всех сфер: технической, экономической, социальной и т.д. В технической сфере, которой мы и ограничимся, объектами управления являются разнообразные машины, механизмы, устройства и т.д. В общем, это оборудование, задействованное в технологическом процессе. Субъектами управления являются люди и разного рода автоматы, которым люди доверили выполнять свою работу по управлению оборудованием.

Автоматизация технологического процесса заключается в разработке и внедрении систем автоматизированного и автоматического управления. В автоматизированных системах часть работы по управлению ведется автоматами, а другая часть выполняется людьми. В автоматических системах все задачи управления решаются автоматами. (Как говорят, автоматы полностью исключили пресловутый «человеческий фактор»).

Для описания процесса управления и, тем более, для разработки автоматических систем управления требуется формализовать объект, построить некоторую абстракцию, выделив то, что действительно важно, и отбросив все остальное за ненадобностью. С точки зрения системы управления гидравлическим прессом (если у этой системы есть «точка зрения») совершенно не важно, в какой цвет он покрашен, но важно текущее положение его рабочего органа, давление в гидроцилиндре и т.д.

На рис. 2.1. показан некоторый обобщенный объект управления.

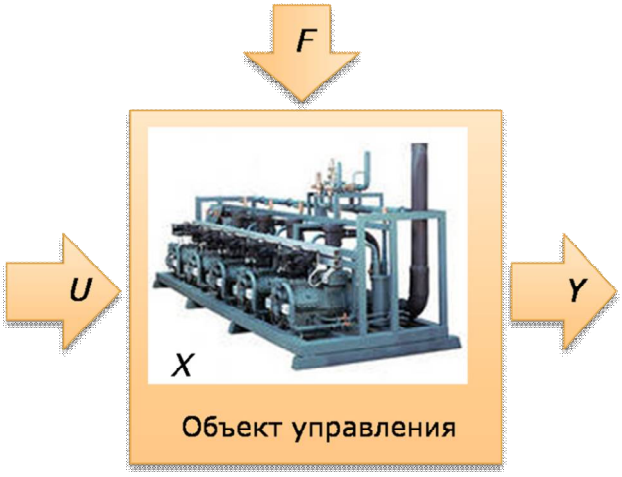


Рис. 2.1. Объект управления.

Состояние объекта управления *описывается* множеством переменных $X = \{x_1,x_2,\ldots,x_n\}$, которые и называются *переменными состояния*. Это могут быть различные физические величины: температура, давление, уровень, положение в пространстве, скорость его изменения и т.д. Важно то, что набор этих переменных описывает состояние объекта *однозначно* в том смысле, что зная их значения, можно найти значения всех других переменных. Например, в линейных электрических цепях все токи и напряжения можно найти, зная токи индуктивностей и напряжения на емкостях. Поэтому токи индуктивностей и напряжения на емкостях могут быть переменными состояния линейной электрической цепи. Переменные состояния влияют друг на друга, но не связаны между собой жестко и поэтому могут считаться независимыми.

Состояние объекта *контролируется* путем измерения некоторых переменных $Y = \{y_1,y_2,\ldots,y_l\}$, которые называются *выходными переменными*. Множества X и Y могут пересекаться или даже полностью совпадать, что является идеальным с точки зрения управления вариантом. Однако в большинстве случаев переменные состояния в полном составе измерению не доступны. Часто в системах управления используются косвенные измерения, например, об уровне или расходе судят по перепаду (разности давлений), а о влажности по показаниям двух термометров: сухого и влажного.

Состояние объекта *зависит* от оказываемых на него воздействий (*входных воздействий*): *возмущающих* $F = \{f_1,f_2,\ldots,f_k\}$ (*возмущений*) и *управляющих* $U = \{u_1,u_2,\ldots,u_m\}$ (*управлений*). Возмущения «приходят» из внешнего мира, а управления формируются системой управления.

Для организации процесса управления объектом обязательно требуется знать две вещи:

1) для чего нужно им управлять *(цель управления)*;

2) как можно им управлять *(модель объекта управления)*.

Цель управления всегда формулируется в терминах того технологического процесса, в котором объект управления задействован, например это может быть:

выработка продукта с заданными (наилучшими) характеристиками;

обеспечение оптимальной и безопасной работы оборудования:

перемещение из одной точки в другую за заданное (минимальное) время и т.д.

Цель управления позволяет определить желаемое состояние объекта в любой момент времени процесса, т.е. осуществить *планирование* (иногда применяют термин *программирование*) (рис. 2.2).

Рис. 2.2. Планирование.

В общем случае задача планирования – это сложная задача оптимизации по нескольким критериям. Попробуйте, например, придумать, как изменять температуру в пастеризаторе, чтобы обеспечить минимальную себестоимость молочного продукта, максимальный выпуск и наилучшее качество.

Планирование может выполняться вручную (например, специалистом-технологом) или автоматически (например, специализированной компьютерной программой технологической подготовки производства).

Модель объекта связывает его внутренние переменные X с входными воздействиями U, F и измеряемыми величинами Y. Модель может быть простой и даже неформальной:

если приоткрыть кран с горячей водой, температура будет повышаться и выходной сигнал термопары увеличится.

Однако в большинстве случаев используются формальные модели разного вида. Для непрерывных систем, например, используются дифференциальные уравнения, для дискретных – таблично или графически заданные функции.

Модель объекта управления позволяет построить *регулятор*, который будет определять управляющие воздействия $U(t)$ на основе переработки информации о

желаемых (заданных) значениях переменных состояния или выходов;

текущих значениях выходных величин объекта;

текущих значениях возмущений (если они измеряются) (рис. 2.3).

Рис. 2.3. Регулятор.

В самом общем смысле, вырабатывая управляющие сигналы, регулятор выполняет план, т.е. добивается, чтобы текущее состояние объекта было равным желаемому, или, по крайней мере, стремилось к нему.

Таким образом:

Управление = планирование + регулирование (рис. 2.4).

С другой стороны, под регулированием обычно понимают стабилизацию или воспроизведение непрерывных величин (уровень, давление и т.д.). В системах регулирования некоторые величины могут изменяться дискретно, и чаще всего это – управляющие воздействия (выключился нагреватель, включился вентилятор). Однако время «изменяется непрерывно», точнее «непрерывно» принимаются решения.

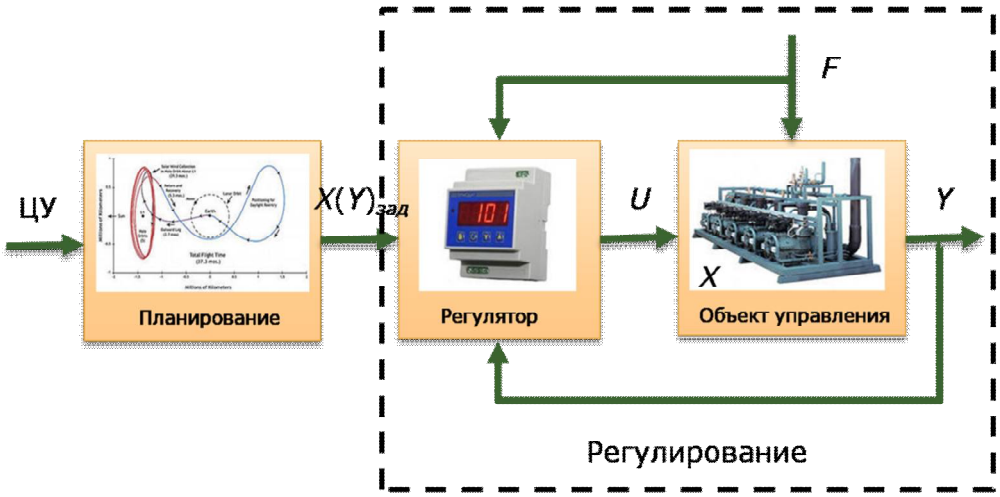


Рис. 2.4. Управление.

Системы управления дискретными операциями называют системами программно-логического управления. В таких системах состояние объекта и системы управления в целом изменяется скачкообразно, причем число возможных состояний ограниченно. Само время в таких системах изменяется как бы от события к событию, точнее так принимаются решения. Наступление какого-либо события приводит к изменению состояния системы и ее выходов:

при нажатии кнопки «Пуск» система переходит в состояние приготовления раствора, включается система подачи ингредиентов…

2.2. Регулирование

2.2.1. Классификация систем автоматического регулирования

В большей части теория управления не занимается вопросами планирования. Считается, что желаемое состояние объекта в любой момент времени известно (определено на более высоком уровне управления). Это не касается «продвинутых» разделов теории, таких, например, как оптимальные и адаптивные системы.

Как уже было сказано, задача регулирования – выполнение «плана», т.е. «воспроизведение» заданного (требуемого) состояния объекта управления. Это состояние характеризуется (описывается) непрерывными величинами, которые должны оставаться неизменными или изменяться во времени так, чтобы технологический процесс протекал наилучшим образом. Задача системы автоматического регулирования – автоматически сформировать необходимые для этого управляющие воздействия. Эту задачу решает регулятор.

Как строятся системы автоматического регулирования и как работают регуляторы? Легче всего это узнать, изучив классификацию систем автоматического регулирования. При этом помимо решения поставленной задачи мы освоим необходимую терминологию.

Системы автоматического регулирования классифицируются по множеству признаков. Мы рассмотрим лишь основные из них.

Классификация по способу управления (на основе каких сигналов формируется управление) показана на рис. 2.5.

Рис. 2.5. Классификация по способу управления.

Простейший вариант – это *разомкнутая САР с управлением по заданию*. В этом случае регулятор формирует управляющий сигнал пропорционально входному задающему сигналу.

Способ может применяться только в случае, если поведение объекта определяется в основном управляющим воздействием и мало зависит от возмущений.

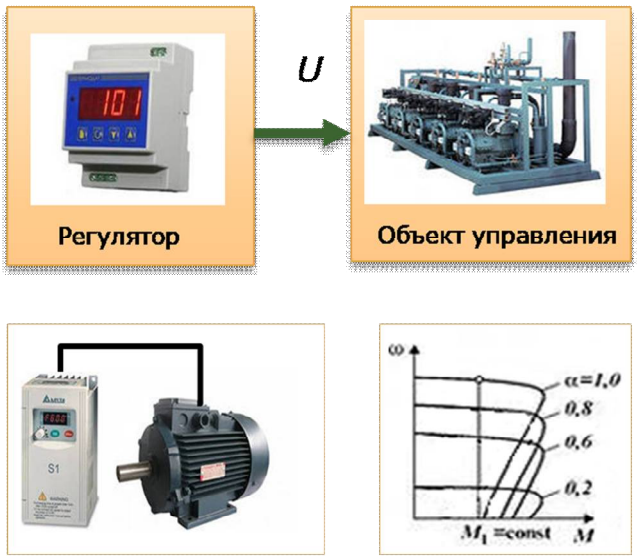


Рис. 2.6. Разомкнутая САР с управлением по заданию.

В качестве примера на рис. 2.6 приведена система «преобразователь частоты – асинхронный двигатель». Регулируемая величина – угловая скорость вращения ротора двигателя. Управляющее, оно же задающее, воздействие – частота питающего напряжения. Эта частота выставляется, например, вручную потенциометром на лицевой панели преобразователя и контролируется на его цифровом дисплее. На рисунке показаны зависимости угловой скорости двигателя от момента сопротивления на его валу при различных частотах (α – это отношение частоты к номинальной, равной в России 50Гц). Если речь идет о двигателях средней и большой мощности, то наклон рабочих участков их механических характеристик очень небольшой: при изменении нагрузки от нуля до максимальной скорость «просаживается» всего лишь на несколько процентов. Поэтому в большинстве случаев можно считать, что скорость определяется только частотой и не зависит от нагрузки. Следовательно, нет необходимости измерять скорость или нагрузку (если, конечно, не рассматривать вариант очень точного регулирования).

В целом же можно сказать, что регулирование по заданию – это, по сути, ручное, а никакое не автоматическое регулирование, а регулятор – вовсе не регулятор, а просто преобразователь задающего сигнала (положение потенциометра) в управляющий (частота).

Разомкнутые САР с управлением по возмущению применяются, если поведение объекта определяется одним или несколькими возмущающими воздействиями, значения которых можно измерить.

Пусть, например, в отдельном здании размещено необслуживаемое оборудование, для нормальной работы которого требуется с некоторой точностью поддерживать определенную температуру воздуха. При постоянных условиях теплообмена (никто не заходит, не открывает окна и т.д.) температура внутри помещения зависит в основном от температуры окружающей среды и расхода теплоносителя через отопительные радиаторы. Разработчику системы регулирования каким-то образом известна зависимость расхода от температуры воздуха «за бортом», обеспечивающая «нормальную» температуру в помещении. Эту зависимость он и реализовал в качестве закона регулирования. Упрощенно этот закон выглядит следующим образом: чем ниже температура «за бортом», тем больше нужно установить расход теплоносителя, «сильнее» открыв регулирующий клапан. Формально закон задается математической формулой, связывающей степень открытия клапана с температурой окружающей среды, или соответствующей таблицей. Регулятор измеряет температуру, вычисляет требуемое положение клапана и устанавливает его в это положение.

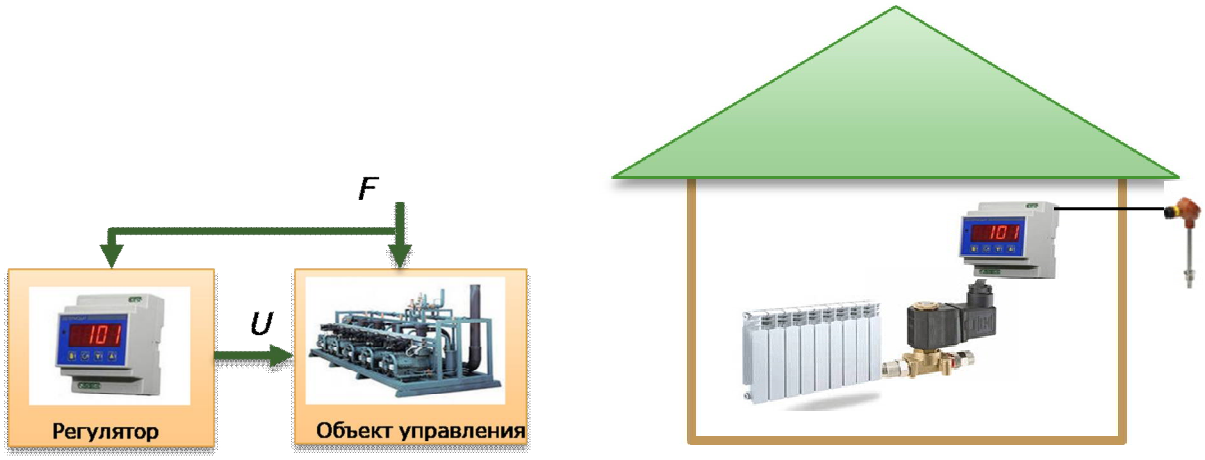


Рис. 2.7. Разомкнутая САР с управлением по возмущению.

На этом самом месте «вдумчивый читатель» должен задать автору несколько вопросов (автор все-таки надеется, что читатель «вдумчивый»).

Первый вопрос (технический, а потому легкий): *как регулятор установит клапан в нужное положение? Если клапан снабжен электроприводом, то регулятор может дать сигнал на его открытие или сигнал на его закрытие, но никак не может дать сигнал «открыть клапан на 30%». Другими словами, нужна система регулирования положения клапана.*

Все верно: такая система нужна и более того, в случае с электрическим приводом она не может быть построена без определения положения. Будем считать, что такая система есть, положение клапана измеряется или каким-то образом вычисляется. Например, задействован специальный прибор контроля положения исполнительного механизма, снабженный импульсным датчиком и способный самостоятельно приводить клапан в заданное положение

по сигналу регулятора. Кроме того, привод клапана может быть и *пневматическим*, например, мембранного или сильфонного типа. Тогда положение клапана определяется управляющим давлением, а это давление формируется *электропневматическим преобразователем* пропорционально выходному сигналу регулятора. Тогда измерение положение клапана вообще не требуется.

Второй вопрос более важный. *Требуемая тепловая мощность радиаторов зависит, конечно, от температуры окружающей среды, но не только от нее. Имеют значение также и другие факторы, например, сила и направление ветра. С другой стороны, очевидно, что выдаваемая радиаторами тепловая мощность пропорциональна положению регулирующего клапана только тогда, когда расходная характеристика клапана (зависимость расхода от степени открытия) линейна, а давление и температура теплоносителя неизменны и равны номинальным. При таком количестве допущений ожидать, что система будет точно поддерживать температуру в помещении, не приходится.*

«Вдумчивый читатель» абсолютно прав. Без измерения температуры непосредственно в помещении построить качественную систему ее регулирования принципиально невозможно. Поэтому основным способом управления является *управление по отклонению* в *замкнутой системе автоматического регулирования*.

В *замкнутой системе, работающей по отклонению*, рис. 2.8, регулятор вырабатывает управляющий сигнал U по *отклонению*, или *ошибке регулирования*, E. Ошибка регулирования – это разность между заданным значением регулируемой величины и ее фактическим, измеренным значением. Закон управления (зависимость $U(E)$) может быть как очень простым, так и достаточно сложным.

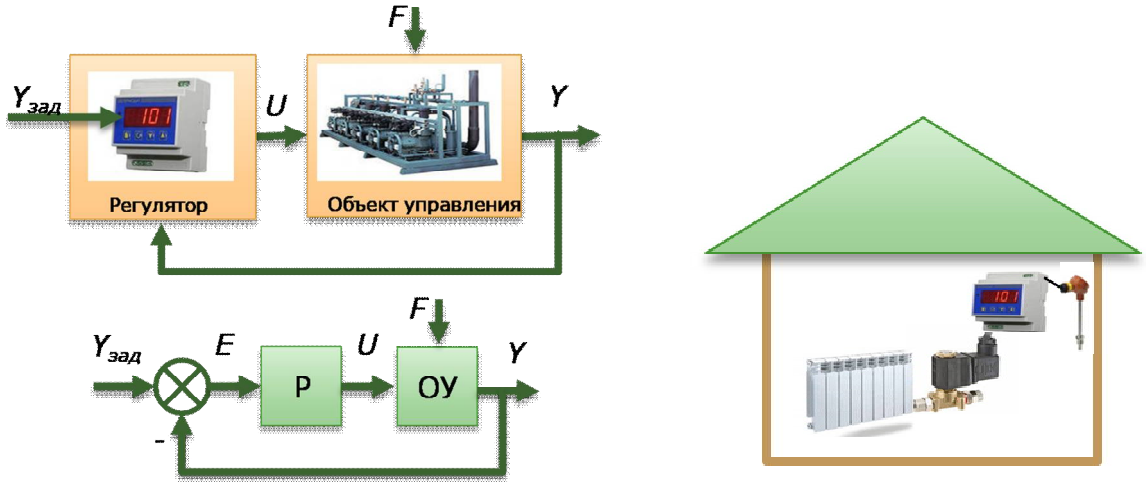


Рис. 2.8. Замкнутая САР с управлением отклонению.

В простейшем случае применяются *релейные регуляторы*, работающие по принципу «включено/выключено». Функционируют они примерно так: температура упала ниже задания на 1°С (ошибка достигла значения 1°С), – регулятор включает нагреватель, температура достигла задания (ошибка снизилась до нуля), – нагреватель отключается. У нас, правда, никакого нагревателя нет, у нас нужно открывать и закрывать клапан. Пусть тогда это будет *запорный* клапан с быстрым *электромагнитным* приводом. Релейные системы просты, но обладают существенным недостатком: в большинстве случаев процесс регулирования имеет колебательный характер. Чтобы повысить точность, нужно снизить амплитуду колебаний, а это можно сделать, только снизив порог срабатывания

реле. Например, включать нагрев при ошибке равной не 1°С, а 0,1°С. Однако такое решение не свободно от недостатков.

Во-первых, в нашем случае оно, скорее всего, просто не сработает, так как амплитуда и частота колебаний температуры в большей степени зависят от инерционных свойств объекта (помещения), а в меньшей – от порога срабатывания реле. Не стоит ожидать того, что при включении нагрева температура тут же пойдет вверх. Нет, если она ранее падала, она еще некоторое время будет падать и только потом начнет расти.

Во-вторых, если и удастся (в других системах) снизить амплитуду колебаний, то очевидно увеличится их частота, а увеличение частоты колебаний означает увеличение частоты срабатывания различных коммутирующих элементов: электромагнитных реле и пускателей, тех же самых электромагнитных клапанов и т.д. Ясно, что это скажется на них не самым благоприятным образом.

Точное регулирование без колебаний возможно в системах *непрерывного регулирования*, в которых регулятор вместо того, чтобы что-то включать и отключать, плавно изменяет («подбирает») мощность управляющего сигнала такую, чтобы ошибка стремилась к нулю. Такие системы всегда сложнее и дороже, чем релейные, потому что вместо дешевой коммутирующей аппаратуры в ней задействована более дорогая преобразовательная техника, а вместо запорной арматуры – регулирующая.

Простейшим законом непрерывного регулирования является пропорциональный, когда управляющее воздействие формируется пропорционально ошибке регулирования. Проще говоря: чем больше ошибка, тем больше воздействие. Применяются и более сложные законы, о чем речь пойдет позднее.

В целом управление по отклонению – способ универсальный, но ему присущ один недостаток: регулятор реагирует на *уже имеющиеся* отклонения, он не может «предсказать» будущее течение процесса. «Предсказать будущее» можно, только измеряя возмущающие воздействия и принимая «контрмеры», еще до того, как эти воздействия приведут к нежелательным изменениям в поведении объекта. Такой подход реализуется в *комбинированных системах,* в которых управление по отклонению дополняется «компенсацией» наиболее «сильных» возмущений (рис. 2.9).

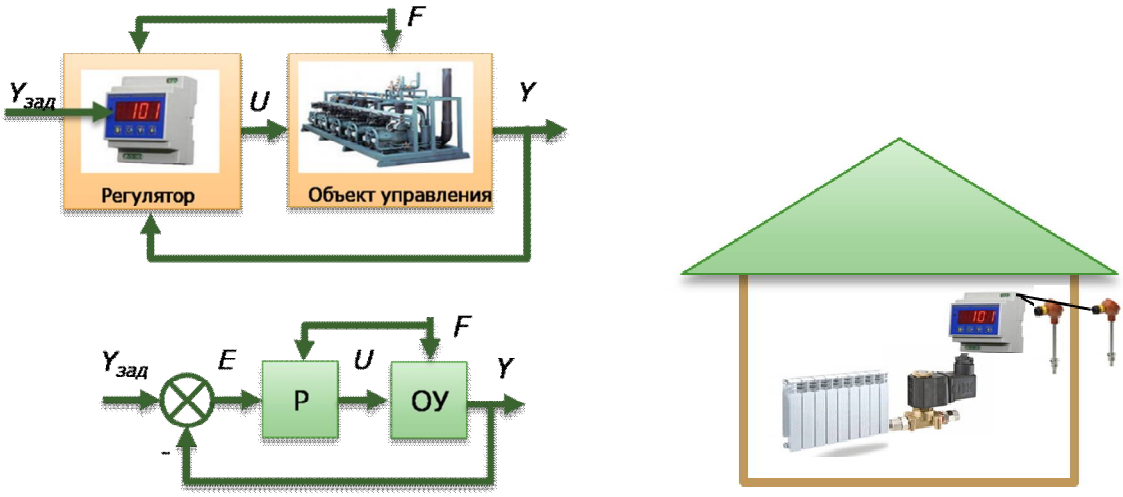


Рис. 2.9. Комбинированная САР.

Построенная по данному принципу система регулирования температуры будет работать следующим образом. «Основной алгоритм» перемещает регулирующий клапан в соответствии с «графиком» в зависимости от температуры окружающей среды. Возникающие под действием неконтролируемых возмущений (посещение людей, усиление ветра и т.д.) отклонения температуры от задания «подавляются» регулятором, работающим по отклонению. Этот регулятор формирует дополнительное перемещение регулирующего клапана. Другими словами, сигналы регуляторов, работающих по возмущению и по отклонению, складываются. Возможны, конечно, и другие варианты организации работы регуляторов в комбинированных системах.

Классификация САР по виду задания показана на рис. 2.10.

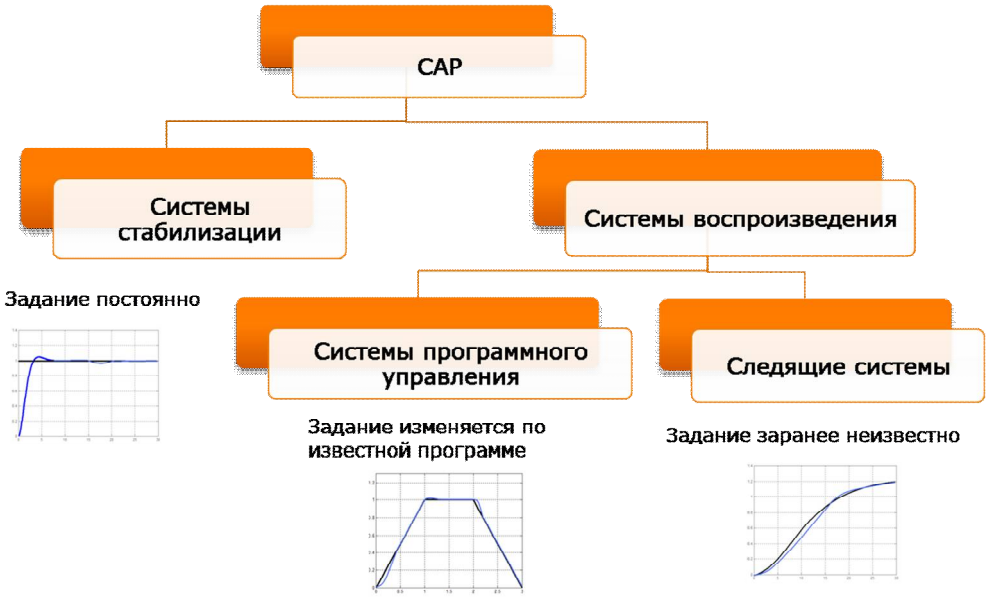


Рис. 2.10. Классификация САР по виду задания.

В зависимости от вида задания САР разделяют на системы стабилизации и системы воспроизведения. В первом случае задание постоянно (условно), во втором – изменяется во времени.

Очень большая часть промышленных систем регулирования являются системами стабилизации. Дело в том, что оптимальное протекание многих технологических процессов возможно только при точном поддержании их параметров на заданных уровнях. Так, например, даже небольшие отклонения давления и температуры пара приводят к значительным изменениям вырабатываемой мощности и КПД паровой турбины (большие же отклонения могут и вовсе привести к серьезным авариям). Поэтому соответствующие регуляторы должны обеспечивать очень точное регулирование давления и температуры в установившихся режимах и не допускать значительных отклонений в переходных (например, при наборе мощности).

Автоматическая стабилизация часто выступает в роли «нулевой ступени» защиты. Если регулятору удается удержать технологический параметр в отведенном для него диапазоне, считается, что процесс протекает нормально. При выходе регулируемой величины из этого диапазона вступают в действие другие

средства: сигнализация, переход на дистанционный режим управления, автоматическая защита.

Не следует считать, что в системах стабилизации задание для регулятора не изменяется вообще: его можно корректировать в узком диапазоне для тех или иных целей. Например, всем известно стандартное значение напряжение в бытовой сети. Однако ГОСТ устанавливает и *нормально допустимые отклонения* напряжения (±5%). Отклонение напряжения от стандартного значения – не обязательно свидетельство плохой работы поставщика электрической энергии. Оно может поддерживаться намерено, для создания условий для перетоков мощности в энергосистеме.

Системы автоматической стабилизации в большинстве случаев используют простые законы регулирования. Это связано с тем, что они работают при малых отклонениях регулируемых переменных от их номинальных значений. В таких условиях объекты управления ведут себя «смирно» в том смысле, что их поведение можно описать простыми моделями. А простым объектам и не нужны сложные регуляторы.

Системы воспроизведения делятся на *системы программного управления* и *следящие*.

В *системах программного управления* задание изменяется по заранее известной (т.е. еще для разработчика самой системы) программе. Например, термическая обработка твердых сплавов предусматривает операции нагрева, выдержки и охлаждения. График изменения задания по температуре во времени похож на трапецию.

В *следящих системах* задание может изменяться заранее неизвестным образом. Человек поворачивает маленькую ручку на пульте управления, а система в ответ поворачивает на тот же угол «тарелку» громадного телескопа. Сама система не может знать, на какой угол повернет ручку человек, и уж конечно, ее разработчику это было и вовсе неизвестно. В общем случае задание может формировать не только человек, но и системы управления более высокого уровня. Например, в случае с телескопом, – это система определения (вычисления) положения небесного тела.

К следящим системам, как правило, предъявляются более жесткие требования, чем к системам программного управления и тем более системам стабилизации. Они должны обладать хорошими динамическими характеристиками: не только точно, но и быстро выполнять изменяющиеся во времени задания. В системах программного управления график изменения задания известен, и разработчик может, по крайней мере, протестировать на модели поведение системы регулирования. Для следящих систем это сделать невозможно, поэтому приходится использовать «самые адекватные» (а, значит и самые сложные) модели объекта управления и обеспечивать характеристики системы с «избытком».

По *виду используемых сигналов* разделяют непрерывные и дискретные системы (рис. 2.11).

Рис. 2.11. Классификация САР по виду используемых сигналов.

В *непрерывных системах* все сигналы непрерывны. В настоящее время в строгом смысле таких систем почти не осталось, так как практически везде задействована цифровая обработка сигналов. Непрерывным или аналоговым является всегда объект управления, поскольку речь идет о регулировании непрерывно изменяющихся величин, кроме того к «непрерывной части» САР можно отнести некоторые механизмы и измерители.

В *дискретных системах* производится процесс *дискретизации* или *квантования*. Этот процесс заключает в замене непрерывного сигнала его дискретными отчетами.

В *релейных системах* сигнал квантуется по уровню. Число уровней чаще всего два или три, но может быть и больше.

Электромагнитное реле может находиться в двух состояниях: «не сработало» и «сработало», им соответствуют два уровня напряжения и тока нагрузки в цепях их контактов. Это так называемое «двухпозиционное реле».

Система коммутации электропривода регулирующего и запорного клапана выдает три возможных сигнала управления: «включить привод на открытие клапана», «включить привод на закрытие клапана», «остановить привод». Это – «трехпозиционное реле».

Система управления главным приводом лифта (без частотного управления) обеспечивает пять вариантов поведения кабины: движение вверх и вниз на максимальной скорости, движение вверх и вниз на пониженной скорости и останов. Это – «пятипозиционное реле».

В *импульсных системах* производится квантование по времени, этот процесс называется импульсной модуляцией. В зависимости от того, какой параметр импульсов изменяется в функции входного непрерывного сигнала, различают амплитудно-импульсную, широтно-импульсную, фазоимпульсную, частотно-импульсную модуляции (АИМ, ШИМ, ФИМ и ЧИМ соответственно). Строго говоря, ШИМ, ФИМ и ЧИМ занимаются также и квантованием по

уровню, так как высота импульсов при этих видах модуляции не изменяется (по крайней мере, по модулю).

В теории управления разработаны специальные методы анализа и синтеза импульсных систем.

В *цифровых системах* квантование осуществляется как по уровню, так и по времени.

Квантование по уровню связано с цифровой формой представления значения сигнала. Пусть, например, вычислением управляющего сигнала занимается 8-ми разрядный микропроцессор. Тогда число возможных уровней цифрового сигнала составляет $2^8 = 256$. Современные ПЛК и промышленные компьютеры способны обрабатывать 16-ти и 32-разрядные целочисленные слова и даже дробные числа в представлениях с фиксированной и плавающей точкой, а число уровней дискретизации определяется разрядностью аналого-цифровых и цифроаналоговых преобразователей (АЦП и ЦАП). Самые распространенные АЦП – 12-ти разрядные. Менее распространенные – 14-ти и 16-ти разрядные. ЦАП обычно имеют разрядность 10-12, реже – 14. Таким образом, число уровней довольно велико и в этом плане цифровой сигнал практически не отличим от аналогового. Поэтому при анализе и синтезе САР квантованием по уровню практически всегда можно пренебречь.

Квантование по времени связано с циклическим характером работы цифровой управляющей аппаратуры. Упрощенно цикл работы ПЛК включает «считывание» входов, выполнение управляющей программы и «обновление» выходов. В большинстве случаев время цикла не фиксировано и зависит от «длины» программы. Есть, конечно, сторожевой таймер, Watchdog timer, перезагружающий контроллер, если выполнение программы «слишком затянулось», но он – всего лишь средство защиты от логических ошибок в программе. В «нормальных условиях» время цикла может быть совсем небольшим, порядка одной или нескольких миллисекунд. Однако это совсем не значит, что частота дискретизации достигает 1кГц. Самое «слабое звено» ПЛК и модулей ввода с точки зрения быстродействия – это АЦП. Конечно, существуют «скоростные» преобразователи, но далеко не все ПЛК оснащены ими (это было бы излишне дорого). Время опроса аналоговых входов некоторых современных ПЛК может достигать несколько десятых долей секунды. Допустим, программа пересчитывается 1000 раз в секунду, а время опроса аналогового входа составляет 0,5 с. Это означает, что в 998 случаев из 1000 программа обрабатывает «старые» данные, т.е., по сути, работает «вхолостую»! Таким образом, реальная частота дискретизации определяется самым медленным устройством, т.е. АЦП, и может быть даже в современных системах весьма небольшой. При этом если речь идет о регулировании быстроизменяющихся параметров, поведение цифровой системы будет существенно отличаться от ее аналогового «эквивалента», причем в худшую сторону. В таких условиях систему рассматривают как импульсную с АИМ.

В целом можно сказать, что классификация САР по виду используемых сигналов напрямую не связана с составом технических средств системы. Принадлежность САР к тому или другому классу говорит о логике ее работы, ис-

пользуемых алгоритмах регулирования и методах теории управления, задействованных при анализе и синтезе.

Так, например, в промышленности широко применяются системы регулирования с ШИМ. Можно рассмотреть систему регулирования температуры с нагревательным элементом. Допустим, закон регулирования в данный момент требует подачи на нагревательный элемент 50% мощности. А «технически» система построена так, что мощность нагревательного элемента изменять нельзя, его можно только включить или выключить. Если сигнал нельзя «разделить» по уровню, почему бы не «разделить» его по времени? Может быть, объект управления даже и не заметит такой подмены, ведь температура меняется медленно? Так и поступают: нагревательный элемент включают периодически, например, один раз в 30 с, при этом первые 15 с он включен, а вторые выключен. Разумеется, если потребуется увеличить мощность до 66,(6)%, время включения нужно будет увеличить до 20 с, а время отключения – уменьшить до 10 с. По такому же принципу, кстати, построена система управления мощностью обыкновенной микроволновой печи. Все такие системы по логике своей работы являются непрерывными (или хотя бы «*квази*непрерывными»), хотя в них есть и «реле» и «импульсы».

Широко применяемые в несложных системах автоматизации микропроцессорные регуляторы и контроллеры часто реализуют релейные законы регулирования (т.е., по сути, что-то включают при одном значении ошибки регулирования и выключают при другом). Поэтому соответствующие системы регулирования являются релейными, а не цифровыми.

Расчет типовых регуляторов, впоследствии реализуемых программно, часто производят методами «непрерывной теории», и если квантование по времени практически не сказывается на качестве процесса регулирования, полученные системы вполне можно считать непрерывными.

По виду математической зависимости выхода от входа принято разделять системы на *линейные* и *нелинейные*.

Для линейных систем применим так называемый *принцип суперпозиции*: *реакция системы на линейную комбинацию входных воздействий равна линейной комбинации реакций на входные воздействия.* Этот принцип родился и живет исключительно в голове человека, ибо в природе он нигде, или почти нигде, в точности не выполняется. Человеку же проще с помощью него воспринимать реальность. В основу принципа положено два утверждения:

1) факторы (сигналы, возмущения) действуют на объект (систему) независимо друг от друга, произведенные ими эффекты (реакции) складываются;

2) эффект пропорционален силе фактора (рис. 2.12).

На самом деле линейных систем нет, так как оба этих утверждения в общем случае неверны. Факторы обычно имеют разную физическую природу и эффект от одного из них зависит от силы других, причем даже при неизменности последних этот эффект не пропорционален силе фактора. Мир в целом нелинеен, и объекты управления не являются исключением. Невозможно в принципе построить и линейный регулятор, хотя бы потому что мощность сигнала управления всегда ограничена.

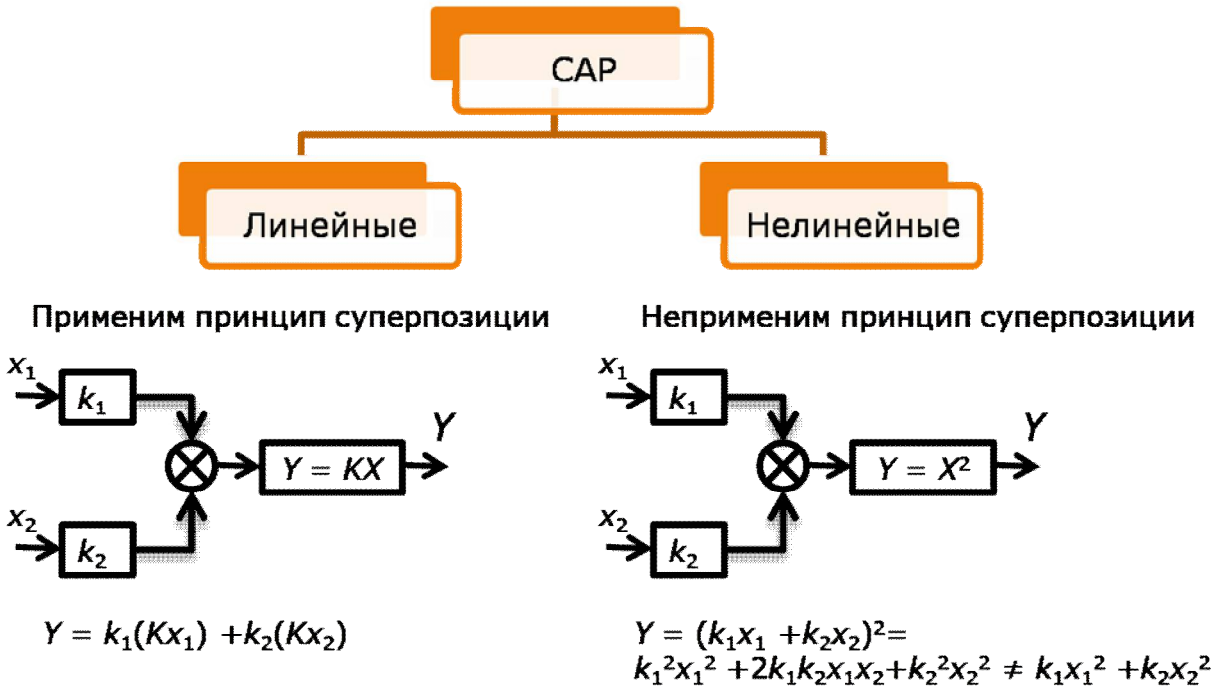


Рис. 2.12. Классификация САР по виду математической зависимости выхода от входа.

Однако не все так плохо, и принцип суперпозиции имеет вполне определенное отношение к реальности. Более того, он даже математически обоснован. Помните, конечно, такие слова, как «дифференциал», «ряд Тейлора», «частное и полное приращение функции нескольких переменных» и т.д.? Если рассматривать мир малых изменений, тот этот «микромир» оказывается линейным. При небольших изменениях переменных нелинейная система (правда, не всякая, а только такая, в которой все переменные изменяются непрерывно, без «скачков») ведет себя как линейная. Таким образом, линейные модели хорошо подходят для систем стабилизации, в которых отклонения переменных от их номинальных значений невелики.

Общая теория линейных систем управления разработана, а теория нелинейных – нет, и никогда не будет. Дело в том, что можно, например, дать определение *слона*, но нельзя дать определения «*неслона*». В попытках определить «неслона» можно дать определение бегемоту, шимпанзе, акуле, но все равно не получится «неслон».

Теория линейных систем, основанная на принципе суперпозиции, оперирует обыкновенными дифференциальными уравнениями и передаточными функциями, которые сами по себе могут описывать только линейные объекты. Благодаря такому математическому аппарату разработаны общие методы анализа и синтеза систем.

В теории нелинейных систем общего математического аппарата нет, поэтому она решает частные задачи и рассматривает отдельные классы систем, например, релейные. Существует, по-видимому, единственный более или менее надежный способ исследования нелинейных систем – это имитационное компьютерное моделирование.

По количеству регулируемых величин системы автоматического регулирования разделяют на одномерные и многомерные (рис. 2.13).

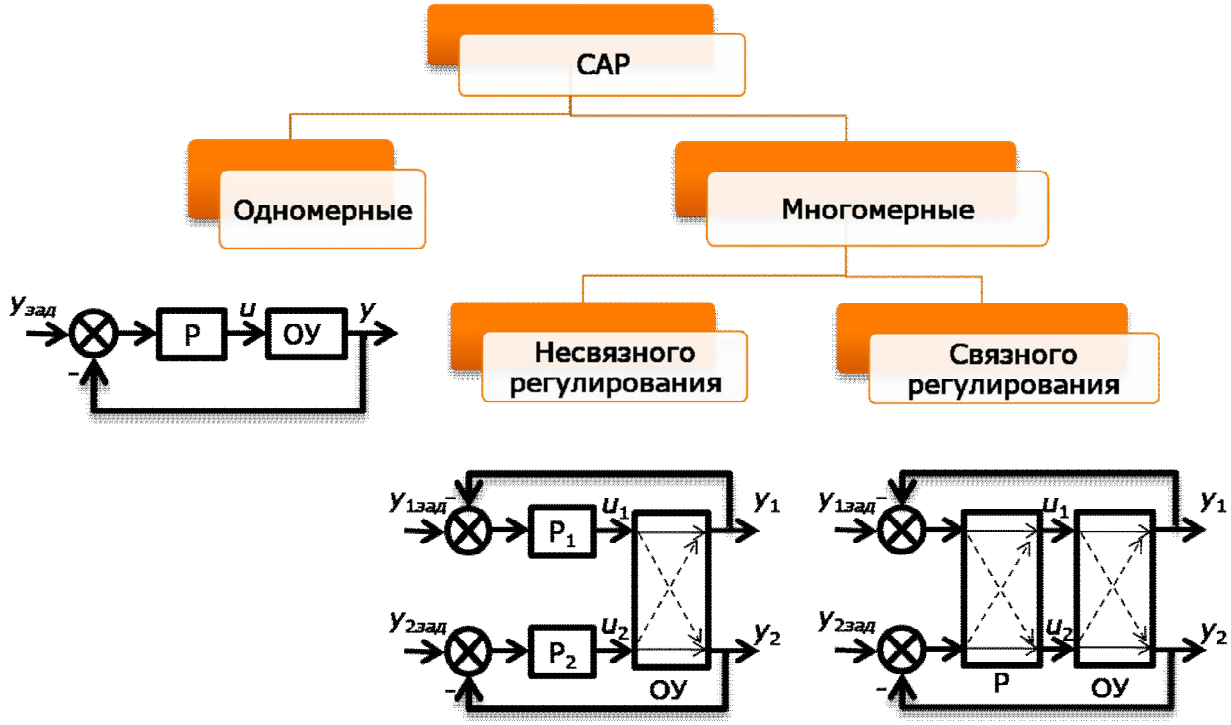


Рис. 2.13. Классификация САР по количеству регулируемых величин.

В многомерных системах регулируемых величин несколько. Управляющих воздействий, как правило, но не всегда, столько же, сколько и регулируемых величин. За каждой регулируемой величиной «закреплено» свое управление. В подогревателе сетевой воды, применяемом на ТЭЦ, регулируемыми величинами являются температура воды на выходе и уровень конденсата в аппарате. Температуру регулируют, изменяя расход греющего пара, а уровень конденсата – изменяя его расход на сливе. При этом важно то обстоятельство, что объект регулирования один, его нельзя разделить на две независимые части, поскольку имеет место «перекрестное» влияние управляющих воздействий на «чужие» регулируемые величины. Так, изменение расхода пара приводит к изменению количества конденсата и его уровня, а изменение расхода конденсата, через изменение его уровня и условий теплообмена, – к изменению температуры воды на выходе из аппарата.

Иногда число управляющих воздействий меньше числа регулируемых величин. В качестве примера можно привести паровую турбину. Угловая скорость вращения (и, следовательно, частота электрического тока) и активная мощность, вырабатываемая генератором, регулируются путем изменения расхода пара, подаваемого в турбину. Никакого противоречия здесь нет, поскольку не существует жесткой зависимости между расходом пара и скоростью вращения ротора турбины, а между расходом пара и мощностью она есть. Быстрая система регулирования угловой скорости поддерживает частоту вращения неизменной, быстро, но незначительно изменяя расход пара, а медленная система регулирования мощности медленно, но сильно изменяет расход пара, поддерживая мощность. Здесь уместна следующая аналогия. Водитель с помощью пе-

дали газа (или система «круиз-контроль», автоматически) поддерживает постоянную скорость автомобиля, несмотря на то, что наклон дороги медленно меняется. При этом на прямых участках двигатель машины развивает небольшую мощность (требуется небольшой вращающий момент), а на подъеме мощность увеличивается.

На практике для многосвязных объектов в большинстве случаев применяются *системы несвязного регулирования*, состоящие из нескольких «независимых» контуров. При расчете регуляторов перекрестные связи в объекте могут вообще не учитывать, считая их возмущениями. Если эти связи слабы, полученные системы могут быть вполне работоспособны. В случае сильных перекрестных связей возможны неприятные последствия в виде затянутых и колебательных переходных процессов и даже неустойчивой работы системы. Причина состоит в том, что регулятор, рассчитанный для простого объекта, получает в «довесок» к нему соседний контур, через который на выход объекта проникает его весьма сложно «обработанный» управляющий сигнал. В совокупности реальный объект становится гораздо сложнее и с ним «простому» регулятору трудно справиться. В качестве примера рассмотрим следующую «чисто гипотетическую» ситуацию.

Вы, стоя под душем, регулируете температуру воды. Почему-то температура воды уменьшается, Вы увеличиваете подачу горячей воды, приоткрывая соответствующий вентиль. По причине того, что система ГВС устроена «не совсем правильно», давление в линии горячей воды падает. Система регулирования давления, расположенная в тепловом пункте, увеличивает обороты циркуляционного насоса в контуре ГВС, расход горячей воды увеличивается. В результате Вы получаете «двойной эффект»: мало того, что вы, приоткрыв клапан, уменьшили гидравлическое сопротивление линии, так еще и насос увеличил подачу. Температура начала расти, но так быстро, как Вы (регулятор) совсем не ожидали.

Существуют методы расчета «сложных» регуляторов, учитывающие наличие соседних контуров. Однако более перспективными являются *системы связного управления*. В них регулятор один, но он формирует сразу все управляющие сигналы на основе информации обо всех ошибках регулирования. Внутри регулятора налаживаются свои перекрестные связи. Основной идеей при этом является компенсация действия перекрестных связей внутри самого объекта.

По постоянству параметров (математического описания) системы автоматического регулирования делятся на *стационарные* и *нестационарные* (рис. 2.14).

Под параметрами здесь понимаются не переменные/сигналы, а различные физические характеристики и коэффициенты, входящие в уравнения, описывающие в объект. В стационарных системах они неизменны, в нестационарных – изменяются во времени или в зависимости от режима работы. По большей части понятие стационарности относится к объекту управления.

На рис. 2.14 нестационарность объекта регулирования уровня связана с изменением параметра S (площади поверхности) при изменении уровня.

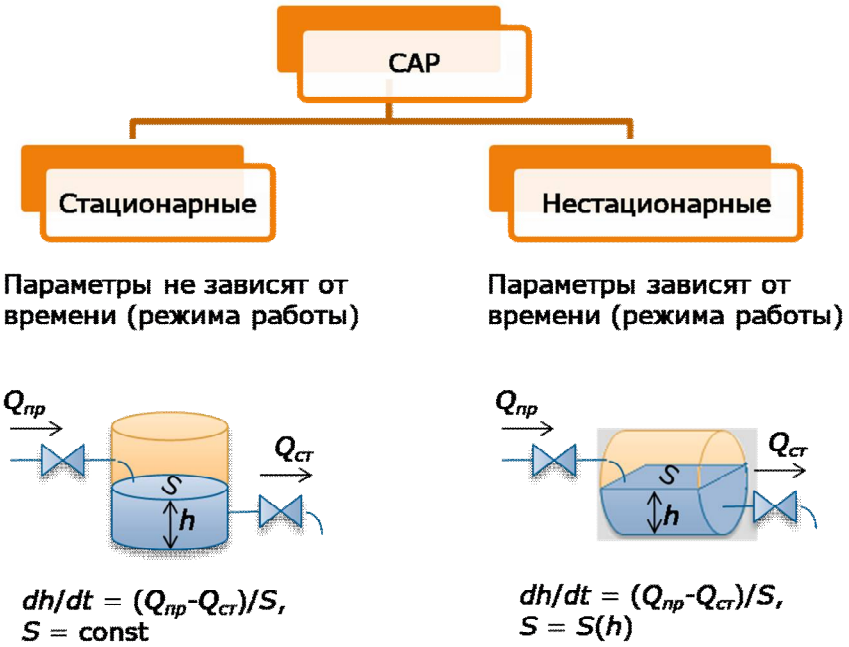


Рис. 2.14. Классификация САР по постоянству параметров.

При малых и больших уровнях под действием разницы расходов на притоке и стоке бак будет заполняться или опустошаться быстро, при среднем уровне – медленно. Если зависимость $S(h)$ внести в дифференциальное уравнение, оно станет нелинейным. Таким образом, нестационарность объекта в данном случае обусловлена нелинейностью. В других случаях изменения параметров связано с параллельно протекающими процессами во «внешнем мире». Так, например:

«чувствительность» самолета к положению рулей и элеронов очень сильно зависит от высоты и скорости полета;

при нагреве электрического двигателя значительно увеличивается активное сопротивление его обмотки, что приводит к изменению его динамических свойств и т.д.

Нестационарность объекта – это, безусловно, вредное явление, мешающее регулированию. Для «борьбы» с нестационарностью существуют два принципиально разных подхода.

Во-первых, можно попытаться построить такой регулятор, который будет обеспечивать удовлетворительную работу системы при всех возможных значениях параметров объекта. Такой регулятор называют «грубым», т.е. нечувствительным, или «робастным» (от англ. robust).

Во-вторых, можно построить адаптивный (приспосабливающийся) регулятор, который будет менять свои настройки при изменении параметров объекта. Причем последние могут измеряться или оцениваться.

2.2.2. Что такое передаточная функция и почему не нужно ее бояться

2.2.2.1. Интегрирование и дифференцирование

Как было показано в предыдущей главе, управление непрерывными процессами заключается в регулировании различных физических величин. Выбор алгоритма (закона) регулирования и его параметров невозможно сделать без

знания модели объекта управления. В простейших случаях эта модель может быть даже неформальной (например: «если включить нагреватель, температура будет расти, а если выключить – падать»), но без какой-либо модели организовать управление нельзя в принципе.

Промышленные объекты регулирования в большинстве своем являются сложными и не допускают применения неформальных моделей, поскольку построенные на их основе системы регулирования оказываются неработоспособными или, по крайней мере, работают плохо: неточно, медленно и т.д. Сложные объекты требуют сложного описания, и здесь уже без математики не обойтись. При этом большинство объектов являются *динамическими*, т.е. их поведение не только зависит от внешних воздействий, но и «развивается во времени».

Для математического описания динамических объектов управления используются *дифференциальные уравнения* и их аналоги – *передаточные функции.*

Не стоит впадать в панику. Все не так страшно, как кажется на первый взгляд. При всей сложности объектов и систем регулирования дифференциальные уравнения и передаточные функции, по сути, описывают всего лишь два физических явления, которые наблюдаются в объектах и системах, а именно *инерционность* и *временную задержку в передаче сигналов*.

Все объекты и системы управления в целом обладают инерцией: их состояние не может мгновенно изменяться под действием внешних воздействий. Другими словами: текущее состояние инерционного объекта зависит не только от внешних воздействий, оказываемых на них в данный момент времени, но и от предыдущего состояния самого объекта.

Инерционность объекта:

мешает управлению, поскольку замедляет реакцию объекта на управляющие сигналы;

помогает управлению, поскольку замедляет реакцию объекта на возмущения.

Инерционность связана с процессами накопления в объекте материальной среды или энергии:

в баке и ресивере накапливаются жидкость и газ, вследствие чего уровень жидкости и давление газа не могут изменяться мгновенно («катастрофы» в виде взрывов не принимаем во внимание);

в конденсаторе и катушке индуктивности накапливаются энергии электростатического (заряд) и электромагнитного полей, поэтому напряжение на конденсаторе и ток катушки также не могут мгновенно изменяться;

любое материальное тело, движущееся в пространстве, накапливает механическую энергию, его положение и скорость могут изменяться только *плавно*;

любое нагретое тело накапливает тепловую энергию, его температура также не может изменяться скачками и т.д.

Накопление вещества или энергии описывается математической операцией *интегрирования.* Эта операция очень наглядно демонстрируется на примере наполнения и опустошения емкости (бака) (рис. 2.15).

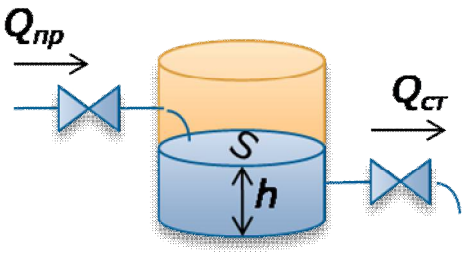


Рис. 2.15. К демонстрации операции интегрирования:
$Q_{пр}$ – объемный расход притока;
$Q_{ст}$ – объемный расход стока;
S – площадь поверхности жидкости.

Если расходы на притоке и на стоке не равны друг другу, уровень изменяется: растет, если $Q_{пр} > Q_{ст}$, и уменьшается, если $Q_{пр} < Q_{ст}$.

Рассмотрим ситуацию, когда разница расходов $\Delta Q = Q_{пр} - Q_{ст}$ изменяется во времени (рис 2.16).

Пусть в начальный момент времени уровень равен нулю. Под действием максимальной положительной разницы расходов уровень начинает быстро расти; при нулевой разнице скорость роста падает до нуля, а сам уровень достигает максимума, далее уровень снижается до нуля под действием отрицательной разности. В каждый момент времени скорость изменения уровня пропорциональна разности расходов, а сам уровень – *накопленным ранее* объемом жидкости или *проинтегрированной* разницей объемных расходов. Итоговое изменение уровня (равное нулю) пропорционально окрашенной «площади» функции $\Delta Q(t)$ на рис. 2.16 с учетом того, что при $\Delta Q(t) > 0$ соответствующая часть площади «положительна», а при $\Delta Q(t) < 0$ – «отрицательна». То же самое справедливо для любого промежуточного момента времени.

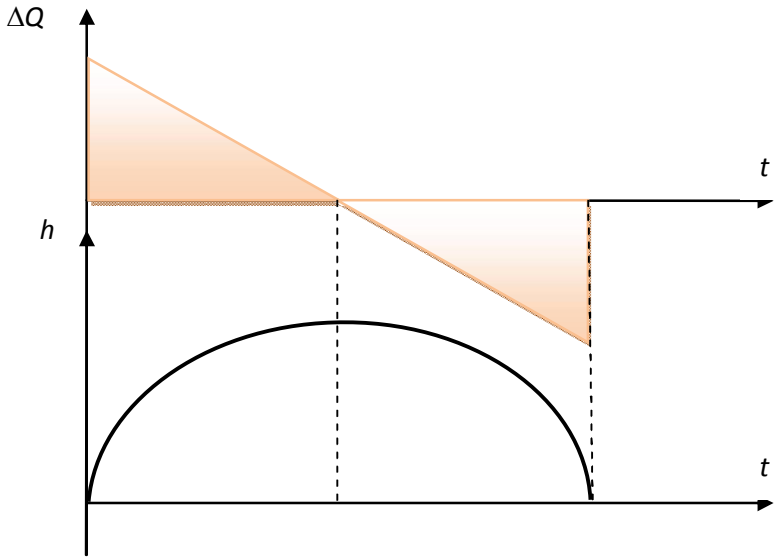


Рис. 2.16. Процесс наполнения и опустошения емкости.

Именно так «работает» интегрирование. Формально связь между уровнем и разницей расходов устанавливается следующим образом.

Изменение объема жидкости в баке dV за время $d\tau$ можно определить как

$$dV = (Q_{пр} - Q_{ст})d\tau.$$

Тогда изменение уровня равно

$dh = dV/S = (Q_{пр} - Q_{ст})\, d\tau /S.$

Интегрируя, для момента времени t, получим:

$$h = \int_0^t dh = \frac{1}{S}\int_0^t (Q_{пр} - Q_{ст}) d\tau .$$

Таким образом, уровень *пропорционален интегралу* от разницы расходов, т.е., возвращаясь к «физике», просто накопленному объему воды. Интегрирование – это всего лишь накопление и опустошение.

Возникает резонный вопрос: а где же здесь тогда дифференциальные уравнения? Дифференциальные уравнения оперируют *производными* как результатами операции *дифференцирования*, а его здесь нет.

Все верно: никакого дифференцирования ни здесь, ни в природе вообще нет. Ни в одном объекте управления реально «не производится» операции дифференцирования, более того, эта операция вообще считается физически нереализуемой! Однако в алгоритмах регулирования эту операцию все-таки «пытаются» реализовать – точнее, реализуют приближенно. Об этом пойдет речь далее.

Сами дифференциальные уравнения объектов и систем управления являются, по сути, просто другой (и часто более удобной!) формой записи интегральных уравнений. Операция дифференцирования (взятия производной) обратна операции интегрирования. Если «цель» интегрирования – определить накопленное ранее значение, то «цель» дифференцирования – найти скорость изменения величины (учитывая, что в нашем случае независимой величиной является время).

В дифференциальном виде уравнение, описывающее поведение уровня, можно записать следующим образом:

$$\frac{dh}{dt} = \frac{1}{S}(Q_{пр} - Q_{ст}) .$$

Теперь это уравнение связывает производную по времени, т.е. скорость изменения уровня с разностью расходов. Эта скорость прямо пропорциональна разности расходов. При положительной разнице она положительна, и уровень растет, при отрицательной – отрицательна, и уровень падает.

Объекты и системы управления принято описывать структурными схемами, показывающими прохождение и преобразование сигналов. Каждый блок, входящий в состав структурной схемы производит некоторое преобразование сигнала.

В линейных объектах и системах осуществляется всего три типа преобразований:

суммирование (включая вычитание, как суммирование с обратным знаком);

масштабирование (умножение на постоянный коэффициент);

интегрирование (накопление).

Все эти преобразования задействованы в схеме, приведенной на рис. 2.17.

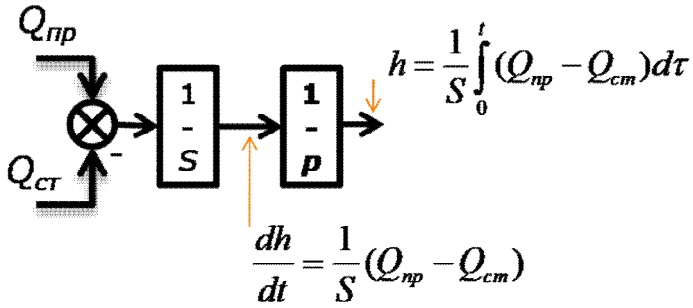


Рис. 2.17. Структурная схема.

На выходе сумматора формируется сигнал разности расходов $Q_{пр} - Q_{ст}$. Далее этот сигнал умножается на постоянный коэффициент $1/S$ и интегрируется интегратором.

Почему интегратор обозначен как «$1/p$»? Дело в том, что интегрирование, как уже упоминалось, является операцией, обратной дифференцированию, а *операторное представление* дифференцирования принято выполнять с помощью символа «p». Другими словами, «p» – это оператор или *символ* дифференцирования. Соответственно, «$1/p$» – это оператор или символ интегрирования.

Структурная схема читается таким образом, что сигнал, пройдя некоторый блок, как бы умножается на некоторый оператор, хотя на самом деле никакого умножения может и не происходить. Так, если некоторый сигнал y есть скорость изменения (производная по времени) другого сигнала x, т.е. $y = dx/dt$, то в операторной форме можно записать: $y = px$. С другой стороны, сигнал x является интегралом (накопленным значением) сигнала y, поэтому можно записать: $x = (1/p)y$.

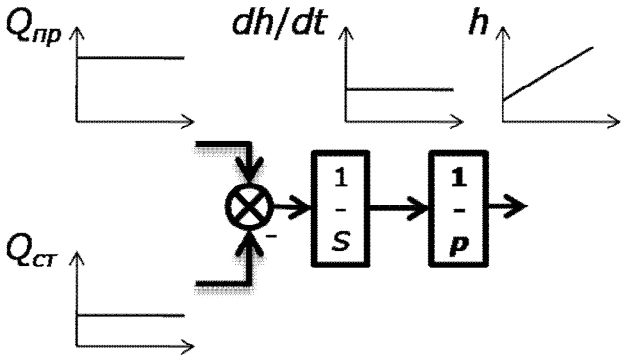


Рис. 2.18. Увеличение уровня.

Выходной сигнал интегратора увеличивается, если сигнал на его входе больше нуля (рис. 2.18).

Выходной сигнал интегратора уменьшается, если входной сигнал меньше нуля (рис. 2.19).

Выход интегратора остается неизменным, если сигнал на его входе равен нулю (рис. 2.20).

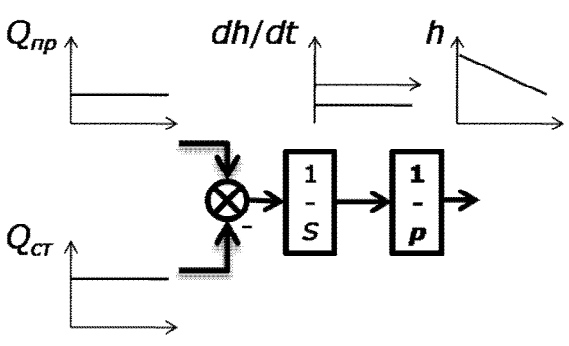


Рис. 2.19. Уменьшение уровня.

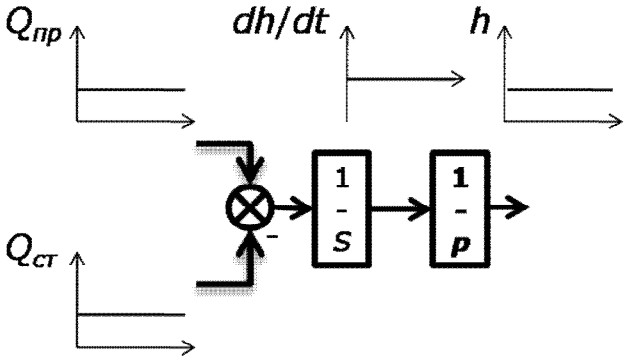


Рис. 2.20. Стабилизация уровня.

2.2.2.2. Самовыравнивание

Рассмотренный выше объект регулирования уровня – это объект *без самовыравнивания*: при снятии внешнего воздействия он не возвращается в исходное состояние. Если перестать наполнять и опустошать бак, уровень жидкости останется неизменным. Многие объекты управления «устроены» по-другому: при снятии воздействия они сами возвращаются в исходный режим.

Рассмотрим объект с самовыравниванием: процесс нагрева однородного тела. Тело нагревается некоторым нагревателем. Оно характеризуется теплоемкостью C, Дж/°С и коэффициентом теплоотдачи A, Дж/(°С×сек).

Теплоемкость – это количество тепла, требуемое для нагрева тела на 1°С.

Коэффициент теплоотдачи – количество тепла, отдаваемое телом в единицу времени в окружающею среду при превышении его температуры над температурой среды на 1°С.

К телу подводится тепловая энергия мощностью P, Вт. Состояние тела описывается превышением температуры Ω (разностью температур тела и окружающей среды).

Составим уравнение теплового баланса.

Тепло, подводимое к телу за время $d\tau$ (левая часть уравнения)

1) частично аккумулируется самим телом, что приводит к увеличению его температуры на $d\Omega$ градусов;

2) частично отдается в окружающую среду:

$P \times d\tau$	=	$C \times d\Omega$	+	$A \times \Omega \times d\tau$
Вт × сек = Дж		Дж/°С × °С = Дж		Дж/(°С×сек) × °С × сек = Дж
подводимое тепло		*аккумулируется телом*		*отдается в окружающую среду*

Поделив левую и правую части уравнения на $d\tau$, сразу получим дифференциальное уравнение объекта:

$$P = C\frac{d\Omega}{dt} + A\Omega,$$

$$\frac{C}{A}\frac{d\Omega}{dt} + \Omega = \frac{1}{A}P, \text{ или } T\frac{d\Omega}{dt} + \Omega = kP,$$

где T – *тепловая постоянная времени*, с; k – *коэффициент передачи*, °С/Вт.

Решение этого дифференциального уравнения при постоянной мощности нагревателя выглядит следующим образом:

$$\Omega = kP + \left(\Omega(0) - kP\right)e^{-t/T},$$

где Ω(0) – начальное значение превышения температуры.

При Ω(0) = 0 получим:

$$\Omega = kP\left(1 - e^{-t/T}\right).$$

График изменения температуры при Ω(0) = 0 и P = const показан на рис. 2.21.

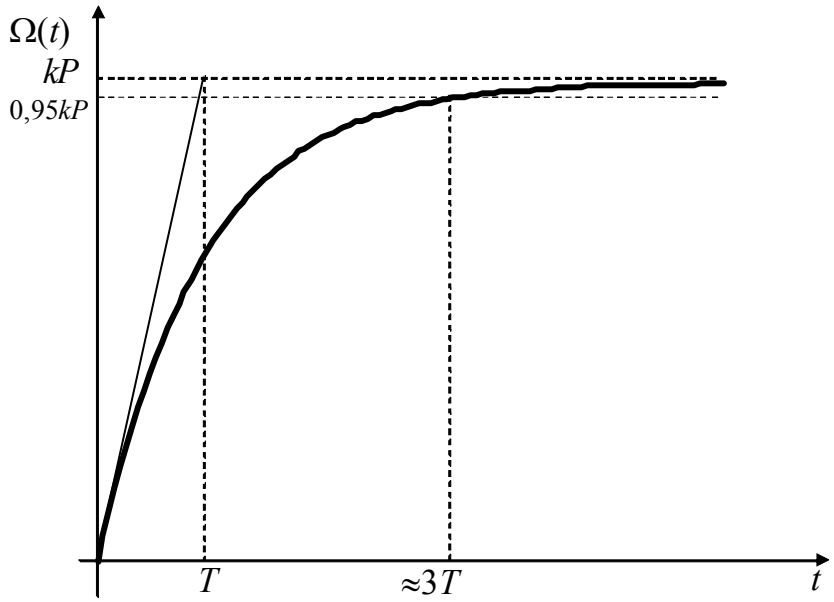


Рис. 2.21. График изменения температуры.

Процесс нагрева «*заканчивается*»: температура достигает некоторого (*установившегося*) значения, соответствующего мощности нагревателя и больше не растет. Выразим из дифференциального уравнения скорость изменения температуры:

$$\frac{d\Omega}{dt} = \frac{k}{T}P - \frac{1}{T}\Omega .$$

Выразим из уравнения теплового баланса изменение температуры $d\Omega$:

$$d\Omega = (P/C - A/C\,\Omega)\times d\tau$$

Интегрированием получим:

$$\Omega = \int_0^t \left(\frac{1}{C}P - \frac{A}{C}\Omega\right)d\tau = \int_0^t \left(\frac{k}{T}P - \frac{1}{T}\Omega\right)d\tau .$$

То же самое можно получить, интегрируя дифференциальное уравнение.

По уравнению для превышения температуры можно построить структурную схему (модель) объекта (рис. 2.22).

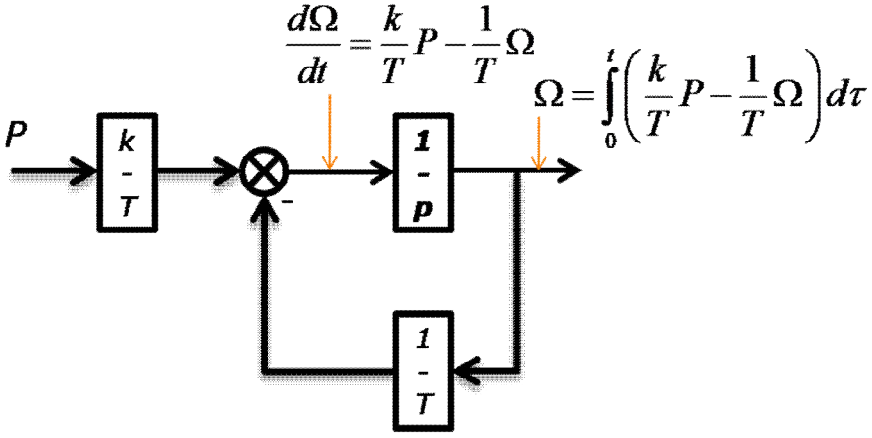


Рис. 2.22. Модель объекта.

Исходным блоком для построения модели является интегратор. На его выходе – превышение температуры, а на входе – скорость его изменения. Из дифференциального уравнения интегратор «делает» интегральное. Остальные блоки решают «технические задачи», обеспечивая «сборку» уравнения.

Важным является то, что в отличие от примера с баком, в данной модели интегратор оказался *охваченным обратной связью* по выходной переменной. Действие этой обратной связи и обеспечило эффект самовыравнивания. Рассмотрим процесс нагрева, анализируя изменения сигналов модели (рис. 2.23).

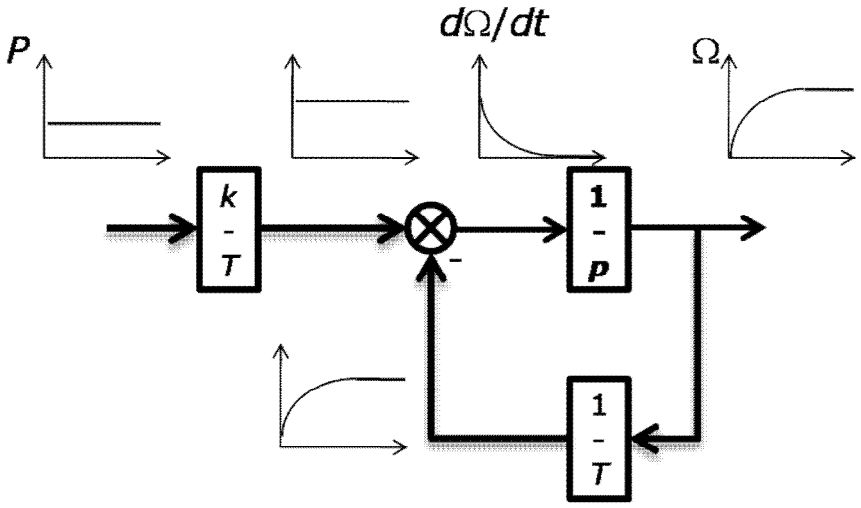


Рис. 2.23. Процесс нагрева.

В начале процесса тело было «холодным» и не отдавало тепло в окружающую среду. Вся подведенная энергия расходовалась на нагрев тела. Поэтому скорость роста температуры была максимальной.

По мере увеличения температуры все большая часть подведенной энергии стала отдаваться в окружающую среду, а все меньшая – расходоваться на нагрев. Скорость роста температуры постепенно уменьшается.

В конце процесса вся подведенная энергия отдается в окружающую среду и температура не растет.

Таким образом, объект с самовыравниванием самостоятельно «организует» *баланс входных воздействий* на входе интегратора за счет внутренней обратной связи.

В *статическом режиме* (когда входные воздействия и выходные величины неизменны), связь между выходами и входами определяет коэффициент передачи:

$$\Omega = kP = (1/A)P.$$

Это уравнение можно записать в виде

$$A\,\Omega = P,\ \text{Дж}/(°\text{C}\times\text{сек}) \times °\text{C} = \text{Дж}/\text{сек}.$$

Количество энергии, отдаваемой в окружающую среду в единицу времени (мощность тепловых потерь) равно количеству энергии, отдаваемой телу нагревателем в единицу времени (мощности нагревателя).

Постоянная времени T имеет вполне определенный физический смысл: это время, в течение которого выходная величина достигла бы установившегося значения, если бы изменялась с постоянной начальной скоростью. В приведенном примере это означает отсутствие теплоотдачи в окружающую среду (обратной связи). В этом случае уравнение теплового баланса имело бы вид

$$P \times d\tau = C \times d\Omega,$$

откуда

$$d\Omega/d\tau = P/C = (k/T)P.$$

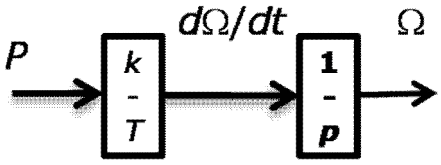


Рис. 2.24. Структурная схема.

Структурная схема, соответствующая данному уравнению, показана на рис. 2.24.

Решение уравнения при $\Omega(0) = 0$:

$$\Omega = (k/T)P \times t.$$

Если бы нагрев происходил с постоянной начальной скоростью, установившееся значение температуры $\Omega = kP$ действительно было бы достигнуто за время T.

Дифференциальное уравнение связывает между собой превышение температуры Ω, скорость его изменения $d\Omega/dt$ и мощность нагревателя P. Оно имеет *первый порядок*: порядок старшей производной равен единице.

2.2.2.3. Дифференциальные уравнения и структурные схемы: построение и анализ

В общем случае дифференциальное уравнение n-го порядка связывает между собой производные выходного сигнала y от первой до n-й включительно, само значение выходного сигнала, а также производные входного сигнала u от первой до m-й включительно и само значение входного сигнала:

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \ldots + a_1 \frac{dy}{dt} + a_0 y = b_m \frac{d^m u}{dt^m} + b_{m-1} \frac{d^{m-1} u}{dt^{m-1}} + \ldots + b_1 \frac{du}{dt} + b_0 u,$$

где a_i, b_i – постоянные коэффициенты (параметры модели).

Физический смысл второй производной – это скорость изменения скорости сигнала (первой производной), т.е. ускорение. Третья производная (скорость изменения ускорения) называется *рывком*. Для производных выше третьего порядка общепринятых названий не имеется.

В уравнении $n \geq m$, иначе система или объект, которым оно описывается, не является физически реализуемым.

Однако часто при идеализированном описании некоторых элементов систем автоматического регулирования, например, ПИД регуляторов, это правило не соблюдается.

Откуда в дифференциальном уравнении появляются производные выходного сигнала, мы уже выяснили, а вот вопрос о том, как в нем возникают про-

изводные входного, требует особого пояснения. Тем более что мы знаем, что никакого дифференцирования в реальных объектах нет, – только интегрирование (накопление).

Рассмотрим уравнение второго порядка общего вида

$$\frac{d^2y}{dt^2} + a_1\frac{dy}{dt} + a_0 y = b_2\frac{d^2u}{dt^2} + b_1\frac{du}{dt} + b_0 u..$$

Выразим вторую производную (ускорение) выходного сигнала:

$$\frac{d^2y}{dt^2} = -a_1\frac{dy}{dt} - a_0 y + b_2\frac{d^2u}{dt^2} + b_1\frac{du}{dt} + b_0 u\ .$$

Проинтегрируем уравнение два раза:

$$\frac{dy}{dt} = -a_1 y - \int a_0 y + b_2\frac{du}{dt} + b_1 u + \int b_0 u =$$

$$= -a_1 y + b_2\frac{du}{dt} + b_1 u + \int\left(b_0 u - a_0 y\right),$$

$$y = b_2 u + \int\left[b_1 u - a_1 y + \int\left(b_0 u - a_0 y\right)\right].$$

Мы получили интегральное уравнение, связывающее выходную величину с входной. Попытаемся по данному уравнению построить схему модели на интеграторах.

Шаг 1. Итак, у нас есть входной сигнал *u* и выходной сигнал *y* (рис. 2.25).

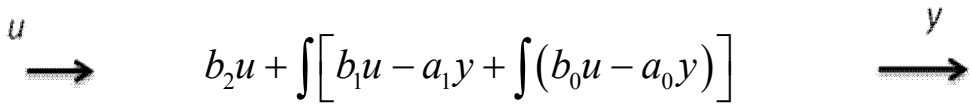


Рис. 2.25. Построение структурной схемы, шаг 1.

Шаг 2. Сигнал *y* – это сумма сигналов b_2u (а сигнал *u* уже у нас есть!) и интеграла, т.е. выходного сигнала интегратора. На входе интегратора – сигнал, сформированный в квадратных скобках (рис. 2.26).

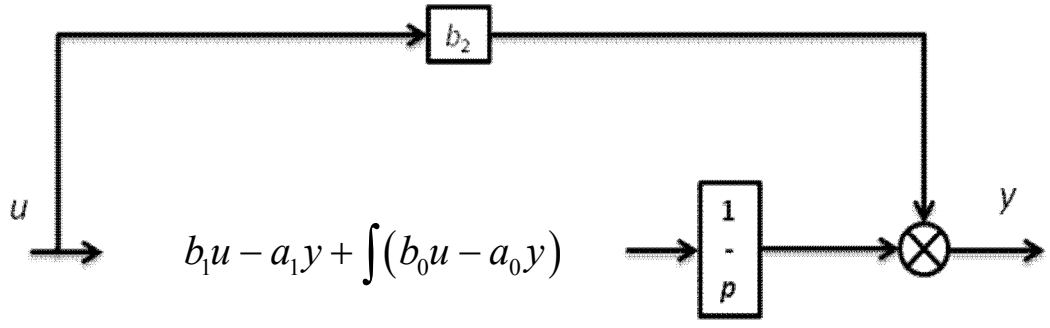


Рис. 2.26. Построение структурной схемы, шаг 2.

Шаг 3. Сигнал на входе интегратора формируется как сумма трех сигналов. Первые два мы без труда получим умножением *u* и *y* на b_1 и a_1 соответственно, а третий – это интеграл, т.е. сигнал на выходе интегратора (рис. 2.27).

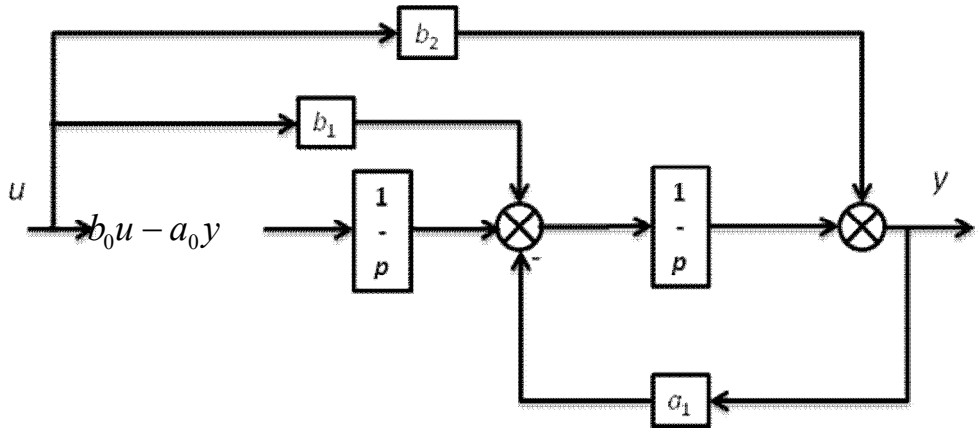


Рис. 2.27. Построение структурной схемы, шаг 3.

Шаг 4 и заключительный (рис. 2.28).

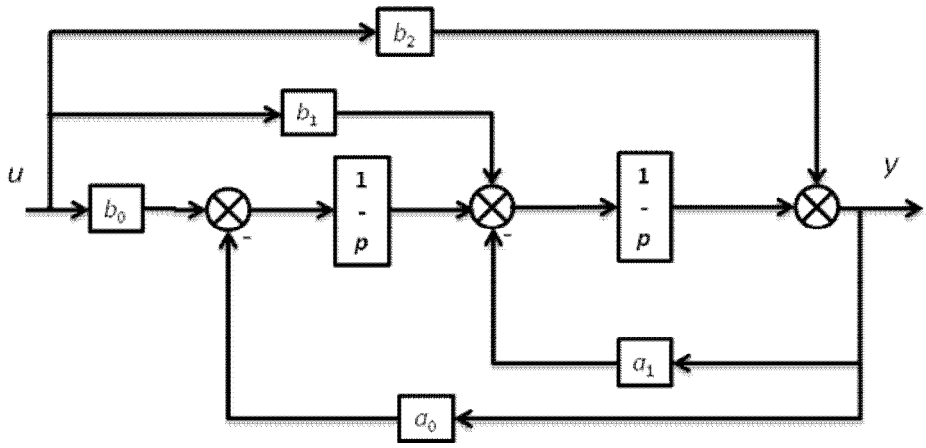


Рис. 2.28. Построение структурной схемы, шаг 4.

Вот так просто составляются структурные схемы. Главное – не бояться.

Теперь о том, зачем это все нужно. По структурной схеме видно, как входной сигнал влияет на выходной. Это влияние может быть непосредственным и «быстрым» (через коэффициент) или опосредованным и «медленным» (через интегратор). «Коэффициент» – точнее, блок, реализующий масштабирование, сигнал не задерживает: в любой момент времени сигнал на выходе блока равен произведению сигнала на его входе и масштабного коэффициента. Интегратор же демонстрирует куда более сложное поведение, суть которого мы уже выяснили.

Структурная схема позволяет найти начальное и конечное значения выходной величины в переходном процессе (рис. 2.29).

Пусть, например, до момента времени $t = t_0$ система находилась в состоянии покоя, ни один из сигналов (на стрелках схемы) не изменялся, а входное воздействие отсутствовало, $u(t_0 - 0) = 0$. Что такое «минус нуль»? Это буквально означает «непосредственно до». «До чего» будет ясно далее.

Сигналы на выходах интеграторов (обозначим их x_1 и x_2) тоже были равны нулю. Иначе «пасьянс не сойдется», так как любое ненулевое значение на выходе какого-либо интегратора благодаря обратной и прямой связям приведет к тому, что на его входе или на входе «соседа» появится ненулевой сигнал и «машина поедет», начнется процесс интегрирования, который никак не вяжется с состоянием покоя.

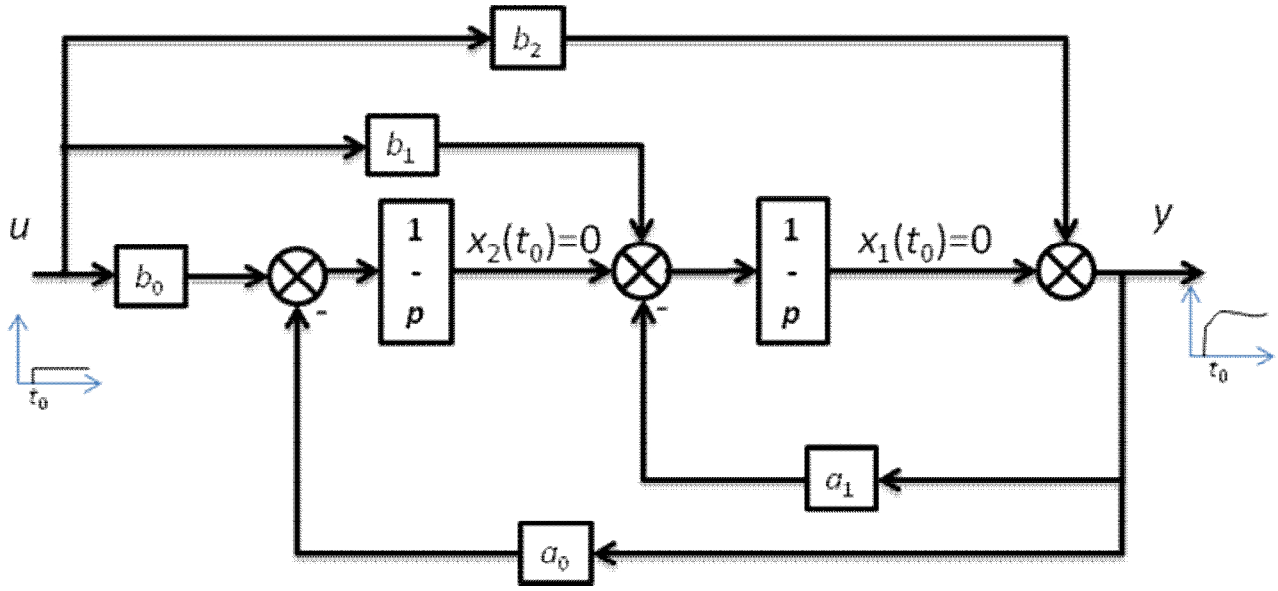


Рис. 2.29. Определение начальных и конечных значений.

Пусть, далее, в момент времени $t = t_0$ входной сигнал скачкообразно изменился и стал равен единице: $u(t_0 + 0) = 1$ («плюс нуль» означает «непосредственно после»). Сигналы x_1 и x_2 измениться мгновенно не могут (как не может мгновенно измениться уровень воды в баке), поэтому на схеме и нет никаких «плюс/минус нуль», а вот выходной сигнал y – вполне. Структурная схема дает нам возможность не только определить начальное значение y, но и начальную скорость его изменения.

Значение выходного сигнала вначале процесса определяется легко:

$$y(t_0+0) = x_1(t_0) + b_2u(t_0+0) = b_2.$$

Поскольку сигнал u в течение всего процесса неизменен, скорость изменения выходного сигнала равна скорости изменения сигнала x_1, т.е. значению сигнала на входе второго интегратора. В момент времени $t = t_0 + 0$ $y = b_2$, $x_2 = 0$, $u = 1$, следовательно,

$$\frac{dy}{dt}(t_0 + 0) = -a_1y(t_0 + 0) + x_2(t_0) + b_1u(t_0 + 0) = -a_1b_2 + b_1.$$

Далее. Наконец, переходный процесс закончился. Что это означает? Это означает, что система вновь пришла в состояние покоя и интеграторы больше не «интегрируют», на их входах – нули. Посмотрим на вход первого интегратора. Если на его входе – нуль, значит $b_0u = a_0y$, откуда

$$y = \frac{b_0}{a_0}u = \frac{b_0}{a_0}.$$

То же самое можно получить и из дифференциального уравнения. По окончании переходного процесса все производные становятся равными нулю, потому что никакие сигналы не изменяются, и дифференциальное уравнение перестает быть дифференциальным и становится *алгебраическим*:

$$a_0y = b_0u\ .$$

Структурная схема системы также дает представление о том, как выходная величина влияет сама на себя, а точнее на свои скорость и ускорение (на свои производные, в общем случае). Чтобы разобраться с этим вопросом, упростим нашу систему, избавившись от производных входного сигнала в дифференциальном уравнении, т.е. положив $b_1 = b_2 = 0$.

Для начала рассмотрим самый простой случай: $a_0 = a_1 = 0$:

$$\frac{d^2y}{dt^2} = b_0 u\,,\ y = \iint b_0 u.$$

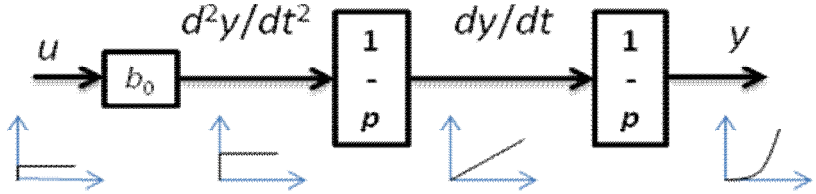


Рис. 2.30. Переходный процесс в системе с двумя интеграторами.

Схема демонстрирует второй закон Ньютона «во всей его красе»: ускорение выходной величины прямо пропорционально входному сигналу (силе). Коэффициент b_0 – величина, обратная массе.

При скачкообразном изменении управляющего сигнала от нуля до некоторого значения мгновенно изменится ускорение, скорость (первая производная) начнет линейно расти, а положение y – расти по параболе. Другими словами сначала и сразу изменится ускорение, потом начнет изменяться скорость, и только потом – само перемещение. Понятно, что слово «потом» описывает здесь причинно-следственную связь между понятиями, на самом деле все будет происходить одновременно, но ускорение изменится действительно скачком, скорость будет изменяться медленно, а перемещение – еще медленнее.

При $a_1 \neq 0$ процесс будет несколько иным (рис. 2.31).

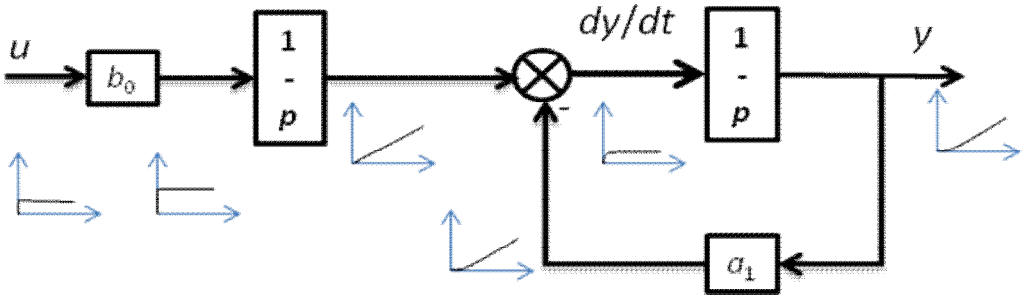


Рис. 2.31. Переходный процесс при наличии обратной связи.

Выходная величина y непосредственно влияет на скорость своего изменения. Обратная связь *стабилизирует* скорость роста выходной величины, причем, чем больше коэффициент a_1, тем меньше сигнал на входе второго интегратора и тем меньше скорость изменения входной величины.

Если $a_1 \neq 0$ и $a_2 \neq 0$ стабилизируется не только скорость, но и сама выходная величина. Характер процесса в системе определяется соотношением a_1 и a_2. На рис. 2.32 показан один из вариантов «развития событий».

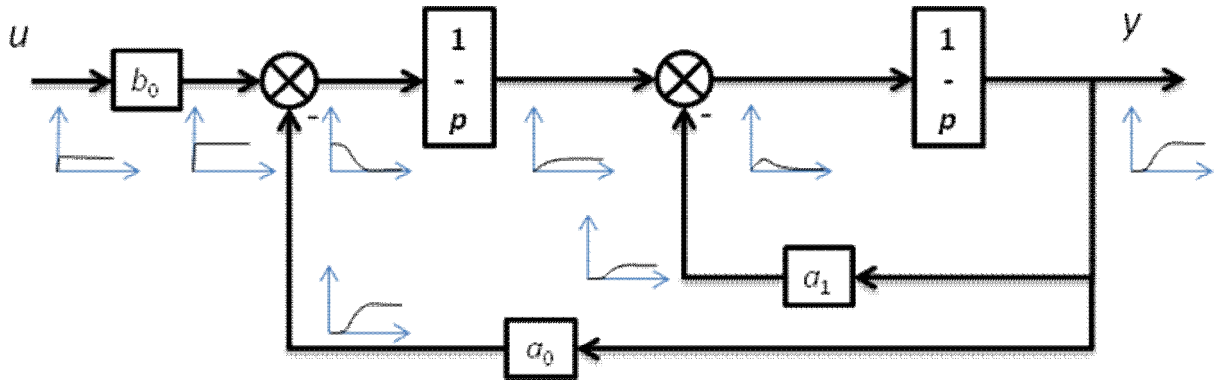


Рис. 2.32. Переходный процесс при наличии двух обратных связей (апериодический).

Работу системы можно рассматривать с такой точки зрения. Есть два контура: внутренний и внешний. Во внутреннем контуре формируется сама выходная величина y, а внешний контур управляет внутренним, выдавая ему «задание». Если внутренний контур работает быстро, по крайней мере, быстрее внешнего, все идет «как по маслу»: задание выполняется хорошо. А контур работает тем быстрее, чем больше коэффициент обратной связи.

В рассмотренном варианте процессы носят *апериодический* (не колебательный) характер. Если уменьшить коэффициент a_1, не изменяя при этом коэффициент a_0 (или, наоборот, увеличить a_0 при неизменном a_1), процессы станут колебательными. Внутренний контур перестанет успевать выполнять задание «в срок», его реакция будет «запаздывать» (рис. 2.33).

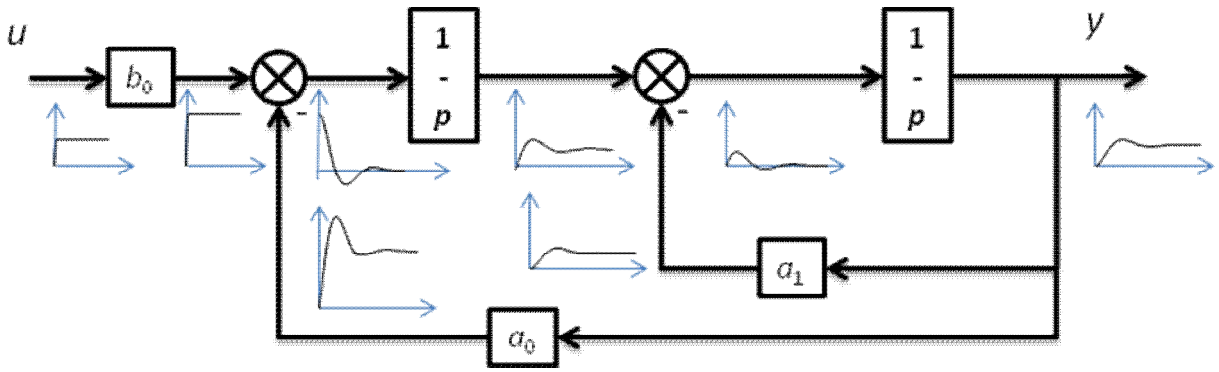


Рис. 2.33. Переходный процесс при наличии двух обратных связей (колебательный).

Здесь уместна такая аналогия. Вы управляете автомобилем с «плохой чувствительностью» к педалям газа и тормоза. Вам нужно проехать сто метров и остановиться точно у стоп-линии. С трудом Вам удалось разогнать автомобиль. Подъезжая к стоп-линии, Вы начали тормозить, но машина плохо слушается. В результате Вы заехали за стоп-линию, включили заднюю скорость и … процесс повторился. После нескольких челночных движений с затухающим размахом Вам все-таки удалось остановиться в нужном месте.

Еще лучше аналогия с управлением автомобилем будет прослеживаться, если преобразовать схему, как говорят математики, *эквивалентным образом* (т.е. так, чтобы ее *внешнее поведение* не изменилось). Сначала так (рис. 2.34).

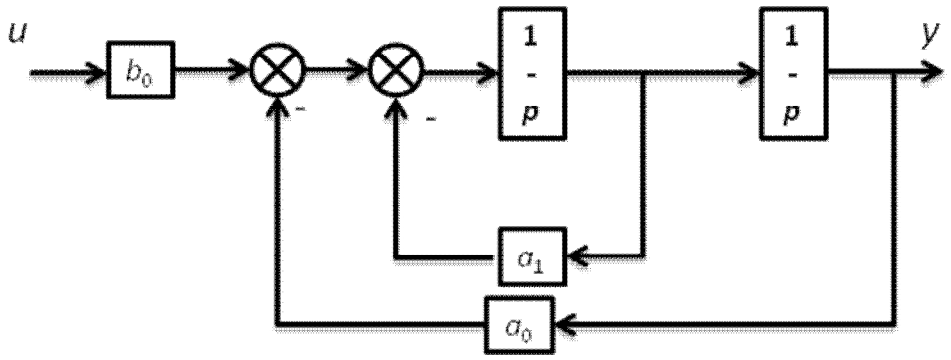


Рис. 2.34. Преобразование структурной схемы, шаг 1.

А потом вот так (рис. 2.35).

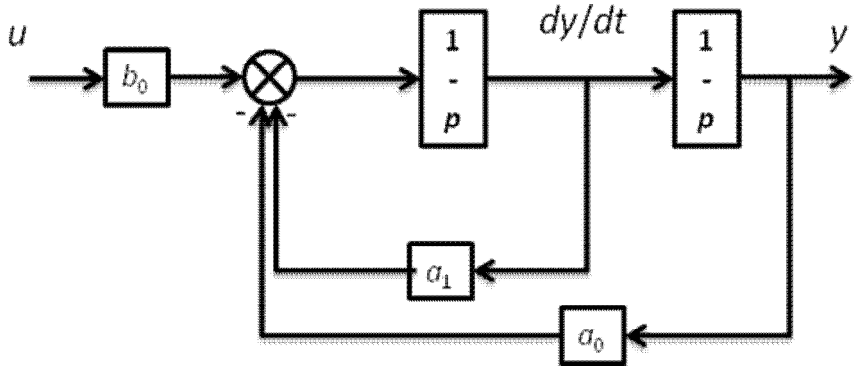


Рис. 2.35. Преобразование структурной схемы, шаг 2.

Мы всего лишь поменяли местами интегратор и контур. Поскольку между ними ничего не было, такая замена никак не скажется на внешнем поведении. Действительно, положив в уравнении

$$\frac{d^2y}{dt^2} = -a_1\frac{dy}{dt} - a_0y + b_2\frac{d^2u}{dt^2} + b_1\frac{du}{dt} + b_0u\,.$$

$b_1 = b_2 = 0$, получим

$$\frac{d^2y}{dt^2} = -a_1\frac{dy}{dt} - a_0y + b_0u\,.$$

Проинтегрировав два раза, имеем

$$y = \int\int\left[-a_1\frac{dy}{dt} - a_0y + b_0u\right].$$

(здесь мы просто не стали «избавляться» от производной). Сравнив уравнение со схемой, видим, что они полностью согласуются: выражение под двойным интегралом описывает сигнал на входе первого интегратора схемы.

Вообще же тот факт, что одному дифференциальному уравнению соответствует несколько схем, не должен нас смущать. На самом деле этих схем бесконечное множество! Дело в том, что одно и то же внешнее поведение может быть достигнуто с помощью самых разных внутренних структур.

Однако вернемся к нашему «автомобильному» примеру. Если y – это путь, пройденный автомобилем, то сигнал на входе второго интегратора (выходе первого) – это скорость автомобиля, а сигнал на входе первого – его ускорение. Согласно нашей схеме, гипотетическая система управления формирует ускорение тем большее, чем больше входной сигнал u (некий аналог заданного пути) и чем меньше сам путь y и скорость автомобиля dy/dt. Тут все зависит от соотношения a_1 и a_0.

При относительно большом значении a_1 система будет больше ориентироваться на сигнал по скорости и «сбрасывать газ» при достижении даже небольшой скорости. В результате мы имеем хорошие шансы плавно «подкатиться» к стоплинии и остановиться прямо перед ней. С другой стороны, такая «аккуратность» наверняка приведет к затягиванию процесса: ехать-то мы будем медленно!

При относительно большом значении a_0 система будет больше ориентироваться на сигнал по положению, следовательно, автомобиль разовьет большую скорость и скорее всего не успеет затормозить. Мы окажемся за стоплинией, придется включать заднюю передачу и т.д.

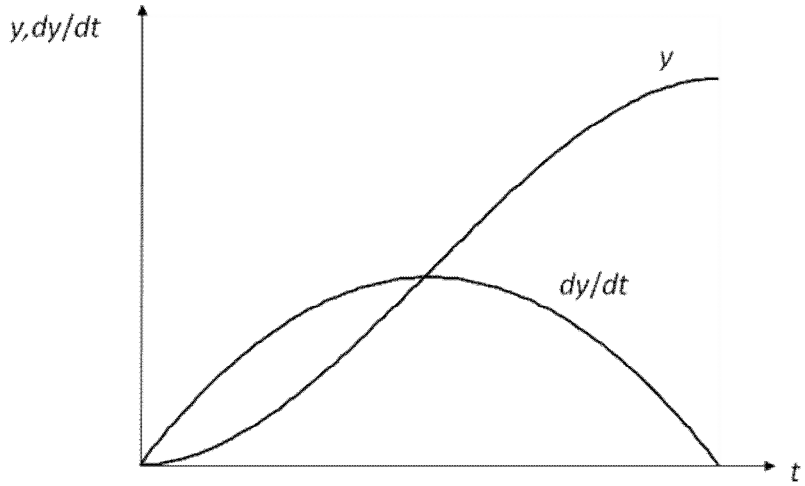


Рис. 2.36. Переходный процесс.

Очевидно, что существует некоторое оптимальное соотношение между a_1 и a_0, при котором автомобиль успеет сбросить скорость до нуля непосредственно перед стоп-линией (рис. 2.36).

Однако нахождение этого соотношения – пока не наша задача.

В предельном случае, когда $a_1 = 0$, стабилизации скорости изменения выходной величины не происходит, и колебания вообще не затухают (рис. 2.37).

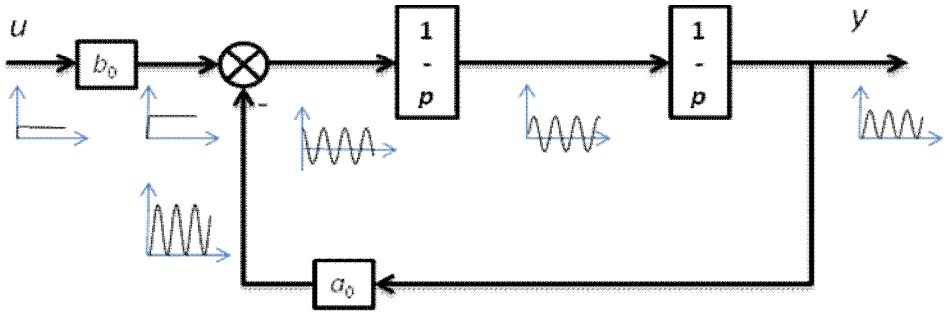


Рис. 2.37. Переходный процесс при отсутствии внутренней ОС.

При скачкообразном изменении сигнала u первый интегратор (*накопитель*, если помните) начинает увеличивать выходной сигнал. Прекратит это делать только тогда, когда разность сигналов на выходе сумматора станет равным нулю. В этом бы «месте» процессу и «остановиться», да не тут-то было. Пер-

вый интегратор уже накопил большое значение на выходе, «компенсировать» этот сигнал некому и второму интегратору придется его «интегрировать», т.е. изменять свой выходной сигнал. А это приведет к продолжению процесса и изменению знака разности.

Продолжая аналогию с процессом управления автомобилем, можно представить себе ситуацию, что Вы начинаете сбрасывать газ только при наезде на стоп-линию. Естественно, при таком управлении Вы и будете бесконечно долго ездить туда-сюда.

Здесь, конечно, аналогия неполная. Вы же не будете сбрасывать газ, а нажмете на тормоз. А так как система торможения автомобиля «сильнее» «системы разгона», автомобиль все-таки, в конце концов, остановится.

Таким образом, уже в системе второго порядка есть предпосылки для возникновения колебательных процессов. «Внутренняя» обратная связь *по скорости* выполняет функцию *демпфирования*, т.е. подавления колебаний.

В целом же можно сделать следующие выводы:

1. Дифференциальное уравнение описывает динамические свойства объекта, т.е. процессы накопления и расходования вещества или энергии.

2. Чем больше порядок дифференциального уравнения (порядок старшей производной выхода), тем сложнее поведение объекта. Так, при первом порядке инерционно изменяется только сама величина, а скорость изменяется мгновенно, при втором – инерционны сама величина и ее скорость, а ускорение изменяется мгновенно и т.д.

3. Наличие в дифференциальном уравнении младших производных и самой выходной величины говорит о наличии внутренних обратных связей, создающих возможность самовыравнивания.

4. Наличие в дифференциальном уравнении производных входной величины говорит о способности непосредственного воздействия (в обход интеграторов) входной величины на выход, его скорость и т.д. Если порядок старшей производной входа равен порядку старшей производной выхода, входной сигнал мгновенно проникает на выход, если на единицу меньше – мгновенно изменяет его скорость и т.д.

Осталось ответить на вопрос, вынесенный в заголовок параграфа: что такое *передаточная функция*? Тут, кажется, читатель будет разочарован, поскольку ничего нового он, по сути, не узнает. Дело в том, что говоря попростому, передаточная функция и есть дифференциальное уравнение, только представленное в виде *оператора*, т.е. некоторой «штуки», на которую нужно «умножить» входной сигнал, чтобы получить выходной. Пусть, например, у нас есть дифференциальное уравнение общего вида

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \ldots + a_1 \frac{dy}{dt} + a_0 y = b_m \frac{d^m u}{dt^m} + b_{m-1} \frac{d^{m-1} u}{dt^{m-1}} + \ldots + b_1 \frac{du}{dt} + b_0 u.$$

Как мы помним, есть оператор p, означающий взятие производной, т.е $dy/dt = px$. Тогда p^2 – оператор второй производной, p^3 – третьей и т.д.

Следовательно, дифференциальное уравнение можно записать в операторной форме:

$$a_n p^n y + a_{n-1} p^{n-1} y + \ldots + a_1 p y + a_0 y = b_m p^m u + b_{m-1} p^{m-1} u + \ldots + b_1 p u + b_0 u,$$

или

$$y(a_n p^n + a_{n-1} p^{n-1} + \ldots + a_1 p + a_0) = u(b_m p^m + b_{m-1} p^{m-1} + \ldots + b_1 p + b_0).$$

Выразив отсюда формально отношение выхода к входу, получим передаточную функцию:

$$\frac{y}{u} = \frac{b_m p^m + b_{m-1} p^{m-1} + \ldots + b_1 p + b_0}{a_n p^n + a_{n-1} p^{n-1} + \ldots + a_1 p + a_0} = W(p).$$

Отметим, что на самом деле $W(p)$ – не отношение *сигналов* $y(t)/u(t)$. Это оператор, обозначающий, что функции $y(t)$ и $u(t)$ связаны друг с другом соответствующим дифференциальным уравнением. Поэтому формальное выражение

$$y = W(p)u$$

говорит нам о том, что для нахождения $y(t)$ требуется решить дифференциальное уравнение, представленное передаточной функцией $W(p)$, подставив в него известную функцию $u(t)$.

Существует и другой смысл передаточной функции, связанный с разделом математики «Операционное исчисление». В этом разделе рассматриваются т.н. Лапласовы изображения сигналов $y(p)$ и $u(p)$, где p – оператор Лапласа (а не дифференцирования, как у нас). В таком случае передаточная функция $W(p)$ есть отношение этих изображений при нулевых начальных условиях. Но мы в такие «дебри» лезть не будем.

2.2.2.4. Представление в пространстве состояний

В «продвинутых» разделах теории автоматического управления используют описание динамических объектов и систем в пространстве состояний. И хотя мы не планируем добираться до этих разделов, иметь представление о таком описании нужно. Причин тому, по крайней мере, две. Во-первых, может получиться так, что исходное описание объекта изначально задано в таком виде и, следовательно, нам придется каким-то образом преобразовать его в передаточные функции. Во-вторых, как мы увидим позже, программирование нетиповых регуляторов производится именно в пространстве состояний. Линейные модели в пространстве состояний иллюстрируются уже знакомыми нам схемами на интеграторах.

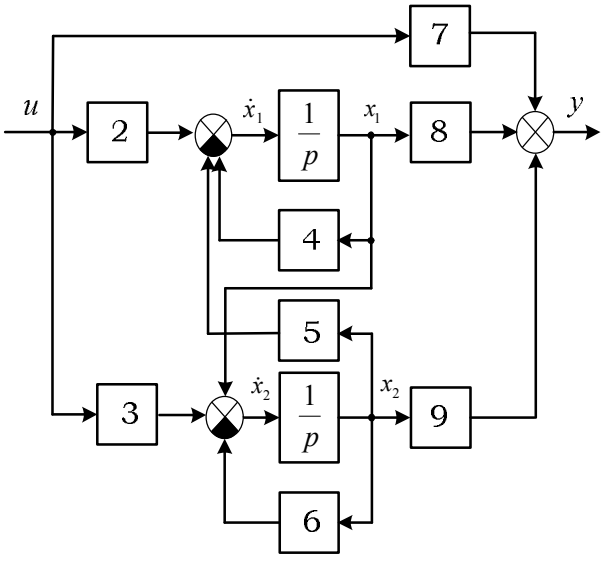


Рис. 2.38. Схема модели в пространстве состояний.

Пусть, например, имеется следующая схема (рис. 2.38).

Переменные состояния – это те переменные, которые не могут мгно-

венно изменяться при изменении входного воздействия. В принципе, сигналов – кандидатов на роль этих переменных – в схеме много. На схеме сигналы – это «стрелки», и большая часть из них (но не все!) изменяются «плавно».

Однако мы никогда не ошибемся, если в качестве переменных состояния выберем выходные сигналы интеграторов. Действительно, выход интегратора мгновенно изменяться не может. Таким образом, сколько в схеме интеграторов (какой порядок системы), столько и переменных состояния.

А дальше все просто. Сигналы на входах интеграторов – это производные переменных состояния. Зная это, по схеме легко составить дифференциальные уравнения первого порядка для всех переменных состояния. Далее остается только записать алгебраическое выражение для выходной величины.

Для рассматриваемой схемы получим следующие уравнения:

$$\begin{cases} \dot{x}_1(t) = -4x_2(t) - 5x_2(t) + 2u; \\ \dot{x}_2(t) = x_1(t) - 6x_2(t) + 3u. \end{cases}$$

$$y(t) = 8x_1(t) + 9x_2(t) + 7u.$$

Следует отметить, что такое описание годится и для систем с несколькими входами и выходами. Тогда в описании появятся переменные u_1, u_2 …, и, кроме того, у нас будет несколько алгебраических уравнений выходов y_1, y_2 ….

Уравнения состояний обычно представляют в матричном виде:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{pmatrix} -4 & -5 \\ 1 & -6 \end{pmatrix} \times \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{pmatrix} 2 \\ 3 \end{pmatrix} \times u,$$

$$\dot{X} \quad = \quad A \quad \times \quad X \quad + \; B \; \times u.$$

Аналогично поступают и с уравнением (или уравнениями) выхода:

$$y = \begin{pmatrix} 8 & 9 \end{pmatrix} \times \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + 7u,$$

$$y = \quad C \quad \times X \quad +Du.$$

Таким образом, мы получили *матрицы описания в пространстве состояний*:

$$A = \begin{pmatrix} -4 & -5 \\ 1 & -6 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \quad C = \begin{pmatrix} 8 & 9 \end{pmatrix}, \quad D = 7,$$

которые называются матрицами *состояния*, *входа*, *выхода* и *прямого обхода* соответственно.

Имея описание в пространстве состояний, легко найти передаточную функцию (в случае системы с одним входом и одним выходом) или *передаточную матрицу* (матрицу передаточных функций – для многомерных систем). Запишем уравнение состояний в операторной форме:

$$pX = AX + Bu,$$

откуда

$$(pE - A)X = Bu,$$

где E – единичная матрица (квадратная) тех же размеров, что и матрица A. Появилась она по причине того, что по правилам математики из скалярного оператора p нельзя вычесть матрицу A. Матрица E помогает решить эту проблему. Продемонстрируем это на нашем примере.

$$pX - AX = \begin{bmatrix} px_1 \\ px_2 \end{bmatrix} - \begin{bmatrix} -4x_1 - 5x_2 \\ x_1 - 6x_2 \end{bmatrix} = \begin{bmatrix} px_1 + 4x_1 + 5x_2 \\ px_2 - x_1 + 6x_2 \end{bmatrix},$$

$$(pE - A)X = \left(p\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} -4 & -5 \\ 1 & -6 \end{pmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \left(\begin{pmatrix} p & 0 \\ 0 & p \end{pmatrix} - \begin{pmatrix} -4 & -5 \\ 1 & -6 \end{pmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} =$$

$$= \begin{pmatrix} p+4 & 5 \\ -1 & p+6 \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} px_1 + 4x_1 + 5x_2 \\ -x_1 + px_2 + 6x_2 \end{bmatrix}.$$

Как мы видим, матрица E ничего не испортила. Далее выразим вектор X:

$$X = (pE - A)^{-1} Bu$$

и подставим полученное выражение в уравнение выхода:

$$y = CX + Du = C(pE - A)^{-1} Bu + Du = \left(C(pE - A)^{-1} B + D \right) u.$$

Передаточная функция (или матрица) должна связывать вход (входы) и выход (выходы):

$$y = W(p)u,$$

следовательно,

$$W(p) = C(pE - A)^{-1} B + D = \frac{C(pE - A)^{+} B}{\det(pE - A)} + D.$$

Здесь знак «+» означает *союзную* или *присоединенную* матрицу, а det (determinant) – оператор нахождения определителя.

К счастью, в настоящее время нам не приходится заниматься матричными вычислениями «врукопашную». В программе Matlab вводом (конструированием) и преобразованием (конверсией) моделей занимаются функции `ss` (state space, пространство состояний) и `tf` (transfer functions, передаточные функции) пакета Control.

Вернемся к нашему примеру, сформируем матрицы, создадим объект `ss` и получим передаточную функцию системы с помощью конвертора `tf`:

```
A = [-4 -5;1 -6]
A =

    -4    -5
     1    -6
```

```
B=[2;3]
B =

     2
     3
C = [8 9]

C =

     8     9
D = 7
D =

     7
sys = ss(A,B,C,D)
sys =

  A =
       x1  x2
   x1  -4  -5
   x2   1  -6

  B =
       u1
   x1   2
   x2   3

  C =
       x1  x2
   y1   8   9

  D =
       u1
   y1   7
Continuous-time state-space model.
sys = tf(sys)
sys =

  7 s^2 + 113 s + 305
  -------------------
    s^2 + 10 s + 29
Continuous-time transfer function.
```

Теперь рассмотрим обратную задачу. Предположим, у нас есть передаточная функция некоторой системы или *подсистемы*, например, регулятора. Нам нужно получить уравнения в пространстве состояний, потому что, например, их легко программно реализовать.

Общий вид любой физически реализуемой передаточной функции следующий:

$$W(p) = d_0 + \frac{c_{n-1}p^{n-1} + c_{n-2}p^{n-2} + ... + c_1 p + c_0}{p^n + a_{n-1}p^{n-1} + ... + a_1 p + a_0}.$$

У нас есть безынерционная часть (коэффициент d_0) и инерционная часть (дробь), у которой порядок числителя на единицу меньше порядка знаменателя. Соответственно формируется и выходной сигнал:

$$y = d_0 u + \frac{c_{n-1}p^{n-1} + c_{n-2}p^{n-2} + \ldots + c_1 p + c_0}{p^n + a_{n-1}p^{n-1} + \ldots + a_1 p + a_0} u = d_0 u + y_1,$$

где y_1 – это сигнал на выходе инерционной части, описываемой дифференциальным уравнением:

$$\frac{d^n y_1}{dt^n} + a_{n-1}\frac{d^{n-1}y_1}{dt^{n-1}} + \ldots + a_1\frac{dy_1}{dt} + a_0 y_1 = c_{n-1}\frac{d^{n-1}u}{dt^{n-1}} + c_{n-2}\frac{d^{n-2}u}{dt^{n-2}} + \ldots + c_1\frac{du}{dt} + c_0 u.$$

Оказывается, имеется возможность (и даже не одна!) без всяких вычислений составить схему модели в пространстве состояний и записать соответствующие уравнения. Существует несколько так называемых *канонических форм* моделей в пространстве состояний, одна из которых, *каноническая форма управляемости* далее и рассматривается.

На рис. 2.39 показана схема модели для системы с нашей передаточной функцией. Интеграторы на схеме расположены последовательно, выход одного является входом другого. Коэффициенты знаменателя передаточной функции стали коэффициентами отрицательных обратных связей, заведенных с выходов интеграторов на вход самого первого из них. Выходной сигнал инерционной части y_1 формируется как взвешенная сумма выходных сигналов интеграторов, причем коэффициентами этой суммы стали коэффициенты числителя передаточной функции.

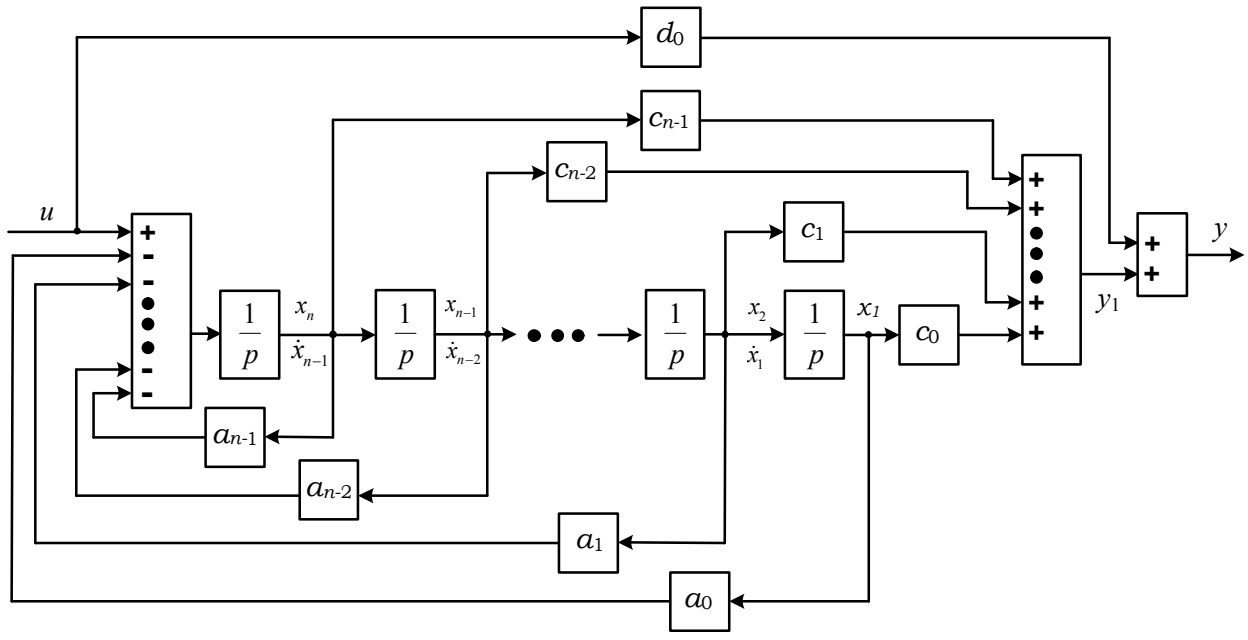


Рис. 2.39. Схема модели в канонической форме управляемости.

Обозначив сигналы на выходах интеграторов, как показано на схеме, получим уравнения в пространстве состояний:

$$\begin{cases} \dot{x}_1 = x_2; \\ \dot{x}_2 = x_3; \\ \ldots \\ \dot{x}_{n-1} = x_n; \\ \dot{x}_n = -a_0 x_1 - a_1 x_2 - \ldots - a_{n-2} x_{n-1} - a_{n-1} x_n + u. \end{cases}$$

$$y = c_0 x_1 + c_1 x_2 + \ldots + c_{n-2} x_{n-1} + c_{n-1} x_n + d_0 u.$$

Матрицы описания:

$$A = \begin{pmatrix} 0 & 1 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 0 & 1 \\ -a_0 & -a_1 & \ldots & -a_{n-2} & -a_{n-1} \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 0 \\ \ldots \\ 0 \\ 1 \end{pmatrix},$$

$$C = \begin{pmatrix} c_0 & c_1 & \ldots & c_{n-2} & c_{n-1} \end{pmatrix}, \quad D = d_0 .$$

Очевидно, что если в исходной передаточной функции отсутствуют какие бы то ни было элементы (степени оператора p), соответствующие коэффициенты в описании будут равны нулю.

Таким образом, по передаточной функции сразу можно записать матрицы и уравнения в пространстве состояний и составить схему модели на интеграторах.

Может вызвать затруднение вопрос о том, как получить коэффициент d_0, если в исходном виде мы имеем передаточную функцию, у которой порядок числителя равен порядку знаменателя. Очень просто: нужно поделить полином числителя на полином знаменателя. Целая часть такого деления есть отношение коэффициентов при старшей степени оператора p полиномов. Коэффициенты числителя дробной части легко находятся. Продемонстрируем это на нашем примере:

$$W(p) = \frac{7p^2 + 113p + 305}{p^2 + 10p + 29} = 7 + \frac{c_1 p + c_0}{p^2 + 10p + 29} =$$

$$= \frac{7p^2 + 70p + 203}{p^2 + 10p + 29} + \frac{c_1 p + c_0}{p^2 + 10p + 29},$$

откуда $c_1 = 43$, $c_0 = 102$.

2.2.2.5. Запаздывание

Пора вспомнить, что помимо инерционности в некоторых (далеко не всех!) объектах и системах действует еще один фактор, определяющий их динамику, – временно́е запаздывание.

Запаздывание (*чистое*, или *транспортное запаздывание*) не связано с накоплением среды или энергии, а представляет собой просто временную задержку при передаче сигнала или среды. Например, задержка при передаче радиосигнала связи между Землей и Марсом составляет от 3 до 22 минут в зависимости от взаимного расположения планет (рис. 2.40).

Рис. 2.40. Запаздывание. Управление марсоходом.

В промышленности и сельском хозяйстве и запаздывание типичными примерами запаздывания являются разного рода транспортеры, нории и т.д. рис. 2.41.

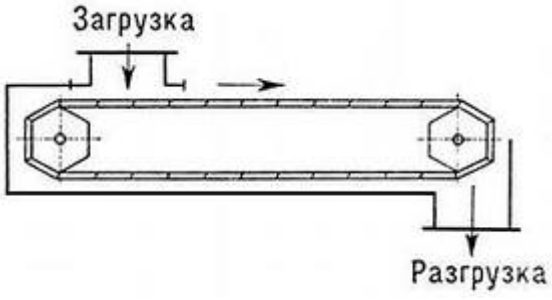


Рис. 2.41. Запаздывание. Транспортер.

Время запаздывания в таких механизмах легко определяется через расстояние от места загрузки до места разгрузки (или измерения веса) и скорость движения.

Запаздывание может быть описано уравнением

$$y(t) = u(t-\tau),$$

где $y(t)$ – выходной сигнал в момент времени t, $u(t-\tau)$ – входной сигнал в момент времени $t-\tau$, τ - временная задержка.

В запаздывании нет ни дифференцирования, ни интегрирования, поэтому передаточная функция запаздывания не может быть получена без применения специального математического аппарата и дается здесь без вывода:

$$W(p) = \frac{y}{u} = e^{-\tau p}.$$

Таким образом, $e^{-\tau p}$ – оператор сдвига по времени (назад) на время τ.

Если запаздывание имеет место, то данный оператор присутствует в передаточной функции системы. При описании в пространстве состояний запаздывание проявляется в том, что некоторые переменные оказываются *сдвинутыми во времени*, например, в уравнениях состояний переменная x_1 фигурирует как $x_1(t-\tau)$.

В отличие от инерционности, которая, как было сказана выше, играет противоречивую роль, одновременно и помогая, и мешая управлению, запаздывание однозначно считается вредным явлением. Попробуйте, поуправляйте марсоходом, если он начинает выполнять Ваши команды спустя 22 с после их подачи, и столько же времени требуется для получения «отклика»! Наверняка довольно быстро марсоход окажется лежащим «брюхом кверху» на дне какого-нибудь кратера.

2.2.2.6. Структурные преобразования

Почему в теории управления используются передаточные функции, а не дифференциальные уравнения в чистом виде? Во-первых, по передаточным функциям легко определяются частотные характеристики объектов и систем, широко используемые в задачах анализа и синтеза. Во-вторых, с передаточными функциями проще работать. В частности, зная передаточные функции элементов системы, легко определить передаточную функцию системы в целом. Рассмотрим основные *правила структурных преобразований* линейных систем.

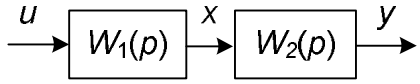


Рис. 2.42. Последовательное соединение.

Последовательное соединение звеньев представлено на рис. 2.42.

Поступая формально, получим

$$y = W_2(p)x = W_2(p)W_1(p)u.$$

Следовательно, передаточная функция последовательно соединенных звеньев равна произведению их передаточных функций.

Но кто дал нам право умножать сигнал на передаточную функцию? Ведь передаточная функция – это «всего лишь» операторная форма представления дифференциального уравнения, связывающего вход и выход. И выражение $x = W_1(p)u$ обозначает, таким образом, только то, что для нахождения $x(t)$ нужно решить дифференциальное уравнение, представленное передаточной функцией $W_1(p)$, подставив в него $u(t)$. К счастью, формальное правило работает. Покажем это на примере.

Пусть

$$W_1(p) = \frac{x}{u} = \frac{b_{11}p + b_{10}}{p^2 + a_{11}p + a_{10}},$$

$$W_2(p) = \frac{y}{x} = \frac{b_{21}p + b_{20}}{p^2 + a_{21}p + a_{20}}.$$

Применяя правило, получим:

$$W_1(p)W_2(p)=\frac{y}{u}=\frac{\left(b_{11}p+b_{10}\right)\left(b_{21}p+b_{20}\right)}{\left(p^2+a_{11}p+a_{10}\right)\left(p^2+a_{21}p+a_{20}\right)}=$$

$$=\frac{b_{11}b_{21}p^2+\left(b_{11}b_{20}+b_{10}b_{21}\right)p+b_{10}b_{20}}{p^4+\left(a_{21}+a_{11}\right)p^3+\left(a_{20}+a_{11}a_{21}+a_{10}\right)p^2+\left(a_{11}a_{20}+a_{10}a_{21}\right)p+a_{10}a_{20}}.$$

А теперь давайте «без правил». Дифференциальные уравнения, соответствующие $W_1(p)$ и $W_2(p)$:

$$\frac{d^2x}{dt^2}+a_{11}\frac{dx}{dt}+a_{10}x=b_{11}\frac{du}{dt}+b_{10}u,$$

$$\frac{d^2y}{dt^2}+a_{21}\frac{dy}{dt}+a_{20}y=b_{21}\frac{dx}{dt}+b_{20}x.$$

Для того, чтобы найти «общее» дифференциальное уравнение, нам, очевидно, следует избавиться от ненужной нам промежуточной переменной x. Но это не так-то просто сделать, поскольку в правой части второго дифференциального уравнения присутствуют две функции: $x(t)$ и $dx(t)/dt$. Будем считать их двумя независимыми функциями времени и найдем дифференциальные уравнения при условии, что эти функции действуют *по отдельности*. Начнем с первой функции. Для этого рассмотрим «усеченное» уравнение второго звена

$$\frac{d^2y}{dt^2}+a_{21}\frac{dy}{dt}+a_{20}y=b_{20}x$$

и выразим из него x:

$$x=\frac{1}{b_{20}}\left[\frac{d^2y}{dt^2}+a_{21}\frac{dy}{dt}+a_{20}y\right].$$

Нам потребуются также первая и вторая производные этого сигнала:

$$\frac{dx}{dt}=\frac{1}{b_{20}}\left[\frac{d^3y}{dt^3}+a_{21}\frac{d^2y}{dt^2}+a_{20}\frac{dy}{dt}\right],\quad \frac{d^2x}{dt^2}=\frac{1}{b_{20}}\left[\frac{d^4y}{dt^4}+a_{21}\frac{d^3y}{dt^3}+a_{20}\frac{d^2y}{dt^2}\right].$$

Теперь подставим полученные выражения в дифференциальное уравнение первого звена:

$$\frac{1}{b_{20}}\left[\left(\frac{d^4y}{dt^4}+a_{21}\frac{d^3y}{dt^3}+a_{20}\frac{d^2y}{dt^2}\right)+a_{11}\left(\frac{d^3y}{dt^3}+a_{21}\frac{d^2y}{dt^2}+a_{20}\frac{dy}{dt}\right)+\right.$$

$$\left.+a_{10}\left(\frac{d^2y}{dt^2}+a_{21}\frac{dy}{dt}+a_{20}y\right)\right]=b_{11}\frac{du}{dt}+b_{10}u,$$

откуда

$$\frac{d^4y}{dt^4}+\left(a_{21}+a_{11}\right)\frac{d^3y}{dt^3}+\left(a_{20}+a_{11}a_{21}+a_{10}\right)\frac{d^2y}{dt^2}+\left(a_{11}a_{20}+a_{10}a_{21}\right)\frac{dy}{dt}+$$

$$+a_{10}a_{20}y=b_{11}b_{20}\frac{du}{dt}+b_{10}b_{20}u.$$

То же самое проделаем со второй функцией $dx(t)/dt$. Для этого рассмотрим уравнение

$$\frac{d^2y}{dt^2} + a_{21}\frac{dy}{dt} + a_{20}y = b_{21}\frac{dx}{dt}.$$

У нас небольшая проблема: из этого выражения нельзя однозначно выразить x, так как нам пришлось бы для этого интегрировать уравнение, а интегрирование дает множество решений в зависимости от начальных условий (да и не нужны нам сейчас никакие интегралы). Придется избавиться от x и в уравнении для первого звена, просто продифференцировав его:

$$\frac{d^3x}{dt^3} + a_{11}\frac{d^2x}{dt^2} + a_{10}\frac{dx}{dt} = b_{11}\frac{d^2u}{dt^2} + b_{10}\frac{du}{dt}.$$

Теперь найдем нужные нам производные $x(t)$:

$$\frac{dx}{dt} = \frac{1}{b_{21}}\left[\frac{d^2y}{dt^2} + a_{21}\frac{dy}{dt} + a_{20}y\right], \quad \frac{d^2x}{dt^2} = \frac{1}{b_{21}}\left[\frac{d^3y}{dt^3} + a_{21}\frac{d^2y}{dt^2} + a_{20}\frac{dy}{dt}\right],$$

$$\frac{d^3x}{dt^3} = \frac{1}{b_{21}}\left[\frac{d^4y}{dt^4} + a_{21}\frac{d^3y}{dt^3} + a_{20}\frac{d^2y}{dt^2}\right]$$

и подставим полученные выражения в продифференцированное уравнение для первого звена:

$$\frac{1}{b_{21}}\left[\left(\frac{d^4y}{dt^4} + a_{21}\frac{d^3y}{dt^3} + a_{20}\frac{d^2y}{dt^2}\right) + a_{11}\left(\frac{d^3y}{dt^3} + a_{21}\frac{d^2y}{dt^2} + a_{20}\frac{dy}{dt}\right) + \right.$$

$$\left. + a_{10}\left(\frac{d^2y}{dt^2} + a_{21}\frac{dy}{dt} + a_{20}y\right)\right] = b_{11}\frac{d^2u}{dt^2} + b_{10}\frac{du}{dt},$$

откуда

$$\frac{d^4y}{dt^4} + \left(a_{21} + a_{11}\right)\frac{d^3y}{dt^3} + \left(a_{20} + a_{11}a_{21} + a_{10}\right)\frac{d^2y}{dt^2} + \left(a_{11}a_{20} + a_{10}a_{21}\right)\frac{dy}{dt} +$$

$$+ a_{10}a_{20}y = b_{11}b_{21}\frac{d^2u}{dt^2} + b_{10}b_{21}\frac{du}{dt}.$$

Левые части уравнений для двух случаев (независимо действующих функций $x(t)$ и $dx(t)/dt$) полностью совпали. Немного вспомним математику. Она утверждает, что если имеются два дифференциальных уравнения с одинаковыми левыми частями:

$$a_n\frac{d^ny}{dt^n} + a_{n-1}\frac{d^{n-1}y}{dt^{n-1}} + \ldots + a_1\frac{dy}{dt} + a_0y = F_1(t),$$

$$a_n\frac{d^ny}{dt^n} + a_{n-1}\frac{d^{n-1}y}{dt^{n-1}} + \ldots + a_1\frac{dy}{dt} + a_0y = F_2(t),$$

причем $y_1(t)$ – решение первого, а $y_2(t)$ – решение второго, то $y_1(t) + y_2(t)$ будет решением уравнения:

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \ldots + a_1 \frac{dy}{dt} + a_0 y = F_1(t) + F_2(t).$$

(Принцип суперпозиции еще пока никто не отменял!)

На самом деле складываются не решения целиком, а только вынужденные составляющие «движения», т.е. частные решения неоднородных дифференциальных уравнений. Свободное же «движение» систем, описываемых двумя этими уравнениями, т.е. общее решение однородного уравнения, – *одно и то же*, см. п. 2.2.2.8.

Таким образом, чтобы учесть оба фактора ($x(t)$ и $dx(t)/dt$) мы должны сложить правые части полученных для каждого фактора дифференциальных уравнений. В результате окончательно получим:

$$\frac{d^4 y}{dt^4} + \left(a_{21} + a_{11}\right) \frac{d^3 y}{dt^3} + \left(a_{20} + a_{11} a_{21} + a_{10}\right) \frac{d^2 y}{dt^2} + \left(a_{11} a_{20} + a_{10} a_{21}\right) \frac{dy}{dt} +$$

$$+ a_{10} a_{20} y = b_{11} b_{21} \frac{d^2 u}{dt^2} + \left(b_{11} b_{20} + b_{10} b_{21}\right) \frac{du}{dt} + b_{10} b_{20} u.$$

По данному уравнению запишем передаточную функцию соединения, которая (о чудо!) полностью совпадает с полученной ранее чисто формальным способом:

$$W(p) = \frac{y}{u} =$$

$$= \frac{b_{11} b_{21} p^2 + \left(b_{11} b_{20} + b_{10} b_{21}\right) p + b_{10} b_{20}}{p^4 + \left(a_{21} + a_{11}\right) p^3 + \left(a_{20} + a_{11} a_{21} + a_{10}\right) p^2 + \left(a_{11} a_{20} + a_{10} a_{21}\right) p + a_{10} a_{20}}.$$

Конечно, одним примером ничего доказать нельзя, можно только *продемонстрировать*. Но существует метод, позволяющий находить дифференциальные уравнения различных соединений по дифференциальным уравнениям входящих в них звеньев. Этот метод называется *методом уравнивающих операторов* [7], и его применение показывает, что во всех структурных преобразованиях передаточными функциями можно пользоваться как *операторами*, т.е. *формально* считать, что выходной сигнал звена определяется как произведение входного сигнала на передаточную функцию звена.

Если передаточную функцию рассматривать как отношение изображений $y(p)$ и $u(p)$ сигналов по Лапласу, то выражение $y(p)=W(p)u(p)$ справедливо и без всяких «оговорок», просто в силу определения самой передаточной функции (но, правда, только при нулевых начальных условиях).

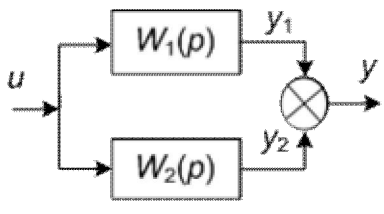


Рис. 2.43. Параллельное соединение.

Поэтому далее отвлекаться на «разъяснения» мы не будем.

Параллельное соединение звеньев представлено на рис. 2.43.

Здесь все тоже очевидно:

$y = y_1 + y_2 = W_1(p)\,u + W_2(p)\,u = (W_1(p) + W_2(p))u.$

Передаточная функция параллельно соединенных звеньев равна сумме их передаточных функций.

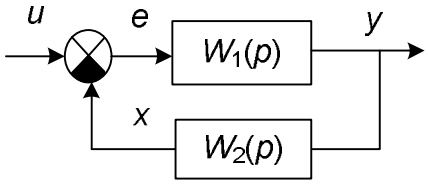


Рис. 2.44. Соединение в виде отрицательной обратной связи.

Соединение в виде обратной связи

Рассмотрим соединение в виде наиболее распространенной отрицательной обратной связи (рис. 2.44).

Чтобы найти передаточную функцию соединения, необходимо просто избавиться от ненужных переменных:

$$y = W_1(p)e = W_1(p)(u - x) = W_1(p)(u - W_2(p)y) = W_1(p)u - W_1(p)W_2(p)y,$$

$$y + W_1(p)W_2(p)y = W_1(p)u,$$

откуда

$$W(p) = \frac{y}{u} = \frac{W_1(p)}{1 + W_1(p)W_2(p)}.$$

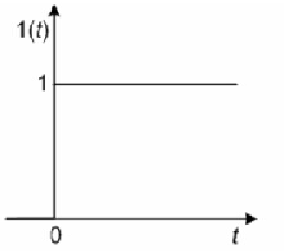
В случае если обратная связь положительна, в знаменателе, как нетрудно проверить, будет знак минус.

2.2.2.7. Характеристики систем автоматического регулирования

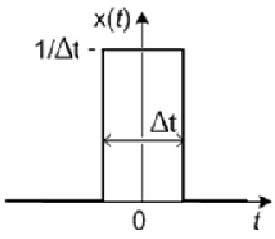
Системы автоматического регулирования, как и любые другие системы, имеют свои *характеристики*: числовые параметры и зависимости, которые позволяют определить, хороша ли система, удовлетворяет она всем предъявляемым требованиям (которые также формулируются в виде минимально достаточных характеристик).

Свойства элементов и систем управления определяются по их реакции на типовые воздействия.

Основными такими воздействиями являются: *ступенчатые*, *импульсные*, *гармонические* (рис. 2.45).

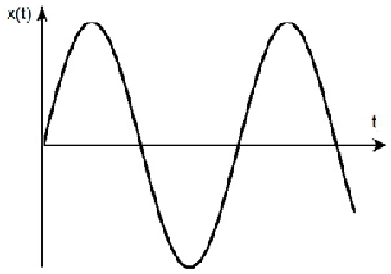


$$1(t) = \begin{cases} 1, & t \geq 0, \\ 0, & t < 0, \end{cases}$$



$$\delta(t) = \begin{cases} 0, & t \neq 0, \\ \infty, & t = 0, \end{cases}$$

$$\int_{-\infty}^{\infty} \delta(t) \cdot dt = 1.$$



$$x(t) = X_m \sin(\omega t),$$

Рис. 2.45. Типовые воздействия.

Первое типовое воздействие называется *единичным ступенчатым воздействием*. В некоторый момент времени, от которого мы начинаем отчет, входной воздействие мгновенно изменяется от нуля до единицы. В реальности, конечно, такого не бывает: ничего не изменяется мгновенно. Но, как говорится, «все познается в сравнении». Если нас интересует процесс накопления мух и мотыльков в комнате с открытым окном при включении света ночью, нам совершенно незачем вычислять время разогрева вольфрамовой нити в лампочке.

Почему воздействие принято «единичным», тоже понятно. Было бы странным принять его равным, скажем, 27,101973. В целом можно сказать, что такое воздействие «символизирует» некоторый процесс, который одним словом можно назвать «включением» (или «выключением»). Система регулирования включается в работу, оператор изменяет задание и переводит систему на автоматическое управление, нагрузка генератора мгновенно изменяется при включении нового потребителя и т.д.

Реакция системы на единичное ступенчатое воздействие называется *переходной характеристикой h(t)* (рис. 2.46).

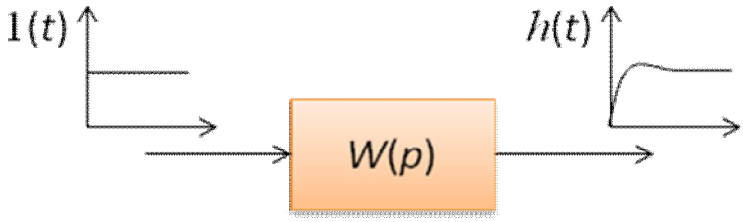


Рис. 2.46. Определение переходной характеристики.

Если речь идет об изменении задания для системы регулирования (как показано на рис. 2.46), то выходная (регулируемая) величина должна, без сомнения, достаточно быстро и без лишних колебаний прийти к новому заданному значению. Если ступенчато изменилось возмущение, то в идеале мы вообще не должны этого заметить, т.е. идеальная реакция на возмущение должна быть равна нулю. Однако, вспомнив классификацию систем, мы вынуждены прийти к неутешительному выводу: полное подавление возмущений возможно лишь в идеально настроенной системе регулирования по возмущению или в комбинированной системе. На практике реакция будет иметь место, но она должна быть минимальной: с минимальным остаточным отклонением (или лучше вообще без него), максимально быстро затухающей, без сильных колебаний.

Второе типовое воздействие называется единичным импульсным. Это импульс бесконечно малой длительности и бесконечно большой амплитуды, площадь которого равна единице. Здесь мы вообще попадаем в «terra incognita» («неизвестная земля», лат.). Что это значит: «импульс бесконечно малой длительности и бесконечно большой амплитуды, площадь которого равна единице»? На самом деле ничего сложного здесь нет. Действительно, при достаточно быстром возмущении объекты управления «реагируют» не на амплитуду, а на «площадь» воздействия (не на «мощность», а на «энергию»). Возьмем, к примеру, все тот же процесс нагрева. Предположим, речь идет о нагреве 100 литров воды и в первый раз Вы включили нагревательный элемент мощностью 100

кВт на 1 с, а во второй – мощностью 10 кВт на 10 сек. В обоих случаях Вы подвели (и достаточно быстро) к воде энергию, равную 100 кДж. И реакция, как и следовало ожидать, будет практически одинаковой. Другое дело, если Вы включите 10-ти ваттный нагреватель: тогда, может быть, и реакции никакой не последует, сколько бы Вы не держали его во включенном состоянии. Вся энергия будет просто отдаваться на нагрев окружающей среды. Так вот, чтобы избежать последнего сценария, и придумано это самое бесконечно малое время, а про «единицу» мы уже говорили.

Реакция системы на единичный импульс называется *импульсной переходной характеристикой* $\omega(t)$ (рис. 2.47).

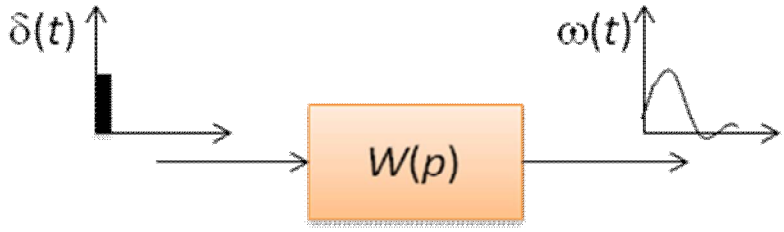


Рис. 2.47. Определение импульсной переходной характеристики.

Таким образом, эта характеристика показывает способность системы реагировать на быстрые «ударные» воздействия (возмущения).

У нее есть и другой, «сугубо математический» смысл, который мы здесь приведем без подробных пояснений, как «магическую формулу»: импульсная переходная характеристика (другое название – *весовая функция*) есть оригинал преобразования Лапласа передаточной функции системы. Т.е. в некотором смысле передаточная функция – это «зашифрованная фотография» импульсной характеристики.

По переходным характеристикам оценивают *быстродействие* и *колебательность* системы. Что касается быстродействия, то тут никаких вопросов возникнуть не должно: чем быстрее система работает (выполняет задание, подавляет возмущения), тем лучше. Однако не все так просто. Теоретически мы можем добиться любого быстродействия. Хотим нагреть воду в 10 раз быстрее – возьмем нагревательный элемент в 10 раз мощнее, и даже, если нам надо «мгновенно», что-нибудь придумаем – какие-нибудь «микровзрывы» организуем. Практически же речь идет о том, чтобы на базе *имеющейся* техники добиться максимального быстродействия.

С колебательностью все немного сложнее. Непонятно, зачем вообще нужна эта самая колебательность. Почему нельзя остановиться прямо у «стоп-линии» и больше не ездить туда-сюда? Дело в том, что если Вы остановились прямо у «стоп-линии», то, скорее всего до этого слишком медленно ехали. Максимально быстрый процесс – это процесс с небольшими колебаниями. Именно появление колебаний свидетельствует о том, что система управления «выжала по-полной», тогда как неколебательный процесс чаще всего говорит, что «есть резервы». Здесь уже не работает аналогия с автомобилем, так как «включение задней» не рассматривается как признак отсутствия мастерства. Вполне допустимо, что регулируемая величина в переходном процессе времен-

но слегка превысит задание, ведь до этого она была намного дальше от него по модулю! Зато мы отчетливо видим «рвение» системы. Гораздо хуже, когда эта самая величина медленно-медленно ползет к заданному значению. Конечно, есть отдельные случаи, когда колебательность недопустима, но пока мы их не рассматриваем.

Гармонические воздействия на систему «подают», когда уже точно нужно «раздеть ее до нитки». С математической точки зрения и переходная и импульсная переходная характеристики системы несут полную информацию о ее свойствах. Но это так, когда эти характеристики представлены в виде формул. Если они же они даны как графические зависимости, разглядеть за ними динамические свойства системы также трудно, как идентифицировать марку автомобиля по показаниям очевидцев, находящихся за 100 метров от ДТП в темное время суток. Разные системы дают примерно один и тот же отклик. Другое дело, если на камерах слежения Вы увидите все детали автомобиля под разными углами, передвигающегося с разными скоростями. Так и здесь: Вы видите отклик системы на входные колебания с разными частотами. Какие-то частоты передаются почти без изменений, какие-то вообще подавляются, а какие-то, наоборот, усиливаются.

В целом частотные характеристики описывают способность системы передавать гармонический сигнал в зависимости от его частоты. Особенностью линейных систем, если они *устойчивы* (про устойчивость см. ниже), является то обстоятельство, что они никакие сигналы не «портят»: подали на них постоянный сигнал, – на выходе после переходного процесса наблюдаем постоянную величину, подали линейно возрастающий, такой же сигнал видим и на выходе. Также обстоит дело и с гармоническим воздействием: система не меняет форму сигнала и его частоту, изменяется лишь амплитуда и фаза колебаний (рис. 2.48).

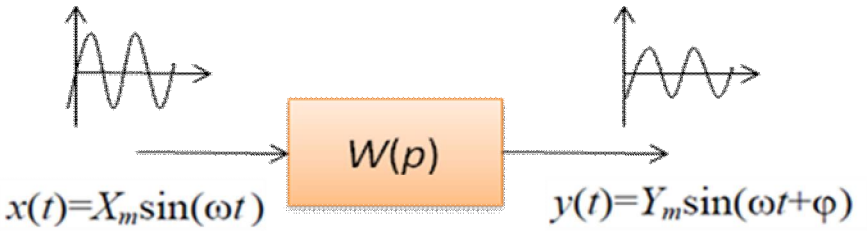


Рис. 2.48. Определение частотных характеристик.

*Амплитудно-частотная характеристика (АЧХ) – зависимость отно*шения амплитуды выходных колебаний к амплитуде входных от частоты:

$$A(\omega)=\frac{Y_m}{X_m}(\omega).$$

Фазочастотная характеристика (ФЧХ) – зависимость разности фаз выходных и входных колебаний от частоты: $\varphi(\omega)$.

На рис. 2.49 показаны АЧХ и ФЧХ системы с передаточной функцией

$$W(p)=\frac{1}{p^3+2p^2+p+1}.$$

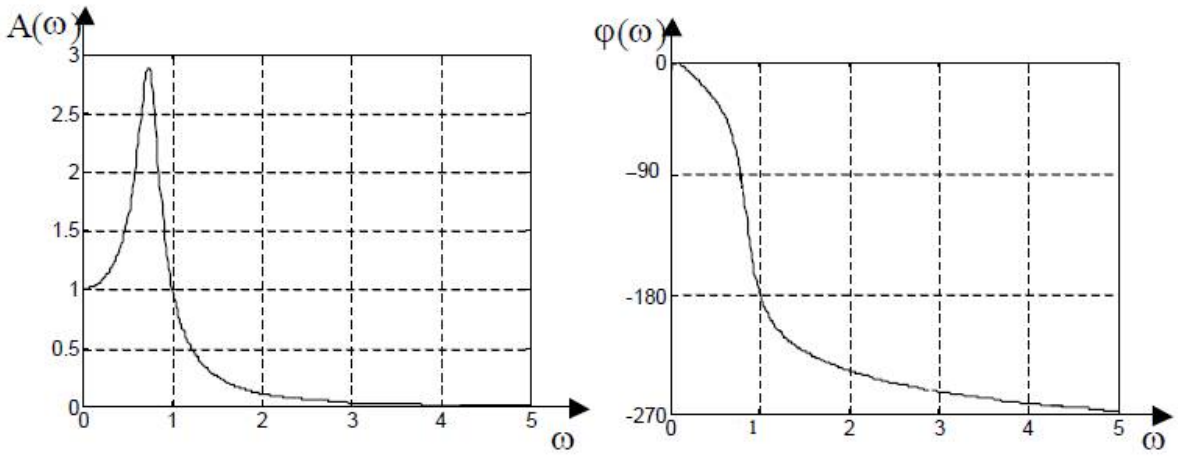


Рис. 2.49. Частотные характеристики системы

О чем можно судить по этим характеристикам? О многом, если вообще не обо всем. Начальное значение АЧХ (при ω = 0) – это коэффициент передачи системы. Коэффициент передачи показывает способность передачи постоянного сигнала, а постоянный сигнал и есть сигнал нулевой частоты. «Ширина» АЧХ говорит о быстродействии системы: чем шире АЧХ, тем *уже* переходная характеристика и наоборот. Действительно, если система может передавать большие частоты, то она «быстрая» и, наоборот, быстрая система способна передавать высокочастотные сигналы. Наличие максимума на ненулевой частоте свидетельствуют о колебательности переходных процессов, причем, чем больше этом максимум, тем сильнее колебательность. Сама же частота, на которой этот максимум наблюдается, близка частоте колебаний. На рис. 2.50 показан переходная характеристика системы. Частота колебаний примерно равна 0,7 рад/сек.

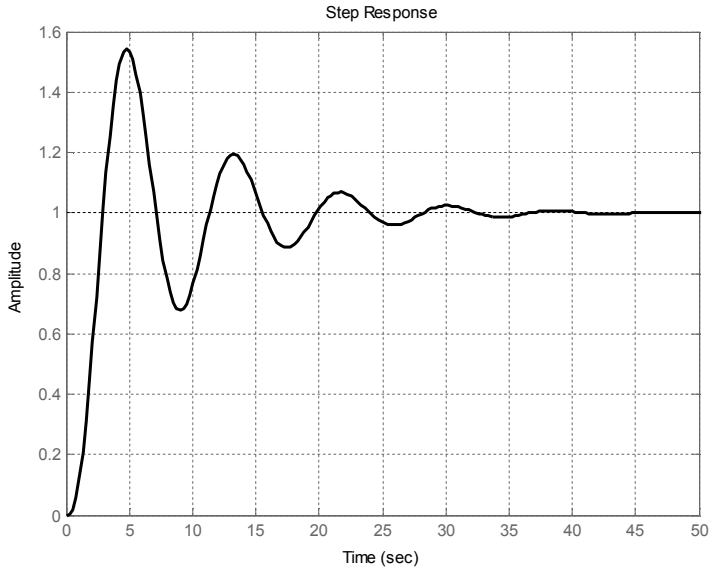


Рис. 2.50. Переходная характеристика системы.

Предельный фазовый сдвиг (значение, к которому стремится ФЧХ при ω → ∞) говорит об относительном порядке системы (разности между порядком знаменателя и порядком числителя передаточной функции). При этом порядок

(степень полинома) знаменателя умножается на –90°, а порядок числителя на +90°. Так, в нашем случае знаменатель дает –270°, а числитель 0°.

АЧХ и ФЧХ-системы часто представляют в логарифмическом виде.

Логарифмические характеристики используют логарифмическую шкалу частот. Единицей измерения на данной шкале обычно является декада (дек) – интервал, на концах которого значения частоты отличаются друг от друга в 10 раз.

Логарифмическая амплитудно-частотная характеристика (ЛАЧХ) определяется формулой

$$L(\omega) = 20\lg A(\omega)$$

и измеряется в децибелах. Что это за *децибелы* такие? Бел – это не только фамилия американского ученого, но и, как не странно, единица измерения десятичного логарифма отношения мощностей сигналов. Если $\lg(P_2/P_1) = 1$, то есть $P_2/P_1 = 10$, то говорят, что P_2 больше P_1 на 1 Бел. А вот если $10\lg(P_2/P_1) = 1$, или $P_2/P_1 \approx 1{,}26$, говорят, что P_2 больше P_1 на 1 децибел. У нас же речь идет об отношении не мощностей сигналов, а их амплитуд. Всем известно, что мощность пропорциональна квадрату амплитуды. Вспомним хотя бы закон Джоуля – Ленца. Отсюда в формуле появляется число 20:

$$10\lg\frac{P_2}{P_1} = 10\lg\frac{kA_2^2}{kA_1^2} = 20\lg\frac{A_2}{A_1}$$

Логарифмическая фазочастотная характеристика (ЛФЧХ) (рис. 2.51), как и ФЧХ, измеряется в градусах или радианах.

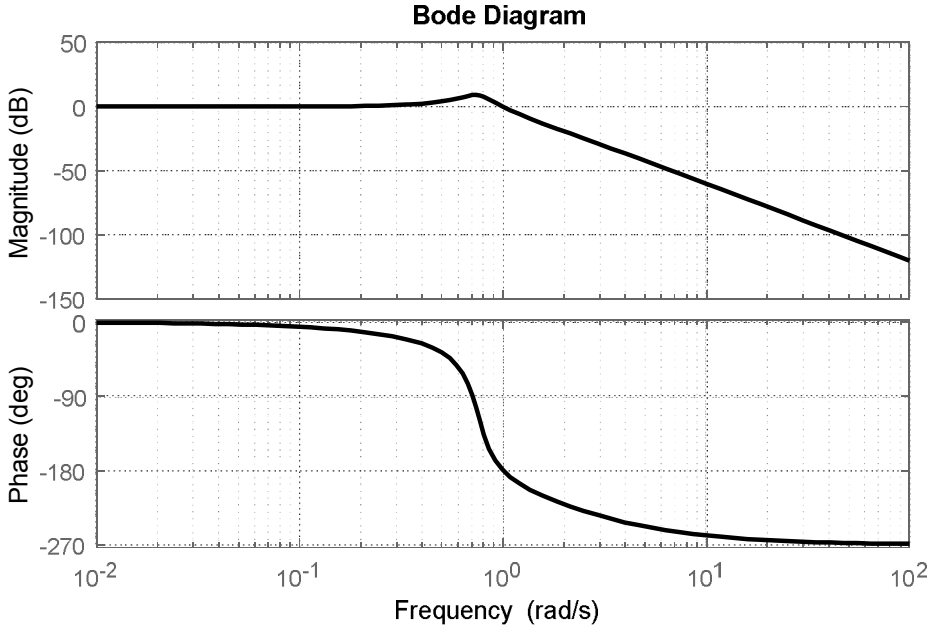


Рис. 2.51. Логарифмические частотные характеристики системы.

АЧХ и ФЧХ системы объединяются в *амплитудно-фазовую частотную характеристику* АФЧХ (рис. 2.52):

$$W(j\omega) = A(\omega)e^{j\varphi(\omega)}.$$

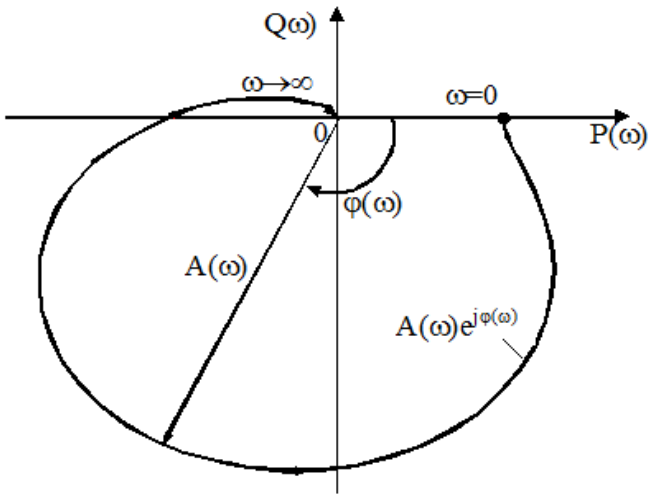


Рис. 2.52. Амплитудно-фазовая частотная характеристика системы.

Здесь АЧХ играет роль модуля, а ФЧХ – аргумента комплексной функции частоты.

АФЧХ можно представить в виде суммы вещественной (ВЧХ) и мнимой (МЧХ) частотных характеристик:

$$A(\omega)e^{j\varphi(\omega)} = A(\omega)\cos\varphi(\omega) + jA(\omega)\sin\varphi(\omega) = P(\omega) + jQ(\omega)\,.$$

Выражение для АФЧХ можно получить из передаточной функции подстановкой $p \to j\omega$. Это легко доказать. Пусть на вход системы с передаточной функцией

$$W(p) = \frac{y}{u} = \frac{p^m + b_{m-1}p^{m-1} + \ldots + b_1 p + b_0}{p^n + a_{n-1}p^{n-1} + \ldots + a_1 p + a_0}$$

подается гармонический сигнал с частотой ω и амплитудой $A_{вх}$:

$$u = A_{вх}\sin(\omega t) = A_{вх}e^{j\omega t}\,.$$

Поскольку речь идет о вынужденном движении (см. п. 2.2.2.8), выходной сигнал системы – тоже гармонический, той же частоты, но с другой амплитудой $A_{вых}$ и со сдвигом фазы φ:

$$y = A_{вых}\sin(\omega t + \varphi) = A_{вых}e^{j(\omega t + \varphi)}$$

Поскольку формально $y = W(p)u$, получим

$$A_{вых}e^{j(\omega t+\varphi)}\left(p^n + a_{n-1}p^{n-1} + \ldots + a_1 p + a_0\right) =$$
$$= A_{вх}e^{j\omega t}\left(p^m + b_{m-1}p^{m-1} + \ldots + b_1 p + b_0\right).$$

На самом деле мы всего лишь записали дифференциальное уравнение в операторном виде, подставив в него «известные» нам выражения для входного и выходного сигналов. Умножение на оператор p в степени k означает, что нужно взять k-ю производную сигнала по времени. Поэтому левая часть уравнения примет вид:

$$A_{вых}e^{j(\omega t+\varphi)}\left[p^n + a_{n-1}p^{n-1} + ... + a_1 p^1 + a_0\right] =$$

$$= A_{вых}(j\omega)^n e^{j(\omega t+\varphi)} + a_{n-1}A_{вых}(j\omega)^{n-1}e^{j(\omega t+\varphi)} + ...$$

$$+ a_1 A_{вых} j\omega e^{j(\omega t+\varphi)} + a_0 A_{вых}e^{j(\omega t+\varphi)} =$$

$$= A_{вых}e^{j(\omega t+\varphi)}\left[(j\omega)^n + a_{n-1}(j\omega)^{n-1} + ... + a_1 j\omega + a_0\right].$$

Аналогично можно записать и для правой части:

$$A_{вх}e^{j\omega t}\left[p^m + b_{m-1}p^{m-1} + ... + b_1 p + b_0\right] =$$

$$= A_{вх}(j\omega)^m e^{j\omega t} + b_{m-1}A_{вх}(j\omega)^{m-1}e^{j\omega t} + ... + b_1 A_{вх} j\omega e^{j\omega t} + b_0 A_{вх}e^{j\omega t} =$$

$$= A_{вх}e^{j\omega t}\left[(j\omega)^m + b_{m-1}(j\omega)^{m-1} + ... + b_1 j\omega + b_0\right].$$

Однако левая часть должна быть равна правой, поэтому

$$A_{вых}e^{j(\omega t+\varphi)}\left[(j\omega)^n + a_{n-1}(j\omega)^{n-1} + ... + a_1 j\omega + a_0\right] =$$

$$= A_{вх}e^{j\omega t}\left[(j\omega)^m + b_{m-1}(j\omega)^{m-1} + ... + b_1 j\omega + b_0\right].$$

Отсюда

$$\frac{A_{вых}e^{j(\omega t+\varphi)}}{A_{вх}e^{j\omega t}} = \frac{(j\omega)^m + b_{m-1}(j\omega)^{m-1} + ... + b_1 j\omega + b_0}{(j\omega)^n + a_{n-1}(j\omega)^{n-1} + ... + a_1 j\omega + a_0}.$$

Отношение $A_{вых}/A_{вх}$ есть значение АЧХ на частоте ω, а

$$\frac{e^{j(\omega t+\varphi)}}{e^{j\omega t}} = e^{j\varphi},$$

где φ – есть не что иное, как значение ФЧХ на той же частоте. В левой же части уравнения мы видим все ту же передаточную функцию, только вместо оператора p – оператор $j\omega$. Поэтому

$$\frac{A_{вых}e^{j(\omega t+\varphi)}}{A_{вх}e^{j\omega t}} = A(\omega)e^{j\varphi} = W(j\omega).$$

Таким образом, по передаточной функции легко находится АФЧХ системы и далее все интересующие нас ее частотные характеристики.

2.2.2.8. Устойчивость

Первая паровая машина, нашедшая промышленное применение, была построена еще в конце XVII в. Однако настоящую промышленную революцию вызвало появление в 80-х гг. XVIII в. паровой машины Джеймса Уатта. Среди многочисленных изобретений, сделанных для этой машины, был и центробежный регулятор частоты вращения – регулятор Уатта. Примерно в середине XIX века были зафиксированы многочисленные случаи неудовлетворительной работы регуляторов частоты на самых различных паровых двигателях. Вместо стабилизации частоты регуляторы начинали ее «раскачивать» или имело место «рыскание» – незатухающие колебания скорости. К решению проблемы подключились многие инженеры и математики того времени. В результате появилась теория устойчивости систем автоматического регулирования, из которой, по-видимому, и родилась теория автоматического управления.

Общее определение устойчивости можно сформулировать следующим образом:

Система устойчива, если ее координаты (переменные состояния, выходные величины) остаются ограниченными в условиях действия на систему ограниченных по величине воздействий.

Отметим, что неустойчивая система неработоспособна, так как внешнее воздействие приводит к теоретически неограниченному изменению ее переменных и поэтому САР не может выполнить ни задачу стабилизации, ни задачу слежения. На практике все переменные системы ограничены своими предельными значениями, что, однако, не меняет сути дела.

Для линейных систем справедливы следующие утверждения:

1) устойчивость системы – ее внутреннее свойство, которое не зависит от величины и характера приложенных к системе воздействий;

2) устойчивость определяется для системы в целом: не может одна часть системы быть устойчивой, а другая – неустойчивой. Если все же такой эффект имеет место, то это свидетельствует о том, что исследуется не одна, а две системы – либо полностью независимые друг от друга, либо с односторонним влиянием устойчивой системы на неустойчивую.

В отличие от середины XIX в., сейчас уже многие люди умеют решать линейные дифференциальные уравнения с постоянными коэффициентами. Такие люди, даже не занимаясь вопросами управления, вполне могут определить, устойчива или нет та или иная линейная система.

Как известно, движение системы, находящейся под внешним воздействием, состоит из двух составляющих (рис. 2.53):

1) *вынужденное движение*, обычно повторяющее по форме внешнее воздействие;

2) *свободное (собственное) движение* системы, выведенной из состояния равновесия.

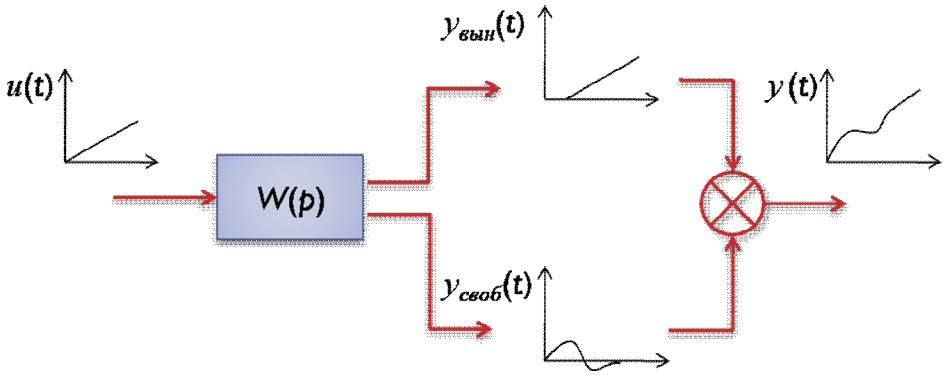


Рис. 2.53. Движение системы под действием внешнего воздействия.

Вынужденное движение определяется как частное решение неоднородного уравнения (т.е. *всего* уравнения при подстановке в ее правую часть выражения для $u(t)$), а свободное – как общее решение однородного (с нулевой правой частью):

$$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \ldots + a_2 \ddot{y} + a_1 \dot{y} + a_0 y = 0.$$

Именно свободное движение определяет основные свойства системы, в том числе устойчивость. Его общий вид:

$$y_{св}(t) = C_1 e^{p_1 t} + C_2 e^{p_2 t} + ... + C_n e^{p_n t}.$$

где $C_1, C_2, \ldots, C_n$ – постоянные интегрирования; $p_1, p_2, \ldots, p_n$ – корни характеристического уравнения:

$$a_n p^n + a_{n-1} p^{n-1} + ... + a_2 p^2 + a_1 p + a_0 = 0.$$

Эти корни называют *полюсами* передаточной функции.

Чтобы понять, что к чему, рассмотрим пример.

Пусть у нас имеется, например, система с передаточной функцией

$$W(p) = \frac{y}{u} = \frac{5p+1}{p^2+3p+2}.$$

Ей соответствует дифференциальное уравнение:

$$\frac{d^2 y}{dt^2} + 3\frac{dy}{dt} + 2y = 5\frac{du}{dt} + u.$$

Пусть $u(t) = 2t - 1$.

Попробуем его решить при условии, что система до подачи входного воздействия находилась в полном покое.

Найдем вынужденное движение. Для этого подставим выражение для $u(t)$ в дифференциальное уравнение:

$$\frac{d^2 y}{dt^2} + 3\frac{dy}{dt} + 2y = 9 + 2t.$$

Математики настоятельно советуют выбирать структуру вынужденного решения подобной структуре правой части уравнения:

$$y_{вын}(t) = A + Bt.$$

Подставим это выражение в уравнение и получим:

$$0 + 3B + 2(A + Bt) = (2A + 3B) + 2Bt = 9 + 2t,$$

откуда найдем $B = 1$, $A = 3$. Таким образом, $y_{вын}(t) = 3 + t$. Поскольку решение найдено, значит, его структура была выбрана верно.

Найдем свободное движение как общее решение однородного дифференциального уравнения

$$\frac{d^2 y}{dt^2} + 3\frac{dy}{dt} + 2y = 0.$$

Характеристический полином (он же знаменатель передаточной функции):

$$p^2 + 3p + 2 = 0.$$

Полином имеет корни $p_1 = -1$, $p_2 = -2$, в чем можно убедиться непосредственной подстановкой. Свободная составляющая движения:

$$y_{св}(t) = C_1 e^{-1t} + C_2 e^{-2t}.$$

Каждое из двух слагаемых само по себе является решением однородного уравнения. Проверим первое слагаемое (множитель C_1 можно не брать в расчет, все равно он сокращается):

$$\frac{d^2}{dt^2}\left(e^{-t}\right)+3\frac{d}{dt}\left(e^{-t}\right)+2e^{-t}=(-1)^2e^{-t}+3\times(-1)e^{-t}+2e^{-t}=$$

$$=\left[(-1)^2+3\times(-1)+2\right]e^{-t}=0.$$

Еще не поняли, в чем суть? Тогда повторим для второго:

$$\frac{d^2}{dt^2}\left(e^{-2t}\right)+3\frac{d}{dt}\left(e^{-2t}\right)+2e^{-2t}=(-2)^2e^{-2t}+3\times(-2)e^{-2t}+2e^{-2t}=$$

$$=\left[(-2)^2+3\times(-2)+2\right]e^{-2t}=0.$$

Если и сейчас не понятно, тогда нам далеко до Эйлера. В квадратных скобках мы имеем все тот же характеристический полином, но только с подставленным в него корнем. А раз это корень, то полином равен нулю. Вся «прелесть» экспоненциальной функции состоит в том, что, сколько ее не дифференцируй, она все равно останется самой собой, только впереди добавляется множитель. А чтобы «пасьянс сошелся», нужно включать в показатели степени корни характеристического полинома. Вот так мы и повторили открытие Эйлера.

Полное решение дифференциального уравнения имеет вид:

$$y(t)=y_{вын}(t)+y_{св}(t)=3\ +t+C_1e^{-1t}+C_2e^{-2t}.$$

Найдем постоянные интегрирования C_1 и C_2. Для этого нужно знать дополнительные условия $y(0)$ и $dy/dt(0)$. До подачи воздействия система находилась в покое. Значит ли это, что начальные условия нулевые? Для математиков, наверное, – да, а для нас – нет (см. п. 2.2.2.3). Структурная схема, соответствующая нашему дифференциальному уравнению, показана на рис. 2.54

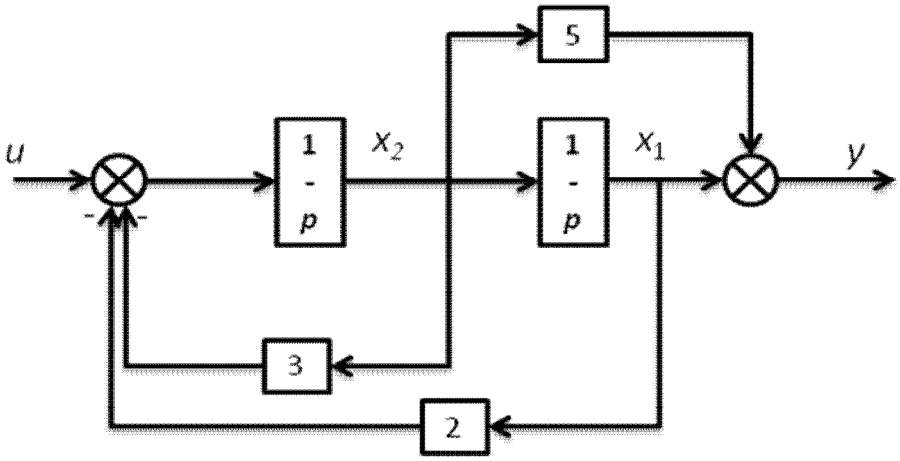


Рис. 2.54. Структурная схема.

Для нас состояние полного покоя означает, что сигналы на выходах интеграторов в начальный момент времени равны нулю: $x_1(0) = x_2(0) = 0$.

Выходной сигнал: $y(0) = x_1(0) + 5x_2(0) = 0$.

Его производная:

$$\frac{dy}{dt}(0)=\frac{dx_1}{dt}(0)+5\frac{dx_2}{dt}(0).$$

Однако согласно схеме

$$\frac{dx_1}{dt} = x_2, \quad \frac{dx_2}{dt} = -2x_1 - 3x_2 + u.$$

Таким образом:

$$\frac{dy}{dt}(0) = x_2(0) + 5\left(-2x_1(0) - 3x_2(0) + u(0)\right) = 5u(0) = -5.$$

Из условий $y(0) = 0$, $dy/dt(0) = -5$ получим:

$$y(0) = 3\ +0 + C_1 e^{-1\cdot 0} + C_2 e^{-2\cdot 0} = 3 + C_1 + C_2 = 0,$$

$$\frac{dy}{dt}(0) = 0\ +1 + (-1)C_1 e^{-1\cdot 0} + (-2)C_2 e^{-2\cdot 0} = 1 - C_1 - 2C_2 = -5.$$

Таким образом, имеем систему уравнений:

$$\begin{cases} C_1 + C_2 = -3; \\ -C_1 - 2C_2 = -6. \end{cases}$$

Решая ее, получим: $C_1 = -12$, $C_2 = 9$. Окончательно решение примет вид:

$$y(t) = 3\ + t - 12e^{-1t} + 9e^{-2t}.$$

Проверить это решение в Matlab можно с помощью кода:

```
syms y(t);
Dy = diff(y);
D2y = diff(y,2);
dsolve(D2y+3*Dy + 2*y == 9 + 2*t,y(0)==0,Dy(0)==-5)
ans =

t - 12*exp(-t) + 9*exp(-2*t) + 3
```

Следующий фрагмент кода строит график функции $y(t)$, рис. 2.55, а заодно проверяет правильность нахождения начальных условий:

```
t = 0:.1:10;
y = 3+t-12*exp(-t) + 9*exp(-2*t);
plot(t,y)
W = tf([5 1],[1 3 2]);
u = 2*t-1;
y1 = lsim(W,u,t);
plot(t,y,t,y1), grid
```

На самом деле строятся два графика: непосредственно по полученному выражению и график, рассчитанный функцией `lsim` по передаточной функции.

Графики полностью совпали, что свидетельствует о правильности расчетов. Тонкая линия, подрисованная «вручную», – это график вынужденного движения $y_{вын}(t) = 3 + t$.

В рассмотренном примере у характеристического полинома оказались простые отрицательные корни, и свободное движение с течением времени экспоненциально затухает.

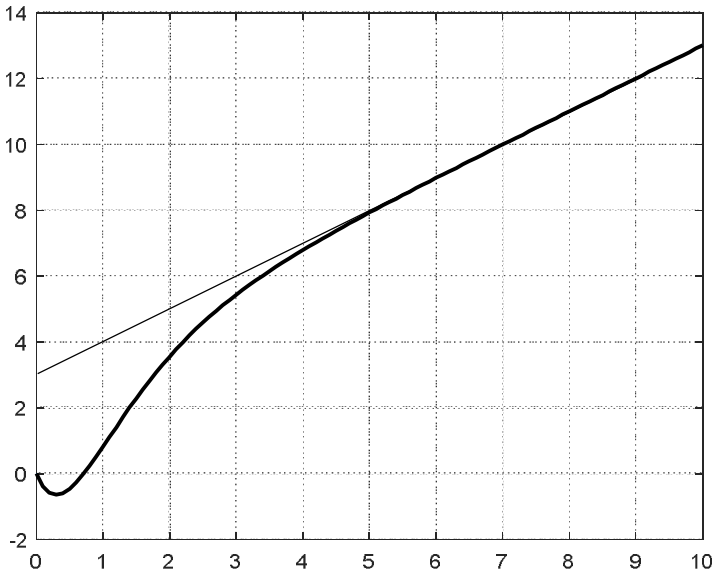


Рис. 2.55. Реакция системы.

Часто среди корней попадаются и комплексно сопряженные пары: $p_{i,i+1} = \alpha_i \pm j\beta_i$. Вспомним решение квадратного уравнения и тот ужас, который нас охватывал, когда дискриминант оказывался отрицательным. На самом деле, ничего страшного в этом нет: квадратный корень из отрицательного числа существует, просто он «мнимый»:

$$\sqrt{-4} = 2\sqrt{-1} = 2j.$$

В таком случае в свободном решении появляются слагаемые вида

$$C_i e^{(\alpha_i + j\beta)t} + C_{i+1} e^{(\alpha_i - j\beta)t} = e^{\alpha_i t}\left[C_i\left(\cos(\beta t) + j\sin(\beta t)\right) + C_{i+1}\left(\cos(\beta t) - j\sin(\beta t)\right)\right] =$$
$$= e^{\alpha_i t}\left[\left(C_i + C_{i+1}\right)\cos(\beta t) + j\left(C_i - C_{i+1}\right)\sin(\beta t)\right].$$

Очевидно, что свободное движение системы должно описываться вещественной функцией времени. Следовательно, множители перед косинусом и синусом должны быть вещественными числами. Обозначим их A и B:

$$C_i e^{(\alpha_i + j\beta)t} + C_{i+1} e^{(\alpha_i - j\beta)t} = e^{\alpha_i t}\left[A\cos(\beta t) + B\sin(\beta t)\right].$$

Связь между постоянными:

$$A = C_i + C_{i+1}, \quad B = j\left(C_i - C_{i+1}\right),$$

$$C_i = \frac{B + jA}{2j}, \quad C_{i+1} = \frac{jA - B}{2j}.$$

Запись решения в комплексном виде обычно не практикуется. Решение уравнений сразу записывается в вещественном виде (через A и B).

Как видно из выражений, пара комплексно-сопряженных корней дает колебательную составляющую свободного движения. При этом колебания затухают, если $\alpha < 0$, или «расходятся», если $\alpha > 0$. В случае, когда $\alpha = 0$, амплитуда колебаний остается неизменной.

Бывает и так, что корни равны друг другу (*кратные корни*). Кратным корням $p_i = p_{i+1} = p_{i+2} = \ldots p_{i+k}$ в решении соответствует сумма

$$C_i e^{p_i t} + C_{i+1} t e^{p_i t} + C_{i+1} t^2 e^{p_i t} + \ldots + C_n t^k e^{p_i t}.$$

Независимо от значений постоянных интегрирования свободная составляющая будет с течением времени стремиться к нулю (затухать), если все корни характеристического уравнения отрицательные или имеют отрицательные вещественные части. Если хотя бы у одного корня вещественная часть положительна, соответствующая экспонента будет неограниченно расти со временем, а, следовательно, такая система неустойчива.

Необходимым и достаточным условием устойчивости линейной системы является отрицательность вещественных частей всех корней ее характеристического полинома.

Частными случаями являются следующие два:

1) нулевые корни. В системе с нулевыми корнями имеются «свободные», т.е. неохваченные обратными связями, интеграторы. Как ведет себя интегратор, мы уже знаем. У полиномов замкнутых систем, конечно, нулевых коней быть никак не может, а для разомкнутых – это вполне нормально. Система, характеристический полином которой имеет нулевые корни, находится на *апериодической (неколебательной) границе устойчивости*;

2) чисто мнимые комплексно сопряженные пары. Как уже было сказано выше, в этом случае в системе устанавливаются незатухающие колебания (*автоколебания*), с частотой, равной модулю корней из пары. Система, характеристический полином которой имеет мнимые корни, находится на *колебательной границе устойчивости*.

По каким причинам система автоматического регулирования может стать неустойчивой? Конкретных причин может быть много, но все они так или иначе сводятся к следующим «общим».

Во-первых, обыкновенное «головотяпство». Перепутали полярность при подключении датчика, получили не *отрицательную*, а *положительную* обратную связь в главном контуре. В результате регулятор будет работать не на уменьшение, а на увеличение ошибки. При этом переходный процесс будет расходящимся апериодическим, что соответствует положительным вещественным корням в характеристическом полиноме;

Во-вторых, причиной неустойчивости может стать «чрезмерное усердие» регулятора. Вы, моясь под душем, почувствовали себя не совсем комфортно: что-то «похолодало». Приоткрывая кран горячей воды, Вы слегка «перестарались» и стало несколько «жарковато». Пытаясь поправить ситуацию, Вы снижаете подачу горячей воды, но опять в большей степени, чем это было нужно: стало холоднее, чем «в первый раз». И так далее. Температура будет «раскачиваться», подобно качелям, и Вам придется срочно покинуть душевую кабину. Конечно, человек не будет действовать описанным способом. Но кто помешает так действовать «безумной» машине? Точнее, не «безумной», а просто неправильно настроенной. Процесс расходящихся колебаний соответствует паре комплексно сопряженных корней с положительной вещественной частью.

Рассмотрим влияние коэффициента регулятора на устойчивость и качество переходных процессов на примере системы, приведенной на рис. 2.56.

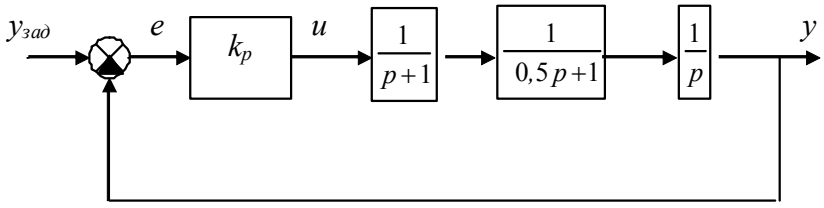


Рис. 2.56. Структурная схема САР.

Разомкнутая система нейтральна, так как состоит из трех устойчивых звеньев и одного нейтрального. Ее передаточная функция равна

$$W_{раз}(p) = \frac{2k_p}{p^3 + 3p^2 + 2p}.$$

Корни характеристического полинома разомкнутой системы: 0, –1, –2. Передаточная функция замкнутой системы

$$W_{зам}(p) = \frac{W_{раз}(p)}{1 + W_{раз}(p)} = \frac{2k_p}{p^3 + 3p^2 + 2p + 2 + 2k_p}.$$

На рис. 2.57 приведен *корневой годограф* системы – зависимость положения корней ее характеристического полинома на комплексной плоскости от коэффициента передачи регулятора.

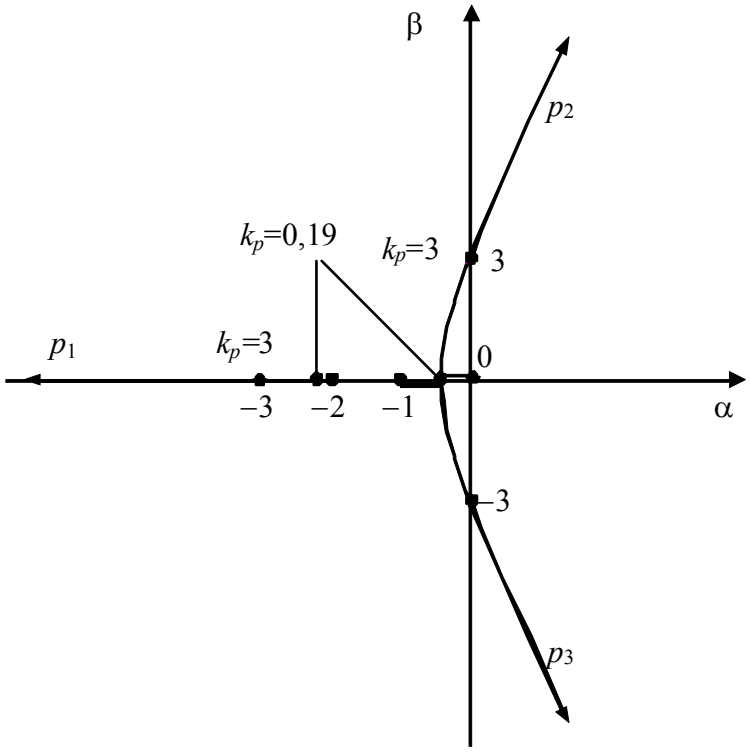


Рис. 2.57. Корневой годограф системы.

На рис. 2.58, 2.59 показаны переходные характеристики системы с различными значениями k_p.

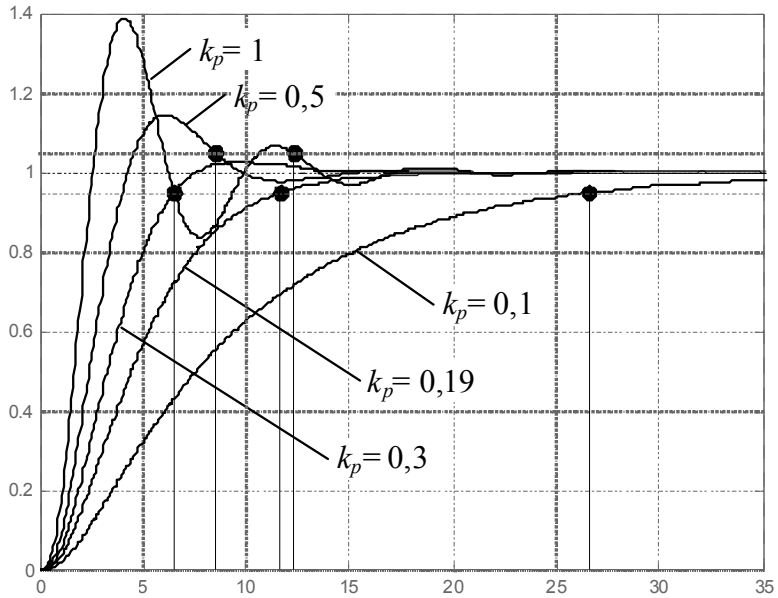


Рис. 2.58. Переходные характеристики системы при k_p = 0,1; 0,19; 0,3; 0,5; 1.

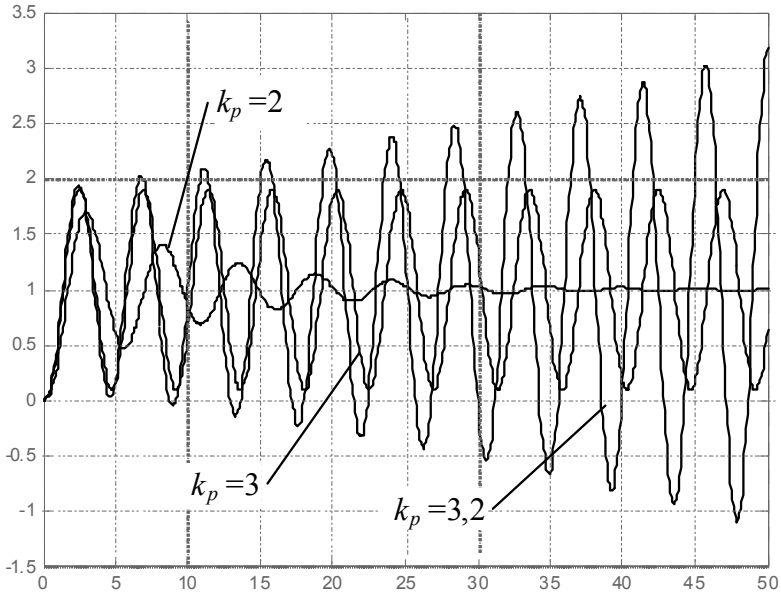


Рис. 2.59. Переходные характеристики системы при k_p = 2; 3; 3,2.

При $k_p \to 0$ корни характеристического полинома замкнутой системы стремятся к полюсам $W_{раз}(p)$: 0, –1, –2.

В диапазоне коэффициентов передачи от k_p = 0 до $k_p \approx 0{,}19$ происходит «сближение» полюсов p_2 и p_3 и уменьшение полюса p_1. Переходные характеристики системы носят апериодический характер, причем с увеличением k_p быстродействие системы растет.

При $k_p \approx 0{,}19$ полюса p_2 и p_3 становятся равными –0,42. Полюс p_1 принимает значение –2,15. Дальнейшее увеличение k_p приводит к тому, что полюса p_2 и p_3 вновь «расходятся», становясь комплексно-сопряженными. Вещественные части полюсов и модуль мнимых частей увеличиваются.

Полюс p_1 по-прежнему уменьшается, оказывая все меньшее влияние на характер движения системы. Переходная характеристика становится колебательной, причем перерегулирование и частота колебаний с ростом k_p увеличиваются. Время регулирования сначала уменьшается и достигает минимума при $k_p \approx 0{,}3$, когда колебания системы укладываются в 5% коридор. В дальнейшем время регулирования вновь увеличивается из-за возрастающей колебательности системы.

При $k_p = 3$ полюса p_2 и p_3 становятся чисто мнимыми, равными $\pm 3j$, и система выходит на границу устойчивости. Ее переходная характеристика принимает вид незатухающих колебаний с угловой частотой $\omega = 3$ рад/сек.

Дальнейшее увеличение коэффициента передачи делает систему неустойчивой. При этом ее поведение определяется в основном парой правых комплексно-сопряженных корней p_2 и p_3.

Приведенный пример позволяет сделать выводы, которые, хотя и являются частными, тем не менее, справедливы для большинства САР.

Увеличение коэффициента усиления разомкнутой системы, вызванное увеличением коэффициента передачи регулятора, приводит к тому, что все возрастающую роль при формировании переходной характеристики играет пара комплексно-сопряженных полюсов системы, наиболее близкая к мнимой оси. Вещественная часть полюсов этой пары уменьшается по модулю, а мнимая, наоборот, возрастает. Как следствие, увеличиваются частота колебаний и перерегулирование переходной характеристики, т.е. повышается колебательность системы. Время переходного процесса сначала уменьшается до определенного значения, потом вновь увеличивается. Максимальное быстродействие наблюдается для процессов с небольшим перерегулированием.

При определенном (критическом) значении коэффициента усиления два полюса системы становятся чисто мнимыми, и система выходит на границу устойчивости. Переходные процессы имеют вид расходящихся колебаний. Дальнейшее увеличение коэффициента делает систему неустойчивой.

Третьей причиной неустойчивости может быть «неправильная» структура системы. Некоторые, т.н. *структурно неустойчивые*, системы невозможно сделать устойчивыми никакими изменениями коэффициентов.

Пусть имеется объект регулирования с передаточной функцией

$$W_{об} = \frac{k_{об}}{p(T_{об}p + 1)}.$$

На объект оказывает воздействие исполнительный механизм с передаточной функцией

$$W_{им} = \frac{k_{им}}{p}.$$

Требуется построить замкнутую систему регулирования.

Самым простым решением видится применение усилителя с дифференциальным входом, который будет управлять исполнительным механизмом с помощью усиленного сигнала разности между заданным и фактическим значениями выходной величины (рис. 2.60).

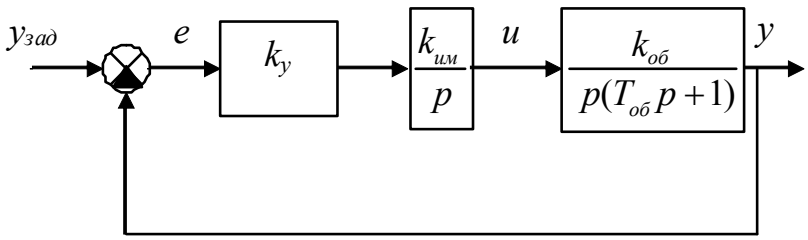


Рис. 2.60. Структурная схема системы.

Построенный таким образом регулятор, включающий усилитель и исполнительный механизм, реализует интегральный закон регулирования – управляющее воздействие пропорционально интегралу ошибки.

Передаточная функция замкнутого контура:

$$W_{зам}(p)=\frac{W_{раз}(p)}{1+W_{раз}(p)}=\frac{k_y k_{им} k_{об}}{T_{об}p^3+p^2+k_y k_{им} k_{об}}.$$

Передаточная функция имеет одну «странность»: в знаменателе отсутствует слагаемое с оператором p в первой степени. Это дает нам право вынести «вердикт» без каких-либо дополнительных исследований: замкнутая система неустойчива! Объяснение тут очень простое.

Любой полином может быть разложен на произведение элементарных множителей, соответствующих его корням. Если система устойчива, все корни – либо отрицательные вещественные, либо комплексно сопряженные пары с отрицательной вещественной частью. Поэтому множителей может быть только два типа:

$(p+\alpha_i)$ – для вещественных корней $p_i=-\alpha_i$ и

$(p+\alpha_i+j\beta_i)\,(p+\alpha_i-j\beta_i)=(p^2+2\alpha_i p+\alpha_i^2+\beta_i^2)$ – для комплексно сопряженных пар $p_{i,i+1}=-\alpha_i\pm j\beta_i$.

Таким образом, характеристический полином устойчивой системы можно представить произведением «скобок», внутри которых присутствуют все необходимые степени оператора p (первая и нулевая или вторая, первая и нулевая). Кроме того, все коэффициенты в «скобках» – положительные числа. Абсолютно невозможно, чтобы при этом в «итоговом» полиноме, который является произведением «скобок», может «пропасть» какая-либо степень оператора p.

Степень может «пропасть» только в результате сокращение слагаемых, что возможно, если в скобках стоят «вперемежку» и положительные и отрицательные числа. Но тогда система точно неустойчива, поскольку в «скобках» устойчивой системы никаких отрицательных чисел быть не может.

На практике выводы из рассмотрения примера звучат так: *интегральный регулятор для объекта без самовыравнивания неприменим.*

В теории они «зафиксированы» как *критерий Стодолы*:

Для устойчивости системы необходимо (но недостаточно), чтобы все коэффициенты характеристического полинома были отличны от нуля и имели один и тот же знак.

В критерии, правда, не говорится, что все коэффициенты должны быть положительны, и это правильно, так как полиномы, отличающиеся только зна-

ками всех своих слагаемых, очевидно, имеют одинаковые корни (минус единица, как множитель, ни на что не влияет). Кроме того, мы всегда можем, не изменяя передаточной функции, изменить одновременно знаки числителя и знаменателя и из любого отрицательного полинома получить положительный и наоборот.

Важно отметить, что критерий носит только необходимый характер: если его условия соблюдаются, система необязательно устойчива. Действительно, положительные коэффициенты полинома вполне могут «сложиться», даже если в части скобок присутствуют отрицательные числа. Однако для полиномов первого и второго порядка критерия Стодолы и вполне *достаточно*. Системы третьего порядка уже требуют «дополнительной проверки».

Разработан, конечно, ряд критериев, имеющих не только необходимый, но и достаточный характер. Это алгебраические критерии Раусса, Гурвица, и частотные критерии Михайлова, Найквиста [8]. Они позволяют без вычисления корней путем некоторых манипуляций с коэффициентами полинома или частотными характеристиками системы определить, устойчива она или нет, как «далеко» она находится от границы устойчивости и т.д.

Разработка этих критериев была продиктована тем обстоятельством, что в прошлом определить корни полинома порядка выше третьего было проблематично. И хотя в наше время с помощью компьютера и численных методов мы можем вычислить корни полинома любой степени, многие критерии остаются актуальными, поскольку положены в основу методов анализа и синтеза систем.

Последним вопросом, который мы разберем в этом пункте, будет вопрос о влиянии числителя передаточной функции на поведение системы.

Как мы выяснили выше, на устойчивость числитель не влияет, но это не означает, конечно, что он не влияет вообще ни на что. Пусть у нас имеется система с передаточной функцией

$$W(p)=\frac{y}{u}=\frac{-5p+1}{p^2+3p+2}.$$

Система устойчива, так как полюса передаточной функции – отрицательные вещественные числа: –2 и –3 (см. пример выше). Однако минус в полиноме числителя сразу «бросается в глаза» и «вызывает подозрения». С помощью Matlab построим переходную характеристику системы (рис. 2.61):

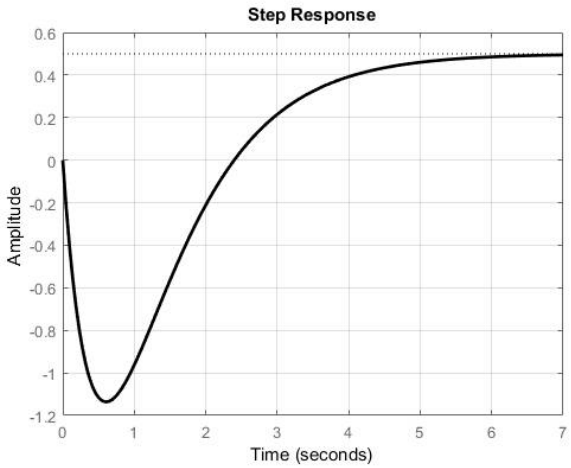


Рис. 2.61. Переходная характеристика системы

```
W = tf([-5,1],[1,3,2]);
step(W)
grid
```

Подозрения подтверждаются: система ведет себя, мягко говоря, не очень хорошо. На начальном участке характеристики имеется большой «провал

вниз». Потом система все-таки из этого «провала» выбирается и благополучно приходит к положенному ей согласно коэффициенту передачи значению 0,5. Но зачем она выбрала такой «извилистый» путь к цели?

Очевидно, что дело в отрицательном элементе числителя передаточной функции. Дифференциальное уравнение системы

$$\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 2y = -5\frac{du}{dt} + u$$

показывает нам, что в ней действуют два противоположно направленных «фактора»: один (пусть и временно) «тянет» систему «вниз», а другой – «вверх».

В общем случае все определяет не знаки коэффициентов числителя, а его корни, которые называются *нулями*. Если имеются положительные вещественные нули или комплексно-сопряженные нули с положительной вещественной частью, от системы можно ожидать подобные «странности» поведения (в нашем случае у системы один нуль, равный 0,2).

Такие системы называются устойчивыми *неминимально-фазовыми* системами.

Минимально-фазовая система – это система с однозначной связью между амплитудно-частотной и фазочастотной характеристиками. Она имеет минимальный (по модулю) фазовый сдвиг среди всех систем с одинаковыми АЧХ. Отсюда и такое ее название.

Доказано, что минимально-фазовыми являются системы, нули и полюса передаточной функции которых отрицательные вещественные числа или комплексно-сопряженные числа с отрицательной вещественной частью. Таким образом, неустойчивые системы всегда неминимально-фазовые. Однако могут быть и устойчивые неминимально-фазовые системы, в чем мы только что убедились.

Рассмотрим физический пример устойчивого неминимально-фазового объекта. Пусть опять это будет наш любимый объект регулирования уровня. Что будет, если мы увеличили подачу воды в бак? Уровень, безусловно, начнет расти! Оказывается, не всегда. Закроем бак «плотной крышкой» и «поставим на огонь». По одной трубке будем подавать в него холодную воду под давлением, а через другую, врезанную в верхней части бака, будет выходить пар. Допустим, мы наполнили бак водой до некоторой отметки и довели воду до кипения. Далее нам удалось стабилизировать процесс, подобрав такое положение регулирующего крана на подаче, что уровень остается неизменным, поскольку объемный расход приходящей воды равен объемному расходу уходящего пара. И тут нам захотелось вдруг резко увеличить подачу воды. Как ни странно, уровень сначала пойдет вниз и только спустя некоторое время начнет расти. Объясняется это просто. Уровень в баке зависит не только от количества воды в нем, но и от количества пара, растворенного в воде, причем чем больше температура, тем больше пара растворено в воде. Увеличивая подачу холодной воды, мы остужаем смесь и снижаем интенсивность парообразования, что приводит к временному падению уровня. Примерно такой же эффект наблюдал каждый, кто снимал крышку с кастрюли, из которой начал «убегать» при закипании суп.

На самом деле, чтобы описанный процесс имел место, первоначальное паросодержание воды должно быть достаточно высоким, т.е. парообразование должно быть очень интенсивным. Кроме того, многое зависит еще и от давления в баке. В случае с супом, уровень падает именно по причине резкого снижения давления, а не температуры.

Управление неминимально-фазовым объектом, даже если он устойчив, конечно же, вызывает определенные трудности. Представьте себе, что Вы поворачиваете рулевое колесо вправо, а автомобиль сначала отклоняется влево, и только потом начинает поворачивать вправо. Научиться управлять таким автомобилем довольно сложно.

2.2.2.9. Точность

Система автоматического регулирования должна *выполнять задание*. Если она устойчива, то она его, наверное, *как-то* выполняет (что, в общем, не совсем верно, как мы увидим далее). Как должно выполняться задание? Ответ очевиден: *быстро* и *точно*. Скорость выполнения задания можно оценить по переходным и частотным характеристикам, рассмотренным в п. 2.2.2.6. В принципе, по ним же можно судить и о точности, но это не совсем удобно.

Если мы имеем дело с *системой стабилизации*, задача которой – точно поддерживать заданное значение регулируемой величины, противодействуя возмущениям, то для нас важным является качество *статических* или *установившихся* режимов, т.е. таких режимов, в которых переменные системы не изменяются во времени (хотя, справедливости ради, следует сказать, что это важно для любых систем регулирования, поскольку все они время от времени пребывают в этих режимах).

Качество САР в статике оценивается величиной *статической ошибки* – разности между заданным значением выходной величины и ее действительным значением в установившемся режиме при постоянных внешних воздействиях.

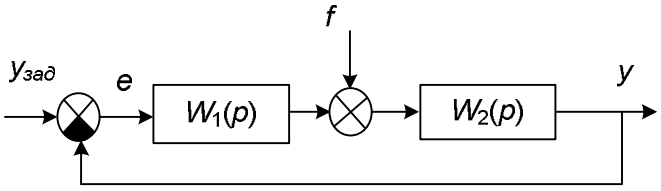


Рис. 2.62. Статическая система.

На рис. 2.62 показана типичная структура замкнутой системы автоматического регулирования с одним возмущением, приложенным где-то «посередине» прямого канала.

Ошибка регулирования может быть выражена через задающее и возмущающее воздействия следующим образом:

$$e = \frac{1}{1+W_1(p)W_2(p)}\,y_{зад} + \frac{W_2(p)}{1+W_1(p)W_2(p)}\,f.$$

Если $W_1(p)$ и $W_2(p)$ не имеют нулевых полюсов (интеграторов, не охваченных обратными связями), передаточные функции в статике можно заменить коэффициентами передачи.

Коэффициент передачи, напомним, – это то, что останется от передаточной функции при подстановке в нее $p = 0$. При наличии интеграторов знаменатель передаточной функции «заканчивается» на p, и мы формально получим «бесконечность». Этот вариант будет рассмотрен ниже.

На рис. 2.63 показана расчетная структура.

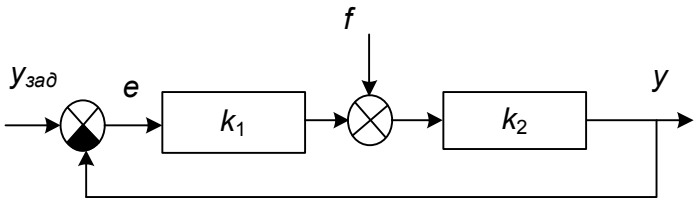


Рис. 2.63. К расчету статической ошибки в статической системе.

Теперь ошибку регулирования можно определить через коэффициенты передачи:

$$e = \frac{1}{1+k_1k_2}\,y_{зад} + \frac{k_2}{1+k_1k_2}\,f.$$

Системы без свободных интеграторов в прямом канале называются *статическими системами*.

Анализируя полученное выражение, приходим к выводу, что в статических системах точность повторения постоянного задания будет тем выше, чем больше общий коэффициент передачи прямого канала k_1k_2. Эффективность подавления возмущения будет тем выше, чем больше коэффициент передачи звеньев прямого канала, находящихся «перед возмущением», т.е. k_1. Коэффициент k_2 здесь играет небольшую роль, так как произведение k_1k_2 обычно намного больше единицы и этот коэффициент «как бы сокращается».

Таким образом, чтобы повысить точность, нам нужно повышать коэффициент k_1, т.е., по сути, коэффициент передачи регулятора. Однако до бесконечности повышать его нельзя, т.к. при больших значениях коэффициента система становится недопустимо колебательной, а потом и теряет устойчивость (мы это видели в предыдущем пункте). Если, увеличивая коэффициент регулятора, добиться требуемой точности, сохраняя при этом приемлемую динамику, не удается, приходится изменять закон регулирования.

Когда прямой канал системы включает в себя хотя бы один интегратор, статическая ошибка по заданию будет равна нулю. Такая система называется *астатической.*

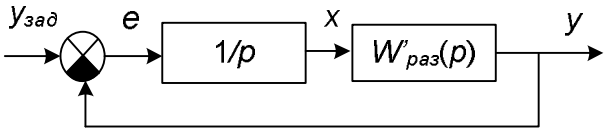


Рис. 2.64. К расчету статической ошибки в астатической системе.

При отсутствии возмущения структуру астатической системы можно представить в виде, показанном на рис. 2.64.

В статике все переменные системы неизменны, поэтому выходной сигнал интегратора неизменен: x = const. Зная, как работает интегратор, мы можем с уверенностью утверждать, что это возможно только, если $e = 0$.

Коэффициент передачи такой системы по каналу $y_{зад} \to y$ равен единице:

$$W_{раз}(p) = \frac{1}{p}W'_{раз}(p) = \frac{p^m + \ldots + b_0}{p^n + \ldots + a_1p},$$

$$W_{зам}(p) = \frac{W_{раз}(p)}{1+W_{раз}(p)} = \frac{\dfrac{p^m + \ldots + b_0}{p^n + \ldots + a_1p}}{1+\dfrac{p^m + \ldots + b_0}{p^n + \ldots + a_1p}} = \frac{p^m + \ldots + b_0}{p^n + \ldots + b_0}.$$

Ошибка по возмущению зависит от того, в какой части системы относительно места приложения возмущения находится интегратор.

Так, если интегрирование происходит «после» возмущения (например, когда объект не обладает самовыравниванием) (рис. 2.65), ошибка будет иметь место.

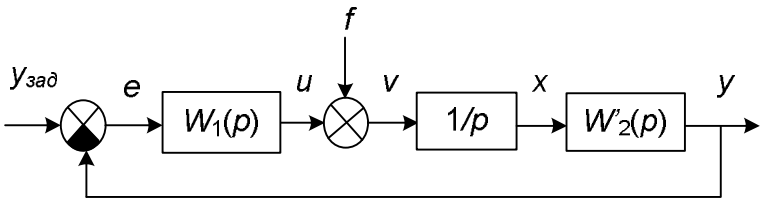


Рис. 2.65. К расчету статической ошибки по возмущению в астатической системе – возмущение «до» интегратора.

Действительно, в статике x = const, следовательно, $v = 0$, откуда $u = -f$, $e = u/k_1 = -f/k_1$.

Если, наконец, $W_1(p)$ имеет нулевые полюса, статическая ошибка равна нулю независимо от того, имеются ли нулевые полюса в $W_2(p)$ или нет (рис. 2.66).

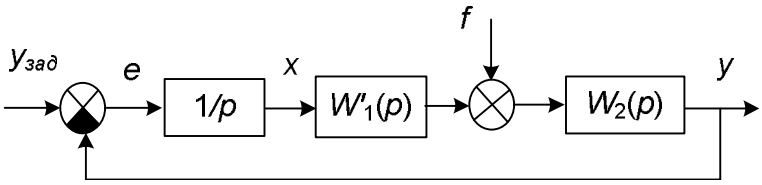


Рис. 2.66. К расчету статической ошибки по возмущению в астатической системе – возмущение «после» интегратора.

Действительно, в статике, когда x = const, $e = 0$ независимо от вида передаточной функции $W_2(p)$.

Таким образом, *условием отсутствия установившейся ошибки является наличие нулевых полюсов («свободных интеграторов») в звеньях системы, расположенных в прямом канале до места приложения возмущения.*

Фактически это означает необходимость введения в закон регулирования интегральной составляющей.

Для *систем воспроизведения* основной задачей является точное воспроизведение *изменяющегося* задания. Точность воспроизведения может быть оценена по частотным характеристикам системы, поскольку они дают возможность определить диапазон частот, на котором система передает сигнал без существенного уменьшения амплитуды и больших фазовых сдвигов.

На практике часто используют другой подход, рассматривая реакцию системы на типовые воздействия, изменяющиеся с постоянной производной: первой (скоростью) или второй (ускорением). Качество системы в таких режимах характеризуется величиной *динамической ошибки* в установившемся динамическом режиме (по затуханию свободного движения).

Наличие и величина ошибки зависят от *порядка воздействия* (производной) и *порядка астатизма* системы (числа интегрирующих звеньев прямого канала).

При $y_{зад} = a$ = const мы имеем нулевой порядок воздействия (по существу, – это статика), при $y_{зад} = vt$ – первый порядок, при $y_{зад} = at^2/2$ – второй и т.д.

Рассмотрим поведение системы под воздействием линейно изменяющегося задания.

1) пусть порядок астатизма системы равен нулю (рис. 2.67).

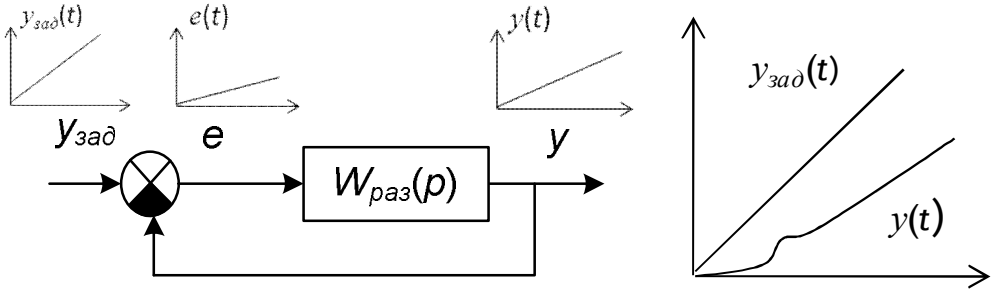


Рис. 2.67. К расчету динамической ошибки в статической системе.

На рисунке показано, как изменяются сигналы во времени. Откуда нам это известно? Включим здравый смысл и порассуждаем.

Предположим, что ошибка равна нулю, но тогда и на выходе был бы нуль, поскольку $W_{раз}(p)$ в стационарном режиме обращается в $k_{раз}$. Но $y_{зад}$ постоянно увеличивается, следовательно, и ошибка в таком случае должна постоянно расти.

Далее, допустим, ошибка – постоянная величина, но тогда, аналогично, и на выходе была бы постоянная величина, и опять ничего не сходится.

Остается предположить, что ошибка линейно растет, также себя будет вести и выходная величина. Но скорость роста выходной величины меньше скорости роста задания, поэтому и ошибка постоянно растет. Все сошлось.

Определим скорость роста ошибки. Передаточная функция без интеграторов преобразует линейно изменяющийся сигнал точно так же, как постоянный сигнал, т.е. «помножает» скорость изменения входного сигнала на коэффициент передачи:

$$\frac{dy}{dt} = k_{раз}\frac{de}{dt},$$

но

$$\frac{de}{dt} = \frac{dy_{зад}}{dt} - \frac{dy}{dt},$$

следовательно,

$$\frac{de}{dt} = \frac{dy_{зад}}{dt} - k_{раз}\frac{de}{dt},$$

откуда

$$\frac{de}{dt} = \frac{1}{1+k_{раз}}\frac{dy_{зад}}{dt}.$$

Таким образом, чем больше коэффициент передачи разомкнутой системы, тем медленнее растет ошибка. Но нам-то хотелось, наверное, чтобы ошибка не росла вовсе. В этом смысле рассмотренный вариант системы не справляется с задачей регулирования.

Если порядок воздействия больше порядка астатизма, ошибка неограниченно возрастает.

2) порядок астатизма системы равен единице (рис. 2.68).

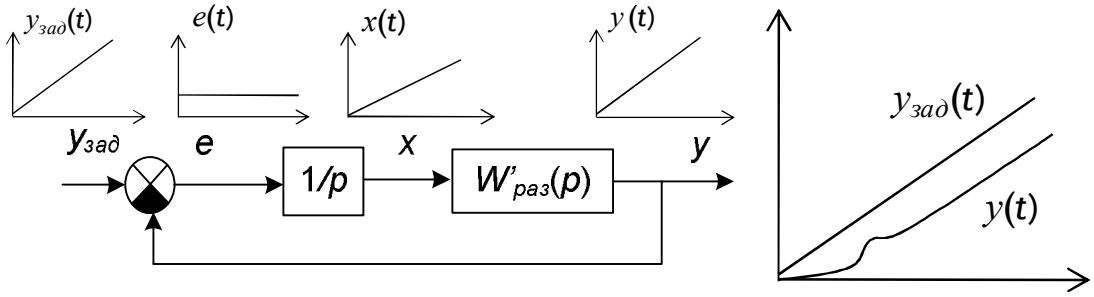


Рис. 2.68. К расчету динамической ошибки в астатической системе с астатизмом первого порядка.

Если предположить, что ошибка равна нулю, тогда на выходе интегратора должна быть постоянная величина. Следовательно, постоянная величина – и на выходе системы. Но тогда ошибка никак не может равняться нулю. Ничего не сходится.

Но если ошибка – постоянная величина, отличная от нуля, сигнал на выходе интегратора линейно возрастает, то же самое происходит и на выходе системы, и все сходится: разность двух сигналов (задания и выходной величины), изменяющихся с одинаковой скоростью действительно будет постоянной величиной.

Таким образом, если порядок воздействия равен порядку астатизма, установившаяся ошибка постоянна.

Значение ошибки нетрудно определить: если скорость изменения задания равна v, скорость изменения выходного сигнала также равна v, скорость изменения сигнала x и ошибка e равны $v/k'_{раз}$, где $k'_{раз} = W'_{раз}(0)$ – коэффициент передачи $W'_{раз}(p)$. Чем больше коэффициент $k'_{раз}$, тем меньше ошибка.

Такая система вполне работоспособна, так как выполняет задание, пусть и не совсем точно.

3) порядок астатизма системы равен двум (рис. 2.69).

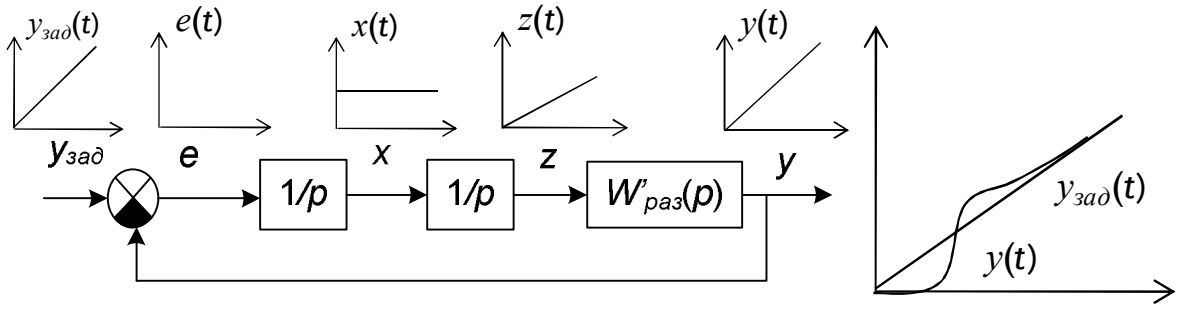


Рис. 2.69. К расчету динамической ошибки в астатической системе с астатизмом второго порядка.

В этом случае ошибка действительно равна нулю, сигнал на выходе первого интегратора постоянный, сигнал на выходе второго интегратора – линейно возрастает, как и сигнал на выходе системы.

Если порядок воздействия меньше порядка астатизма, установившаяся ошибка воспроизведения равна нулю.

Это, конечно, идеальный случай, который полностью демонстрирует нам «бесценную» роль интегратора: он способен менять порядок сигнала и формировать выходной сигнал нужной скорости.

Разумеется, все вышесказанное имеет какой-то смысл, если система устойчива. К сожалению, сами по себе интеграторы плохо влияют на устойчивость, см. п. 2.2.2.8. Поэтому, если мы хотим получить точную систему, нам придется не только «поставить» интеграторы, то и предусмотреть компенсацию их негативного влияния.

2.2.3. Как построить регулятор

2.2.3.1. Постановка задачи

Синтез системы состоит в выборе и настройке технических средств регулирования: первичных и вторичных измерительных преобразователей, контроллеров, усилителей, силовых преобразователей, исполнительных механизмов и т.д.

«Технический» этап синтеза, т.е. собственно выбор технических средств, проводится исходя из требований, предъявляемых технологическим процессом и других, в частности экономических, требований.

На «математическом» этапе синтеза рассматривается модель системы, и определяются структура и параметры закона регулирования.

Обычно «технический» этап предшествует «математическому», т.е. закон регулирования реализуется на базе уже имеющихся технических средств. Однако в особо ответственных случаях возможен и обратный порядок синтеза, когда сначала определяется закон регулирования, а потом «под него» выбирается техническое обеспечение.

Для определения закона регулирования необходимо составить расчетную схему САР, в которой четко отделены друг от друга ее *неизменяемая* и *изменяемая* части.

К неизменяемой части, кроме объекта регулирования, относят все те технические средства, параметры которых не подлежат изменению. Эту часть в процедурах синтеза принято называть просто «объектом». Таким образом, «математический» объект следует отличать от «технического» объекта. В него, помимо математического описания собственно объекта регулирования, может входить математическое описание измерительных преобразователей, исполнительных механизмов и других средств.

Изменяемая часть системы называется регулятором, или, иногда, *корректирующим устройством*.

Кроме алгоритмической схемы контроллера, в ее реализации могут участвовать и другие элементы САР. Часто, например, закон регулирования формируется совместно контроллером и исполнительным механизмом. Структура типичной системы регулирования по отклонению показана на рис. 2.70.

«С точки зрения математики» показанная выше схема должна быть разделена на «регулятор» и «объект». Возможные варианты такого разделения показаны на рис. 2.71.

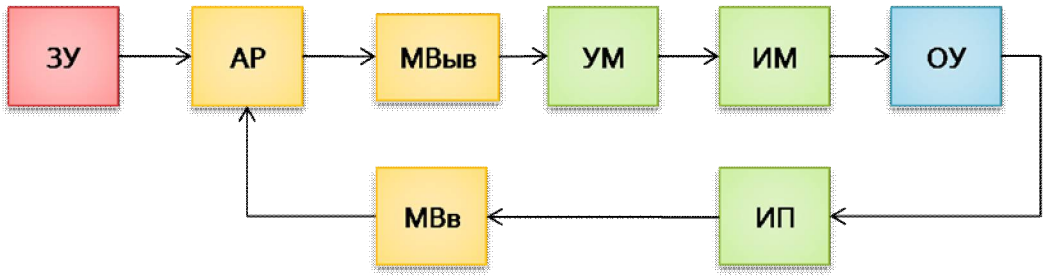


Рис. 2.70. Структура системы автоматического регулирования:
ЗУ – задающее устройство; АР – алгоритм регулирования; МВв, МВыв – модули ввода-вывода; УМ – усилитель мощности; ИМ – исполнительный механизм; ИП – измерительный преобразователь; ОУ – объект управления.

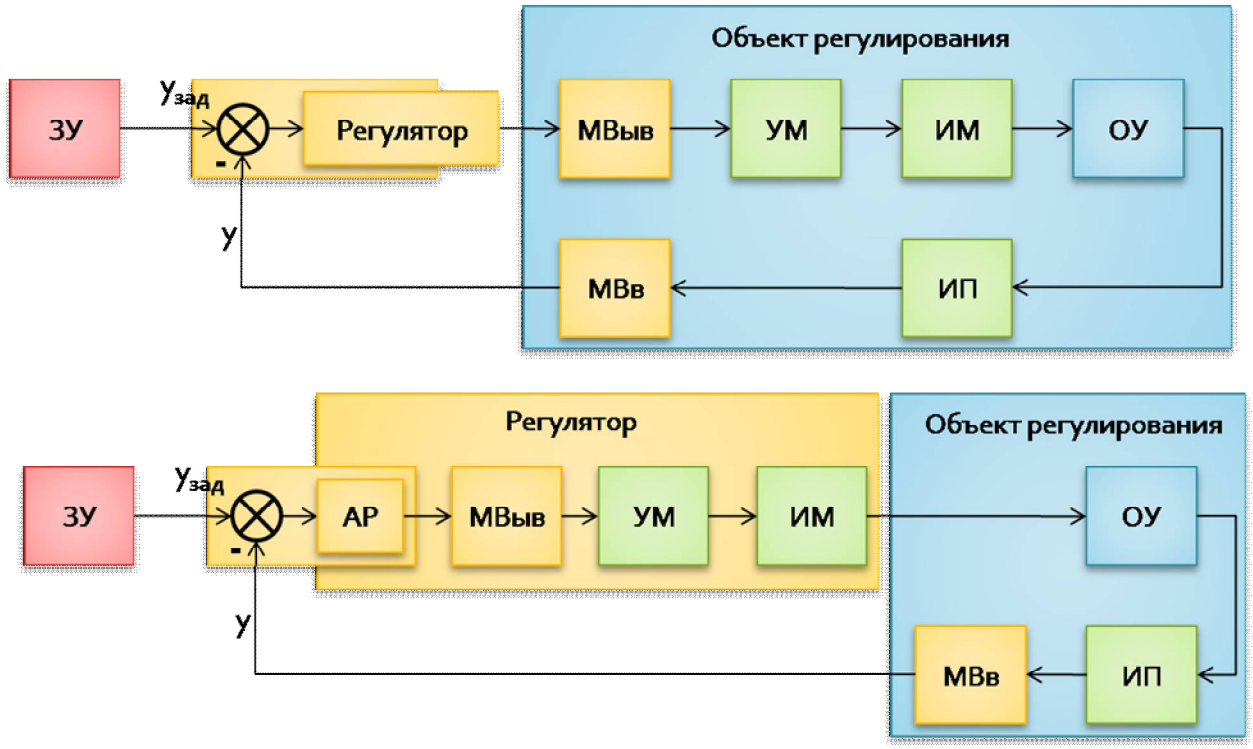


Рис. 2.71. Варианты разделения элементов системы.

Таким образом, в простейшем случае расчетная схема САР включает всего два элемента: *объект* и *регулятор*, рис. 2.72.

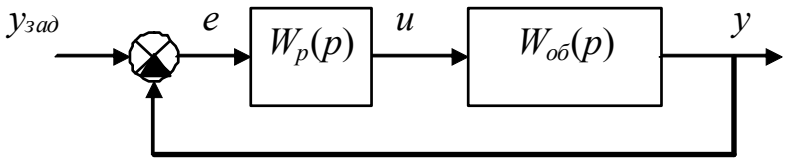


Рис. 2.72. Математическая (расчетная) структура простейшей одноконтурной системы.

Задача синтеза простейшей линейной системы автоматического регулирования может быть сформулирована следующим образом: для объекта с передаточной функцией $W_{об}(p)$ определить передаточную функцию регулятора $W_p(p)$ – такую, чтобы обеспечивалось требуемое (либо наилучшее при заданных ограничениях) качество САР.

Различают задачи *параметрического* и *структурно-параметрического* синтеза системы. В первом случае структура передаточной функции $W_p(p)$ известна и необходимо найти только ее параметры, во втором требуется определить и структуру, и параметры $W_p(p)$.

2.2.3.2. «Теоретические» методы синтеза регуляторов

«Теоретическими» эти методы названы не потому, что они не применяются на практике. Наоборот, сейчас, во время цифровой обработки сигналов, никаких ограничений по их применению нет. Применялись они и раньше, в «аналоговую эпоху», но в ограниченном количестве, поскольку техническая реализация каждого такого регулятора требовала разработки специальных электронных схем, что для повсеместного промышленного применения было дорого.

Существуют так называемые *общие* подходы структурно-параметрического синтеза регуляторов в классе одномерных линейных непрерывных систем. Построенные с помощью них системы в некотором смысле *идеальны*, так как обладают свойствами *эталона*.

В большинстве случаев задается *эталонная передаточная функция* замкнутой системы. Она может быть построена различными методами.

Часто эталонную передаточную функцию строят в *классе низкочастотных фильтров*. Считается, что «хорошая» замкнутая система, имеющая необходимую точность в стационарных режимах и заданное качество переходных процессов, должна быть, как правило, близка по своим свойствам идеальному низкочастотному фильтру с АЧХ вида (рис. 2.73).

Рис. 2.73. АЧХ идеального низкочастотного фильтра.

Система идеально воспроизводит сигналы с частотами до ω_0 и полностью подавляет сигналы большей частоты, которые «считаются» шумами (помехами).

Передаточные функции, приближенно реализующие идеальный низкочастотный фильтр, строятся на основе *полиномов Баттерворта*, *Чебышева*, *Лежандра* [9].

Общий вид эталонной передаточной функции:

$$W_{эт}(p) = \frac{\omega_0^n}{p^n + a_1\omega_0 p^{n-1} + a_2\omega_0^2 p^{n-2} + \ldots + a_{n-1}\omega_0^{n-1} p + \omega_0^n} = \frac{\omega_0^n}{D(p)}.$$

где $a_1 - a_{n\text{-}1}$ – коэффициенты полинома, определяющие «качество» протекания процессов, ω_0 – «среднегеометрический корень», параметр, определяющий

время протекания процессов. Чем больше ω_0, тем больше быстродействие. Продемонстрируем это с помощью небольшой программы Matlab:

```
w0 = 1;
W1 = tf(w0^3,[1 2*w0 2*w0^2 w0^3])
W1 =

           1
  --------------------
  s^3 + 2 s^2 + 2 s + 1

Continuous-time transfer function.
w0 = 3;
W2 = tf(w0^3,[1 2*w0 2*w0^2 w0^3])
W2 =

           27
  ----------------------
  s^3 + 6 s^2 + 18 s + 27

Continuous-time transfer function.
step(W1,W2)
```

Программа строит переходные характеристики двух систем, передаточные функции которых отличаются только параметром ω_0. Результат работы программы показан на рис. 2.74. Более быстрый процесс соответствует $\omega_0 = 3$.

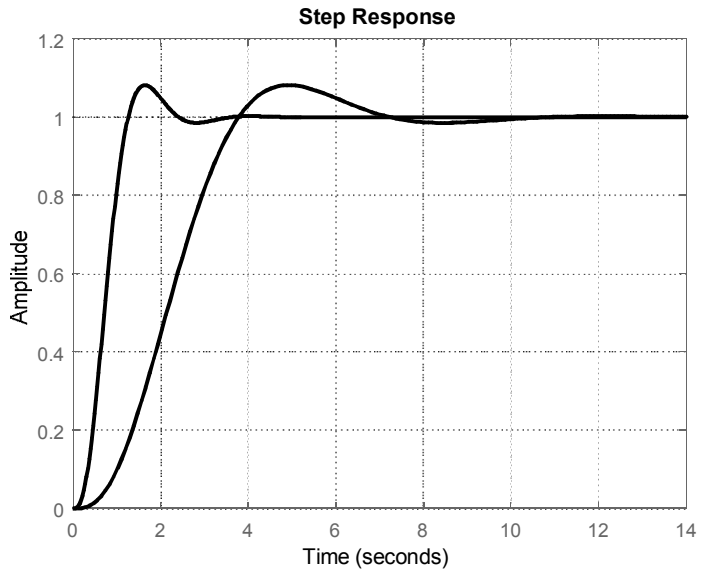


Рис. 2.74. Переходные характеристики систем с различными значениями ω_0.

Длительность второго процесса в три раза больше. В остальном характеристики не отличаются: если вторую сжать в три раза вдоль временной оси, она «сольется» с первой.

Как известно, динамические свойства линейной системы в основном определяются распределением корней ее характеристического полинома (знаменателя передаточной функции) на комплексной плоскости.

Баттервортом доказано, наилучшее приближение к АЧХ идеального низкочастотного фильтра будет получено, если корни характеристического полинома $D(p)$ лежат на полуокружности радиусом ω_0 (рис. 2.75).

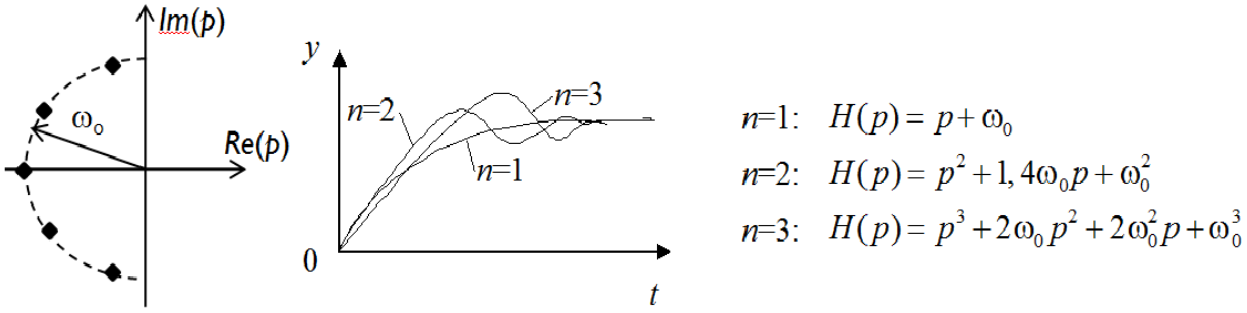


Рис. 2.75. Распределение корней по Баттерворту.

Переходные процессы при этом имеют колебательный характер, при этом с увеличением порядка колебательность возрастает.

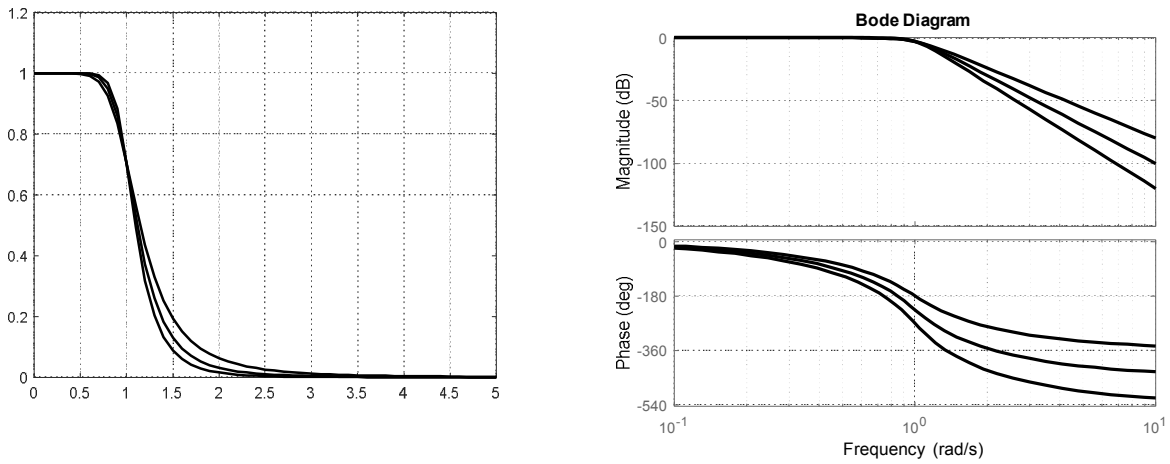
Программа, приведенная ниже, строит частотные и переходные характеристики систем 4, 5 и 6 порядка с распределением корней по Баттерворту при $\omega_0 = 1$.

```
[B4,A4] = butter(4,1,'s'); W4 = tf(B4,A4)
W4 =

                    1
  ----------------------------------------
  s^4 + 2.613 s^3 + 3.414 s^2 + 2.613 s + 1

Continuous-time transfer function.

[B5,A5] = butter(5,1,'s'); W5 = tf(B5,A5)
W5 =

                          1
  -----------------------------------------------------
  s^5 + 3.236 s^4 + 5.236 s^3 + 5.236 s^2 + 3.236 s + 1

Continuous-time transfer function.

[B6,A6] = butter(6,1,'s'); W6 = tf(B6,A6)
W6 =

                                1
  -----------------------------------------------------------------
  s^6 + 3.864 s^5 + 7.464 s^4 + 9.142 s^3 + 7.464 s^2 + 3.864 s + 1

Continuous-time transfer function.

figure(1), bode(W4,W5,W6)
figure(2), step(W4,W5,W6)
w = 0:.1:5;
[A4,ph4] =bode(W4,w);
[A5,ph5] =bode(W5,w);
[A6,ph6] =bode(W6,w);
figure(3), plot(w,A4(:),w,A5(:),w,A6(:)), grid
```

Результаты работы программы показаны на рис. 2.76 и 2.77.

На рис. 2.76 верхние кривые описывают систему четвертого порядка, нижние – шестого (посередине, конечно, – пятого). Видно, что чем выше порядок системы, тем ближе ее частотные характеристики к характеристикам иде-

ального фильтра. Однако с увеличением порядка уменьшается быстродействие, а колебательность растет, см. рис. 2.77: здесь самая «быстрая» кривая принадлежит системе четвертого порядка, а самая «медленная» – шестого.



а) АЧХ б) ЛАЧХ и ФЧХ

Рис. 2.76. Частотные характеристики систем 4,5 и 6 порядков с распределением корней по Баттерворту ($\omega_0 = 1$).

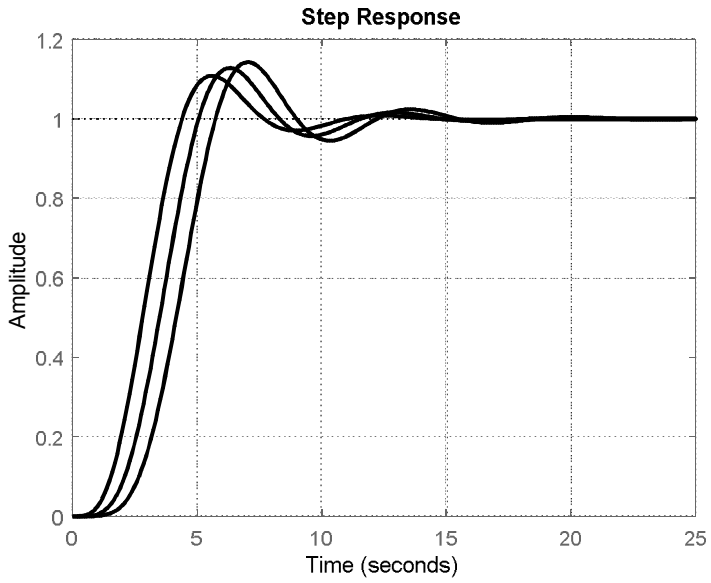


Рис. 2.77. Переходные характеристики систем 4,5 и 6 порядков с распределением корней по Баттерворту ($\omega_0 = 1$).

Если по каким-либо причинам колебательность нежелательна, может быть использовано другое распределение корней – *биноминальное*. В этом случае все корни выбираются действительными и одинаковыми, равными ω_0, а характеристический полином представляет собой не что иное, как бином Ньютона:

$$D(p) = (p + \omega_0)^n.$$

Доказано, что в этом случае мы получаем максимально быстрые апериодические (неколебательные) переходные процессы. С увеличением порядка

системы ее свойства также приближаются к свойствам идеального фильтра (хотя и значительно хуже, чем при распределении по Баттерворту), а быстродействие снижается.

Имеются и другие подходы к конструированию эталонных передаточных функций, в том числе и таких, у которых порядок числителя больше нуля [9]. Следует отметить, что какой бы подход не использовался, выбирать эталонную передаточную функцию произвольным образом, а точнее произвольно задавать через ω_0 быстродействие системы, нельзя. Ресурсы управления всегда ограничены. Нельзя заставить медленный объект реагировать быстро: для этого потребуется приложить к нему очень сильное воздействие. Такое, какое реальная аппаратура сформировать не сможет. В результате большую часть времени система будет работать на предельных уровнях управления, а это означает (в лучшем случае), что заложенные в нее законы регулирования не работают.

После того как эталонная передаточная функция системы получена, необходимо найти передаточную функцию регулятора. Как это ни странно, сделать это довольно легко.

Из эталонной передаточной функции замкнутой системы можно получить эталонную передаточную функцию разомкнутой системы:

$$W_{раз,эт}(p) = \frac{W_{зам,эт}(p)}{1 - W_{зам,эт}(p)}.$$

Эту передаточную функцию должны совместно «сформировать» регулятор и объект, т.е.

$$W_р(p) W_{об}(p) = W_{раз,эт}(p).$$

Следовательно,

$$W_р(p) = \frac{W_{раз,эт}(p)}{W_{об}(p)}.$$

Все очень просто и понятно, но в теории автоматического управления этот метод имеет гордое название: *метод динамической компенсации*.

На самом деле, конечно, выше изложена только идея этого метода. Здесь есть некоторое количество «подводных камней», которые мы рассмотрим пока без математических выкладок:

1) рассчитанный регулятор может оказаться достаточно сложным, т.е. иметь передаточную функцию высокого порядка. Однако если мы имеем достаточные «вычислительные мощности» и быстрый ввод-вывод, мы можем реализовать любую передаточную функцию, лишь бы она была физически реализуемой (порядок числителя не более порядка знаменателя и никакого отрицательного запаздывания, т.е. «предварения»);

2) передаточная функция регулятора физически нереализуема. Здесь возможны два варианта: либо мы исхитримся и реализуем регулятор «приближенно», неточно, либо его нужно пересчитать, взяв за основу другую эталонную передаточную функцию системы;

3) для некоторых объектов, оказывается, метод вообще неприменим. Причина прячется в самом слове «компенсация». Мы «затаскиваем» обратную динамику объекта в регулятор и там уже ее «компенсируем». По сути, речь идет о сокращении элементов числителя и знаменателя дроби. Но то, что прекрасно сокращается на бумаге и в нашей голове, совершенно необязательно сокращается на практике. Нам никогда не известна совершенно точная модель объекта, тем более, что его свойства подвержены изменениям во времени. В результате мы получаем, например, не 5/5 = 1, а 5/5,001 ≠ 1.

Можно, конечно, подумать: какая разница, ведь речь идет о каких-то малых величинах. Малые величины могут привести к большим проблемам. Представьте себе, что вы завозите в химическую лабораторию 10 тонн очень опасного вируса, а нейтрализуете только 9999, 99 кг. А 10 грамм вполне достаточно, чтобы через час покончить не только с сотрудниками лаборатории, но и с жителями города, в котором она расположена. А может быть и так: нейтрализованы все 10 тонн, а 10 грамм «антидота» осталось не использовано и проникло в атмосферу. А антидот – это тоже вирус, такой же опасный как исходный… То же самое и здесь: «плохой» объект из-за неполной компенсации полностью «испортит» и регулятор. И мы уже имеем два «плохих парня» в одной системе и, конечно, вести себя хорошо она не будет.

«Плохим» объектом является *неминимально-фазовый объект*, т.е. объект, у которого передаточная функция имеет «правые» нули или полюса. Попробуем доказать, что в этом случае система будет неустойчивой. Представим передаточную функцию объекта отношением двух полиномов:

$$W_{об}(p)=\frac{B_{об}(p)}{A_{об}(p)}.$$

Аналогично эталонная передаточная функция замкнутой системы:

$$W_{зам,эт}(p)=\frac{B_{эт}(p)}{A_{эт}(p)}.$$

Найдем передаточную функцию регулятора:

$$W_{р}(p)=\frac{W_{раз,эт}(p)}{W_{об}(p)}=\frac{1}{W_{об}(p)}\times\frac{W_{зам,эт}(p)}{1-W_{зам,эт}(p)}=$$

$$=\frac{A_{об}(p)}{B_{об}(p)}\times\frac{B_{эт}(p)/A_{эт}(p)}{1-B_{эт}(p)/A_{эт}(p)}=\frac{A_{об}(p)}{B_{об}(p)}\times\frac{B_{эт}(p)}{A_{эт}(p)-B_{эт}(p)}.$$

Передаточная функция разомкнутой системы:

$$W_{раз}(p)=W_{р}(p)W_{об}(p)=\frac{\left[A_{об}(p)B_{эт}(p)\right]}{\left[B_{об}(p)\left(A_{эт}(p)-B_{эт}(p)\right)\right]}\times\frac{B_{об}(p)}{A_{об}(p)}.$$

Здесь элементы, заключенные в квадратные скобки, относятся к регулятору, т.е. *заведомо реализуются неточно*.

Передаточная функция замкнутой системы:

$$W_{зам}(p)=\frac{W_{раз}(p)}{1+W_{раз}(p)}=$$

$$=\frac{\left[A_{об}(p)B_{эт}(p)\right]B_{об}(p)}{\left[B_{об}(p)\left(A_{эт}(p)-B_{эт}(p)\right)\right]A_{об}(p)+\left[A_{об}(p)B_{эт}(p)\right]B_{об}(p)}=$$

$$=\frac{\left[A_{об}(p)B_{эт}(p)\right]B_{об}(p)}{\left[B_{об}(p)A_{эт}(p)\right]A_{об}(p)-\left[B_{об}(p)B_{эт}(p)\right]A_{об}(p)+\left[A_{об}(p)B_{эт}(p)\right]B_{об}(p)}.$$

А теперь давайте сокращать. Сокращения у нас – двух типов: слагаемых и множителей. Если регулятор реализуется неточно, то вообще говоря, нельзя делать ни того, ни другого. Но, если принять во внимание степени полиномов, сокращение слагаемых в знаменателе $W_{зам}(p)$ ни к чему особо страшному не приведет. Действительно, степень сокращаемых полиномов меньше степени оставшегося, так как степени полиномов числителей $B(p)$ обычно меньше (во всяком случае, не больше) степеней полиномов знаменателей $A(p)$, поэтому в результате сокращения степень знаменателя не изменится, незначительно изменятся только коэффициенты полинома. Малые «вариации» коэффициентов не могут привести к существенным изменениям свойств (конечно, если только мы задали правильный «эталон»).

$$W_{зам}(p)\ \approx\frac{\left[A_{об}(p)B_{эт}(p)\right]B_{об}(p)}{\left[B_{об}(p)A_{эт}(p)\right]A_{об}(p)}.$$

Если сократить еще и множители числителя и знаменателя, мы получим то, к чему стремились, – «эталон»:

$$W_{зам}(p)\ \approx\frac{\left[B_{эт}(p)\right]}{\left[A_{эт}(p)\right]}.$$

Но сокращать таким образом выражение мы вообще не имеем права, поскольку любые, даже самые малые, «расхождения» в коэффициентах полиномов не позволят нам это сделать! Поэтому вернемся к предыдущей формуле. В ее знаменателе – произведение полиномов числителя и знаменателя объекта. Знаменатель «взят» прямо из объекта, а числитель – из регулятора. Если корни этих полиномов – правые, замкнутая система будет неустойчивой.

Пусть наш объект описывается передаточной функцией:

$$W_{об}(p)=\frac{-p+1}{p^2+p+1}.$$

Это устойчивый, но не минимально-фазовый объект.

Зададим эталон с биноминальным распределением полюсов:

$$W_{зам,эт}(p)=\frac{1}{p^2+2p+1}.$$

Передаточная функция эталонной разомкнутой системы:

$$W_{раз,эт}(p)=\frac{W_{зам,эт}(p)}{1-W_{зам,эт}(p)}=\frac{1}{p^2+2p}.$$

Передаточная функция регулятора:

$$W_p(p)=\frac{W_{раз,эт}(p)}{W_{об}(p)}=\frac{p^2+p+1}{-p^3-p^2+2p}.$$

Уже на этом месте можно и остановиться, так как мы получили неустойчивый регулятор. Однако посмотрим, как поведет себя замкнутая система, а точнее ее Simulink-модель (рис. 2.78).

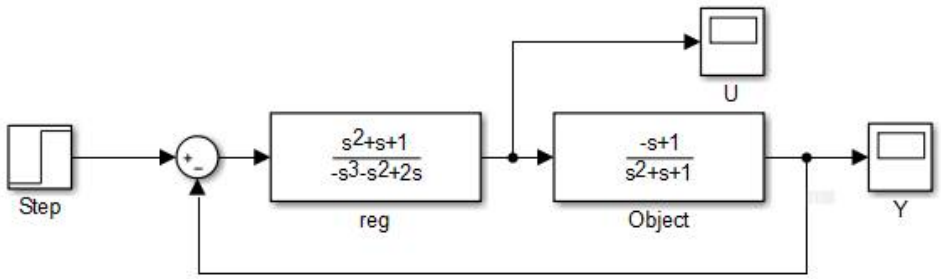


Рис. 2.78. Simulink-модель системы.

Графики изменения выходной величины *Y* и управляющего воздействия *U* при подаче на вход единичного ступенчатого воздействия показаны на рис. 2.79.

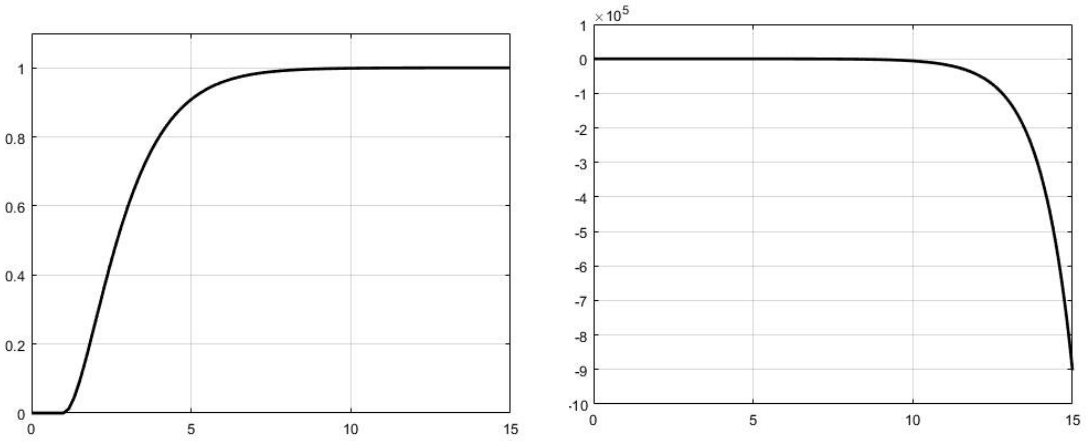


Рис. 2.79. Переходный процесс в системе: *Y*(*t*) и *U*(*t*).

Выходная величина ведет себя «прилично»: выполняет задание, демонстрируя эталонное поведение. А про управляющее воздействие ничего хорошего сказать нельзя.

Уже на 15-й секунде ее значение достигло громадной величины 9×10^5. Результаты вызывают большие подозрения. Каким образом при таком воздействии на объект его реакция остается вполне достойной? Здесь явно скрыт какой-то подвох.

Продолжим исследования и увеличим время расчета (рис. 2.80).

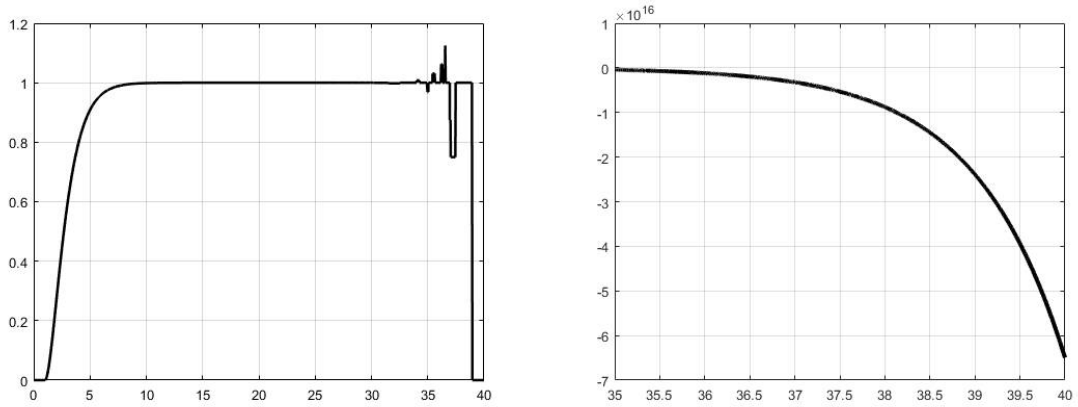


Рис. 2.80. Переходный процесс в системе: $Y(t)$ и $U(t)$ при увеличении расчетного интервала.

Примерно на 34-й секунде «полет перестал быть нормальным» и начались всякие «безобразия». Управление при этом достигло совершенно «астрономического» значения. Чтобы разобраться с этим, заглянем внутрь объекта, задав ее модель схемой на интеграторах (рис. 2.81).

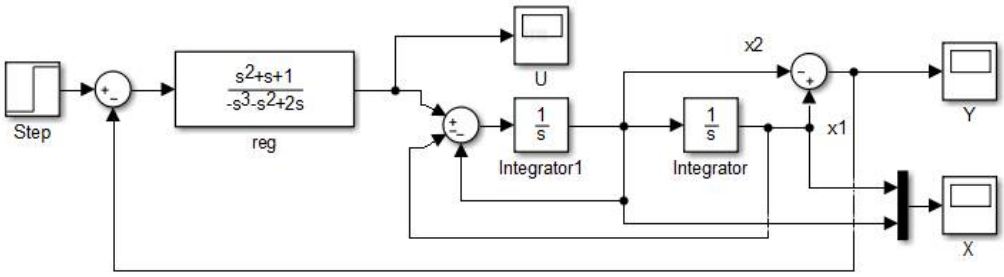


Рис. 2.81. Детализированная Simulink-модель системы.

Выходной сигнал объекта формируется как разность двух его координат: $Y = x_1 - x_2$. Если посмотреть, как ведут эти координаты в переходном процессе (рис. 2.82), то окажется, что они подобно управлению, по существу уходят в минус бесконечность, причем не каждая сама по себе, а «рука об руку». На левом графике кривые не различимы из-за масштаба, но увеличивая начальный участок (правый график), мы обнаружим, что разница между сигналами все-таки есть, что и так было понятно, поскольку разность координат и формирует выходной сигнал. Таким образом, мы имеем дело с «наглым обманом»: внутри системы происходят «чудовищные дела» при вполне «благовидном» внешнем поведении.

Что же тогда произошло на 34-й секунде процесса? Говоря по-простому, Simulink «устал врать». Simulink – принципиально *неточная* система, поскольку в его основе лежит численное (а значит, *приближенное*) интегрирование дифференциальных уравнений.

Точность, конечно, можно повысить, уменьшая шаг интегрирования, но это не меняет сути дела. Кроме того, Simulink оперирует переменными с плавающей точкой (double, 8 байт на число), которые принципиально не могут хранить точные значения очень больших чисел. При возрастании числа все большую часть переменной начинает занимать *порядок* числа (степень двойки), а все меньшую *мантисса* (значащая часть числа). При «обрезании» мантисса

теряет младшие разряды. В результате действия этих двух факторов (приближенность вычислений и неточность хранения значений) неминуемо наступает «хаос». Результату вычитания доверять становится совершенно невозможно.

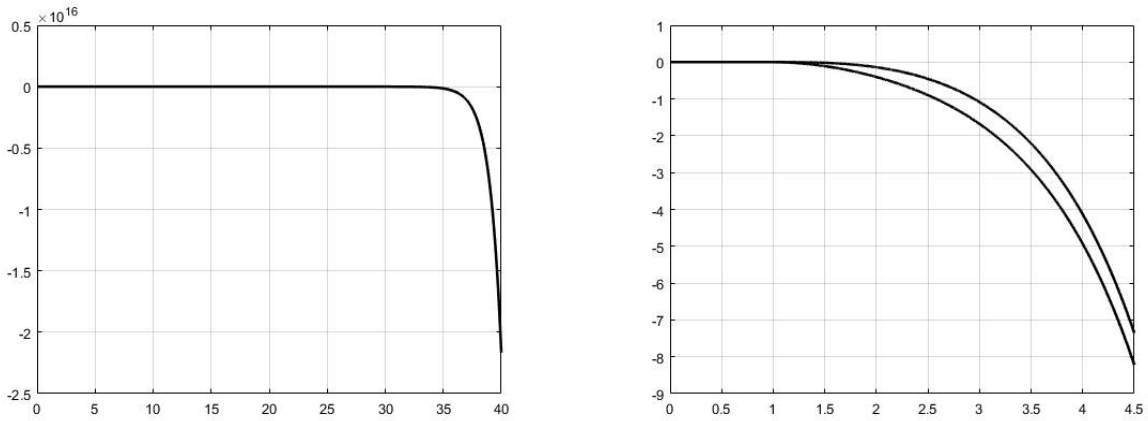


Рис. 2.82. Переходный процесс изменения внутренних координат объекта.

Реальная система, скорее всего, дожидаться 34-ой секунды не будет и «рухнет» сразу, так как реальный регулятор всегда реализован неточно. К тому же выходной сигнал реального регулятора всегда ограничен, поэтому долго сохранять иллюзию того, что «все в порядке» у него не получится.

Приведенный пример демонстрирует принципиальную невозможность *последовательной* компенсации плохих свойств объекта. Попытка сделать из «волка» «агнца» с помощью «человека с плеткой» обречена на провал. Но это не значит, что «плохим» объектом вообще нельзя управлять.

Можно смягчить требования, отказаться от идеи достижения эталона и добиваться всего лишь *приемлемого* поведения системы.

Можно вместо *последовательной*, применить *параллельную* компенсацию. Если мы не можем на объект *надавить*, его (объект) можно *подменить*. Один из вариантов такого подхода (в случае неустойчивого объекта) – охватить объект стабилизирующей обратной связью и сделать его устойчивым.

Помимо метода динамической компенсации существуют, конечно, и другие методы структурно-параметрического синтеза, например, метод уравнений синтеза, модальное управление и др. [9].

2.2.3.3. «Практические» методы синтеза регуляторов

Эти методы названы «практическими» не потому, что они не имеют теоретического обоснования. Такое обоснование, конечно, есть, просто дано оно было позднее, чем появились соответствующие регуляторы. Речь идет о регуляторах, которые имеют заранее заданную (и довольно простую) структуру, а их «синтез» состоит в настройке немногочисленных параметров (т.е. мы говорим о параметрическом синтезе). Несмотря на то, что сейчас можно достаточно дешево реализовать сложные регуляторы, на практике в большинстве случаев применяются простые законы регулирования. Просто все уже давно знают, как их построить и настроить. И они будут работать, может быть и не лучшим, но вполне приемлемым образом.

Начать следует, конечно, с релейных регуляторов. Как уже говорилось выше, работают они самым простым образом: что-то включают, или отключают. Закон регулирования обычно задается *статической характеристикой регулятора* – графической зависимостью выходного сигнала от входного. Выходной сигнал регулятора может иметь несколько значений (уровней), например, «включено»/«выключено», «включен нагрев»/«включено охлаждение»/«все выключено» (одновременно включать и нагрев и охлаждение даже самый простой регулятор, конечно, не будет) и т.д. Входным сигналом регулятора в системе, работающей по отклонению, всегда является ошибка регулирования, т.е. разность между заданным и фактическим значениями выходной величины.

По существу в релейном регуляторе настройке подлежат два параметра: *порог срабатывания* и *гистерезис* или, как говорят иначе, *зона неоднозначности.*

Рассмотрим, например, статическую характеристику некоего гипотетического «утюга». Предположим нагревательный элемент должен включаться, если температура падает более чем на 1°С ниже заданной (т.е. ошибка регулирования превышает 1°С), и отключаться, если температура превышает заданное значение более чем на 1ºС (т.е. ошибка становится меньше –1ºС) (рис. 2.83).

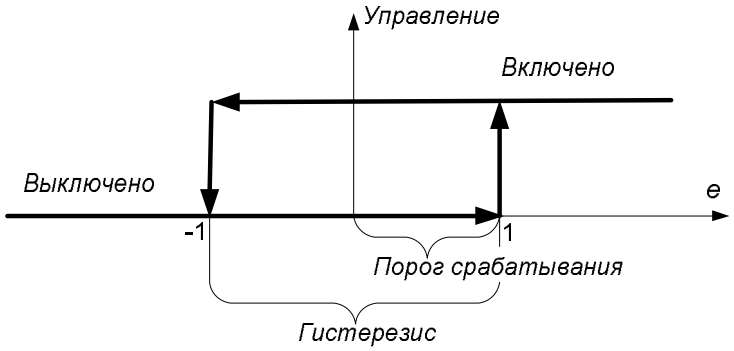


Рис. 2.83. Статическая характеристика двухпозиционного релейного регулятора.

В данном случае порог срабатывания равен одному градусу Цельсия, а гистерезис – двум. Зачем нам вообще нужен гистерезис? Этот вопрос мы уже слегка затрагивали выше, в начале главы. Но здесь можно рассмотреть его подробнее.

Если мы имеем дело с малоинерционным объектом (а утюг, сам по себе, – такой), то «разнесение» «точек» включения и отключения позволит нам ограничить частоту срабатывания коммутирующего аппарата (реле) и самого нагревательного элемента, что, безусловно, «продлит им жизнь».

Если объект управления обладает большой инерционностью и реагирует на управление с большой временной задержкой, казалось бы, гистерезис совсем не нужен. Но это совсем не так. Здесь уже дело не в объекте, а в измерительном преобразователе регулируемой величины, выходной сигнал которого в большинстве случаев подвержен «флуктуациям», случайным колебаниям, что может быть обусловлено различными факторами, например, помехами. Соответственно также будет вести себя и сигнал ошибки. И если в момент времени $t = 0$ $e = 0{,}0001$, то в момент $t = 0{,}1$ $e = –0{,}0002$, а в момент $t = 0{,}2$ $e = 0{,}00015$. При отсутствии гистерезиса релейный элемент будет постоянно переключаться и, в конце концов, выйдет из строя. Однако если включить гистерезис и установить его «шире» максимальной амплитуды колебаний сигнала измерительного преобразователя, «ложных» переключений можно избежать.

В рассмотренном двухпозиционном регуляторе *один* порог срабатывания и *один* гистерезис. Понятно, что если регулятор трех- и более позиционный, соответственно получим два и более значения порогов срабатывания и гистерезисов. На рис. 2.84 показана статическая характеристика трехпозиционного реле.

Такой регулятор может использоваться, например, в следящей системе регулирования положения и управлять электродвигателем.

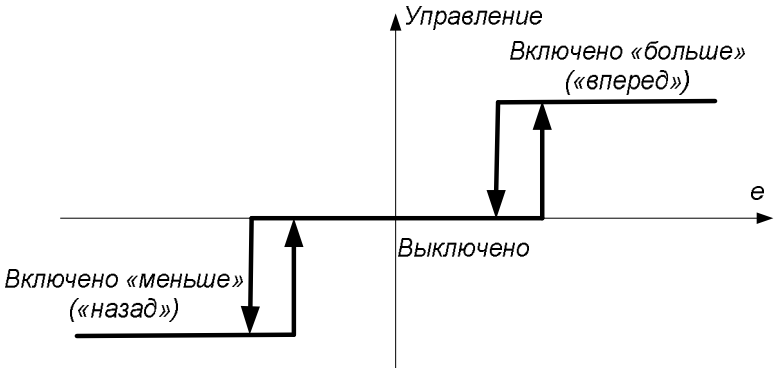


Рис. 2.84. Статическая характеристика трехпозиционного релейного регулятора.

В большинстве релейных систем процесс регулирования носит колебательный характер. Так как количество уровней управляющего воздействия ограничено, реле не в состоянии «подобрать» необходимую для объекта «мощность» управления, и вынуждено постоянно переключаться. Пороги срабатывания и гистерезисы выбирают таким образом, чтобы амплитуда и частота колебаний регулируемой величины не превышали допустимых значений. (Частота колебаний регулируемой величины и есть частота переключения реле, поскольку эти колебания и вызываются переключениями.)

Однако некоторые релейные системы могут находиться в установившихся состояниях. Для того чтобы это было возможно, релейный элемент должен иметь нулевой уровень управления («выключено»), а объект не должен обладать самовыравниванием. Попросту говоря, необходимо, чтобы объект был способен «останавливаться» в любом произвольном положении и оставаться в нем при нулевом управлении. Вспомним, как ведет себя интегратор. Именно так он себя и ведет. Упомянутая выше релейная следящая система – пример таких систем. Остановленный двигатель (условный интегратор) «назад не поедет» (если только извне его кто-нибудь не раскрутит). Точность регулирования в таких системах целиком определяется шириной участка нулевого управления, т.н. *зоны нечувствительности*, так как выключенный двигатель в принципе может остановиться только в этой зоне, причем в любом ее «месте». Чтобы увеличить точность, нужно уменьшить зону нечувствительности, но это грозит тем, что двигатель может и не успеть остановиться внутри нее. Тогда он будет включен повторно, уже в другую сторону, и, возможно, начнется совершенно ненужный процесс колебаний. Таким образом, существует какая-то предельная величина зоны нечувствительности. Если по соображениям точности она нас устраивает, релейная система может быть «принята в эксплуатацию», в против-

ном случае ничего не поделаешь, придется переходить к более сложным регуляторам.

Разработаны методы анализа и синтеза релейных систем, в частности *метод гармонического баланса* [8]. С одной стороны (анализ) эти методы позволяют по передаточной функции объекта управления и характеристикам реле (пороги срабатывания, гистерезисы) найти амплитуду и частоту колебаний (или доказать их отсутствие). С другой стороны (синтез) они дают возможность определить характеристики реле исходя из требований, предъявляемых к процессу регулирования (допустимые амплитуда и частота колебаний или отсутствие колебаний вообще). Правда, в отличие от анализа, синтез, как мы уже говорили, может быть и безуспешным. Кроме этих «теоретических» методов в нашем распоряжении имеются системы имитационного моделирования, например, Matlab Simulink, в которых мы можем построить модель системы, посмотреть, как она себя ведет, и даже провести настройку регулятора.

Если релейная система не в состоянии обеспечить выполнения требований к процессу регулирования, приходится переходить к непрерывным законам. Как уже говорилось, на практике это означает удорожание системы, так как теперь вместо дешевого коммутатора (реле) нам потребуется более дорогой преобразователь, способный выдавать любые уровни управляющего воздействия (в определенных пределах, конечно).

На практике широкое распространение получили линейные регуляторы, реализующие один из так называемых типовых законов регулирования [10]:

– пропорциональный (П);

– интегральный (И);

– пропорционально-интегральный (ПИ);

– пропорционально-дифференциальный (ПД);

– пропорционально-интегрально-дифференциальный (ПИД).

Рассмотрим ПИД закон регулирования, – другие законы можно считать его частными случаями. Идеальный ПИД регулятор имеет передаточную функцию

$$W_p(p) = k_p\left(1 + \frac{1}{T_u p} + T_\partial p\right)$$

и формирует управляющее воздействие на объект в виде:

$$u(t) = k_p\left(e(t) + \frac{1}{T_u}\int_0^t e(\tau)d\tau + T_\partial \frac{de(t)}{dt}\right),$$

где k_p – коэффициент передачи регулятора; T_u , T_∂ – постоянные времени интегрирования и дифференцирования, $e(t)$ – входной сигнал регулятора (ошибка регулирования – разность между заданным и фактическим значениями регулируемой величины), τ – вспомогательная переменная интегрирования.

Управляющее воздействие, формируемое ПИД регулятором, пропорционально сумме ошибки регулирования, интеграла ошибки и ее производной по времени.

Воздействие на объект *пропорционально* разности между заданным и действительным значениями выходной величины является общим принципом автоматического (да и ручного) регулирования. Выражаясь образно, пропорциональная составляющая как бы «олицетворяет» *настоящее время*: регулятор реагирует на величину ошибки в текущий момент.

Введение в закон регулирования *интегральной составляющей* позволяет устранить статическую ошибку в САР. При этом, поскольку интегратор вводится в систему до места приложения всех возмущающих воздействий, в статике система будет нечувствительной ко всем возмущениям. Интегрирование – это, по сути, *накопление*, поэтому можно сказать, что интегральная составляющая работает *в прошлом времени*.

Дифференциальная составляющая ПИД закона призвана ускорить реакцию регулятора на изменения задающего и возмущающих воздействий. Это достигается за счет того, что в начальный момент переходного процесса, вызванного изменением какого-либо воздействия, когда величина ошибки $e(t)$ еще мала, чтобы пропорциональная, а тем более интегральная составляющие регулятора смогли ее «почувствовать», скорость изменения ошибки уже достаточно большая, и дифференциальная составляющая эффективно реагирует на нее. Поэтому в некотором смысле дифференциальная составляющая «предсказывает» *будущее*.

Таким образом, ПИД регулятор работает как бы и в настоящем и в прошлом и в будущем времени. Возможно, таким когда-то он и задумывался в начале 20 века, во всяком случае, к теории автоматического управления, как нам представляется, ПИД регулятор имеет весьма опосредованное отношение, хотя на практике применяется повсеместно.

Передаточная функция ПИД регулятора в чистом виде нереализуема, так как предполагают использование производной входного сигнала, вследствие чего порядок числителя его передаточной функции больше порядка ее знаменателя:

$$W_p(p) = k_p\left(T_\partial p + 1 + \frac{1}{T_u p}\right) = \frac{k_p\left(T_u T_\partial p^2 + T_u p + 1\right)}{T_u p}.$$

Поэтому такой регулятор и назван идеальным. Реальные ПИД регуляторы формируют приведенный выше закон регулирования приближенно. В частности в программной реализации производная ошибки может быть заменена разностью двух отчетов, деленной на интервал времени между измерениями. Пойдя в этом направлении, можно перейти к полностью цифровым реализациям регулятора, которые описываются уже *разностными* уравнениями и дискретными передаточными функциями на основе z-преобразования [11].

Однако даже если идеально реализовать идеальный регулятор, остается одна существенная проблема. Сигнал измерительного преобразователя часто зашумлен. В нем присутствуют высокочастотные помехи, наведенные в измерительном канале различной аппаратурой. Может быть, в принципе, зашумлен и сигнал задания, если он приходит через аналоговый вход. Так или иначе, если

шум не отфильтровывается входным модулем, он проникает на вход регулятора. Если не предусмотреть фильтрацию и здесь, шум будет усилен дифференциальной составляющей (дифференцирование усиливает сигналы высокой частоты, причем, чем больше частота, тем больше усиление). В результате регулятор будет работать неправильно: вместо того, чтобы обрабатывать «полезный сигнал», он будет реагировать на шум.

Если «соединить» фильтрацию и регулирование, можно получить физически реализуемый регулятор (без операции дифференцирования). Такой подход предполагает использование дополнительного звена – *фильтра* (или *балласта*), в роли которого обычно выступает инерционное звено первого порядка с единичным коэффициентом передачи. Это звено может быть установлено «на входе» *всего* регулятора:

$$W_p(p) = \frac{k_p}{T_{ф(б)}p + 1}\left(1 + \frac{1}{T_u p} + T_д p\right) = \frac{k_p\left(T_u T_д p^2 + T_u p + 1\right)}{T_{ф(б)} T_u p^2 + T_u p},$$

или отдельно, «на входе» дифференцирующей составляющей:

$$W_p(p) = k_p\left(1 + \frac{1}{T_u p} + \frac{1}{T_{ф(б)}p + 1} T_д p\right) = \frac{k_p\left(T_u T_д p^2 + \left(T_u T_д + T_u + T_{ф(б)}\right)p + 1\right)}{T_{ф(б)} T_u p^2 + T_u p}.$$

В зависимости от «отношения» к этому звену, его называют либо *фильтром*, либо *балластом*.

Фильтром его называют, когда хотят выделить его полезную функцию – фильтрацию входного сигнала. Балластом звено называют, когда хотят подчеркнуть, что оно искажает динамические свойства регулятора.

Слова «на входе» не случайно взяты в кавычки. На самом деле звено не устанавливается на входе, а непосредственно «внедряется» в структуру регулятора, «устраняя» дифференцирование и делая регулятор физически реализуемым. Теперь порядок числителя передаточной функции равен порядку знаменателя, следовательно, регулятор можно собрать на интеграторах, сумматорах и масштабных коэффициентах. В отличие от дифференцирования, интегрирование ослабляет шумы, причем тем лучше, чем выше их частота.

По сути, фильтрация шумов и обеспечение физической реализуемости ПИД регулятора – это одна и та же задача.

На практике наибольшее распространение получил ПИ регулятор, поскольку он малочувствителен к помехам и его проще настраивать. И это несмотря на то, что он значительно проигрывает ПИД регулятору в быстродействии.

Синтез (определение настроек) регуляторов производится различными методами, и этих методов довольно много. Принципиально имеет значение *качество* модели объекта управления. Не имея модели (или самого объекта), настроить регулятор, понятно, принципиально невозможно.

Часто дело обстоит так, что объект управления описан предельно упрощенной моделью, полученной не путем анализа процессов, происходящих в

нем, а в результате простейших экспериментов. Чрезвычайно распространены различные методы снятия и обработки *кривых разгона*, когда экспериментально определяют что-то вроде переходной характеристики объекта, а по ней находят параметры простой передаточной функции.

Рассмотрим типичный вариант: *метод точки перегиба*. Пусть у нас будет регулирование температуры с воздействием на регулирующий клапан на линии подачи теплоносителя. В «номинальном» режиме температура должна быть 70ºС, при этом клапан открыт на 60%. Выведем объект в этот режим и дадим ему «успокоиться». После этого приоткроем клапан еще на 10%, до 70%. С помощью регистрирующей аппаратуры получим «отклик» объекта, рис. 2.85.

Объект обладает самовыравниванием и, судя по графику, должен быть описан передаточной функций *высокого порядка*, т.е. *не первого*. Об этом говорит начальное «искривление» графика: объекты первого порядка «стартуют», как мы видели, «на максимальной скорости», а здесь скорость набирается «постепенно».

Но мы, согласно методу, не будем обращать внимание на данное обстоятельство и опишем объект передаточной функцией первого порядка, а все «огрехи» спишем на запаздывание:

$$W_{об}(p) = \frac{k_{об}}{T_{об}p + 1} \times e^{-\tau_{об}p}.$$

Постоянную времени $T_{об}$ и время запаздывания $\tau_{об}$ найдем проведя прямую через точку перегиба разгонной характеристики по касательной к ней (рис. 2.85).

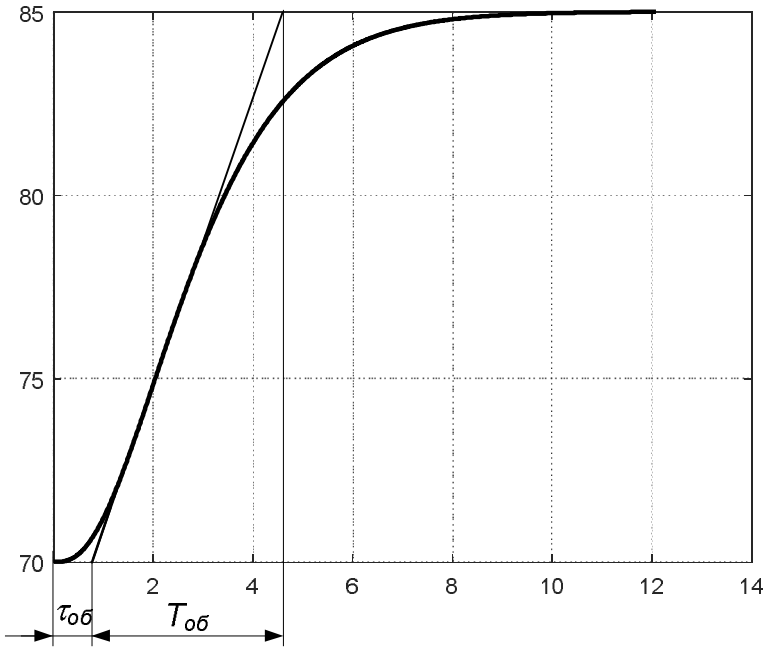


Рис. 2.85. Кривая разгона и ее обработка.

В нашем случае получим: $T_{об} \approx 3{,}9$ мин, $\tau_{об} \approx 0{,}7$ мин. Коэффициент передачи найдем как отношение приращения значения температуры к приращению хода клапана:

$$k_{об} = \frac{\Delta t}{\Delta h} = \frac{15^{\circ}}{10\%} = 1,5\,^{\circ}/\%.$$

Таким образом, получаем:

$$W_{об}(p) = \frac{1,5}{3,9p+1} \times e^{-0,7p}.$$

Имея такое упрощенное описание объекта, адекватность которого в общем случае не гарантируется, применять какие-либо сложные методы расчета настроек регулятора нет никакого смысла. Поэтому используют упрощенные методы, например, *формульный*. Существуют простые формулы для определения коэффициентов настроек регулятора по известным $k_{об}$, $T_{об}$, $\tau_{об}$. Причем имеются разные формулы для разных *типовых* процессов, например, *процесса с 20% перерегулированием*, *максимально быстрого неколебательного процесса* и др.[12].

Приведем в качестве примера формулы для определения настроек ПИД регулятора для процесса с 20% перерегулированием:

$$k_p = \frac{1,2}{k_{об}\left(\tau_{об}/T_{об}\right)},$$

$$T_и = 2\tau_{об}, \quad T_д = 0,4\tau_{об}.$$

Более точным методом определения настроек регулятора по упрощенному описанию объекта считается применение специальных номограмм, учитывающих различные «нелинейные эффекты», которые не могут учесть формулы. На рис. 2.86 приведен пример такой номограммы для определения оптимальных настроек ПИ регулятора по известным $k_{об}$, $T_{об}$, $\tau_{об}$.

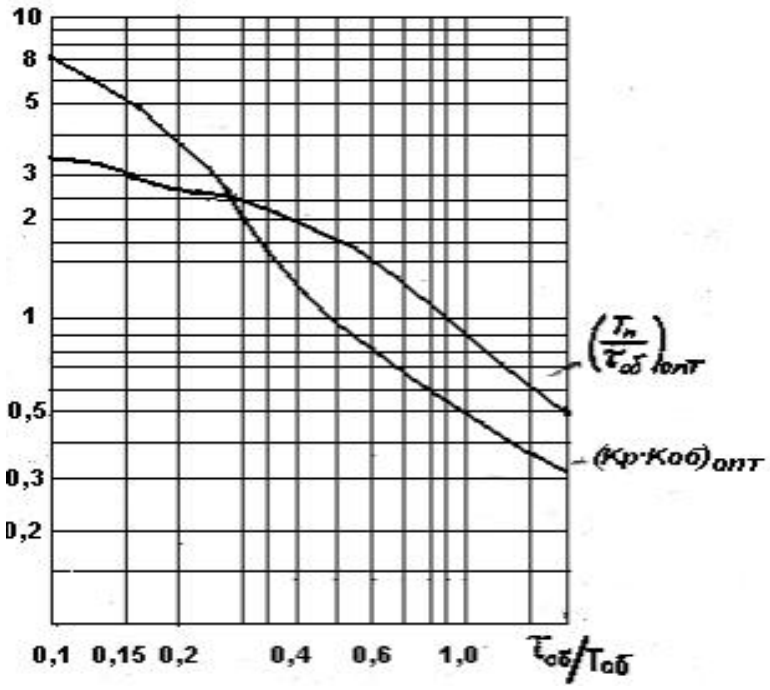


Рис. 2.86. Номограмма для определения настроек ПИ-регулятора.

Однако и формулы и номограммы получены исходя из того, что объект *действительно* описывается звеном первого порядка с запаздыванием, т.е. *ведет себя* как это звено. Но зная, как мы получили передаточную функцию, никакой уверенности в этом нет. «Раскроем карты»: исходная разгонная характеристика получена в Matlab с помощью кода:

```
W = tf(1.5,[1 3 3 1]);
[y,t]=step(W);
y = 70+y*10;
plot(t,y);
grid
```

Таким образом, «реальный» объект описывался передаточной функцией:

$$W_{об}(p)=\frac{1,5}{p^3+3p^2+3p+1}.$$

А теперь сравним поведение «реального» объекта с нашей упрощенной моделью:

```
W1 = tf(1.5,[3.9 1]);
W1.InputDelay = 0.7;
[y1,t1] = step(W1);
y1 = 70 + y1*10;
plot(t,y,t1,y1);
grid
```

Результаты показаны на рис. 2.87.

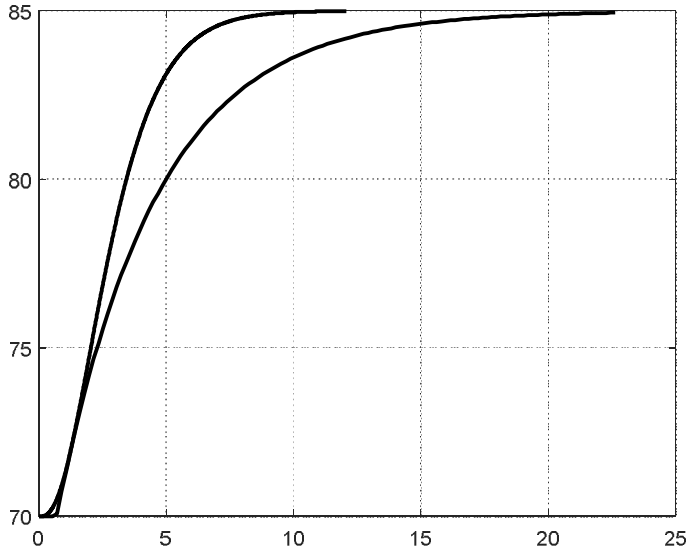


Рис. 2.87. Разгонная характеристика объекта и ее «аппроксимация».

Как мы видим, наша модель весьма несовершенна. И это притом, что построена она была в лабораторных условиях: никаких шумов и посторонних влияний, которые всегда имеют место на практике. Причина состоит в принципиальной невозможности описать сложный объект простой моделью.

Далее, если мы используем «неправильную» модель для определения настроек регулятора, мы получим «неправильный» регулятор и «неправильное» поведение системы. На рис. 2.88 показаны Simulink-диаграмма с моделями двух замкнутых системам регулирования и результаты имитационного моделирования.

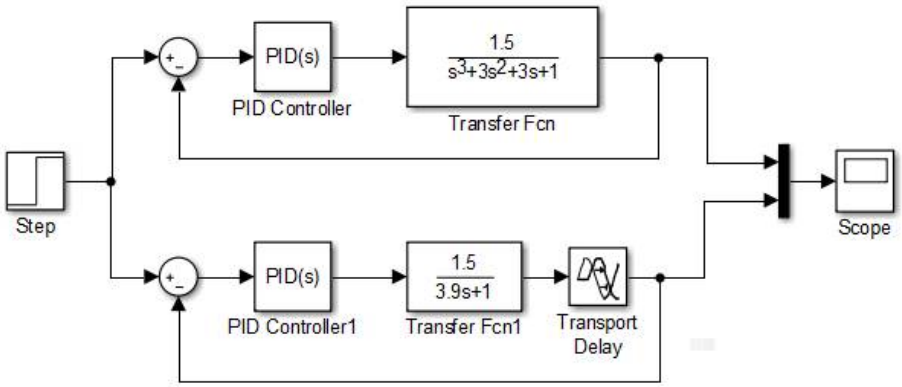


Рис. 2.88. Simulink-диаграмма.

В обеих системах регуляторы настроены одинаково, на простую модель. На рис. 2.89 жирная кривая линия – это реакция нижней, «эталонной», системы регулирования, а тонкая – реакция верхней, «реальной».

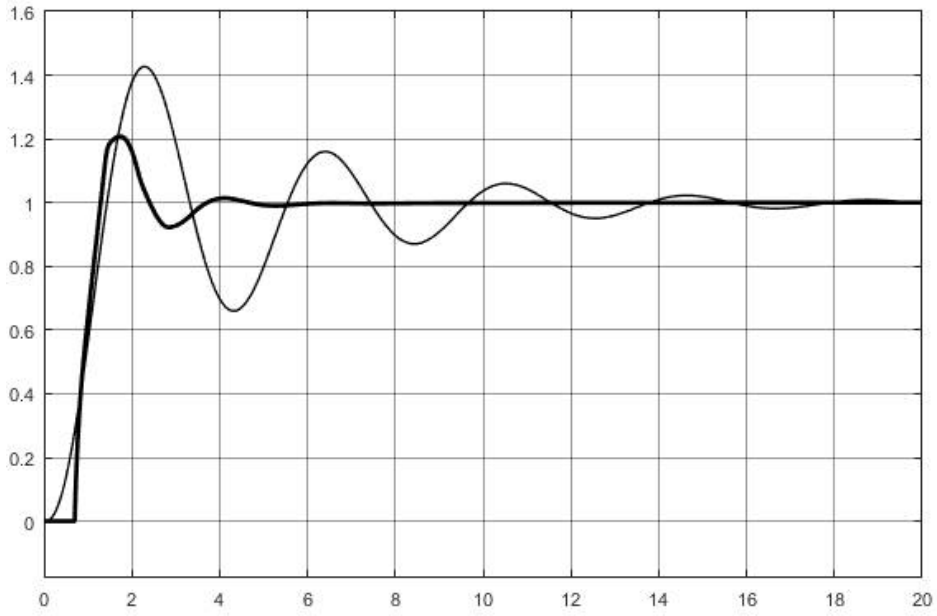


Рис. 2.89. Результаты моделирования.

В нашем случае использование простой модели для настройки регулятора привело к значительному ухудшению свойств системы (увеличению колебательности). В других случаях может быть и еще хуже, – реальная система вообще может оказаться неустойчивой.

«Утешением» для нас может послужить то, что существуют методы определения по кривой разгона передаточных функций второго (*метод Орманса*) и даже произвольного порядка (*метод последовательного логарифмирования, метод площадей* и др.) [13]. В особо ответственных случаях можно экспериментально снять и частотные характеристики, по которым передаточную функцию восстановить значительно проще. Есть также методы пассивного эксперимента, когда математическое описание объекта находят путем обработки «длинных» архивных записей его поведения.

И все же «настоящая» модель – это модель *аналитическая*. Уравнения такой модели действительно отражают процессы, проходящие в объекте. Как правило, такая модель достаточно сложная и чаще всего нелинейная.

Для настройки регулятора модель *линеаризуют*, т.е. находят линейные дифференциальные уравнения, более или менее адекватно описывающие поведение объекта в некотором диапазоне изменения переменных.

Если рассматривается система стабилизации, когда задание почти не меняется, то такой диапазон достаточно узок. Фактически можно считать, что система будет работать в одной, *рабочей точке*. Линеаризовать нелинейное уравнение в окрестности рабочей точке достаточно просто. Для этого уравнение разлагают в ряд Тейлора по всем входящим в него факторам и отбрасывают все нелинейные компоненты. В результате получаем линейное уравнение в отклонениях от рабочей точки [8]. Если система действительно далеко не будет отклоняться от рабочей точки, такое описание, скорее всего, будет вполне адекватным. С другой стороны, имея вполне адекватное описание объекта, можно так хорошо настроить регулятор, чтобы он не допустил больших отклонений от рабочей точки. Круг замкнулся.

С системами воспроизведения, особенно со следящими системами, дело обстоит сложнее. Здесь у нас никакой рабочей точки нет, а есть большой диапазон изменения переменных. Придется проводить линеаризацию в нескольких точках (на границах диапазона, где-то «посередине» и т.д.), как-то осреднять параметры полученных передаточных функций и уже по осредненной передаточной функции объекта (а может быть, и по самой «плохой»!) находить настройки регулятора. Потом обязательно следует проверить работу регулятора во всем диапазоне изменения переменных на имитационной модели с нелинейным объектом. К счастью, системы с ПИД регулятором (хорошо настроенным!), как правило, являются довольно «жесткими»: их поведение не очень сильно изменяется при значительных изменениях параметров объекта.

Имея «качественное» описание объекта, настройку регулятора нужно, естественно, проводить «качественными», т.е. теоретически обоснованными, методами. Таких методов тоже достаточно много. Поскольку ПИД регулятор и его «младшие братья» – структуры *ограниченные*, в этих методах не ставится задача достижения эталона, как в п. 2.2.3.2. Требуется всего лишь, чтобы заданного значения достиг какой-нибудь один «скромный» показатель качества системы. Например, известно, что системы, у которых максимум АЧХ лежит в пределах 1,2 – 1,3 в большинстве случаев работают «неплохо». Соответственно разработан метод настройки регулятора на заданный *частотный показатель колебательности*. Имеется также методы настройки на заданный *корневой показатель колебательности* [14], на заданный *запас устойчивости по амплитуде* [15] и т.д. и т.д.

Большинство методов «работают» с частотными характеристиками и часто не дают однозначного решения, поскольку заданного показателя можно достичь при различных вариантах настройки регулятора (особенно это касается полного, ПИД регулятора, так как у него целых три настраиваемых параметра). Тогда применяют дополнительные критерии выбора, например, *критерий максимального подавления низкочастотных возмущений* [14].

Ну и, наконец, есть возможность настроить регулятор вообще без модели, *на самом объекте*. Рассмотрим, например, классический *метод незатухающих колебаний* или *метод Зиглера – Никольса.* Согласно нему в работающей систе-

ме отключаются интегральная и дифференциальная составляющие регулятора ($T_и = \infty$, $T_д = 0$), т.е. регулятор становится пропорциональным. Последовательно увеличивая коэффициент передачи регулятора и одновременно подавая небольшой скачкообразно изменяющийся сигнал задания, добиваются возникновения в системе незатухающих колебаний. При этом фиксируют значения критического коэффициента усиления регулятора $k_{кр}$ и периода критических колебаний в системе $T_{кр}$. По ним находят настройки регулятора. Для ПИД регулятора это делается по формулам:

$$k_р = 0,6k_{кр},\quad T_и = \frac{T_{кр}}{2},\quad T_д = \frac{T_{кр}}{8}.$$

Имеются и другие подобные методы, например, метод затухающих колебаний [12]. На основе таких методов разработаны различные алгоритмы самонастройки программно реализованных регуляторов для ПЛК.

В Matlab настройку ПИД регулятора выполняет специальная программа `pidtune`. Ее возможности ограниченны линейными моделями объекта. Однако программа интегрирована и в Simulink-блок `PID Controller`, а Simulink умеет проводить линеаризацию нелинейных моделей. Поэтому `PID Controller` может самостоятельно настраиваться для любых, в том числе нелинейных объектов. У пользователя имеется возможность влиять на настройку, задавая «пожелания» относительно быстродействия и колебательности системы.

2.2.3.4. Синтез многоконтурных систем автоматического регулирования

Автоматизированные и неавтоматизированные системы управления чем бы то ни было всегда иерархичны. Они включают несколько расположенных друг под другом уровней управления (принятия решений), причем нижние уровни подчиняются верхним. Чем ниже уровень, тем «локальнее» задача, которая перед ним ставится. Нижние уровни работают «быстро», решая «мелкие» задачи, а верхние работают «медленно», но задачи, стоящие перед ними – «крупные». Каждая «крупная» задача на верхнем уровне превращается в набор заданий, спускаемых «вниз». Что-то подобное реализуется в многоконтурных системах автоматического регулирования.

Многоконтурная (каскадная) система включает ряд вложенных друг в друга контуров регулирования. Внешний контур отвечает за регулирование основной (выходной) величины системы, внутренние – за регулирование «промежуточных» величин. Еще одно название таких систем – *системы подчиненного регулирования*. Каждый контур включает в себя свой регулятор, задание для которого формируется регулятором внешнего контура.

На рис. 2.90 показана схема *двухконтурной* системы.

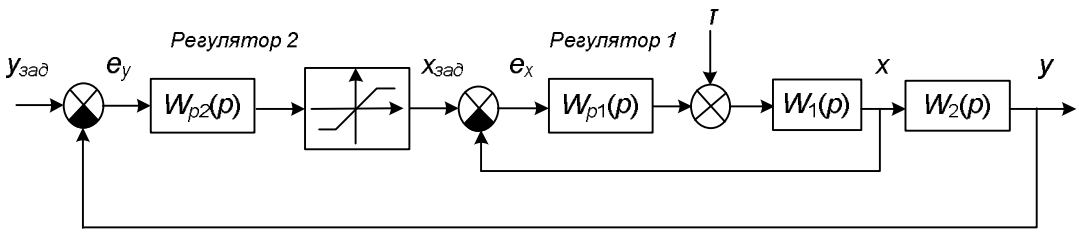


Рис. 2.90. Двухконтурная система регулирования.

Объект представлен последовательным соединением двух звеньев с передаточными функциями $W_1(p)$ и $W_2(p)$. Предполагается, что есть возможность измерять некоторую промежуточную величину x, которая формируется на выходе звена с передаточной функцией $W_1(p)$. За основную регулируемую величину y «отвечает» регулятор с передаточной функцией $W_{p2}(p)$. Его выходной сигнал ограничен с помощью звена ограничения. Регулятор дает задание внутреннему контуру, в котором производится регулирование величины x.

При построении многоконтурных систем обычно преследуется две цели:

1) ограничение на допустимом уровне внутренних координат (в данном случае $x(t)$). Необходимость такого ограничения на практике вызвана требованиями безопасной работы объекта. Например, в системе регулирования скорости электропривода помимо основной задачи (самого регулирования скорости) может решаться и вспомогательная – ограничение тока двигателя. Если внутренний контур настроен правильно, то ограничивая задание на его входе, мы тем самым ограничиваем и выходную величину контура;

2) повышение качества регулирования. Так как внутренний контур системы является менее инерционным, чем внешний, его реакция на возмущения, действующие в нем, может быть настолько быстрой, что выходная величина системы не успеет значительно отклониться от задания и «основному» регулятору не придется «сильно напрягаться». Все «местные проблемы» должны решаться «на местах». Стабилизация внутренних координат делает систему менее чувствительной к возмущениям.

На рис. 2.91, 2.92 представлены Simulink-модели одноконтурной и двухконтурной систем управления.

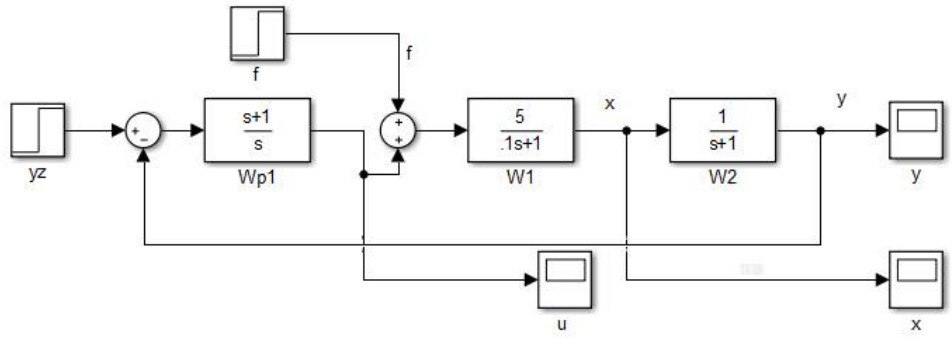


Рис. 2.91. Simulink-модель одноконтурной системы регулирования.

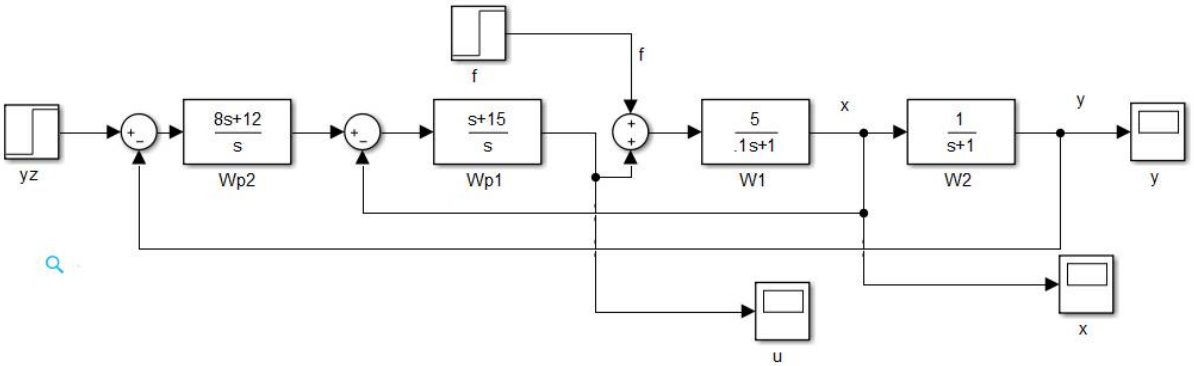


Рис. 2.92. Simulink-модель двухконтурной системы регулирования.

В обеих системах все регуляторы реализуют ПИ-закон регулирования. Ограничением внутренней координаты мы пока не занимаемся. На рис. 2.93 показаны графики изменения сигналов $y(t)$ и $x(t)$ при подаче задающего (в момент времени 0) и возмущающего (в момент времени 2,5) воздействий.

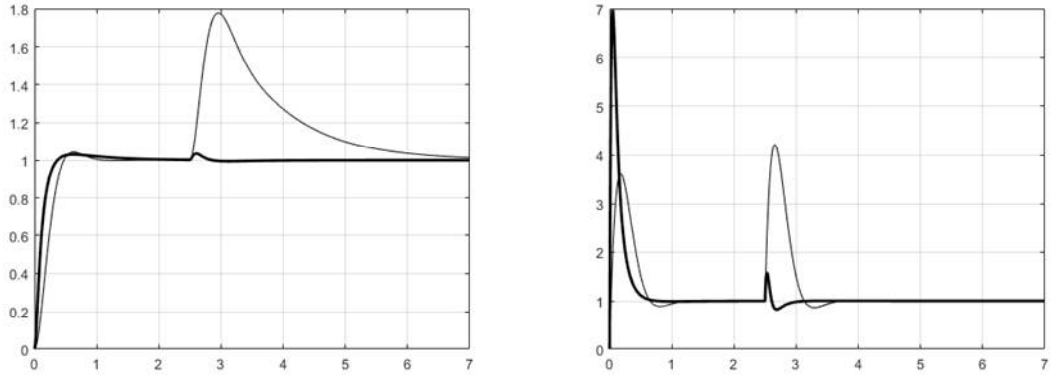


Рис. 2.93. Переходные процессы $y(t)$ и $x(t)$ в одноконтурной (тонкая линия) и двухконтурной (жирная линия) системах регулирования.

Как видно из рисунков, в двухконтурной системе возмущение подавляется намного эффективнее, чем в одноконтурной. В то же время изменение задающего сигнала обе системы отработали почти одинаково, если не обращать внимания на то, что в двухконтурной системе промежуточная величина $x(t)$ совершила резкий скачок в начале переходного процесса. Этот скачок объясняется тем, что в двухконтурной системе установлен «более мощный» регулятор основной регулируемой величины y, чем в одноконтурной. Зачем нам понадобился «более мощный» регулятор, мы сейчас выясним.

На рис. 2.94 показаны графики изменения управляющего воздействия $u(t)$ в двух системах.

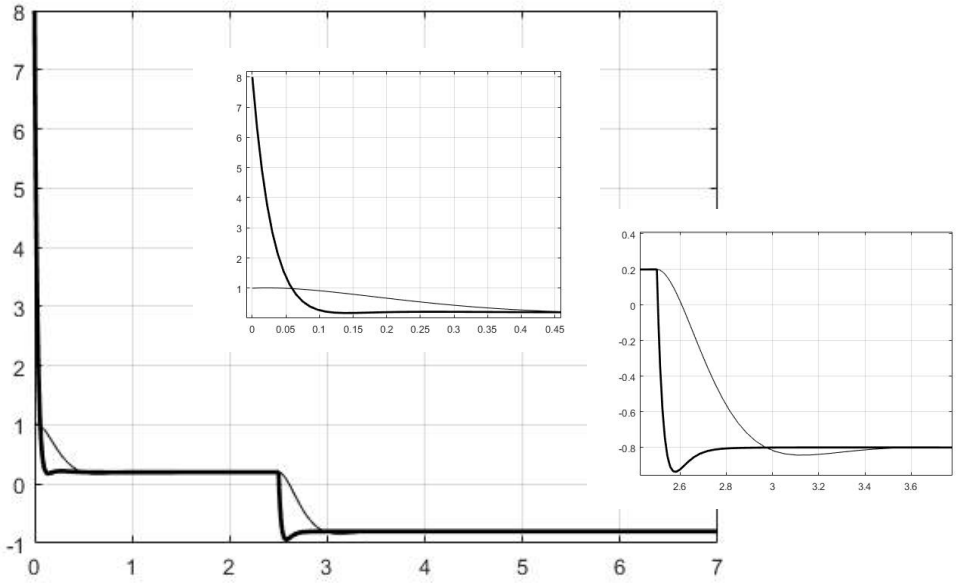


Рис. 2.94. Графики изменеия управляющего воздействия $u(t)$ в одноконтурной (тонкая линия) и двухконтурной (жирная линия) системах регулирования.

И при отработке задания, и при отработке возмущения в двухконтурной системе управление вело себя более «агрессивно». Важно понять, с какой стороны возникла «агрессия». При подаче возмущения, очевидно, «основной удар» взял на себя регулятор внутреннего контура. Ведь проблема (отклонение *x* от задания) возникла именно у него, основной регулятор пока еще почти ничего не почувствовал, подопечная ему величина *y* изменилась совсем немного.

Однако при отработке задания все было совсем по-другому. Основной регулятор двухконтурной системы, обнаружив большую ошибку на своем входе, немедленно командует своему «подчиненному»: «голову расшиби, но задание выполни».

В одноконтурной системе регулятор один, он «знает» (а точнее, это мы за него знаем), свой объект и в отсутствие возмущений может с ним справиться «малыми силами». В двухконтурной системе основной регулятор «знает», что в подчинении у него «контора», которая преследует свои собственные цели («забота» о своей «родной» координате *x*), и которой совершенно нет никакого дела до решения основной задачи (регулирования величины *y*). «Зная», что объект у него сложный и «своенравный», основной регулятор вынужден «давить» на «подчиненного», чтобы достигнуть показателей не худших, чем в одноконтурной системе (при этом мы вместе с регулятором «вздыхаем»: «легче самому сделать, чем другим объяснить»). И подчиненному ничего не остается, как форсировать управление (а что ему остается делать, задание же надо выполнять!).

Однако все это «работает», пока основной регулятор не проявляет «волюнтаризм» и ставит перед подчиненным «реальные цели». Если регулятор скорости не хочет угробить двигатель, он должен свои желания как-то ограничить. Именно для этого и ограничивается выходной сигнал регулятора внешнего контура, см. рис. 2.90.

На рис. 2.95 показана Simulink-модель двухконтурной системы с ограничением внутренней величины *x* в пределах от –2 до 2. Ограничение задания подчиненному регулятору производится блоком Saturation. Помимо непосредственного ограничения происходит также и «замораживание» интегральной составляющей основного регулятора путем подачи на вход его интегратора нулевого сигнала на время действия ограничения.

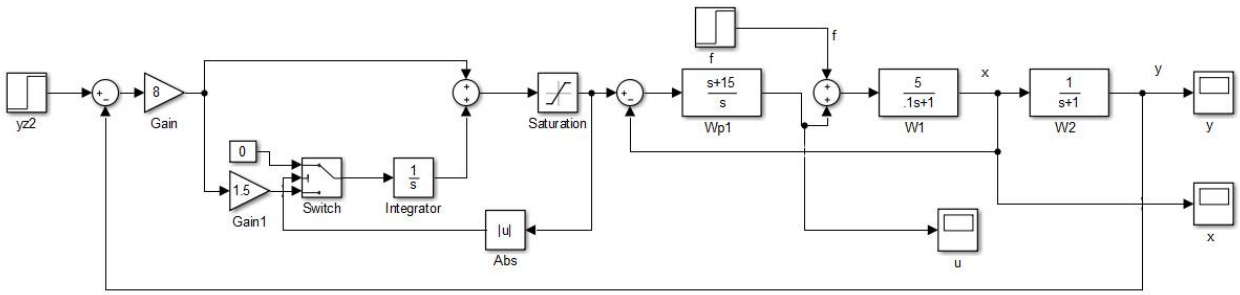


Рис. 2.95. Simulink-модель двухконтурной системы регулирования с ограничением.

Переключатель Switch подает на вход интегратора сигнал с первого своего входа, если на втором входе значение сигнала равно или больше двух единиц (так он настроен). «Замораживание» интегральной составляющей необхо-

димо для того, чтобы остановить интегрирование, пока действует ограничение. Если этого не сделать, интегратор будет продолжать увеличивать свой выходной сигнал и «развернется» только тогда, когда ошибка регулирования изменит знак. После этого потребуется еще некоторое время для того, чтобы вернуться «в норму», что еще больше отложит момент выхода из ограничения. В результате процесс регулирования может стать излишне колебательным, а в худшем случае колебания не прекратятся никогда (наша система станет вести себя почти также как релейная).

На рис. 2.96, 2.97 показаны переходные процессы в каскадной системе с ограничением величины *x*. Ограничение действовало в течение первых примерно 0,5 с, а потом было снято. Величина *x* ненадолго и «недалеко» вышла из отведенного ей коридора. Это объясняется тем, что регулятор внутреннего контура настроен на процесс с небольшим перерегулированием. На практике никаких проблем это не создаст: мы всегда можем задать пороги срабатывания ограничения «с запасом», в крайнем случае, можно и регулятор настроить на апериодический процесс.

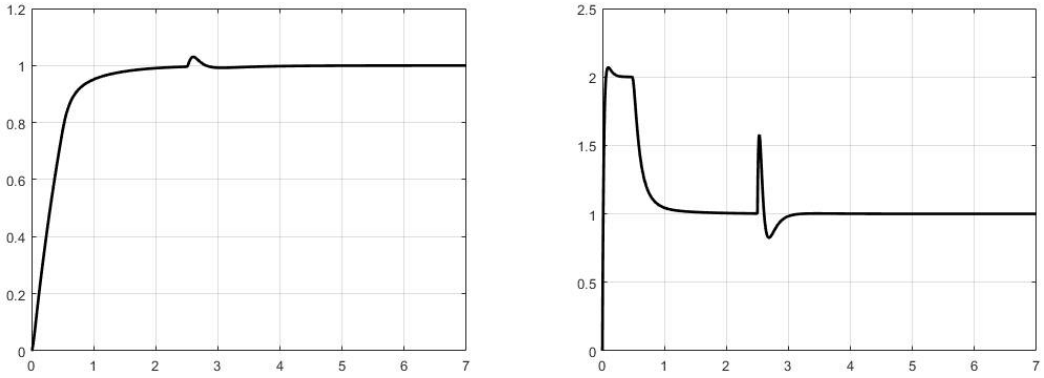


Рис. 2.96. Переходные процессы $y(t)$ и $x(t)$ в двухконтурной системе регулирования с ограничением.

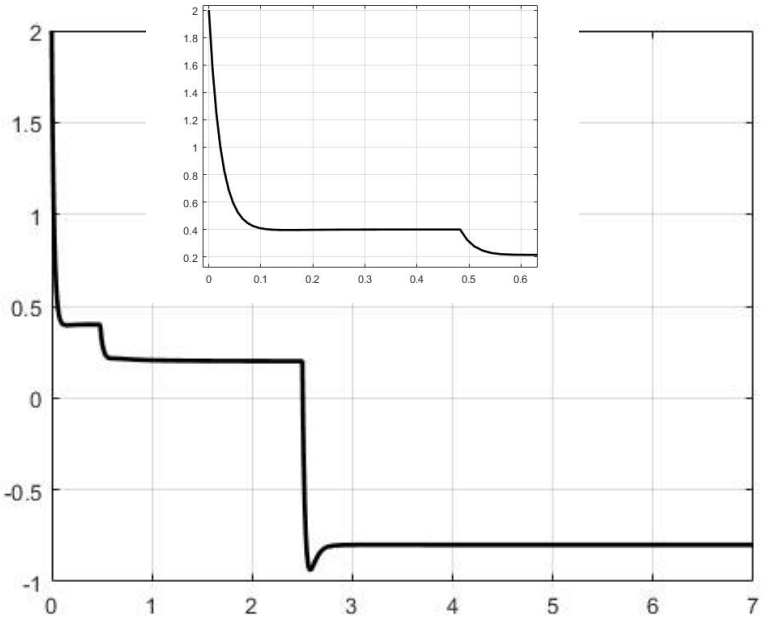


Рис. 2.97. График изменения управляющего воздействия $u(t)$ в двухконтурной системе регулирования с ограничением.

Как и следовало ожидать, при отработке возмущения ограничение в силу не вступало, регулятор внутреннего контура решил проблему самостоятельно. Вместе с тем ограничение замедлило отработку задания, что тоже вполне объяснимо.

Расчет регуляторов многоконтурных систем осуществляется теми же методами, что и одноконтурных.

Начинают расчет с самого внутреннего контура. Определив параметры (или даже структуру) внутреннего регулятора, рассчитывают передаточную функцию контура:

$$W_{к1}(p) = \frac{W_{p1}(p)W_1(p)}{1 + W_{p1}(p)W_1(p)}.$$

Далее определяется передаточная функция эквивалентного объекта для внешнего регулятора и рассчитывается он сам.

Так, для системы, приведенной на рисунке, регулятор $W_{p1}(p)$ настраивается (рассчитывается) для объекта $W_1(p)$, а регулятор $W_{p2}(p)$ – для *эквивалентного* объекта с передаточной функцией:

$$W_{2\,экв}(p) = W_{к1}(p)W_2(p).$$

Таким образом, передаточная функция эквивалентного объекта может быть достаточно сложной, даже если передаточные функции объекта $W_1(p)$ и $W_2(p)$ простые. Это дает нам основание использовать сложные методы синтеза (настройки) регулятора внешнего контура.

Как мы видим, при расчетах ограничение внутренних координат не учитывается, однако совсем не лишним будет провести имитационное моделирование рассчитанной системы уже с ограничениями.

2.2.3.5. Синтез комбинированных систем автоматического регулирования

Часто одноконтурные АСР, работающие по отклонению, не обеспечивают удовлетворительного качества регулирования вследствие воздействия на объект существенных возмущений. Если имеется возможность автоматического измерения наиболее «сильного» возмущающего воздействия на объект, то применяется *комбинированная* АСР.

Действие контролируемого возмущения компенсируется специальным устройством (алгоритмом) – *компенсатором*, а с помощью регулятора, находящегося в контуре обратной связи, устраняется действие других (не контролируемых) возмущений.

Компенсация возмущений осуществляется путем введения дополнительного управляющего воздействия либо на вход регулятора, либо на его выход, рис. 2.98, 2.99.

В обеих схемах объект представлен двумя передаточными функциями: по управлению $W_{ou}(p)$ и по возмущению $W_{of}(p)$. Если возмущение можно измерить, вторую передаточную функцию всегда можно определить – хотя бы экспериментально.

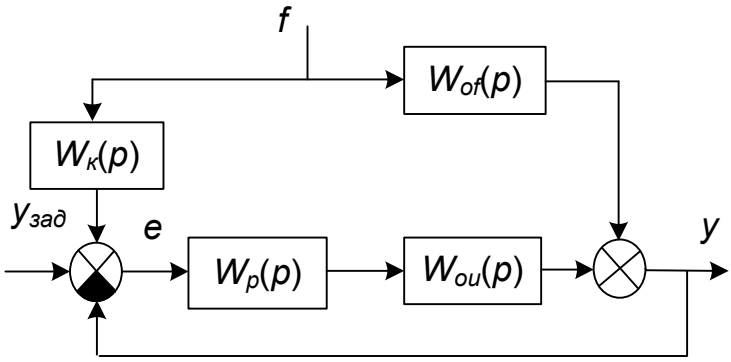


Рис. 2.98. Схема комбинированной системы при подаче компенсирующего сигнала на вход регулятора.

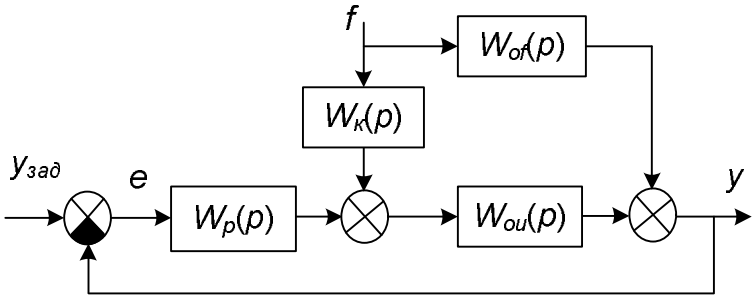


Рис. 2.99. Схема комбинированной системы при подаче компенсирующего сигнала на выход регулятора.

Раньше, в эпоху аналоговых регуляторов применялась в основном первая схема, так как измерительный блок регулятора, вычисляющий взвешенную сумму сигналов, действительно был на его «входе». Сейчас в основном применяется второй подход, поскольку сложить два сигнала в настоящее время – это всего лишь применить операцию сложения двух или более переменных в программе управления, а компенсатор во второй схеме имеет более простую структуру, чем в первой. Поэтому далее рассмотрим только второй вариант.

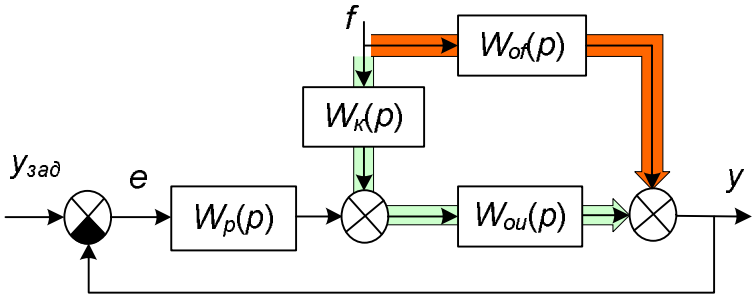


Рис. 2.100. Компенсация возмущения.

Идея компенсации возмущения очень проста: предполагается взаимное подавление двух сигналов друг другом (рис. 2.100).

Условие полной ompенсации:

$$f \times W_{of}(p) + f \times W_к(p) W_{ou}(p) = 0.$$

откуда

$$W_к(p) = -\frac{W_{of}(p)}{W_{ou}(p)}.$$

Если нам каким-то чудесным образом удалось абсолютно точно определить передаточные функции объекта, а потом также абсолютно точно реализовать компенсатор, мы получим идеальную систему, полностью нечувствительную к возмущению. Увы, на практике такого не бывает, и точная компенсация невозможна.

Более того, в ряде случаев передаточная функция компенсатора оказывается вообще физически нереализуемой. Например, порядок числителя $W_к(p)$ может оказаться больше порядка знаменателя, и тогда нам придется приближенно реализовать дифференцирование. Может получиться и так, что $W_к(p)$ имеет «отрицательное запаздывание» (предварение). Пусть

$$W_{ou}(p)=\frac{1}{p+1}e^{-p},\ W_{of}(p)=\frac{2}{0,5p+1}e^{-0,5p},$$

тогда

$$W_к(p)=-\frac{W_{of}(p)}{W_{ou}(p)}=\frac{2p+2}{0,5p+1}e^{0,5p}.$$

Эта передаточная функция физически нереализуема: реакция компенсатора должна на 0,5 единиц времени опережать входное воздействие. Откуда нам заранее знать, когда возмущение начнет изменяться?

В таких случаях применяют приближенную реализацию компенсатора (т.н. *реальный* компенсатор). Передаточную функцию реального компенсатора (или ее параметры) обычно находят из условия максимальной близости его частотных характеристик характеристикам *идеального* (т.е. рассчитанного) компенсатора в существенном диапазоне частот (т.е. в том диапазоне частот, в котором действительно может изменяться возмущение). При этом как минимум требуют полной компенсации возмущения на нулевой частоте, т.е. в статическом режиме, а уж этого легко добиться, зная коэффициенты передачи объекта по управлению и возмущению.

2.2.3.6. Синтез многосвязных систем автоматического регулирования

К многосвязным относятся объекты регулирования, имеющие несколько регулируемых величин и несколько каналов воздействия.

Такие объекты встречаются во многих современных сложных системах. К ним относятся, например, синхронные генераторы, подвижные объекты, различные установки тепловых электрических станций и др.

Частным и простейшим, но в то же время и наиболее часто встречающимся на практике, случаем многосвязного объекта является двусвязный объект регулирования (рис. 2.101).

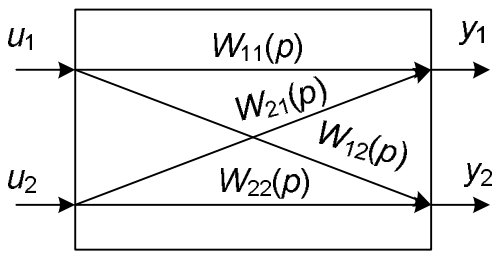


Рис. 2.101. Двухсвязный объект регулирования.

Перекрестные каналы связи можно понимать как каналы возмущения, которые приводят к взаимному влиянию двух контуров регулирования, ухудшая их устойчивость. Важно однако понимать, что мы имеем дело не с *настоящим* возмущением, которое не зависит от работы нашей системы, а с *искусственным*, которая наша система сама себе и устраивает.

Как уже говорилось в п. 2.2.1, для объектов с взаимосвязанными координатами применяют два типа автоматических систем регулирования:

несвязанного регулирования (рис. 2.102);

связного (автономного) регулирования (рис. 2.103).

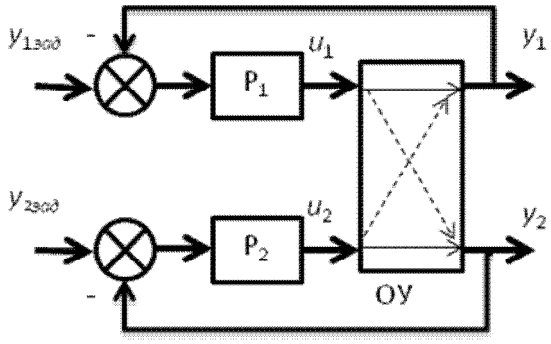


Рис. 2.102. Структура системы несвязного регулирования.

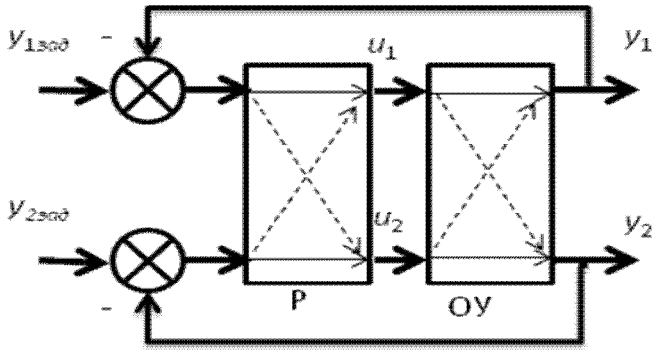


Рис. 2.103. Структура системы связного регулирования.

В первом случае мы имеем две условно независимые скалярные системы регулирования, а во втором – одну, векторную.

Структура системы несвязного регулирования в развернутом виде показана на рис. 2.104.

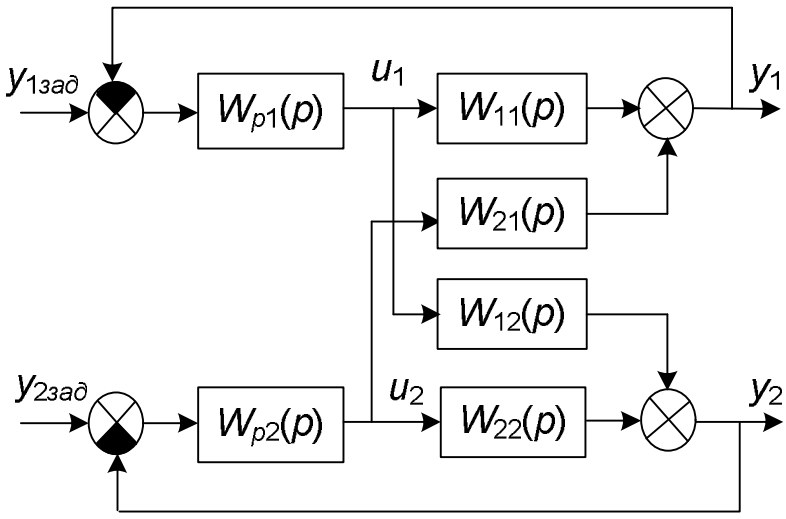


Рис. 2.104. Структура системы несвязного регулирования в развернутом виде.

Для определения настроек регуляторов в системах несвязного регулирования может использоваться метод автономной настройки, согласно которому настроечные параметры регуляторов $W_{p1}(p)$ и $W_{p2}(p)$ определяют без учёта перекрёстных связей $W_{12}(p)$ и $W_{21}(p)$.

Однако такой подход оправдан, только если влияние перекрестных связей мало. В противном случае это может привести к некачественно работающей или даже неустойчивой системе. Это объясняется тем, что в эквивалентный объект для одного из регуляторов ходит не только прямой канал собственно объекта, но и перекрестные каналы, а также замкнутый контур со вторым регулятором.

Структурная схема эквивалентного объекта для регулятора $W_{p1}(p)$ показана на рис. 2.105.

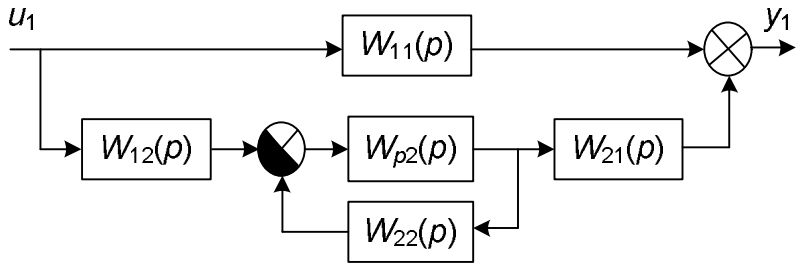


Рис. 2.105. Эквивалентный объект для первого регулятора.

Передаточная функция объекта определяется следующим выражением:

$$W_1(p) = W_{11}(p)\left[1 + k_{св}(p)W_{2зам}(p)\right],$$

где $k_{св}(p)$ – передаточная функция комплекса, характеризующего степень взаимной связи регулируемых переменных:

$$k_{св}(p) = \frac{W_{12}(p)W_{21}(p)}{W_{11}(p)W_{22}(p)},$$

$W_{2зам}(p)$ – передаточная функция замкнутой системы регулирования второго контура по каналу задания:

$$W_{2зам}(p) = \frac{W_{p2}(p)W_{22}(p)}{1 + W_{p2}(p)W_{22}(p)}.$$

Передаточная функция эквивалентного объекта для второго регулятора находится аналогично.

Комплекс взаимной связи $k_{св}(p)$ характеризует влияние перекрестных каналов на качество регулирования. Если модуль комплекса связи на нулевой частоте близок к нулю $k_{св}(j0)\approx 0$, то перекрестные связи достаточно слабы и их влиянием можно пренебречь; при этом $W_1(p) \approx W_{11}(p)$ и $W_2(p) \approx W_{22}(p)$.

Если

$$k_{св}(j\omega) = \frac{W_{12}(j0)W_{21}(j0)}{W_{11}(j0)W_{22}(j0)} = \frac{k_{12}k_{21}}{k_{11}k_{22}} > 1,$$

т.е. произведение коэффициентов усиления по перекрестным каналам больше, чем по основным, то u_1 следует использовать в качестве регулирующего воздействия для переменной y_2 и наоборот.

В любом случае, если перекрестные связи сильны, расчет регуляторов нужно производить на эквивалентные объекты. Но тут возникает проблема: чтобы рассчитать $W_{p1}(p)$ нужно знать $W_{p2}(p)$ и наоборот. Решение этой проблемы возможно только методом последовательных приближений.

На первом этапе рассчитываются настройки регуляторов $W_{p1}(p)$ и $W_{p2}(p)$ по передаточным функциям прямых каналов объекта $W_{11}(p)$ и $W_{22}(p)$. Результаты этого расчета считаются первым приближением.

Далее пересчитываются настройки регулятора $W_{p1}(p)$ для эквивалентного объекта при значениях настроек регулятора $W_{p2}(p)$, полученных в первом приближении и, «наоборот», настройки регулятора $W_{p2}(p)$ для эквивалентного объекта при значениях настроек регулятора $W_{p1}(p)$. Результаты этого расчета считаются вторым приближением.

Если значения настроек регуляторов, рассчитанные в первом и втором приближениях, отличаются не более чем на 20%, то можно принять результаты второго приближения, как окончательные. В противном случае требуется продолжить расчет, повторяя вычисления.

Система несвязного регулирования строится из условия *автономности* (независимости) одной регулируемой величины от другой.

Применение данного принципа к рассматриваемой системе равносильно применению принципа *инвариантности*, то есть независимости регулируемой переменной от изменения входной величины другого канала.

$$\begin{cases} \Delta y_1(y_2) \equiv 0; \\ \Delta y_2(y_1) \equiv 0. \end{cases} \Rightarrow \begin{cases} \Delta y_1(u_2) \equiv 0; \\ \Delta y_2(u_1) \equiv 0. \end{cases}$$

«Три черточки» означает *тождество*, т.е. *безусловное равенство* –равенство вне зависимости от значений входящих в них переменных, в отличие от *уравнения*, в котором равенство соблюдается только при определенных значениях этих переменных.

Реализация данного принципа может быть осуществлена с помощью различных структур, содержащих элементы, компенсирующие перекрестные связи объекта. Компенсация перекрестных связей позволяет автономно настраивать регуляторы, причем используя наиболее простые их структуры.

Структурная схема наиболее простого с точки зрения практической реализации варианта системы автономного регулирования (рис. 2.106).

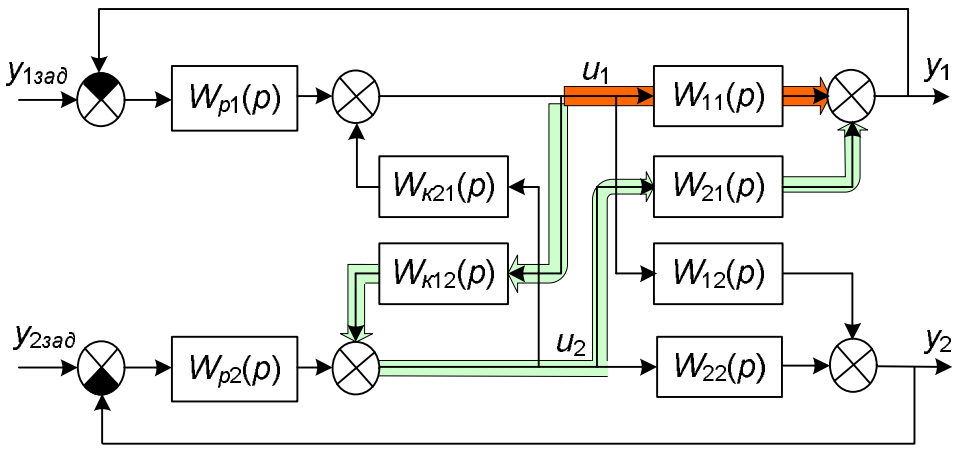


Рис. 2.106. Структура автономной двусвязной системы регулирования.

Передаточные функции компенсаторов находятся из условия инвариантности:

$$u_1\left[W_{12}(p)+W_{к12}(p)W_{22}(p)\right]=0,$$
$$u_2\left[W_{21}(p)+W_{к21}(p)W_{11}(p)\right]=0,$$

откуда:

$$W_{к12}(p)=-\frac{W_{12}(p)}{W_{22}(p)},\quad W_{к21}(p)=-\frac{W_{21}(p)}{W_{11}(p)}.$$

В общем случае для объекта с *n* входами и *n* выходами математическое описание компенсатора находится в виде *передаточной матрицы* исходя из принципа *динамической развязки каналов* [9]. Мы рассматриваем обобщенную структуру системы, элементами которой выступают не передаточные функции, а передаточные матрицы (рис. 2.107).

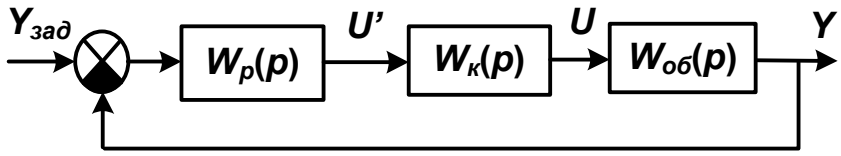


Рис. 2.107. Обобщенная структура автономной системы регулирования.

Применительно к системе, показанной на рис. 2.106, эти матрицы примут следующий вид.

Передаточная матрица объекта:

$$W_{об}(p)=\begin{pmatrix}W_{11}(p) & W_{21}(p)\\ W_{12}(p) & W_{22}(p)\end{pmatrix}.$$

Передаточная матрица регулятора:

$$W_{р}(p)=\begin{pmatrix}W_{р1}(p) & 0\\ 0 & W_{р2}(p)\end{pmatrix}.$$

Передаточная матрица компенсатора:

$$W_{к}(p)=\begin{pmatrix}\dfrac{1}{1-W_{к12}(p)W_{к21}(p)} & \dfrac{W_{к21}(p)}{1-W_{к12}(p)W_{к21}(p)}\\ \dfrac{W_{к12}(p)}{1-W_{к12}(p)W_{к21}(p)} & \dfrac{1}{1-W_{к12}(p)W_{к21}(p)}\end{pmatrix}.$$

Как мы видим, при простой, казалось бы, структуре, компенсатор имеет довольно сложное описание. Причиной тому является наличие в компенсаторе замкнутого контура с положительной обратной связью (знак минус в знаменателях передаточных функций). Это должно нас настораживать в плане устойчивости системы.

В общем случае передаточная матрица эквивалентного объекта для регулятора (т.е. по каналу $U' \to Y$):

$$W_{экв.об}(p) = W_{об}(p) W_{к}(p).$$

В результате компенсации перекрестных связей объекта мы должны получить «идеальный» *диагонализированный* эквивалентный объект (без перекрестных связей):

$$W^{ид}_{экв.об}(p) = diag W_{об}(p) = \begin{pmatrix} W_{11}(p) & 0 & \dots & 0 \\ 0 & W_{22}(p) & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & W_{nn}(p) \end{pmatrix}.$$

Из этого следует, что

$$W_{об}(p) W_{к}(p) = W^{ид}_{экв.об}(p).$$

Таким образом, передаточная матрица компенсатора может быть найдена следующим образом:

$$W_{к}(p) = W^{-1}_{об}(p) W^{ид}_{экв.об}(p).$$

Как мы видим, формула компенсатора практически совпадает с формулой для регулятора по методу динамической компенсации, рассмотренному в п. 2.2.3.2. Только там компенсацией динамики объекта занимался сам регулятор, а здесь этим делом занимается отдельное «устройство». И проблема с устойчивостью системы остается той же самой, что и в методе динамической компенсации: «плохие» объекты таким способом компенсации не поддаются. Более того, если ранее мы сразу могли отличить «плохой» объект от «хорошего», то сейчас это сделать несколько труднее (хотя «правило», приведенное в п. 2.2.3.2, и продолжает действовать).

Пусть, например, объект описывается передаточной матрицей:

$$W_{об}(p) = \begin{pmatrix} \dfrac{1}{2p+1} & \dfrac{0,6}{0,04p+1} \\ \dfrac{1,4}{0,6p+1} & \dfrac{2}{0,03p+1} \end{pmatrix}.$$

Казалось бы, «ничего не предвещает беды»: объект, безусловно, устойчив, и даже, кажется, минимально-фазовый, так как вообще не имеет нулей. Но не все так просто. Нули имеются, и это особые нули, которые называются «переходными». Нули дискретной передаточной функции $W(p)$ можно определить как полюса обратной ей передаточной функции $W^{-1}(p)$. Для матриц все – то же самое. Найдем $W^{-1}_{об}(p)$ в Matlab (последние версии пакета Control умеют находить обратные передаточные матрицы):

```
W11 = tf(1,[2 1]);
W12 = tf(1.4,[.6 1]);
W21 = tf(.6,[.04 1]);
W22 = tf(2,[.03 1]);
% Передаточная матрица объекта
```

```
Wo = [W11 W21; W12 W22];
Wom1 = Wo^-1
Wm1 =

  From input 1 to output...
       -40 s^3 - 1087 s^2 - 2200 s - 833.3
   1:  -----------------------------------
              s^2 + 177.2 s - 483.3

       1.4 s^3 + 82.37 s^2 + 1207 s + 583.3
   2:  -----------------------------------
              s^2 + 177.2 s - 483.3

  From input 2 to output...
       9 s^3 + 319.5 s^2 + 657.5 s + 250
   1:  --------------------------------
             s^2 + 177.2 s - 483.3

       -0.3 s^3 - 18 s^2 - 279.2 s - 416.7
   2:  -----------------------------------
              s^2 + 177.2 s - 483.3

Continuous-time transfer function.
```

Наша обратная передаточная матрица – матрица неустойчивой системы, поскольку коэффициенты ее характеристического полинома разного знака. Найдем нули $W_{об}(p)$ с помощью стандартной функции `zero`:

```
zero(Wo)
Warning: Use TZERO to compute the transmission zeros of a SI-
SO or MIMO model.
> In DynamicSystem/zero (line 28)

ans =

 -179.8540
    2.6874
```

Программа, конечно, «повозмущалась» тем, что мы не воспользовались специальной функцией `tzero` (transmission zeros, переходные нули), но в итоге вызвала эту функцию самостоятельно (она довольно «умная»). Но это не важно. Главное, мы видим один положительный нуль, который превратится в положительный полюс обратной передаточной матрицы. Проверим это:

```
pole(Wom1)

ans =

 -179.8540
    2.6874
 -179.8540
    2.6874
```

Неустойчивый компенсатор сделает всю систему регулирования неустойчивой. Попытка поменять каналы воздействия ни к чему не приведет, поскольку ни нули, ни полюса передаточной матрицы при этом не изменятся. Поэтому здесь необходимы другие решения, изложение которых выходит за рамки книги. По крайней мере, проверить устойчивость системы мы уже можем…

2.3. Программно-логическое управление

2.3.1. Алгоритмы программного управления

«Великая и ужасная» теория автоматического управления, по крайней мере, в том виде, в каком она изложена в большинстве учебников, занимается в основном автоматическим управлением непрерывными объектами, т.е. такими, состояние которых описывается непрерывными величинами. Это может быть температура, давление, расход и т.д. Более того, большая часть теории управления посвящена не управлению в целом, а только его части, а именно регулированию. Задачи планирования рассматриваются только в некоторых разделах, таких, например, как «Оптимальное управление». Между тем автоматизация технологических процессов далеко не исчерпывается построением систем автоматического регулирования. Существует множество процессов, где требуется автоматизировать некоторую *последовательность действий*, а управление осуществляется *по программе*, в которой переход от одного шага к другому осуществляется по некоторым условиям, включающим и *время*. Такие объекты будем называть *дискретными*, а управление ими – *программно-логическим управлением*.

Специфика дискретных технологических процессов как объектов управления состоит в следующем:

1) объект управления описывается дискретными переменными (количество, номер операции и т.д.);

2) число состояний объекта ограниченно («установка», «обработка №1» и т.д.);

3) управление объектом осуществляется с помощью «команд» (включить/выключить и др.);

4) информация о состоянии объекта формируется датчиками, оповещающими о наступлении определенных «событий» («обработка завершена», «устройство находится в заданном положении» и т.д.);

5) последовательность изменения состояний объекта часто имеет циклический характер.

Различают два типа дискретных устройств автоматики:

1) *комбинационные устройства.* Их выходные сигналы в каждый момент времени однозначно определяются входными сигналами в этот же момент времени и не зависят от того, какие значения принимали входные сигналы в предшествующие моменты времени. Другими словами, комбинационные устройства не обладают *памятью*;

2) *собственно «автоматы».* Сигналы на их выходе зависят не только от входных сигналов в данный момент времени, но и от того, какие значения

входные сигналы принимали в предшествующие моменты времени. Память автомата определяется различными внутренними состояниями, которые он может принимать под воздействием входных сигналов и сохранять их при изменении последних.

2.3.2. Комбинационные «устройства»

В качестве примера комбинационной системы рассмотрим гипотетическую систему управления освещением и вентиляцией помещения.

Функции системы:

1) автоматическое включение вентилятора, если содержание некоего опасного газа в помещении достигло определенного значения (применяется датчик «загазованности»);

2) автоматическое включение освещения при снижении освещенности помещения ниже определенного предела (применяется фотореле) в присутствие человека (имеется датчик присутствия);

3) ручное управление вентиляцией и освещением (только человеком, непосредственно находящимся в помещении).

Структура системы управления в упрощенном виде показана на рис. 2.108. Дискретные входы контроллера питаются от внутреннего источника питания и представлены на схеме «половинками» оптических пар гальванической развязки. К входам подключены датчики присутствия, загазованности и освещенности, а также переключатели, позволяющие принудительно включить или выключить вентилятор и освещение. Выходные цепи контроллера показаны упрощенно, без «промежуточной» коммутирующей аппаратуры. Сами выходы котроллера представлены контактами его выходных реле.

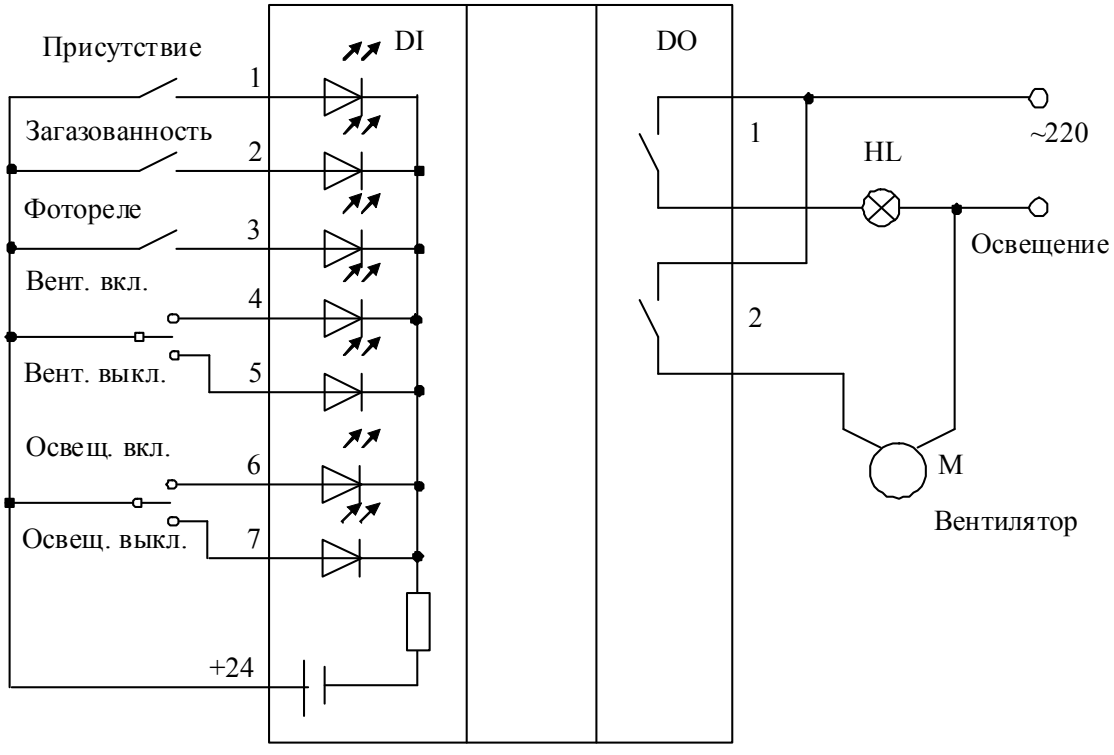


Рис. 2.108. Структура системы управления в упрощенном виде.

Введем обозначения входов и выходов котроллера (табл.2.1).

Таблица 2.1

Обозначение входов и выходов

Входы:	
I_1	Присутствие
I_2	Загазованность
I_3	Фотореле
I_4	Вентилятор включить
I_5	Вентилятор выключить
I_6	Освещение включить
I_7	Освещение выключить
Выходы:	
Q_1	Освещение
Q_2	Вентилятор

«Интуитивно» составленные алгоритмы включения освещения и вентиляции могут иметь вид:

$$Q_1 = I_1 I_6 + I_1 \overline{I}_3 \overline{I}_7,$$

$$Q_2 = \overline{I}_1 I_2 + I_1 I_2 \overline{I}_5 + I_1 I_4.$$

Вся эта «тарабарщина» означает следующее:

Освещение *нужно включить* (Q_1), если

[*человек находится в помещении* (I_1) **И** *дан приказ на включение освещения* (I_6)]

ИЛИ (+)

[*человек находится в помещении* (I_1) **И** *фотореле не сработало* ($\overline{I}_3$) **И** *нет приказа выключить освещение* ($\overline{I}_7$)].

Вентиляцию *нужно включить* (Q_2), если

[*человек не находится в помещении* ($\overline{I}_1$) **И** *помещение загазовано* (I_2)]

ИЛИ

[*человек находится в помещении* (I_1) **И** *помещение загазовано* (I_2) **И** *нет приказа выключить вентилятор* ($\overline{I}_7$)]

ИЛИ

[*человек находится в помещении* (I_1) **И** *есть приказ включить вентилятор* (I_6)].

Произведение логических переменных (*конъюнкция*) – это операция «И», сложение (*дизъюнкция*) – операция «ИЛИ».

В сложных случаях интуитивные решения могут оказаться неверными, если при их составлении не учтены все возможные комбинации, или, будучи верными, они могут обладать *избыточностью*.

Регулярное решение состоит в «правильном» составлении логических уравнений и их приведении к так называемым *минимальным нормальным формам*.

Минимальная конъюнктивная нормальная форма (МКНФ) – это конъюнкция с минимальным числом элементарных дизъюнкций с минимальным числом аргументов (либо самих, либо их отрицаний) данной функции. Иными словами, МКНФ – это минимальная реализация логической функции в виде «произведения» «сумм».

Минимальная дизъюнктивная нормальная форма (МДНФ) – это дизъюнкция с минимальным числом элементарных конъюнкций с минимальным числом аргументов (либо самих, либо их отрицаний) данной функции. Таким образом, МДНФ – это минимальная реализация логической функции в виде «суммы» «произведений». Представленные выше уравнения – попытка построить законы управления в МДНФ.

Теперь к вопросу о том, зачем нужны именно *минимальные* формы. Дело в том, что раньше логика управления реализовалась на базе элементарных логических микросхем, а еще раньше – на базе релейно-контакторных схем. Поэтому очень важно было найти наиболее «компактный» закон управления, что позволяло задействовать минимальное количество комплектующих. Сейчас вся логика реализуется программно, поэтому «компактный» закон означает «компактную» программу. В настоящее время, по крайней мере, в сфере «общепромышленных» систем управления, в большинстве случаев возможности микропроцессорных средств превосходят потребности, и требование к оптимальности кода уже не является решающим. Это, конечно, не означает, что программы можно писать «как попало».

Вернемся к задаче и попробуем решить ее «правильным образом».

1) Составим *таблицы истинности* для двух каналов управления.

Управление освещением «описано» в табл. 2.2.

Таблица 2.2

Таблица истинности для управления освещением

I_1	I_3	I_6	I_7	$\boldsymbol{Q_1}$
1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**

Входы:	
I_1	Присутствие
I_3	Фотореле
I_6	Освещение включить
I_7	Освещение выключить
Выходы:	
Q_1	Освещение

В таблице истинности сопоставляются значения входов со значениями выхода. Теоретически таблица должна содержать намного больше строк, включая все возможные комбинации входных сигналов (а их в случае четырех входов имеется $2^4 = 16$). Однако никакая теория не должна отменять здравый смысл, поэтому мы ограничились только комбинациями, имеющими отношение к решению задачи. Так, в таблице отсутствуют строки с $I_1 = 0$, поскольку заранее известно, что в отсутствие человека освещение включаться не должно. Кроме того, нет комбинаций $I_6 = I_7 = 1$, так как переключатель, подключенный к этим входам не способен сформировать такой сигнал (см. схему).

При помощи таблицы истинности составим закон управления освещением. Для этого нам потребуются только те строки, в которых $Q_1 = 1$. Объединение «внутри строки» производится с помощью операции «**И**», так как требуется, чтобы все условия одновременно выполнялись, а объединение строк – с помощью операции «**ИЛИ**», поскольку для включения освещения достаточно, чтобы возникла хотя бы одна из возможных ситуаций:

$$Q_1 = I_1\left(\overline{I}_3\overline{I}_6\overline{I}_7 + \overline{I}_3 I_6\overline{I}_7 + I_3 I_6\overline{I}_7\right).$$

То, что множитель I_1 вынесен за скобку фактически означает, что освещение включается только в присутствии человека.

Как мы видим, полученный «научным способом» закон управления освещением получился более сложным, чем тот, что сформулирован интуитивно. Преимущество его состоит в следующем: интуиция может нас и подвести, а теория при правильном применении вроде как и не должна. Рефлексия по поводу правильности или неправильности применения теории привела бы нас в такие дебри, из которых не так-то просто выбраться, поэтому будем считать, что лучше все-таки теорию применять. Тем более что эта теория дает нам возможность упростить выражение, и, применяя т.н. *законы булевой алгебры*, составить минимальную дизъюнктивную нормальную форму.

Упрощение логических выражений подобно упрощению алгебраических: мы добавляем что-то, что не изменяет истинности, группируем, сокращаем и т.д.

На первом этапе добавим в скобки слагаемое, равное второму слагаемому из исходного уравнения:

$$Q_1 = I_1\left(\overline{I}_3\overline{I}_6\overline{I}_7 + \overline{I}_3 I_6\overline{I}_7 + I_3 I_6\overline{I}_7\right) = I_1\left(\overline{I}_3\overline{I}_6\overline{I}_7 + \overline{I}_3 I_6\overline{I}_7 + \overline{I}_3 I_6\overline{I}_7 + I_3 I_6\overline{I}_7\right).$$

Здесь мы работаем практически по принципу «каши маслом не испортишь»: если добавленное произведение дает ноль («ложь»), то, конечно, это ничем не повредит уравнению, а если единица («истина») – тоже, так как в сумме с «оригиналом» все равно получится единица. У нас же не арифметическая сумма, а логическая.

Теперь можно сгруппировать слагаемые и упростить выражение:

$$Q_1 = I_1\left(\overline{I}_3\overline{I}_6\overline{I}_7 + \overline{I}_3 I_6\overline{I}_7 + \overline{I}_3 I_6\overline{I}_7 + I_3 I_6\overline{I}_7\right) =$$

$$= I_1\left(\overline{I}_3\overline{I}_7\left[\overline{I}_6 + I_6\right] + I_6\overline{I}_7\left[\overline{I}_3 + I_3\right]\right) = I_1\left(\overline{I}_3\overline{I}_7 + I_6\overline{I}_7\right).$$

Суммы в квадратных скобках равны единице («истине»). Действительно, утверждение, что «нет приказа включить освещение или есть такой приказ»

всегда истинно, так же как и утверждение «нет сигнала фотореле или есть такой сигнал».

С точки зрения теории полученное выражение упростить уже невозможно, но мы ведь знаем больше, чем теория?! Второе произведение из скобок буквально означает: «есть приказ включить освещение, и нет приказа его выключить». Однако если есть приказ включить освещение, то никак не может быть приказа его выключить, просто в силу особенности нашей схемы: у нас применяется *переключатель*. Говоря формально, комбинация $I_6 = I_7 = 1$ невозможна, поэтому

$$I_6\overline{I}_7 = I_6$$

и, следовательно,

$$Q_1 = I_1\left(\overline{I}_3\overline{I}_7 + I_6\overline{I}_7\right) = I_1\left(\overline{I}_3\overline{I}_7 + I_6\right).$$

Сравнив полученное решение с полученным ранее «интуитивным», убеждаемся, что интуиция нас все-таки не подвела (или «мы правильно применяли теорию», или «теория наконец-то согласуется с практикой»).

Аналогично построим закон управления вентилятором (табл. 2.3).

Таблица 2.3

Таблица истинности для управления вентилятором

I_1	I_2	I_4	I_5	Q_2
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	1	0	0	**1**
1	1	0	1	**0**
1	1	1	0	**1**
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	1	0	0	**1**
0	1	0	1	**1**
0	1	1	0	**1**

Входы:	
I_1	Присутствие
I_2	Загазованность
I_4	Вентилятор включить
I_5	Вентилятор выключить
Выходы:	
Q_2	Вентилятор

Уравнение, составленное по таблице истинности:

$$Q_2 = I_1\left(\overline{I}_2 I_4\overline{I}_5 + I_2\overline{I}_4\overline{I}_5 + I_2 I_4\overline{I}_5\right) + \overline{I}_1\left(I_2\overline{I}_4\overline{I}_5 + I_2\overline{I}_4 I_5 + I_2 I_4\overline{I}_5\right).$$

Используя прием, примененный выше, получим

$$Q_2 = I_1\left(\overline{I}_2 I_4 \overline{I}_5 + I_2 \overline{I}_4 \overline{I}_5 + I_2 I_4 \overline{I}_5 + I_2 I_4 \overline{I}_5\right) + \overline{I}_1 I_2\left(\overline{I}_4 \overline{I}_5 + \overline{I}_4 I_5 + I_4 \overline{I}_5\right) =$$
$$= I_1\left(I_4 \overline{I}_5\left[\overline{I}_2 + I_2\right] + I_2 \overline{I}_5\left[\overline{I}_4 + I_4\right]\right) + \overline{I}_1 I_2 = I_1\left(I_4 \overline{I}_5 + I_2 \overline{I}_5\right) + \overline{I}_1 I_2 =$$
$$= I_1 I_4 + I_1 I_2 \overline{I}_5 + \overline{I}_1 I_2.$$

Как «погибла» вторая круглая скобка? Заключенные в ней слагаемые исчерпывают все возможные варианты, поэтому в сумме дают единицу («истину»). Строго говоря, слагаемых должно быть четыре: не хватает комбинации $I_4 I_5$ (дан приказ включить вентилятор, и дан приказ выключить вентилятор). Однако наша система (см. схему), слава Богу, не допускает такой ситуации.

Составляющая $I_1 I_4$ «возникла» из-за того, что

$I_4 \overline{I}_5 = I_4$, так как комбинация $I_4 = I_5 = 1$ нереальна.

Как мы видим, и в случае с управлением вентиляцией регулярное решение совпало с интуитивным.

Далее полученные выражения используются при построении программы контроллера. Они легко реализуются на таких языках программирования ПЛК, как FBD (Function Block Diagram), LD (Ladder Diagram), STL (Statement List). Ниже приведены программы, составленные в среде CoDeSys.

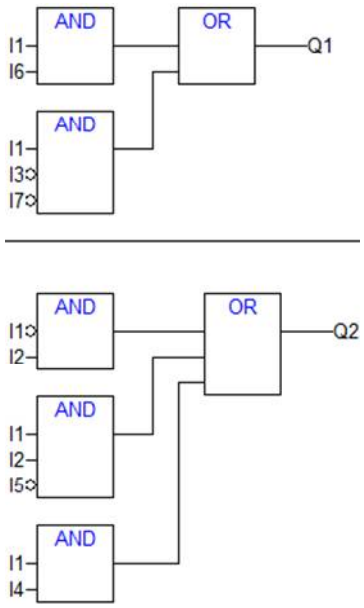


Рис. 2.109. Программа на языке FBD.

Программа на языке FBD (рис. 2.109) построена на блоках AND (И) и OR (ИЛИ). Единственный момент, который требует пояснения, – «кружочки» на некоторых входах. Это – *логическая инверсия* (отрицание) сигналов.

Программа на языке LD показана на рис. 2.110. Она очень похожа на электрическую релейную схему. В ней есть «контакты» и «катушки». Последовательное соединение контактов реализует логику И, параллельное – ИЛИ. «Ток» до катушки дойдет, если для него есть «путь» через контакты. Язык LD и был, собственно, придуман для тех, кто привык работать с электрическими схемами, чтобы облегчить им переход от «железной» реализации алгоритма управления к программной.

FBD и LD – графические языки. «Нормальные люди», конечно, будут программировать на языке текстовом. Программа на ST занимает всего две строчки:

```
Q1:=(I1 AND I6) OR (I1 AND NOT I3 AND NOT I7);
Q2:=(NOT I1 AND I2) OR (I1 AND I2 AND NOT I5) OR (I1 AND I4);
```

Программа также не требует особых пояснений: в ней используются только логические операции AND (И), OR (ИЛИ) и NOT (НЕ). Скобки в данном случае излишни и введены только для простоты понимания: операция AND имеет больший *приоритет* (т.е. выполняется раньше), чем операция OR.

Рис. 2.110. Программа на языке LD.

2.3.3. Автоматы

Разберем автоматы на конкретном примере – системе управления смесительной установкой.

2.3.3.1. Смесительная установка

На рис. 2.111, 2.112 показана простейшая смесительная установка в процессе работы. Пример взят из [16].

После запуска системы в работу открывается электромагнитный клапан Y1, и емкость начинает заполняться компонентом № 1 (фаза 1).

При достижении уровня 2 срабатывает датчик SL2, закрывается клапан Y1 и открывается клапан Y2. В емкость начинает поступать компонент № 2 (фаза 2).

После заполнения емкости до уровня 1 по сигналу от датчика SL1 закрывается клапан Y2 и включается привод мешалки M (фаза 3).

Через 15 минут он выключается, смесь готова. Для её выгрузки открывается клапан Y3. Окончание процесса фиксируется датчиком SL3 (фаза 4).

После закрытия клапана Y3 установка готова к новому циклу приготовления смеси.

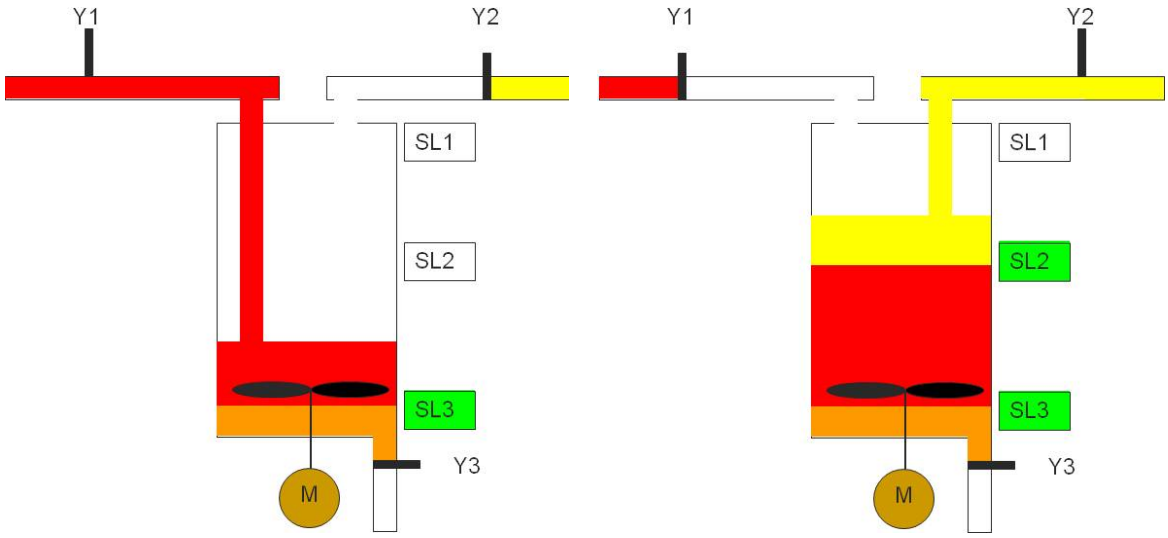


Рис.2.111. Смесительная установка, фазы работы 1, 2.

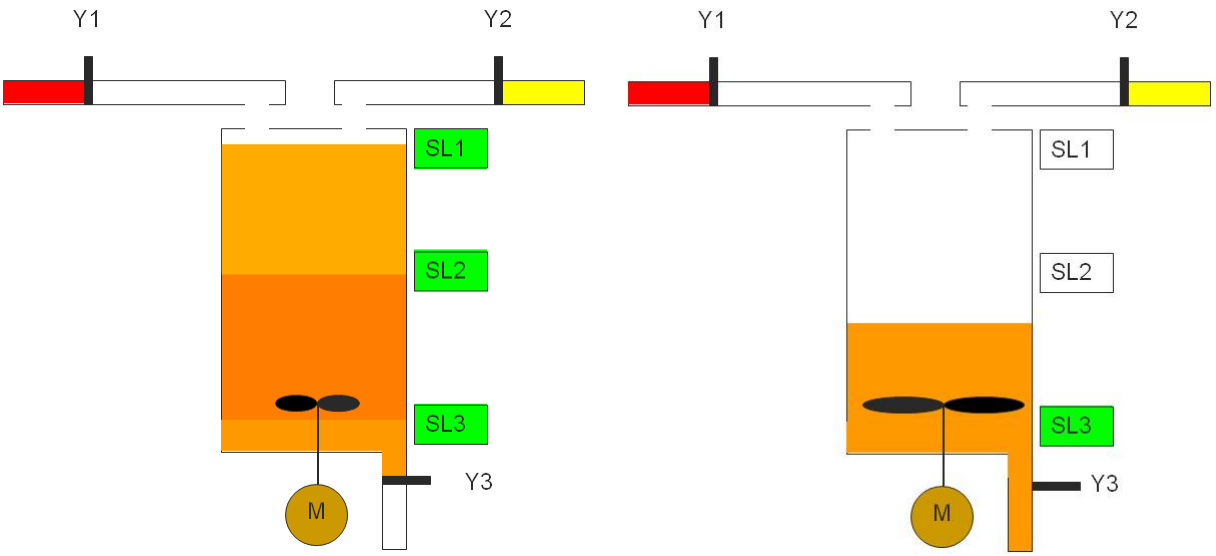


Рис. 2.112. Смесительная установка, фазы работы 3, 4.

На рис. 2.113 показана релейно-контактная схема управления процессом, взятая также из [16].

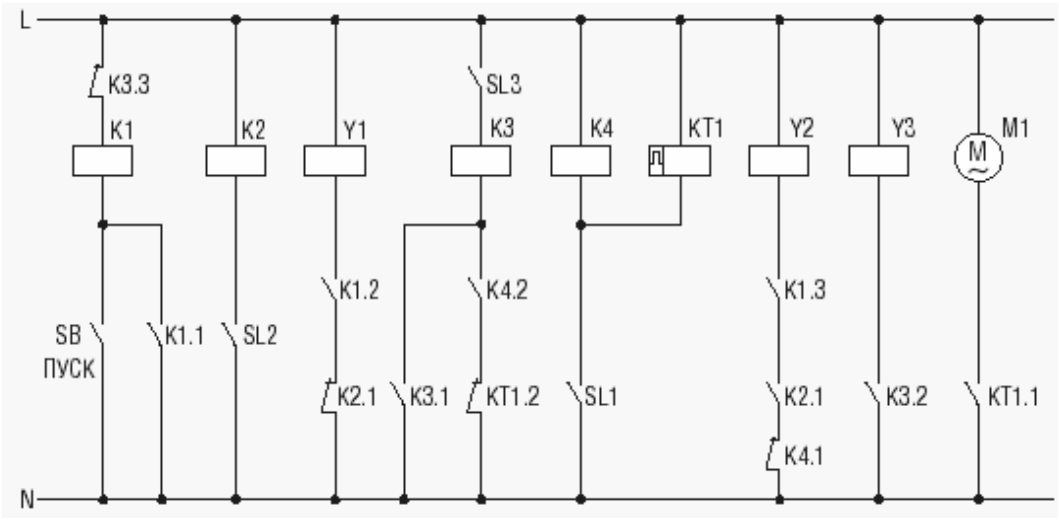


Рис. 2.113. Релейно-контактная схема управления:

Схема включает:
единственную кнопку SB «Пуск» для запуска процесса;
катушки электромагнитов клапанов Y1, Y2, Y3;
двигатель мешалки M1;
контакты датчиков уровня SL1, SL2, SL3;
четыре электромагнитных реле K1-K4;
реле времени KT1, выполняющее функцию «пульс-таймера».
Работа схемы начинается с нажатия кнопки SB «Пуск» (рис. 2.114).
Катушка реле K1 получает питание (контакт K3.3 замкнут). При этом замыкаются контакты:
K1.1, шунтируя кнопу SB «Пуск»;
K1.2, который включает электромагнитный клапан Y1;
K1.3, подготавливая цепь Y2 к работе.
Начинается подача ингредиента № 1.

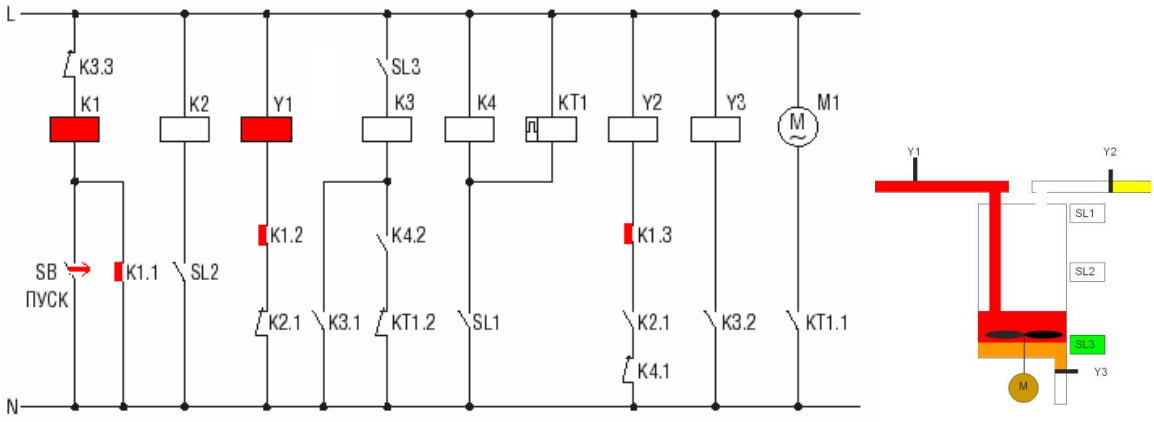


Рис. 2.114. Релейно-контактная схема управления в работе (фаза 1).

При достижении уровня 2 срабатывает датчик уровня SL2, через его контакт подается питание на катушку промежуточного реле K2 (рис. 2.115).

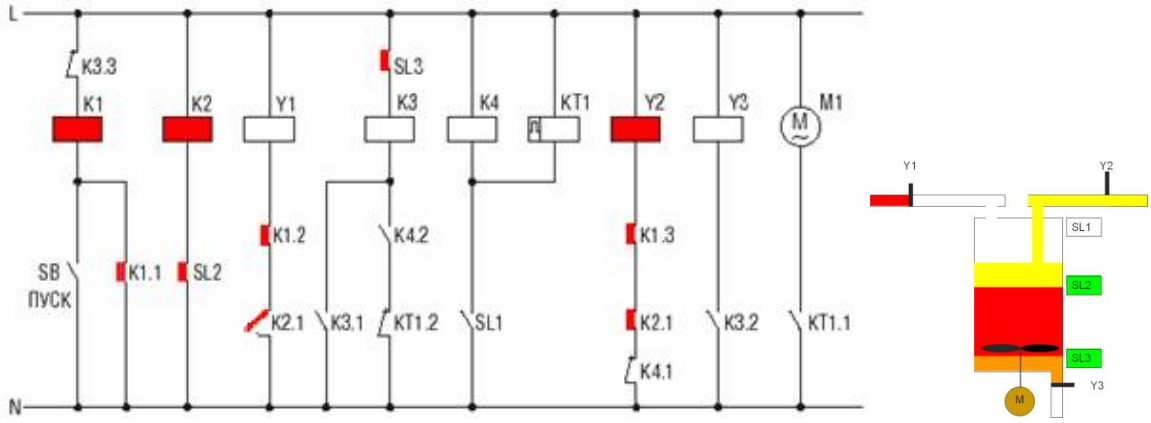


Рис. 2.115. Релейно-контактная схема управления в работе (фаза 2).

Реле K2:

размыкает нормально закрытый контакт K2.1 в цепи электромагнитного клапана Y1, последний закрывается и подача ингредиента №1 прекращается;

замыкает нормально открытый контакт в цепи электромагнитного клапана Y2, последний открывается и начинается подача ингредиента №2 (контакт K1.3 уже замкнут).

При достижении уровня 1 срабатывает датчик уровня SL1, через его контакт подается питание на катушки промежуточного реле K4 и реле времени KT1 (рис. 2.116).

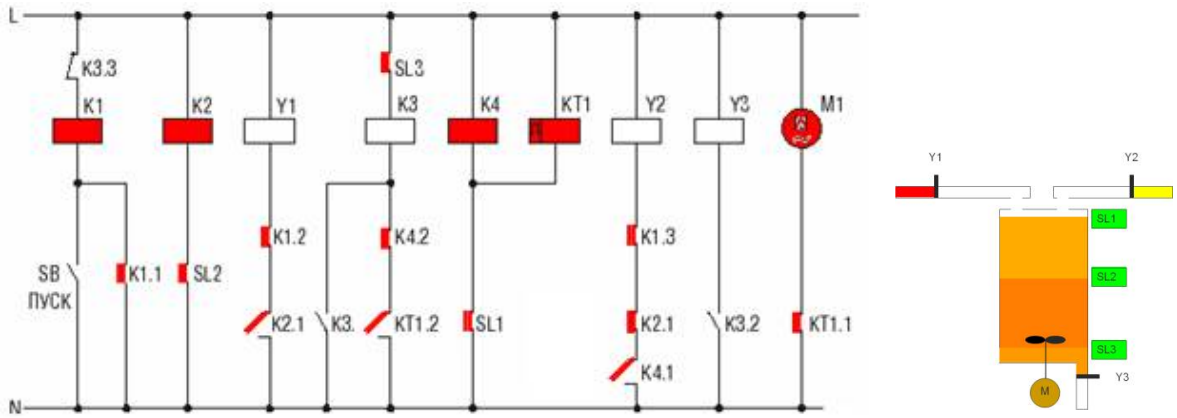


Рис. 2.116. Релейно-контактная схема управления в работе (фаза 3).

Реле К4:

размыкает нормально закрытый контакт в цепи Y2, выключая подачу ингредиента № 2;

замыкает нормально открытый контакт в цепи катушки промежуточного реле К3, подготавливая его к включению.

Реле времени КТ1, получая питание, мгновенно:

замыкает контакт КТ1.1 в цепи привода мешалки М. Мешалка включается;

размыкает контакт КТ1.2 в цепи катушки реле К3, не допуская его включения;

начинает выдержку времени.

По истечении заданного времени контакт КТ1.1 размыкается и мешалка останавливается, КТ1.2 замыкается, и на катушку реле К3 подается напряжение (рис. 2.117).

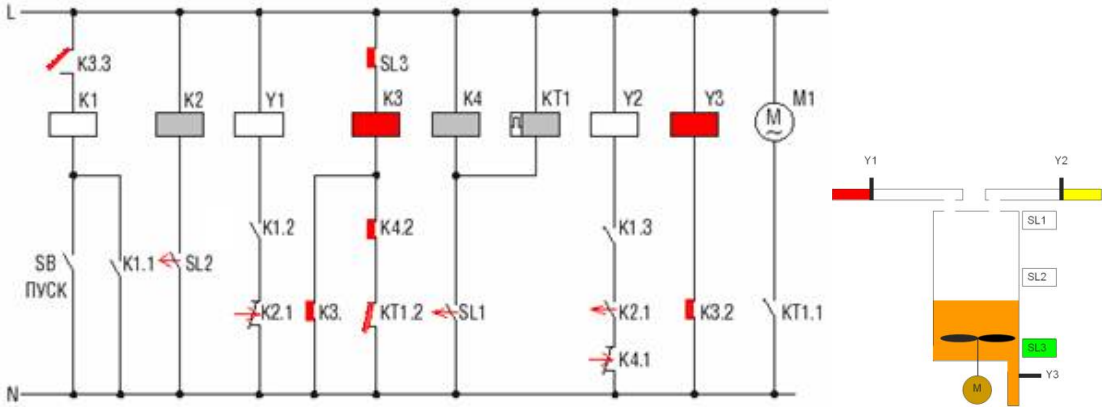


Рис.2.117. Релейно-контактная схема управления в работе (фаза 4).

Срабатывая, реле К3:

замыканием контакта К3.1 шунтирует контакт К4.2, не позволяя последнему обесточить реле при падении уровня смеси ниже уровня 1;

контактом К3.2 включает электромагнитный клапан Y3. Начинается слив смеси;

контактом К3.3 обесточивает катушку реле К1, не допуская тем самым включения клапанов Y1 и Y2 во время слива.

При достижении уровнем смеси уровня 3, датчик уровня SL3 размыкает свой контакт в цепи катушки К3, последняя теряет питание, клапан Y3 закрывается и схема приходит в первоначальное состояние (рис. 2.118).

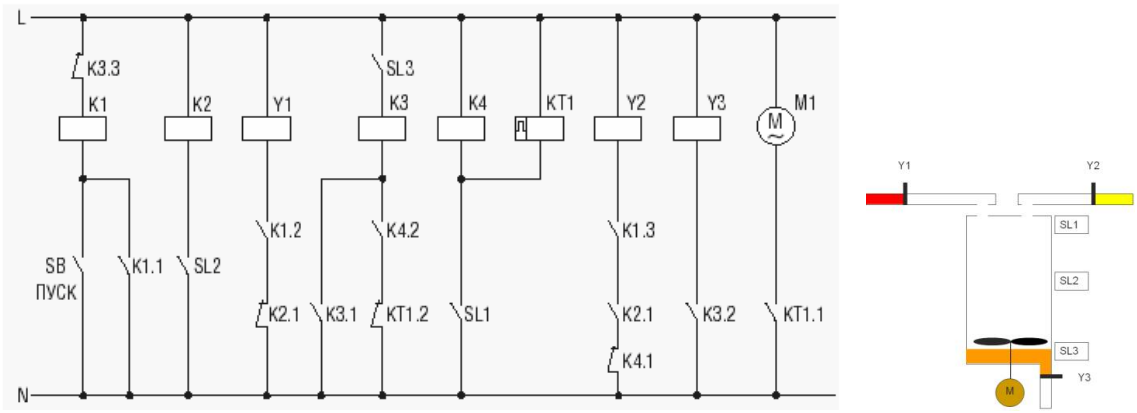


Рис. 2.118. Релейно-контактная схема управления в работе (первоначальное состояние).

На первый взгляд может показаться, что разработать такую схему очень сложно, и здесь не обойтись без применения метода «проб и ошибок». Однако на самом деле это не так, и схема составляется на вполне «регулярной» основе.

Вся последовательность действий делится на две части: приготовление и слив смеси. Кроме того, система может ожидать нажатия кнопки «Пуск» и ничего не делать (рис. 2.119).

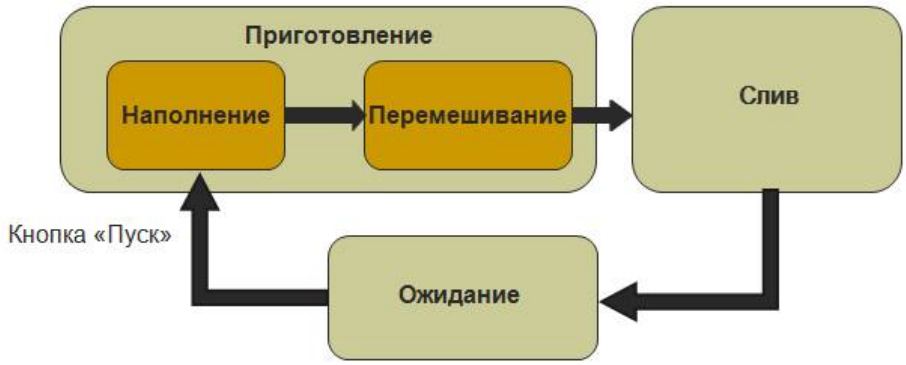


Рис. 2.119. Последовательность действий.

Таким образом, можно считать, что схема может находиться в трех ***состояниях***: *ожидания*, *приготовления* и *слива*. В каждом состоянии реализуется своя логика отклика на сигналы датчиков. В состоянии ожидания система реагирует только на кнопку «Пуск», переходя при ее нажатии в состояние приготовления. В состоянии приготовления она управляет клапанами Y1, Y2 и мешалкой M по сигналам датчиков SL1, SL2 и реле времени. По истечении выдержки времени система переходит в состояние слива. В состоянии слива она управляет клапаном Y3 по сигналу датчика SL3 и переходит в состояние ожидания, когда уровень опустится ниже отметки 3.

В идеале в каждом состоянии логика управления и логика переходов от одного состояния к другому должны быть чисто комбинаторными. Выходные сигналы должны формироваться как логические функции входных сигналов, аналогично должны быть установлены условия переходов. Однако на практике это не всегда так. И в нашем случае мы сначала отойдем от идеала, показав, как была построена данная схема, однако позже мы к нему вернемся.

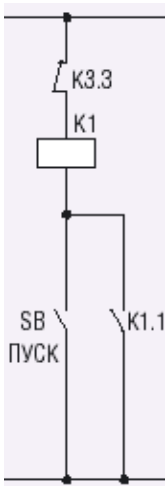
Поскольку в каждом состоянии действует логика, отличная от логики в других состояниях, система должна знать в каком состоянии она находится. Иначе говоря, система должна запоминать свое состояние. Для этой цели в схеме задействованы два реле: K1 и K3. Чем они отличаются от реле K2 и K4, можно легко понять, внимательно посмотрев на схему. Реле K1 и K3 при подаче напряжения на их катушки, формируют альтернативные цепи своего питания, поскольку их контакты установлены в цепях их же катушек. Это позволяет разделить логики включения и выключения реле: они включаются теперь при одних условиях, а выключаются при других. Например, K1 включается при нажатии кнопки «Пуск», а отключается при размыкании нормально закрытого контакта K3.3. Таким образом, реле K1 и K2 являются *элементами памяти.* Совместно со своей «обвязкой» они играют роль *триггеров*, запоминающих свое состояние. Нетрудно догадаться, за какое состояние отвечает каждое реле:

K1 – за приготовление, K3 – за слив. Одновременно они работать не могут, а в случае, когда оба реле выключены, мы имеем состояние ожидания. Кроме фиксации состояний эти реле посредством замыкания своих контактов «разрешают» работу тех цепей, которые задействованы в соответствующих состояниях.

Реле K2 и K4 ничего не запоминают и служат лишь для размножения контактов. Они напрямую управляются датчиками SL2 и SL1. Если бы эти датчики имели нужное число контактов, потребности в реле K2 и K4 не было бы вообще. Все их контакты мы бы просто заменили контактами датчиков. К сожалению, датчики чаще всего имеют один или два контакта, поэтому приходится применять промежуточные реле.

Теперь посмотрим не на то, *как работает* схема, а на то, как были сформированы ее цепи. Оказывается, что если мы определились с состояниями и реле, которые их запоминают, каждая цепь может быть прочитана (и составлена!) независимо от других.

Фаза приготовления начинается, только если закончен слив, при нажатии кнопки «Пуск», рис. 2.120.



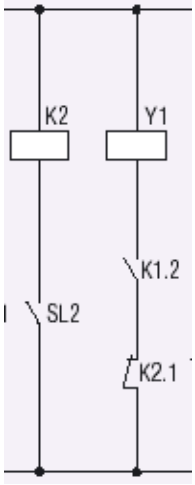
← запрет на пуск если не закончен слив (K3 – реле «слива»)

← K1 – реле «приготовления»

← шунтирование кнопки «Пуск» (запоминание состояния «Приготовление»).

Рис. 2.120. Управление реле K1.

Во время приготовления клапаны Y1 и Y2 управляются контактами SL1 и SL2, рис. 2.121, 2.122.

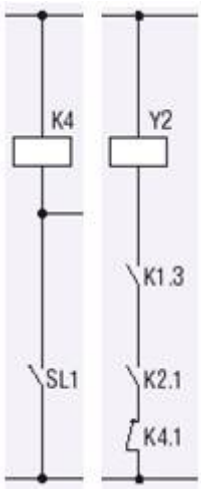


← включение клапана Y1 возможно только в режиме «Приготовление»

← выключение по достижению уровня 2.

Рис. 2.121. Управление клапаном Y1.

Во время работы клапанов Y1 и Y2 клапан Y3 обязательно закрыт.



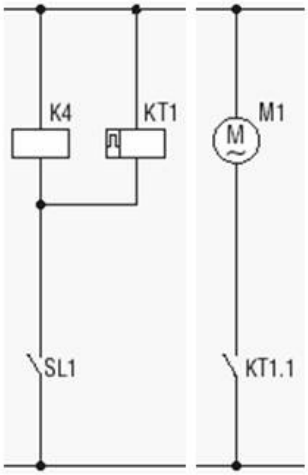
← работа клапана возможна только в режиме «Приготовление»

← включение по достижению уровня 2.

← выключение по достижению уровня 1.

Рис. 2.122. Управление клапаном Y2.

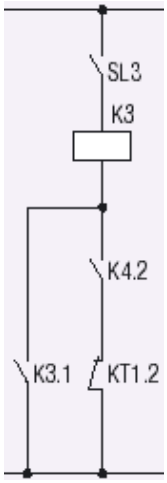
Мешалка включается после достижения уровня 1 и работает заданное время (рис. 2.123).



← включение мешалки в течение выдержки времени реле времени

Рис. 2.123. Управление мешалкой.

Фаза слива начинается после работы мешалки. В это время клапан Y3 управляется контактом SL3 (рис. 2.124, 2.125).

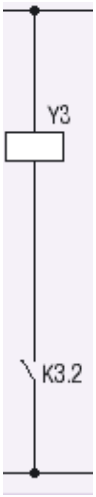


← слив продолжается пока не достигнут уровень 3 («вниз»)

← K3 – реле «слива»

← контакты K4.2 и KT1.2 замкнуты одновременно, если достигнут уровень 1 и закончена выдержка времени реле времени. Контакт K3.1 шунтирует цепь включения, удерживая реле K3 после падения уровня (фиксация режима «слив»).

Рис. 2.124. Управление реле K3.



← клапан Y3 включен только в режиме «слив».

Рис. 2.125. Управление клапаном Y3.

И вот здесь мы, как и было «обещано», и «отошли от идеала». Действительно, по какому условию должен быть выполнен переход в состояние слива? Согласно схеме это происходит, если замкнуты контакты K4.2 (уровень достиг отметки 1) и KT1.2 (реле времени или не получает питание, или закончило выдержку времени). Реле K4 и KT1 получают питание одновременно. Первое из них замыкает свой контакт, а второе – размыкает (оно его вновь замкнет по истечению выдержки времени). Если по каким-либо причинам, получив питание, реле времени «замешкается» и не успеет разомкнуть контакт, начнется слив. А ведь мешалка и не включалась! Таким образом, работоспособность схемы основана на допущении, что некоторые действия происходят «в нужной последовательности»: «сначала» должен разомкнуться KT1.2, а «потом» замкнуться K4.2.

Конечно, на практике обеспечить нужную последовательность вполне реально, применив, например, реле с разными временами срабатывания или искусственно понизив быстродействие реле K4 с помощью дополнительных элементов электрической схемы. Однако все эти меры – из области «хитроумных решений», выходящих за «теоретические рамки». Поэтому их следует по возможности избегать. Далее мы увидим, как это можно сделать.

Перейдем теперь к реализации системы на базе ПЛК. На рис. 2.216 показана схема подключения к ПЛК датчиков и исполнительных механизмов. Здесь задействован контроллер LOGO фирмы Siemens.

К входам контроллера подключены кнопка пуска и датчики уровня. Контроллер управляет приводом мешалки и тремя электромагнитными клапанами. Таймер, естественно, реализован «внутри» ПЛК.

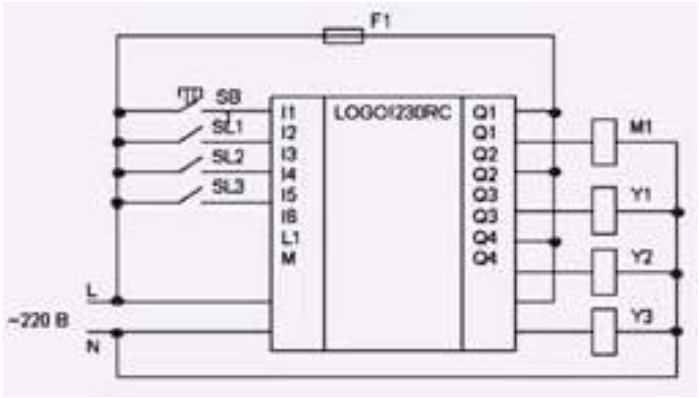


Рис. 2.126. Система управления на базе ПЛК.

Первое, что придет в голову людям, привыкшим писать программы для персонального компьютера, будет выглядеть примерно так:

```
while (!SL3) // пока уровень ниже отметки 3
{
     if (SB)     // кнопка пуск нажата
     {
          Y1 = 1;   // открытие клапана Y1
          while (!SL2); // пока уровень ниже отметки 2
          Y1 = 0;    // закрытие клапана Y1
          Y2 = 1;   // открытие клапана Y2
          while (!SL1); // пока уровень ниже отметки 1
          Y2 = 0;    // закрытие клапана Y2
          M = 1;    // включение мешалки
          delay(5);  //временная задержка
          M = 0;    // выключение мешалки
          Y3 = 1; // открытие клапана Y3
          while (SL3); // пока уровень выше отметки 3
          Y3 = 0; // закрытие клапана Y3
       }
}
```

Здесь использован язык C/C++.

На большинстве ПЛК это не будет работать вообще, и дело тут, конечно, не в языке программирования. Такая структура программы полностью противоречит самой концепции ПЛК.

Большинство операционных систем ПЛК однозадачные и работают циклически. Цикл включает:

1) считывание входных сигналов модулей УСО и их отображение в памяти в «образе процесса» – специальной области памяти, куда программа пользователя может обращаться для «общения с внешним миром»;

2) выполнение программы пользователя, вычисляющей на основе входных сигналов (переменных) и текущего состояния новые значения выходных сигналов (переменных), которые далее помещаются в «образ процесса»;

3) запись новых значений управляющих сигналов в выходные модули УСО из «образа процесса»;

4) выполнение сервисных и диагностических функций, в том числе и связь через порты или промышленную сеть с внешними системами.

В каждом цикле ОС программа пользователя должна выполняться целиком (полностью завершаться). А наша программа постоянно «висит» в цикле `while (!SL3)` и не завершается никогда. Попытка запустить такую программу скорее всего приведет к постоянным перезагрузкам ПЛК. Перезагрузки будет осуществлять *сторожевой таймер* (*watch dog*) – специальная «аппаратура», ослеживающая время выполнения программы пользователя и перезапускающая ПЛК при зависании программы.

Программа вполне бы могла работать под управлением операционной системы типа MS DOS, которая практически полностью передает контроль над машиной прикладной программе и почти не вмешивается в ее работу. И такие системы есть. Например, PC-совместимые контроллеры ADAM 5510 фирмы

Advantech с ОС ROM-DOS. Однако это скорее исключение, чем правило, поэтому далее мы будем ориентироваться на «классические» ПЛК.

Программа пользователя для «классического» ПЛК должна работать как автомат, принимающий входные сигналы, и в соответствии со своим текущим состоянием, переходить или нет в новое состояние, возможно, изменяя при этом выходные сигналы. Примерный вид такой программы на языке C/C++ приведен ниже:

```
/*state – состояние системы 0-пауза, 1-приготовление, 2-слив,
timerStart(t) - запустить таймер с уставкой t, timerStop() -
остановить/сбросить таймер, timerState() – получить выходной
сигнал таймера, 1 – продолжение выдержки времени*/
switch (state)
{
     case 0:
          timerStop();
          if (SB) {Y1 = 1; state = 1;} break;
     case 1:
          if (M==1&&!timerState())
               {M=0; Y3 = 1; state = 2;}
          else if (SL1&&!M) {timerStart(5); Y2 = 0; M = 1;}
          else if (SL2){Y1 = 0; Y2 = 1;} break;
     case 2:
          timerStop();
          if (!SL3) {Y3 = 0; state = 0;}
}
```

Программа использует т.н. switch-case технологию: в зависимости от состояния (`state`) управление передается в один из фрагментов кода (`case`) «по метке».

Прокомментируем наиболее сложный `case 1`:

`if (M==1&&!timerState)`

если мешалка работает и нет сигнала от таймера;

`{M=0; Y3 = 1; state = 2;}`

выключить мешалку, включить слив и перейти в состояние 2;

`else if (SL1&&!M) {timerStart(5); Y2 = 0; M = 1;}`

иначе если отметка 1 достигнута, а мешалка не работает, запустить таймер, выключить Y2 и запустить мешалку;

`else if (SL2){Y1 = 0; Y2 = 1;} break;`

иначе, если достигнута отметка 2, выключить Y1, включить Y2, выйти из переключателя.

Отметим одну очень важную «тонкость»: для упрощения программы проверку условий нужно проводить «от конца к началу». Так, мы сперва проверяем условие самого последнего «варианта» – мешалка работает и ее пора выключить. Если бы начали проверять с первого («достигнута отметка 2»), то нам бы пришлось усложнять условие (добавить «и не достигнута отметка 1»), иначе, очевидно, программа работала бы неправильно.

Программная реализация самым естественным образом решает «проблему» релейной схемы: «какой контакт раньше сработает?». Возможность обра-

щаться к любым переменным позволяет нам правильно оценить ситуацию. Так, в релейной схеме мешалка «неотделима» от реле времени, и если нормально закрытый контакт реле замкнут, совершенно непонятно, то ли мешалка *еще не работала*, то ли она *уже отработала*. Проблема там решалась следующим образом: если контакт K4.2 разомкнут, то мешалка *еще не работала*, а если замкнут, – она *уже отработала*. Однако при этом приходилось наделяться на правильную последовательность срабатывания контактов. А здесь никакой неоднозначности нет. Если мешалка работает, а таймер остановлен, это очевидно означает, что он *уже остановлен* и что мешалка свое *отработала* и ее надо выключать. Понятно, что устранить неопределенность в релейной схеме можно было бы с помощью дополнительного реле, контролирующего работу мешалки, но это целое реле, которое денег стоит…

Но на самом деле приведенная программа просто демонстрирует перенос «хитроумных» решений с аппаратного уровня на программный. Проблема в том, что сам автомат изначально был спроектирован не совсем правильно. Позднее мы рассмотрим правильное решение, а пока до конца разберемся с тем, что имеем.

В [16] предложена программа для контроллера LOGO фирмы Siemens (это совсем простой контроллер, поэтому его еще называют «программируемым реле»). Программа представляет собой диаграмму функцинальных блоков, каждый из которых выполняет определенное преобразование выходных сигналов в выходные (рис. 2.127).

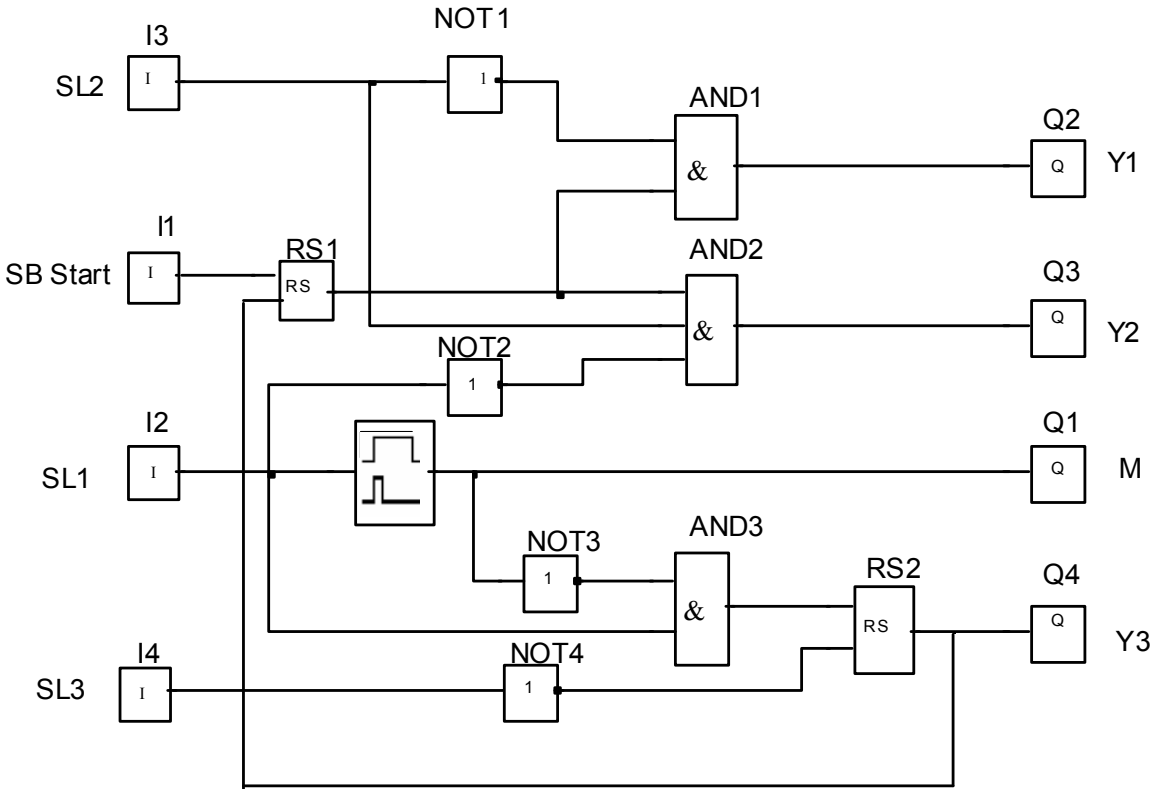


Рис. 2.127. Алгоритмическая схема программы управления.

Основными элементами диаграммы являются RS-тринггеры и интервальный таймер. Также в ней задействованы блоки ввода-вывода и логические эле-

менты. Триггеры RS1 и RS2 с приоритетом по сбросу запоминают состояния «Приготовление» и «Слив» соответственно. Интервальный таймер работает как реле времени, рассмотренное выше: срабатывает по переднему фронту входного сигнала удерживает единицу на выходе в течение заданного времени.

Рассмотрим отдельно «цепи управления» выходами и смены состояния.

Клапан Y1_включается, если

НЕ достигнут уровень датчика SL2

И

система находится в режиме приготовления (рис. 2.128).

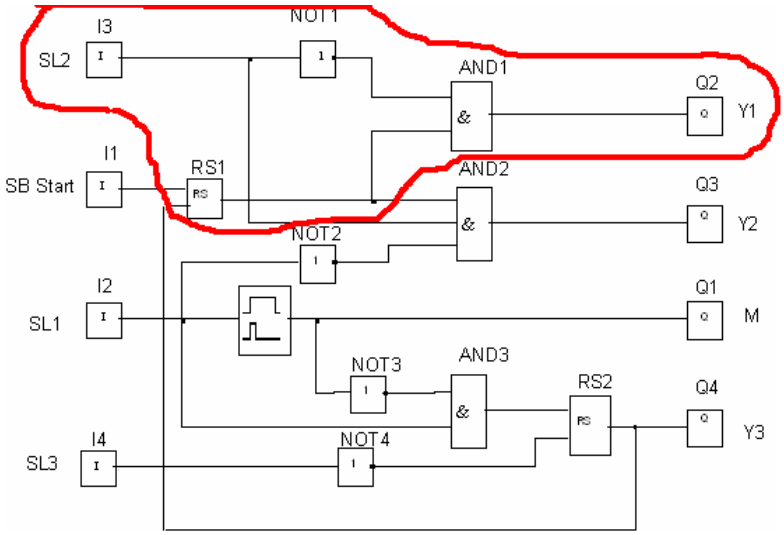


Рис. 2.128. Управление выходом Y1.

Клапан Y2 включается если

система находится в режиме приготовления

И

достигнут уровень датчика SL2

И

НЕ достигнут уровень датчика SL1 (рис. 2.129).

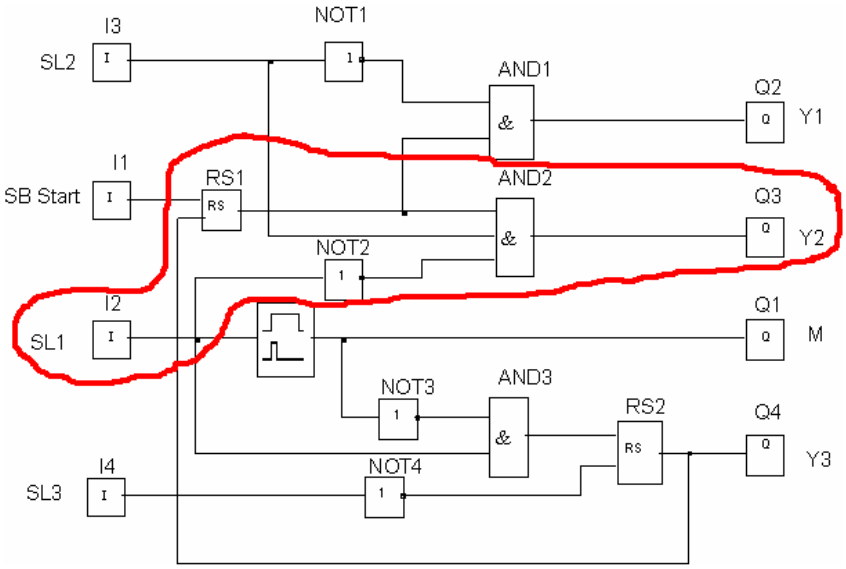


Рис. 2.129. Управление выходом Y2.

Мешалка работает в течение заданного времени (рис. 2.130).

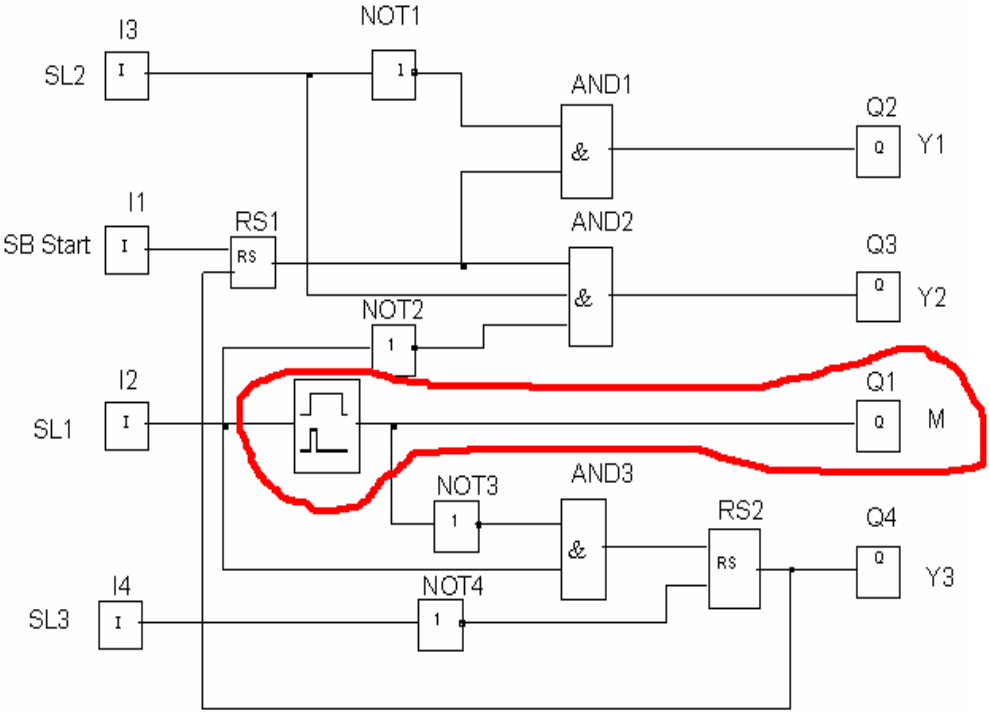


Рис. 2.130. Управление выходом М.

Клапан Y3 открыт только в режиме «слив» (рис. 2.131).

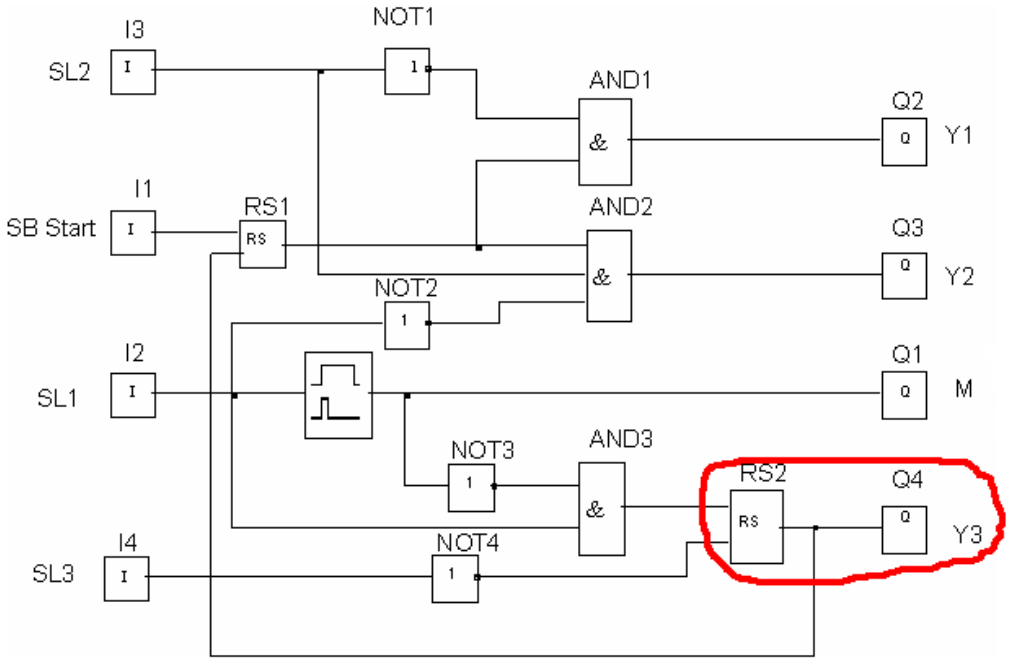


Рис. 2.131. Управление выходом Y3.

Состояние «приготовление» *устанавливается,* если
нажата кнопка «Пуск»
И
система не находится в режиме «слив» и
сбрасывается при переходе в режим «слив» (рис. 2.132).

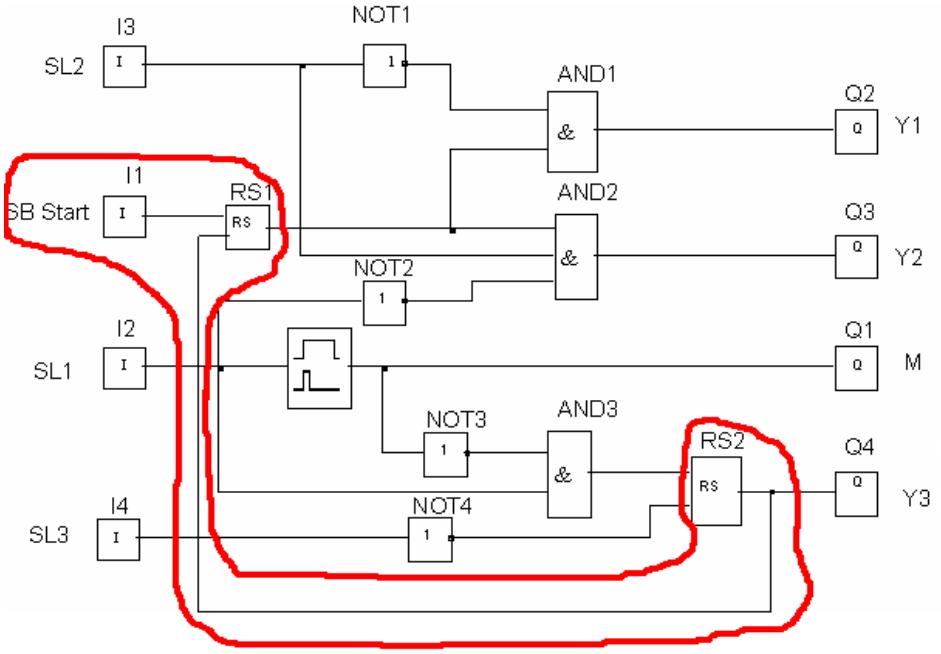


Рис. 2.132. Управление состоянием «приготовление».

Состояние «слив» *устанавливается,* если
на выходе интервального таймера – `FALSE`
И
достигнут уровень датчика SL1;
сбрасывается при опускании уровня ниже датчика SL3 (рис.2.133).

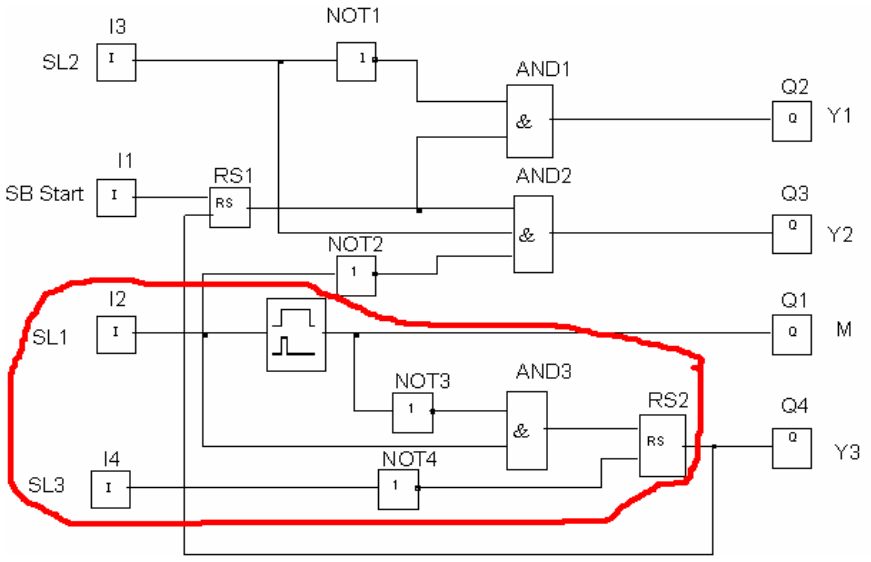


Рис. 2.133. Управление состоянием «слив».

Следует отметить, что здесь имеет значение порядок пересчета блоков. Если забыть, что перед нами программа и представить себе соответствующую электронную схему на «реальных» элементах, то ее поведение может оказаться неправильным. Действительно, в момент времени срабатывания датчика SL1 сигнал логической единицы с блока I2 подается непосредственно на второй вход блока AND3. Таймер *еще* не сработал, поэтому на первом входе блока AND3 также логическая единица. В результате сигнал подается на вход SET триггера RS2 и система переводится в режим «слив». Примерно в это же самое

время сработает таймер и включит мешалку, но слив остановить уже невозможно, и поэтому мешалка проработает совсем недолго – пока уровень не опустится ниже первой отметки.

Однако мы имеем с программой, в которой блоки пересчитываются сверху вниз и справа налево (в некоторых системах можно контролировать порядок пересчета блоков и изменять его произвольным образом). Поэтому в момент срабатывания датчика сначала будет пересчитан интервальный таймер, вследствие чего на его выходе установится единица, которая не допустит смены режима.

«Прелесть» программной реализации состоит в том, что мы имеем возможность контролировать время не только на уровне «шага» (в электронной цифровой схеме «шаг» – это интервал между тактовым импульсами, в программной реализации – интервал между вызовами программы пользователя), но и на уровне «микрошага», задавая произвольным образом порядок вычислений. Это дает возможность «обходить» разнообразные проблемы, возникающие, в частности, при неправильном проектировании алгоритмов. С другой стороны, программа усложняется, а повышение сложности создает предпосылки для новых ошибок. Идеальным было бы не использовать «микрошаг» вообще, т.е. писать программы, поведение которых не зависело бы от порядка выполнения операций, и таким образом можно было бы считать, что все они выполняются одновременно, как в электронной схеме. Именно о таких автоматах пойдет речь дальше. Но, к сожалению, этот идеал не достижим.

2.3.3.2. Синтез автоматов

Как мы уже знаем, автоматы имеют в своем составе элементы памяти, запоминающие его состояние. Сигналы на выходе автомата зависят не только от входных сигналов, но и от того, в каком состоянии он находится.

Для проектирования автоматов может использоваться *теория конечных автоматов* [17].

Согласно этой теории автоматом называют дискретный преобразователь информации, способный принимать различные состояния, переходить под воздействием входных сигналов из одного состояния в другое и выдавать выходные сигналы. Если множество состояний автомата, а также множества входных и выходных сигналов конечны, то автомат называют конечным автоматом.

Проектируя комбинационную схему, мы стремимся получить логические зависимости, связывающие входы и выходы, в самом простом их виде, для того, чтобы в электронной схеме было как можно меньше логических элементов, или чтобы получить наиболее компактный программный код (хотя это и нет так актуально по сравнению со схемой).

Теория конечных автоматов предлагает построить идеальный автомат, т.е. автомат с минимально возможным числом состояний. Для реализации такого автомата потребуется минимальное число элементов памяти, что важно и для электронной схемы по понятным причинам, и для программной реализации, поскольку на самом деле может существенно упростить программу.

Входные и выходные сигналы автомата, а также его состояния кодируются конечной совокупностью символов.

Пусть, например, на вход автомата поступает 3 сигнала, автомат формирует два выходных сигнала и может находиться в четырех состояниях. Тогда применим следующую кодировку (рис. 2.134).

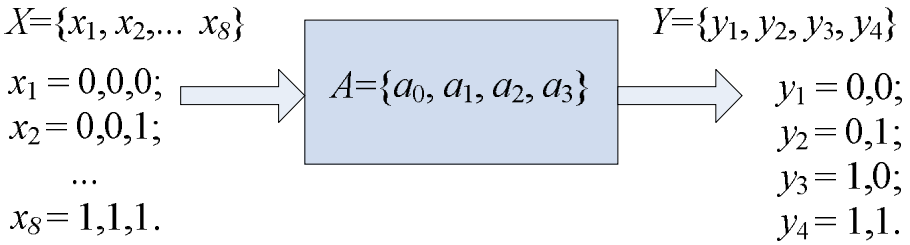


Рис. 2.134. Кодирование.

Автоматы работают «пошагово», в дискретные моменты времени, которые обычно обозначают числами t=0,1,2,...,n. В каждый момент такого дискретного времени на вход автомата поступает *один* сигнал (например, x_2), фиксируется определенное *состояние* автомата (например, a_3) и на выходе формируется один сигнал (например, y_4).

Таким образом, для описания конечного автомата, нужно задать:

1) множество возможных входных сигналов $X=\{x_1,x_2,...,x_m\}$;

2) множество возможных выходных сигналов $Y=\{y_1,y_2,...,y_k\}$;

3) множество возможных внутренних состояний автомата $A=\{a_0, a_1,...,a_n\}$;

4) *функцию переходов*, определяющую состояние автомата $a(t+1)$ в момент дискретного времени $t+1$ в зависимости от состояния автомата $a(t)$ и значения входного сигнала $X(t)$ в момент времени t;

5) *функцию выходов*, определяющую зависимость выходного сигнала автомата $y(t)$ от состояния автомата и значения входного сигнала в момент времени t.

Кроме того, обычно фиксируется одно из внутренних состояний в качестве начального, и у нас, конечно, это – a_0, не зря же состояния, в отличие от входов и выходов пронумерованы, начиная с нуля. Обычно это состояние *покоя*, *ожидания*, *бездействия*.

Существует два типа конечных автоматов: *автоматы Мили* и *автоматы Мура*. Функции переходов и выходов автомата Мили имеют вид:

$$a(t+1)=f[a(t), x(t)], \qquad y(t)=\varphi[a(t), x(t)].$$

Выходной сигнал в данном такте определяется внутренним состоянием и входным сигналом в данном такте.

Автоматы Мура отличаются тем, что выходной сигнал такого автомата в данном такте определяется только внутренним состоянием автомата в этом же такте.

$$a(t+1)=f[a(t), x(t)], \qquad y(t)= \varphi[a(t)].$$

Любой автомат Мили может быть «переделан» в эквивалентный ему автомат Мура, и наоборот, поэтому далее ограничимся только автоматами Мили.

Функции переходов и выходов обычно задаются таблицами переходов и выходов либо графами автомата.

Пусть, например, для нашего автомата таблицы имеют следующий вид, табл. 2.4, 2.5.

Таблица 2.4

Таблица переходов

	a_0	a_1	a_2	a_3
x_1	a_1	a_1	a_2	a_3
x_2	a_0	a_2	a_2	a_3
x_3	a_0	a_1	a_3	a_3
x_4	a_0	a_1	a_2	a_2
x_5	a_0	a_1	a_1	a_3
x_6	a_0	a_2	a_2	a_3
x_7	a_0	a_1	a_3	a_3
x_8	a_0	a_1	a_2	a_0

Таблица 2.5

Таблица выходов

	a_0	a_1	a_2	a_3
x_1	y_2	y_2	y_1	y_1
x_2	y_1	y_3	y_3	y_1
x_3	y_1	y_1	y_4	y_4
x_4	y_1	y_1	y_3	y_3
x_5	y_1	y_2	y_2	y_1
x_6	y_1	y_3	y_3	y_1
x_7	y_1	y_1	y_4	y_4
x_8	y_1	y_1	y_1	y_1

Граф автомата показан на рис. 2.135.

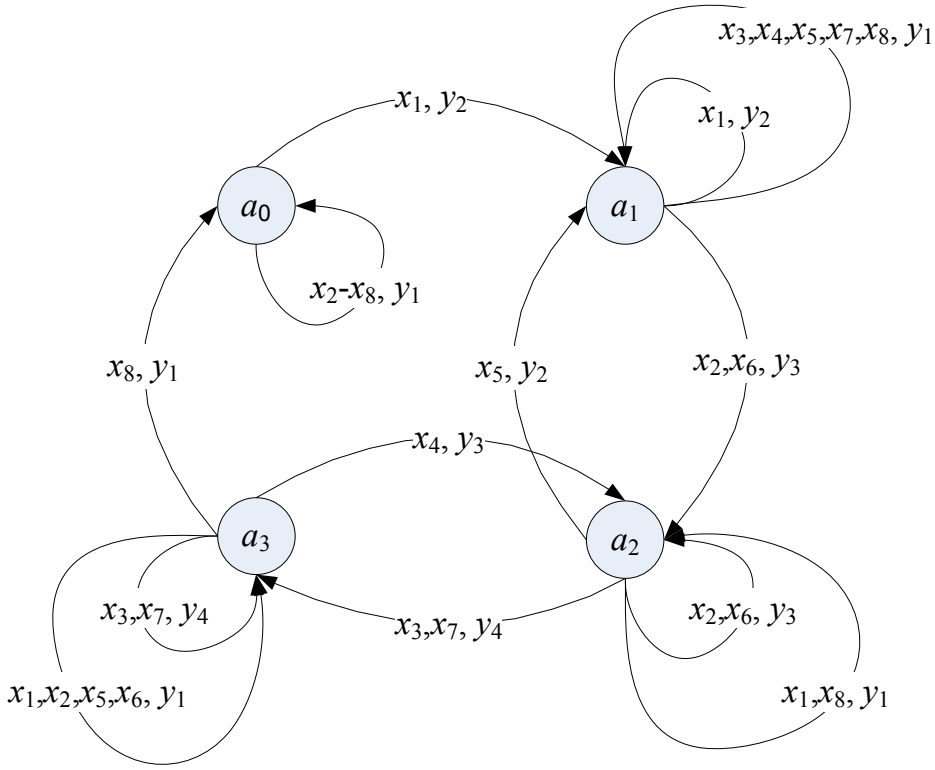


Рис. 2.135. Граф автомата.

Сопоставляя таблицы с графом, достаточно просто понять, как «это работает». Если автомат находился в начальном состоянии a_0, то подача входных сигналов в последовательности x_1, $x_2 \ldots x_8$ приводит к смене состояний и выдаче выходных сигналов соответственно в последовательностях:

$a_1,\ a_2,\ a_3, a_2,\ a_1,\ a_2,\ a_3, a_0$;
$y_2,\ y_3,\ y_4,\ y_3,\ y_2,\ y_3,\ y_4,\ y_1$.

«Возвраты» можно не совершать, если использовать «усеченную» последовательность

x_1, x_2(или x_6), x_3(или x_7), x_8.

При «неправильной» последовательности смены состояния не происходит, и на выход выдается y_1.

Если нам удалось описать технологический процесс такими таблицами ли графом, и притом мы с абсолютной уверенностью можем ответить за каждую «клеточку» или «стрелочку», то можно считать, что «дело сделано», так как программная реализация не представляет никаких затруднений. Как и всегда, сложность – не в программировании, а в алгоритмизации.

Однако, конечно, теория конечных автоматов на этом не останавливается (иначе, что это была бы за *теория*!), а предлагает оптимизировать автомат, путем исключения «лишних» состояний, которые, возможно, в нем имеются.

Поскольку в нашем автомате сомневаться не хочется, рассмотрим пример из [18]. Исходный автомат с одним входом и одним выходом имеет целых шесть состояний, что само по себе довольно подозрительно, табл. 2.6, 2.7.

Таблица 2.6

Таблица переходов

	a_0	a_1	a_2	a_3	a_4	a_5
x_1	a_0	a_0	a_4	a_1	a_0	a_0
x_2	a_1	a_2	a_3	a_2	a_3	a_3

Таблица 2.7

Таблица выходов

	a_0	a_1	a_2	a_3	a_4	a_5
x_1	y_1	y_1	y_2	y_2	y_1	y_1
x_2	y_2	y_1	y_1	y_1	y_1	y_1

На первом этапе оптимизации внимательно изучаем таблицы и обнаруживаем, что состояния a_4 и a_5 ничем не отличаются: действительно и переходы из них одни и те же, и выходные сигналы формируются одинаковые. По какому недомыслию так получилось, совершенно непонятно (на практике такая ситуация маловероятна). Поэтому от состояния a_5 можно избавиться без всяких сожалений, табл. 2.8, 2.9.

Таблица 2.8

Таблица переходов

	a_0	a_1	a_2	a_3	a_4
x_1	a_0	a_0	a_4	a_1	a_0
x_2	a_1	a_2	a_3	a_2	a_3

Таблица 2.9

Таблица выходов

	a_0	a_1	a_2	a_3	a_4
x_1	y_1	y_1	y_2	y_2	y_1
x_2	y_2	y_1	y_1	y_1	y_1

Граф автомата показан на рис. 2.136.

Далее – немного сложнее. Из таблицы выходов мы видим, что у нас есть несколько состояний, в которых выходы формируются одинаково. Сгруппируем эти состояния:

a_1, a_4 – группа z_1;

a_2, a_3 - группа z_2.

Возможно, эти состояния подлежат объединению.

Посмотрим, куда совершаются переходы из состояний, входящих в группы. Возьмем состояния группы z_1 (рис. 2.137).

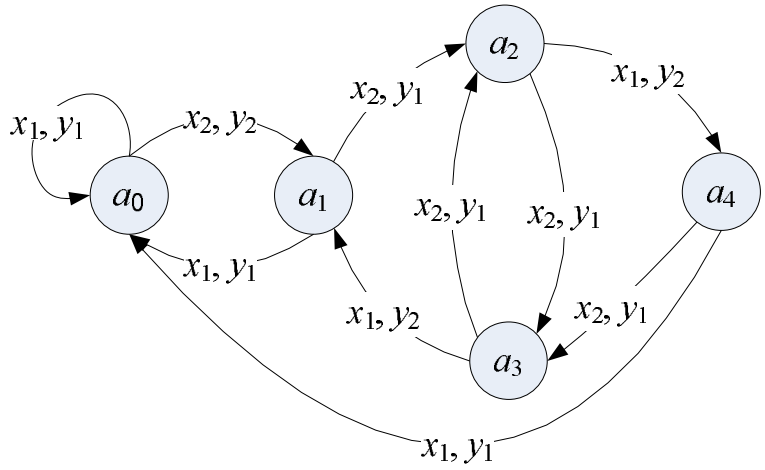


Рис. 2.136. Граф автомата.

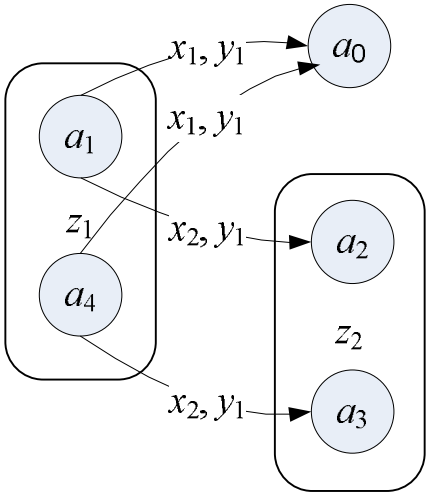


Рис. 2.137. Переходы из состояний группы z_1.

При подаче сигнала x_1 из обоих состояний переходы совершаются в одно и то же состояние a_0. При подаче сигнала x_2 переходы совершаются в разные состояния a_2 и a_3. Однако оба этих состояний принадлежат группе z_2, и в них реализована одна и та же логика управления выходами.

Таким образом, на следующем такте автомат будет выдавать выходные сигналы независимо от того, в каком состоянии он был ранее: a_1 или a_4. Можно ли уже теперь сделать вывод, что состояния a_1 и a_4 полностью эквивалентны? Нет, поскольку мы не знаем, что будет дальше. Рассмотрим развитие процесса (рис. 2.138).

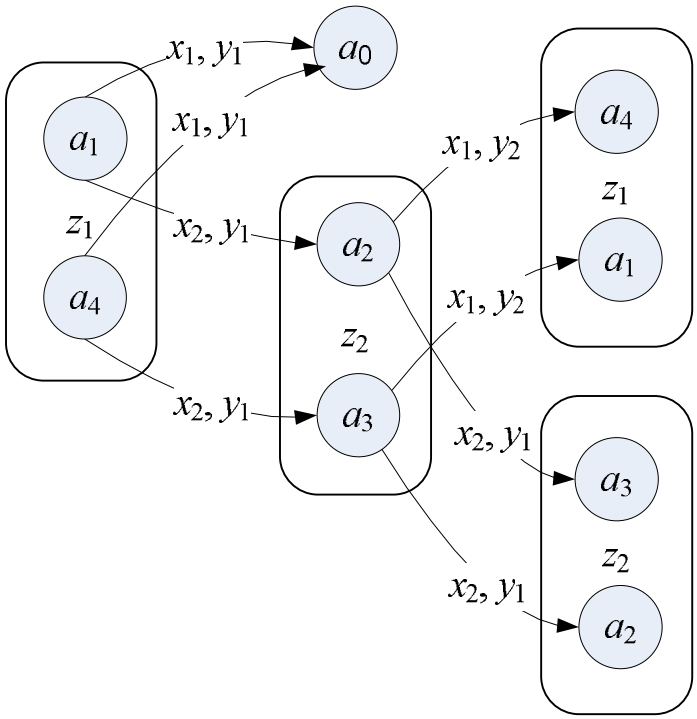


Рис. 2.138. Переходы из состояний группы z_1, продолжение.

При подаче сигнала x_1 из обоих состояний группы z_2 автомат переходит в состояния a_4 и a_1, принадлежащие группе z_1, а при подаче сигнала x_2 – в со-

стояния a_3 и a_2, принадлежащие группе z_2. Совершенно очевидно, что дальнейшие исследования бесполезны: мы просто будем ходить «по кругу» от группы z_1 к группе z_2 и обратно.

Состояния a_1 и a_4 полностью эквивалентны и могут быть заменены одним состоянием. «Попутно» выяснилось, что также эквивалентны и состояния a_2 и a_3, поскольку они тоже участвуют «в круговом движении» (в общем случае для каждой группы нужно проводить отдельные исследования).

В результате объединения состояний получим автомат, функции переходов и выходов которого показаны в табл. 2.10, 2.11.

Таблица 2.10

Таблица переходов

	a_0	a_{14}	a_{23}
x_1	a_0	a_0	a_{14}
x_2	a_1	a_{23}	a_{23}

Таблица 2.11

Таблица выходов

	a_0	a_{14}	a_{23}
x_1	y_1	y_1	y_2
x_2	y_2	y_1	y_1

Граф автомата приведен на рис. 2.139.

Этот автомат не подлежит дальнейшей минимизации, поскольку все столбцы таблицы выходов различны, а следовательно, никакие состояния уже не группируются. Но в общем случае процедура, подобная описанной выше, может выполняться несколько раз.

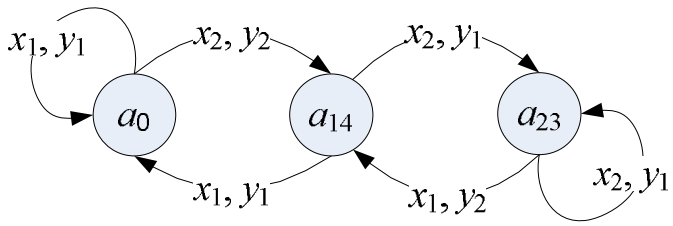


Рис. 2.139. Граф минимизированного автомата.

На самом деле нами здесь рассмотрена лишь идея, лежащая в основе минимизации числа состояний автомата. В теории конечных автоматов эта идея вылилась в многочисленные формальные методы, которые рассматривать мы не будем ввиду сложности соответствующего математического аппарата.

На практике часто мы имеем ситуацию, когда на вход автомата некоторые входные сигналы никогда не подаются (точнее, в нормальных условиях не должны подаваться). Так, например, возвращаясь к примеру со смесительной установкой, можно сказать, что комбинация, когда сработал датчик уровня на отметке 1 и не сработал датчик на отметке 2 является невозможной (на самом деле она возможна в случае выхода из строя одного из датчиков, об этом ниже). Бывает также, что определенные комбинации невозможны только в определенных состояниях, а в других возможны. Например, в состоянии ожидания вдруг ни с того ни сего сработал датчик уровня на отметке 2. Такие комбинации при синтезе автомата могут не рассматриваться: в соответствующих клетках таблиц переходов и выходов можно поставить «прочерк». В процедуре синтеза «прочерк» означает любое, какое нам нужно, значение (мы заранее считаем, что реальный автомат будет работать без ошибок со стороны аппаратуры), что, конечно, очень удобно, так как нам дает большую свободу. Как мы увидим ниже, «отловить» ошибки аппаратуры достаточно легко и «вне» автомата, поэтому можно не усложнять автомат рассмотрением ненормальных комбинаций. С

другой стороны никто не запрещает нам оставаться в рамках теории. Для этого необходимо «учредить» дополнительное состояние «авария», в которое отправлять систему при любой «неправильной» комбинации на входе. Однако это существенно усложняет сам автомат.

В целом для построения программного автомата следует:

1) «интуитивно» определить состояния, в которых может находиться система, закодировать входы и выходы и составить таблицы переходов и таблицы выходов. На этом этапе чем больше состояний, тем лучше, потому что не устраненный избыток состояний – это всего лишь «лишний» код в будущей, вполне работоспособной, системе. Недостаток же состояний может грозить полным крахом разработки или, по крайней мере, – необходимостью «внесения» в код разного рода чуждых теории «заплаток», которые рано или поздно все равно могут быть «прорваны». Как говорится, «лучше перебдеть, чем недобдеть». При первоначальном выборе состояний нужно исходить из правила: в каждом состоянии действует своя, отличная от других, логика зависимости выхода от входа. В качестве первого приближения можно считать, что каждая элементарная стадия технологического процесса есть отдельное состояние управляющего автомата;

2) попытаться все-таки провести минимизацию количества состояний по изложенной выше методике. Если за исходный автомат мы ручаемся, то правильность работы оптимизированного автомата мы можем проверить в нашей имитационной системе. В случае неудачи всегда можно вернуться к исходному варианту.

2.3.3.3. Построение автомата для смесительной установки

При проектировании автомата для управления смесительной установкой сначала будем полагать, что оборудование работает без сбоев и «запрещенных» комбинаций входных сигналов нет. Поэтому входная переменная может принимать всего шесть значений:

x_1 = **НЕ** SL3 **И** **НЕ** SB;
x_2 = **НЕ** SL3 **И** SB;
x_3 = SL3 **И НЕ** SL2;
x_4 = SL2 **И НЕ** SL1;
x_5 = SL1 **И НЕ** T;
x_6 = SL1 **И** T.

Значения выходной переменной:

y_0 = ничего не делать;
y_1 = Y1;
y_2 = Y2;
y_3 = M,T;
y_4 = Y3.

Здесь индексацию выходов мы начали с нуля, как бы подчеркивая при этом «пассивность» соответствующего выхода. Отдельными состояниями будем пока считать каждый этап работы системы:

a_0 = ожидание;
a_1 = наполнение первым ингредиентом;
a_2 = наполнение вторым ингредиентом;
a_3 = перемешивание;
a_4 = слив.

Таблицы переходов и выходов исходного автомата с «легендой» показаны на рис. 2.140.

Таблица переходов

	a_0	a_1	a_2	a_3	a_4
x_1	a_0	a_1	-	-	a_0
x_2	a_1	a_1	-	-	a_1
x_3	-	a_1	-	-	a_4
x_4	-	a_2	a_2	-	a_4
x_5	-	-	a_3	a_4	a_4
x_6	-	-	-	a_3	-

Таблица выходов

	a_0	a_1	a_2	a_3	a_4
x_1	y_0	y_1	-	-	y_0
x_2	y_1	y_1	-	-	y_1
x_3	-	y_1	-	-	y_4
x_4	-	y_2	y_2	-	y_4
x_5	-	-	y_3	y_4	y_4
x_6	-	-	-	y_3	-

x_1 = **НЕ** SL3 **И НЕ** SB;
x_2 = **НЕ** SL3 **И** SB;
x_3 = SL3 **И НЕ** SL2;
x_4 = SL2 **И НЕ** SL1;
x_5 = SL1 **И НЕ** T;
x_6 = SL1 **И** T;
y_0 = ничего не делать;
y_1 = Y1;
y_2 = Y2;
y_3 = M,T;
y_4 = Y3;
a_0 = ожидание;
a_1 = наполнение первым ингредиентом;
a_2 = наполнение вторым ингредиентом;
a_3 = перемешивание;
a_4 = слив.

Рис. 2.140. Таблицы переходов и выходов. Шаг 1.

Прочерки в таблицах говорят о том, что в соответствующих состояниях соответствующие входные сигналы подаваться не могут.

На первом этапе проведем минимизацию автомата *неправильно*, специально чтобы получить в итоге релейную схему и программу контроллера, рассмотренные ранее. Состояния a_1 и a_2 можно объединить, поскольку столбцы в таблицах совпадают (прочерки можно трактовать как любое значение). В результате получим новое состояние a_{12} («наполнение») и таблицы, показанные на рис. 2.141.

Таблица переходов

	a_0	a_{12}	a_3	a_4
x_1	a_0	a_{12}	-	a_0
x_2	a_1	a_{12}	-	a_1
x_3	-	a_{12}	-	a_4
x_4	-	a_{12}	-	a_4
x_5	-	a_3	a_4	a_4
x_6	-	-	a_3	-

Таблица выходов

	a_0	a_{12}	a_3	a_4
x_1	y_0	y_1	-	y_0
x_2	y_1	y_1	-	y_1
x_3	-	y_1	-	y_4
x_4	-	y_2	-	y_4
x_5	-	y_3	y_4	y_4
x_6	-	-	y_3	-

x_1 = **НЕ** SL3 **И НЕ** SB;
x_2 = **НЕ** SL3 **И** SB;
x_3 = SL3 **И НЕ** SL2;
x_4 = SL2 **И НЕ** SL1;
x_5 = SL1 **И НЕ** T;
x_6 = SL1 **И** T;
y_0 = ничего не делать;
y_1 = Y1;
y_2 = Y2;
y_3 = M,T;
y_4 = Y3;
a_0 = ожидание;
a_{12} = наполнение;
a_3 = перемешивание;
a_4 = слив.

Рис. 2.141. Таблицы переходов и выходов. Второй шаг.

А сейчас мы вместе с разработчиками схемы и программы сделаем небольшой шаг в сторону от теории и объединим состояния a_{12} и a_3. Теория утверждает, что объединить их нельзя, так как столбцы в таблице выходов не

совпадают, даже несмотря на прочерки. Посмотрим, что будет, если мы не послушаемся теории и все-таки объединим состояния, назвав новое состояния «приготовлением» (рис. 2.142).

Таблица переходов

	a_0	a_{123}	a_4
x_1	a_0	a_{123}	a_0
x_2	a_1	a_{123}	a_1
x_3	-	a_{123}	a_4
x_4	-	a_{123}	a_4
x_5	-	a_4	a_4
x_6	-	a_{123}	-

Таблица выходов

	a_0	a_{123}	a_4
x_1	y_0	y_1	y_0
x_2	y_1	y_1	y_1
x_3	-	y_1	y_4
x_4	-	y_2	y_4
x_5	-	y_3	y_4
x_6	-	y_3	-

x_1 = **НЕ** SL3 **И НЕ** SB;
x_2 = **НЕ** SL3 **И** SB;
x_3 = SL3 **И НЕ** SL2;
x_4 = SL2 **И НЕ** SL1;
x_5 = SL1 **И НЕ** T;
x_6 = SL1 **И** T.
y_0 = ничего не делать;
y_1 = Y1;
y_2 = Y2;
y_3 = M,T;
y_4 = Y3.
a_0 = ожидание;
a_{123} = приготовление;
a_4 = слив.

Рис. 2.142. Таблицы переходов и выходов. Третий шаг.

Выход y_3 связан с входами x_5 и x_6: при включении таймера x_5 мгновенно заменяется на x_6.

Если сначала пересчитывается таблица состояний, а потом – таблица выходов, то сначала по сигналу x_5 мы получаем смену состояний на a_4 (слив), а потом *уже в этом состоянии* формируется сигнал y_4 (включение клапана на сливе). Система работает неправильно, стадия перемешивания пропущена. Однако если сначала пересчитывается таблица выходов, а потом – таблица состояний, то сначала включается таймер и мешалка, а, следовательно, на входе автомата x_5 меняется на x_6, а потом, при пересчете состояния мы остаемся в состоянии a_{123}. Все нормально.

Таким образом, если обеспечить пересчет выходов быстрее, чем смену состояний, состояния a_{12} и a_3 действительно можно объединить. Теперь можно вспомнить и то, как мы это «обеспечивали»: в релейной схеме мы предполагали, что контакт KT1.2 сработает быстрее, чем K4.2 (см. рис. 2.124), а в программе для ПЛК таймер «автоматически» пересчитывался раньше, чем триггер RS2 (см. рис. 2.133).

Настало время соединить теорию с практикой.

Сначала решим проблему «прочерков» и запрещенных комбинаций. Реальная аппаратура работает со сбоями и ошибками. Теория говорит, что для их «отлова» нужно рассматривать все возможные сочетания входных сигналов и, возможно, вводить новые состояния и выходы.

В нашей системе мы имеем 4 *реальных* входных сигнала (3 датчика уровня плюс кнопка «Пуск»). Кроме того у нашего автомата есть еще один вход, который одновременно является и выходом, а именно сигнал таймера T – сигнал включения мешалки. Вообще говоря, работу мешалки неплохо было бы контролировать и *реально*, т.е. установить *реальный* датчик вращения мешалки, но сейчас уже поздно об этом вспоминать. Так или иначе, входных сигналов 5, и число возможных их сочетаний равно $2^5 = 32$.

Дополнительным состоянием будет состояние «Авария». Дополнительным выходом будет сигнал «Авария!».

Представляете себе соответствующие таблицы состояний и выходов?

Но, к счастью, не все так плохо. В табл. 2.12 показаны все возможные комбинации входных сигналов и выделены «запрещенные», а точнее, – *аварийные*.

Таблица 2.12

Кодировка входов полного автомата

Обозначение	T	SL1	SL2	SL3	SB
x_1	0	0	0	0	0
x_2	0	0	0	0	1
x_3	0	0	0	1	0
x_4	0	0	0	1	1
x_5	0	0	1	0	0
x_6	0	0	1	0	1
x_7	0	0	1	1	0
x_8	0	0	1	1	1
x_9	0	1	0	0	0
x_{10}	0	1	0	0	1
x_{11}	0	1	0	1	0
x_{12}	0	1	0	1	1
x_{13}	0	1	1	0	0
x_{14}	0	1	1	0	1
x_{15}	0	1	1	1	0
x_{16}	0	1	1	1	1
x_{17}	1	0	0	0	0
x_{18}	1	0	0	0	1
x_{19}	1	0	0	1	0
x_{20}	1	0	0	1	1
x_{21}	1	0	1	0	0
x_{22}	1	0	1	0	1
x_{23}	1	0	1	1	0
x_{24}	1	0	1	1	1
x_{25}	1	1	0	0	0
x_{26}	1	1	0	0	1

Обозначение	T	SL1	SL2	SL3	SB
x_{27}	1	1	0	1	0
x_{28}	1	1	0	1	1
x_{29}	1	1	1	0	0
x_{30}	1	1	1	0	1
x_{31}	1	1	1	1	0
x_{32}	1	1	1	1	1

В классических ПЛК обновление выходов производится после выполнения программы пользователя. Как говорил Штирлиц, *запоминается последнее*. Поэтому «отлов» отказов аппаратуры, перевод выходов системы в безопасное состояние и формирование сигнала «Авария» можно выполнить в конце программы. Ниже показана модификация программы управления на языке C/C++.

```
switch (state)
{
      case 0:
            timerStop();
            if (SB) {Y1 = 1; state = 1;} break;
      case 1:
            if (M==1&&!timerState())
                  {M=0; Y3 = 1; state = 2;}
            elseif (SL1&&!M) {timerStart(5); Y2 = 0; M = 1;}
            elseif (SL2){Y1 = 0; Y2 = 1;} break;
      case 2:
            timerStop();
            if (!SL3) {Y3 = 0; state = 0;}
}
if (SL2 && !SL3 || SL1 && !SL2 || SL1 && !SL3 || timerState() && !SL1 ||
timerState() && !SL2 || timerState() && !SL3)
{
      Y1 = Y2 = Y3 = M = 0; fault = 1;
}
```

Аналогично можно дополнить и алгоритмическую схему, рассмотренную в п. 2.3.3.1 (см. рис. 2.143). На схеме для экономии места показаны не все «проверки» из вышеприведенного кода. Выходные сигналы управления клапанами и мешалкой формируются с участием сигнала `fault`: при `fault=TRUE` все выходы выключаются.

Отметим, что ни в приведенном коде, ни в алгоритмической схеме состояние «Авария» отдельно не реализовано. Автомат действует сам по себе, а «защита» при срабатывании просто «перечеркивает» результаты его работы. Чем это грозит на практике? Предположим, произошел отказ оборудования. Программа выдала сигнал аварии. Ремонтная бригада устранила отказ, и программа как, ни в чем не бывало, продолжила свою работу. При этом автомат остался в том состоянии, которое было достигнуто ранее.

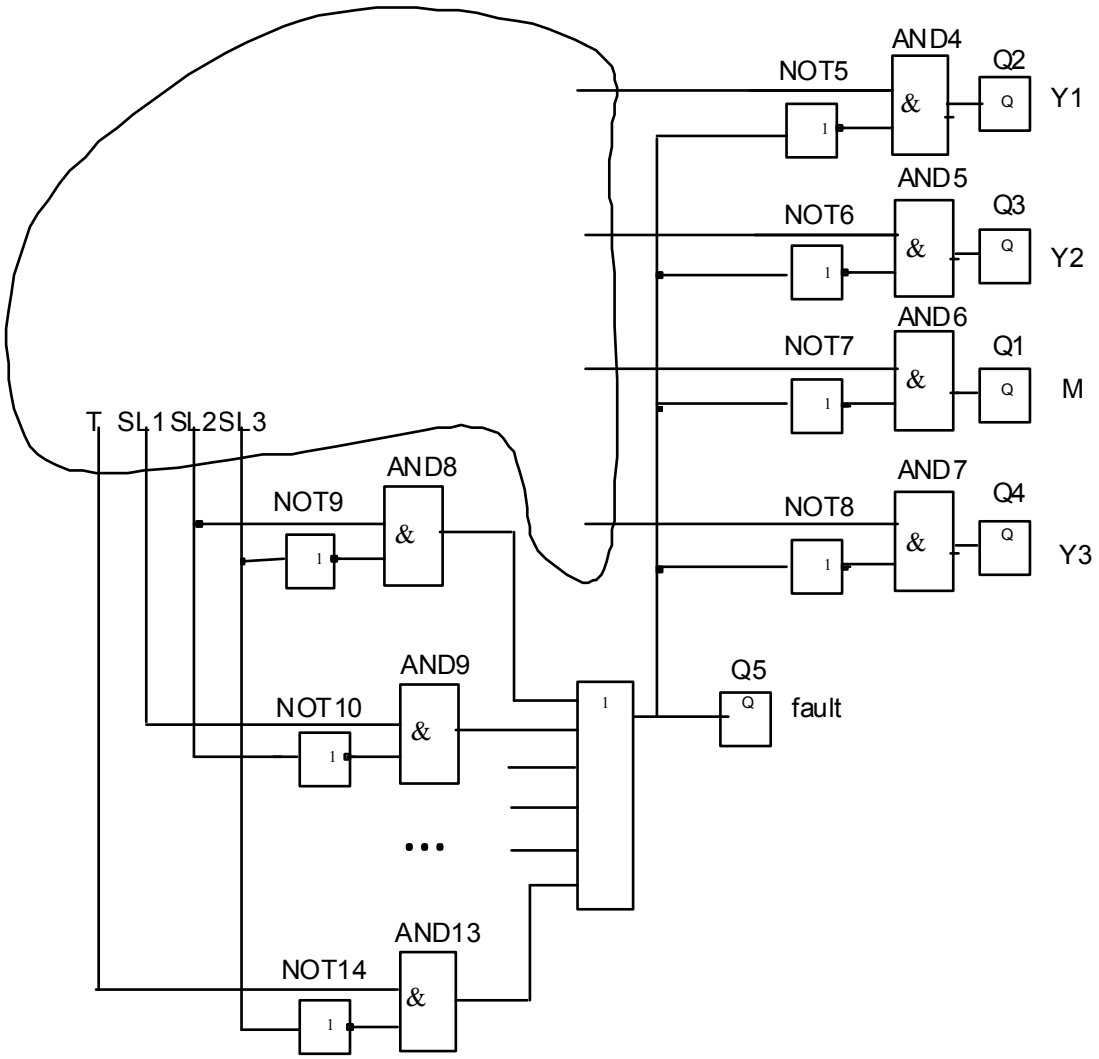


Рис. 2.143. Дополнение к алгоритмической схеме.

Правильно ли поведет себя автомат далее, в общем случае неизвестно. В результате длительного простоя, например, может испортиться продукт и тогда его нужно слить в дренаж и начинать все сначала.

Наша программа на языке C/C++ точно поведет себя неправильно. Она демонстрирует одну очень «популярную» ошибку программирования ПЛК. Пусть, например, авария произошла в состоянии «Слив». «Защита» обесточила электромагнит клапана Y3 и реальный слив прекратился. После ликвидации аварии автомат продолжил работу в состоянии «Слив», но клапан Y3 не откроется: в `case 2` это просто не предусмотрено! Команда на открытие клапана дается еще в состоянии приготовления («напоследок» перед переходом в состояние «Слив»), а в самом состоянии «Слив» может быть подана только команда на закрытие клапана (при падении уровня ниже третьей отметки). Ошибка следует из того, что мы, предполагая правильную последовательность действий, «разделили» управление по состояниям: в одном состоянии включаем то, что должно работать в другом. Приказ на включение дается в момент смены состояний.

Так делать категорически не рекомендуется. Нельзя надеяться «на прошлое». Общее правило должно быть следующим:

Управление выходами в каждом состоянии должно быть организовано независимо от того, как автомат попал в данное состояние и в какое состояние он планирует в будущем отправиться. При этом в каждом состоянии нужно управлять ***всеми*** *выходами.*

Следование этому правилу поможет избежать многочисленных проблем и упростить само программирование.

Мы сейчас не будем исправлять замеченную ошибку, потому что далее нас ждет существенная «ревизия» всего автомата. Отметим только, что алгоритмическая схема, показанная на рис. 2.143, описанной болезнью не страдает, поскольку при каждом пересчете заново формирует *все* свои выходы.

В общем случае можно утверждать, что автомату желательно иметь отдельное состояние «Авария» (а может быть даже и несколько таких состояний – в зависимости от вида или тяжести аварий) по нескольким причинам:

1) попав в это состояние, автомат получает некоторую свободу действий. Он может, например, заняться «идентификацией» аварии с выдачей сообщения оператору;

2) выходя из этого состояния (по сигналу «Сброс», который формируется обычно оператором), автомат может «спокойно» определиться (сам или, получив «подсказку» от того же оператора), в какое состояние ему следует перейти.

Автоматы без состояния «Авария» после ликвидации отказов аппаратуры в большинстве случаев требуется «сбрасывать», перезагружая ПЛК, чтобы они принудительно перешли в «безопасное» начальное состояние. Для малых систем такое решение может быть вполне приемлемым. Но если ПЛК управляет сложным процессом и в нем одновременно «действуют» десятки автоматов, перезагрузка всего ПЛК из-за одного «недотепы» недопустима.

Теперь настало время перейти к правильному синтезу и правильной реализации автомата для смесительной установки. Давайте еще раз внимательно изучим исходные таблицы переходов и выходов (рис. 2.144).

Таблица переходов

	a_0	a_1	a_2	a_3	a_4
x_1	a_0	a_1	-	-	a_0
x_2	a_1	a_1	-	-	a_1
x_3	-	a_1	-	-	a_4
x_4	-	a_2	a_2	-	a_4
x_5	-	-	a_3	a_4	a_4
x_6	-	-	-	a_3	-

Таблица выходов

	a_0	a_1	a_2	a_3	a_4
x_1	y_0	y_1	-	-	y_0
x_2	y_1	y_1	-	-	y_1
x_3	-	y_1	-	-	y_4
x_4	-	y_2	y_2	-	y_4
x_5	-	-	y_3	y_4	y_4
x_6	-	-	-	y_3	-

x_1 = **НЕ** SL3 **И НЕ** SB;
x_2 = **НЕ** SL3 **И** SB;
x_3 = SL3 **И НЕ** SL2;
x_4 = SL2 **И НЕ** SL1;
x_5 = SL1 **И НЕ** T;
x_6 = SL1 **И** T.
y_0 = ничего не делать;
y_1 = Y1;
y_2 = Y2;
y_3 = M,T;
y_4 = Y3.
a_0 = ожидание;
a_1 = наполнение первым ингредиентом;
a_2 = наполнение вторым ингредиентом;
a_3 = перемешивание;
a_4 = слив.

Рис. 2.144. Таблицы переходов и выходов. Первый шаг.

Следуя курсом, проложенным разработчиками релейной и алгоритмической схем, мы пропустили одно важное обстоятельство: оказывается, объединить можно не только состояния a_1 и a_2, но и состояния a_0, a_3 и a_4! Причем без

всяких отходов от теории: у второй группы состояний, так же как и у первой, полностью совпадают столбцы в обеих таблицах, если не принимать во внимание прочерки. В результате получаем два новых состояния:

a_{034} – «Ждать-мешать-сливать»;

a_{12} – «Наполнение».

После объединения состояний таблицы переходов и выходов примут вид, показанный на рис. 2.145. Дальнейшая минимизация автомата очевидно невозможна.

Таблица переходов

	a_{034}	a_{12}
x_1	a_{034}	a_{12}
x_2	a_{12}	a_{12}
x_3	a_{034}	a_{12}
x_4	a_{034}	a_{12}
x_5	a_{034}	a_{034}
x_6	a_{034}	-

Таблица выходов

	a_{034}	a_{12}
x_1	y_0	y_1
x_2	y_1	y_1
x_3	y_4	y_1
x_4	y_4	y_2
x_5	y_4	y_3
x_6	y_3	-

x_1 = **НЕ** SL3 **И НЕ** SB;
x_2 = **НЕ** SL3 **И** SB;
x_3 = SL3 **И НЕ** SL2;
x_4 = SL2 **И НЕ** SL1;
x_5 = SL1 **И НЕ** T;
x_6 = SL1 **И** T.
y_0 = ничего не делать;
y_1 = Y1;
y_2 = Y2;
y_3 = M,T;
y_4 = Y3.
a_{034} = «ждать-мешать-сливать»;
a_{12} = наполнение.

Рис. 2.145. Таблицы переходов и выходов. Второй шаг.

Реализация автомата на языке C/C++:

```
/*state – состояние системы 0-«ждать-мешать-сливать», 1-наполнение, 2 - авария, fault - выходной сигнал «Авария», state_prev - предыдущее состояние*/

if (SL2 && !SL3 || SL1 && !SL2 || SL1 && !SL3 || timerState() && !SL1 || timerState() && !SL2 || timerState() && !SL3)
{
     if (state!=2) state_prev = state;
     state = 2;
}
switch (state)
{
     case 0:
          Y1 = Y2 = fault = 0;
          Y3 = SL3 && !timerState();
          M = timerState();
          if (!SL3 && SB) state = 1; break;
     case 1:
          Y1 = !SL2; Y2 = SL2 && !SL1;
          timerStop(); M = fault = 0;
          if (SL1)
          {
          timerStart(5);
          M = 1;
               state = 0;
          }
          break;
```

```
    case 2:
        Y1 = Y2 = Y3 = M = 0; fault = 1; timerStop();
        if (Reset)
            state = state_prev;
}
```

Поскольку управление выходами при отказах оборудования теперь сосредоточено в отдельном состоянии «авария», «отлов» отказов происходит в начале программы. Это позволяет при авариях не выполнять «ненужного» кода, а сразу же перейти в нужную секцию. При этом в переменной `state_prev` сохраняется значение состояния на момент перехода. При выходе из аварии по сигналу `Reset` производится переход в предыдущее «рабочее» состояние. Как говорилось выше, это решение не является «универсальным»: возможно, при авариях смесь следует «забраковать».

«Развивая тему», можно сказать, что в общем случае и «отлов» отказов следует производить «внутри состояний» хотя бы потому, что в одних состояниях некоторые комбинации входных сигналов вполне допустимы, а в других запрещены. Кроме того, в общем случае желательно контролировать время протекания процессов. В нашем примере было бы неплохо, если бы аварийный сигнал формировался, в частности, когда процесс наполнения слишком затянулся. Однако, как говорил К. Прутков, «нельзя объять необъятное». Поэтому пока остановимся на достигнутом. Более «продвинутые» системы будет рассмотрены в следующей главе.

2.3.3.4. Реализация автомата для смесительной установки на языках программирования ПЛК

Автомат может быть «написан» на любых языках программирования ПЛК стандарта МЭК 61131-3. Мы рассмотрим специфику реализации автомата на текстовом языке ST и двух графических: LD и SFC.

Создадим в CoDeSys проект для контроллера «None», т.е. будем работать только в режиме эмуляции (контроллер «None» в других режимах работать и не умеет). Для отладки программ создадим простенький экран визуализации (рис. 2.146).

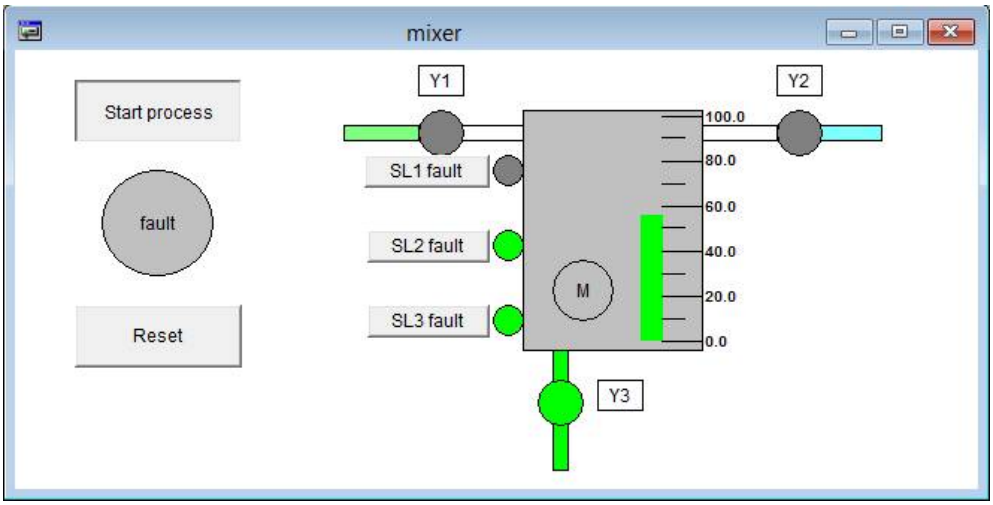


Рис. 2.146. Экран визуализации в работе.

Список глобальных переменных контроллера:

```
VAR_GLOBAL
     Start_process:BOOL:=FALSE;
     Y1,Y2,Y3,M,SL1,SL2,SL3, fault:BOOL:=FALSE;
     SL1_fault, SL2_fault,SL3_fault, Reset:BOOL:=FALSE;
     level: REAL:=0;
END_VAR
```

Переменная `Start_process` привязана к соответствующей кнопке экрана. Здесь мы внесли некоторое «новаторство», это кнопка с фиксацией (по сути – переключатель), что позволит нам заставить «установку» работать циклически. Переменные `Y1,Y2,Y3` управляют цветом трубопроводов и кружочков, символизирующих клапаны, `SL1,SL2,SL3` – кружочков, символизирующих датчики уровня. С помощью кнопок с фиксацией SL1_fault, SL2_fault, SL3_fault мы будем «вредить» автомату, вводя аварии датчиков путем установки соответствующих переменных. Кнопка Reset без фиксации, как ни странно, управляет переменной `Reset`. Переменная `level` управляет шкалой. Как видите, все очень просто и незатейливо.

Ниже приведен листинг программы `Object`, моделирующей объект управления:

```
PROGRAM Object
VAR_INPUT
END_VAR
VAR
     level_speed:INT;
END_VAR
-----------------------------------------------------------
level_speed:=BOOL_TO_INT(Y1) + BOOL_TO_INT(Y2) -
             BOOL_TO_INT(Y3);

level:=level+level_speed*0.4;
IF level<0 THEN level:=0;
ELSIF level>100 THEN level:=100;
END_IF

IF level>79 THEN
     SL1:=NOT SL1_fault;
ELSE
     SL1:=SL1_fault;
END_IF
IF level>40 THEN
     SL2:= NOT SL2_fault;
ELSE
     SL2:=SL2_fault;
END_IF
IF level>7 THEN
     SL3:=NOT SL3_fault;
ELSE
     SL3:=SL3_fault;
END_IF
```

Задача программы – рассчитать уровень смеси в установке и сформировать дискретные сигналы датчиков. Для этого в программе по сигналам управления клапанами формируется значение локальной переменной `level_speed` (скорость изменения уровня). Если все работает правильно, это значение может быть равно 0 (ни один клапан не открыт), 1 (открыт один из клапанов на притоке) или –1 (открыт клапан на стоке). Если «что-то пошло не так», возможны и другие варианты, например, 2 (открыты оба клапана на притоке). Объект управления должен быть готов ко всему. Само значение уровня (глобальной переменной `level`) получается путем прибавления к нему же самому значения `level_speed`, помноженного на коэффициент, который был экспериментально подобран так, чтобы уровень изменялся не слишком быстро, но и не слишком медленно. Получается что-то вроде численного интегрирования.

Срабатывание датчиков `SL1, SL2, SL3` происходит на отметках 79, 40, и 7 соответственно. При этом в случае отсутствия отказов, фиксируемых в переменных `SL1_fault, SL2_fault, SL3_fault` срабатывание датчиков будет «правильным», а при их наличии – «неправильным», т.е. *инверсным*.

Главная программа `PLC_PRG` ничего не делает, кроме как вызывает программу `Object` и одну из программ управления `control_st`, `control_ld` или `control_sfc`, написанных на разных языках:

```
Object;
control_st;
(*control_sfc;*)
(*control_ld;*)
```

Рассмотрим программу `control_st`:

```
PROGRAM control_st
VAR
     state:BYTE:=0;
     T:TP;
     state_prev: BYTE;
END_VAR
-----------------------------------------------------------------
IF (SL2 AND NOT SL3) OR (SL1 AND NOT SL2) OR (SL1 AND NOT SL3) OR
(M AND NOT SL1) OR (M AND NOT SL2) OR (M AND NOT SL3) THEN
     IF (state<>2) THEN
          state_prev := state;
          state := 2;
     END_IF
END_IF

CASE state OF
0:
          Y1 := Y2 := fault := FALSE;
          T(IN:=SL1, PT:=t#5s);
          M := T.Q;
          Y3 := SL3 AND  NOT M;
          IF (NOT SL3 AND Start_process) THEN
               state := 1;
          END_IF
```

```
1:
          Y1 := NOT SL2; Y2 := SL2 AND NOT SL1;
          T(IN:=FALSE); M := fault := FALSE;
          IF SL1 THEN
               T(IN:=TRUE, PT:=t#5s);
               M:=TRUE;
               state:=0;
          END_IF
2:
          Y1 := Y2 := Y3 := M := FALSE; fault := TRUE;
          T(IN:=FALSE);
          IF Reset THEN state := state_prev; END_IF
END_CASE
```

Если не обращать внимание на синтаксис, то она очень похожа на программу на языке C/C++. Отличие состоит только в управлении таймером. В CoDeSys таймер – это экземпляр библиотечного функционального блока, который *сам по себе где-то там* не работает, – его нужно *вызывать*, чтобы он пересчитал свои выходы. Есть также определенная проблема с реализацией таймера конкретного типа `TP` (пульс-таймера). В отличие от «нормального», таймера типа `TON` (с задержкой включения), `TP`-таймер не сбрасывается при «выключении», а продолжает формировать свой импульс. Из-за этого программа, как и две остальные, в следующей ситуации ведет себя не так, как задумывалось.

Если при перемешивании возникает авария, мешалка, конечно, будет выключена, но таймер продолжит считать время. Если сбросить аварию до истечении выдержки, новый отчет времени не начнется, и смесь останется «недомешанной». Остается надеяться, что на восстановление системы потребуется больше времени, чем нужно для перемешивания компонентов (или, по крайней мере, в инструкции нужно написать, чтобы быстро не перезапускали).

Программа управления на языке LD в целом повторяет логику программы на ST. Основное отличие состоит в типах данных переменных. В LD удобнее оперировать битовыми переменными, поэтому вместо переменной `state` типа `BYTE` будем использовать несколько переменных типа `BOOL`. Секция объявлений программы:

```
PROGRAM control_ld
VAR
     s0:BOOL:=TRUE;
     s1,s2,s0p,sp,a:BOOL;
     T:TP;
END_VAR
```

Здесь `s0` – бит, установленный в состоянии «ждать-мешать-сливать», `s1` – бит, установленный в состоянии «наполнение», а `s2` – бит, установленный в состоянии «авария». Переменная `sp` хранит предыдущее рабочее состояние (0 или 1), `a` – вспомогательная переменная, `T` – по-прежнему таймер типа `TP`.

Конечно, для запоминания трех состояний хватило бы и двух битов, мы могли бы закодировать состояния, например, следующим образом:

0,0 – «ждать-мешать-сливать»;

0,1 – «наполнение»;

1,0 – «авария».

Однако при такой схеме нам потребовались бы дополнительные схемы «раскодирования», что усложнило бы программу и ухудшило ее читаемость. В то же время для запоминания предыдущего рабочего состояния задействован только один бит `sp` (0 – «ждать-мешать-сливать», 1 – «наполнение»).

LD-диаграмма приведена на рис. 2.147, 2.148.

Кратко прокомментируем «цепи» программы.

В первой цепи проверяются все условия отказов и в случае, если отказ произошел и система еще не находится в состоянии «авария», на один такт работы контроллера устанавливается признак аварии (бит `a`). На этом такте с помощью катушек S (set) и R (reset) запоминается предыдущее состояние (цепи 2,3) и производится перевод системы в состояние «авария» (цепь 4).

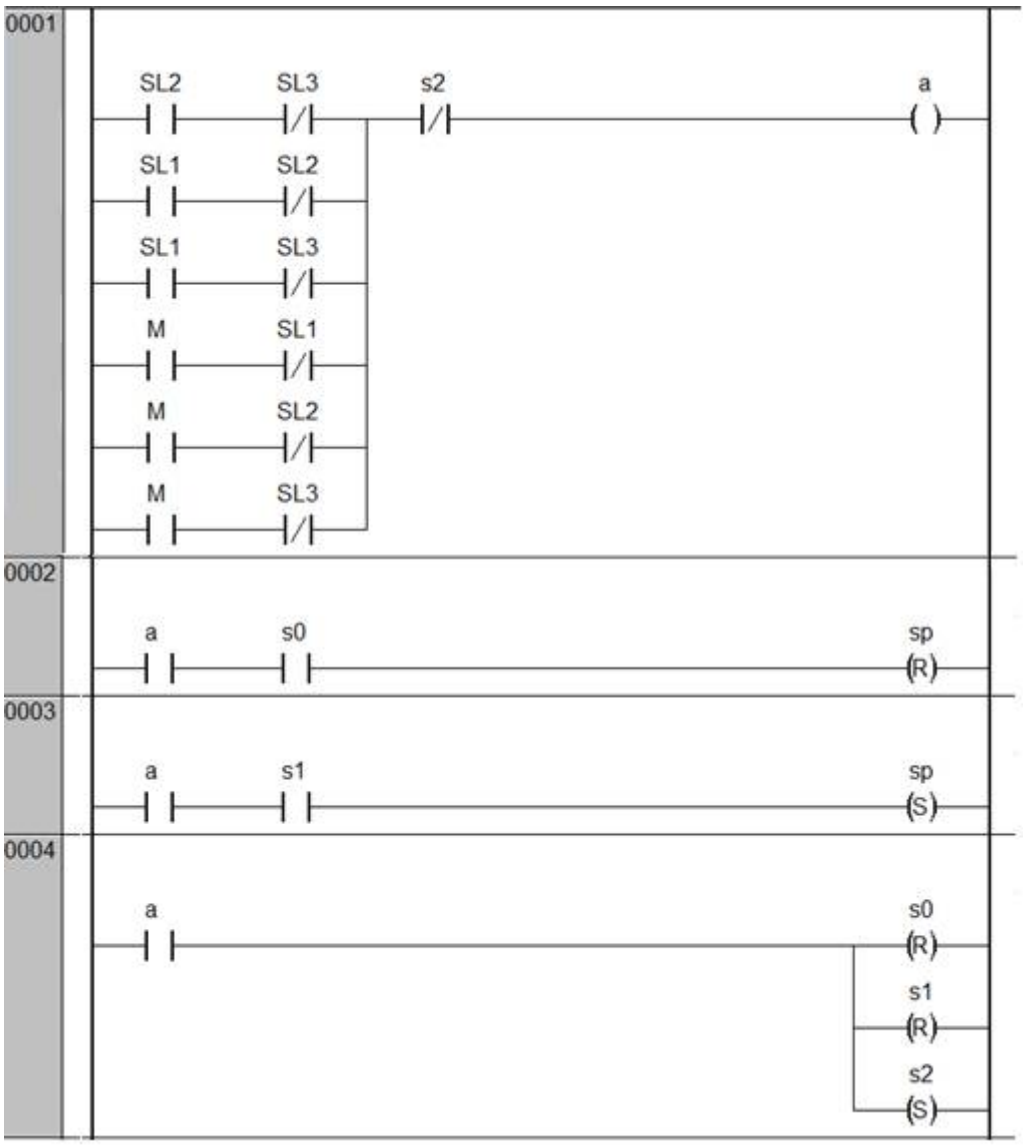


Рис. 2.147. Программа `control_ld`, фрагмент 1.

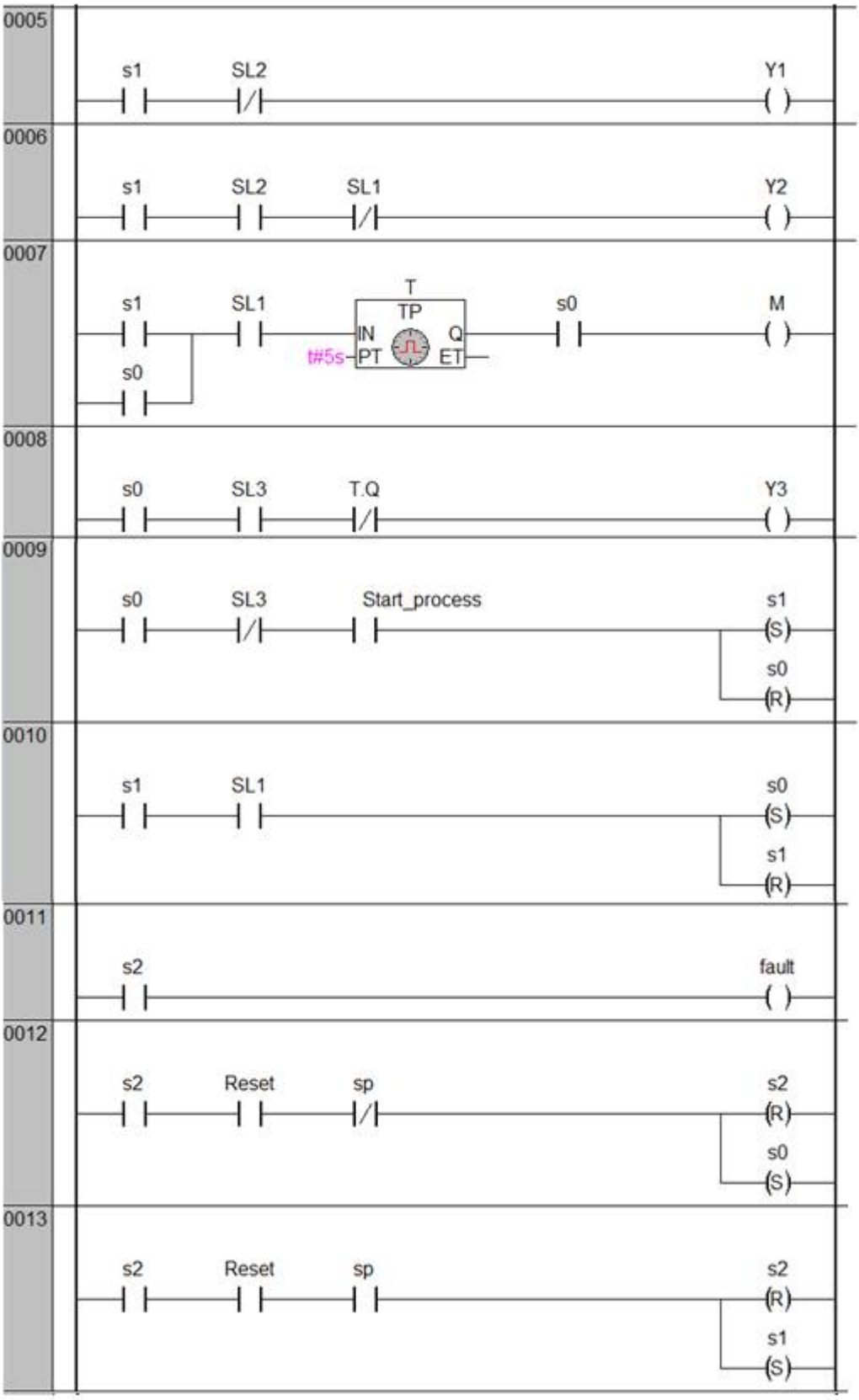


Рис. 2.148. Программа `control_ld`, фрагмент 2.

В цепях 5,6 формируется управление клапанами Y1 и Y2. Клапаны должны включаться только в состоянии «наполнение», поэтому цепи содержат нормально открытый контакт `s1`.

В цепи 7 запускается таймер и мешалка. Как было сказано выше, в CoDeSys таймер типа `TP` продолжает отчет времени и после снятия сигнала с входа `IN`. Если жестко увязать выход таймера с включением мешалки, то в состоянии «авария» мешалка продолжит свою работу. Чтобы это предотвратить, на выходе таймера установлен нормально открытый контакт `s0`.

Цепь 8 управляет клапаном Y3. Открытие клапана возможно только в режиме `s0` при уровне выше отметки 3 и после того, как закончилась выдержка времени. Ненадолго остановимся на этой цепи. Предположим, отказ случился во время перемешивания. Система перешла в состояние «авария», но таймер продолжил отсчет времени. Пока шел ремонт, таймер досчитал до предустановленного значения и сбросил свой выход `Q`. Согласно логике цепи после перехода обратно, в режим `s0`, мгновенно должен начаться слив. Однако перед пересчетом цепи 8 контроллер пересчитывал цепь 7 и вновь запустил таймер. Выдержка времени началась заново, а выход таймера `Q` установился в TRUE, поэтому слив вновь отложен. Аналогичным образом обстоит дело и в программе на ST: пересчет таймера осуществляется раньше, чем управление клапаном. Таким образом, мы опять используем «микрошаг», хотя вроде бы решили от него отказаться. Вся проблема состоит в том, что на этапе проектировании автомата никакого состояния «авария» предусмотрено не было. Мы его искусственно внесли в систему уже после проектирования. Автомат без состояния «авария» работает без «микрошага»: порядок операций в любом состоянии можно изменять произвольным образом. Но как только мы отошли от теории и занялись «самодеятельностью», потребовались и «нестандартные» решения.

В цепях 9 и 10 выполняются переходы из состояния `s0` в состояние `s1` и, соответственно, наоборот, из `s1` в `s0`. В цепи 11 производится включение сигнала «Авария!» в состоянии `s2`.

В цепях 12 и 13 производится переход из состояния `s2` в предыдущее рабочее состояние, «номер» которого сохранился в переменной `sp`. Попробуем исправить замеченную ранее «ошибку»: если в состоянии «авария» таймер не досчитал до предустановленного значения, при возврате в состояние «ждать-мешать-сливать» мешалка отработает меньше положенного ей времени (ведь пока ремонтировали, она не работала). Для этого достаточно запретить переход в режим `s0` при «активном» таймере, пусть это и будет некоторой «грубостью» по отношению к оператору, нажавшему кнопку Reset (рис. 2.149). Ничего страшного, «пропишем в инструкции».

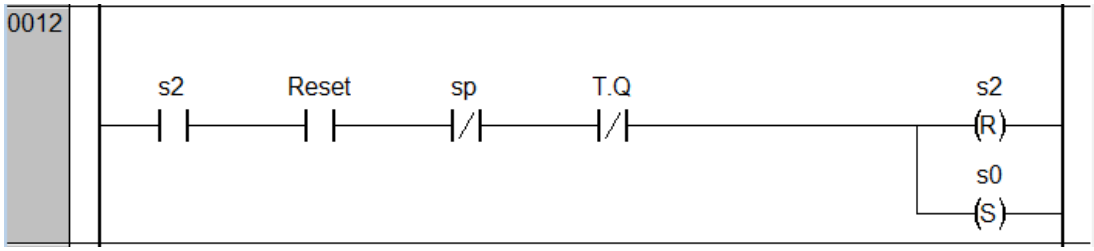


Рис. 2.149. Модификация цепи 12.

Аналогичное исправление внесем в код ST-программы:

```
2:
     Y1 := Y2 := Y3 := M := FALSE; fault := TRUE; T(IN:=FALSE);
     IF Reset AND NOT T.Q THEN state := state_prev; END_IF
```

Так всегда: не хотите нормально проектировать программу, получайте проблемы и решайте их с помощью программных «заплаток».

Программа на языке SFC представляет собой совокупность *шагов* (*steps*) и *переходов (transitions)* между шагами. Внутри шагов выполняются некоторые действия, а переходы производятся по условиям. И шаги, и переходы программируются на других языках, – сам язык SFC не предлагает соответствующих средств. Результат выполнения «программы» перехода должен быть типа `BOOL`. Если он равен `TRUE`, переход выполняется, если `FALSE`, – нет.

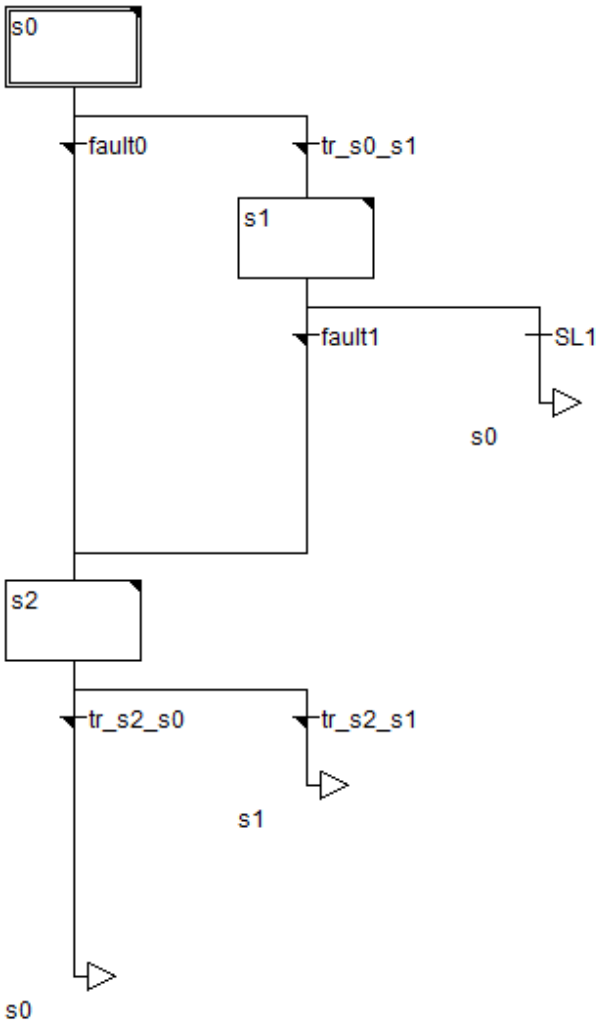


Рис. 2.150. SFC-диаграмма.

Язык очень хорошо подходит для программирования автоматов (скорее всего, он для этого и предназначен). Мы можем выделить для каждого состояния отдельный шаг, и тогда нам вообще не придется заботиться о запоминании текущего состояния.

Секция объявлений программы:

```
PROGRAM control_sfc
VAR
     T: TP;
     sp: BOOL;
END_VAR
```

SFC-диаграмма показана на рис. 2.150. Диаграмма содержит три шага s0, s1 и s2, и, конечно же, это наши состояния: «ждать-мешать-сливать», «наполнение» и «авария».

Программы для шагов и переходов мы вполне можем позаимствовать из ST-программы:

Шаг `s0`:

```
Y1 := Y2 := fault := FALSE;
T(IN:=SL1, PT:=t#5s);
M := T.Q;
Y3 := SL3 AND  NOT M;
sp:=FALSE;
```

Шаг `s1`:

```
Y1 := NOT SL2;
Y2 := SL2 AND NOT SL1;
```

```
T(IN:=FALSE);
M := fault := FALSE;
IF SL1 THEN
     T(IN:=TRUE, PT:=t#5s);
     M:=TRUE;
END_IF
sp:=TRUE;
```

Шаг `s2`:

```
Y1 := Y2 := Y3 := M := FALSE;
fault := TRUE;
T(IN:=FALSE);
```

Переходы `fault0` и `fault1`:

```
(SL2 AND NOT SL3) OR (SL1 AND NOT SL2) OR (SL1 AND NOT SL3)
OR(M AND NOT SL1) OR (M AND NOT SL2) OR (M AND NOT SL3)
```

Переход `tr_s0_s1`:

```
NOT SL3 AND Start_process
```

Переход `tr_s2_s0`:

```
Reset AND NOT sp AND NOT T.Q
```

Переход `tr_s2_s1`:

```
Reset AND sp
```

Переход из состояния `s1` в состояние `s0` – «незапрограммированный» (у его черточки отсутствует черный уголочек). В CoDeSys имя «незапрограммированного» перехода и есть его условие. В нашем случае, действительно, переход из `s1` в `s0` производится при срабатывании датчика `SL1`.

Все три программы управления были опробованы в нашей имитационной системе. Результаты наблюдались на экране визуализации. Как и следовало ожидать, они продемонстрировали абсолютно идентичное поведение и полностью реализовали автомат *в том виде, в котором он был задуман*.

3. РАЗРАБОТКА ИМИТАЦИОННЫХ МОДЕЛЕЙ НА ПРИМЕРАХ

3.1. Системы автоматического регулирования

3.1.1. Простейшая система регулирования с ПИ-регулятором

3.1.1.1. Разработка и имитация регулятора

Рассмотрим простейшую одноконтурную систему регулирования, в состав которой входят только объект и регулятор. На самом деле, как было сказано выше, любая одноконтурная система состоит только из объекта и регулятора, другое дело, что мы вкладываем в эти понятия. Здесь мы полагаем, что объект – это совокупность технических средств, управляемых единственным *аналоговым сигналом*, который формируется регулятором. В качестве примеров таких систем можно привести

систему регулирования температуры с воздействием на мощность тиристорного регулятора напряжения нагревательного элемента,

систему регулирования давления с воздействием на преобразователь частоты, питающий двигатель насоса или компрессора и т.д.

Сам «аналоговый» сигнал может быть вовсе и не аналоговым, а цифровым и передаваться в указанное устройство по промышленной сети, но сути дела это не меняет: устройство выдает выходное напряжение, частоту, мощность или другой параметр пропорционально значению входного управляющего сигнала. Задача регулятора – «подобрать» такое управление, чтобы регулируемая величина (температура, давление или что-то еще) была равна заданной, а точнее была максимально близка ней, несмотря на все возмущения, действующие на объект. Алгоритм регулятора реализуется программой ПЛК.

Пусть каким-то образом получена передаточная функция объекта по управлению в виде:

$$W_{об}(p) = \frac{k}{Tp+1} e^{-\tau p} \ .$$

Возмущение не контролируется (не измеряется), и получить передаточную функцию объекта по возмущению не представляется возможным. Поэтому возмущение условно прикладывается к тому же входу объекта, что и управление (складывается с управлением или вычитается из него), что формально означает, что возмущение измеряется в тех же единицах, что и управление: для «нейтрализации» возмущения некоторой «мощности» требуется равное изменение «мощности» управляющего сигнала. Тогда структура системы регулирования примет вид, показанный на рис. 3.1.

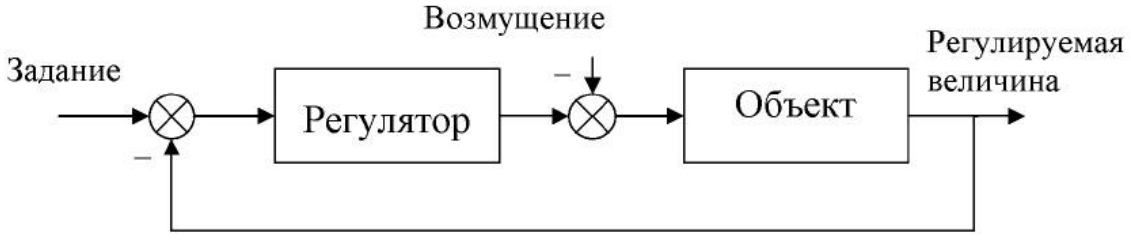


Рис. 3.1. Структура системы автоматического регулирования

Однако это всего лишь модель системы *автоматического* регулирования. На практике такая система дополняется средствами ручного управления, сигнализации, защиты, архивирования и т.д. Эти средства так же требуют моделирования, и мы их рассмотрим позже.

Пусть $k = 1$, $T = 30$ с, $\tau = 10$ с.

Simulink-модель замкнутого контура показана на рис. 3.2.

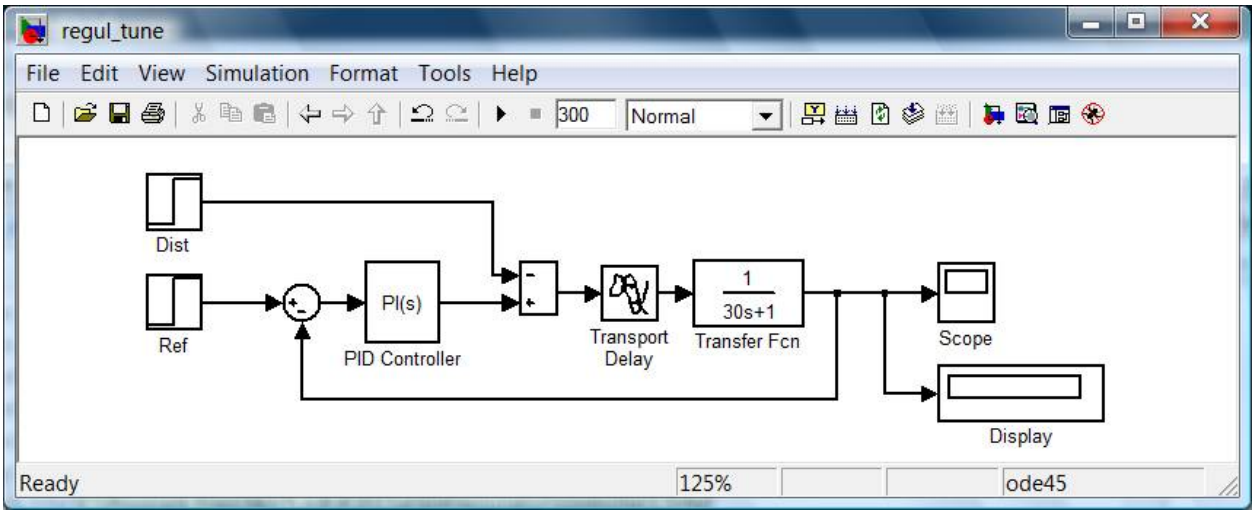


Рис. 3.2. Simulink-модель контура регулирования.

Регулятор настроен с помощью встроенного средства настройки (рис. 3.3).

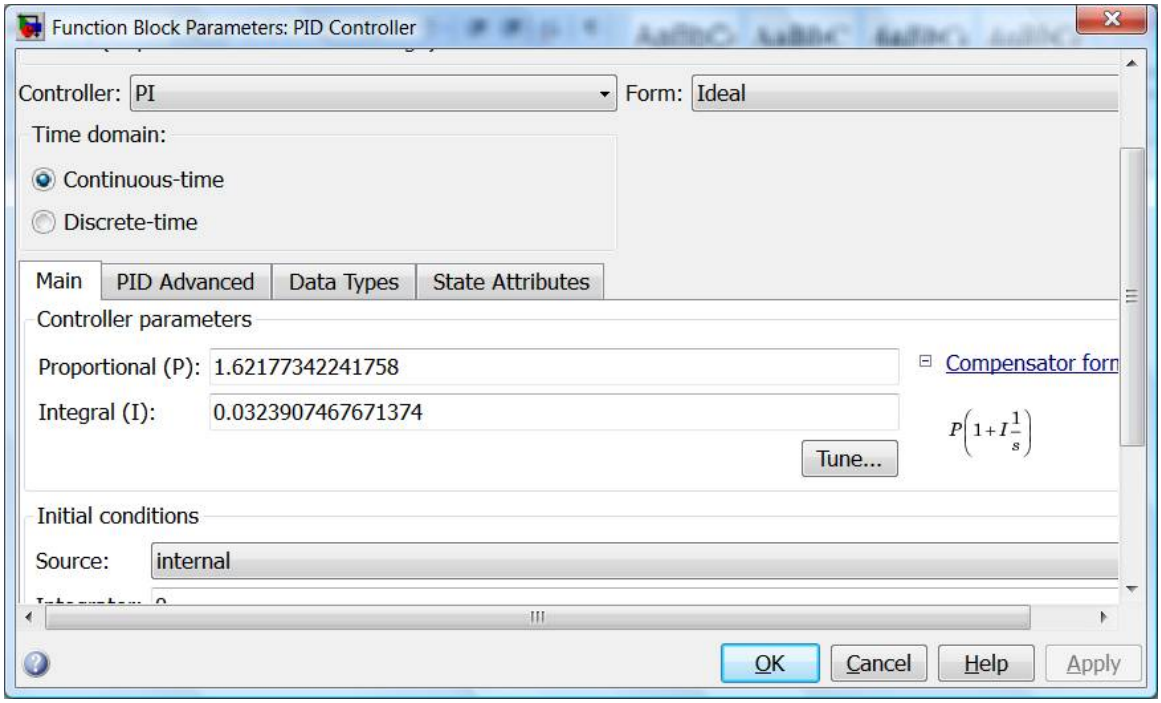


Рис. 3.3. Автонастройка регулятора.

Реакция системы на задающий сигнал и возмущение (изменяющиеся с нуля до единицы в моменты времени $t = 0$ с и $t = 100$ с соответственно) показана на рис. 3.4.

Далее перейдем к реализации комбинированной модели, в которой функцию регулятора будет выполнять программа ПЛК.

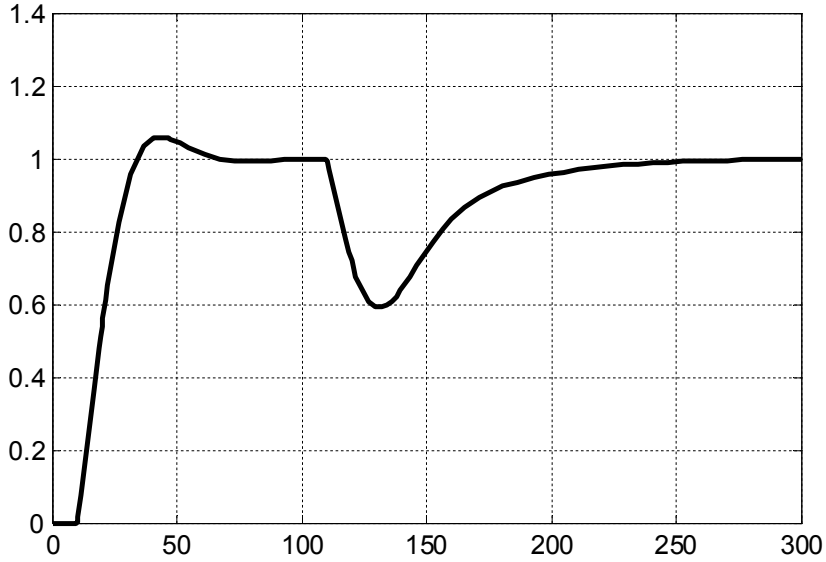


Рис. 3.4. Реакция системы на задающий сигнал и возмущение.

Стандартный ПИД регулятор из библиотеки Util.lib CoDeSys [3], показанный на рис. 3.5, реализует закон регулирования в виде:

$$Y = Y\_OFFSET + KP\left(e(t) + \frac{1}{TN}\int e(t) + TV\frac{de(t)}{dt} \right),$$

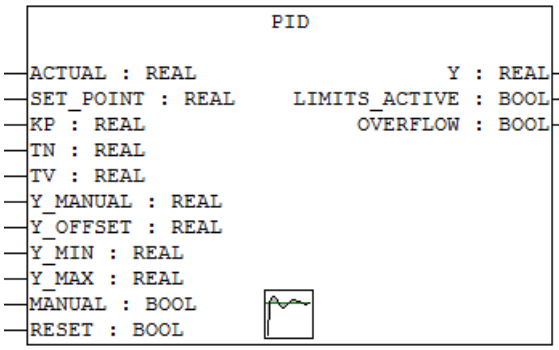


Рис. 3.5. Функциональный блок PID.

где *Y_OFFSET* – стационарное (номинальное выходное) значение; *KP* – коэффициент передачи; *TN* – постоянная интегрирования; *TV* – постоянная дифференцирования (*обе постоянные – в секундах, а не в «ms», как ошибочно указано в документации!*); *e*(*t*) – сигнал ошибки регулирования.

В нашем случае *KP* = 1,62, *TN* = 1/0,032 =31,25 с, *TV* = 0.

На входы SET_POINT и ACTUAL подаются уставка регулятора (сигнал задания) и реальное значение регулируемой величины (сигнал обратной связи).

Выходной сигнал Y ограничен минимальным Y_MIN и максимальным Y_MAX значениями. При достижении основным выходом границ выход LIMITS_ACTVE устанавливается в TRUE.

Если входной сигнал MANUAL равен TRUE, то регулирование выключается, и на выход Y начинает транслироваться значение входного сигнала Y_MANUAL. При переходе значения MANUAL в FALSE происходит рестарт регулятора.

При переполнении интегральной составляющей работа регулятора останавливается, а выход OVERFLOW устанавливается в TRUE. Для сброса интегральной составляющей в нуль и перезапуска регулятора используется вход RESET.

Наша первая программа будет состоять из одного POU: программы PLC_PRG, «написанной» на языке CFC.

Секция объявлений программы:

```
PROGRAM PLC_PRG
VAR
     Controller: PID;
END_VAR
```

Диаграмма программы показана на рис. 3.6.

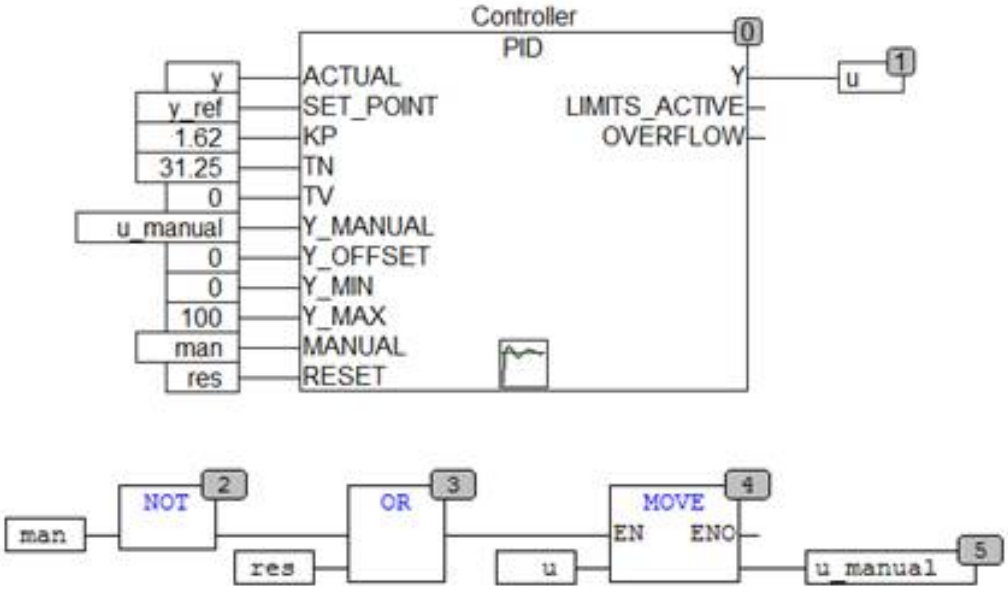


Рис. 3.6. Программа ПЛК.

В программе задействовано несколько переменных, которые мы заранее объявили как глобальные:

```
VAR_GLOBAL
     y_ref,y:REAL; (*задание и выход объекта*)
     u:REAL; (*управление*)
     u_manual:REAL:=0;(*сигнал управления в ручном режиме*)
     man:BOOL:=FALSE; (*сигнал перехода в ручной режим*)
     res:BOOL:=FALSE; (*сигнал сброса регулятора*)
END_VAR
```

Переменная `u_manual` в ручном режиме, если неактивен сигнал сброса, будет формироваться непосредственно пользователем (оператором), а в автоматическом (и в ручном, если активен сигнал сброса) ее значение повторяет значение выхода регулятора. Это позволит осуществлять «безударное» переключение системы с автоматического на ручной режим управления: выходной сигнал `u` не сможет изменяться «скачком». Отметим, что безударность переключения с ручного режима на автоматический обеспечивается самим функциональным блоком PID.

Переменные `y_ref`, `y`, `u`, `res` включены в список обмена по OPC.

Программу управления в таком виде можно откомпилировать и загрузить в предварительно запущенный виртуальный контроллер PLC WinNT.

Вернемся к Simulink-модели. Модифицируем модель таким образом, чтобы было возможным сравнить поведение систем регулирования с «внутренним» и «внешним» регуляторами.

С помощью утилиты `opctool` создадим блок чтения переменных `res`, `u`, `y_ref` и блок записи переменной `y`. Блок OPC Configuration будет создан автоматически.

На вход модели системы регулирования подадим сигнал `y_ref`. В настройках блока PID Controller активируем опцию External Reset = level (внешний сброс по уровню) и подадим на появившийся вход сброса сигнал `res`. Это позволит сбрасывать «модельный» регулятор одновременно с программным. Установим диапазон изменения выходного сигнала «модельного» регулятора таким же, как у «программного»: от 0 до 100. Значение параметра Anti-windup method – clamping («заморозка» интегральной составляющей регулятора на время действия ограничения выходного сигнала).

Создадим копию объекта управления, на вход которой заведем сигнал управления `u`, формируемый «программным» регулятором. Схему формирования возмущения сделаем общей для двух систем. Введем в нее блок Slider Gain, который позволит изменять возмущение вручную, не прерывая процесс имитационного моделирования.

Предусмотрим возможность наблюдать сигналы управления и выходы двух систем.

В результате получим Simulink-диаграмму (рис. 3.7).

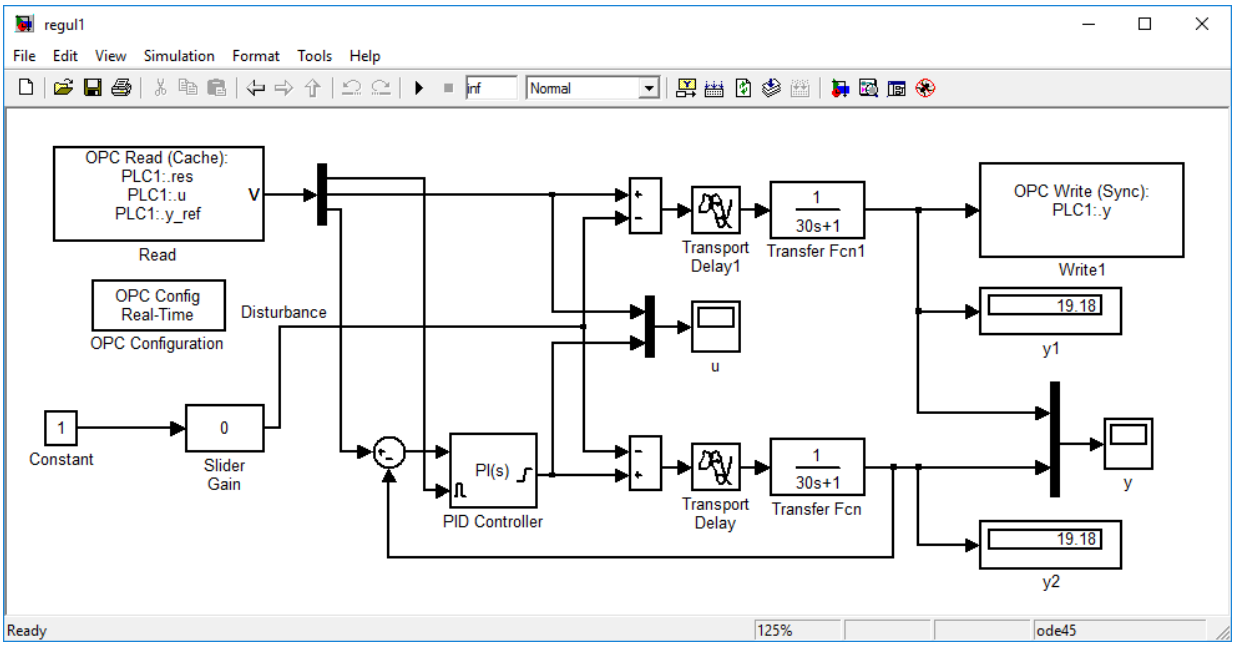


Рис. 3.7. Simulink-диаграмма в окончательном виде.

Запустим модель на вычисление. В CoDeSys, подключившись к работающему виртуальному контроллеру, вручную изменим переменную `y_ref`, установив задание равным 30%. Наблюдая переходные процессы в блоках Scope Simulink-диаграммы, дождемся их окончания, после чего с помощью блока Slider Gain подадим возмущение, равное 10%. Наблюдаем переходные процессы по возмущению. Результаты показаны на рис. 3.8, 3.9.

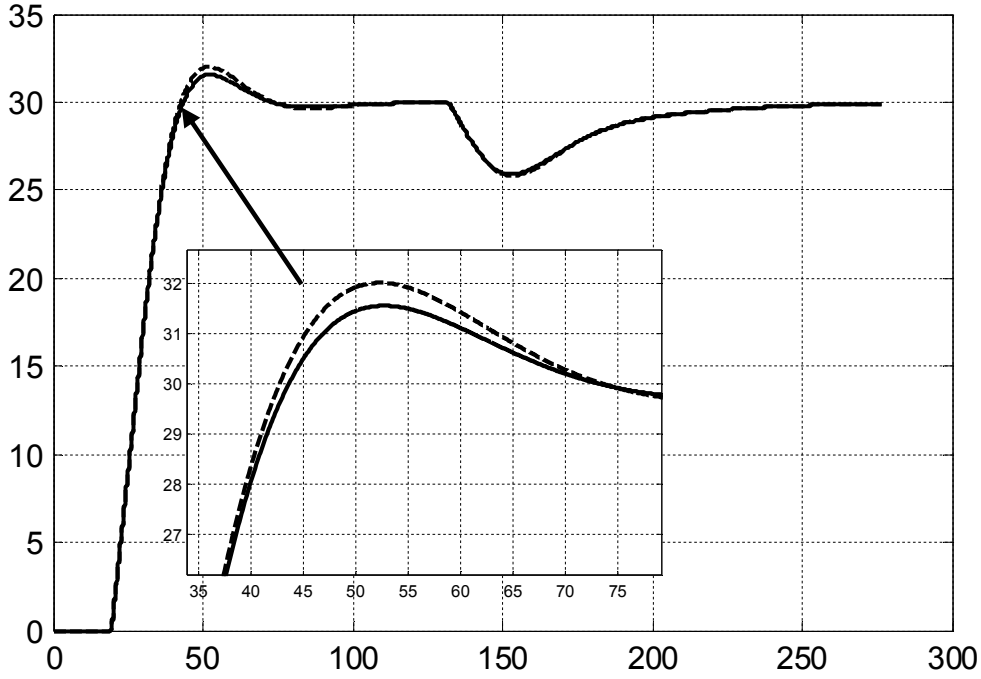


Рис. 3.8. Изменение регулируемой величины в эталонной системе и системе с программным регулятором при отклонении задания на 30% и возмущения на 10%.

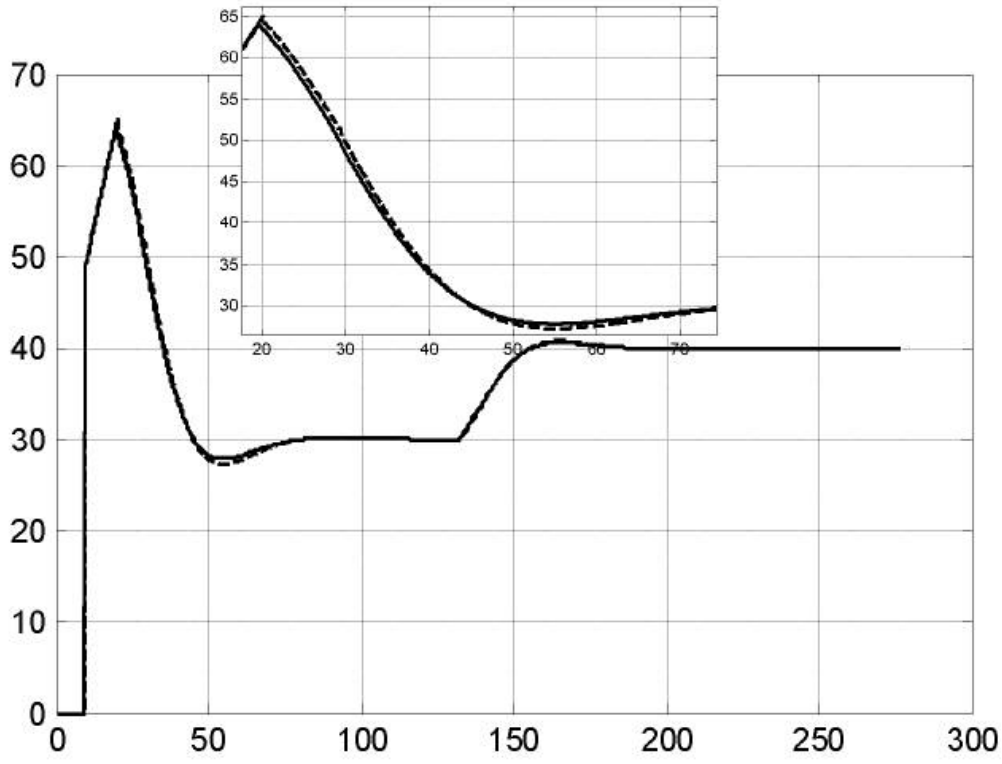


Рис. 3.9. Изменение управляющего воздействия в эталонной системе и системе с программным регулятором при отклонении задания на 30% и возмущения на 10%.

Далее проделаем то же самое со значениями задания 60% и возмущения 30%. Результаты – на рис. 3.10, 3.11.

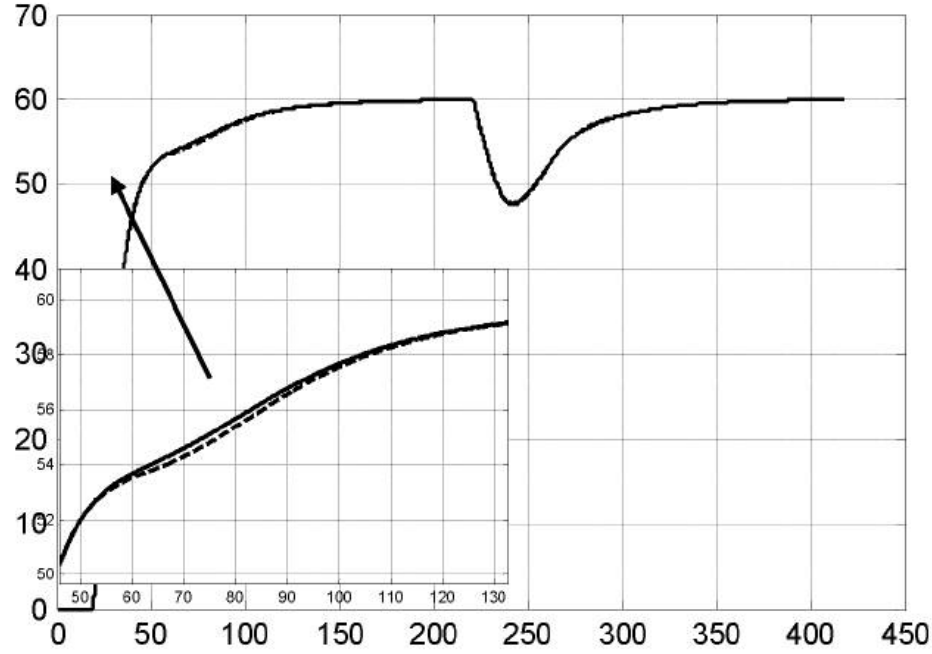


Рис. 3.10. Изменение регулируемой величины в эталонной системе и системе с программным регулятором при отклонении задания на 60% и возмущения на 30%.

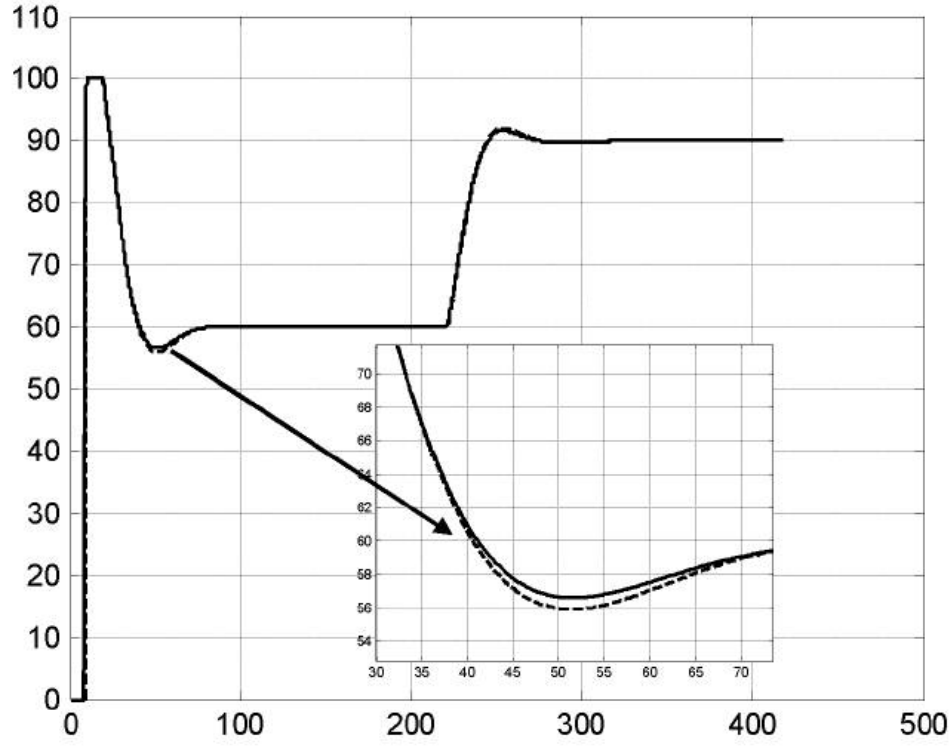


Рис. 3.11. Изменение управляющего воздействия в эталонной системе и системе с программным регулятором при отклонении задания на 60% и возмущения на 30%.

Результаты моделирования показывают, что поведение системы с «программным» регулятором достаточно близко к поведению «эталонной» системы с «модельным» регулятором. Незначительные отличия связаны, по-видимому, с задержкой передачи сигналов между программами по OPC.

В дальнейшем «эталонная» система нами не будет задействована, и поэтому составляющие ее блоки можно полностью удалить из Simulink-диаграммы. Однако мы «на всякий случай» не будем этого делать.

3.1.1.2. Структура системы

Перейдем к разработке «высших» уровней управления.

Сразу же отметим, что наша модель пока ограничивается только имитацией системы регулирования и не затрагивает всех вопросов управления оборудованием (которое даже и не указано), поэтому это всего лишь «заготовка», «типовое решение» и не более того.

В реальных системах аппаратура устанавливается внутри шкафов и пультов управления. На лицевые панели шкафов и пультов выносятся разнообразные органы управления, индикации и сигнализации: переключатели режимов, задатчики сигналов, блоки ручного управления, сигнальные лампы, показывающие приборы и т.д. С помощью всех этих органов осуществляется контроль процесса и управление им. Возможны два варианта подключения органов индикации и управления:

1) непосредственно к техническим средствам контроля параметров объекта и воздействия на него (датчикам, измерительным преобразователям, исполнительным механизмам). С помощью специальных переключателей «Руч./Авт.» формируются различные электрические цепи для управления в ручном и автоматическом режимах, причем в ручном режиме ПЛК не задействуется вообще. Однако знать текущий режим ПЛК в большинстве случаев должен, поэтому на его отдельный вход заводится соответствующий сигнал (например, для этого может использоваться дополнительный контакт переключателя режима в цепи дискретного входа ПЛК);

2) к ПЛК. В этом случае во всех режимах управление объектом осуществляется программой ПЛК через его выходы. В автоматическом режиме выполняется подпрограмма автоматического управления, в ручном – команды, сформированные органами управления после некоторой обработки «транслируются» на выходы контроллера.

Первый вариант всегда дороже, так как требует более сложной аппаратной реализации (необходимо разработать сложные электрические схемы соединений, предусматривающие всевозможные блокировки и защиты), тогда как во втором варианте вся сложность «ложится на плечи» программного обеспечения контроллера. Несмотря на гибкость второго варианта, он обладает серьезным недостатком: при отказе контроллера вся система управления выходит из строя. Поэтому для ответственных случаев требуется горячее резервирование контроллера.

В настоящее время все многочисленные органы контроля и управления все чаще заменяются одной или несколькими операторскими панелями с сенсорными экранами. Панели подключаются к ПЛК по последовательным линиям или посредством промышленных сетей. Специальное программное обеспечение для персональных компьютеров позволяет достаточно легко разрабатывать пользовательские интерфейсы операторских панелей и связывать их элементы

(программные кнопки, переключатели, индикаторы и т.д.) с технологическими переменными. Преимуществами такого решения являются компактность, гибкость и низкая стоимость (для систем с большим количеством органов управления и индикации), однако у него есть и недостатки: в случае отказа панели теряется возможность ручного управления процессом, а в случае отказа контроллера управляемость процессом теряется полностью.

На более высоком уровне находятся SCADA-системы. Эти программы работают на операторских станциях – обычных или промышленных компьютерах. Функционал SCADA-систем частично повторяет функционал операторских панелей, однако в отличие от последних SCADA-системы занимаются сбором и анализом данных, визуализацией и управлением технологическим процессом в целом, тогда как панели, как правило, «привязаны» к отдельным установкам.

При распределении обязанностей между щитом управления (операторской панелью) и SCADA-системой могут возникнуть проблемы, связанные с установлением приоритетов. Например, мы можем предусмотреть возможность формирования управляющего сигнала тремя «действующими лицами»:

1) оператором щита/панели (режим ручного управления);

2) программой автоматического управления контроллера (режим автоматического управления);

3) оператором станции – пользователем SCADA-системы (назовем это режимом дистанционного управления).

В системах, допускающих ручное управление без ПЛК, очевидно, приоритет за оператором щита: он всегда может перевести систему в ручной режим, отстранив ПЛК от управления, и тем самым полностью «взять власть в свои руки».

В системах, не работающих без ПЛК, «окончательный» выбор режима – за программой контроллера, но составить-то ее должны мы сами! Здесь принципиально возможны различные подходы. Рассмотрим один из наиболее «вероятных».

Оператор щита/панели самостоятельно выбирает один из двух вариантов: ручное управление или «автоматическое». При этом выбрав второй вариант, он просто отстраняется от непосредственного управления и доверяет его «системе», в которую входят как программа, так и оператор более высокого уровня. Сам оператор щита может остаться «на посту», чтобы продолжать наблюдать за процессом и в случае неблагоприятного развития событий вновь взять управление на себя, либо покинуть «пост», доверившись «системе». Конечно, в последнем случае он должен быть уверен, что в «системе» «еще остались люди», но этот вопрос может быть решен, например, по внутренней связи. Оператор SCADA-системы в случае необходимости может перевести систему управления из автоматического в дистанционный режим и самостоятельно управлять процессом, а потом вновь вернуться к автоматическому режиму. Единственное, что он не может, – это включить ручное управление, так же как и оператор щита – дистанционное.

Таким образом, получается, что наибольший приоритет имеет ручной режим управления, наименьший – автоматический, дистанционный занимает промежуточную позицию. Однако в случае использования операторской пане-

ли вполне возможна ситуация, когда панель требуется по каким-либо причинам «заблокировать», т.е. полностью запретить управление посредством ее органов. Но при этом контроллеру должно быть известно, что контроль процесса ведется SCADA-системой или хотя бы в том, что SCADA-система *находится на связи*.

Предположим, что в нашей «воображаемой» системе задействованы операторская панель и SCADA-система. Они взаимодействуют между собой посредством каких-то физических интерфейсов, до которых нам нет никакого дела. Модель операторской панели мы сначала оформим в виде экрана визуализации CoDeSys, а SCADA-систему будем использовать «живьем» (настоящую, Trace Mode) (рис. 3.12).

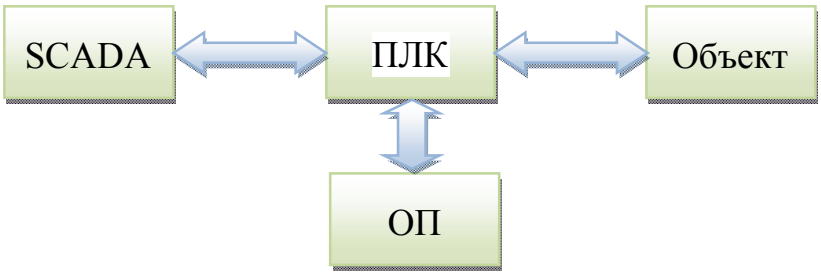


Рис. 3.12. Структура системы имитационного моделирования.

Определимся с функциями панели:

1) отображение задания, регулируемой величины и управляющего воздействия;

2) переключение режима, изменение уставки регулятора и сигнала управления в ручном режиме, сброс регулятора *при условии, что панель не заблокирована*;

3) сигнализация при отклонениях регулируемой величины от уставки более чем на 10%.

Панель может быть заблокирована SCADA-системой. Разблокирование панели происходит автоматически в случае потери связи со SCADA-системой.

Каковы же функции SCADA-системы? Практически те же самые плюс архивация. Архивации подлежат все аналоговые сигналы, а также такие события, как смена режима управления, выход регулируемой величины за пределы допустимого диапазона и т.д.

3.1.1.3. Разработка программы контроллера

Начнем с доработки программы контроллера. Теперь она помимо собственно *регулирования* будет заниматься еще много чем. Поэтому «локализуем» регулятор в отдельную программу, для чего переименуем `PLC_PRG` в `CONTROL` и исключим из нее все «ненужное» (рис. 3.13).

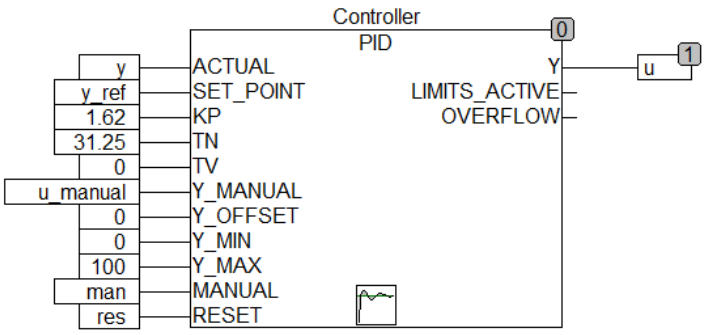


Рис. 3.13. Программа CONTROL.

```
PROGRAM CONTROL
VAR
     Controller: PID;
END_VAR
```

Разработку программы для ПЛК всегда следует начинать с назначения «главных» переменных, отвечающих за обмен информацией с «внешним миром» и хранение «ключевых» данных. Эти переменные мы сделаем глобальными, их объявление с необходимыми комментариями приведено ниже.

```
VAR_GLOBAL
(*Входы контроллера*)
     y:REAL; (*регулируемая величина*)

     reset_process:BOOL:=FALSE; (*сброс системы после аварии –
                                аппаратная кнопка без фиксации*)

(*Выходы контроллера*)
     u:REAL; (*Управление*)
     no_HMI:BOOL:=FALSE; (*потеряна связь с панелью и SCADA –
                          сигнализация*)
     stop:BOOL:=FALSE; (*полная остановка системы*)

(*"Внутренние" переменные*)
     y_ref:REAL; (*задание регулятора, также  подается в Simulink-
                  модель*)
     u_manual:REAL:=0; (*сигнал ручного управления*)
     man:BOOL:=FALSE; (*сигнал перевода в режим ручного
                       управления*)
     res:BOOL:=FALSE; (*сброс регулятора, также  подается в
                       Simulink-модель*)
     block_panel:BOOL:=FALSE; (*блокирование панели*)

(*Переменные обмена с панелью оператора*)
     (*входы:*)
     y_ref_from_OP:REAL:=0; (*задание с панели*)
     y_ref_update_from_OP:BOOL:=FALSE; (*обновить задание с
                                        панели*)
     u_manual_from_OP:REAL:=0; (*сигнал ручного управления с
                                панели*)
     man_from_OP:BOOL:=FALSE; (*сигнал перехода на ручной режим с
                               панели*)
     res_from_OP:BOOL; (*сброс регулятора с панели*)
     tic_tac_from_OP:BOOL; (*тестовый сигнал наличия связи с
                            панели*)
     (*выходы:*)
     y_ref_to_OP, y_to_OP, u_to_OP:REAL; (*задание, выход и
                                          управление на панель*)
     res_to_OP:BOOL; (*сброс регулятора на панель*)
     state_to_OP:SINT; (*режим управления на панель*)
     block_panel_to_OP:BOOL; (*панель заблокирована на панель*)
     SCADA_link_to_OP:BOOL:=TRUE; (*связь со SCADA на панель*)
     err_gt_10_to_OP, err_lt_m10_to_OP, norma_to_OP:BOOL; (*сигналы
```

```
                                                    контроля на панель*)
     tic_tac_to_OP:BOOL; (*тестовый сигнал наличия связи на
                          панель*)

(*Переменные обмена со SCADA-системой*)
     (*входы:*)
     y_ref_from_SCADA:REAL:=0; (*задание от SCADA*)
     y_ref_update_from_SCADA:BOOL:=FALSE; (*обновить задание от
                                           SCADA*)
     u_manual_from_SCADA:REAL:=0; (*сигнал дистанционного
                                   управления от SCADA*)
     dist_from_SCADA:BOOL:=FALSE; (*переход на дистанционное
                                   управление от SCADA*)

     block_panel_from_SCADA: BOOL:=FALSE; (*заблокировать панель
                                           от SCADA*)
     res_from_SCADA:BOOL:=FALSE; (*сброс регулятора от SCADA*)
     tic_tac_from_SCADA:BOOL; (*тестовый сигнал наличия связи от
                               SCADA*)
     (*выходы:*)
     y_to_SCADA:REAL; (*выходной сигнал в SCADA*)
     y_ref_to_SCADA:REAL; (*задание в SCADA*)
     u_to_SCADA:REAL; (*управление в SCADA*)
     res_to_SCADA:BOOL; (*сброс регулятора в SCADA*)
     state_to_SCADA:SINT; (*режим управления в SCADA*)
     block_panel_to_SCADA: BOOL:=TRUE; (*панель заблокирована в
                                         SCADA*)
     OP_link_to_SCADA:BOOL:=TRUE; (*связь с панелью в SCADA*)
     tic_tac_to_SCADA:BOOL; (*тестовый сигнал наличия связи в
                             SCADA*)
END_VAR
```

Рассмотрим список глобальных переменных по группам.

1. Входы и выходы контроллера. В реальной системе эти переменные привязываются к реальным входам и выходам контроллера. В нашем случае объект управления имитируется Simulink-моделью, связь с которой осуществляется по протоколу OPC. В идеале для «общения» с объектом достаточно двух переменных – для регулируемой величины и управляющего сигнала. Однако мы задействуем еще два «дискретных выхода»: один будет оповещать персонал о потере связи со всеми устройствами человеко-машинного интерфейса, второй – полностью выключать оборудование, в случае, если при потере связи регулятор в течение определенного времени не сможет привести процесс в норму. Оба выхода могут «управлять» какими-либо средствами сигнализации: сигнальными лампами, гудками и т.д. Помимо этого предусмотрим «дискретный вход» для сброса системы после аварии.

2. «Внутренние» переменные: текущее задание, сигналы ручного управления, сброса регулятора и блокировки панели. Поскольку мы решили не перестраивать ранее разработанную Simulink-диаграмму, сигналы задания и сброса регулятора будут по-прежнему в нее подаваться (они используются для управления «модельным» регулятором).

3. Переменные обмена с панелью оператора. От панели контроллер получает сигналы задания, ручного управления, сброса регулятора, блокировки панели и др. В панель отправляются фактические значения регулируемой величины, задания и управления, информация о режиме управления, блокировке панели и наличии связи со SCADA-системой. Кроме того, мы предполагаем, что панель не в состоянии самостоятельно проводить вычисления, и поэтому отправляем на нее сформированные контроллером сигналы о текущем состоянии процесса регулирования: «ошибка больше 10%», «ошибка меньше -10%», «норма».

В целом взаимодействие с панелью (как и со SCADA-системой) должно быть построено, и это важно, по следующему принципу: операторы будет наблюдать на экранах не только свои «пожелания», но и фактическое положение дел. Недостаточно просто «отдать приказ», необходимо также и услышать отклик «есть!» (или не услышать). Поэтому есть переменные-приказы от панели (SCADA-системы) и переменные-отклики от контроллера, которые обязательно должны «отображаться».

Помимо этого, будет реализовано программное обнаружение, как на стороне контроллера, так и на стороне панели (и SCADA-системы), потери связи. В реальной ситуации пользовательская программа контроллера может «узнать» о потере связи от его операционной системы, которая занимается обменом по сети или последовательному интерфейсу. Мы в нашей имитационной системе лишены такой возможности.

Панель и контроллер вообще находятся «в одном адресном пространстве» и связь между ними осуществляется через «общие» переменные. В случае же обмена между контроллером и SCADA-системой OPC-сервер, через который этот обмен происходит, сам по себе «пассивен», а для контроллера вообще «прозрачен», т.е. для контролера «его не существует». Поэтому обнаружение потери связи и будет реализовано программно, для чего будут задействованы переменные «tic_tac». Идея простая:

контроллер посылает в панель или SCADA-систему значение булевой переменной (единицу или нуль);

они отвечают инвертированным значением;

контроллер вновь отсылает полученное значение в панель или SCADA-систему;

если в течение определенного времени контроллер (5 сек.) замечает, что ответы «повторяются», это означает, что связь потеряна.

4) переменные обмена со SCADA-системой. Эти переменные в основном аналогичны таковым для панели оператора. Основные отличия:

SCADA-система, безусловно, способна выполнять любые вычисления, поэтому сигналы типа «ошибка больше 10%» отправлять ей «не обязательно»;

SCADA-система не только принимает сигнал о блокировке панели, но и может сама сформировать сигнал «Заблокировать панель».

Все глобальные переменные для простоты включены в список тегов OPC-сервера, хотя фактически переменные `no_HMI`, `stop`, `man`, `block_panel` и

все переменные из раздела «Переменные обмена с панелью оператора» в обмене по OPC участвовать не будут.

Далее рассмотрим имитацию панели оператора. На рис. 3.14 – «конечный результат», панель в работе. На панели размещены два тренда: в одном отображаются графики изменения задания и регулируемой величины, на другом – график изменения управляющего воздействия. Текущие значения задания, регулируемой величины и управления, кроме того, выводятся в прямоугольниках под трендами. Управляющий сигнал определяет также положение ползунка.

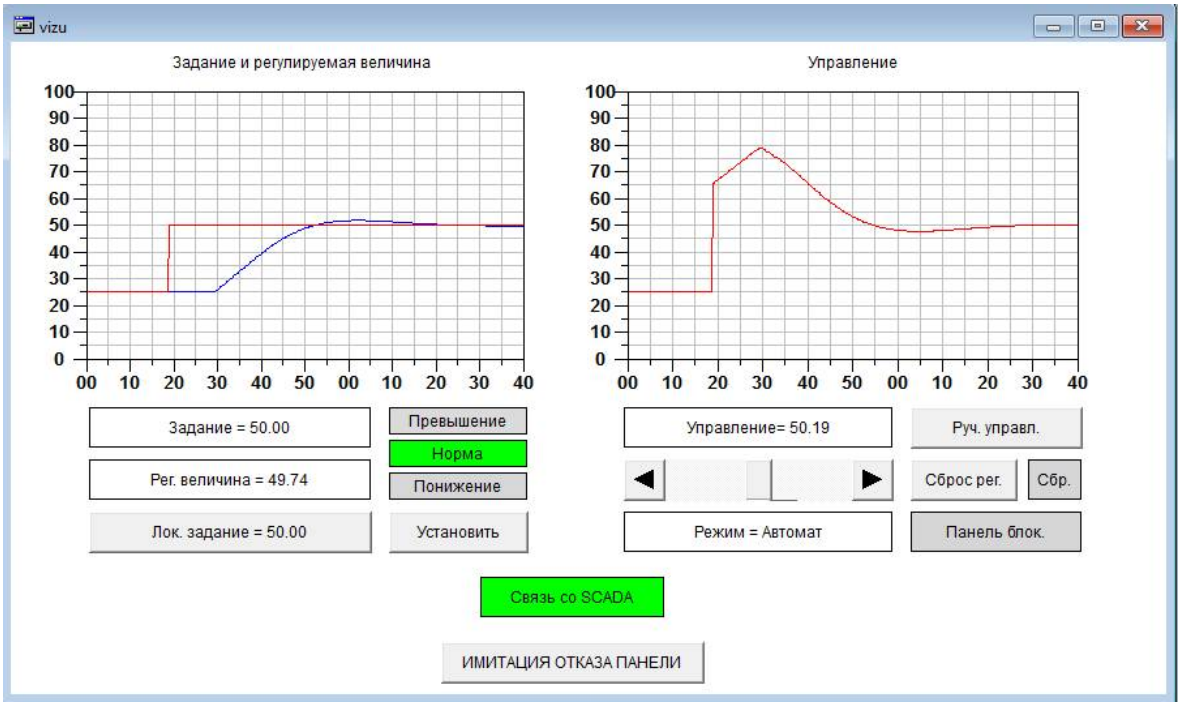


Рис. 3.14. Окно визуализации vizu в работе.

С помощью кнопки ввода можно ввести локальное задание, а путем нажатия на кнопку «Установить» – отправить запрос на изменение задания контроллеру. Качественную оценку текущего состояния процесса регулирования дают индикаторы «Превышение», «Понижение» (загораются красным при выходе регулируемой величины из 10% коридора относительно уровня задания) и «Норма» (загорается зеленым при нахождении регулируемой величины в 10% коридоре).

С помощью переключателя «Руч. управл.» выполняется запрос на переход в режим ручного управления. В этом режиме задавать управляющий сигнал можно, изменяя положение ползунка вручную. Информация о режиме работы отображается в прямоугольнике под ползунком.

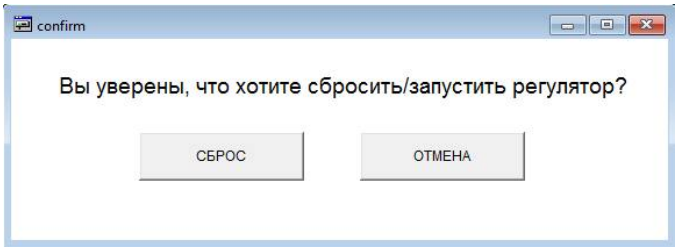


Рис. 3.15. Окно визуализации confirm в работе.

При нажатии на кнопку «Сброс рег.» появляется вспомогательное окно для подтверждения намерения сбросить регулятор (переключатель «СБРОС») или отказа от него (кнопка «ОТМЕНА») (рис. 3.15).

Нажатие любой из кнопок приводит к возврату к основной визуализации.

Справа от кнопки «Сброс рег.» – индикатор сброса регулятора, при сбросе он загорается красным. Ниже располагается индикатор блокировки панели (загорается красным при блокировке). При блокировке «панель» продолжает работать, однако контроллер не выполняет ее команд.

В нижней части располагается индикатор связи контроллера со SCADA-системой (при потере связи меняет цвет с зеленого на красный) и переключатель имитации отказа панели. При отказе панели она теряет связь с контроллером и перестает обновлять экран.

Разработка визуализаций в CoDeSys – задача совсем не сложная. Все устроено просто и интуитивно понятно, поэтому далее мы ограничимся лишь кратким перечислением привязок элементов визуализации к переменным программы и не будем рассматривать технические вопросы настройки самих элементов. При необходимости исчерпывающую информацию можно найти в [3].

Для «оживления» визуализации вполне можно использовать глобальные переменные из секции «Переменные обмена с панелью оператора», см. выше. Однако это нарушает нашу концепцию, – ведь мы считаем панель «отдельным устройством» и поэтому не должны пользоваться «чужими» переменными.

Создадим программу `PANEL` (язык ST) в которой объявим полный набор локальных переменных для привязки к элементам визуализации.

```
PROGRAM PANEL
VAR
     (*ОТОБРАЖЕНИЕ*)
     y_ref, y, u:REAL; (*задание, выход и управление*)
     err_gt_10, err_lt_m10, norma:BOOL; (*сигналы контроля *)
     block_panel:BOOL; (*блокировать панель - на панель*)
     SCADA_link:BOOL:=TRUE; (*связь со SCADA на панель*)
     state: STRING; (*режим управления*)
     res:BOOL; (*сброс регулятора*)

     (*УПРАВЛЕНИЕ И ВВОД*)
     y_ref_local:REAL:=0; (*локальное задание*)
     y_ref_update:BOOL:=FALSE; (*обновить задание*)
     man:BOOL:=FALSE; (*сигнал перехода на ручной режим*)
     do_reset:BOOL:=FALSE; (*сброс регулятора*)
     ERROR:BOOL:=FALSE; (*отказ панели*)
END_VAR
```

То, что в некоторые переменные имеют имена, совпадающие с именами глобальных переменных, не должно нас смущать: программа все равно в первую очередь будет обращаться к локальным переменным, а в привязках визуализации мы будем указывать полные имена, например, `PANEL.y_ref`.

На рис. 3.16, 3.17 повторно показаны основное и дополнительное окна визуализации, но уже с указанием привязок и необходимыми уточнениями.

Ниже приведена секция кода программы `PANEL`.

В нормальном режиме (в отсутствии отказа) программа в основном занимается копированием значений «выходных переменных контроллера», предназначенных для панели, в локальные переменные, привязанные к органам ото-

бражения, и, обратно, – значений локальных переменных, привязанных к органам ввода – в соответствующие «входные переменные контроллера». «Попутно» формируется строка с информацией о режиме и инвертируется «tic-tac». При отказе программа вообще ничего не делает, лишь формирует строку `'Ошибка связи'`.

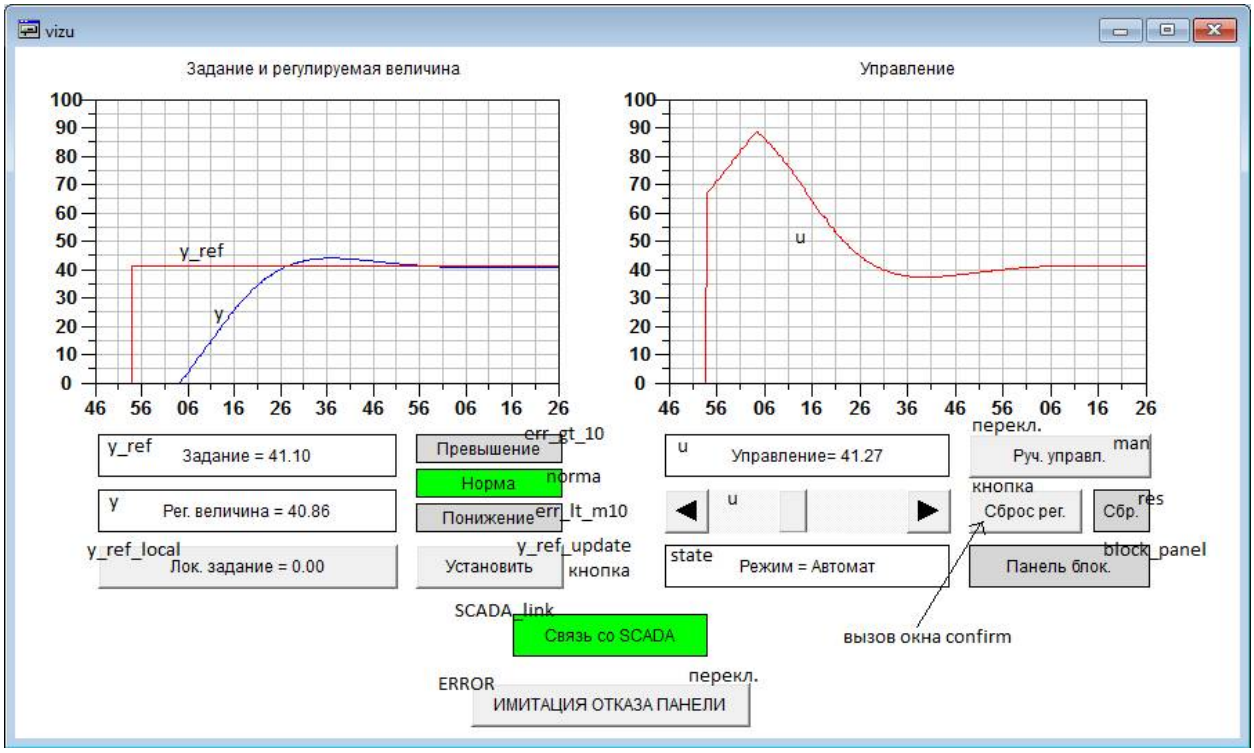


Рис. 3.16. Окно визуализации vizu с привязками.

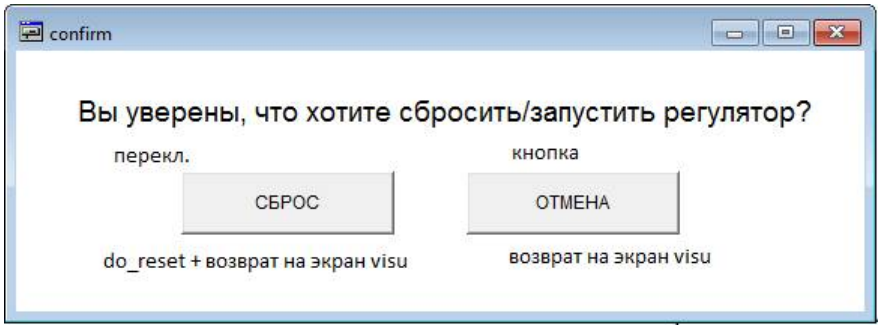


Рис. 3.17. Окно визуализации confirm с привязками.

Секция кода программы `PANEL`:

```
IF NOT ERROR THEN
     y_ref:= y_ref_to_OP;
     y:= y_to_OP;
     err_gt_10:=err_gt_10_to_OP;
     err_lt_m10:=err_lt_m10_to_OP;
     norma:=norma_to_OP;
     block_panel:=block_panel_to_OP;
     SCADA_link:=SCADA_link_to_OP;
     res:=res_to_OP;
     CASE state_to_OP OF
    -1:
          state:='Авария';
```

```
            u:=u_to_OP;
        0:
            state:='Ручной';
        1:
            state:='Автомат';
            u:=u_to_OP;
        2:
            state:='Дистанц.';
            u:=u_to_OP;
        END_CASE

        y_ref_from_OP:=y_ref_local;
        y_ref_update_from_OP:=y_ref_update;
        u_manual_from_OP:=u;
        man_from_OP:=man;
        res_from_OP:=do_reset;

        tic_tac_from_OP:=NOT tic_tac_to_OP;
    ELSE
        state:='Ошибка связи';
    END_IF
```

Снова может возникнуть вопрос: зачем нам нужна такая программа с кучей локальных переменных, когда мы вполне могли бы управлять визуализацией с помощью глобальных переменных? Частично ответ на этот вопрос уже был дан. Приведем здесь еще два аргумента:

1) с помощью этой программы можно «заморозить панель» и сообщить контроллеру об ее отказе, не нарушая нашей концепции;

2) реализованное в программе и визуализации поведение соответствует самым простым, «безмозглым», операторским панелям. Современные панели способны самостоятельно производить различные вычисления и формировать экраны на основе их результатов или, наоборот, производить вычисления по результатам действий оператора, т.е. по сути, они уже являются полноценными компьютерами. Для имитации таких панелей без программы уже никак не обойтись.

Заметим также, что программа не занимается обнаружением потери связи, поскольку в реальных панелях всегда это делается «аппаратно»: если связь по каким-либо причинам пропадает, панель самостоятельно выводит сообщение об этом.

Далее перейдем к главной программе контроллера. Секция объявлений переменных:

```
PROGRAM PLC_PRG
VAR
    (*режим управления*)
    state:SINT:=0; (*-1 - авария,1 - авт., 2 - дист.*)
    (*таймеры связи*)
    SCADA_EXCHANGE_WATCHDOG_1, SCADA_EXCHANGE_WATCHDOG_0 : TON;
    OP_EXCHANGE_WATCHDOG_1, OP_EXCHANGE_WATCHDOG_0 : TON;
    (*таймер регулирования*)
    STOP_CONTROL_TIMER:TON;
```

```
     (*флаги ошибок связи*)
     SCADA_FAIL:BOOL:=FALSE;
     OP_FAIL:BOOL:=FALSE;
END_VAR
```

Секция кода:

```
(*Запись по интерфейсам*)
y_to_SCADA:=y_to_OP:=y;
y_ref_to_SCADA:=y_ref_to_OP:=y_ref;
u_to_SCADA:=u_to_OP:=u;
block_panel_to_SCADA:= block_panel_to_OP:= block_panel;
state_to_SCADA:=state_to_OP:=state;
res_to_SCADA:=res_to_OP:=res;

(*Проверка связи со SCADA и панелью*)
(*по умолчанию – все нормально*)
SCADA_link_to_OP:=OP_link_to_SCADA:=TRUE;
SCADA_FAIL:=OP_FAIL:=FALSE;
(*ретрансляция tic_tac*)
tic_tac_to_SCADA:= tic_tac_from_SCADA;
tic_tac_to_OP:= tic_tac_from_OP;
(*запуск таймеров*)
SCADA_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_SCADA, PT:=t#5s);
SCADA_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_SCADA, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_OP, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_OP, PT:=t#5s);
(*установка флагов*)
SCADA_FAIL:= SCADA_EXCHANGE_WATCHDOG_1.Q OR
             SCADA_EXCHANGE_WATCHDOG_0.Q;
OP_FAIL:= OP_EXCHANGE_WATCHDOG_1.Q OR
          OP_EXCHANGE_WATCHDOG_0.Q;
no_HMI:= SCADA_FAIL AND OP_FAIL;

block_panel:=block_panel_from_SCADA;

(*Обработка ошибок связи*)
IF  SCADA_FAIL THEN
     block_panel:=OP_FAIL;
     SCADA_link_to_OP:=FALSE;
     y_ref_from_SCADA:=y_ref;
     y_ref_update_from_SCADA:=FALSE;
END_IF
IF OP_FAIL THEN
     block_panel:=TRUE;
     OP_link_to_SCADA:=FALSE;
     y_ref_from_OP:=y_ref;
     y_ref_update_from_OP:=FALSE;
END_IF

(*Обновление задания*)
IF y_ref_update_from_SCADA AND NOT SCADA_FAIL THEN
     y_ref:=y_ref_from_SCADA;
END_IF
```

```
IF y_ref_update_from_OP AND NOT block_panel THEN
     y_ref:=y_ref_from_OP;
END_IF

(*Формирование сигнала сброса регулятора*)
res:= (res_from_SCADA AND NOT SCADA_FAIL) OR
      (res_from_OP AND NOT block_panel);

(*Формирование сигналов контроля для панели*)
IF y_ref - y  < -10 THEN
     err_gt_10_to_OP:=TRUE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=FALSE;
ELSIF y_ref - y  > 10 THEN
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=TRUE;
     norma_to_OP:=FALSE;
ELSE
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=TRUE;
END_IF

stop:=FALSE; (*по умолчанию оборудование работает*)
(*Режимы управления*)
CASE state OF
-1: (*авария*)
     y_ref:=0;
     u_manual:=0;
     res:=TRUE;
     stop:=TRUE;
     IF reset_process THEN
          state:=0;
     END_IF
0: (*ручное управление*)
     u_manual:=u_manual_from_OP;
     man:=TRUE;
     (*переходы в другие режимы*)
     IF block_panel OR NOT man_from_OP OR OP_FAIL THEN
          IF dist_from_SCADA  AND NOT SCADA_FAIL THEN
               state:=2;
          ELSE
               state:=1;
          END_IF
     END_IF
1: (*автоматическое управление*)
     u_manual:=u;
     man:=FALSE;
     (*переходы в другие режимы*)
     IF man_from_OP AND NOT block_panel THEN
          state:=0;
     ELSIF dist_from_SCADA  AND NOT SCADA_FAIL THEN
          state:=2;
```

```
     END_IF
     (*Если нет связи со SCADA и панелью, установка "безопасного"
     уровня задания*)
     IF NO_HMI THEN
          y_ref:=50;
     END_IF
     (*Если связь с HMI утеряна и процесс в течение 1 мин.
     не удается вернуть в норму – сброс регулятора и останов
     оборудования*)
     STOP_CONTROL_TIMER(IN:=no_HMI AND
               (err_gt_10_to_OP OR err_lt_m10_to_OP), PT:=t#1m);
     IF STOP_CONTROL_TIMER.Q THEN
          state:=-1;
     END_IF

2: (*дистанционное управление*)
     u_manual:=u_manual_from_SCADA;
     man:=TRUE;
     (*переходы в другие режимы*)
     IF (man_from_OP AND NOT block_panel)  THEN
          state:=0;
     ELSIF NOT dist_from_SCADA OR SCADA_FAIL THEN
          state:=1;
     END_IF
END_CASE
(*Регулятор*)
CONTROL;
(*Панель*)
PANEL;
```

Программа отправляет SCADA-системе и операторской панели текущие значения регулируемой величины, задания, управляющего воздействия, а также оповещает их о блокировке панели, текущем режиме управления и сбросе регулятора.

Далее происходит проверка связи со SCADA-системой и панелью. Предварительно переменные, предназначенные для оповещения о наличии связи, и флаги наличия связи устанавливаются в состояние, соответствующее «оптимистическому» прогнозу. Сигналы `tic_tac` от SCADA и операторской панели ретранслируются обратно в SCADA и панель (там они должны инвертироваться). Два из четырех таймеров запускаются в зависимости от значений сигналов `tic_tac`. Если в течение 5 с какой-либо запущенный таймер не будет остановлен по причине инверсии сигнала `tic_tac`, установятся флаги `SCADA_FAIL` или `OP_FAIL`.

Обработка ошибок связи состоит в следующем. При обрыве связи со SCADA-системой блокировка панели снимается (при условии, что с ней самой связь остается) и панели сообщается об обрыве связи со SCADA. Переменной, хранящей задание от SCADA, присваивается значение текущего задания, а приказ обновить задание отменяется. При обрыве связи с операторской панелью она блокируется, SCADA-системе сообщается об обрыве связи с панелью. Пе-

ременной, хранящей задание от панели, присваивается значение текущего задания, а приказ обновить задание отменяется.

Отметим, что еще до обработки ошибок в переменную `block_panel` заносится последнее значение, полученное от SCADA. Таким образом, оно может быть «пересмотрено» в случае обрыва связи со SCADA (принудительное снятие блокировки) или панелью (принудительная установка блокировки).

Далее происходит обновление текущего задания и сигнала сброса регулятора по сигналам со SCADA-системы или операторской панели при обязательном наличии связи с ними, формируются сигналы контроля процесса регулирования для панели.

Обработка режимов регулирования производится в конструкции `CASE OF` в зависимости значения переменной `state`:

– в режиме авария сигналы задания и ручного управления обнуляются, регулятор сбрасывается и оборудование останавливается. При поступлении аппаратного сигнала сброса происходит переход в режим ручного управления (который должен быть подготовлен на панели управления, иначе будет «перескок» в автоматический режим);

– в режиме ручного управления текущее значение управляющего сигнала «считывается» с панели оператора, сигнал перевода программного регулятора в ручной режим устанавливается в `TRUE`. Переход в другие режимы осуществляется в случае блокировки панели, «отказа» от ручного управления со стороны оператора панели или потери связи с ней. При этом, если получен приказ перейти в режим дистанционного управления при наличии связи со SCADA, этот приказ выполняется, в противном случае происходит переход в режим автоматического управления;

– в режиме автоматического управления сигнал ручного управления «повторяет» значение текущего управления для обеспечения безударности будущего переключения, а сигнал перевода программного регулятора в ручной режим устанавливается в `FALSE`. Переходы в режимы ручного и дистанционного управления производятся по приказам от панели и SCADA при наличии связи с ними. При полной потери связи с человеко-машинным интерфейсом устанавливается «безопасный» уровень задания и с помощью специального таймера контролируется процесс на предмет наличия ошибки регулирования внутри 10% коридора. Если в течение 1 минуты ошибка находится вне этого коридора (сработал таймер) происходит переход в режим «Авария»;

– в режиме дистанционного управления текущее значение управляющего сигнала берется из SCADA, сигнал перевода программного регулятора в ручной режим устанавливается в `TRUE`. Переход в ручной режим производится по приказу с панели, если она не заблокирована. При снятии приказа на дистанционное управление и при потере связи со SCADA система переводится в автоматический режим.

В конце программы `PLC_PRG` вызываются программа регулирования `CONTROL` и имитационная программа обслуживания визуализации (панели) `PANEL`.

3.1.1.4. Разработка SCADA-системы

Разработку SCADA-системы начнем с определением ее «интерфейса» с внешним миром. В Trace Mode – это так называемые «Источники/Приемники». Создадим в них группу OPC, внутри – группу OPC_Сервер, внутри – две OPC-группы: Read и Write.

В группе Read определим и настроим 8 компонентов-источников в соответствии со списком переменных котроллера секции «Обмен со SCADA-системой, выходы» (рис. 3.18).

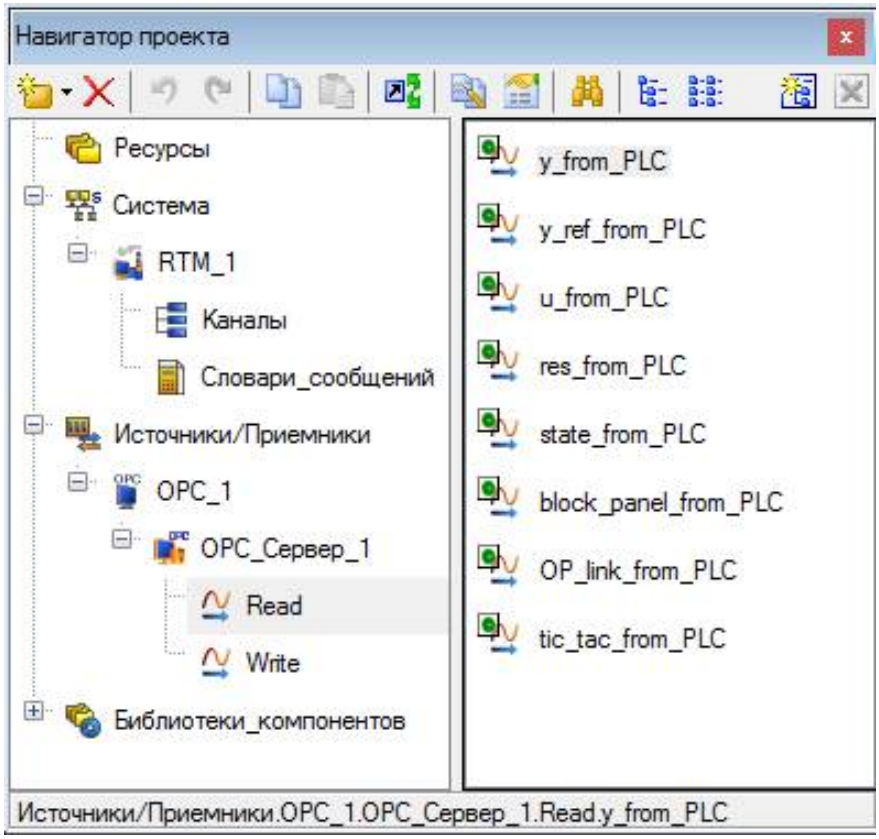


Рис. 3.18. Компоненты-источники.

Настройка канала взаимодействия по OPC при правильно сконфигурированном сервере не представляет никаких трудностей: достаточно нажать кнопку «Обзор», все остальное – понятно.

Все источники будут иметь «направление» «Input», первые три – формата «Аналог», остальные – «Дискрет».

В группе Write совершенно аналогично определим и настроим 7 компонентов-приемников в соответствии со списком переменных котроллера секции «Обмен со SCADA-системой, входы», рис. 3.19.

Все приемники будут иметь «направление» «Output», первые `y_ref_to_PLC` и `u_manual_to_PLC` – формата «Аналог», остальные – «Дискрет».

Для всех аналоговых величин методом автопостроения, т.е. просто перетаскиванием источника или приемника в «Каналы», создадим каналы. На рис. 3.20 показан полный список каналов будущей SCADA-системы.

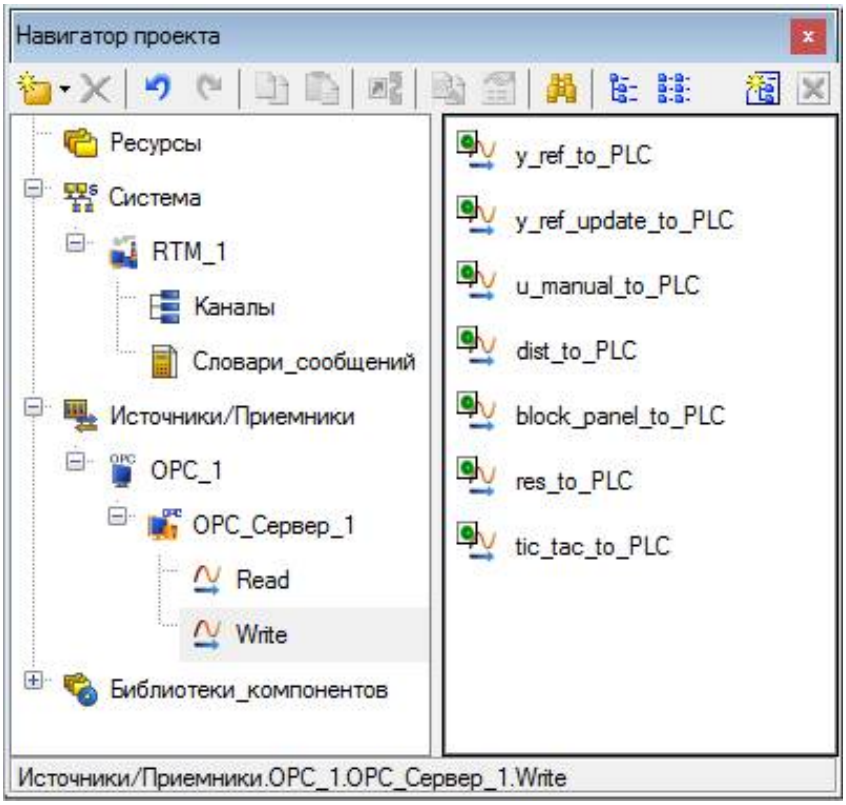


Рис. 3.19. Компоненты-приемники.

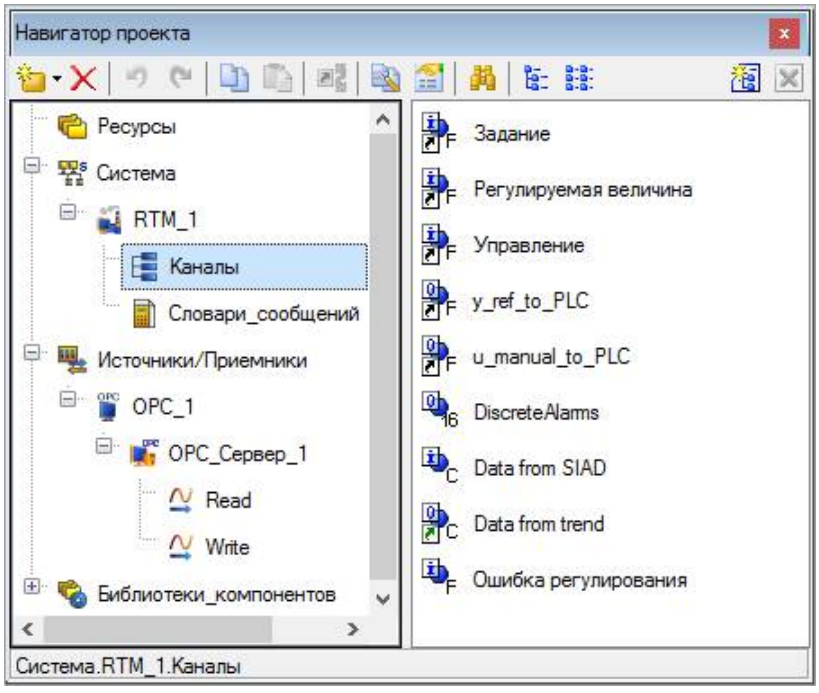


Рис. 3.20. Каналы.

Автопостроением созданы входные каналы «Задание», «Регулируемая величина», «Управление», и выходные `y_ref_to_PLC` и `u_manual_to_PLC`.

Перейдем к разработке экрана визуализации. Как и раньше, поступим следующим образом: сначала покажем конечный результат, а потом разберем детали. На рис. 3.21 показан экран в действии, запущенный под управлением профайлера Trace Mode.

Рис. 3.21. Экран визуализации в работе.

Экран позволяет:

наблюдать значения задания, регулируемой величины и управляющего воздействия (положение ползунков, текстовые поля, графики на тренде, архив);

получать оценку процесса регулирования (сигнализаторы «Повышение», «Норма», «Понижение»);

определять наличие связи контроллера с панелью оператора и SCADA-системы с контроллером (сигнализаторы) и режим работы системы (текст);

просматривать архив сообщений SCADA-системы о режиме работы, процессе регулирования, предельных уровнях управляющего сигнала и ошибках связи;

вводить и устанавливать задание и управляющий сигнал (последний – только в дистанционном режиме) с помощью ползунков и кнопок;

включать блокировку панели, сброс регулятора и дистанционное управление (кнопки) и *наблюдать подтверждения* этих действий со стороны контроллера (сигнализаторы);

извлекать данные из архива в различных временных диапазонах в файлы трех форматов (кнопки) и наблюдать «отчет» о процессе извлечения (текстовое поле).

Аргументы экрана и их привязки показаны на рис. 3.22. Привязки графических элементов экрана к его аргументам показаны на рис. 3.23 в виде текстовых надписей поверх экрана визуализации в «нерабочем» состоянии.

Экран «обслуживается» тремя программами:

Основной программой, которая занимается операциями с заданием и управляющим воздействием, формированием оценки процесса регулирования и большей части сообщений отчета тревог;

`TIC_TAC`, контролирующей связь с ПЛК и формирующей соответствующие сообщения в отчет;

`Data_from_trend`, выполняющей вспомогательные действия при извлечении данных из СПАД-архива.

Первые две программы являются каналами типа «Input», поэтому пересчитываются «автоматически». Они размещены непосредственно в «корне» узла (рис. 3.24). Третья программа – канал типа «Output» и поэтому вызывается «принудительно». Она размещена в группе «Каналы» (см. рис. 3.20).

Прежде чем приступить к программам, рассмотрим сообщения из отчета тревог и каналы, которые их формируют.

Для формирования отчета тревог создано два словаря. Один – для HEX16 («Тревоги»), другой – для FLOAT («Пересечение границ») (рис. 3.25).

Содержание словарей показано на рис. 3.26, 3.27.

В словаре «Тревоги» определены 16 сообщений (восемь пар) (некоторые из которых «пустые») выдаваемых по фактам сброса и установки младших 8 битов канала HEX16. Таким каналом у нас будет канал DiscreteAlarms, настройка которого показана на рис. 3.28.

Имя	Тип	Тип данных	Знач	Привязка
Задание	IN	REAL		Задание:Реальное значение (Система.RTM_1.Каналы)
Рег_величина	IN	REAL		Регулируемая величина:Реальное значение (Система.RTM_1.Каналы)
Управление	IN	REAL		Управление:Реальное значение (Система.RTM_1.Каналы)
state	IN	SINT		state_from-PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
Archive_msg	IN	UDINT		CALL ROOT data from siad:ERR (Система.RTM_1.Каналы)
PLC_no_link	IN	BOOL		TIC_TAC:no_link (Система.RTM_1)
Control_status	IN	INT		Основная программа:Control_status (Система.RTM_1)
regul_disc	IN	BOOL		res_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
panel_blocked	IN	BOOL		block_panel_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
OP_link	IN	BOOL		OP_link_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
block_panel	IN/OUT	BOOL		block_panel_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
u_manual	OUT	REAL		u_manual_to_PLC:Входное значение (Система.RTM_1.Каналы)
coarse_ref	OUT	BOOL		
y_ref_set	OUT	BOOL		
ref_plus	OUT	BOOL		
ref_minus	OUT	BOOL		
control_plus	OUT	BOOL		
control_minus	OUT	BOOL		
coarse_control	OUT	BOOL		
dist_set	OUT	BOOL		dist_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
res_regul	OUT	BOOL		res_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
y_ref_monitor	IN/OUT	REAL		Основная программа:y_ref_monitor (Система.RTM_1)
y_ref_monitor_set	IN/OUT	REAL		Основная программа:y_ref_monitor_set (Система.RTM_1)
control_monitor	IN/OUT	REAL		
control_monitor_set	IN/OUT	REAL		
Archive_From	IN/OUT	UDINT		CALL ROOT data from siad:DR (Система.RTM_1.Каналы)
Archive_To	IN/OUT	UDINT		
Archive_Period	IN/OUT	UINT		CALL ROOT data from siad:Параметр/Глубина выборки (Система.RTM_1.Каналы)
Archice_Q	IN/OUT	UINT		CALL ROOT data from siad:Выходное значение (Система.RTM_1.Каналы)
Data_from_trend_EXEC	IN/OUT	UINT		Data from trend:Отработать (Система.RTM_1.Каналы)

Рис. 3.22. Аргументы экрана и их привязки.

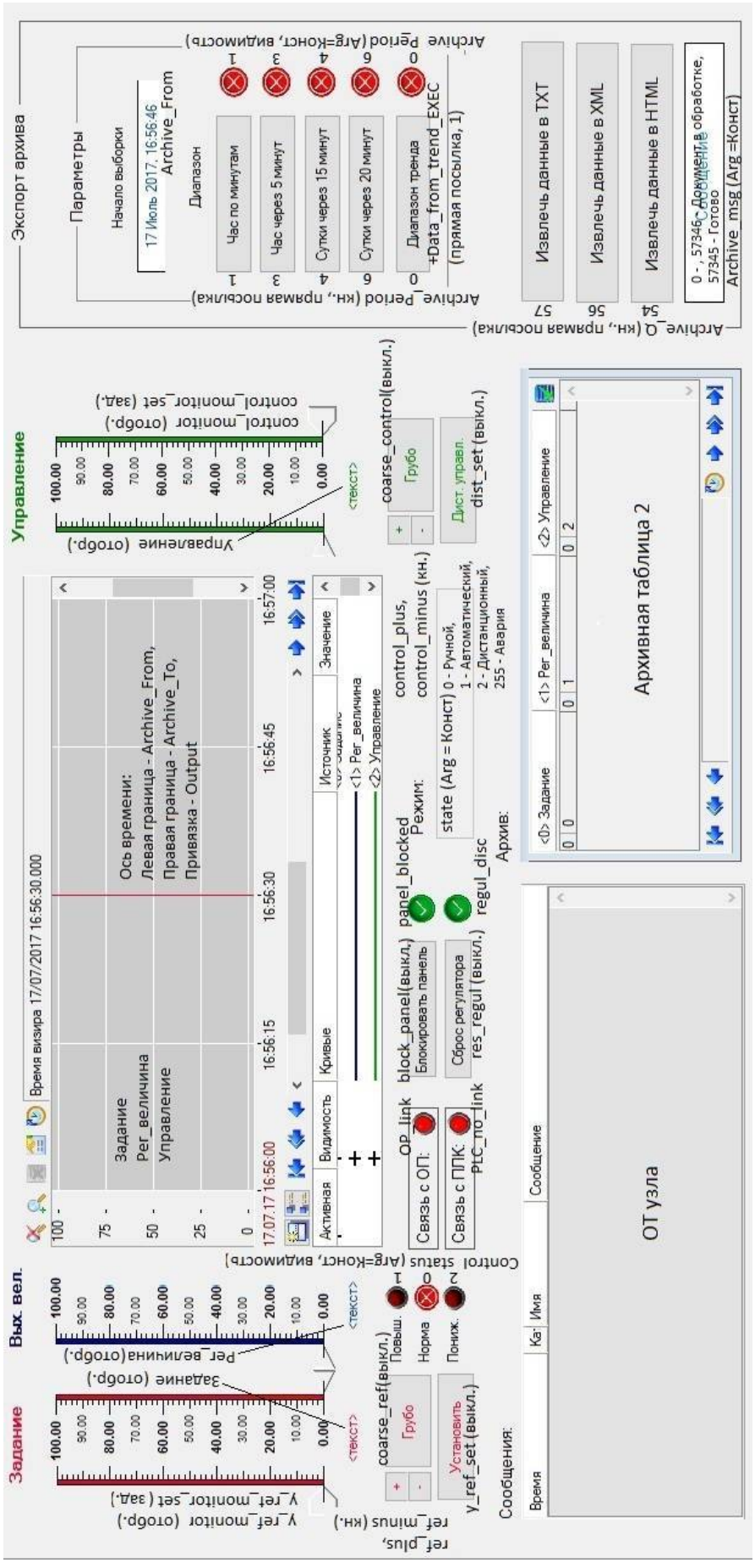


Рис. 3.23. Привязки графических элементов к аргументам экрана.

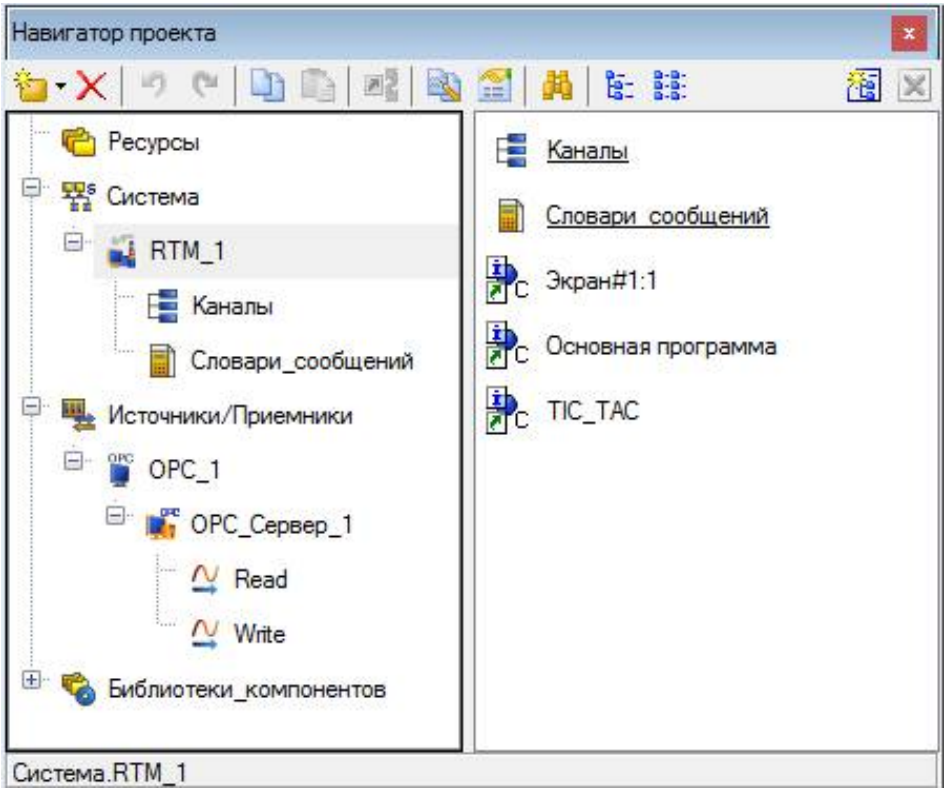


Рис. 3.24. Состав корневого уровня узла.

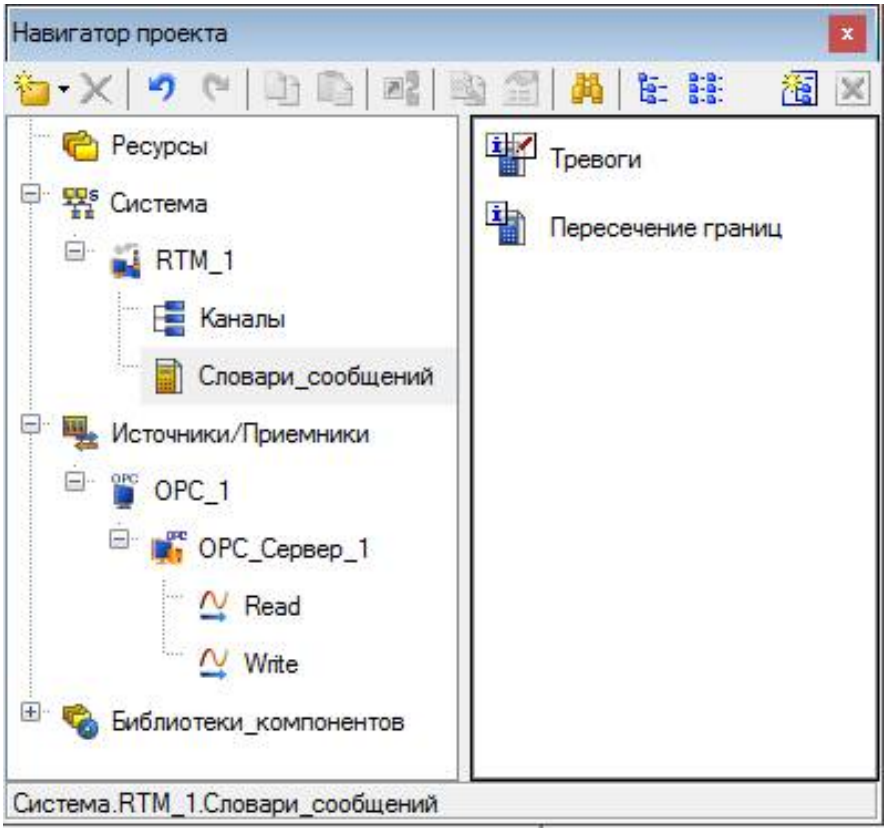


Рис. 3.25. Словари сообщений.

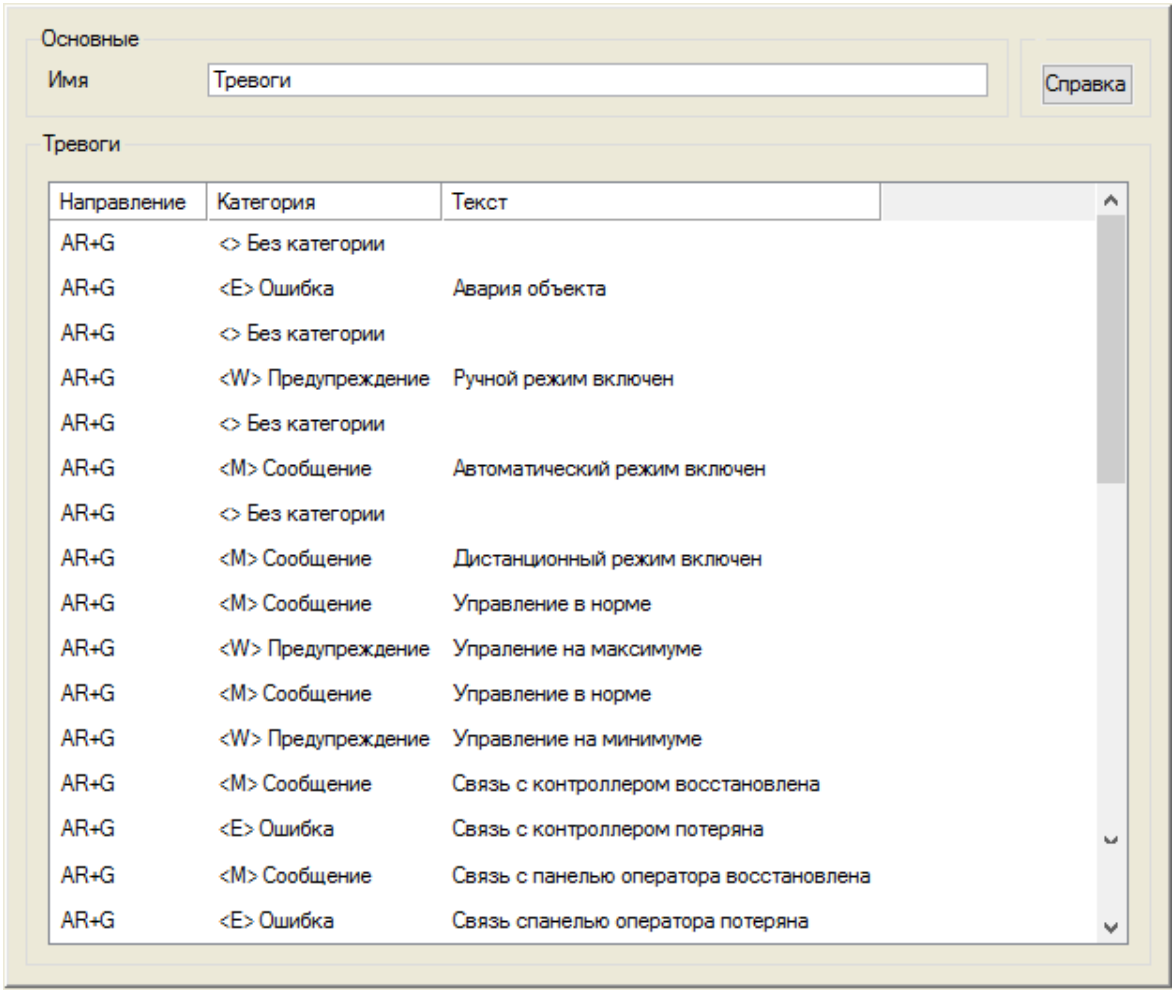


Рис. 3.26. Словарь «Тревоги».

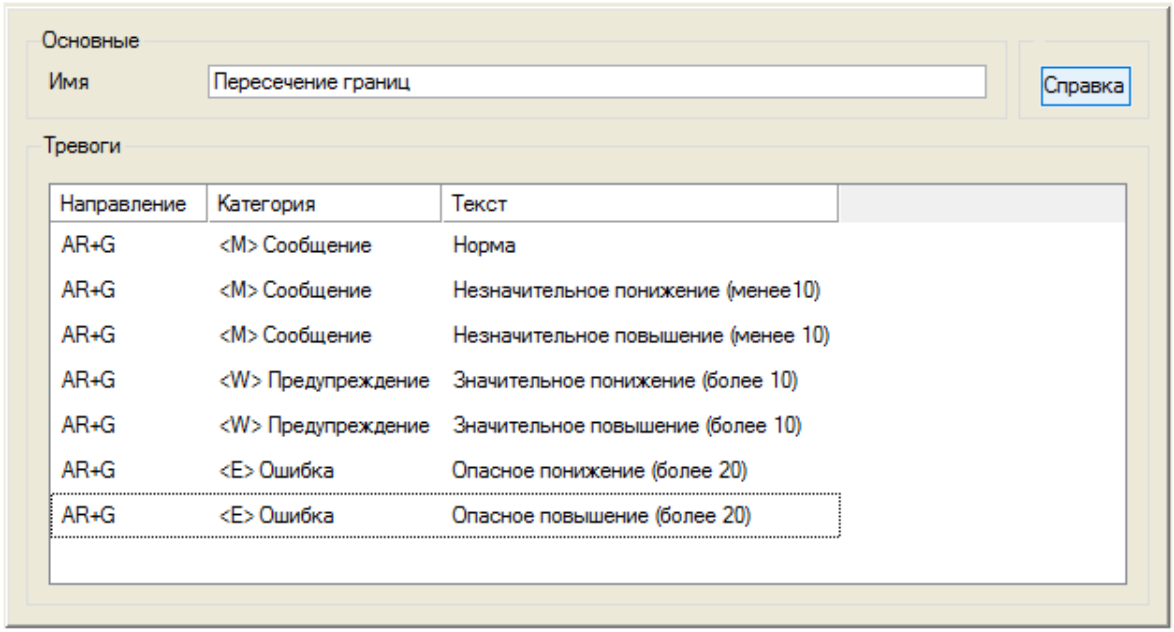


Рис. 3.27. Словарь «Пересечение границ».

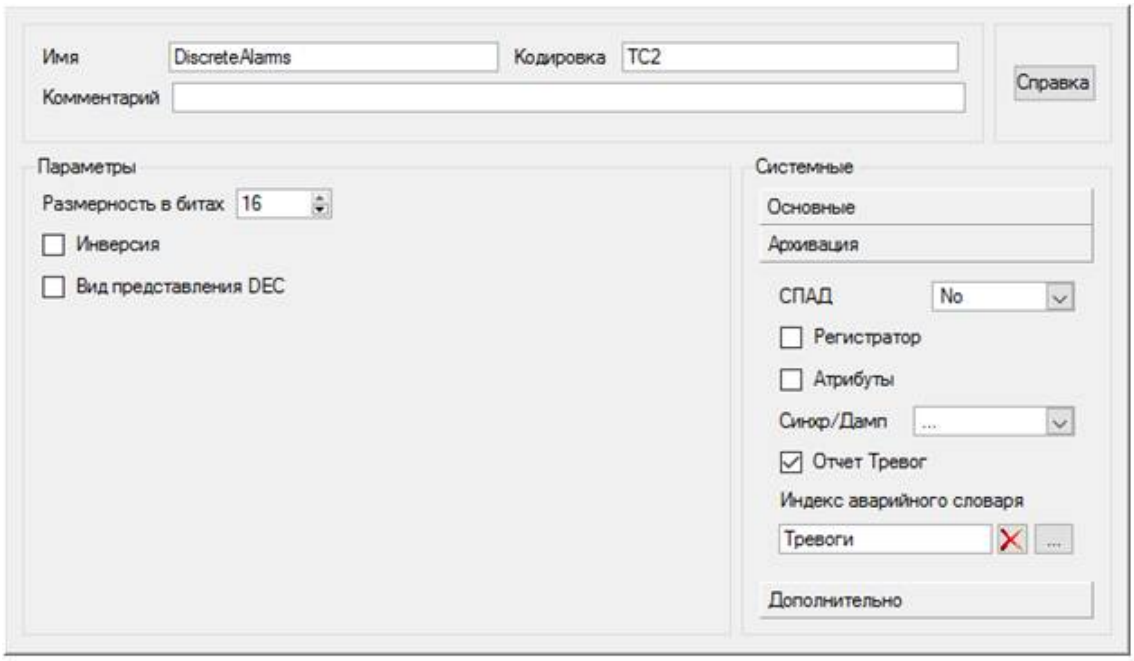


Рис. 3.28. Настройка канала DiscreteAlarms.

Сообщения из словаря «Пересечение границ» будет выдавать канал «Ошибка регулирования», в который Основная программа будет записывать значение ошибки регулирования, т.е. разницы между заданием и реальным значением регулируемой величины. Настройка канала показана на рис. 3.29.

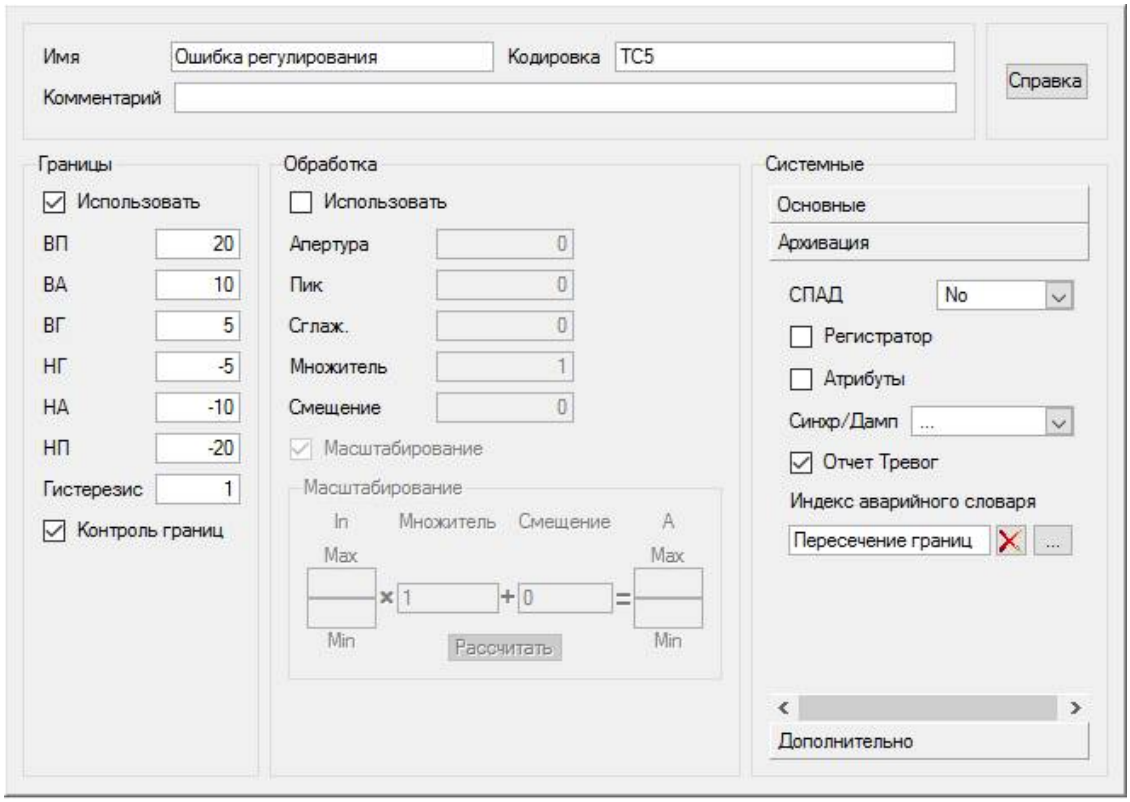


Рис. 3.29. Настройка канала «Ошибка регулирования».

Аргументы Основной программы и их привязки приведены на рис. 3.30.

Имя	Тип	Тип данн	Значен	Привязка
ref_plus	IN	BOOL		Экран#1:1:ref_plus (Система.RTM_1)
ref_minus	IN	BOOL		Экран#1:1:ref_minus (Система.RTM_1)
coarse_y_ref	IN	BOOL		Экран#1:1:coarse_ref (Система.RTM_1)
set_ref	IN	BOOL		Экран#1:1:y_ref_set (Система.RTM_1)
y_ref	IN	REAL		Задание:Реальное значение (Система.RTM_1.Каналы)
control_plus	IN	BOOL		Экран#1:1:control_plus (Система.RTM_1)
control_minus	IN	BOOL		Экран#1:1:control_minus (Система.RTM_1)
coarse_control	IN	BOOL		Экран#1:1:coarse_control (Система.RTM_1)
u	IN	REAL		Управление:Реальное значение (Система.RTM_1.Каналы)
y	IN	REAL		Регулируемая величина:Реальное значение (Система.RTM_1.Каналы)
state	IN	SINT		state_from-PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
OP_link_from_PLC	IN	BOOL	true	OP_link_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
error	OUT	REAL		Ошибка регулирования:Входное значение (Система.RTM_1.Каналы)
y_ref_to_PLC	OUT	REAL		y_ref_to_PLC:Входное значение (Система.RTM_1.Каналы)
y_ref_update	OUT	REAL		y_ref_update_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
u_manual_to_PLC	OUT	REAL		u_manual_to_PLC:Входное значение (Система.RTM_1.Каналы)
Control_status	OUT	INT		
y_ref_monitor	IN/OUT	REAL		Экран#1:1:y_ref_monitor (Система.RTM_1)
y_ref_monitor_set	IN/OUT	REAL		Экран#1:1:y_ref_monitor_set (Система.RTM_1)
control_monitor	IN/OUT	REAL		Экран#1:1:control_monitor (Система.RTM_1)
control_monitor_set	IN/OUT	REAL		Экран#1:1:control_monitor_set (Система.RTM_1)
DiscreteAlarms	IN/OUT	UINT		DiscreteAlarms:Входное значение (Система.RTM_1.Каналы)

Рис. 3.30. Аргументы Основной программы.

Имя	Тип	Тип данных	З	Привязка
tic_tac_from_PLC	IN	BOOL		tic_tac_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
tic_tac_to_PLC	OUT	BOOL		tic_tac_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
no_link	OUT	BOOL		DiscreteAlarms:Бит 7 (Система.RTM_1.Каналы)

Рис. 3.31. Аргументы программы TIC_TAC

Листинг программы:

```
PROGRAM
     VAR_INPUT ref_plus : BOOL; END_VAR
     VAR_INPUT ref_minus : BOOL; END_VAR
     VAR_INPUT coarse_y_ref : BOOL; END_VAR
     VAR_INPUT set_ref : BOOL; END_VAR
     VAR_INPUT y_ref : REAL; END_VAR
     VAR_INPUT control_plus : BOOL; END_VAR
     VAR_INPUT control_minus : BOOL; END_VAR
     VAR_INPUT coarse_control : BOOL; END_VAR
     VAR_INPUT u : REAL; END_VAR
     VAR_INPUT y : REAL; END_VAR
     VAR_INPUT state : SINT; END_VAR
     VAR_INPUT OP_link_from_PLC : BOOL := true; END_VAR
     VAR_OUTPUT error : REAL; END_VAR
     VAR_OUTPUT y_ref_to_PLC : REAL; END_VAR
     VAR_OUTPUT y_ref_update : REAL; END_VAR
     VAR_OUTPUT u_manual_to_PLC : REAL; END_VAR
     VAR_OUTPUT Control_status : INT; END_VAR
     VAR_INOUT y_ref_monitor : REAL; END_VAR
     VAR_INOUT y_ref_monitor_set : REAL; END_VAR
     VAR_INOUT control_monitor : REAL; END_VAR
     VAR_INOUT control_monitor_set : REAL; END_VAR
     VAR_INOUT DiscreteAlarms : UINT; END_VAR

     //ЗАДАНИЕ
     IF coarse_y_ref THEN //грубый ввод задания
          //ползунок идет за мышью
     y_ref_monitor:=y_ref_monitor_set;
     ELSE //точный ввод, реакция на кнопки "+","-"
          IF ref_plus THEN
               y_ref_monitor:=y_ref_monitor+1;
          ELSIF ref_minus THEN
               y_ref_monitor:=y_ref_monitor-1;
          END_IF;
          y_ref_monitor_set:=y_ref_monitor;
     END_IF;
     IF set_ref THEN //Установка задания
          y_ref_update:=TRUE;
          y_ref_to_PLC:=y_ref_monitor;
     ELSE
          y_ref_update:=FALSE;
          y_ref_to_PLC:=y_ref; //отправляем то, что получили
     END_IF;
//ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ
IF state==2 THEN //В режиме дистанционного управления
     IF coarse_control THEN //Грубый ввод задания
          //ползунок идет за мышью
          control_monitor:=control_monitor_set;
     ELSE //точный ввод, реакция на кнопки "+","-"
          IF control_plus THEN
               control_monitor:=control_monitor+1;
          ELSE IF control_minus THEN
```

```
                    control_monitor:=control_monitor-1;
               END_IF;
               control_monitor_set:=control_monitor;
          END_IF;
          //отправляем то, что ввели
          u_manual_to_PLC:=control_monitor;
     ELSE
          control_monitor:=u;
          control_monitor_set:=u;
          u_manual_to_PLC:=u; //отправляем то, что получили
     END_IF;
     //ФОРМИРОВАНИЕ СТАТУСА
     error:=y_ref-y; //ошибка регулирования
     IF error > 10 THEN
          Control_status:= 2;
     ELSIF error < -10 THEN
          Control_status:= 1;
     ELSE
          Control_status:= 0;
     END_IF;
     //ФОРМИРОВАНИЕ СООБЩЕНИЙ
     //сброс тех битов, которые могут быть установлены
     DiscreteAlarms:= DiscreteAlarms & 2#1000000;
     CASE state OF
     255: DiscreteAlarms:=DiscreteAlarms | 2#1; //Авария
     0: DiscreteAlarms:=DiscreteAlarms | 2#10;  //Ручной режим
     1: DiscreteAlarms:=DiscreteAlarms | 2#100; //Автомат. режим
     2: DiscreteAlarms:=DiscreteAlarms | 2#1000;//Дистанц. режим
     END_CASE;
     IF u > 95 THEN //Управление на максимуме
          DiscreteAlarms:=DiscreteAlarms | 2#10000;
     ELSIF u < 5 THEN //Управление на минимуме
          DiscreteAlarms:=DiscreteAlarms | 2#100000;
     END_IF;
     IF OP_link_from_PLC THEN //Связь с ПО потеряна
          DiscreteAlarms:=DiscreteAlarms | 2#10000000;
     END_IF;
END_PROGRAM
```

Переменные программы имеют осмысленные имена, полностью или частично совпадающие с именами соответствующих привязок. Благодаря этому, а также комментариям, программа в основном должна быть понятной.

Некоторых пояснений все же, возможно, требует фрагмент, ответственный за формирование сообщений. Здесь используются побитовые операции И и ИЛИ. Сначала все биты переменной `DiscreteAlarms`, привязанной к входному значению одноименного канала, за исключением седьмого (если считать, начиная с единицы), обнуляются накладыванием соответствующей маски с помощью операции побитового И. Потом при наличии определенных условий устанавливаются биты с первого по восьмой (кроме седьмого) накладыванием соответствующих масок с помощью операции побитового ИЛИ.

У канала DiscreteAlarms, как у любого канала HEX, есть возможность непосредственного обращения к отдельным битам через соответствующие атри-

буты, поэтому можно было бы вместо одной переменной типа UINT задействовать семь переменных типа BOOL, привязав их по отдельности к атрибутам канала DiscreteAlarms. Однако мы не стали этого делать, чтобы не увеличивать и без того немалое количество аргументов программы. В то же время в программе TIC_TAC, которая работает с единственным (как раз седьмым) битом используется именно такой подход.

Аргументы программы TIC_TAC и их привязки показаны на рис. 3.31.

Листинг программы:

```
PROGRAM
     VAR_INPUT tic_tac_from_PLC : BOOL; END_VAR
     VAR_OUTPUT tic_tac_to_PLC : BOOL; END_VAR
     VAR_OUTPUT no_link : BOOL; END_VAR
     VAR_INOUT DiscreteAlarms : UINT; END_VAR


     //РЕВЕРС TIC_TAC
     tic_tac_to_PLC:=NOT tic_tac_from_PLC;
     //ПОДСЧЕТ ЦИКЛОВ ОДИНАКОВЫХ ЗНАЧЕНИЙ TIC_TAC
     IF tic_tac_from_PLC THEN
          counter1:=counter1+(counter1<10);
          counter0:=0;
     ELSE
          counter0:=counter0+(counter0<10);
          counter1:=0;
     END_IF;
     //ПОТЕРЯ СВЯЗИ ПРИ counterX >9
     no_link:= (counter1 > 9) OR (counter0 > 9);
END_PROGRAM
```

Программа оперирует глобальными переменными `counter0` и `counter1` типа SINT (их объявление не показано), которые выполняют ту же роль, что и таймеры в программе контроллера. Основное отличие состоит в том, что здесь контролируется не время между инверсиями переменной `tic_tac`, а число циклов монитора. Программа фиксирует факт отсутствия связи, если инверсии не происходит в течение 10 циклов. Само время цикла монитора настраивается в редакторе узла и по умолчанию составляет 0,55 секунды.

Перейдем к вопросам архивирования.

Отчет тревог, формирование которого мы уже рассмотрели, подлежит записи в текстовый файл. Для этого достаточно в редакторе узла, в соответствующей вкладке задать имя файла и разрешить запись (рис. 3.32).

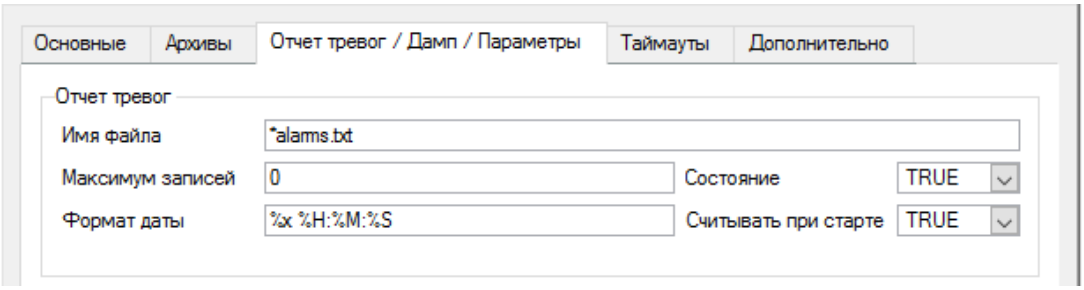


Рис. 3.32. Настройка отчета тревог.

«Звездочка» перед именем файла означает, что он будет размещен в папке узла.

Помимо отчета тревог, система будет вести СПАД (структурированный промышленный архив данных), или англ. SIAD (Structured Industry Archive of Data). В него будут записываться выходные значения каналов «Задание», «Регулируемая величина» и «Управление». Для настройки архива достаточно в редакторе узла задать имя файла и включить архивирование (рис. 3.33.), а в настройках каналов указать номер СПАД-архива (рис. 3.34).

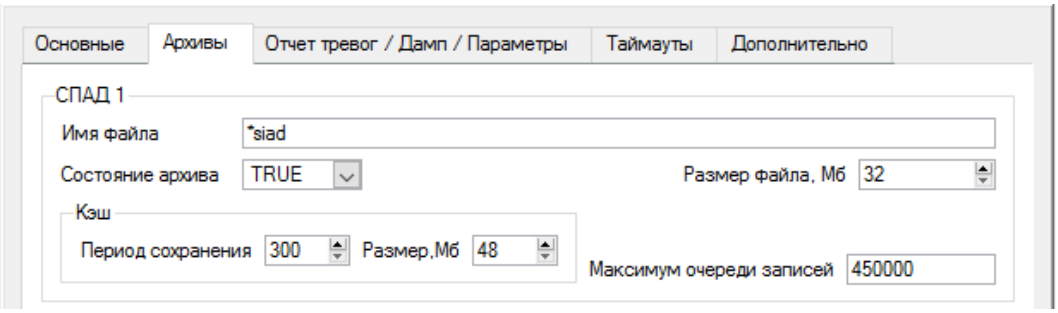


Рис. 3.33. Настройка СПАД-архива.

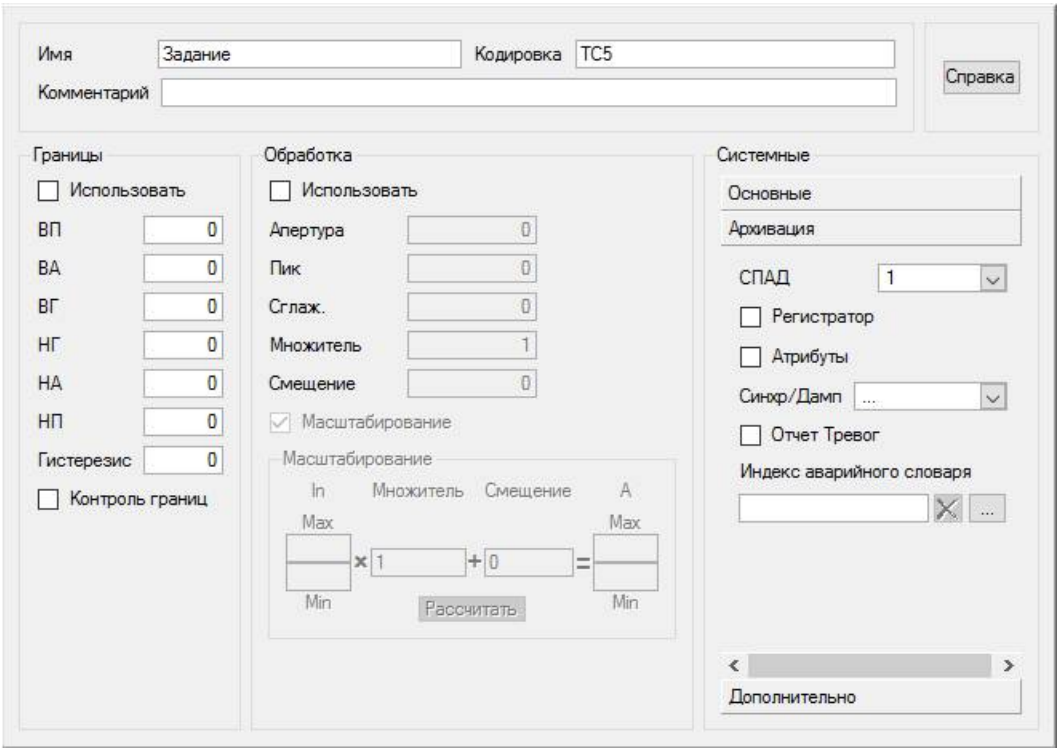


Рис. 3.34. Настройка канала для архивирования.

Просмотр архива на экране монитора можно осуществлять с помощью графических элементов «Тренд» и «Архивная таблица 2».

Для экспорта архива в файлы форматов TXT, XML В HTML создадим канал Data from SIAD класса CALL с типом вызова ROOT (рис. 3.35).

У канала должны быть установлены флаг «Запрос времени значения» и номер архива (рис. 3.36).

Аргументы канала привязываются к реальным значениям архивируемых каналов (рис. 3.37).

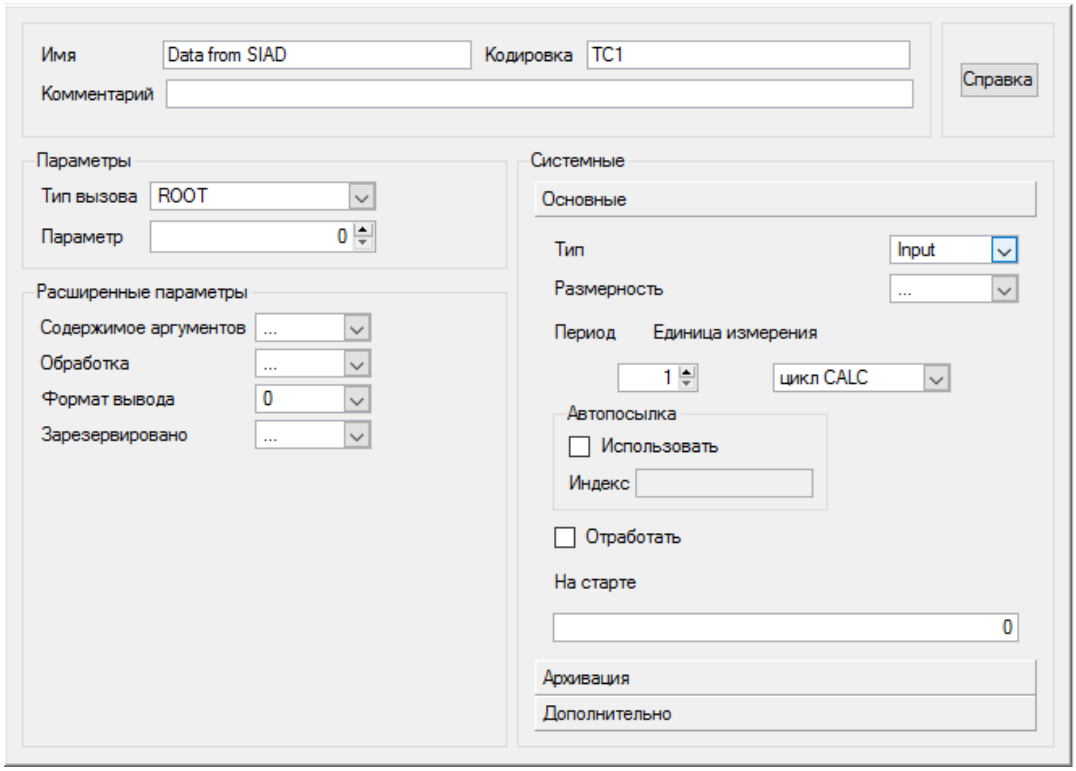


Рис. 3.35. Настройка канала Data from SIAD.

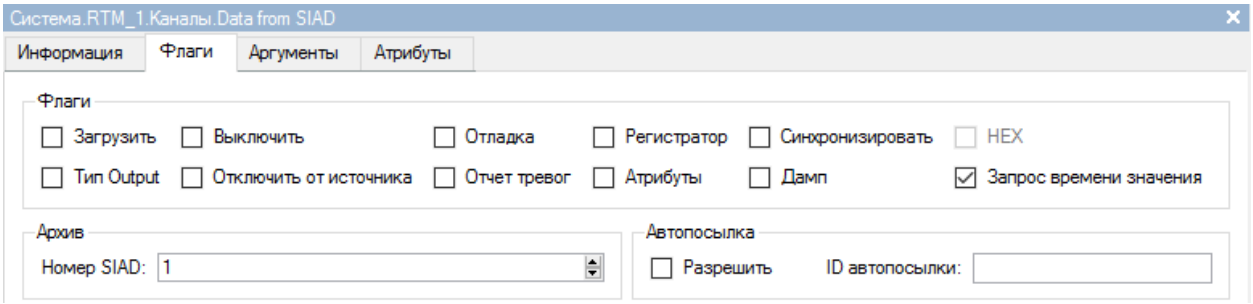


Рис. 3.36. Флаги канала Data from SIAD.

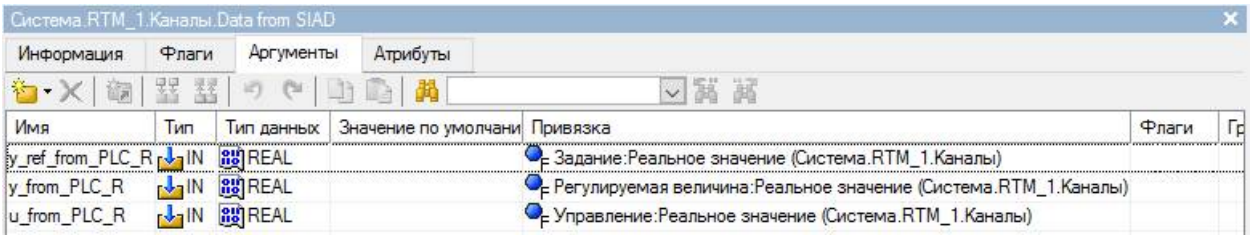


Рис. 3.37. Аргументы канала Data from SIAD.

Канал будет «управляться» программой Data from trend и элементами экрана визуализации (см. рис. 3.23 – тренд и правая панель, рис. 3.22 – аргументы экрана Archive_From, Archive_To, Archive_Period, Archive_Q и Archive_msg).

По сути, речь идет о манипуляциях со следующими атрибутами канала:

1. Выходное значение (9, Q). Запись в этот атрибут определенных значений и приводит к формированию файла с выборкой нужного формата. Мы используем значения 54 (HTML), 56 (XML) и 57 (TXT). Сама запись происходит при нажатии кнопок внизу правой панели.

2. Параметр/Глубина выборки (34, FPrnt), определяющий величину интервалов, на которые разбивается диапазон выборки, и сам диапазон. Мы ис-

пользуем следующие значения: 1 – час по минутам, 3 – час через 5 минут, 4 –сутки через 15 минут, 6 – сутки через 20 минут и 0 – интервал задается атрибутом dT, причем из него выводятся все значения. Атрибутом управляют кнопки правой панели через аргумент экрана Archive_Period.

3. DR(59) – задает время начала выборки. Через аргумент экрана Archive_From этим атрибутом управляют графические элементы «Тренд» и «Дата и время», кроме того он устанавливается программой Data from trend.

4. ERR (240) – сообщения, точнее их численные коды. Записываются в аргумент экрана Archive_msg и отображаются в текстовом поле внизу правой панели.

5. dT (252) – определяет интервал выборки при FPrnt = 0. Этот атрибут рассчитывается программой Data from trend так, чтобы выбрать данные по текущему диапазону тренда.

Программа Data from trend принудительно вызывается при нажатии кнопки «Диапазон тренда». Аргументы программы и их привязки показаны на рис. 3.38.

Имя	Тип	Тип данны	Зна	Привязка
Left_Border_Trend	IN	UDINT		Экран#1:1:Archive_From (Система.RTM_1)
Right_Border_Trend	IN	UDINT		Экран#1:1:Archive_To (Система.RTM_1)
CALL_DR	OUT	UDINT		CALL ROOT data from siad:DR (Система.RTM_1.Каналы)
CALL_dT	OUT	UDINT		CALL ROOT data from siad:dT (Система.RTM_1.Каналы)

Рис. 3.38. Аргументы программы Data from trend.

Код программы:

```
PROGRAM
      VAR_INPUT Left_Border_Trend : UDINT; END_VAR
      VAR_INPUT Right_Border_Trend : UDINT; END_VAR
      VAR_OUTPUT CALL_DR : UDINT; END_VAR
      VAR_OUTPUT CALL_dT : UDINT; END_VAR

      CALL_DR:=Left_Border_Trend;
      CALL_dT:=Right_Border_Trend - Left_Border_Trend - 1;
      if CALL_dT <60 then
            CALL_dT:=60;
      end_if;
END_PROGRAM
```

В программе предусмотрено длины ограничение интервала выборки по минимуму: не менее 60 сек.

3.1.1.5. Апробация системы

«Развертывание» имитационной системы выполнялось в следующей последовательности:

1) запуск виртуального контроллера;

2) загрузка программы CoDeSys в виртуальный контроллер и ее запуск;

3) запуск Matlab и Simulink-модели;

4) запуск профайлера Trace Mode.

После запуска всех программ опробованы все возможности системы:

автоматическое регулирование: отработка изменения задания как с «операторской панели», так и от SCADA-системы, отработка возмущений, сформированных «вручную» в Simulink-диаграмме;

ручное и дистанционное управление;

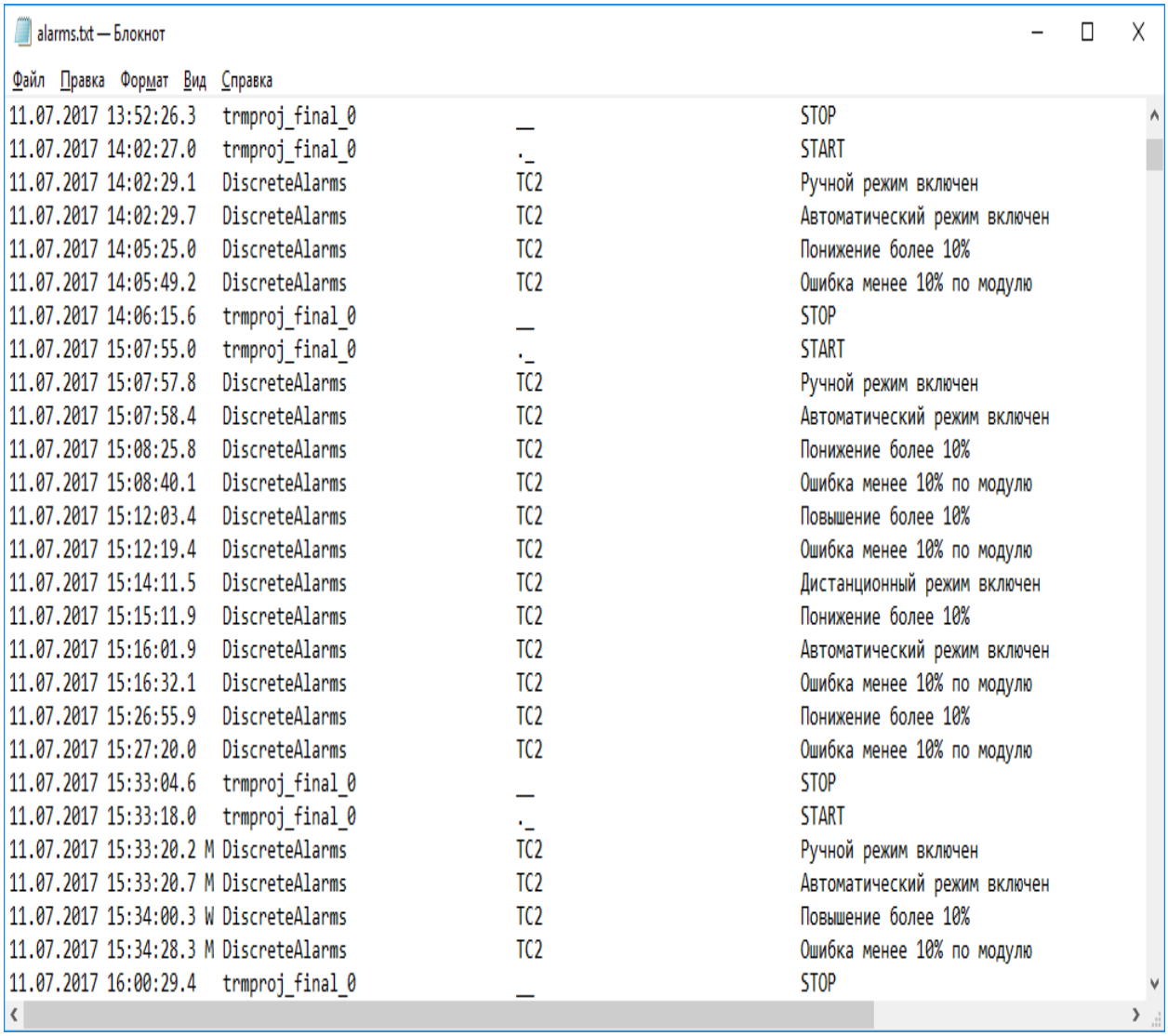
сигнализация о режимах работы, состоянии процесса регулирования на панели оператора и экране профайлера;

блокировка панели оператора со стороны SCADA-системы и автоматическое разблокирование при потере связи (потерю связи можно смоделировать, просто остановив профайлер);

оповещение оператора SCADA-системы о потере связи с панелью (имитируется специальной кнопкой в визуализации CoDeSys) и контроллером (имитируется остановкой программы контроллера);

формирование, вывод и сохранение в файле отчета тревог (рис. 3.39);

формирование и вывод в документы СПАД-архива (рис. 3.40 – 3.42).


alarms.txt — Блокнот

Файл Правка Формат Вид Справка

11.07.2017 13:52:26.3	trmproj_final_0	__	STOP
11.07.2017 14:02:27.0	trmproj_final_0	._	START
11.07.2017 14:02:29.1	DiscreteAlarms	TC2	Ручной режим включен
11.07.2017 14:02:29.7	DiscreteAlarms	TC2	Автоматический режим включен
11.07.2017 14:05:25.0	DiscreteAlarms	TC2	Понижение более 10%
11.07.2017 14:05:49.2	DiscreteAlarms	TC2	Ошибка менее 10% по модулю
11.07.2017 14:06:15.6	trmproj_final_0	__	STOP
11.07.2017 15:07:55.0	trmproj_final_0	._	START
11.07.2017 15:07:57.8	DiscreteAlarms	TC2	Ручной режим включен
11.07.2017 15:07:58.4	DiscreteAlarms	TC2	Автоматический режим включен
11.07.2017 15:08:25.8	DiscreteAlarms	TC2	Понижение более 10%
11.07.2017 15:08:40.1	DiscreteAlarms	TC2	Ошибка менее 10% по модулю
11.07.2017 15:12:03.4	DiscreteAlarms	TC2	Повышение более 10%
11.07.2017 15:12:19.4	DiscreteAlarms	TC2	Ошибка менее 10% по модулю
11.07.2017 15:14:11.5	DiscreteAlarms	TC2	Дистанционный режим включен
11.07.2017 15:15:11.9	DiscreteAlarms	TC2	Понижение более 10%
11.07.2017 15:16:01.9	DiscreteAlarms	TC2	Автоматический режим включен
11.07.2017 15:16:32.1	DiscreteAlarms	TC2	Ошибка менее 10% по модулю
11.07.2017 15:26:55.9	DiscreteAlarms	TC2	Понижение более 10%
11.07.2017 15:27:20.0	DiscreteAlarms	TC2	Ошибка менее 10% по модулю
11.07.2017 15:33:04.6	trmproj_final_0	__	STOP
11.07.2017 15:33:18.0	trmproj_final_0	._	START
11.07.2017 15:33:20.2 M	DiscreteAlarms	TC2	Ручной режим включен
11.07.2017 15:33:20.7 M	DiscreteAlarms	TC2	Автоматический режим включен
11.07.2017 15:34:00.3 W	DiscreteAlarms	TC2	Повышение более 10%
11.07.2017 15:34:28.3 M	DiscreteAlarms	TC2	Ошибка менее 10% по модулю
11.07.2017 16:00:29.4	trmproj_final_0	__	STOP

Рис. 3.39. Фрагмент файла архива тревог.

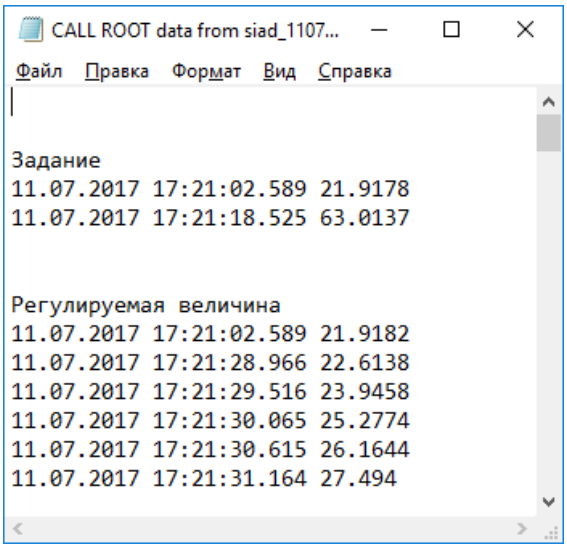


Рис. 3.40. Фрагмент TXT-файла выборки из СПАД-архива.

Книга1 - Microsoft Excel

Главная Вставка Разметка страницы Формулы Данные Рецензирование Вид

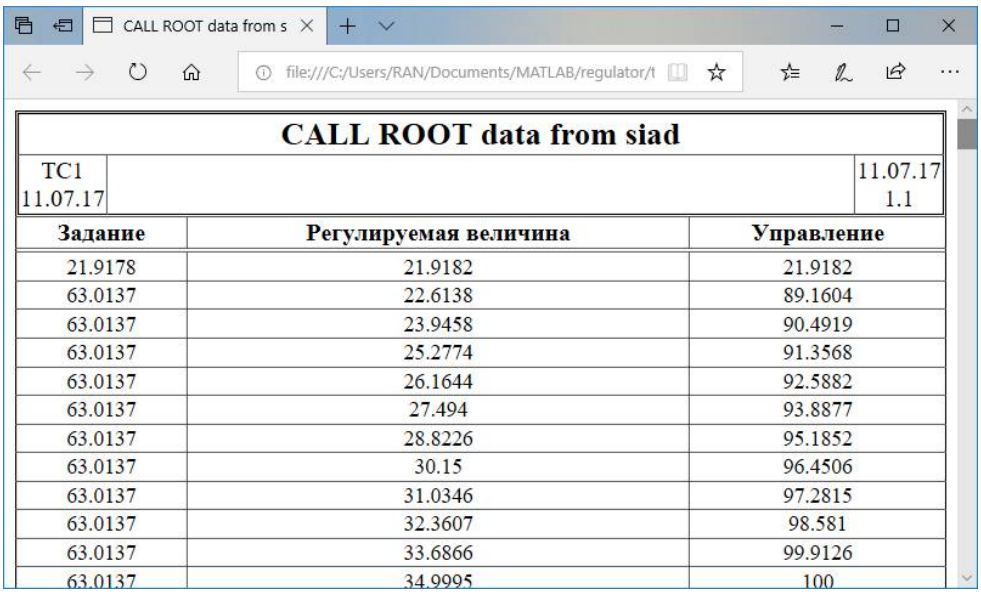
O12

	H	I	J	K	L	M
1	name2	code3	comi	dt	value	fromto
2	Задание	TC5		11.07.2017 17:21:02.589	21,9178	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
3	Задание	TC5		11.07.2017 17:21:18.525	63,0137	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
4	Регулируемая величина	TC5		11.07.2017 17:21:02.589	21,9182	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
5	Регулируемая величина	TC5		11.07.2017 17:21:28.966	22,6138	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
6	Регулируемая величина	TC5		11.07.2017 17:21:29.516	23,9458	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
7	Регулируемая величина	TC5		11.07.2017 17:21:30.065	25,2774	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
8	Регулируемая величина	TC5		11.07.2017 17:21:30.615	26,1644	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
9	Регулируемая величина	TC5		11.07.2017 17:21:31.164	27,494	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
10	Регулируемая величина	TC5		11.07.2017 17:21:31.714	28,8226	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
11	Регулируемая величина	TC5		11.07.2017 17:21:32.265	30,15	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
12	Регулируемая величина	TC5		11.07.2017 17:21:32.814	31,0346	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
13	Регулируемая величина	TC5		11.07.2017 17:21:33.363	32,3607	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
14	Регулируемая величина	TC5		11.07.2017 17:21:33.913	33,6866	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
15	Регулируемая величина	TC5		11.07.2017 17:21:34.463	34,9995	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
16	Регулируемая величина	TC5		11.07.2017 17:21:35.012	35,8605	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
17	Регулируемая величина	TC5		11.07.2017 17:21:35.562	37,1305	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
18	Регулируемая величина	TC5		11.07.2017 17:21:36.111	38,3754	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
19	Регулируемая величина	TC5		11.07.2017 17:21:36.661	39,5956	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
20	Регулируемая величина	TC5		11.07.2017 17:21:37.210	40,3957	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
21	Регулируемая величина	TC5		11.07.2017 17:21:37.760	41,5759	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30
22	Регулируемая величина	TC5		11.07.2017 17:21:38.310	42,7328	start=11.07.2017 17:21:30 end=11.07.2017 17:22:30

Лист1 Лист2 Лист3

Готово 100%

Рис. 3.41. Фрагмент XML-файла выборки из СПАД-архива (открыт в MS Excel и незначительно отредактирован).



CALL ROOT data from siad

TC1 11.07.17		11.07.17 1.1
Задание	**Регулируемая величина**	**Управление**
21.9178	21.9182	21.9182
63.0137	22.6138	89.1604
63.0137	23.9458	90.4919
63.0137	25.2774	91.3568
63.0137	26.1644	92.5882
63.0137	27.494	93.8877
63.0137	28.8226	95.1852
63.0137	30.15	96.4506
63.0137	31.0346	97.2815
63.0137	32.3607	98.581
63.0137	33.6866	99.9126
63.0137	34.9995	100

Рис. 3.42. Фрагмент HTML-файла выборки из СПАД-архива.

3.1.2. Система автоматического регулирования с ИМ постоянной скорости без измерения положения

3.1.2.1. Варианты построения системы

В большинстве систем автоматического регулирование воздействие на объект управления осуществляется изменением подачи газа, жидкости или сыпучего материала с помощью регулирующей арматуры: задвижек, клапанов, заслонок (далее все виды арматуры будем называть *вентилями*). Арматура большей частью оснащается нерегулируемым электрическим приводом переменного тока. Система управления может включить привод на открытие или закрытие вентиля, увеличивая или уменьшая тем самым подачу среды, или выключить привод, зафиксировав подачу. Изменять скорость привода невозможно, однако факты полного закрытия и полного открытия вентиля, как правило, фиксируются, поскольку приводы оснащаются концевыми выключателями.

К чему в системе регулирования отнести вентиль с его приводом? К регулятору или к объекту управления? Как не странно, поступают и так и так. Все зависит от того, снабжен ли привод измерительным преобразователем положения. Если положение не измеряется, вентиль с приводом обычно относят к регулятору. В противном случае – к объекту.

Если положение вентиля не измеряется, тогда с точки зрения теории автоматического управления привод – это *интегратор*, *не охваченный обратной связью*. Правда, этот интегратор не совсем обычный: во-первых, он не способен изменять «скорость» интегрирования пропорционально амплитуде входного сигнала (привод – постоянной скорости), во-вторых, имеется ограничение на выходной сигнал (0 – 100%). Однако, так или иначе, если привод входит в состав регулятора, закон регулирования «автоматически» уже имеет И составляющую, но для того, чтобы она «правильно» работала, потребуется «постараться». Каким-то способом нужно заставить вентиль «плавно» регулировать скорость изменения подачи среды, и способ этот прост: если мы не можем из-

менять амплитуду сигнала на входе интегратора, почему нельзя изменять время подачи на него «полного» сигнала? Ведь для «интегратора» *в конечном итоге* неважно: интегрировать 10 с на «половинной» или 5 с на полной скорости (или даже пять раз по одной секунде на полной скорости). Интегратор – он и есть интегратор, накопитель, другим словом. Какая разница: рубль целиком или два раза по пятьдесят копеек? Таким образом мы приходим к идее импульсной модуляции – изменять длительность (широтно-импульсная модуляция, ШИМ) или частоту (частотно-импульсная, ЧИМ) включения привода вентиля в зависимости от того, какая скорость изменения подачи среды требуется в настоящий момент времени.

В большинстве случаев применяется ШИМ и изменяемым параметром модуляции служит *скважность* импульсов (рис. 3.43).

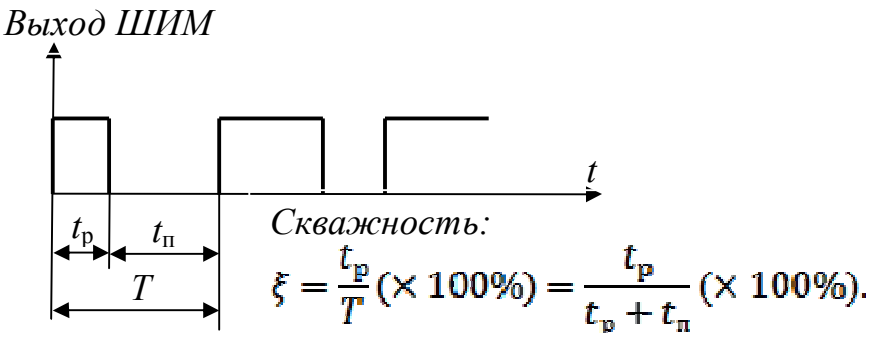


Рис. 3.43. Скважность импульсов.

Скважность – это отношение длительности импульса (включения привода) к периоду модуляции. Она измеряется в относительных единицах (от 0 до 1) или в процентах (от 0 до 100), причем 0% соответствует ситуация, когда привод выключен в течение всего периода модуляции, а 100% – постоянно включен. Мы будем измерять скважность в процентах. Если входной сигнал модулятора также измерять в процентах, он будет иметь смысл желаемой относительной скорости движения привода (от –100% до 100%). Тогда скважность должна быть равна модулю этого сигнала, а направление движения будет определяться его знаком.

Наличие вентиля с приводом, управляемым по принципу ШИМ, вносит свою специфику в реализацию законов регулирования. Предположим, требуется реализовать ПИД закон регулирования с коэффициентом регулятора $k_р$, постоянными интегрирования $T_и$ и дифференцирования $T_д$. Тогда структура «реального» регулятора с вентилем будет следующей (рис. 3.44):

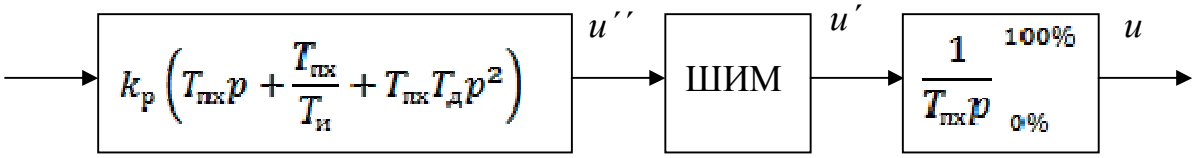


Рис. 3.44. Структура регулятора

Здесь исполнительный механизм (вентиль с приводом) описан передаточной функцией интегрирующего звена с постоянной времени $T_{пх}$, равной

времени полного хода механизма (от полностью закрытого состояния до полностью открытого). Объясняется это следующим образом. Единица измерения всех сигналов – проценты. Широтно-импульсный модулятор, входящий в состав контроллера, способен выдавать три значения сигнала: 0, 100 и –100%, соответствующие выключению механизма и включению его на открытие и закрытие. Предположим, в начальном состоянии вентиль закрыт ($u(0) = 0$) и дана команда на его открытие ($u'(t) = 100$). Выходной сигнал интегратора (положение вентиля) будет изменяться согласно выражению

$$u(t) = \frac{1}{T_{\text{пх}}}\int_0^t u'(\tau)\,d\tau = \frac{100}{T_{\text{пх}}}\int_0^t d\tau = \frac{100}{T_{\text{пх}}} - u(0) = \frac{100t}{T_{\text{пх}}}.$$

При $t = T_{\text{пх}}$ вентиль откроется полностью: $u(T_{\text{пх}}) = 100\%$. Отсюда видно, что постоянная интегрирования равна времени полного хода механизма.

Программа контроллера реализует т.н. ПДД^2 закон регулирования. Если пренебречь звеном ШИМ (о допустимости чего речь пойдет ниже), то передаточная функция «полного» регулятора будет равна произведению передаточной функции ПДД^2 регулятора и передаточной функции исполнительного механизма, т.е. мы получим то, что хотели – ПИД регулятор:

$$k_{\text{р}}\left(T_{\text{пх}}p + \frac{T_{\text{пх}}}{T_{\text{и}}} + T_{\text{пх}}T_{\text{д}}p^2\right) \times \frac{1}{T_{\text{пх}}p} = k_p\left(1 + \frac{1}{T_{\text{и}}p} + T_{\text{д}}p\right).$$

Рассмотренный подход имеет два принципиальных недостатка.

1. Трудность реализации ПДД^2-регулятора. Как мы уже знаем, операция дифференцирования физически нереализуема, а двойного дифференцирования – нереализуема «вдвойне». Поэтому речь может идти только о приближенных реализациях, в состав которых входят «балансные» фильтрующие звенья, см. п. 2.2.3.3, «защищающие» регулятор от высокочастотных помех, но одновременно искажающие его динамику.

2. Широтно-импульсный модулятор, наличием которого мы пренебрегли, вносит дополнительную «погрешность» в поведение регулятора. На рис. 3.45 показаны реакции «обыкновенного» интегратора и интегратора, управляемого ШИМ, на постоянный положительный входной сигнал (меньший 100%).

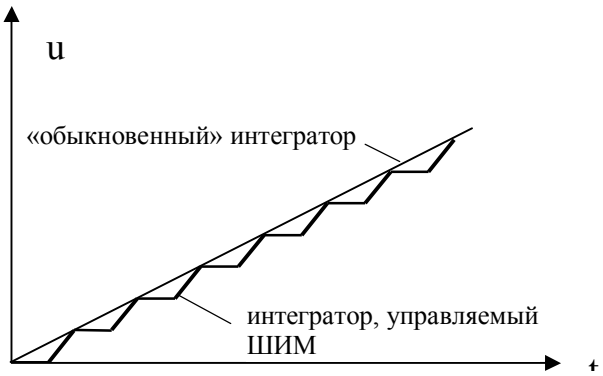


Рис. 3.45. Погрешность, вносимая ШИМ.

Как видно из рисунка, на интервалах модуляции реакция интегратора, управляемого ШИМ, отстает от эталонной, и это отставание будет тем больше, чем больше период модуляции. Теоретически улучшить поведение регулятора можно, увеличив частоту ШИМ, однако на практике мы сталкиваемся с ограничением на максимальную частоту включения привода вентиля. Слишком

частые включения могут быть вообще не восприняты приводом (двигатель просто не успеет запуститься), а если и будут восприняты, то приведут к перегреву двигателя пусковыми токами и выходу его из строя.

С этим же связана еще одна проблема. Даже при небольшой частоте модуляции возникают трудности с отработкой очень коротких и очень длинных импульсов. В первом случае привод не успевает включиться, а во втором – отключиться. Самое простое решение – вообще не включать привод, если рассчитанная длительность импульса меньше некоторой минимально возможной величины (и не отключать – при малом времени паузы). Но при этом может оказаться, что система не в состоянии обеспечить приемлемую точность регулирования в установившемся или квазиустановившемся режиме. Ситуация коротких импульсов – скорее правило, чем исключение: в реальности большую часть времени вентиль должен находиться в покое, иногда лишь корректируя свое положение для устранения небольших ошибок регулирования. Понятно, что точность регулирования будет определяться откликом объекта на минимально возможный ход механизма. Если объект обладает самовыравниванием, система может находиться в установившемся режиме, выполняя задание с соответствующей ошибкой. Объект без самовыравнивания в установившемся режиме может находиться только при условии полного баланса внешних воздействий, поэтому важно максимально точно «подобрать» управляющее воздействие для «нейтрализации» возмущений. Иначе неизбежны частые «челночные» перемещения регулирующего вентиля. Поэтому в таких системах желательно все-таки заданные перемещения реализовать точно.

Одно из решений состоит в «запоминании» и «переносе» импульсов. Пусть, например, минимальная длительность импульса, который может отработать привод, соответствует входному сигналу, равному 2%, а текущее значение сигнала – 1%. Тогда на текущем интервале мы вообще не будем включать привод, а на следующем – включим на время, равное 2% от периода модуляции. Аналогично для «больших» сигналов: если текущее значение 99%, то на данном периоде мы выключать привод совсем не будем (т.е. включим на 100%), зато на следующем – выключим на 2% (т.е. включим на 98%). Таким образом, «неисправимые» в настоящее время ошибки «исправляются» в будущем. Другой вопрос: а не поздно ли их будет исправлять?...

Другой подход состоит в изменении частоты ШИМ таким образом, чтобы подстроиться под возможности аппаратуры. При этом мы получаем некий гибрид ШИМ с ЧИМ. Алгоритм достаточно прост. Пусть минимальное время включения и выключения привода равно $t_{\text{мин}}$. Если текущее значение входного сигнала u'' менее 50%, то привод мы включим ровно на время $t_{\text{мин}}$, а время паузы рассчитаем таким образом, чтобы скважность была равна входному сигналу:

$$\xi = \frac{t_{\text{р}}}{T} \times 100 = \frac{t_{\text{р}}}{t_{\text{р}} + t_{\text{п}}} \times 100 = \frac{t_{\text{мин}}}{t_{\text{мин}} + t_{\text{п}}} \times 100 = u'',$$

$$t_{\text{п}} = t_{\text{мин}} \frac{(100 - u'')}{u''}.$$

Если текущее значение входного сигнала u'' более 50%, то привод будет *выключен* на время $t_{мин}$, а время работы рассчитано в соответствие со скважностью:

$$\xi = \frac{t_р}{T} \times 100 = \frac{t_р}{t_р + t_п} \times 100 = \frac{t_{мин}}{t_р + t_{мин}} \times 100 = u'',$$

$$t_р = t_{мин} \frac{u''}{(100 - u'')}.$$

При u''=50% можно применять и тот и другой подход: в любом случае получим $t_р = t_п = t_{мин}$.

Таким образом, мы вообще не теряем (и не переносим) импульсы и получаем при этом минимально возможный период модуляции при любой заданной скважности.

Однако на самом деле проблема остается нерешенной: в случаях, когда скважность либо очень мала, либо очень велика (по модулю), цикл модуляции «растягивается» из-за чрезмерной длительности паузы или включения привода. Для системы регулирования это равнозначно разрыву контура управления на достаточно большое время, что грозит существенным ухудшением качества процесса вплоть до потери устойчивости. Так, например, при малой ошибке алгоритм включит привод на минимально возможное время, а потом долго будет «выжидать», пока не истечет рассчитанное время паузы, совершенно не участвуя в процессе регулирования. И пока он «ждет», в системе может случиться все, что угодно…

Улучшить ситуацию можно, если отказаться от «выжидания» и постоянно пересчитывать время включения и время отключения в соответствии с текущим сигналом управления.

Наконец, существует подход к формированию импульсов, принципиально отличный от ранее рассмотренных. Не имея измерителя, мы, конечно, не можем определить положение привода в любой момент времени. Однако, зная фактическую скорость перемещения рабочего органа и, следовательно, постоянную времени интегратора, описывающего привод, мы можем рассчитать *приращение положения*, которое имело бы место, если бы на вход интегратора действительно подавался непрерывный сигнал. Далее – дело за малым: включать привод так, чтобы реализовать это приращение с приемлемой точностью и с ограничением на минимальное время включения. Получается своего рода следящая минисистема регулирования приращения релейного типа на основе модели.

Таким образом, выше рассмотрены идеи четырех возможных алгоритмов формирования импульсов управления исполнительным механизмом постоянной скорости:

1) «классическая» ШИМ с отбрасыванием коротких импульсов;

2) «классическая» ШИМ с переносом коротких импульсов;

3) ШИМ с переменным периодом модуляции;

4) «следящая» ШИМ.

3.1.2.2. Разработка алгоритмов и моделирование систем в Simulink

В Simulink-диаграммах широтно-импульсные модуляторы будем задавать маскированными подсистемами на основе интерпретируемых Matlab-функций.

Для первых двух вариантов ШИМ входными параметрами функции являются текущее время, формируемое стандартным блоком Clock, управляющий сигнал и два настроечных параметра: минимальное время импульса/паузы и период модуляции. Последние задаются непосредственно в «маске» (рис. 3.46).

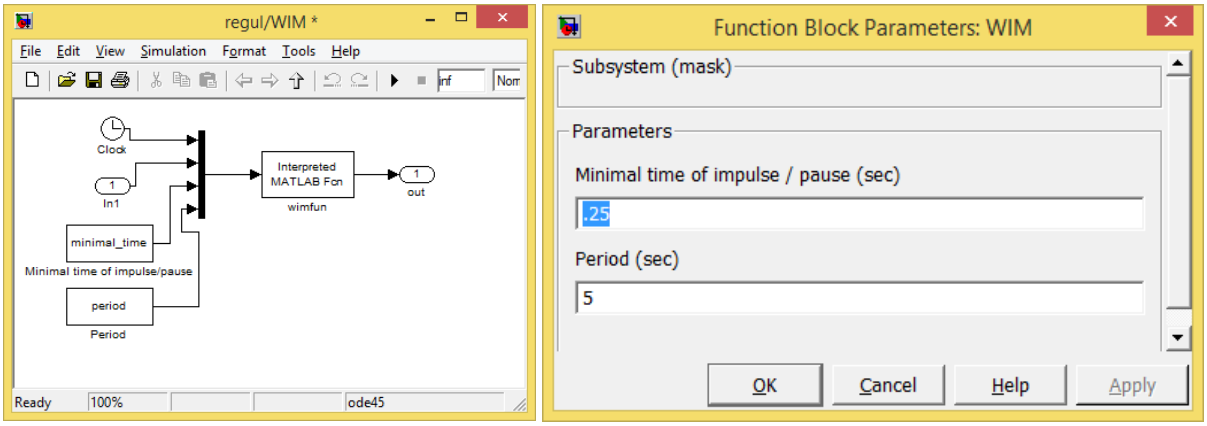


Рис. 3.46. Подсистема ШИМ (варианты 1, 2).

Код функции по варианту 1:

```
function output = winfun(input)
% Сохраняемые между вызовами переменные:
% время начала периода, длительность импульса
% и знак управления
persistent time_of_start time_on sign_u
time = input(1); % текущее время
u = input(2); % управление -100...100
minimal_time = input(3); % минимальное время импульса/паузы
period = input(4); %период ШИМ
% Инициализация
if (time==0)
    module_u = abs(u); % модуль управления
    sign_u = sign(u);
    time_of_start = 0;
    time_on = module_u/100*period;
    time_off = period - time_on; % время паузы
    if time_on < minimal_time
        time_on = 0;
    elseif time_off < minimal_time
        time_on = period;
    end
end
local_time = time - time_of_start; % время от начала периода
% Пересчет времени включения/паузы
if local_time >= period
    module_u = abs(u);
    sign_u = sign(u);
    time_of_start = time;
```

```
    local_time = 0;
    time_on = module_u/100*period;
    time_off = period - time_on;
    if time_on < minimal_time
        time_on = 0;
    elseif time_off < minimal_time
        time_on = period;
    end
end
if local_time < time_on
    output = 100*sign_u;
else
    output = 0;
end
```

Благодаря своей простоте и наличию комментариев код не требует пояснений. Для второго варианта функция будет несколько сложнее, однако разобравшись с первым вариантом, можно разобраться и со вторым:

```
function output = winfun(input)
% Сохраняемые между вызовами переменные:
% время начала периода, время включения (со знаком!),
% "перенесенное время" (со знаком!)
persistent time_of_start time_on additive
time = input(1); % текущее время
u = input(2); % управление -100...100
minimal_time = input(3); % минимальное время импульса/паузы
period = input(4); %период ШИМ
% Начальная инициализация
if (time==0)
    time_of_start = 0;
    additive = 0;
    time_on = u/100*period;
    time_off = period - abs(time_on);
    if abs(time_on) < minimal_time
        additive = time_on;
        time_on = 0;
    elseif time_off < minimal_time
        additive = -time_off*sign(u);
        time_on = period*sign(u);
    end
end
local_time = time - time_of_start; % время от начала периода
% Пересчет времени включения/паузы
if local_time >= period
    time_of_start = time;
    local_time = 0;
    time_on = u/100*period + additive;
    time_off = period - abs(time_on);
    if abs(time_on) < minimal_time
        additive = time_on;
        time_on = 0;
    elseif time_off < minimal_time
        additive = -time_off*sign(time_on);
```

```
            time_on = period*sign(time_on);
        else
            additive = 0;
        end
    end
    if local_time < abs(time_on)
        output = 100*sign(time_on);
    else
        output = 0;
    end
```

В отличие от первого варианта сохраняемая между вызовами переменная `time_on` имеет знак: в течение «положительного» времени привод открывает вентиль, в течение «отрицательного» – закрывает. Переменная `additive` накапливает «перенесенное время» и также имеет знак.

Состав и настройка подсистемы ШИМ с переменным периодом модуляции показаны на рис. 3.47.

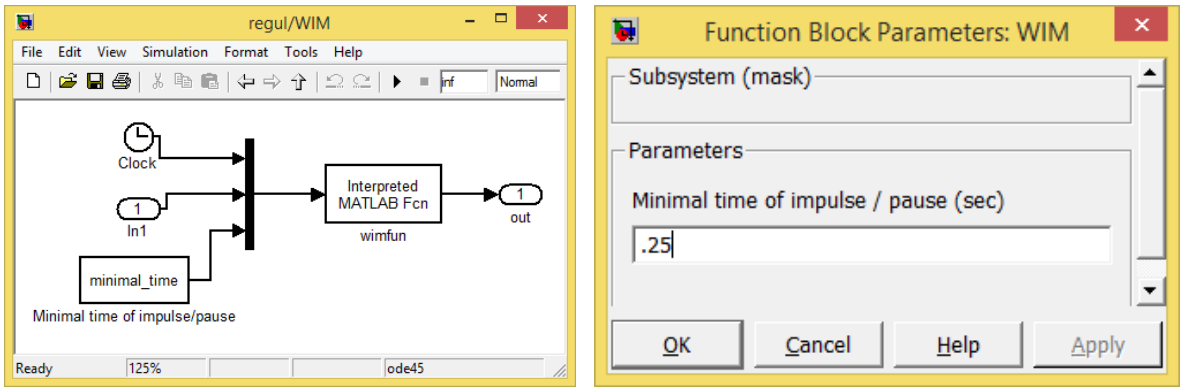


Рис. 3.47. Подсистема ШИМ (вариант 3).

В данном случае единственным параметром настройки является минимальное время импульса/паузы. Код функции-формирователя импульсов приведен ниже.

```
function output = winfun(input)
% Сохраняемые между вызовами переменные: состояние (-100,0,100)
% и время его начала
persistent state time_of_start
time = input(1); % текущее время
u = input(2); % управление -100...100
module_u = abs(u); % модуль управления
minimal_time = input(3); % минимальное время импульса/паузы
% Начальная инициализация: пауза
if (time == 0)
    state = 0;
    time_of_start = 0; %время текущего состояния
end
% время текущего состояния
time_of_state = time - time_of_start;

if (state == 100 || state == -100) % импульс на выходе есть
 if ((module_u < 100)&&(time_of_state > minimal_time))...
```

```
    % импульс может быть снят
    && ((module_u < 50)||(time_of_state>module_u/(100 - module_u)*
       minimal_time))% и должен быть снят
       time_of_start = time;
       state = 0;
 end
else % импульса на выходе нет
 if ((module_u > 0) && (time_of_state > minimal_time))...
     % импульс может быть подан
     && ((module_u > 50)|| (time_of_state>(100 - module_u)/
       module_u*minimal_time)) % и должен быть подан
       time_of_start = time;
       state = 100*sign(u);
 end
end
output = state;
```

«Центральной» переменной является переменная `state` (состояние), которая может принимать три значения:

100 – привод включен на открытие вентиля;

-100 – привод включен на закрытие вентиля;

0 – привод выключен.

В состояниях 100 и -100 выключение *возможно*, если модуль управляющего сигнала меньше 100 и время состояния превышает минимальное время импульса. Выключение *необходимо*, если модуль управляющего сигнала меньше 50 или время включения превышает значение, вычисленное по соответствующей формуле.

В состоянии 0 включение *возможно*, если модуль управляющего сигнала больше 0 и время состояния превышает минимальное время паузы. Включение *необходимо*, если модуль управляющего сигнала больше 50 или время паузы превышает значение, вычисленное по соответствующей формуле.

Отметим, что функция при каждом своем пересчете переопределяет возможность включения/отключения привода в зависимости от текущего значения входного управляющего сигнала. Таким образом, решение принимается не «заранее», на весь будущий период, а «в оперативном порядке». Это, с одной стороны, не позволяет говорить о «математической» обоснованности алгоритма, но с другой – придает ему адаптивные свойства.

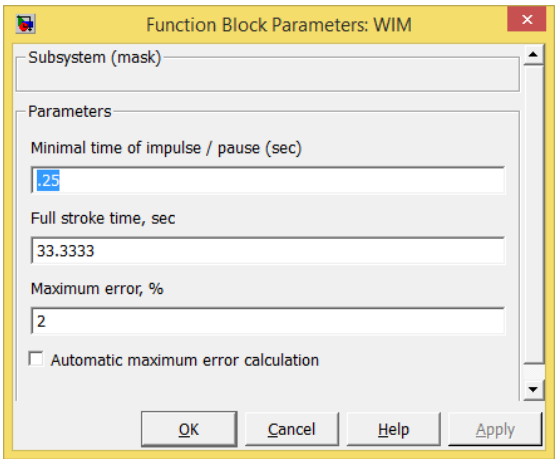


Рис. 3.48. Параметры формирователя импульсов (вариант 4).

Последняя, четвертая, реализация формирователя импульсов, как уже было сказано выше, весьма специфична. Настраиваемыми параметрами подсистемы являются: минимальное время импульса/паузы, время полного хода механизма, максимальная ошибка слежения и возможность ее автоматического вычисления (рис. 3.48).

Внутреннее содержание подсистемы показано на рис. 3.49.

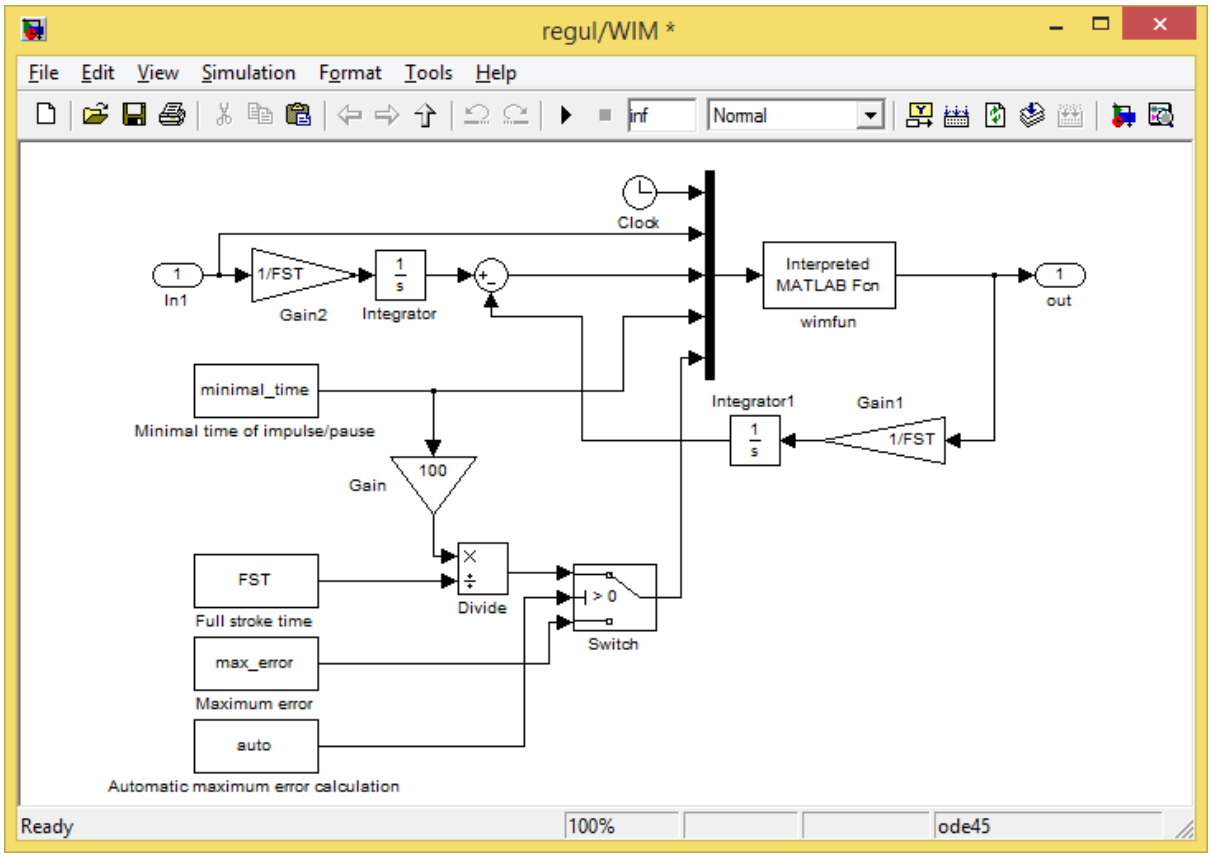


Рис. 3.49. Подсистема ШИМ (вариант 4).

Основная задача функции-формирователя импульсов состоит в поддержании ошибки слежения на допустимом уровне. Ошибка слежения формируется как разность между откликом «идеального привода» (который может реагировать на непрерывный сигнал) и откликом «реального привода» (который управляется сигналами 0, ±100%). Допустимый уровень ошибки задается параметром «Maximum error» (в % хода) или автоматически вычисляется через минимальное время включения/паузы и время полного хода привода (Full stroke time).

Код функции:

```
function output = wimfun(input)
% Сохраняемые между вызовами переменные: состояние
% и время его начала
persistent state time_of_start
time = input(1); % текущее время
u = input(2);
e = input(3); % ошибка накопления
minimal_time = input(4); % минимальное время импульса/паузы
max_error = input(5);
% Начальная инициализация: пауза
if (time == 0)
    state = 0;
    time_of_start = 0;
end
time_of_state = time - time_of_start;
```

```
if time_of_state > minimal_time
    if state == 100
        if e < 0
            time_of_start = time;
            state = 0;
        end
    elseif state == -100
        if e > 0
            time_of_start = time;
            state = 0;
        end
    else
        if (e > max_error) && (u > 0)
            time_of_start = time;
            state = 100;
        elseif (e < -max_error) && (u < 0)
            time_of_start = time;
            state = -100;
        end
    end
end
output = state;
```

Выключение привода происходит при «полной ликвидации» ошибки слежения (т.е. тогда, когда ее знак изменится на противоположный), включение – в момент, когда модуль ошибки превысит максимально допустимое значение.

Для проверки алгоритмов и программ формирования импульсов разработана Simulink-модель (рис. 3.50), рассчитывающая реакции «реального» и «идеального» приводов на затухающий синусоидальный сигнал вида

$$u''(t) = 100e^{-0,25t}\sin\left(\frac{2\pi}{100}t + \frac{\pi}{2}\right).$$

Сигнал по форме приближается к реально действующему в замкнутой системе при отработке ступенчато изменяющего воздействия. Время полного хода механизма принято равным 33,(3) с.

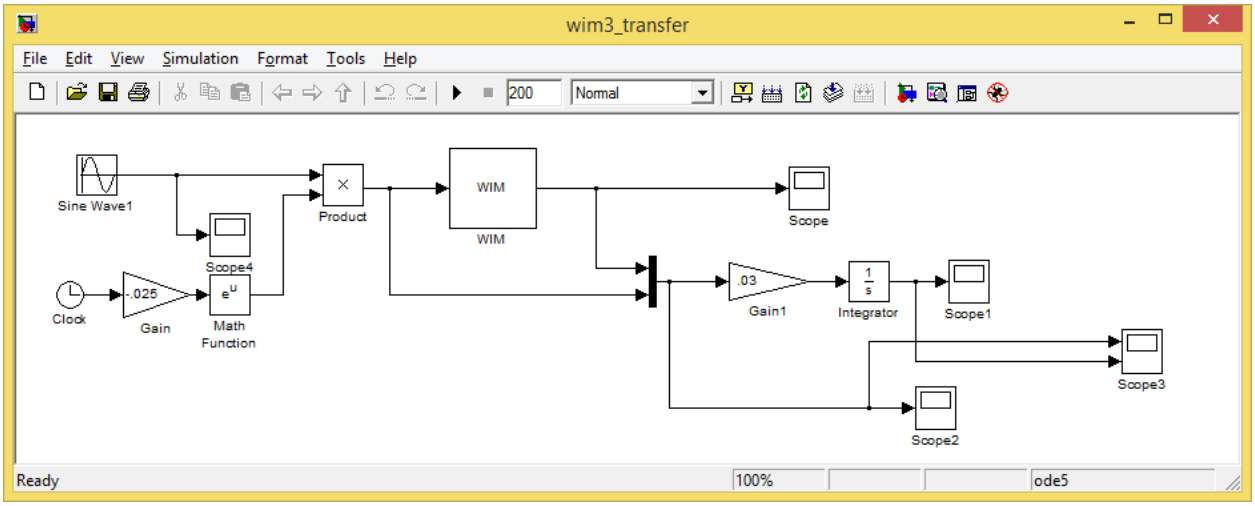


Рис. 3.50. Simulink-модель для проверки алгоритмов ШИМ.

Результаты моделирования сохранялись в блоке Scope3 в переменной SD (структура со временем) (рис. 3.51).

При помощи кода, приведенного ниже, были построены графики формирования импульсов для всех четырех вариантов (рис. 3.52 – 3.56).

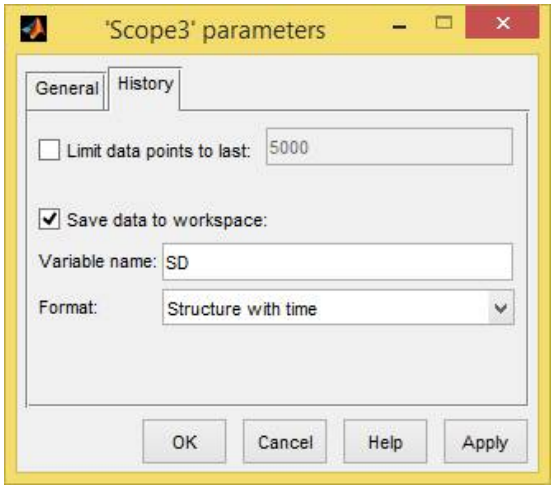


Рис. 3.51. Сохранение результатов моделирования.

```
subplot(2,1,1)
L = plot(SD.time, SD.signals(1).
values(:,2),…
         SD.time,
SD.signals(1).values(:,1)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(2,1,2)
L =plot(SD.time, SD.signals(2).values(:,2),…
        SD.time, SD.signals(2).values(:,1)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
```

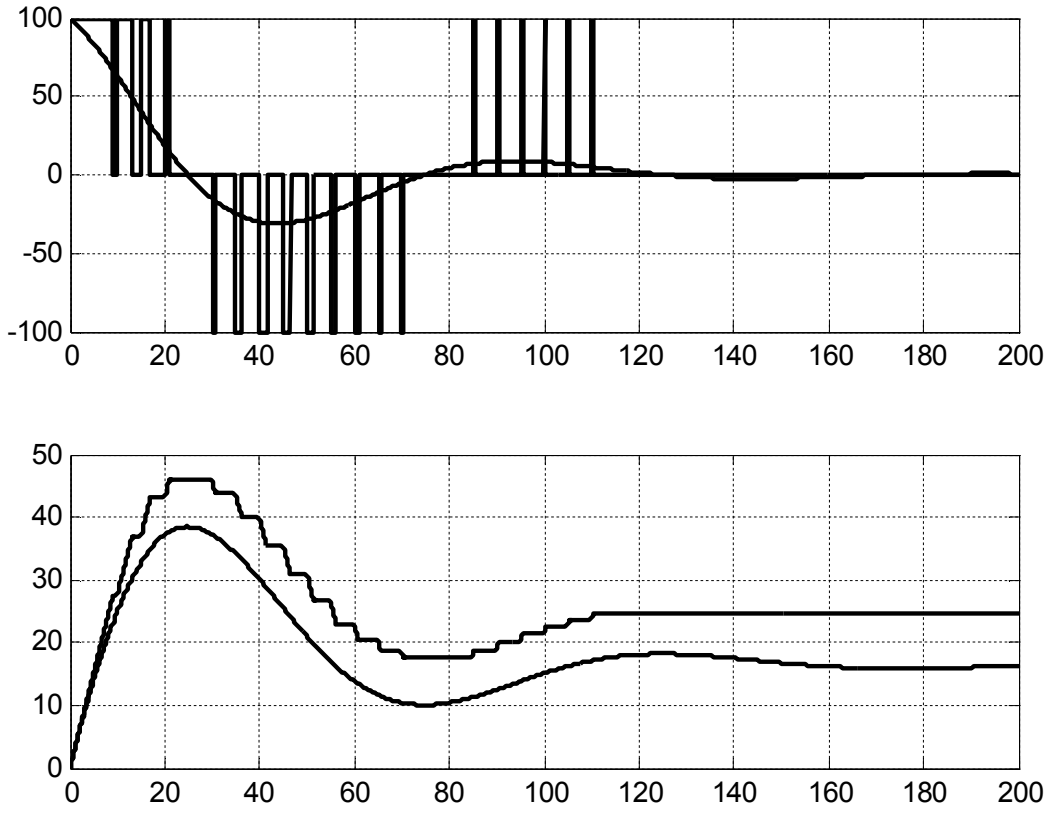


Рис. 3.52. Формирование импульсов и движение привода для варианта 1.

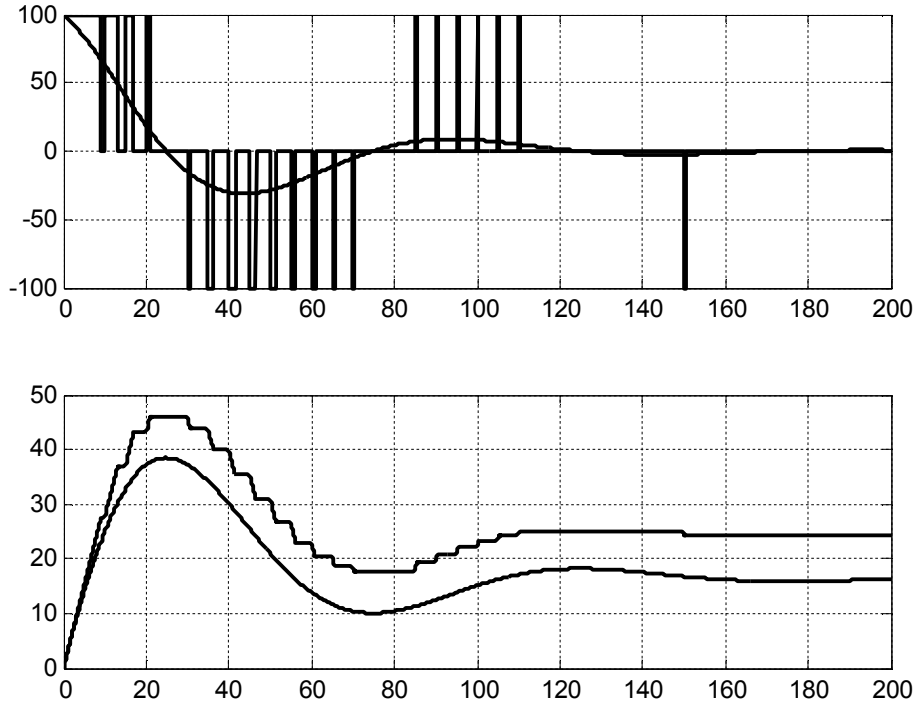


Рис. 3.53. Формирование импульсов и движение привода для варианта 2

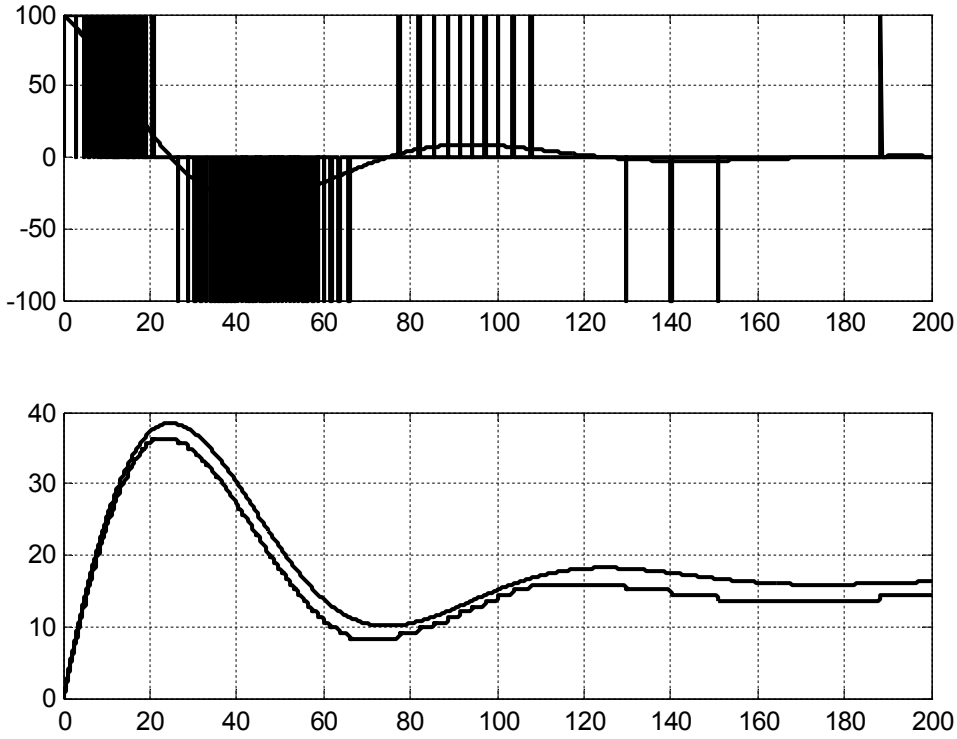


Рис. 3.54. Формирование импульсов и движение привода для варианта 3.

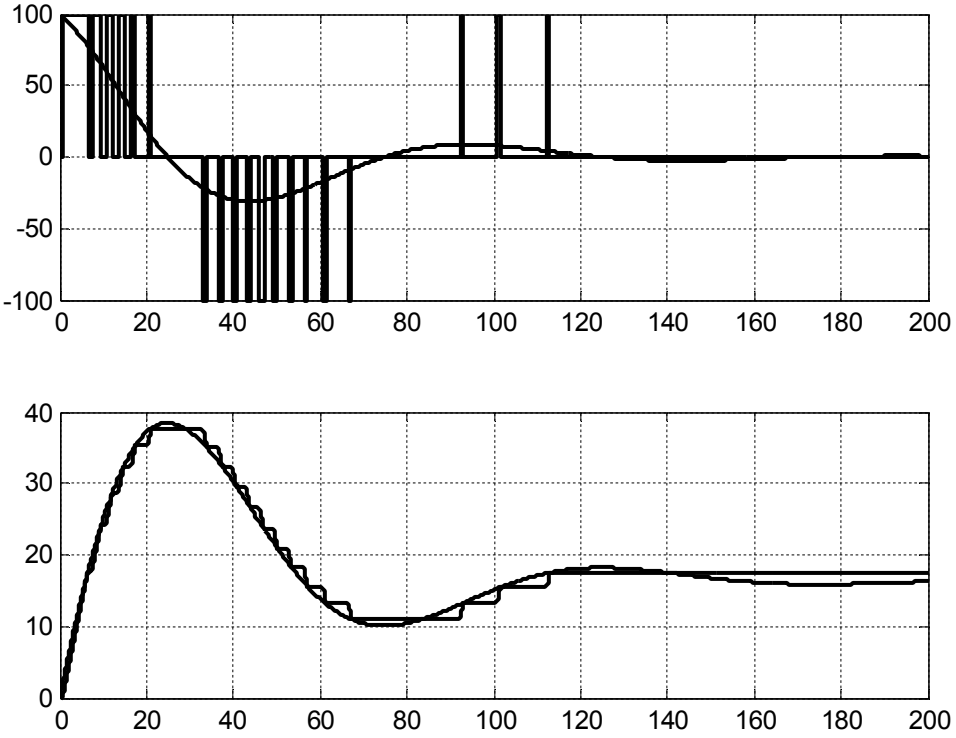


Рис. 3.55. Формирование импульсов и движение привода для варианта 4, максимальная ошибка слежения 2%.

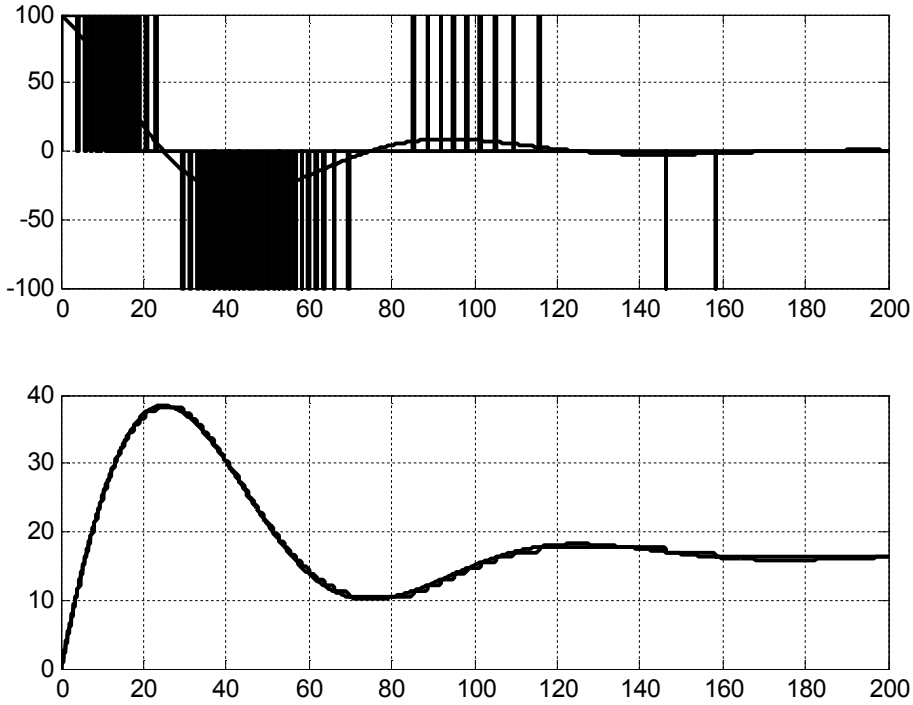


Рис. 3.56. Формирование импульсов и движение привода для варианта 4, максимальная ошибка соответствует минимальному времени включения.

Для всех вариантов минимальное время импульса/паузы было задано равным 0,25 с. Период следования импульсов для вариантов 1 и 2 установлен равным 5 с.

ШИМ с отбрасыванием коротких импульсов продемонстрировала отработку сигнала небольшим числом импульсов, однако ошибка конечного положения привода оказалась достаточно большой (около 10%), что, очевидно, объясняется «несимметричностью» входного сигнала в сочетании с «запаздыванием» реакции на него. Большая ошибка воспроизведения траектории совершенно не означает, что алгоритм неработоспособен: в замкнутой системе регулирования эти «огрехи» будут устранены регулятором в процессе сведения к минимуму ошибки регулирования. Однако понятно, что указанный недостаток должен ухудшить общее поведение системы.

ШИМ с переносом коротких импульсов показала практически аналогичное поведение. Единственный «корректирующий» (накопленный и перенесенный) импульс был подан приблизительно на 150 секунде процесса и кардинально изменить ситуацию не смог.

ШИМ с переменным периодом модуляции за счет «рационального использования времени» показала намного лучшие результаты в плане точности воспроизведения траектории, однако при этом частота включения привода оказалась намного выше, чем в первых двух вариантах.

«Следящая» ШИМ с допустимой ошибкой слежения 2% хода механизма продемонстрировала относительно хорошие результаты, как по точности воспроизведения траектории, так и по частоте включения привода. Уменьшение допустимой ошибки до величины, соответствующей минимальному времени включения привода (0,25 с или 0,75% хода) позволяет получить практически идеальное воспроизведение траектории, но одновременно значительно увеличивает частоту коммутации.

Следует отметить, что все алгоритмы, за исключением первого, продолжают формировать импульсы до тех пор, когда входной сигнал не станет равным нулю. Однако выйти на нуль в замкнутой системе регулирования им не удастся, так как длительность импульса не может быть меньше минимально допустимой величины и, следовательно, невозможно точно вывести вентиль в положение, соответствующее нулевой ошибке. Поэтому в квазиустановившемся режиме неизбежно «рысканье» механизма относительно этого положения в результате коротких включений привода на минимально возможное время попеременно в сторону «больше» и «меньше». Избавиться от «рысканья» можно, изменив алгоритмы формирования импульсов путем введения в них сигнала ошибки регулирования. Но таким образом мы жестко привязываем задачу ШИМ к задаче регулирования, что нежелательно, так как хотелось бы сохранить универсальность алгоритмов. Вместо этого можно применить более простое решение в виде зоны нечувствительности на входе ШИМ.

Для определения эффективности разработанных ранее алгоритмов формирования импульсов в «условиях, приближающихся к реальным», составлена Simulink-модель замкнутой системы регулирования для объекта из п. 3.1.1 (рис. 3.57).

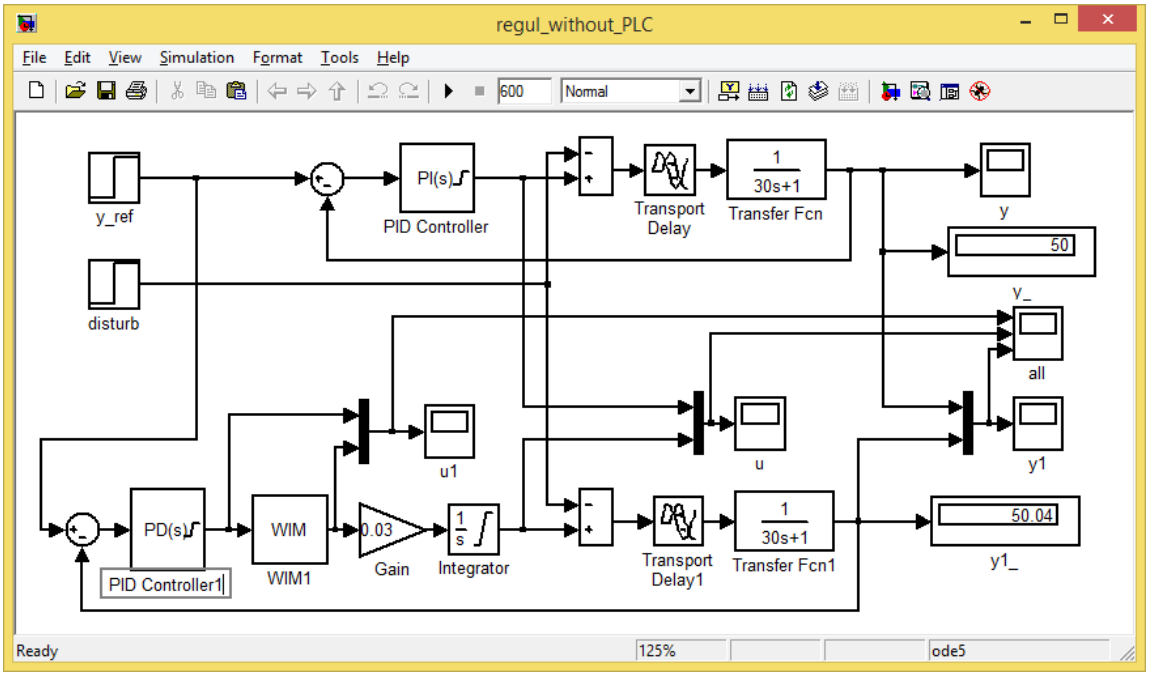


Рис. 3.57. Модели замкнутой системы автоматического регулирования.

Точнее говоря, Simulink-диаграмма содержит две модели: «эталонная» модель с ПИ регулятором, исследованная в п. 3.1.1, и модель системы с исполнительным механизмом постоянной скорости, в состав которой входят формирователь импульсов (блок WIM) и ПД регулятор.

Настройки ПД регулятора были определены исходя из того, чтобы им совместно с исполнительным механизмом был сформирован ПИ закон регулирования:

$$k_p'\left(1+T_д p\right)\times\frac{1}{T_{пх}p}=\frac{k'T_д}{T_{пх}}\left(1+\frac{1}{T_д p}\right)=k_p\left(1+\frac{1}{T_и p}\right),$$

откуда

$$T_д=T_и=\frac{1}{0,032}=31,25;\ k_p'=\frac{k_p T_{пх}}{T_д}=\frac{1,62\cdot 33,33}{31,25}=1,73.$$

Окно настройки блока PD показано на рис. 3.58. Коэффициент фильтра оставлен равным по умолчанию.

На обе системы одновременно подавались следующие воздействия:

1) изменение задания от 0 до 50% в момент времени 0с;

2) изменение возмущения от 0 до 20% в момент времени 300с.

Результаты моделирования сохранялись в блоке Scope «All» в переменной `SD` типа «Структура со временем». С помощью кода, приведенного ниже, были построены графики, показанные на рис. 3.59 – 3.63.

```
subplot(3,1,1)
L = plot(SD.time, SD.signals(1).values(:,2),…
         SD.time, SD.signals(1).values(:,1)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
```

```
subplot(3,1,2)
L =plot(SD.time, SD.signals(2).values(:,2),…
        SD.time, SD.signals(2).values(:,1)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(3,1,3)
L =plot(SD.time, SD.signals(3).values(:,2),…
        SD.time, SD.signals(3).values(:,1)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
```

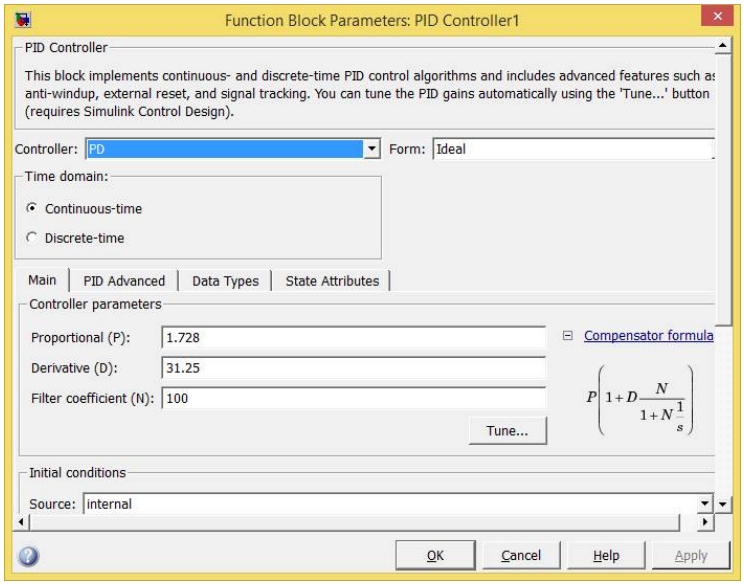


Рис. 3.58. Окно настройки блока PD.

На трех осях каждого рисунка показаны графики изменения:

1) выходных сигналов ПД регулятора и формирователя импульсов;

2) сигналов управления, подаваемых на объект в эталонной системе с ПИ регулятором и в исследуемой системе с ПД регулятором и ШИМ. В первом случае управляющий сигнал формируется непосредственно регулятором, во втором – регулирующим вентилем совместно с его исполнительным механизмом;

3) регулируемых величин эталонной и исследуемой систем.

Рис. 3.59 – 3.63 показывают, что система с исполнительным механизмом постоянной скорости принципиально неспособна отрабатывать ступенчатое изменение задающего сигнала так, как это делает эталонная система. Это и следовало ожидать: наша система не может мгновенно изменять значение управляющего сигнала вслед за изменением сигнала ошибки, как это происходит в эталонной системе, ведь приводу требуется определенное время на перемещение вентиля. В то же время отработка ступенчатого изменения возмущения в обеих системах происходит в основном одинаково. Это объясняется тем, что в данном процессе ошибка регулирования изменяется плавно и привод успевает сформировать управляющее воздействие, близкое к эталонному.

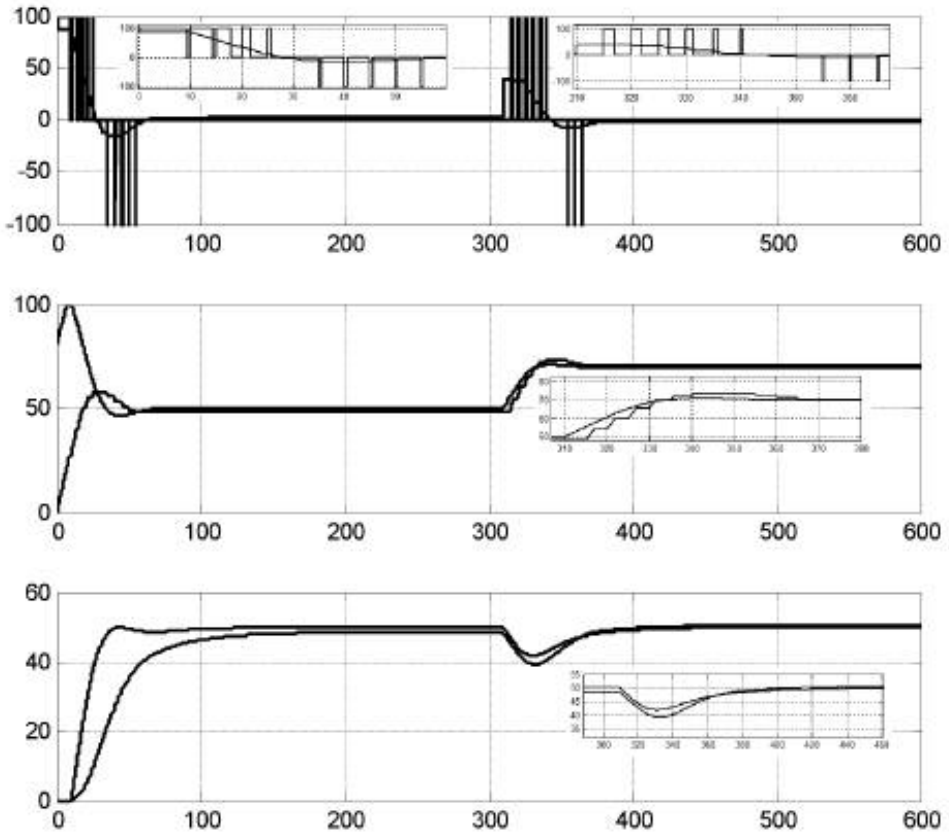


Рис. 3.59. Результаты моделирования (Scope «All») для случая «классической ШИМ» с отбрасыванием коротких импульсов.

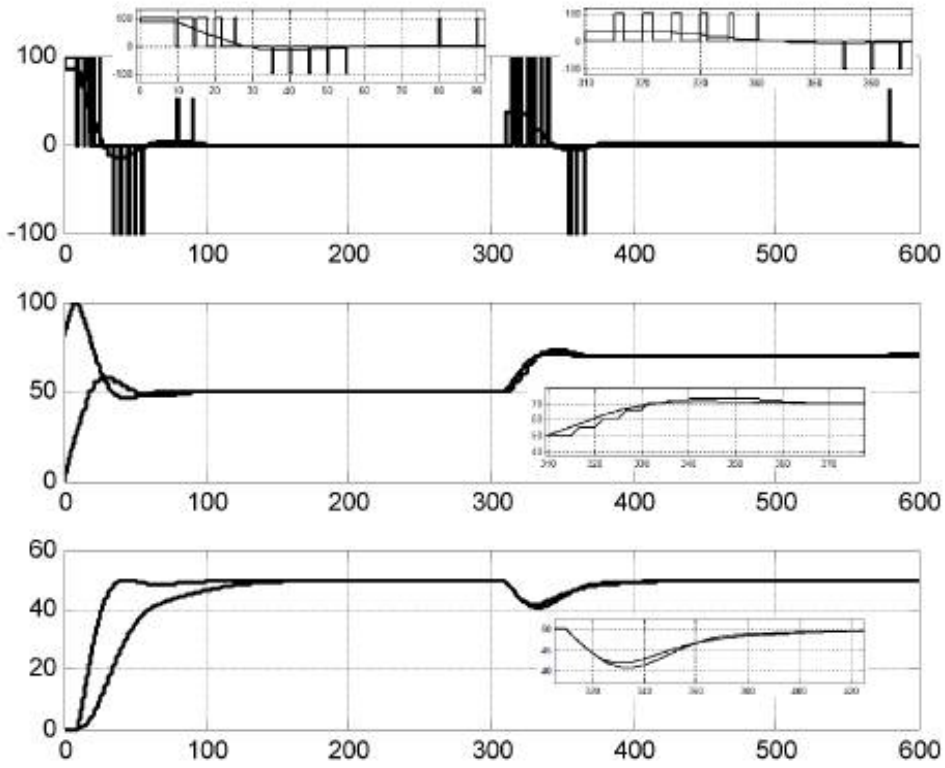


Рис. 3.60. Результаты моделирования (Scope «All») для случая «классической ШИМ» с переносом коротких импульсов.

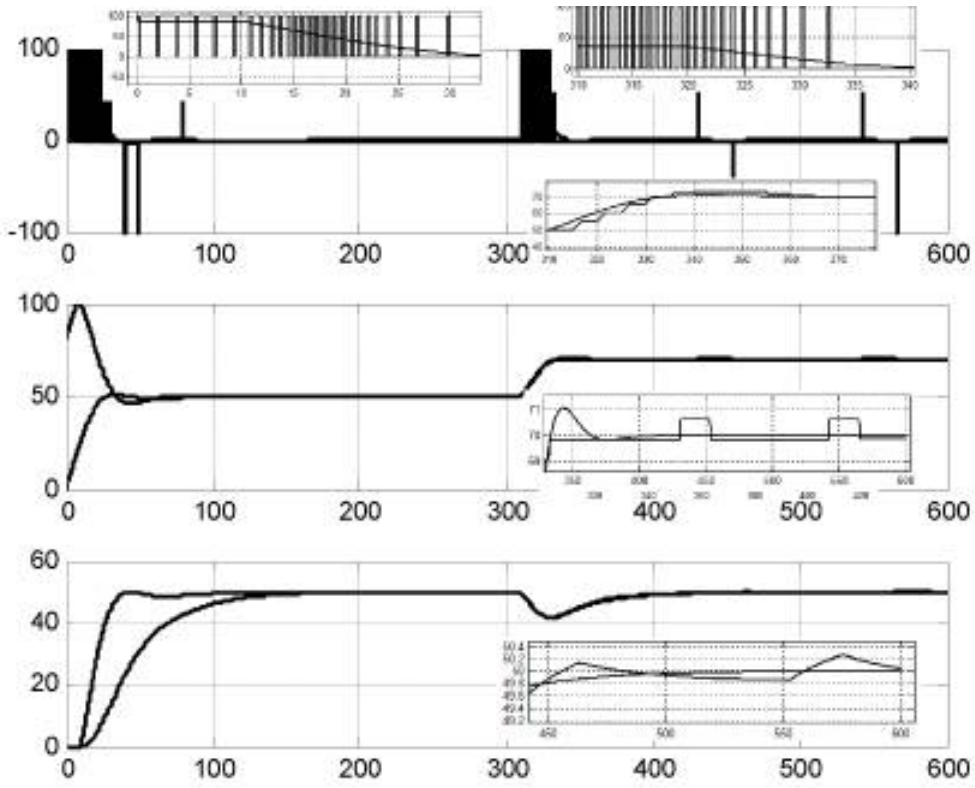


Рис. 3.61. Результаты моделирования (Scope «All»)
для случая ШИМ с переменным периодом.

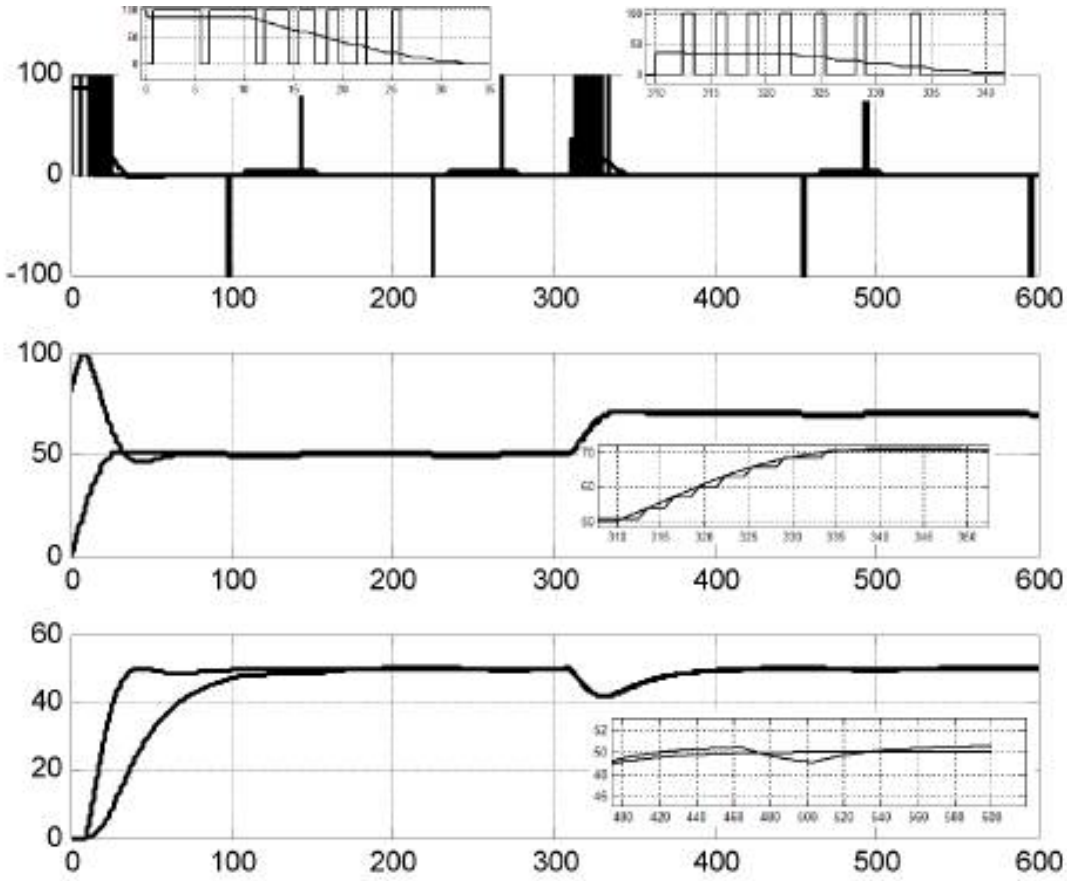


Рис. 3.62. Результаты моделирования (Scope «All») для случая
следящего ШИМ, ошибка слежения 2%..

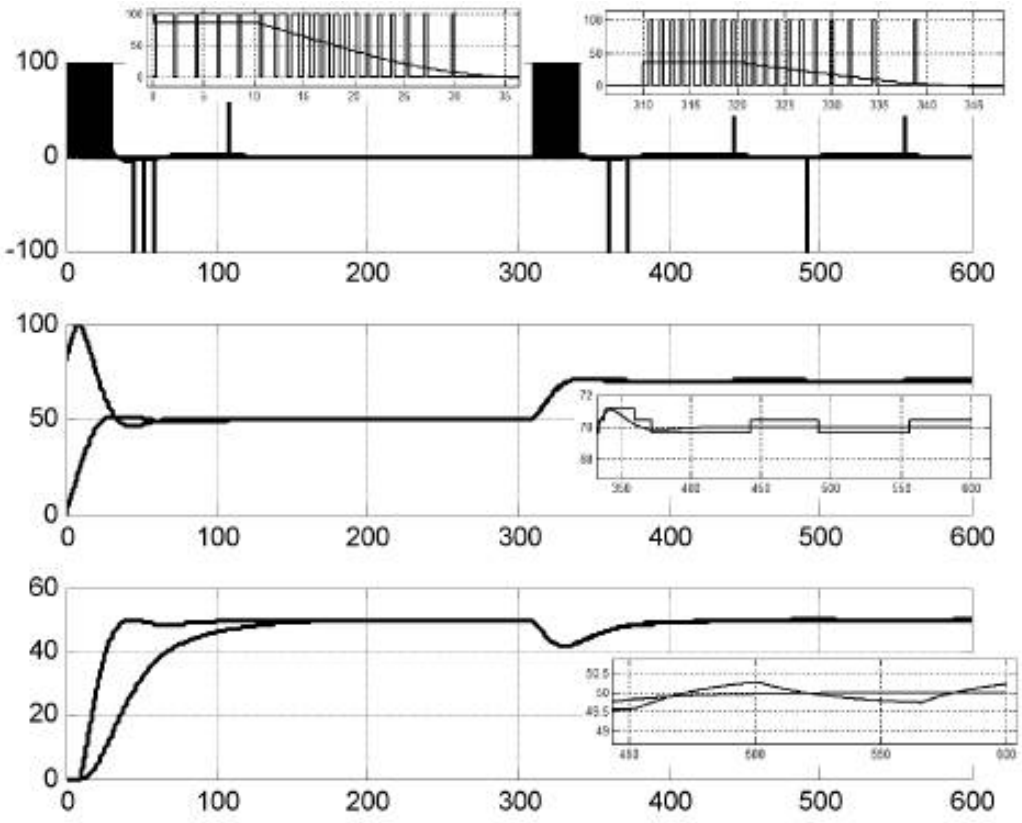


Рис. 3.63. Результаты моделирования (Scope «All») для случая следящего ШИМ, минимальная ошибка слежения.

Сравнивая графики на рис. 3.60 и 3.53, можно сделать вывод, что в замкнутой системе перенос импульсов оказался значительно эффективнее, чем в «разомкнутой». Вероятно, причиной этому является более точное воспроизведение выходного сигнала ПД регулятора формирователем импульсов.

Система с формирователем импульсов с переменным периодом в плане формирования управления и регулирования выходной величины отработала достаточно качественно, однако «расплатой» за это качество является большая частота включения привода и «рыскание» в квазиустановившемся режиме.

Применение следящего алгоритма позволяет при соответствующем выборе допустимой ошибки слежения существенно снизить частоту коммутации привода в переходных режимах по сравнению с предыдущим алгоритмом. Однако уменьшается точность воспроизведения траектории, что компенсируется регулятором замкнутого контура путем формирования корректирующих импульсов. Уменьшение допустимой ошибки до минимального уровня (рис. 3.63), как и было отмечено ранее, приводит к увеличению частоты коммутации привода.

3.1.2.3. Реализация алгоритмов управления на ПЛК

На рис. 3.64 показана Simulink-диаграмма для апробации алгоритмов. В верхней части диаграммы размещены блоки модели объекта регулирования с исполнительным механизмом, в нижнем – модель замкнутой системы регулирования, разработанная и опробованная ранее (эталонная система).

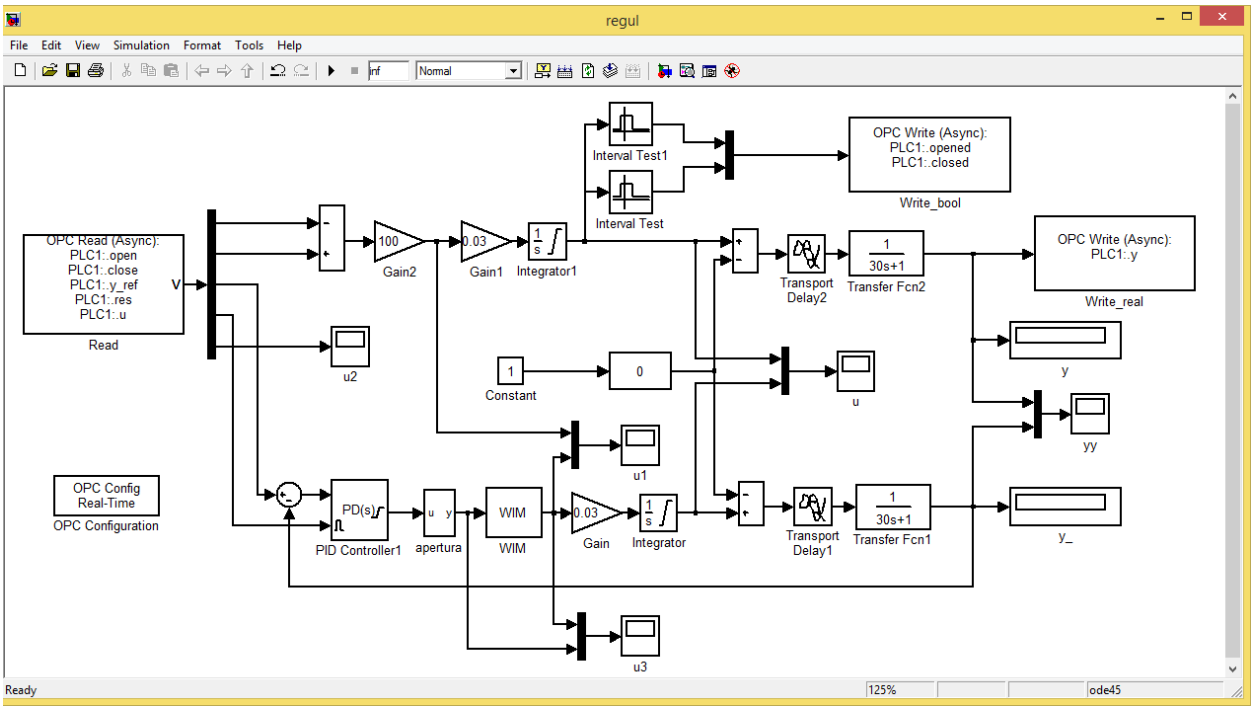


Рис. 3.64. Simulink-модель для апробации алгоритмов управления, реализуемых на ПЛК.

Блок OPC Read принимает от ПЛК следующие сигналы:

включить привод на открытие вентиля «open» (BOOL);

включить привод на закрытие вентиля «close» (BOOL);

задание регулятору «y_ref» (REAL);

сброс регулятора «res» (BOOL);

входной сигнал формирователя импульсов «u» (REAL, в диаграмме не обрабатывается, но отображается).

Сигналы «open» и «close», как и другие OPC-переменные типа BOOL, в Simulink-диаграмме «воспринимаются» следующим образом: TRUE = -1 (все биты слова установлены), FALSE = 0 (все биты сброшены). Для преобразования этих значений в значения +100, 0, -100, которые необходимы для управления моделью привода, задействованы сумматор/«вычитатель» и блок масштабирования.

Блоки Saturation Test формируют сигналы «концевых выключателей привода», которые посредством блока OPC Write «Write_bool» отправляются в ПЛК. Другой блок OPC Write «Write_real» служит для передачи ПЛК значения регулируемой величины.

Блок apertura не пропускает сигнал, сформированный ПД регулятором, на вход формирователя импульсов, если модуль этого сигнала меньше заданной пороговой величины. Статическая характеристика блока приведена на рис. 3.65.

Блок предназначен для исключения «рыскания» исполнительного механизма в квазиустановившемся режиме. Для ШИМ с отбрасыванием коротких импульсов применение блока необязательно. Блок построен на основе интерпретируемой функции Matlab (рис. 3.66), код которой настолько сложен, что комментировать его мы не будем.

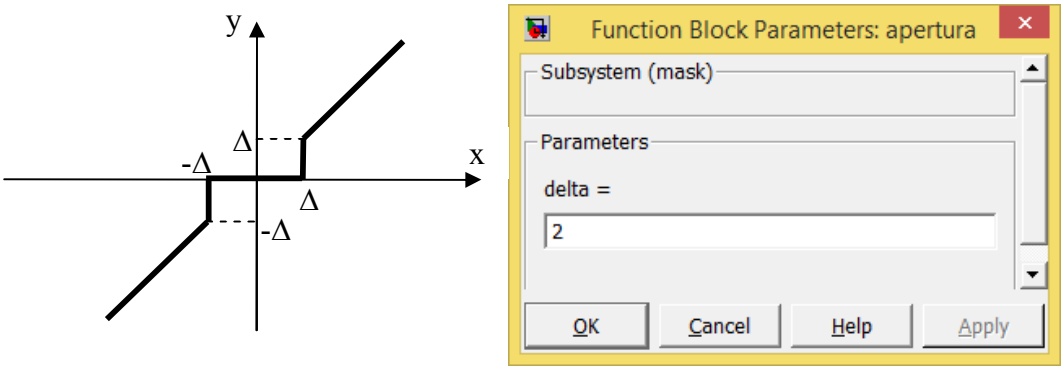
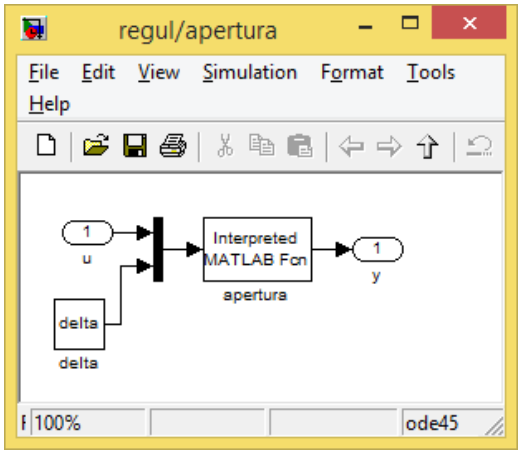


Рис. 3.65. Статическая характеристика окно настройки блока apertura.




```
function y = apertura(u)

    if abs(u(1)) < u(2)
        y = 0;
    else
        y = u(1);
    end
```

Рис. 3.66. Внутреннее содержимое блока apertura.

Программы ПЛК для всех четырех вариантов формирователя импульсов получены путем модификации программы, разработанной в п. 3.1.1.3. Изменением подвергся состав POU, теперь в него будут входить:

программа `REGULATION`, реализующая ПД закон регулирования;

функция `APERTURA` (в программе для классической ШИМ с отбрасыванием коротких импульсов – отсутствует);

функциональный блок `WIM`, формирующий импульсы управления исполнительным механизмом. Код этого блока различен для четырех вариантов формирователя;

программа `MODULATION`, вызывающая функцию `APERTURA` и функциональный блок `WIM`;

программа обслуживания операторской панели `PANEL`;

главная программа `PLC_PRG`.

Программы `CONTROL` в новой реализации нет.

Необходимость в разделении задач регулирования и формирования импульсов появилась по итогам экспериментов со стандартным функциональным блоком `PD` из Util.lib. Выяснилось, что в составе замкнутой системы регулирования блок удовлетворительно работает только тогда, когда он вызывается не «свободно» (по мере возможности), а «по расписанию» (т.е. через равные промежутки времени). Видимо, это связано со спецификой реализации операции дифференцирования.

Программы для первых трех вариантов формирователя импульсов предусматривают выполнение двух задач (рис. 3.67):

1) «General» (общая) со свободным вызовом программы `PLC_PRG`;

2) «REG» с вызовом программы `REGULATION` каждые 200 мс.

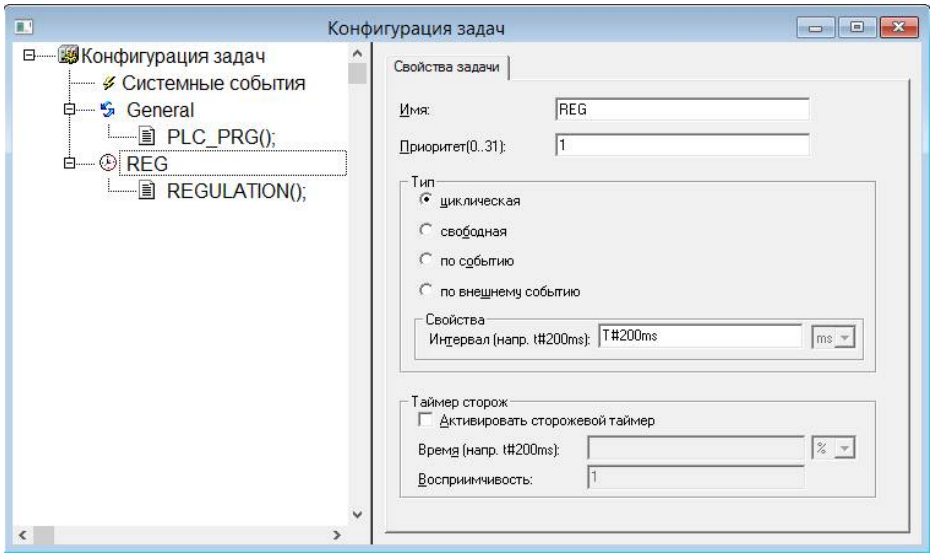


Рис. 3.67. Конфигурация задач ПЛК программ с формирователями импульсов № 1, 2, 3.

Программа `MODULATION` вызывается из `PLC_PRG`.

В программе, реализующей формирователь следящего типа, `MODULATION` вызывается в отдельной задаче «MODUL», запускаемой на исполнение с интервалом 25 мс (рис. 3.68). Это связано с наличием в данном алгоритме формирователя импульсов операций интегрирования.

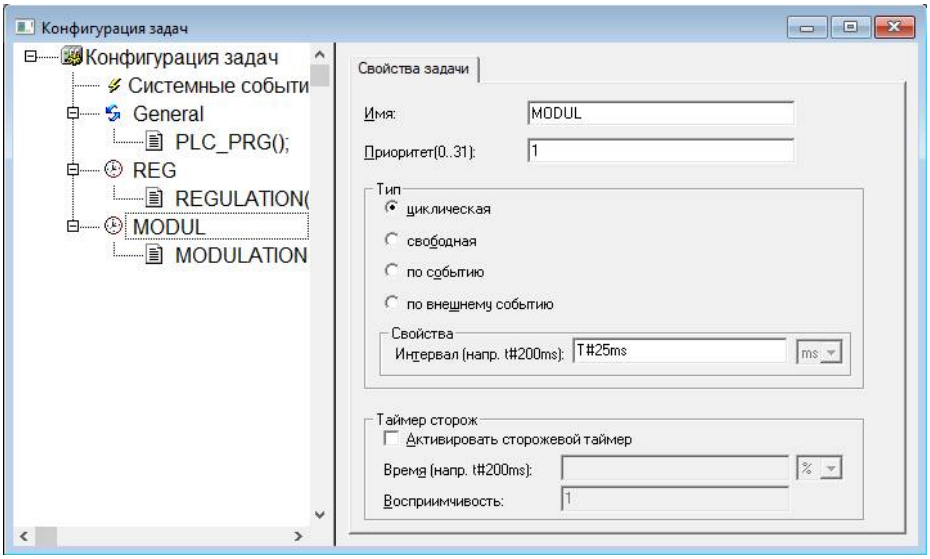


Рис. 3.68. Конфигурация задач ПЛК программ с формирователями импульсов № 4.

Список глобальных переменных (значительная часть которого участвует в «обмене» с «панелью оператора» и «SCADA-системой») подвергся незначительным изменениям, связанным только со способом формирования управляющего воздействия. Если ранее ПЛК (в автоматическом режиме) и операто-

ры панели и SCADA (в ручном и дистанционном режимах соответственно) формировали управляющее воздействие в непрерывном виде, то теперь – в виде команд «включить привод на открытие/закрытие вентиля». Изменился, конечно, и формат «ответной» информации.

Ниже приведен полный список глобальных переменных проекта, причем внесенные изменения выделены жирным шрифтом. Следует обратить внимание на то, что некоторые изменения связаны не с объявлением новых переменных, а с отказом от использования старых, объявления которых просто закомментированы.

```
VAR_GLOBAL

(*Входы контроллера*)
     y:REAL; (*регулируемая величина*)
     opened, closed:BOOL; (*сигналы конечных выключателей*)
     reset_process:BOOL:=FALSE; (*сброс системы после аварии –
                                  аппаратная кнопка без фиксации*)
(*Выходы контроллера*)
     (*u:REAL; (*Управление*)*)
     open:BOOL:=FALSE; (*сигнал управления "Включить привод на
                         открытие вентиля"*)
     close:BOOL:=FALSE; (*сигнал управления "Включить привод на
                          закрытие вентиля "*)
     no_HMI:BOOL:=FALSE; (*потеряна связь с панелью и SCADA –
                           сигнализация*)
     stop:BOOL:=FALSE; (*полная остановка системы*)

(*"Внутренние" переменные*)
     y_ref:REAL; (*задание регулятора, также  подается в Simulink-
                   модель*)
     u_manual:REAL:=0; (*сигнал ручного управления на
                         ПД-регулятор*)
     man:BOOL:=FALSE; (*сигнал перевода в режим ручного
                        управления*)
     res:BOOL:=FALSE; (*сброс регулятора, также  подается в
                        Simulink-модель*)
     block_panel:BOOL:=FALSE; (*блокирование панели*)
     u:REAL; (*Выход ПД-регулятора *)
     (*ТОЛЬКО ДЛЯ ВАРИАНТА С ФОРМИРОВАТЕЛЕМ ИМПУЛЬСОВ №4*)
          modulation_open: BOOL := FALSE;
          modulation_close: BOOL := FALSE;

(*Переменные обмена с панелью оператора*)

     (*входы:*)
     y_ref_from_OP:REAL:=0; (*задание с панели*)
     y_ref_update_from_OP:BOOL:=FALSE; (*обновить задание с
                                         панели*)
     (*u_manual_from_OP:REAL:=0; (*сигнал ручного управления с
                                   панели*)*)
     man_from_OP:BOOL:=FALSE; (*сигнал перехода на ручной режим с
                                панели*)
```

```
      res_from_OP:BOOL; (*сброс регулятора с панели*)
      tic_tac_from_OP:BOOL; (*тестовый сигнал наличия связи с
                              панели*)
      open_from_OP:BOOL; (*включить привод на открытие вентиля*)
      close_from_OP:BOOL; (*включить привод на закрытие вентиля*)

      (*выходы:*)
      y_ref_to_OP, y_to_OP, u_to_OP:REAL; (*задание, выход и
                                          управление на панель*)
      res_to_OP:BOOL; (*сброс регулятора на панель*)
      state_to_OP:SINT; (*режим управления на панель*)
      block_panel_to_OP:BOOL; (*панель заблокирована на панель*)
      SCADA_link_to_OP:BOOL:=TRUE; (*связь со SCADA на панель*)
      err_gt_10_to_OP, err_lt_m10_to_OP, norma_to_OP:BOOL;(*сигналы
                                          контроля на панель*)
      tic_tac_to_OP:BOOL; (*тестовый сигнал наличия связи на
                            панель*)
      opening_to_OP:BOOL; (*привод включен на открытие вентиля*)
      closing_to_OP:BOOL; (*привод включен на закрытие вентиля*)
      opened_to_OP:BOOL; (*вентиль открыт*)
      closed_to_OP:BOOL; (*вентиль закрыт*)

(*Переменные обмена со SCADA-системой*)

      (*входы:*)
      y_ref_from_SCADA:REAL:=0; (*задание от SCADA*)
      y_ref_update_from_SCADA:BOOL:=FALSE; (*обновить задание от
                                          SCADA*)
      (*u_manual_from_SCADA:REAL:=0; (*сигнал дистанционного
                                    управления от SCADA*)*)
      dist_from_SCADA:BOOL:=FALSE; (*переход на дистанционное
                                    управление от SCADA*)
      block_panel_from_SCADA: BOOL:=FALSE; (*заблокировать панель
                                          от SCADA*)
      res_from_SCADA:BOOL:=FALSE; (*сброс регулятора от SCADA*)
      tic_tac_from_SCADA:BOOL; (*тестовый сигнал наличия связи от
                                SCADA*)
      open_from_SCADA:BOOL; (*включить привод на открытие вентиля*)
      close_from_SCADA:BOOL; (*включить привод на закрытие
                              вентиля*)
      (*выходы:*)
      y_to_SCADA:REAL; (*выходной сигнал в SCADA*)
      y_ref_to_SCADA:REAL; (*задание в SCADA*)
      u_to_SCADA:REAL; (*управление в SCADA*)
      res_to_SCADA:BOOL; (*сброс регулятора в SCADA*)
      state_to_SCADA:SINT; (*режим управления в SCADA*)
      block_panel_to_SCADA: BOOL:=TRUE; (*панель заблокирована  в
                                          SCADA*)
      OP_link_to_SCADA:BOOL:=TRUE; (*связь с панелью в SCADA*)
      tic_tac_to_SCADA:BOOL; (*тестовый сигнал наличия связи в
                              SCADA*)
      opening_to_SCADA:BOOL; (*привод включен на открытие вентиля*)
      closing_to_SCADA:BOOL; (*привод включен на закрытие вентиля*)
```

```
    opened_to_SCADA:BOOL; (*вентиль открыт*)
    closed_to_SCADA:BOOL; (*вентиль закрыт*)
END_VAR
```

Программа `REGULATION` занимается только тем, что вызывает экземпляр функционального блока `PD` (рис. 3.69). Настройки ПД регулятора в основном аналогичны настройкам соответствующего блока Simulink-диаграммы. Отличие состоит в отсутствии фильтрации, которая не предусмотрена разработчиками функционального блока.

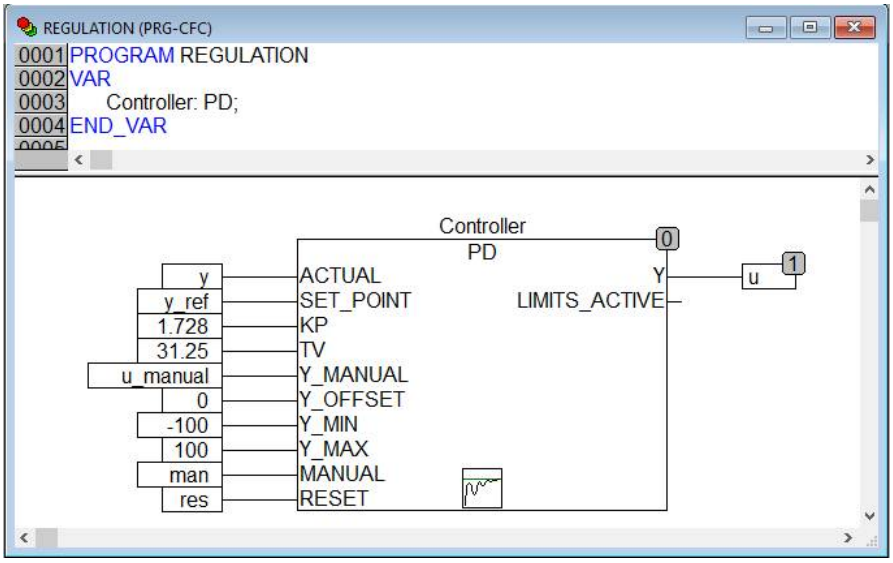


Рис. 3.69. Программа `REGULATION`.

Функция `APERTURA` особых комментариев не требует:

```
FUNCTION APERTURA : REAL
VAR_INPUT
     u, u_minimal: REAL;
END_VAR
VAR
END_VAR
IF ABS(u) < u_minimal THEN
     APERTURA:= 0;
ELSE
     APERTURA:= u;
END_IF
```

Программа `MODULATION` вызывает экземпляр функционального блока `WIM`, код которой различен для различных вариантов формирователя импульсов. Помимо этого есть отличия и в самой программе, рис. 3.70 – 3.72.

Для ШИМ с отбрасыванием коротких импульсов отсутствует необходимость ограничения входного сигнала по минимуму, поэтому функция `APERTURA` не вызывается. Выходные сигналы формирователя подаются непосредственно на выходы контроллера `open` и `close`. В ручном и дистанционном режимах работы системы управления этими выходами будет управлять оператор, а программа `MODULATION` просто не будет вызываться из `PLC_PRG`.

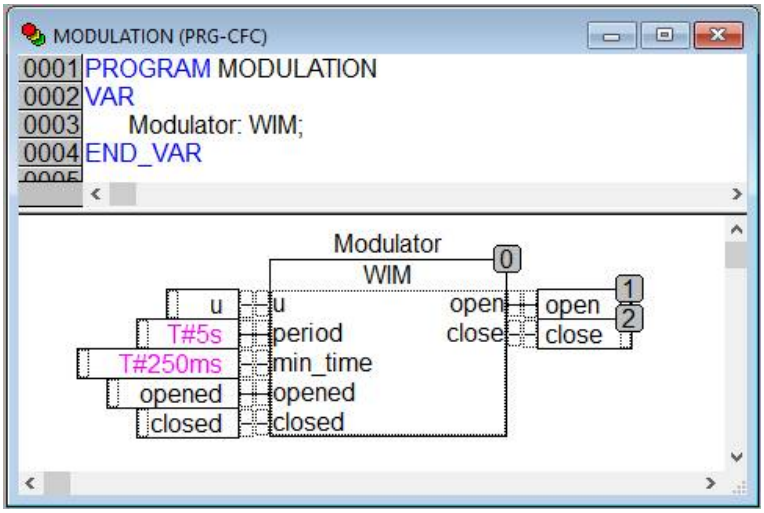


Рис. 3.70. Программа `MODULATION` для варианта с формирователем импульсов № 1.

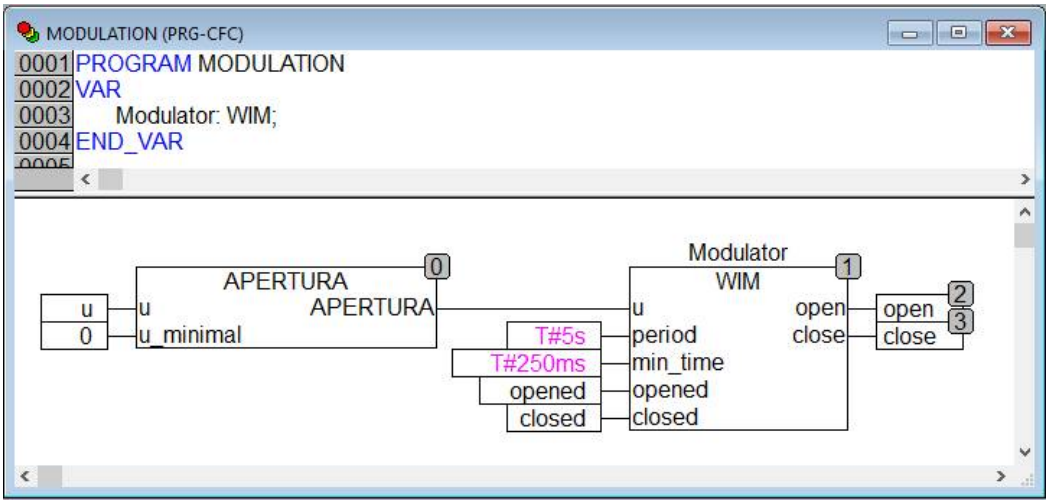


Рис. 3.71. Программа `MODULATION` для вариантов с формирователем импульсов № 2, 3.

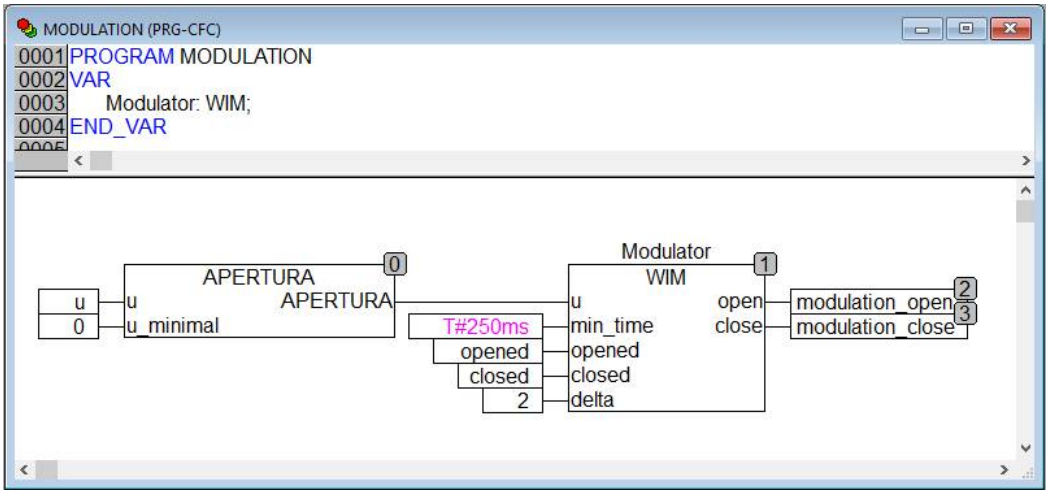


Рис. 3.72. Программа `MODULATION` для варианта с формирователем импульсов № 4.

Вариант для ШИМ с переносом импульсов и ШИМ с переменным периодом отличается вызовом функции APERTURA.

В варианте для следящего формирователя выходные сигналы экземпляра функционального блока WIM записываются в буферные переменные, которые используются только в режиме автоматического управления.

Функциональный блок WIM (формирователь импульсов) реализован в четырех вариантах.

Секции объявления переменных и кода функционального блока WIM для ШИМ с отбрасыванием коротких импульсов (вариант 1):

```
FUNCTION_BLOCK WIM
VAR_INPUT
     u:REAL; (*Входной сигнал*)
     period:TIME; (*Период ШИМ*)
     min_time:TIME; (*Минимальное время включения/отключения*)
     opened, closed:BOOL; (*Сигналы конечных выключателей*)
END_VAR
VAR_OUTPUT
     open:BOOL:=FALSE; (*Включить привод на открытие*)
     close:BOOL:=FALSE; (*Включить привод на закрытие*)
END_VAR
VAR
     timer:TON; (*Таймер для контроля времени*)
     recalc:BOOL:=TRUE; (*Разрешение на пересчет *)
     time_on:TIME; (*Время включения привода*)
     plus:BOOL; (*Направл. включ.: TRUE-"больше",FALSE-"меньше"*)
END_VAR
-------------------------------------------------------------
IF recalc THEN
     time_on:=REAL_TO_TIME(ABS(u)/100*TIME_TO_REAL(period));
     IF time_on < min_time THEN
           time_on:= T#0s;
     ELSIF (period - time_on) < min_time THEN
          time_on:=period;
     END_IF
     plus:=u > 0;
     timer(IN:=FALSE);
END_IF
timer(IN:=TRUE, PT:=period);
recalc:= timer.Q;
IF timer.ET < time_on THEN
     open:=plus AND NOT opened;
     close:= NOT plus AND NOT closed;
ELSE
     open:=close:=FALSE;
END_IF
```

Код получился несколько проще, чем код аналогичной функции в Matlab главным образом за счет того, что в функциональных блоках значения переменных «автоматически» сохраняются между вызовами.

Секции объявления переменных и кода функционального блока WIM для ШИМ с переносом коротких импульсов (вариант 2):

```
FUNCTION_BLOCK WIM
VAR_INPUT
     u:REAL; (*Входной сигнал*)
     period:TIME; (*Период ШИМ*)
     min_time:TIME; (*Минимальное время включения/отключения*)
     opened, closed:BOOL; (*Сигналы конечных выключателей*)
END_VAR
VAR_OUTPUT
     open:BOOL:=FALSE; (*Включить привод на открытие*)
     close:BOOL:=FALSE; (*Включить привод на закрытие*)
END_VAR
VAR
     timer:TON; (*Таймер для контроля времени*)
     recalc:BOOL:=TRUE; (*Разрешение на пересчет *)
     time_on:TIME; (*Время включения привода*)
     time_on_DINT:DINT;    (*Время  включения  привода  в  формате
DINT*)
     additive: DINT;  (*"Добавка на будущее"*)
     plus:BOOL; (*Направл. включ.: TRUE-"больше",FALSE-"меньше"*)
     time_off: TIME; (*Время паузы привода*)
END_VAR
-----------------------------------------------------------------
IF recalc THEN
     (*Расчет времени включения с учетом «добавки»*)
     time_on_DINT:=REAL_TO_DINT(u/100*TIME_TO_REAL(period)) +
               additive;
     IF time_on_DINT > 0 THEN (*время положительное, надо откр.*)
          time_on:= DINT_TO_TIME(time_on_DINT);
          plus:=TRUE;
     ELSE (*время отрицательное, надо закр.*)
          time_on:= DINT_TO_TIME(-time_on_DINT);
          plus:=FALSE;
     END_IF
     time_off:=period - time_on; (*определение времени паузы*)
     IF time_on < min_time THEN (*импульс слишком короткий*)
          additive:=time_on_DINT;
          time_on:= T#0s;
     ELSIF time_off < min_time THEN (*пауза слишком короткая*)
          IF plus THEN
               additive:=-TIME_TO_DINT(time_off);
          ELSE
               additive:=TIME_TO_DINT(time_off);
          END_IF
          time_on:= period;
     ELSE (*импульс и пауза – «нормальные»*)
          additive:=0;
     END_IF
     timer(IN:=FALSE);
END_IF
timer(IN:=TRUE, PT:=period);
recalc:= timer.Q;
IF timer.ET < time_on THEN
     open:=plus AND NOT opened;
```

```
     close:= NOT plus AND NOT closed;
ELSE
     open:=close:=FALSE;
END_IF
```

Основной проблемой при написании кода была невозможность хранения в переменных типа TIME «отрицательного времени» переносимого импульса движения в сторону «меньше» и переносимой паузы в сторону «больше». Поэтому потребовалось преобразование в тип DINT и обратное. В остальном реализуется та же логика, что и в соответствующей функции Matlab.

Секции объявления переменных и кода функционального блока WIM для ШИМ с переменным периодом (вариант 3):

```
FUNCTION_BLOCK WIM
VAR_INPUT
     u:REAL; (*Входной сигнал*)
     period:TIME; (*Период ШИМ*)
     min_time:TIME; (*Минимальное время включения/отключения*)
     opened, closed:BOOL; (*Сигналы конечных выключателей*)
END_VAR
VAR_OUTPUT
     open:BOOL:=FALSE; (*Включить привод на открытие*)
     close:BOOL:=FALSE; (*Включить привод на закрытие*)
END_VAR
VAR
     timer:TON; (*Таймер для контроля времени*)
     state: BOOL; (*Состояние привода TRUE-вкл., FALSE-выкл.*)
     module_u: REAL; (*Модуль сигнала управления*)
     time_of_state: TIME; (*Время текущего состояния*)
END_VAR
-----------------------------------------------------------------
module_u:= ABS(u);
timer(IN:=TRUE, PT:= t#1m);
time_of_state:=timer.ET;
IF state THEN (*импульс есть*)
     IF (module_u < 100) AND (time_of_state > min_time) THEN
(*импульс может быть снят*)
          IF(module_u<50)OR(time_of_state> min_time*
             REAL_TO_DWORD(module_u/(100 - module_u))) THEN
          (*импульс должен быть снят*)
               state:=0;
               open:= FALSE;
               close:= FALSE;
               timer(IN:=FALSE);
           END_IF
     END_IF
ELSE (*импульса нет*)
     IF (module_u >0)AND(time_of_state > min_time) THEN
     (*импульс может быть подан*)
           IF (module_u>50)OR(time_of_state>min_time*
          REAL_TO_DWORD((100-module_u)/module_u)) THEN
          (*импульс должен быть подан*)
               state:=TRUE;
```

```
                open:=(u > 0) AND NOT opened;
                close:=(u < 0) AND NOT closed;
                timer(IN:=FALSE);
            END_IF
     END_IF
END_IF
```

Код практически один к одному повторяет код соответствующей Matlab-функции.

Секции объявления переменных и кода функционального блока `WIM` для следящего формирователя (вариант 4):

```
FUNCTION_BLOCK WIM
VAR_INPUT
     u:REAL; (*Входной сигнал*)
     min_time:TIME; (*Минимальное время включения/отключения*)
     opened, closed:BOOL; (*Сигналы конечных выключателей*)
     delta: REAL; (*Минимальная ошибка "слежения"*)
END_VAR
VAR_OUTPUT
     open:BOOL:=FALSE; (*Включить привод на открытие*)
     close:BOOL:=FALSE; (*Включить привод на закрытие*)
END_VAR
VAR
     timer:TON; (*Таймер для контроля времени*)
     integ1: INTEGRAL; (*Интегратор входного сигнала*)
     integ2: INTEGRAL; (*Интегратор выходного сигнала*)
     x1: REAL; (*Выход integ1*)
     x2: REAL; (*Выход integ2*)
     x: REAL; (*Выходной сигнал в формате -100,0,100*)
     error: REAL; (*x1 - x2*)
END_VAR
-------------------------------------------------------------
integ1(IN:=0.03*u,TM:=50,OUT=>x1);
integ2(IN:=0.03*x,TM:=50,OUT=>x2);
error:=x1-x2;
timer(IN:=TRUE, PT:=min_time);
IF timer.Q THEN
     IF open AND error < 0 THEN
          open:=FALSE;
          x:=0;
          timer(IN:=FALSE);
     ELSIF close AND error > 0 THEN
          close:=FALSE;
          x:=0;
          timer(IN:=FALSE);
     ELSE
          IF error > delta THEN
               open:=TRUE;
               x:=100;
               timer(IN:=FALSE);
          ELSIF error < - delta THEN
               close:=TRUE;
```

```
            x:=-100;
            timer(IN:=FALSE);
        END_IF
    END_IF
END_IF
```

Код функционального блока реализует не только логику соответствующей Matlab-функции, но и подсистемы, из которой она вызывается.

Программа обслуживания виртуальной панели оператора теперь имеет следующий список переменных (изменения выделены жирным шрифтом):

```
PROGRAM PANEL
VAR
    (*ОТОБРАЖЕНИЕ*)
    y_ref, y, u:REAL; (*задание, выход и управление*)
    err_gt_10, err_lt_m10, norma:BOOL; (*сигналы контроля *)
    block_panel:BOOL; (*блокировать панель - на панель*)
    SCADA_link:BOOL:=TRUE; (*связь со SCADA на панель*)
    state: STRING; (*режим управления*)
    res:BOOL; (*сброс регулятора*)
    opening:BOOL; (*привод включен на открытие вентиля*)
    closing:BOOL; (*привод включен на закрытие вентиля*)
    moving: INT; (*для отображения движения привода на тренде*)
    opened:BOOL; (*вентиль открыт*)
    closed:BOOL; (*вентиль закрыт*)

    (*УПРАВЛЕНИЕ И ВВОД*)
    y_ref_local:REAL:=0; (*локальное задание*)
    y_ref_update:BOOL:=FALSE; (*обновить задание*)
    man:BOOL:=FALSE; (*сигнал перехода на ручной режим*)
    do_reset:BOOL:=FALSE; (*сброс регулятора*)
    ERROR:BOOL:=FALSE; (*отказ панели*)
    open:BOOL; (*включить привод на открытие вентиля*)
    close:BOOL; (*включить привод на закрытие вентиля*)
END_VAR
```

Изменился и сам вид главного окна панели (рис. 3.73).

Теперь на втором тренде красным цветом отображается график выходного сигнала ПД регулятора, а синим – импульсы, подаваемые на привод исполнительного механизма, которые представлены значениями переменной `moving`.

Код программы `PANEL` (изменения выделены жирным):

```
IF NOT ERROR THEN
    y_ref:= y_ref_to_OP;
    y:= y_to_OP;
    err_gt_10:=err_gt_10_to_OP;
    err_lt_m10:=err_lt_m10_to_OP;
    norma:=norma_to_OP;
    block_panel:=block_panel_to_OP;
    SCADA_link:=SCADA_link_to_OP;
    res:=res_to_OP;
    opening:=opening_to_OP;
    closing:=closing_to_OP;
    opened:=opened_to_OP;
```

```
     closed:=closed_to_OP;
     moving:=50*BOOL_TO_INT(opening) - 50*BOOL_TO_INT(closing);
     CASE state_to_OP OF
     -1:
          state:='Авария';
          u:=u_to_OP;
     0:
          state:='Ручной';
     1:
          state:='Автомат';
          u:=u_to_OP;
     2:
          state:='Дистанц.';
          u:=u_to_OP;
     END_CASE
     y_ref_from_OP:=y_ref_local;
     y_ref_update_from_OP:=y_ref_update;
     (*u_manual_from_OP:=u;*)
     man_from_OP:=man;
     res_from_OP:=do_reset;
     open_from_OP:=open;
     close_from_OP:=close;
     tic_tac_from_OP:=NOT tic_tac_to_OP;
ELSE
     state:='Ошибка связи';
END_IF
```

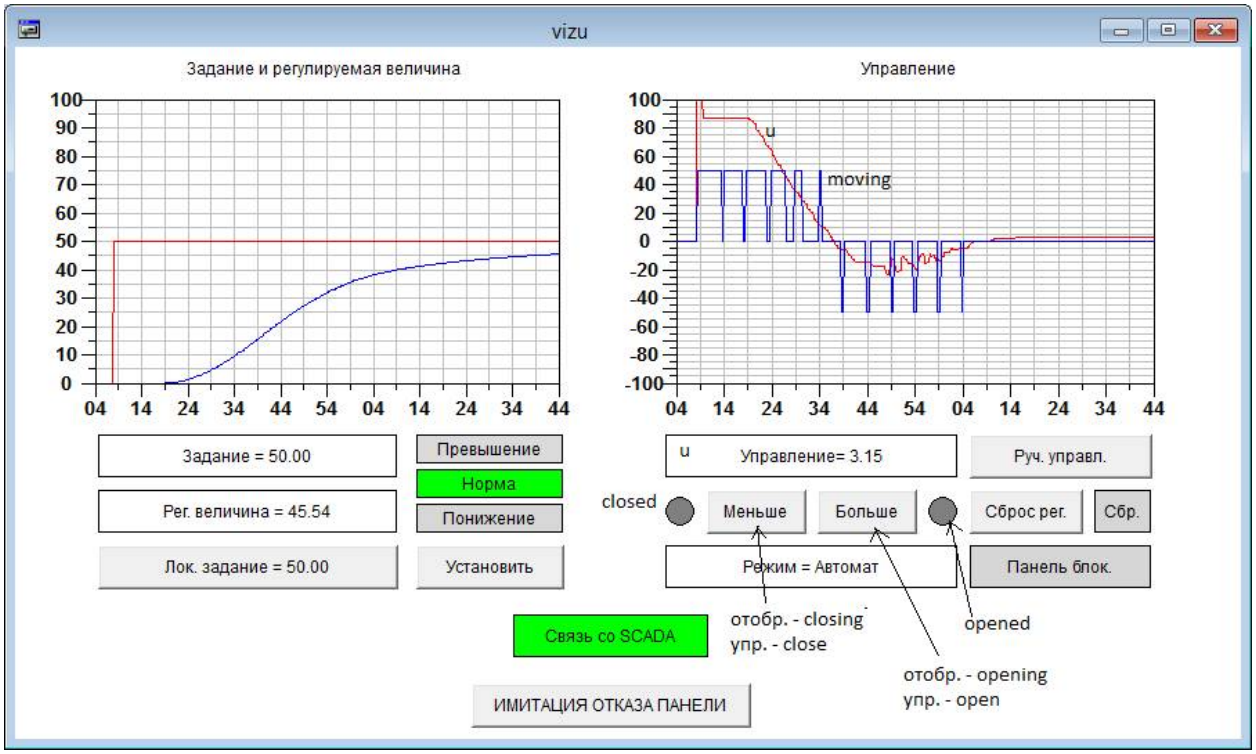


Рис. 3.73. Главное окно панели регулятора с дополнительными привязками.

Для наглядности отображения импульсов принято, что переменная `moving` может принимать значения –50, 0, 50 в зависимости от наличия сигналов `opening` и `closing`.

В программу `PLC_PRG` внесены незначительные изменения.

Ниже приведены разделы объявления и кода программы `PLC_PRG`.

```
PROGRAM PLC_PRG
VAR
     (*режим управления*)
     state:SINT:=0; (*-1 - авария,1 - авт., 2 - дист.*)
     (*таймеры связи*)
     SCADA_EXCHANGE_WATCHDOG_1, SCADA_EXCHANGE_WATCHDOG_0 : TON;
     OP_EXCHANGE_WATCHDOG_1, OP_EXCHANGE_WATCHDOG_0 : TON;
     (*таймер регулирования*)
     STOP_CONTROL_TIMER:TON;
     (*флаги ошибок связи*)
     SCADA_FAIL:BOOL:=FALSE;
     OP_FAIL:BOOL:=FALSE;
END_VAR
-----------------------------------------------------------------
(*Запись по интерфейсам*)
y_to_SCADA:=y_to_OP:=y;
y_ref_to_SCADA:=y_ref_to_OP:=y_ref;
u_to_SCADA:=u_to_OP:=u;
block_panel_to_SCADA:= block_panel_to_OP:= block_panel;
state_to_SCADA:=state_to_OP:=state;
res_to_SCADA:=res_to_OP:=res;
opening_to_SCADA:=opening_to_OP:=open;
closing_to_SCADA:=closing_to_OP:=close;
opened_to_SCADA:=opened_to_OP:=opened;
closed_to_SCADA:=closed_to_OP:=closed;

(*Проверка связи со SCADA и панелью*)
(*по умолчанию - все нормально*)
SCADA_link_to_OP:=OP_link_to_SCADA:=TRUE;
SCADA_FAIL:=OP_FAIL:=FALSE;
(*ретрансляция tic_tac*)
tic_tac_to_SCADA:= tic_tac_from_SCADA;
tic_tac_to_OP:= tic_tac_from_OP;
(*запуск таймеров*)
SCADA_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_SCADA, PT:=t#5s);
SCADA_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_SCADA, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_OP, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_OP, PT:=t#5s);
(*установка флагов*)
SCADA_FAIL:=SCADA_EXCHANGE_WATCHDOG_1.Q OR
            SCADA_EXCHANGE_WATCHDOG_0.Q;
OP_FAIL:= OP_EXCHANGE_WATCHDOG_1.Q OR
     OP_EXCHANGE_WATCHDOG_0.Q;
no_HMI:= SCADA_FAIL AND OP_FAIL;

block_panel:=block_panel_from_SCADA;

(*Обработка ошибок связи*)
IF  SCADA_FAIL THEN
     block_panel:=OP_FAIL;
```

```
     SCADA_link_to_OP:=FALSE;
     y_ref_from_SCADA:=y_ref;
     y_ref_update_from_SCADA:=FALSE;
END_IF
IF OP_FAIL THEN
     block_panel:=TRUE;
     OP_link_to_SCADA:=FALSE;
     y_ref_from_OP:=y_ref;
     y_ref_update_from_OP:=FALSE;
END_IF

(*Обновление задания*)
IF y_ref_update_from_SCADA AND NOT SCADA_FAIL THEN
     y_ref:=y_ref_from_SCADA;
END_IF
IF y_ref_update_from_OP AND NOT block_panel THEN
     y_ref:=y_ref_from_OP;
END_IF

stop:=FALSE; (*по умолчанию оборудование работает*)

(*Формирование сигнала сброса регулятора*)
res:= (res_from_SCADA AND NOT SCADA_FAIL) OR
      (res_from_OP AND NOT block_panel);

(*Формирование сигналов контроля для панели*)
IF y_ref - y  < -10 THEN
     err_gt_10_to_OP:=TRUE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=FALSE;
ELSIF y_ref - y  > 10 THEN
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=TRUE;
     norma_to_OP:=FALSE;
ELSE
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=TRUE;
END_IF

(*Режимы управления*)
CASE state OF
-1: (*авария*)
     y_ref:=0;
     u_manual:=0;
     res:=TRUE;
     stop:=TRUE;
     IF reset_process THEN
          state:=0;
     END_IF
0: (*ручное управление*)
(*u_manual:=u_manual_from_OP;*)
u_manual:= 0;
```

```
     open:=open_from_OP AND NOT opened;
     close:=close_from_OP AND NOT closed;
     man:=TRUE;
     (*переходы в другие режимы*)
     IF block_panel OR NOT man_from_OP OR OP_FAIL THEN
          IF dist_from_SCADA  AND NOT SCADA_FAIL THEN
               state:=2;
          ELSE
               (*однократный запуск регулятора для сброса*)
               res:=TRUE;
               REGULATION;
               res:=FALSE;
               state:=1;
          END_IF
     END_IF
1: (*автоматическое управление*)
     u_manual:=u;
     man:=FALSE;
     (*ДЛЯ ВАРИАНТОВ С ФОРМИРОВАТЕЛЯМИ ИМПУЛЬСОВ №1,2,3*)
          MODULATION;
     (*ДЛЯ ВАРИАНТА С ФОРМИРОВАТЕЛЯМ ИМПУЛЬСОВ №4*)
          open:=modulation_open AND NOT opened;
          close:=modulation_close AND NOT closed;
     (*переходы в другие режимы*)
     IF man_from_OP AND NOT block_panel THEN
          state:=0;
     ELSIF dist_from_SCADA  AND NOT SCADA_FAIL THEN
          state:=2;
     END_IF
     (*Если нет связи со SCADA и панелью, установка "безопасного"
уровня задания*)
     IF NO_HMI THEN
          y_ref:=50;
     END_IF
     (*Если связь с HMI утеряна и процесс в течение 1 мин.
не удается вернуть в норму - авария*)
     STOP_CONTROL_TIMER(IN:=no_HMI AND
           (err_gt_10_to_OP OR err_lt_m10_to_OP), PT:=t#1m);
     IF STOP_CONTROL_TIMER.Q THEN
          state:=-1;
     END_IF

2: (*дистанционное управление*)
     (*u_manual:=u_manual_from_SCADA;*)
     u_manual:= 0;
     man:=TRUE;
     open:=open_from_SCADA AND NOT opened;
     close:=close_from_SCADA AND NOT closed;
     (*переходы в другие режимы*)
     IF (man_from_OP AND NOT block_panel)  THEN
          state:=0;
     ELSIF NOT dist_from_SCADA OR SCADA_FAIL THEN
          (*однократный запуск регулятора для сброса*)
          res:=TRUE;
```

```
            REGULATION;
            res:=FALSE;
            state:=1;
      END_IF
END_CASE
(*Регулятор*)
(*CONTROL;*)
(*Панель*)
PANEL;
```

Прокомментируем только два момента:

1) программа MODULATION, формирующая импульсы, вызывается (варианты № 1, 2, 3) или результаты ее работы используются (вариант № 4) только в режиме автоматического управления. В ручном и дистанционном режимах сигналы open и close формируются операторами непосредственно;

2) при переходе в режим автоматического управления регулятор принудительно сбрасывается путем внепланового вызова программы REGULATION, иначе, как показали эксперименты, стандартный ПД-регулятор начинает работать неправильно. Причина неправильной работы, по-видимому, кроется в том, что разработчики алгоритма не предусмотрели возможности формирования управления «в обход» регулятора (не через входы MANUAL и Y_MANUAL).

3.1.2.4. Модернизация SCADA-системы

В SCADA-системе, разработанной в п. 3.1.1.4, потребовалось сделать совсем немного изменений.

В «Источниках/Приемниках» добавлены OPC-компоненты, обеспечивающие получение информации о состоянии исполнительного механизма и дистанционное управление им (рис. 3.74).

«Приемник» u_manual_to_PLC исключен из группы Write за ненадобностью.

Одноименный канал удален и из списка каналов (рис 3.75).

Вместо u_manual_to_PLC в список добавлен канал «Управление-импульсы». Он будет использоваться для формирования тренда импульсов, подаваемых на исполнительный механизм. Канал ни к чему не привязан, но архивируется в СПАД №1.

На рис. 3.76 показан экран визуализации в действии, запущенный под управлением профайлера Trace Mode.

Внесенные изменения:

1) на графический элемент «Тренд» добавлена кривая для отображения импульсов управления исполнительным механизмом. Кривая, ранее отображавшая непрерывный сигнал управления, теперь отображает выходной сигнал ПД-регулятора. К этому же сигналу теперь привязан и левый ползунок «Управление» (диапазон изменения – -100…100%);

2) на месте правого ползунка «Управление», с помощью которого формировался непрерывный сигнал дистанционного управления, теперь размещены кнопки «Больше», «Меньше» и индикаторы включения привода и достижения им крайних положений («появляются» только при наличии соответствующих

сигналов). Кнопки, ранее отвечавшие за формирование непрерывного управляющего сигнала, удалены;

3) в архивную таблицу добавлен столбец для вывода импульсов управления.

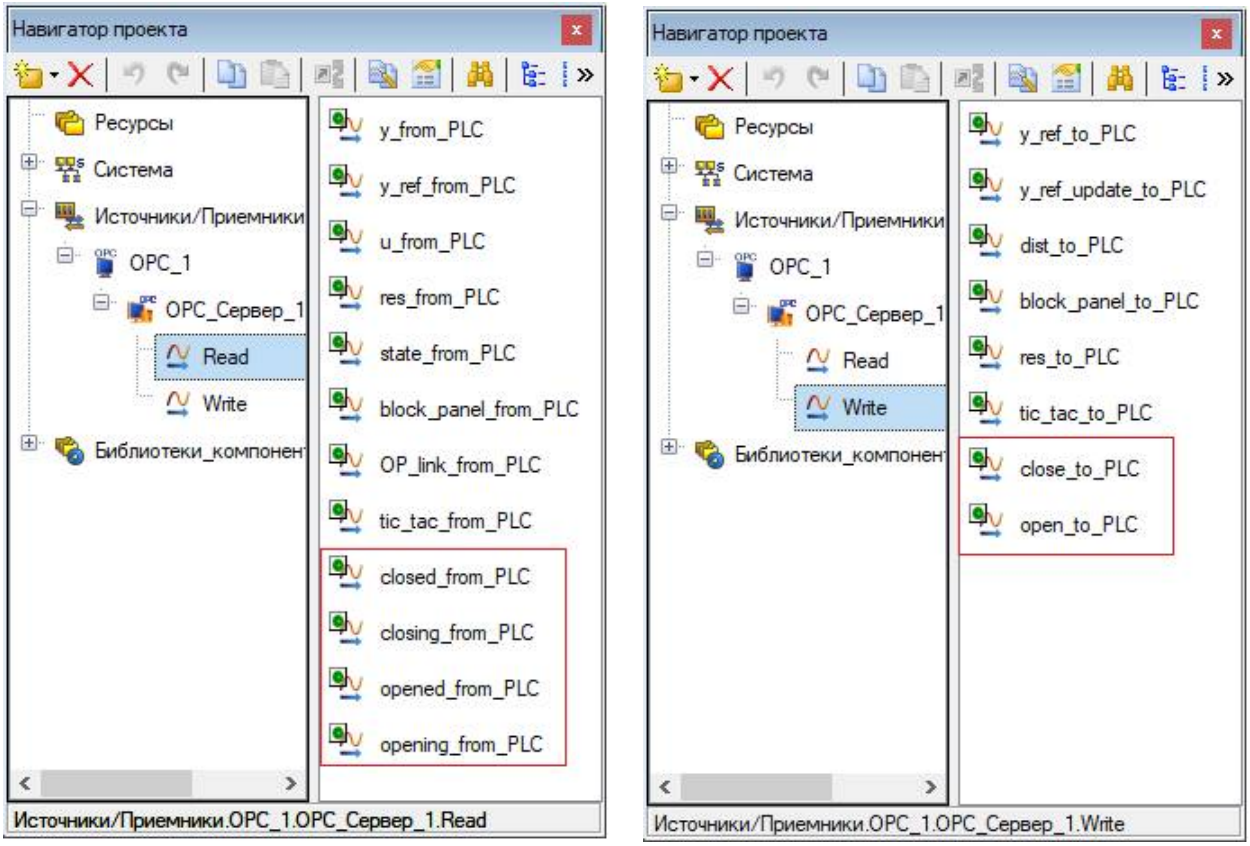


Рис. 3.74. «Источники/Приемники»

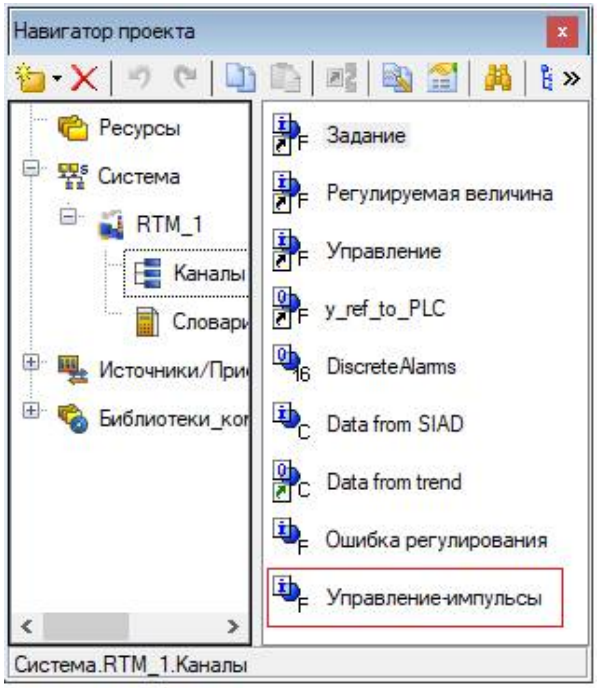


Рис. 3.75. Каналы.

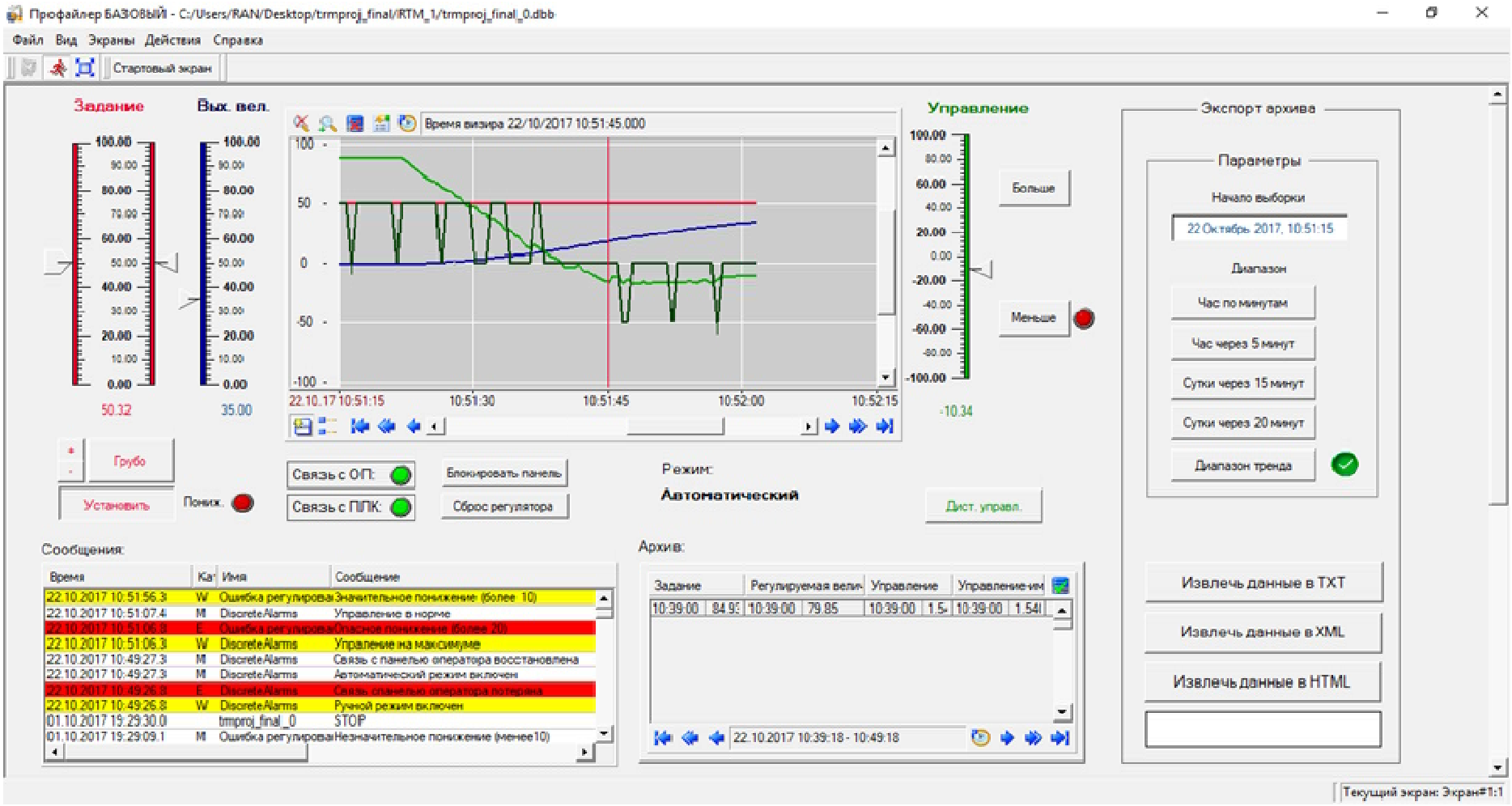
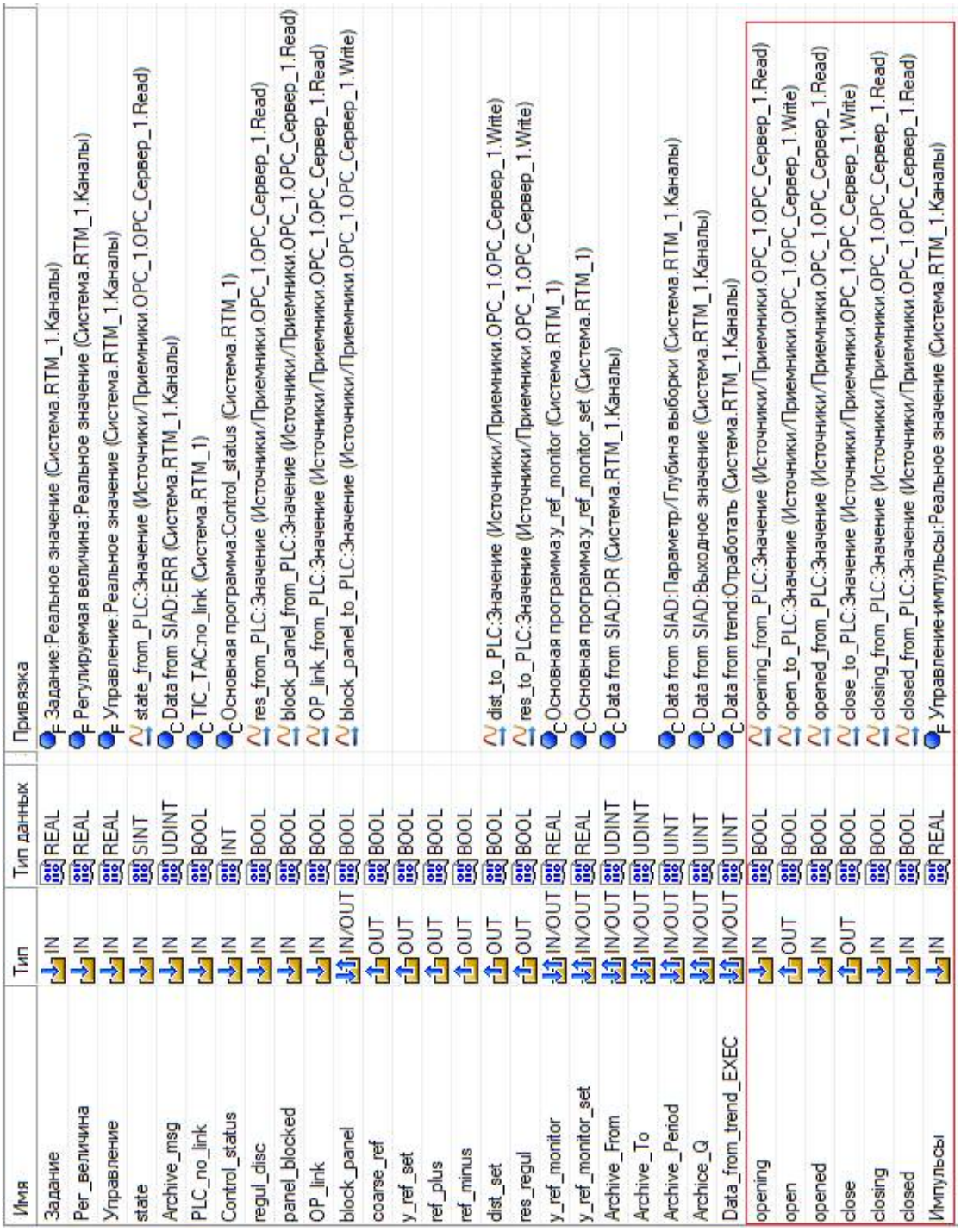


Рис. 3.76. Экран визуализации в работе.

Аргументы экрана и их привязки показаны на рис. 3.77. Из списка аргументов исключены переменные `u_manual`, `control_plus`, `control_minus`, `coarse_control`, `control_monitor`, `control_monitor_set`, ранее отвечавшие за формирование сигнала управления в дистанционном режиме. Вместо них добавлены новые переменные, выделенные на рис. 3.77.



Имя	Тип	Тип данных	Привязка
Задание	IN	REAL	Задание:Реальное значение (Система.RTM_1.Каналы)
Рег_величина	IN	REAL	Регулируемая величина:Реальное значение (Система.RTM_1.Каналы)
Управление	IN	REAL	Управление:Реальное значение (Система.RTM_1.Каналы)
state	IN	SINT	state_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
Archive_msg	IN	UDINT	Data from SIAD:ERR (Система.RTM_1.Каналы)
PLC_no_link	IN	BOOL	TIC_TAC:no_link (Система.RTM_1)
Control_status	IN	INT	Основная программа:Control_status (Система.RTM_1)
regul_disc	IN	BOOL	res_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
panel_blocked	IN	BOOL	block_panel_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
OP_link	IN	BOOL	OP_link_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
block_panel	IN/OUT	BOOL	block_panel_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
coarse_ref	OUT	BOOL	
y_ref_set	OUT	BOOL	
ref_plus	OUT	BOOL	
ref_minus	OUT	BOOL	
dist_set	OUT	BOOL	dist_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
res_regul	OUT	BOOL	res_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
y_ref_monitor	IN/OUT	REAL	Основная программа:y_ref_monitor (Система.RTM_1)
y_ref_monitor_set	IN/OUT	REAL	Основная программа:y_ref_monitor_set (Система.RTM_1)
Archive_From	IN/OUT	UDINT	Data from SIAD:DR (Система.RTM_1.Каналы)
Archive_To	IN/OUT	UDINT	
Archive_Period	IN/OUT	UINT	Data from SIAD:Параметр/Глубина выборки (Система.RTM_1.Каналы)
Archice_Q	IN/OUT	UINT	Data from SIAD:Выходное значение (Система.RTM_1.Каналы)
Data_from_trend_EXEC	IN/OUT	UINT	Data from trend:Отработать (Система.RTM_1.Каналы)
opening	IN	BOOL	opening_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
open	OUT	BOOL	open_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
opened	IN	BOOL	opened_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
close	OUT	BOOL	close_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
closing	IN	BOOL	closing_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
closed	IN	BOOL	closed_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
Импульсы	IN	REAL	Управление-импульсы:Реальное значение (Система.RTM_1.Каналы)

Рис. 3.77. Аргументы экрана и их привязки.

Привязки графических элементов экрана к его аргументам показаны на рис. 3.78 в виде текстовых надписей текстовых надписей поверх экрана визуализации в «нерабочем» состоянии.

Из других компонентов проекта Trace Mode (программ и словарей сообщений) модификация коснулась только Основной программы.

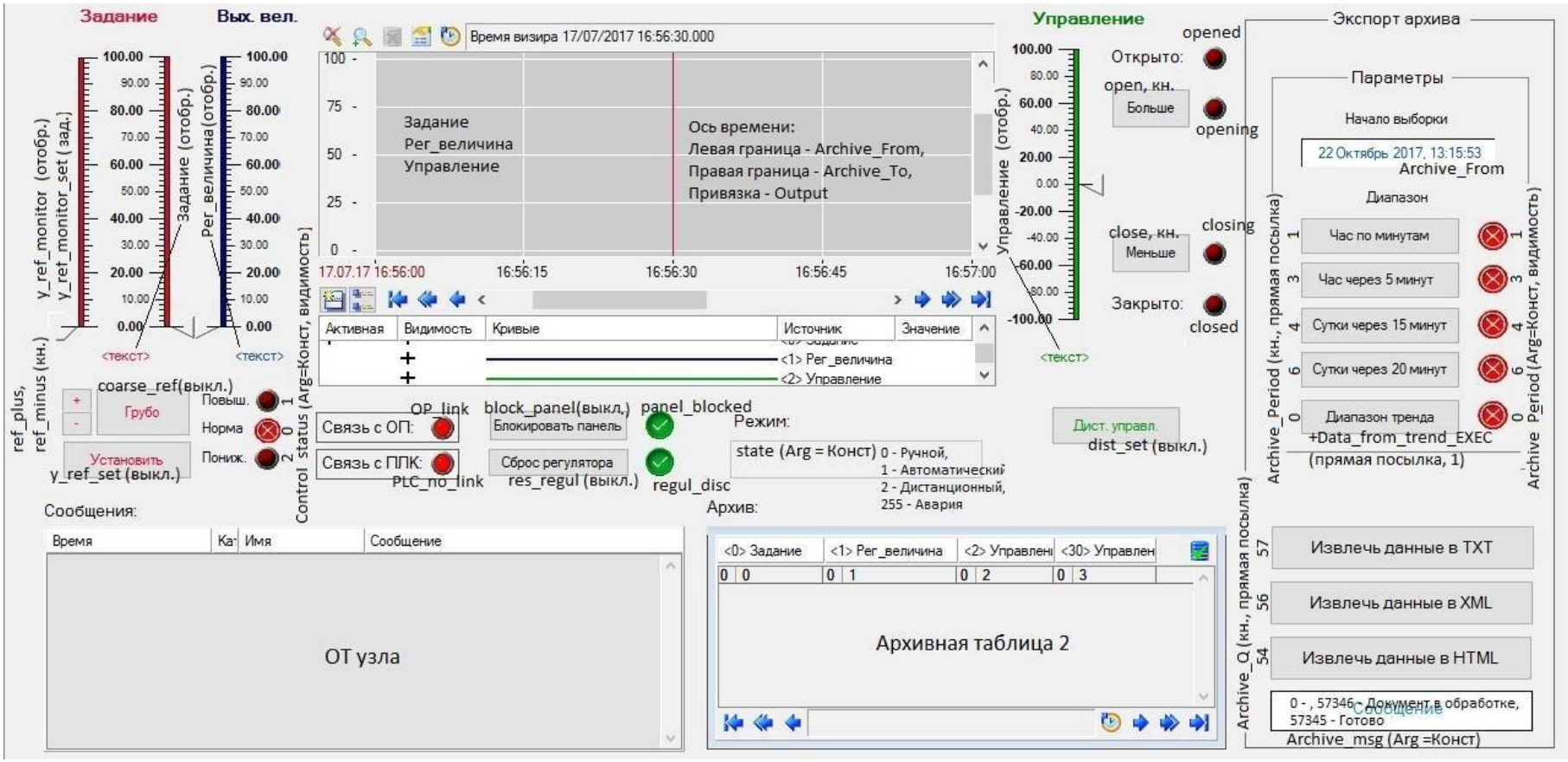


Рис. 3.78. Привязки графических элементов к аргументам экрана.

Листинг обновленной программы:

```
PROGRAM
     VAR_INPUT ref_plus : BOOL; END_VAR
     VAR_INPUT ref_minus : BOOL; END_VAR
     VAR_INPUT coarse_y_ref : BOOL; END_VAR
     VAR_INPUT set_ref : BOOL; END_VAR
     VAR_INPUT y_ref : REAL; END_VAR
     VAR_INPUT y : REAL; END_VAR
     VAR_INPUT state : SINT; END_VAR
     VAR_INPUT OP_link_from_PLC : BOOL := true; END_VAR
     VAR_INPUT opening : BOOL; END_VAR
     VAR_INPUT closing : BOOL; END_VAR
     VAR_INPUT opened : BOOL; END_VAR
     VAR_INPUT closed : BOOL; END_VAR
     VAR_OUTPUT error : REAL; END_VAR
     VAR_OUTPUT y_ref_to_PLC : REAL; END_VAR
     VAR_OUTPUT y_ref_update : REAL; END_VAR
     VAR_OUTPUT Control_status : INT; END_VAR
     VAR_OUTPUT u_impulse : REAL; END_VAR
     VAR_INOUT y_ref_monitor : REAL; END_VAR
     VAR_INOUT y_ref_monitor_set : REAL; END_VAR
     VAR_INOUT DiscreteAlarms : UINT; END_VAR


     //ЗАДАНИЕ
     IF coarse_y_ref THEN //грубый ввод задания
          //ползунок идет за мышью
          y_ref_monitor:=y_ref_monitor_set;
     ELSE //точный ввод, реакция на кнопки "+","-"
          IF ref_plus THEN
               y_ref_monitor:=y_ref_monitor+1;
          ELSIF ref_minus THEN
               y_ref_monitor:=y_ref_monitor-1;
          END_IF;
          y_ref_monitor_set:=y_ref_monitor;
     END_IF;
     IF set_ref THEN //Установка задания
          y_ref_update:=TRUE;
          y_ref_to_PLC:=y_ref_monitor;
     ELSE
          y_ref_update:=FALSE;
          y_ref_to_PLC:=y_ref; //отправляем то, что получили
     END_IF;
     //ФОРМИРОВАНИЕ СТАТУСА
     error:=y_ref-y; //ошибка регулирования
     IF error > 10 THEN
          Control_status:= 2;
     ELSIF error < -10 THEN
          Control_status:= 1;
```

```
    ELSE
        Control_status:= 0;
    END_IF;

    //ФОРМИРОВАНИЕ СООБЩЕНИЙ
    //сброс тех битов, которые могут быть установлены
    DiscreteAlarms:= DiscreteAlarms & 2#1000000;
    CASE state OF
    255: DiscreteAlarms:=DiscreteAlarms | 2#1;
    0: DiscreteAlarms:=DiscreteAlarms | 2#10;
    1: DiscreteAlarms:=DiscreteAlarms | 2#100;
    2: DiscreteAlarms:=DiscreteAlarms | 2#1000;
    END_CASE;

    IF opened THEN
        DiscreteAlarms:=DiscreteAlarms | 2#10000;
    ELSIF closed THEN
        DiscreteAlarms:=DiscreteAlarms | 2#100000;
    END_IF;
    IF NOT OP_link_from_PLC THEN
        DiscreteAlarms:=DiscreteAlarms | 2#10000000;
    END_IF;

    // ФОРМИРОВАНИЕ ТРЕНДА ИМПУЛЬСОВ
    IF opening THEN
        u_impulse:=50;
    ELSIF closing THEN
        u_impulse:=-50;
    ELSE
        u_impulse:=0;
    END_IF;
END_PROGRAM
```

Исправления:

1) удален «раздел» «Дистанционное управление»;

2) скорректирован механизм формирования сообщений «Управление на минимуме» и «Управление на максимуме»: теперь они формируются сигналами opened и closed;

3) добавлен «раздел» «Формирование тренда импульсов».

Соответствующим образом изменился список аргументов (рис. 3.79).

Имя	Тип	Тип данных	Значен	Привязка
ref_plus	IN	BOOL		Экран#1:1:ref_plus (Система.RTM_1)
ref_minus	IN	BOOL		Экран#1:1:ref_minus (Система.RTM_1)
coarse_y_ref	IN	BOOL		Экран#1:1:coarse_ref (Система.RTM_1)
set_ref	IN	BOOL		Экран#1:1:y_ref_set (Система.RTM_1)
y_ref	IN	REAL		Задание:Реальное значение (Система.RTM_1.Каналы)
y	IN	REAL		Регулируемая величина:Реальное значение (Система.RTM_1.Каналы)
state	IN	SINT		state_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
OP_link_from_PLC	IN	BOOL	true	OP_link_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
opening	IN	BOOL		opening_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
closing	IN	BOOL		closing_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
opened	IN	BOOL		opened_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
closed	IN	BOOL		closed_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
error	OUT	REAL		Ошибка регулирования:Входное значение (Система.RTM_1.Каналы)
y_ref_to_PLC	OUT	REAL		y_ref_to_PLC:Входное значение (Система.RTM_1.Каналы)
y_ref_update	OUT	REAL		y_ref_update_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
Control_status	OUT	INT		
u_impulse	OUT	REAL		Управление-импульсы:Входное значение (Система.RTM_1.Каналы)
y_ref_monitor	IN/OUT	REAL		Экран#1:1:y_ref_monitor (Система.RTM_1)
y_ref_monitor_set	IN/OUT	REAL		Экран#1:1:y_ref_monitor_set (Система.RTM_1)
DiscreteAlarms	IN/OUT	UINT		DiscreteAlarms:Входное значение (Система.RTM_1.Каналы)

Рис. 3.79. Аргументы Основной программы.

3.1.2.5. Апробация системы

Из результатов апробации приведем только графики изменения управляющего воздействия и выходной (регулируемой) величины эталонной (Simulink) системы и системы на базе виртуального ПЛК. Тренды сохранены в блоках Scope «u» и «yy» Simulink-диаграммы, показанной на рис. 3.64, в переменных `u` и `y` (рис. 3.80).

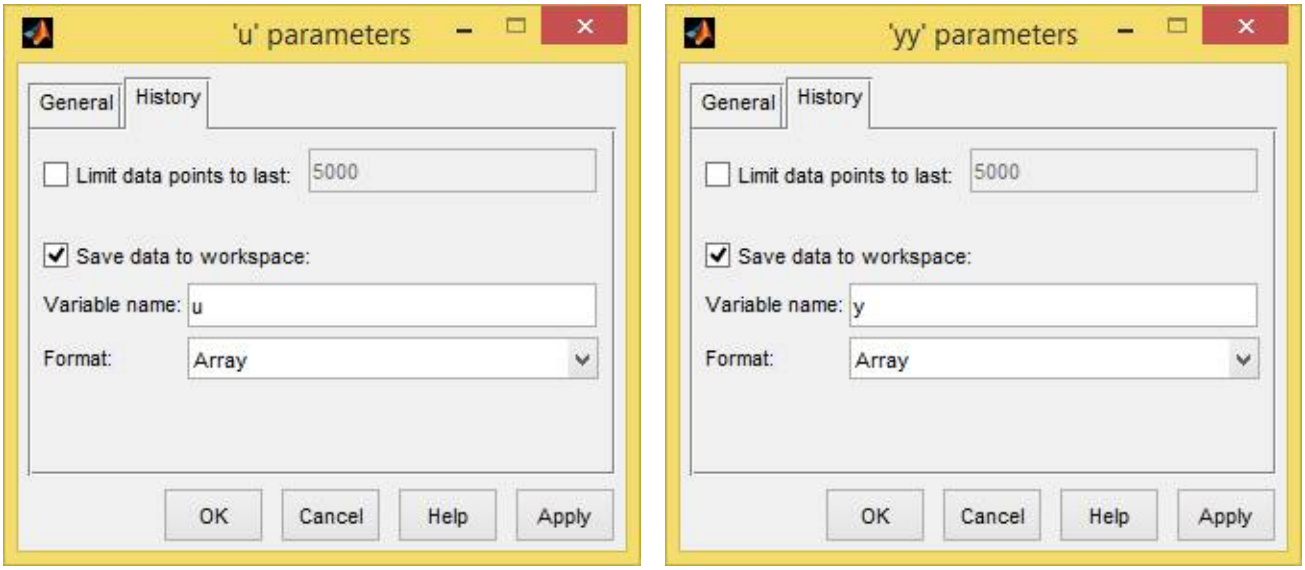


Рис. 3.80. Сохранение трендов.

Графики построены с помощью кода, приведенного ниже:

```
subplot(2,1,1)
L = plot(u(:,1),u(:,2),u(:,1),u(:,3)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(2,1,2)
L = plot(y(:,1),y(:,2),y(:,1),y(:,3)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
```

На рис. 3.81 – 3.84 показаны переходные процессы отработки изменения задающего воздействия и возмущения. Задание (50%) формировалось с экрана «операторской панели» и SCADA-системы, а возмущение (20%) – с помощью блока Slider Gain Simulink-диаграммы.

Результаты апробации в целом говорят о сходности поведения систем регулирования и корректной реализации алгоритмов на ПЛК. Расхождения в графиках объясняются в первую очередь отсутствием синхронизации эталонного и программного формирователей импульсов. Свою роль играют также различия в реализации ПД регуляторов и задержка при обмене по OPC. Однако, как видно по рисункам, расхождения не носят принципиального характера.

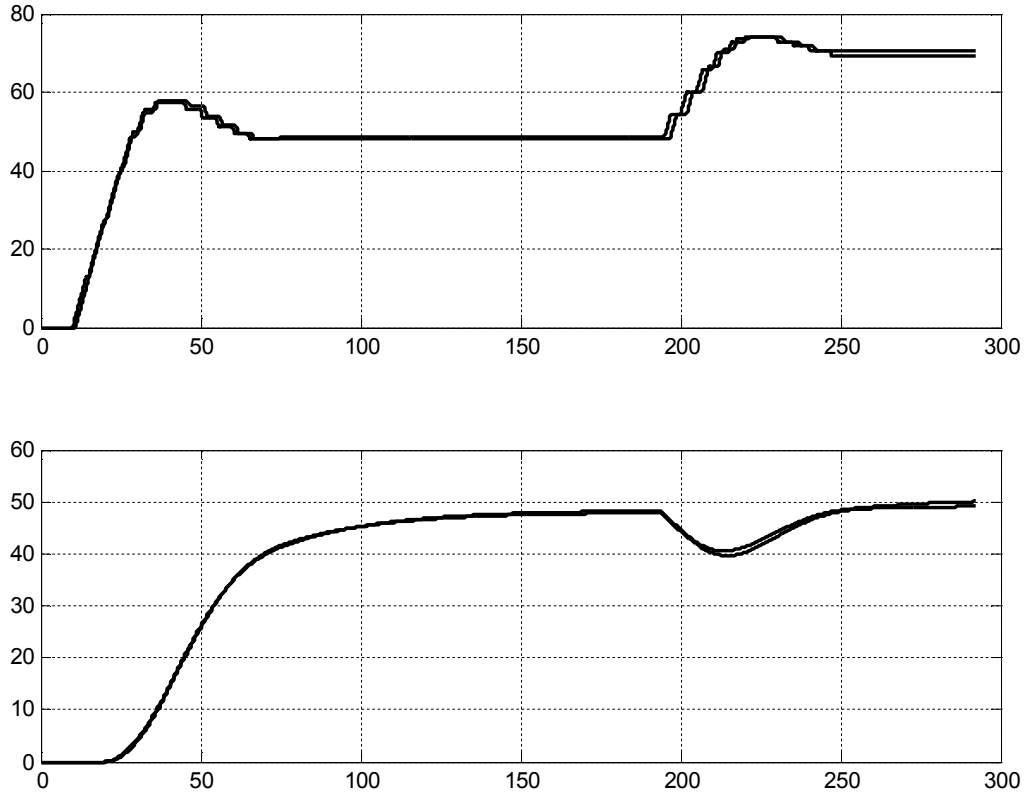


Рис. 3.81. Управление и выходная величина, ШИМ
с отбрасыванием коротких импульсов.

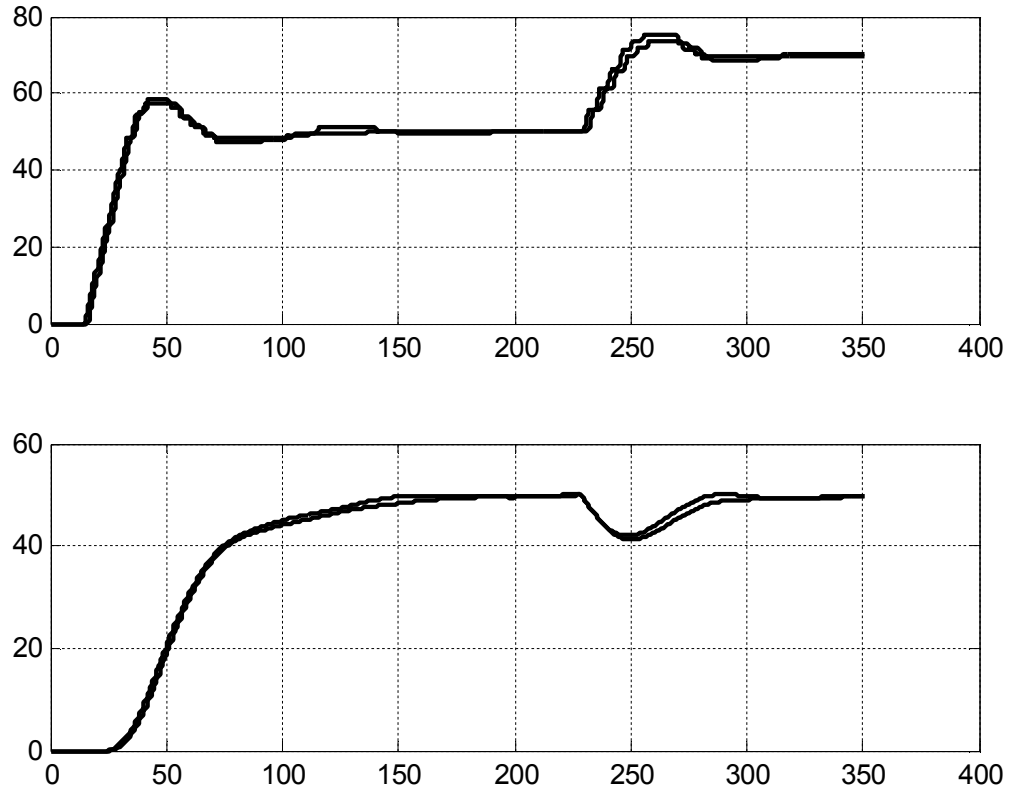


Рис. 3.82. Управление и выходная величина, ШИМ
с переносом коротких импульсов.

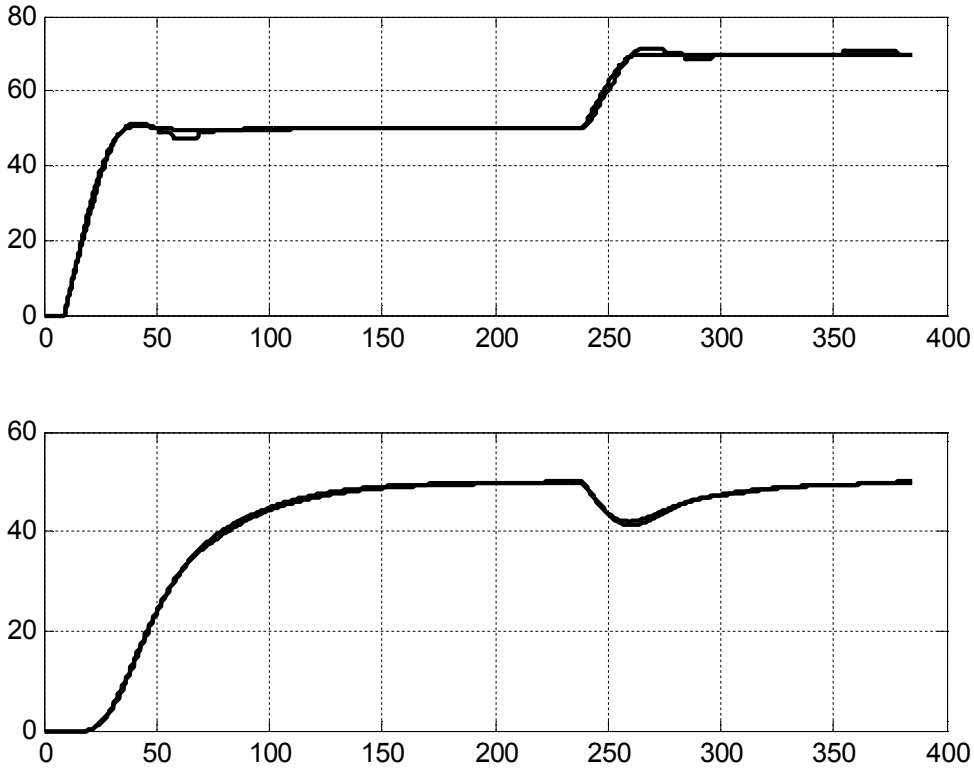


Рис. 3.83. Управление и выходная величина,
ШИМ с переменным периодом.

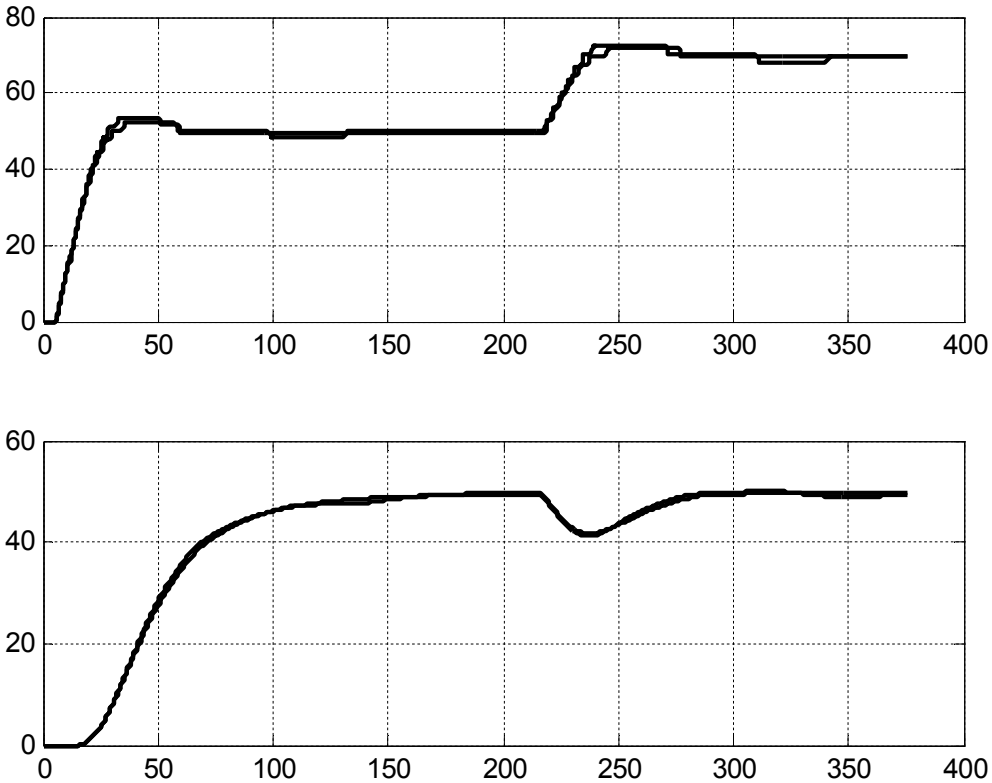


Рис. 3.84. Управление и выходная величина, следящий алгоритм.

3.1.3. Система автоматического регулирования с ИМ постоянной скорости с измерением положения

3.1.3.1. Структура системы регулирования

При наличии измерительного преобразователя положения исполнительного механизма появляется возможность построить каскадную систему с внутренним контуром регулирования положения. Поскольку скорость движения исполнительного механизма изменять нельзя, регулятор внутреннего контура будет релейного типа, рис. 3.85.

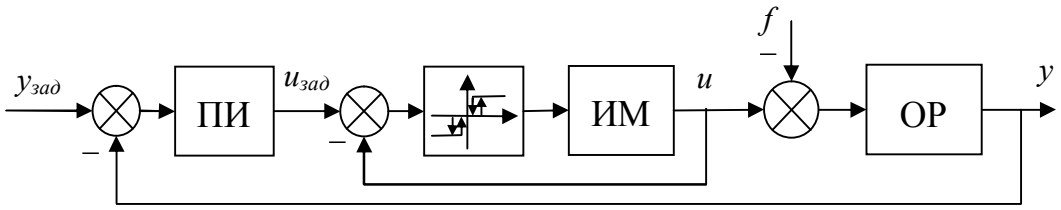


Рис. 3.85. Структура системы.

ПИ регулятор имеет настройки, рассчитанные в п. 3.1.1.1. Задача внутреннего контура – реализовать управляющий сигнал, сформированный ПИ регулятором, изменяя положение привода исполнительного механизма. Сам релейный алгоритм регулирования положения ИМ на практике может быть выполняться на ПЛК или на специальном устройстве контроля и управления исполнительным механизмом. В последнем случае ПЛК управляет этим устройством, выдавая ему задание с помощью аналогового сигнала или по промышленной сети.

При измерении положения ИМ программа ПЛК или устройства управления ИМ может контролировать работу привода: обнаруживать «заклинивание» (команда на включение есть, а положение не изменяется) и «проскальзывание» (наоборот) в промежуточном положении, достижение конечных состояний без концевых выключателей и т.д. Все это, безусловно, повышает надежность и функциональность системы автоматизации.

В нашей модели регулирование положения и контроль состояния привода будет осуществлять виртуальный ПЛК.

3.1.3.2. Моделирование системы в Simulink

Simulink-диаграмма показана на рис. 3.86. В верхней части диаграммы размещена модель «эталонной» системы на базе непрерывного регулятора, в нижней – модель исследуемой двухконтурной системы с релейным регулятором положения ИМ. Настройки блоков Relay и Relay1, совместно формирующих трехпозиционный регулятор, показаны на рис. 3.87. Ширина зоны неоднозначности (гистерезис) принята равной 1% положения ИМ. В обеих системах предусмотрено ограничение выходного сигнала ПИ-регулятора диапазоном от 0 до 100%.

В отличие от ранее рассмотренных, Simulink-диаграмма рассчитывается «решателем» (solver) с постоянным шагом интегрирования ode3 (шаг задан равным 0,01 с). Такое решение обусловлено тем, что в результате эксперимен-

тов выяснилось, что решатели с переменным шагом «впадают в ступор» при входе регулятора в режим ограничения, если выходной сигнал регулятора в дальнейшем обрабатывается релейными блоками.

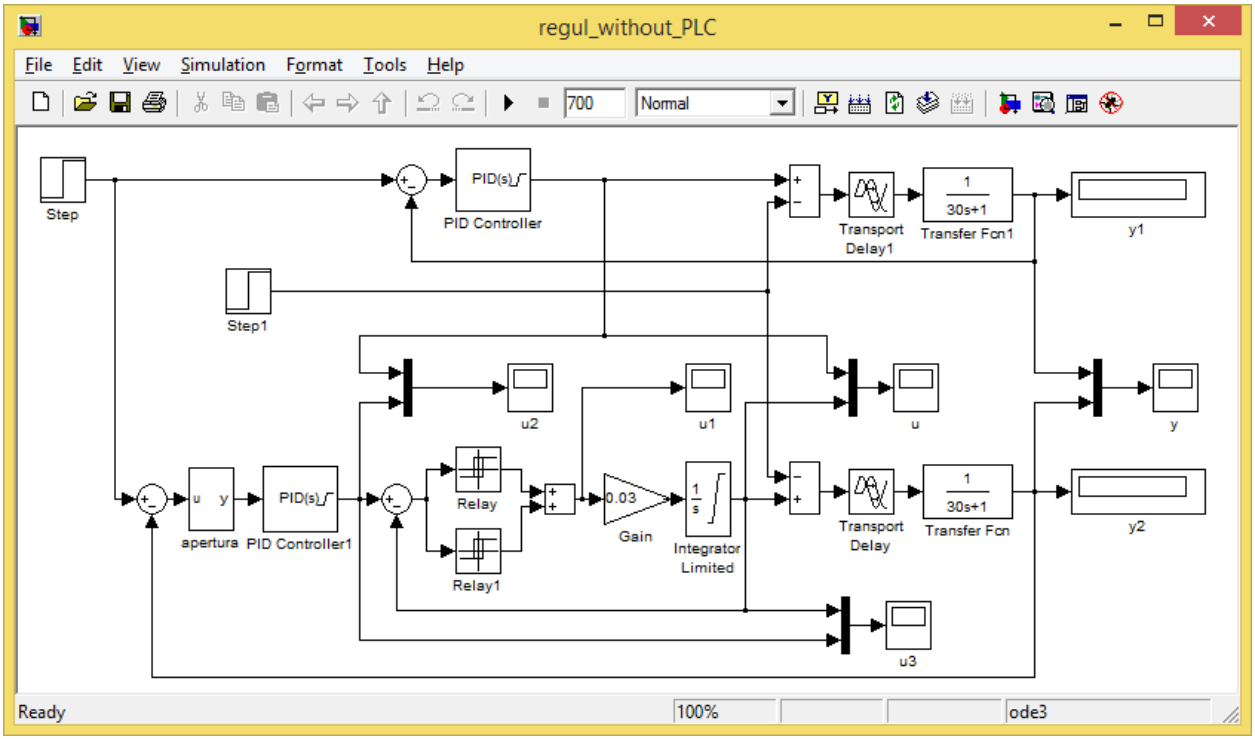


Рис. 3.86. Simulink-диаграмма.

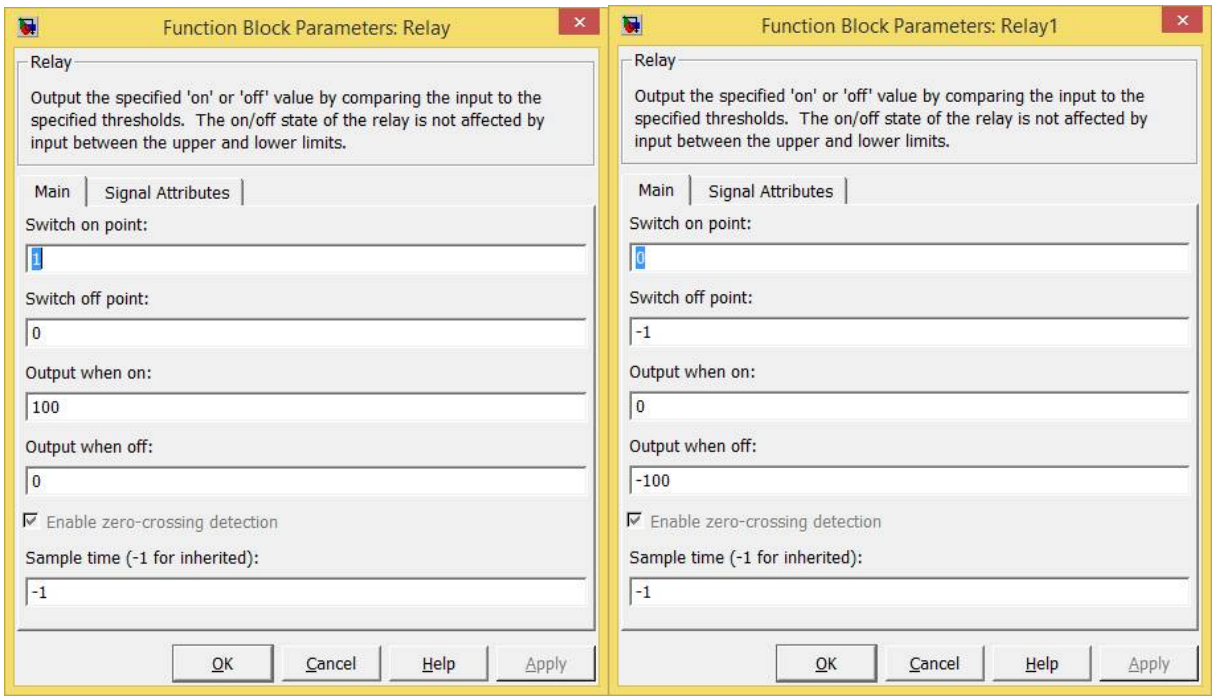


Рис. 3.87. Настройки блоков.

Все блоки Scope Simulink-диаграммы сохраняют тренды в соответствующих переменных. Ниже приведена программа построения графиков по результатам моделирования. Сами графики показаны на рис. 3.88, 3.89.

```
figure(1)
subplot(4,1,1)
L =plot(u2(:,1),u2(:,2),u2(:,1),u2(:,3)); grid
set(gca,'YLim', [0 105]);
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(4,1,2)
L =plot(u1(:,1),u1(:,2)), grid
set(gca,'YLim', [-105 105]);
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0])
subplot(4,1,3)
L =plot(u(:,1),u(:,2),u(:,1),u(:,3)); grid
set(gca,'YLim', [0 105]);
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(4,1,4)
L = plot(y(:,1),y(:,2),y(:,1),y(:,3)); grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
figure(2)
L = plot(u3(:,1),u3(:,2),u3(:,1),u3(:,3)); grid
set(gca,'YLim', [0 105]);
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
```

На системы подавались ступенчатые воздействия:

– изменения задания от 0 до 50% в момент времени $t = 0$ с;

– изменения возмущения от 0 до 20% в момент времени $t = 350$ с.

На рис. 3.88 показаны следующие графики:

– выходных сигналов ПИ-регуляторов (u2);

– импульсов, подаваемых на исполнительный механизм в исследуемой системе (u1);

– управляющих воздействий, подаваемых на объекты управления в обеих системах (u);

– выходных величин объектов (y).

В обеих системах при отработке сигнала задания регуляторы выходят на насыщение (100%), однако регулятор эталонной системы возвращается в нормальный режим гораздо быстрее. Это объясняется тем, что его выходной сигнал непосредственно воздействует на объект, тогда как выходной сигнал регулятора исследуемой системы требуется еще воспроизвести релейной следящей системой внутреннего контура. Как показано на рис. 3.89, для того, чтобы «догнать» задание, внутреннему контуру потребовалось более 30 с.

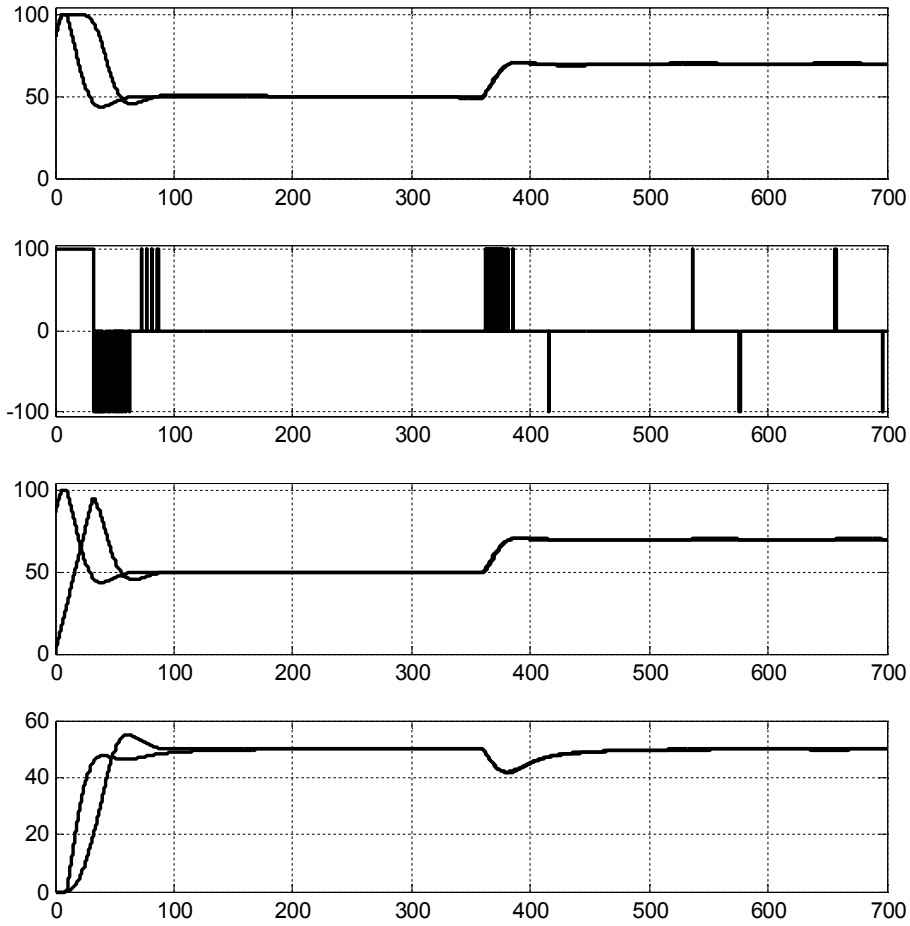


Рис. 3.88. Результаты моделирования:
записи блоков u2, u1, u, y (сверху вниз).

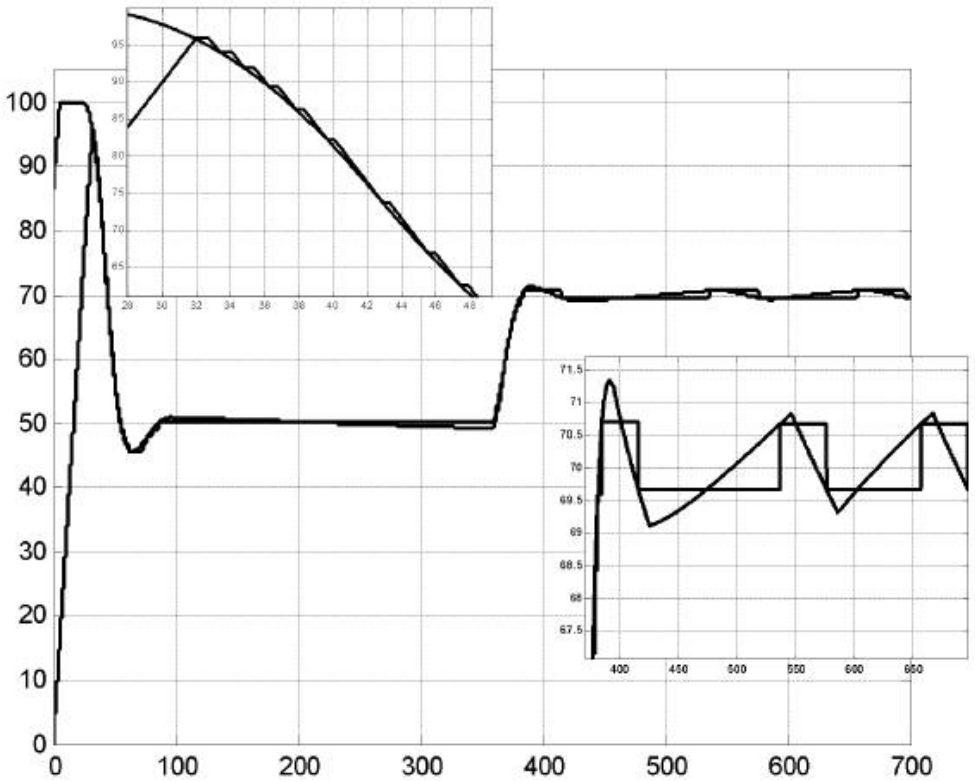


Рис. 3.89. Реализация управляющего сигнала в исследуемой системе (u3).

В дальнейшем исследуемая система ведет себя в основном подобно эталонной. При отработке возмущения, когда все переменные системы изменяются плавно, различия в поведении систем практически не видны.

Графики на рис. 3.88, 3.89 получены при «отключенной» апертуре (`delta` = 0). На увеличенном фрагменте рис. 3.89 показан квазиустановившийся режим после отработки возмущающего воздействия. В этом режиме период колебаний составляет около 120 с, в течение которых привод включается два раза (в сторону «больше» и в сторону «меньше»). В результате исследований выяснилось, что при заданных настройках «решателя» рысканье прекращается уже при `delta` = 0,2 и система приходит в настоящий установившийся режим.

3.1.3.3. Реализация алгоритмов управления на ПЛК

На рис. 3.90 показана Simulink-диаграмма для апробации алгоритмов.

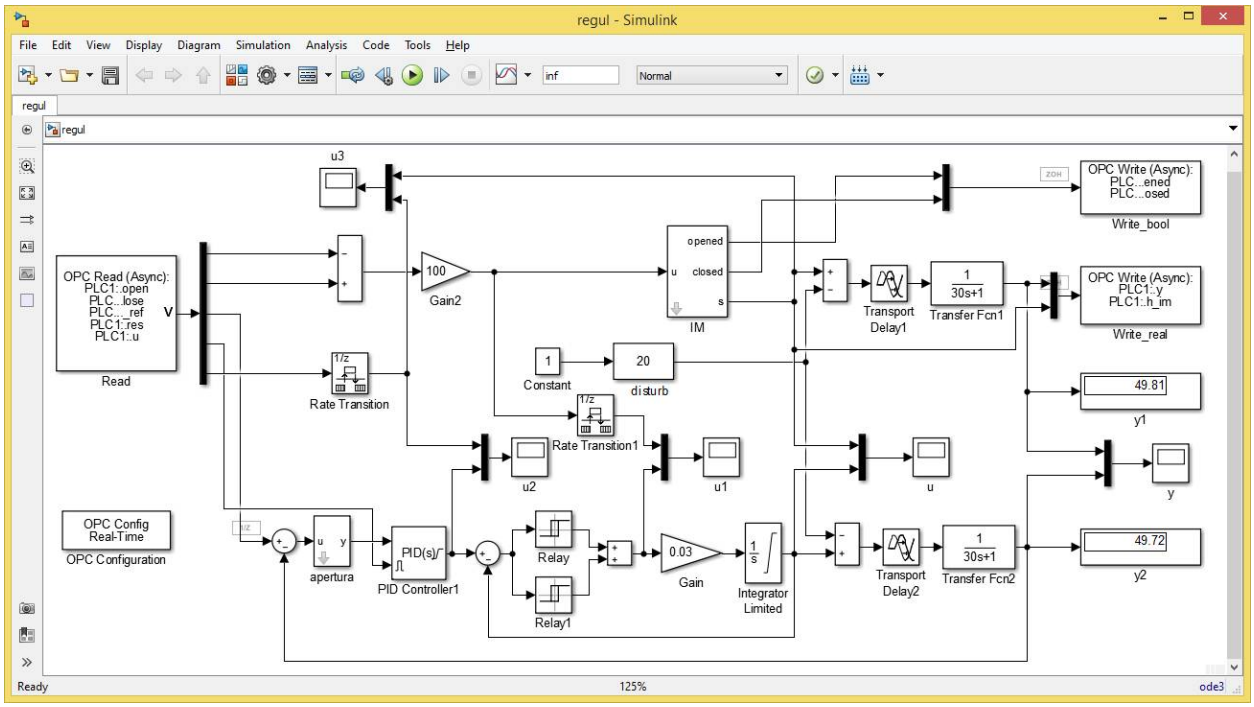


Рис. 3.90. Simulink-модель для апробации алгоритмов управления, реализуемых на ПЛК.

Как и прежде, в верхней части диаграммы размещены блоки модели объекта регулирования с исполнительным механизмом, в нижнем – модель замкнутой системы регулирования, разработанная и опробованная ранее (эталонная система). Отличия данной диаграммы от диаграммы, разработанной в п. 3.1.2.3 и показанной на рис. 3.64, состоят в следующем.

Исполнительный механизм представлен маскированной подсистемой `IM`, рис. 3.91. Подсистема формирует сигналы о достижении механизмом состояний `opened` и `closed` и о текущем его положении `s` (% хода). В окне настройки пользователь задает номинальную скорость движения исполнительного механизма (% хода в с) и вводит «аварии» привода (заклинивание `jam` и проскальзывание `slip`) непосредственно в процессе имитации.

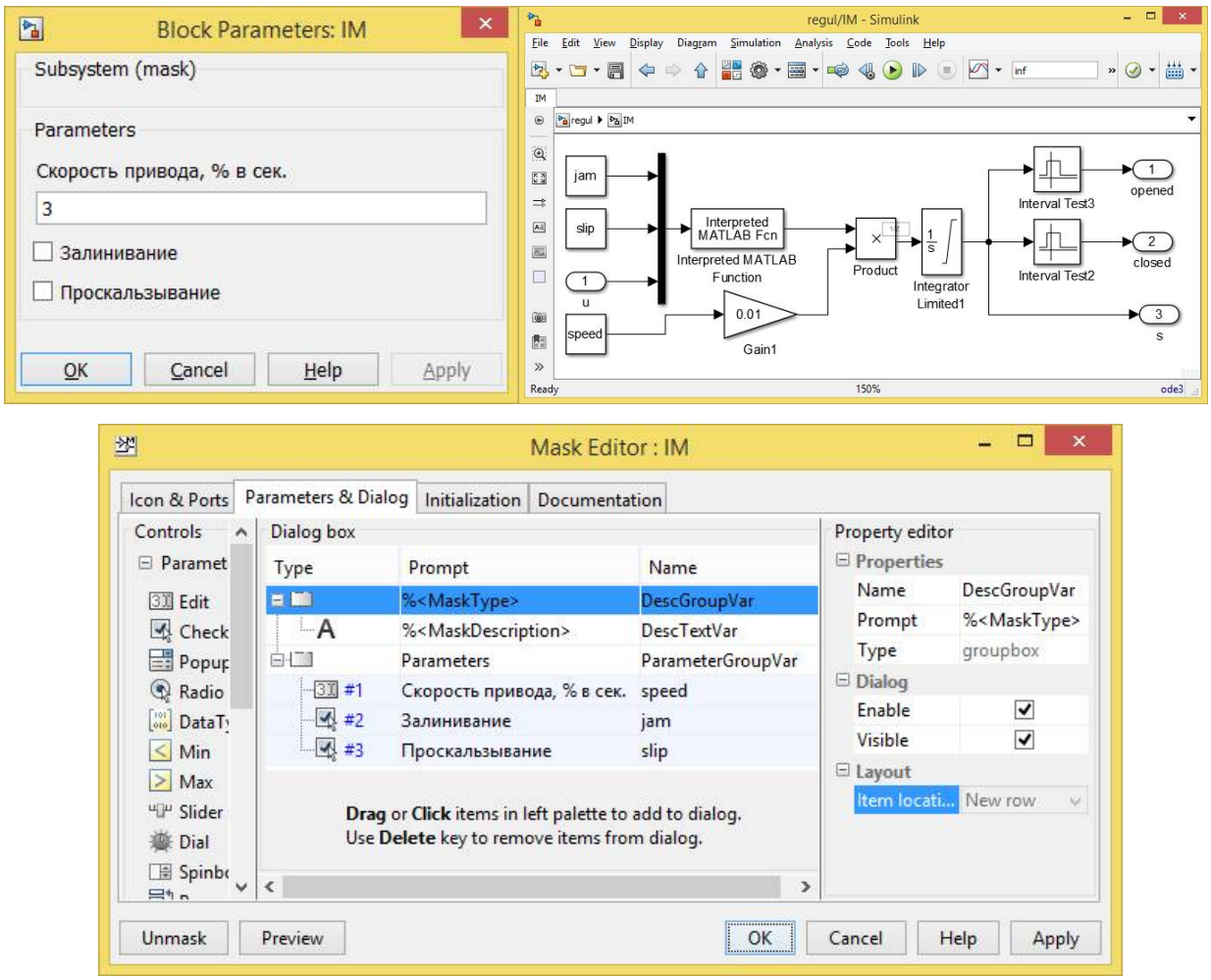


Рис. 3.91. Подсистема IM.

В нашей реализации *заклинивание* означает снижение скорости привода до 50% от номинальной при включенном приводе. *Проскальзывание* означает повышение скорости до 150% при включенном приводе, а также движение на скорости 50% от номинальной при выключенном приводе (в направлении последнего включения). Алгоритм формирования скорости реализуется интерпретируемой MATLAB-функцией `drive_fun.m`, код которой приведен ниже. Для запоминания направления последнего включения в функции задействована сохраняемая между вызовами (persistent) переменная `u_old`.

```
function y = drive_fun(x)
persistent u_old
if ~ exist('u_old')
    u_old = 0;
end
jam = x(1);
slip = x(2);
u = x(3);
    if u~=0, u_old = u; end
    if jam
        y = 0.5*u; return;
    elseif slip
        if u == 0
            y = 0.5*u_old;
```

```
            else
                y = 1.5*u;
            end
        else
            y = u;
        end
    end
```

Обмен данными с OPC-сервером организован практически так же, как это было сделано в п. 3.1.2.3. Единственная дополнительная переменная (положением ИМ, h_im) добавлена во второй блок OPC Write – «Write_real».

Программа ПЛК получена своеобразным «скрещиванием» программ, разработанных в п. 3.1.1.3 и п. 3.1.2.3. В состав POU входят:

функция APERTURA (см. п. 3.1.2.3);

функциональный блок VALVE_CONTROL – релейный регулятор положения исполнительного механизма;

программа CONTROL, реализующая вычисление управляющего воздействия и его реализацию с помощью экземпляров функциональных блоков PID и VALVE_CONTROL;

программа обслуживания операторской панели PANEL;

главная программа PLC_PRG, занимающаяся обменом информации с Simulink-диаграмой и SCADA-системой, управление режимами системы и вызовом программ CONTROL и PANEL;

функциональный блок VA, обнаруживающий аварии привода (заклинивание и проскальзывание);

программа VALVE_ALARM, вызывающая экземпляр функционального блока VA.

Программа VALVE_ALARM работает независимо от PLC_PRG и вызывается на исполнение «по расписанию» (в нашей реализации – через каждые 50 мс.). Таким образом, в целом предусмотрено выполнение контроллером двух задач:

1) «Main» со свободным вызовом программы PLC_PRG;

2) «VA» с вызовом программы VALVE_ALARM каждые 50 мс.

Список глобальных переменных программы ПЛК приведен ниже. Из списка, сформированного в п. 3.1.2.3, исключены переменные, отвечающие за передачу команд «включить привод на открытие/закрытие вентиля» от операторов панели и SCADA-системы в контроллер. Взамен «реабилитированы» переменные u_manual_from_OP и u_manual_from_SCADA, посредством которых операторы будут устанавливать *желаемое положение* вентиля. Кроме того, в список добавлены переменные для передачи информации о фактическом положении, заклинивании и проскальзывании исполнительного механизма.

```
VAR_GLOBAL

(*Входы контроллера*)
    y:REAL; (*регулируемая величина*)
    opened, closed:BOOL; (*сигналы конечных выключателей*)
    h_im:REAL; (*положение исполнительного механизма*)
    reset_process:BOOL:=FALSE; (*сброс системы после аварии –
```

```
                                        аппаратная кнопка без фиксации*)
(*Выходы контроллера*)
     open:BOOL:=FALSE; (*сигнал управления "Открыть"*)
     close:BOOL:=FALSE; (*сигнал управления "Закрыть"*)
     no_HMI:BOOL:=FALSE; (*потеряна связь с панелью и SCADA -
                             сигнализация*)
     stop:BOOL:=FALSE; (*полная остановка системы*)

(*"Внутренние" переменные*)
     y_ref:REAL; (*задание регулятора, также  подается в Simulink-
                   модель*)
     u_manual:REAL:=0; (*сигнал ручного управления*)
     man:BOOL:=FALSE; (*сигнал перевода в режим ручного управле-
ния*)
     res:BOOL:=FALSE; (*сброс регулятора, также  подается в
                              Simulink-модель*)
     block_panel:BOOL:=FALSE; (*блокирование панели*)
     u:REAL; (*выход ПИ-регулятора*)
     jam,slip:BOOL:=FALSE; (*заклинивание и проскальзывание ИМ*)

(*Переменные обмена с панелью оператора*)

     (*входы:*)
     y_ref_from_OP:REAL:=0; (*задание с панели*)
     y_ref_update_from_OP:BOOL:=FALSE; (*обновить задание с
                                          панели*)
     u_manual_from_OP:REAL:=0; (*сигнал ручного управления с
                                  панели*)
     man_from_OP:BOOL:=FALSE; (*сигнал перехода на ручной режим с
                                панели*)
     res_from_OP:BOOL; (*сброс регулятора с панели*)
     tic_tac_from_OP:BOOL; (*тестовый сигнал наличия связи с
                             панели*)
     (*выходы:*)
     y_ref_to_OP, y_to_OP, u_to_OP:REAL; (*задание, выход и выход
                                           на панель*)
     res_to_OP:BOOL; (*сброс регулятора на панель*)
     state_to_OP:SINT; (*режим управления на панель*)
     block_panel_to_OP:BOOL; (*панель заблокирована на панель*)
     SCADA_link_to_OP:BOOL:=TRUE; (*связь со SCADA на панель*)
     err_gt_10_to_OP, err_lt_m10_to_OP, norma_to_OP:BOOL; (*сиг-
                                         налы контроля на панель*)
     tic_tac_to_OP:BOOL; (*тестовый сигнал наличия связи на
                           панель*)
     opening_to_OP:BOOL; (*привод включен на открытие вентиля*)
     closing_to_OP:BOOL; (*привод включен на закрытие вентиля*)
     opened_to_OP:BOOL; (*вентиль открыт*)
     closed_to_OP:BOOL; (*вентиль закрыт*)
     h_im_to_OP:REAL; (*положение исполнительного механизма на
                        панель*)
     jam_to_OP,slip_to_OP:BOOL; (*заклинивание и проскальзывание
                                  ИМ - на панель*)
```

```
(*Переменные обмена со SCADA-системой*)
     (*входы:*)
     y_ref_from_SCADA:REAL:=0; (*задание от SCADA*)
     y_ref_update_from_SCADA:BOOL:=FALSE; (*обновить задание от
                                                SCADA*)
     u_manual_from_SCADA:REAL:=0; (*сигнал дистанционного
                                      управления от SCADA*)
     dist_from_SCADA:BOOL:=FALSE; (*переход на дистанционное
                                      управление от SCADA*)
     block_panel_from_SCADA: BOOL:=FALSE; (*заблокировать панель
                                                от SCADA*)
     res_from_SCADA:BOOL:=FALSE; (*сброс регулятора от SCADA*)
     tic_tac_from_SCADA:BOOL; (*тестовый сигнал наличия связи от
                                  SCADA*)
     (*выходы:*)
     y_to_SCADA:REAL; (*выходной сигнал в SCADA*)
     y_ref_to_SCADA:REAL; (*задание в SCADA*)
     u_to_SCADA:REAL; (*управление в SCADA*)
     res_to_SCADA:BOOL; (*сброс регулятора в SCADA*)
     state_to_SCADA:SINT; (*режим управления в SCADA*)
     block_panel_to_SCADA: BOOL:=TRUE; (*панель заблокирована  в
                                           SCADA*)
     OP_link_to_SCADA:BOOL:=TRUE; (*связь с панелью в SCADA*)
     tic_tac_to_SCADA:BOOL; (*тестовый сигнал наличия связи в
                                SCADA*)
     opening_to_SCADA:BOOL; (*привод включен на открытие вентиля*)
     closing_to_SCADA:BOOL; (*привод включен на закрытие вентиля*)
     opened_to_SCADA:BOOL; (*вентиль открыт*)
     closed_to_SCADA:BOOL; (*вентиль закрыт*)
     h_im_to_SCADA:REAL; (*положение исполнительного механизма
                              в SCADA *)
     jam_to_SCADA,slip_to_SCADA:BOOL; (*заклинивание и проскаль-
                                          зывание ИМ – в SCADA*)
END_VAR
```

Функциональный блок `VALVE_CONTROL` написан на языке ST:

```
FUNCTION_BLOCK VALVE_CONTROL
VAR_INPUT
     yzad,y:REAL; (*заданное и фактическое положение привода*)
     turn_on:REAL; (*порог срабатывания*)
     turn_off:REAL; (*порог "отпускания"*)
     opened:BOOL; (*сигнал концевого выключателя "открыто"*)
     closed:BOOL; (*сигнал концевого выключателя "закрыто"*)
END_VAR
VAR_OUTPUT
     open:BOOL:=FALSE; (*включить привод на открытие вентиля*)
     close:BOOL:=FALSE; (*включить привод на закрытие вентиля*)
END_VAR
VAR
     error:REAL; (*ошибка регулирования положения*)
END_VAR
-----------------------------------------------------------------
```

```
error:=yzad - y;
IF close THEN
     IF error > -turn_off THEN
          close:=FALSE;
     END_IF
ELSIF NOT close AND NOT open THEN
     IF error < -turn_on THEN
          close:=NOT closed;
     ELSIF error > turn_on THEN
          open:=NOT opened;
     END_IF
ELSIF open THEN
     IF error < turn_off THEN
          open:=FALSE;
     END_IF
END_IF
```

Функциональный блок реализует «классический» алгоритм трехпозиционного регулирования с зоной неоднозначности (гистерезисом).

Программа `CONTROL` (язык CFC) показана на рис. 3.92.

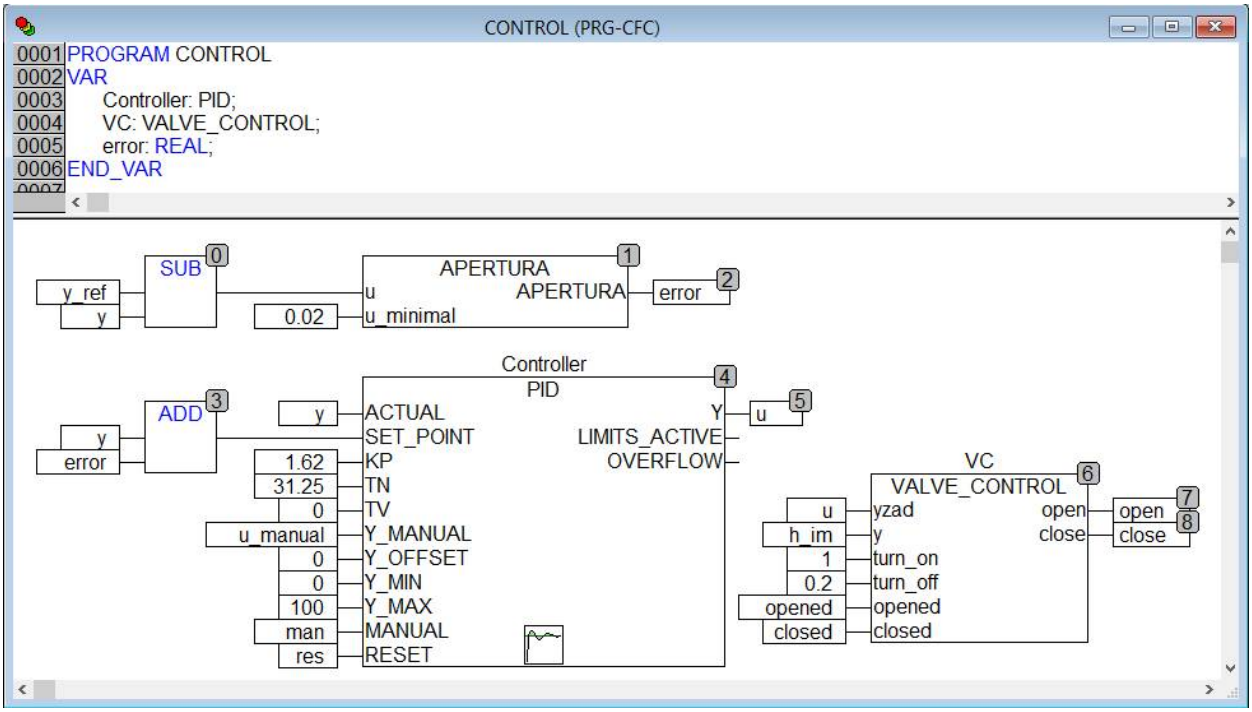


Рис. 3.92. Программа `CONTROL`.

В отличие от блока PID Simulink-диаграммы, функциональный блок `PID` библиотеки CoDeSys требует подачи на свои входы двух отдельных сигналов (регулируемой величины и задания), а не одного (ошибки регулирования). Нетрудно догадаться, что ошибка вычисляется уже «внутри» регулятора. Однако именно сигнал ошибки должен быть обработан функцией `APERTURA` и таким образом, эту функцию нужно «внедрить» в функциональный блок `PID` (как это, кстати, и сделано в библиотеках некоторых контроллеров).

Из-за нежелания модифицировать библиотечные коды, пришлось «пойти на хитрость».

Фактическая ошибка вычисляется отдельно и далее обрабатывается функцией APERTURA.

На вход ACTUAL регулятора, как и положено, подается значение регулируемой величины, а на вход SET_POINT – не само задание, его «восстановленное» значение. «Восстановление» происходит путем сложения регулируемой величины с уже обработанной ошибкой. Если фактическая ошибка достаточно велика и APERTURA «не сработала» (повторила ее на своем выходе), на вход SET_POINT будет подано правильно вычисленное значение сигнала задания. «Внутри» регулятора ошибка будет вычислена повторно. Если же фактическая ошибка мала и APERTURA выдала ноль, на вход SET_POINT будет подано значение регулируемой величины. «Внутри» блока нулевая ошибка будет рассчитана вновь и регулятор будет «считать», что задание выполнено с идеальной точностью.

Программа PANEL подверглась незначительным изменениям. Секция объявления переменных (изменения выделены жирным шрифтом):

```
PROGRAM PANEL
VAR
     (*ОТОБРАЖЕНИЕ*)
     y_ref, y, u:REAL; (*задание, выход и управление*)
     err_gt_10, err_lt_m10, norma:BOOL; (*сигналы контроля *)
     block_panel:BOOL; (*блокировать панель - на панель*)
     SCADA_link:BOOL:=TRUE; (*связь со SCADA на панель*)
     state: STRING; (*режим управления*)
     res:BOOL; (*сброс регулятора*)
     opening:BOOL; (*привод включен на открытие вентиля*)
     closing:BOOL; (*привод включен на закрытие вентиля*)
     moving: INT; (*для отображения движения привода на тренде*)
     opened:BOOL; (*вентиль открыт*)
     closed:BOOL; (*вентиль закрыт*)
     h_im:REAL; (*положение исполнительного механизма*)
     (*УПРАВЛЕНИЕ И ВВОД*)
     y_ref_local:REAL:=0; (*локальное задание*)
     y_ref_update:BOOL:=FALSE; (*обновить задание*)
     man:BOOL:=FALSE; (*сигнал перехода на ручной режим*)
     do_reset:BOOL:=FALSE; (*сброс регулятора*)
     ERROR:BOOL:=FALSE; (*отказ панели*)
     jam:BOOL; (*заклинивание ИМ*)
     slip:BOOL; (*проскальзывание ИМ*)
END_VAR
```

Главное окно панели с изменившимися привязками – на рис. 3.93.

На втором тренде отображаются графики изменения заданного (u) и фактического (h_im) положений вентиля и импульсы, подаваемые на привод исполнительного механизма (moving). С помощью ползунка оператор задает положение механизма в ручном режиме управления.

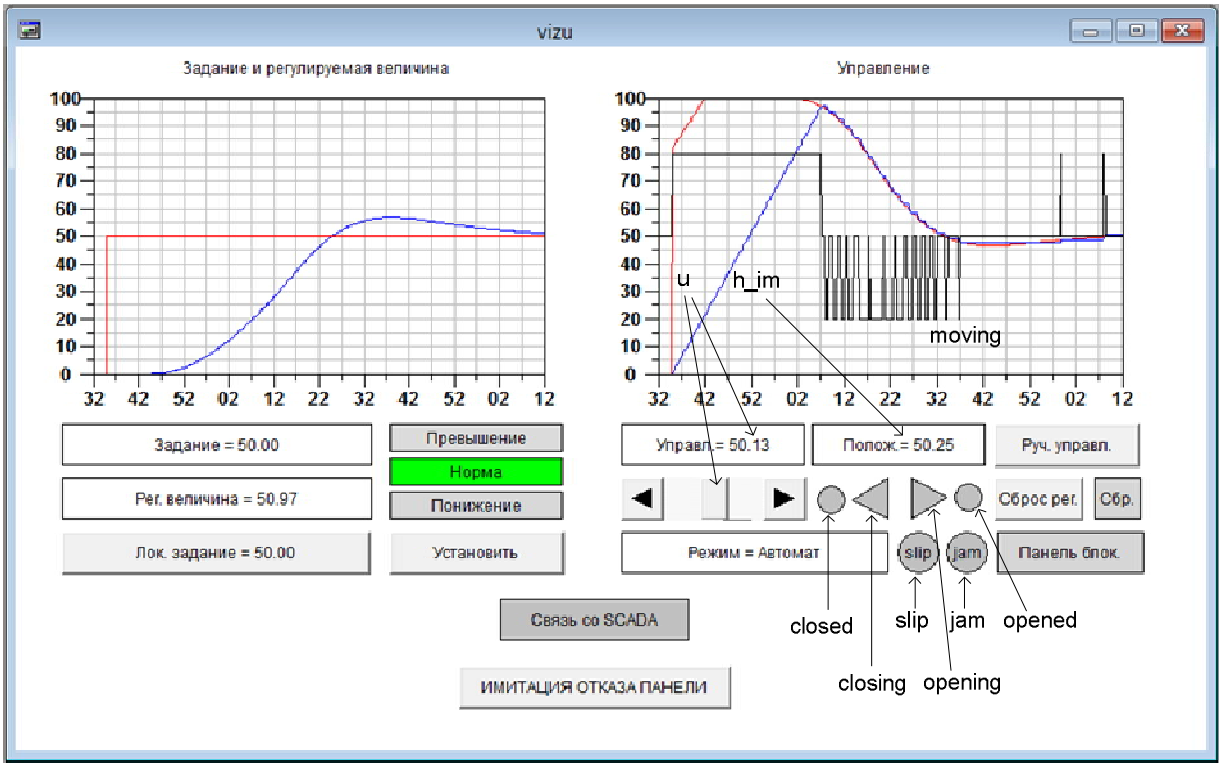


Рис. 3.93. Главное окно панели регулятора с дополнительными привязками.

Код программы PANEL (жирным выделены изменения):

```
IF NOT ERROR THEN
    y_ref:= y_ref_to_OP;
    y:= y_to_OP;
    err_gt_10:=err_gt_10_to_OP;
    err_lt_m10:=err_lt_m10_to_OP;
    norma:=norma_to_OP;
    block_panel:=block_panel_to_OP;
    SCADA_link:=SCADA_link_to_OP;
    res:=res_to_OP;
    opening:=opening_to_OP;
    closing:=closing_to_OP;
    opened:=opened_to_OP;
    closed:=closed_to_OP;
    moving:=30*BOOL_TO_INT(opening)-30*BOOL_TO_INT(closing)+ 50;
    h_im:=h_im_to_OP;
    jam:=jam_to_OP;
    slip:=slip_to_OP;
    CASE state_to_OP OF
    -1:
          state:='Авария';
          u:=u_to_OP;
     0:
          state:='Ручной';
     1:
          state:='Автомат';
          u:=u_to_OP;
     2:
          state:='Дистанц.';
```

```
          u:=u_to_OP;
     END_CASE
     y_ref_from_OP:=y_ref_local;
     y_ref_update_from_OP:=y_ref_update;
     u_manual_from_OP:=u;
     man_from_OP:=man;
     res_from_OP:=do_reset;
     tic_tac_from_OP:=NOT tic_tac_to_OP;
ELSE
     state:='Ошибка связи';
END_IF
```

Для наглядности отображения импульсов принято, что переменная `moving` может принимать значения -20, 50, 80 в зависимости от наличия сигналов `opening` и `closing`.

Программа `PLC_PRG` практически полностью повторяет программу из 3.1.1.3 (изменения выделены жирным):

```
PROGRAM PLC_PRG
VAR
     (*режим управления*)
     state:SINT:=0; (*-1 - авария,1 - авт., 2 - дист.*)
     (*таймеры связи*)
     SCADA_EXCHANGE_WATCHDOG_1, SCADA_EXCHANGE_WATCHDOG_0 : TON;
     OP_EXCHANGE_WATCHDOG_1, OP_EXCHANGE_WATCHDOG_0 : TON;
     (*таймер регулирования*)
     STOP_CONTROL_TIMER:TON;
     (*флаги ошибок связи*)
     SCADA_FAIL:BOOL:=FALSE;
     OP_FAIL:BOOL:=FALSE;
END_VAR
-------------------------------------------------------------------
(*Запись по интерфейсам*)
y_to_SCADA:=y_to_OP:=y;
y_ref_to_SCADA:=y_ref_to_OP:=y_ref;
u_to_SCADA:=u_to_OP:=u;
block_panel_to_SCADA:= block_panel_to_OP:= block_panel;
state_to_SCADA:=state_to_OP:=state;
res_to_SCADA:=res_to_OP:=res;
opening_to_SCADA:=opening_to_OP:=open;
closing_to_SCADA:=closing_to_OP:=close;
opened_to_SCADA:=opened_to_OP:=opened;
closed_to_SCADA:=closed_to_OP:=closed;
h_im_to_SCADA:=h_im_to_OP:=h_im;
jam_to_SCADA:=jam_to_OP:=jam;
slip_to_SCADA:=slip_to_OP:=slip;

(*Проверка связи со SCADA и панелью*)
(*по умолчанию - все нормально*)
SCADA_link_to_OP:=OP_link_to_SCADA:=TRUE;
SCADA_FAIL:=OP_FAIL:=FALSE;
(*ретрансляция tic_tac*)
tic_tac_to_SCADA:= tic_tac_from_SCADA;
```

```
tic_tac_to_OP:= tic_tac_from_OP;
(*запуск таймеров*)
SCADA_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_SCADA, PT:=t#5s);
SCADA_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_SCADA, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_OP, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_OP, PT:=t#5s);
(*установка флагов*)
SCADA_FAIL:= SCADA_EXCHANGE_WATCHDOG_1.Q OR
    SCADA_EXCHANGE_WATCHDOG_0.Q;
OP_FAIL:= OP_EXCHANGE_WATCHDOG_1.Q OR OP_EXCHANGE_WATCHDOG_0.Q;
no_HMI:= SCADA_FAIL AND OP_FAIL;
block_panel:=block_panel_from_SCADA;

(*Обработка ошибок связи*)
IF  SCADA_FAIL THEN
     block_panel:=OP_FAIL;
     SCADA_link_to_OP:=FALSE;
     y_ref_from_SCADA:=y_ref;
     y_ref_update_from_SCADA:=FALSE;
END_IF
IF OP_FAIL THEN
     block_panel:=TRUE;
     OP_link_to_SCADA:=FALSE;
     y_ref_from_OP:=y_ref;
     y_ref_update_from_OP:=FALSE;
END_IF

(*Обновление задания*)
IF y_ref_update_from_SCADA AND NOT SCADA_FAIL THEN
     y_ref:=y_ref_from_SCADA;
END_IF
IF y_ref_update_from_OP AND NOT block_panel THEN
     y_ref:=y_ref_from_OP;
END_IF

stop:=FALSE; (*по умолчанию оборудование работает*)

(*Формирование сигнала сброса регулятора*)
res:= (res_from_SCADA AND NOT SCADA_FAIL) OR
      (res_from_OP AND NOT block_panel);

(*Формирование сигналов контроля для панели*)
IF y_ref - y  < -10 THEN
     err_gt_10_to_OP:=TRUE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=FALSE;
ELSIF y_ref - y  > 10 THEN
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=TRUE;
     norma_to_OP:=FALSE;
ELSE
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=FALSE;
```

```
     norma_to_OP:=TRUE;
END_IF

(*Режимы управления*)
CASE state OF
-1: (*авария*)
          y_ref:=0;
          u_manual:=0;
          res:=TRUE;
          stop:=TRUE;
          IF reset_process THEN
               state:=0;
          END_IF
0: (*ручное управление*)
     u_manual:=u_manual_from_OP;
     man:=TRUE;
     (*переходы в другие режимы*)
     IF block_panel OR NOT man_from_OP OR OP_FAIL THEN
          IF dist_from_SCADA  AND NOT SCADA_FAIL THEN
               state:=2;
          ELSE
               state:=1;
          END_IF
     END_IF
1: (*автоматическое управление*)
     u_manual:=u;
     man:=FALSE;
(*переходы в другие режимы*)
     IF man_from_OP AND NOT block_panel THEN
          state:=0;
     ELSIF dist_from_SCADA  AND NOT SCADA_FAIL THEN
          state:=2;
     END_IF
     (*Если нет связи со SCADA и панелью, установка "безопасного"
уровня задания*)
     IF NO_HMI THEN
          y_ref:=50;
     END_IF
     (*Если связь с HMI утеряна и процесс в течение 1 мин.
 не удается вернуть в норму - авария*)
     STOP_CONTROL_TIMER(IN:=no_HMI AND
     (err_gt_10_to_OP OR err_lt_m10_to_OP), PT:=t#1m);
     IF STOP_CONTROL_TIMER.Q THEN
          state:=-1;
     END_IF
2: (*дистанционное управление*)
     u_manual:=u_manual_from_SCADA;
     man:=TRUE;
(*переходы в другие режимы*)
     IF (man_from_OP AND NOT block_panel)  THEN
          state:=0;
     ELSIF NOT dist_from_SCADA OR SCADA_FAIL THEN
          state:=1;
```

```
     END_IF
END_CASE
(*Регулятор*)
CONTROL;
(*Панель*)
PANEL;
```

Функциональный блок `VA` предназначен для обнаружения аварий привода вентиля (заклинивания и проскальзывания). Код блока:

```
FUNCTION_BLOCK VA
VAR_INPUT
     speed:REAL; (*скорость привода (% в сек.)*)
     start_stop_time: TIME; (*продолжительность пуска/останова*)
     control_time:TIME; (*период контроля*)
     jam_delta: REAL; (*ошибка в % хода, при которой фиксируется
                    заклинивание *)
     slip_delta: REAL; (*ошибка в % хода, при которой фиксируется
                    проскальзывание *)
END_VAR
VAR_OUTPUT
     slip: BOOL; (*проскальзывание*)
     jam: BOOL; (*заклинивание*)
END_VAR
VAR
     state:BYTE:=0; (*0 - привод выключен, 1 - включен *)
     integ: INTEGRAL; (*интегратор*)
     dh_im1: REAL; (*путь, "проделанный" интегратором за время
                     control_time*)
     dh_im2: REAL; (*путь "проделанный" приводом за время
                     control_time*)
     h_im_prev: REAL:=-1; (*предыдущее положение исполнительного
                           механизма*)
     timer: TON; (*вспомогательный таймер*)
     control_timer:TON; (*контрольный таймер*)
END_VAR
-----------------------------------------------------------------

CASE state OF
0: (*привод выключен*)
     IF open OR close THEN
          timer(IN:=TRUE, PT:=start_stop_time);
          IF timer.Q THEN
               state:=1;
               timer(IN:=FALSE);
          END_IF
     END_IF
1: (* привод включен*)
     integ(IN:=speed, TM:=50, RESET:=FALSE, out=>dh_im1);
     IF NOT open AND NOT close THEN
          timer(IN:=TRUE, PT:=start_stop_time);
          IF timer.Q THEN
               state:=0;
               timer(IN:=FALSE);
```

```
          END_IF
     END_IF
END_CASE

IF h_im_prev = - 1 THEN
     h_im_prev:=h_im;
END_IF

IF h_im > h_im_prev THEN
     dh_im2:=dh_im2 +  h_im - h_im_prev;
ELSE
     dh_im2:=dh_im2 +  h_im_prev - h_im;
END_IF
h_im_prev:=h_im;

control_timer(IN:=TRUE, PT:=control_time);
IF control_timer.Q THEN
     IF dh_im1 - dh_im2 >jam_delta THEN
          jam:=TRUE;
          slip:=FALSE;
     ELSIF  dh_im2 - dh_im1 >slip_delta THEN
          jam:=FALSE;
          slip:=TRUE;
     ELSE
          IF dh_im1 > 2*jam_delta THEN
               jam:=FALSE;
          END_IF
          IF dh_im2 > 2*slip_delta THEN
               slip:=FALSE;
          END_IF
     END_IF
     control_timer(IN:=FALSE);
     integ(RESET:=TRUE, out=>dh_im1);
     dh_im2:=0;
END_IF
```

Алгоритм обнаружения аварий основан на сравнении абсолютных значений (без учета направления движения) рассчитанного и реального перемещений рабочего органа исполнительного механизма. Сравнения происходят через интервал времени, заданный входным параметром `control_time`. Расчет перемещения производится интегратором, входным параметром которого является «номинальная» скорость перемещения рабочего органа `speed`.

Алгоритм может находиться в двух состояниях: «привод выключен» и «привод включен». В первом состоянии интегратор не вызывается и переменная `dh_im1`, в которую записывается результат интегрирования, не изменяется. Во втором состоянии интегратор вызывается, и переменная `dh_im1` увеличивается. Переход между состояниями производится по результатам анализа сигналов (глобальных переменных) `open` и `close` с временной задержкой, заданной входным параметром `start_stop_time`.

Движение привода непрерывно контролируется путем сравнения текущего и предыдущего его положений. Если эти положения отличны, то переменная `dh_im2` увеличивается на абсолютное значения приращения.

При срабатывании таймера `control_timer` производится сравнение переменных `dh_im1` и `dh_im2`. Если привод отстал от интегратора, на величину большую, чем `jam_delta`, фиксируется авария «заклинивание». Если интегратор отстал от привода на величину, большую, чем `slip_delta`, фиксируется авария «проскальзывание». Сброс аварий происходит только в случае, если за время `control_time` привод или интегратор проделали «путь» более `2*jam_delta` или `2*slip_delta` соответственно и критичных отставаний зафиксировано не было, т.е. привод действительно «подтвердил» свою работоспособность, а не просто, например, простоял «на месте» в отсутствие приказов. После проверки интегратор и таймер сбрасываются, а переменные `dh_im1` и `dh_im2` устанавливаются в нуль.

Программа `VALVE_ALARM`, выполняемая циклически, вызывает экземпляр функционального блока `VA` (рис. 3.94).

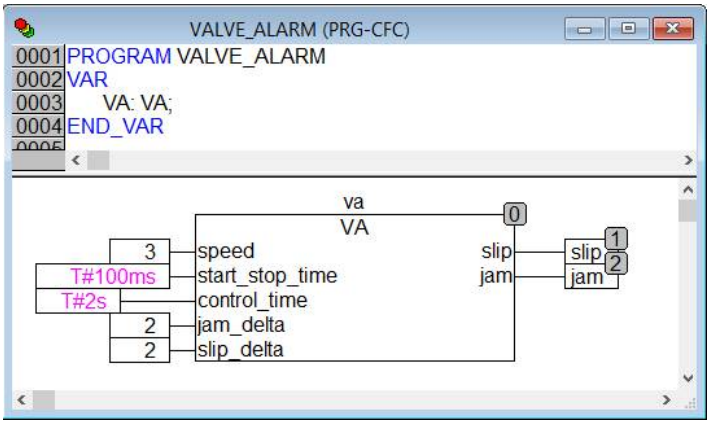


Рис. 3.94. Программа `VALVE_ALARM`.

3.1.3.4. Модернизация SCADA-системы

Рассмотрим изменения, сделанные в SCADA-системе. На рис. 3.95 показаны «Источники/Приемники».

В список «источников» из п. 3.1.2.4 внесены OPC-компоненты, предназначенные для получения информации о положении, заклинивании и проскальзывании исполнительного механизма. Из «приемников», наоборот, исключены компоненты, ранее используемые для дистанционного включения привода. Кроме того, на свое место вернулся компонент `u_manual_to_PLC`, посредством которого контроллеру будет передаваться желаемое положение привода в дистанционном режиме. Таким образом, список «приемников» принял «оригинальный» вид из п. 3.1.1.4.

Соответствующие изменения внесены в список каналов (рис. 3.96).

Канал «Положение ИМ» архивируется в СПАД №1.

На рис. 3.97 показан экран визуализации в действии, запущенный под управлением профайлера Trace Mode.

На рис. 3.98 приведен список аргументов экрана. «Новые» аргументы выделены контуром. На самом деле действительно новыми являются аргументы «Положение_ИМ», «jam» и «slip», все остальные нам знакомы еще по п. 3.1.1.4.

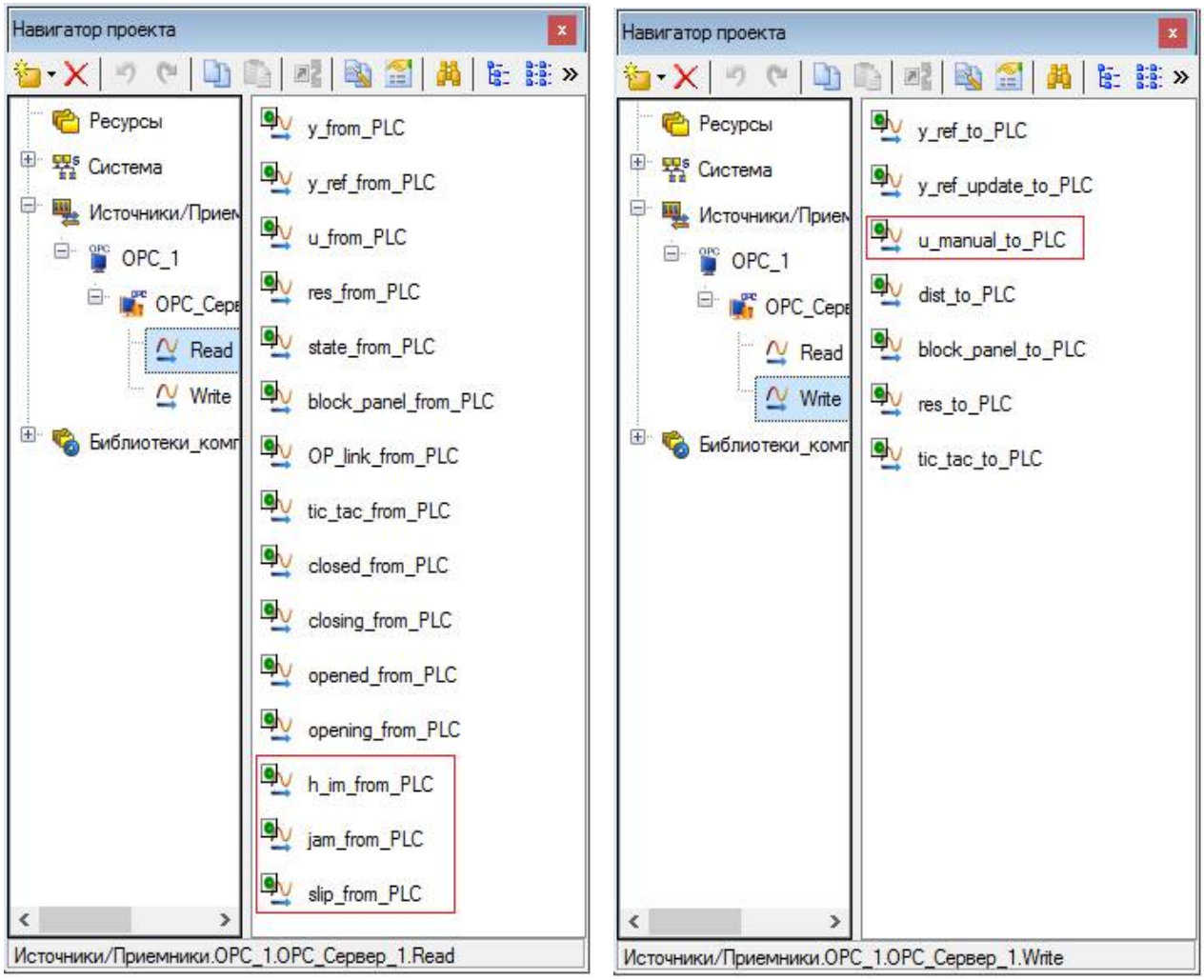


Рис. 3.95. «Источники/Приемники».

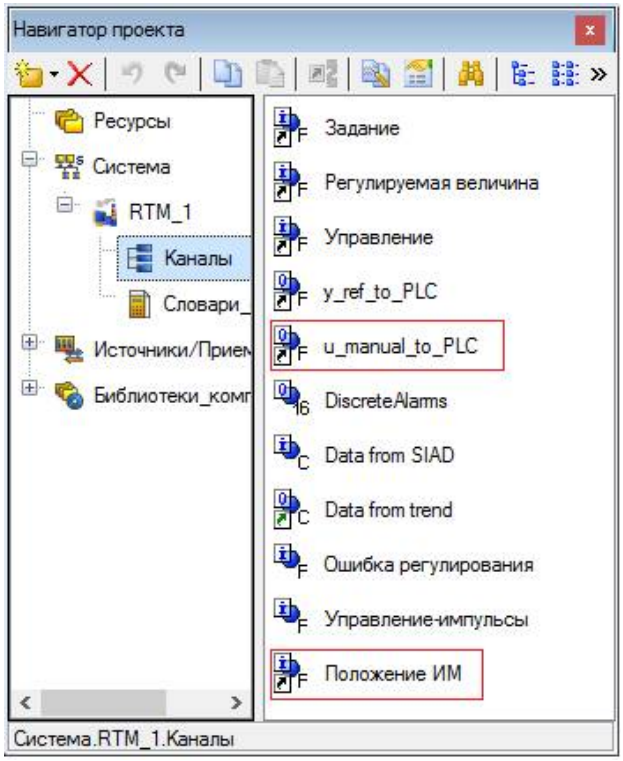


Рис. 3.96. Каналы.

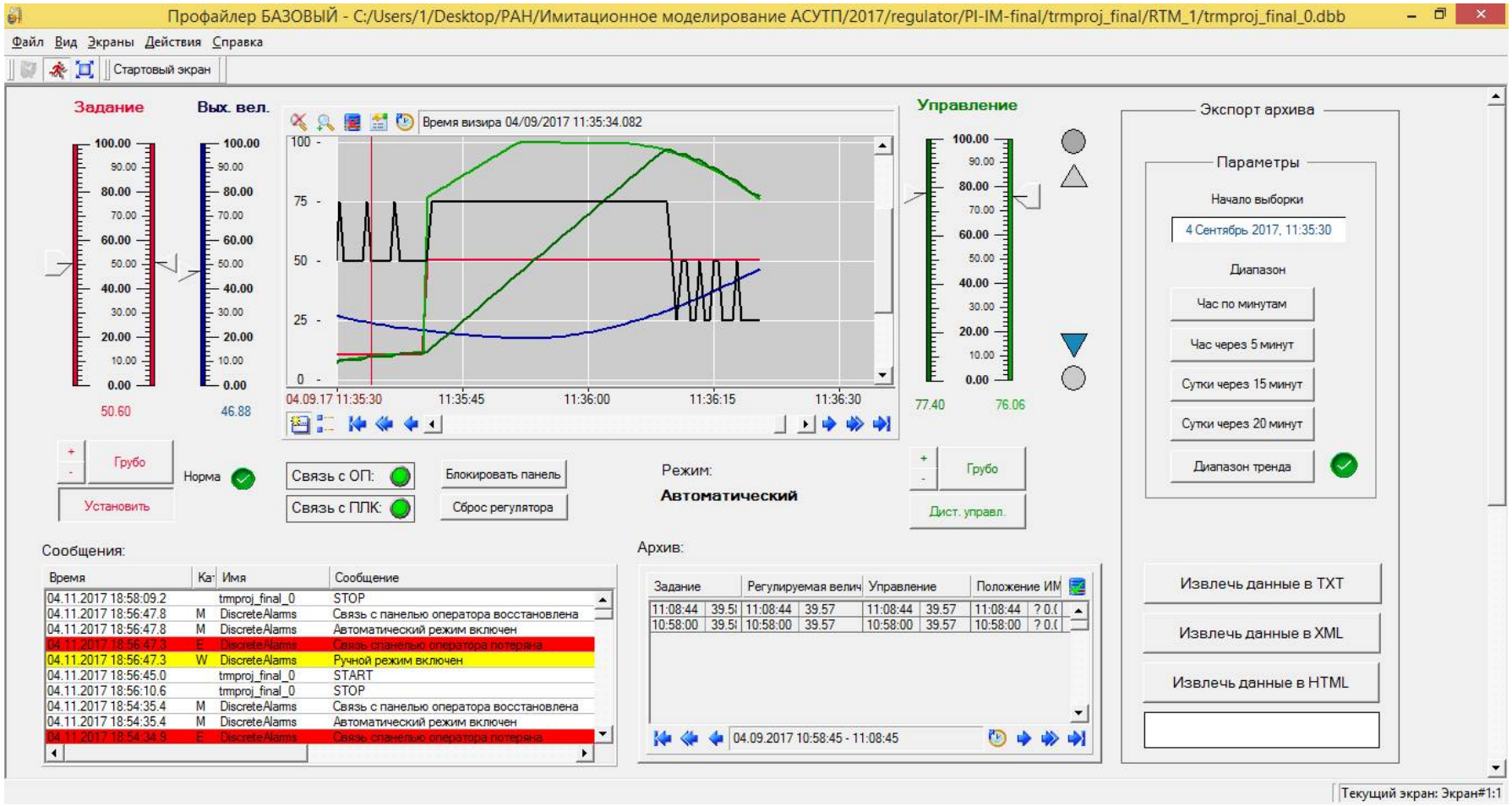
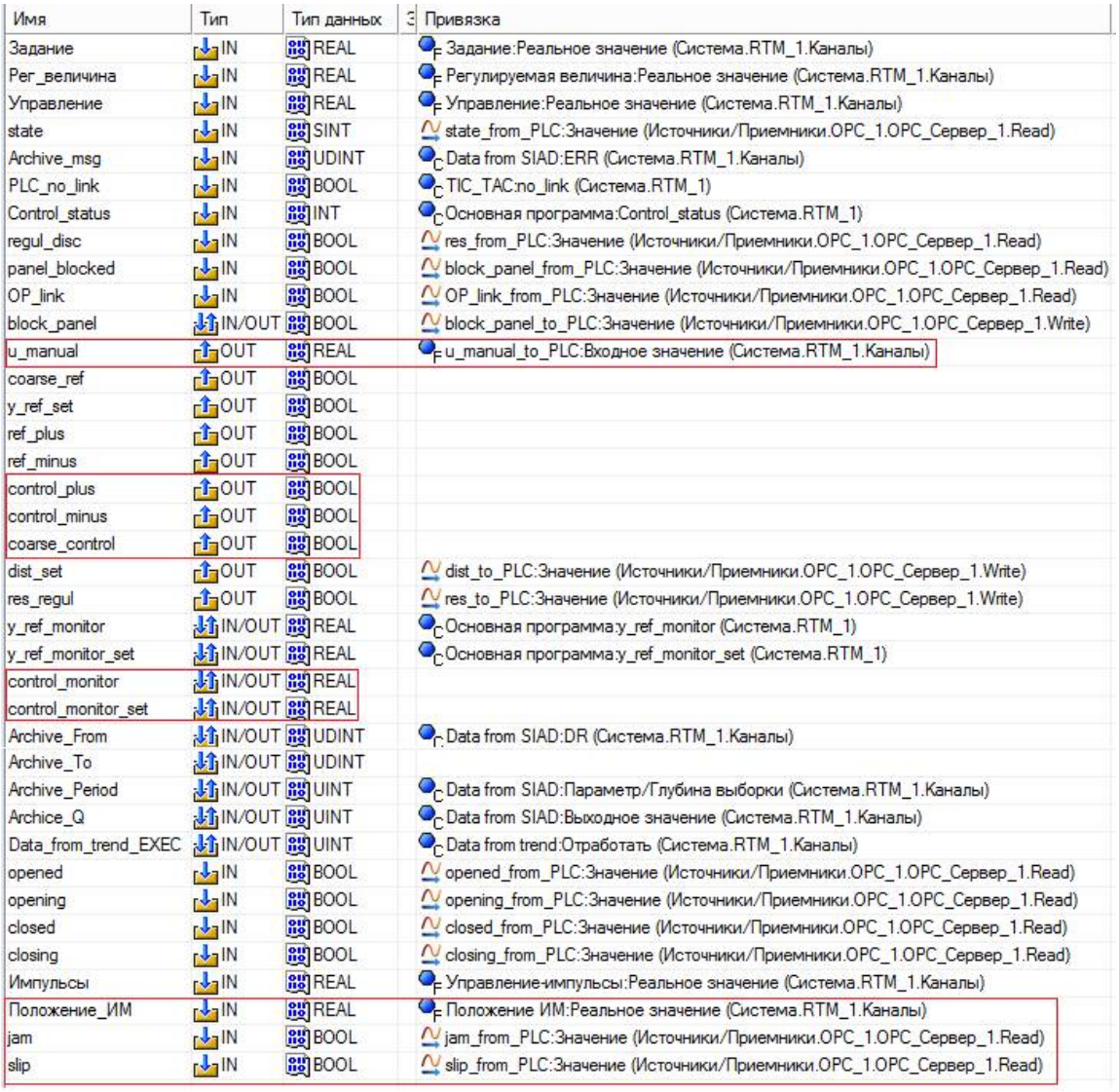


Рис. 3.97. Экран визуализации в работе.



Имя	Тип	Тип данных	Привязка
Задание	IN	REAL	Задание:Реальное значение (Система.RTM_1.Каналы)
Рег_величина	IN	REAL	Регулируемая величина:Реальное значение (Система.RTM_1.Каналы)
Управление	IN	REAL	Управление:Реальное значение (Система.RTM_1.Каналы)
state	IN	SINT	state_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
Archive_msg	IN	UDINT	Data from SIAD:ERR (Система.RTM_1.Каналы)
PLC_no_link	IN	BOOL	TIC_TAC:no_link (Система.RTM_1)
Control_status	IN	INT	Основная программа:Control_status (Система.RTM_1)
regul_disc	IN	BOOL	res_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
panel_blocked	IN	BOOL	block_panel_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
OP_link	IN	BOOL	OP_link_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
block_panel	IN/OUT	BOOL	block_panel_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
u_manual	OUT	REAL	u_manual_to_PLC:Входное значение (Система.RTM_1.Каналы)
coarse_ref	OUT	BOOL	
y_ref_set	OUT	BOOL	
ref_plus	OUT	BOOL	
ref_minus	OUT	BOOL	
control_plus	OUT	BOOL	
control_minus	OUT	BOOL	
coarse_control	OUT	BOOL	
dist_set	OUT	BOOL	dist_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
res_regul	OUT	BOOL	res_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
y_ref_monitor	IN/OUT	REAL	Основная программа:y_ref_monitor (Система.RTM_1)
y_ref_monitor_set	IN/OUT	REAL	Основная программа:y_ref_monitor_set (Система.RTM_1)
control_monitor	IN/OUT	REAL	
control_monitor_set	IN/OUT	REAL	
Archive_From	IN/OUT	UDINT	Data from SIAD:DR (Система.RTM_1.Каналы)
Archive_To	IN/OUT	UDINT	
Archive_Period	IN/OUT	UINT	Data from SIAD:Параметр/Глубина выборки (Система.RTM_1.Каналы)
Archice_Q	IN/OUT	UINT	Data from SIAD:Выходное значение (Система.RTM_1.Каналы)
Data_from_trend_EXEC	IN/OUT	UINT	Data from trend:Отработать (Система.RTM_1.Каналы)
opened	IN	BOOL	opened_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
opening	IN	BOOL	opening_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
closed	IN	BOOL	closed_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
closing	IN	BOOL	closing_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
Импульсы	IN	REAL	Управление-импульсы:Реальное значение (Система.RTM_1.Каналы)
Положение_ИМ	IN	REAL	Положение ИМ:Реальное значение (Система.RTM_1.Каналы)
jam	IN	BOOL	jam_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
slip	IN	BOOL	slip_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)

Рис. 3.98. Аргументы экрана и их привязки.

Отличия от экрана, разработанного в п. 3.1.1.4, следующие:

1) графический элемент «Тренд» помимо задания и регулируемой величины основного контура отображает теперь задание (выходной сигнал ПИ регулятора) и регулируемую величину (положение исполнительного механизма) внутреннего контура. Впрочем, выход ПИ регулятора отображался и ранее, но как *управляющее воздействие*. Кроме того, как и в п. 3.1.2.4, на тренд добавлена кривая для отображения импульсов управления исполнительным механизмом;

2) ползунки группы «Управление» отображают действительное и заданное положение исполнительного механизма. С помощью правого ползунка можно изменять задание в режиме дистанционного управления. Справа от ползунков размещены органы индикации состояния вентиля («открыт», «открывается», «заклинивание», «проскальзывание», «закрывается», «закрыт»). Индикаторы аварий в нормальном состоянии на экране невидимы;

3) в архивную таблицу добавлен столбец для вывода положения исполнительного механизма.

Привязки графических элементов экрана к его аргументам показаны на рис. 3.99 в виде текстовых надписей поверх экрана визуализации в «нерабочем» состоянии.

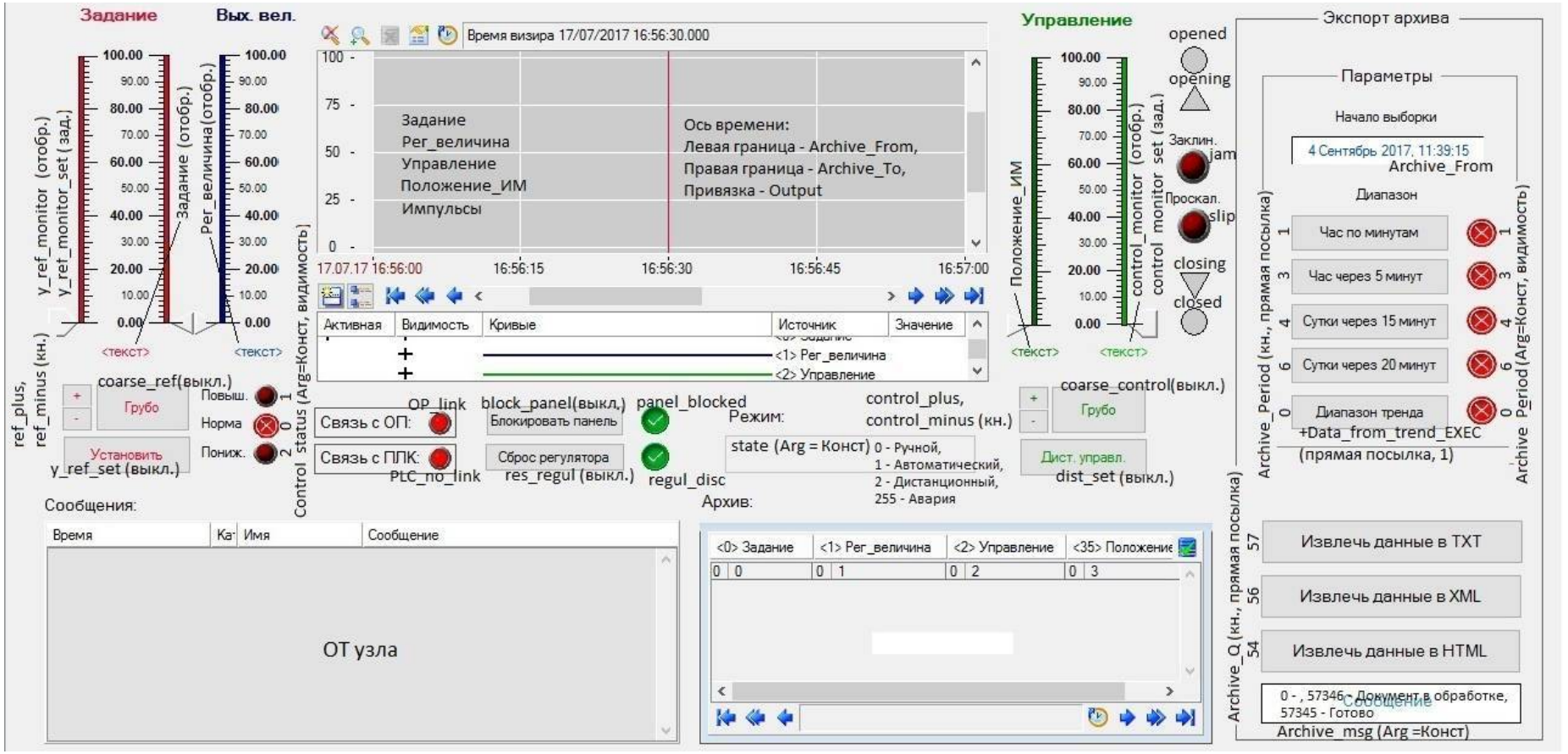


Рис. 3.99. Привязки графических элементов к аргументам экрана.

В словарь «Тревоги» добавлены сообщения о заклинивании и проскальзывании вентиля и устранении этих аварий (рис. 3.100).

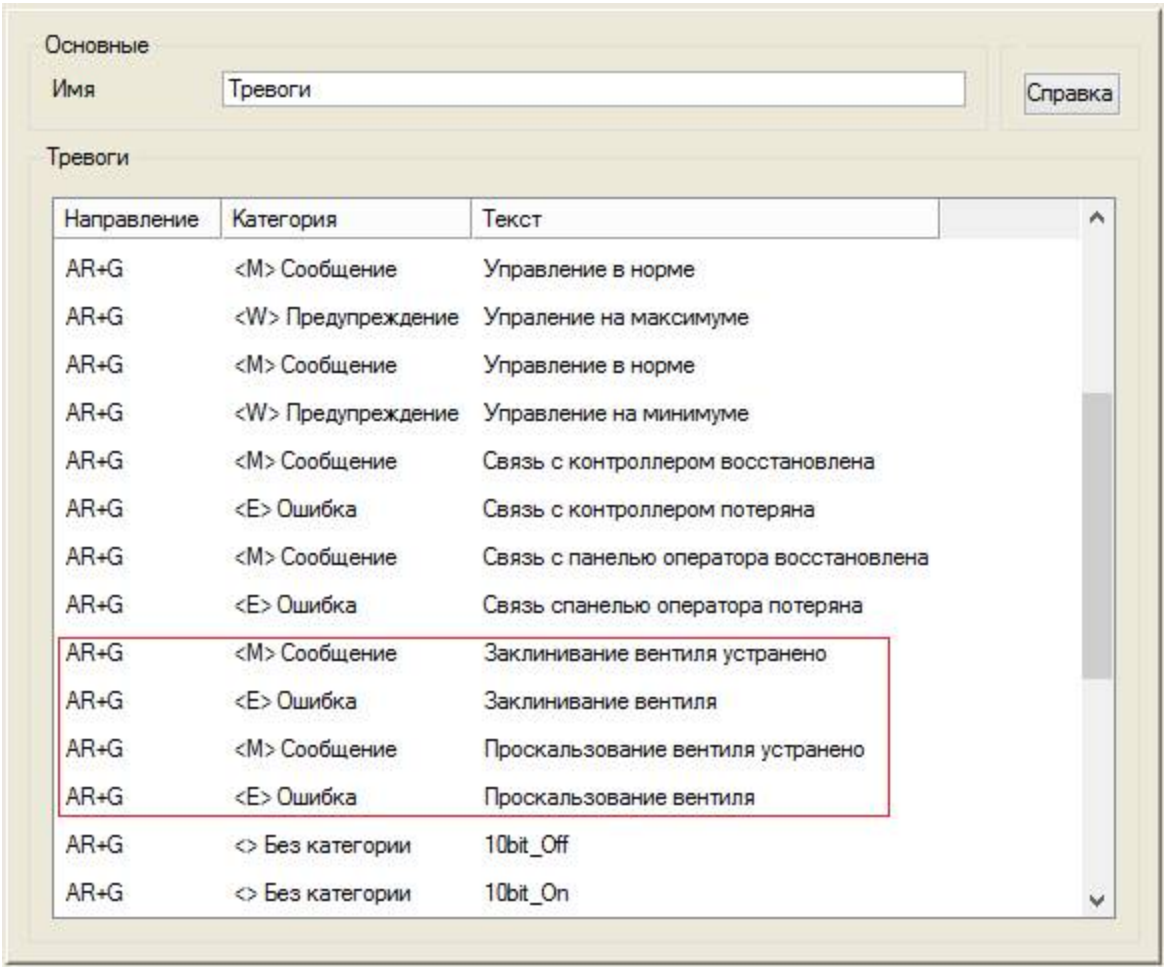


Рис. 3.100. Новые сообщения словаря «Тревоги» .

Основная программа модифицирована незначительно:

```
PROGRAM
     VAR_INPUT ref_plus : BOOL; END_VAR
     VAR_INPUT ref_minus : BOOL; END_VAR
     VAR_INPUT coarse_y_ref : BOOL; END_VAR
     VAR_INPUT set_ref : BOOL; END_VAR
     VAR_INPUT y_ref : REAL; END_VAR
     VAR_INPUT control_plus : BOOL; END_VAR
     VAR_INPUT control_minus : BOOL; END_VAR
     VAR_INPUT coarse_control : BOOL; END_VAR
     VAR_INPUT u : REAL; END_VAR
     VAR_INPUT y : REAL; END_VAR
     VAR_INPUT state : SINT; END_VAR
     VAR_INPUT OP_link_from_PLC : BOOL := true; END_VAR
     VAR_OUTPUT error : REAL; END_VAR
     VAR_OUTPUT y_ref_to_PLC : REAL; END_VAR
     VAR_OUTPUT y_ref_update : REAL; END_VAR
     VAR_OUTPUT u_manual_to_PLC : REAL; END_VAR
     VAR_OUTPUT Control_status : INT; END_VAR
```

```
VAR_INOUT y_ref_monitor : REAL; END_VAR
VAR_INOUT y_ref_monitor_set : REAL; END_VAR
VAR_INOUT control_monitor : REAL; END_VAR
VAR_INOUT control_monitor_set : REAL; END_VAR
VAR_INOUT DiscreteAlarms : UINT; END_VAR
VAR_INPUT closed : BOOL; END_VAR
VAR_INPUT closing : BOOL; END_VAR
VAR_INPUT opened : BOOL; END_VAR
VAR_INPUT opening : BOOL; END_VAR
VAR_OUTPUT u_impulse : REAL; END_VAR
VAR_INPUT jam : BOOL; END_VAR
VAR_INPUT slip : BOOL; END_VAR

//ЗАДАНИЕ
IF coarse_y_ref THEN //грубый ввод задания
     //ползунок идет за мышью
     y_ref_monitor:=y_ref_monitor_set;
ELSE //точный ввод, реакция на кнопки "+","-"
     IF ref_plus THEN
          y_ref_monitor:=y_ref_monitor+1;
     ELSIF ref_minus THEN
          y_ref_monitor:=y_ref_monitor-1;
     END_IF;
     y_ref_monitor_set:=y_ref_monitor;
END_IF;
IF set_ref THEN //Установка задания
     y_ref_update:=TRUE;
     y_ref_to_PLC:=y_ref_monitor;
ELSE
     y_ref_update:=FALSE;
     y_ref_to_PLC:=y_ref; //отправляем то, что получили
END_IF;
//ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ
IF state==2 THEN //В режиме дистанционного управления
     IF coarse_control THEN //Грубый ввод задания
          //ползунок идет за мышью
          control_monitor:=control_monitor_set;
     ELSE //точный ввод, реакция на кнопки "+","-"
          IF control_plus THEN
               control_monitor:=control_monitor+1;
          ELSIF control_minus THEN
               control_monitor:=control_monitor-1;
          END_IF;
          control_monitor_set:=control_monitor;
     END_IF;
     //отправляем то, что ввели
     u_manual_to_PLC:=control_monitor;
ELSE
     control_monitor:=u;
     control_monitor_set:=u;
     u_manual_to_PLC:=u; //отправляем то, что получили
```

```
    END_IF;
    //ФОРМИРОВАНИЕ СТАТУСА
    error:=y_ref-y; //ошибка регулирования
    IF error > 10 THEN
         Control_status:= 2;
    ELSIF error < -10 THEN
         Control_status:= 1;
    ELSE
         Control_status:= 0;
    END_IF;

    //ФОРМИРОВАНИЕ СООБЩЕНИЙ
    //сброс тех битов, которые могут быть установлены
    DiscreteAlarms:= DiscreteAlarms & 2#10000000000;
    CASE state OF
    255: DiscreteAlarms:=DiscreteAlarms | 2#1;
    0: DiscreteAlarms:=DiscreteAlarms | 2#10;
    1: DiscreteAlarms:=DiscreteAlarms | 2#100;
    2: DiscreteAlarms:=DiscreteAlarms | 2#1000;
    END_CASE;
    IF opened THEN
         DiscreteAlarms:=DiscreteAlarms | 2#10000;
    ELSIF closed THEN
         DiscreteAlarms:=DiscreteAlarms | 2#100000;
    END_IF;
    IF NOT OP_link_from_PLC THEN
         DiscreteAlarms:=DiscreteAlarms | 2#10000000;
    END_IF;
    IF jam THEN
         DiscreteAlarms:=DiscreteAlarms | 2#100000000;
    END_IF;
         IF slip THEN
         DiscreteAlarms:=DiscreteAlarms | 2#1000000000;
    END_IF;
    // ФОРМИРОВАНИЕ ТРЕНДА ИМПУЛЬСОВ
    IF opening THEN
         u_impulse:=75;
    ELSIF closing THEN
         u_impulse:=25;
    ELSE
         u_impulse:=50;
    END_IF;
END_PROGRAM
```

Список аргументов Основной программы приведен на рис. 3.101.

Жирным в программе и контуром в списке аргументов выделены исправления и дополнения, сделанные относительно п. 3.1.1.4. Частично они повторяют те изменения, которые уже имели место в п. 3.1.2.4.

Имя	Тип	Тип данных	Значе	Привязка
ref_plus	IN	BOOL		Экран#1:1:ref_plus (Система.RTM_1)
ref_minus	IN	BOOL		Экран#1:1:ref_minus (Система.RTM_1)
coarse_y_ref	IN	BOOL		Экран#1:1:coarse_ref (Система.RTM_1)
set_ref	IN	BOOL		Экран#1:1:y_ref_set (Система.RTM_1)
y_ref	IN	REAL		Задание:Реальное значение (Система.RTM_1.Каналы)
control_plus	IN	BOOL		Экран#1:1:control_plus (Система.RTM_1)
control_minus	IN	BOOL		Экран#1:1:control_minus (Система.RTM_1)
coarse_control	IN	BOOL		Экран#1:1:coarse_control (Система.RTM_1)
u	IN	REAL		Управление:Реальное значение (Система.RTM_1.Каналы)
y	IN	REAL		Регулируемая величина:Реальное значение (Система.RTM_1.Каналы)
state	IN	SINT		state_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
OP_link_from_PLC	IN	BOOL	true	OP_link_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
error	OUT	REAL		Ошибка регулирования:Входное значение (Система.RTM_1.Каналы)
y_ref_to_PLC	OUT	REAL		y_ref_to_PLC:Входное значение (Система.RTM_1.Каналы)
y_ref_update	OUT	REAL		y_ref_update_to_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Write)
u_manual_to_PLC	OUT	REAL		u_manual_to_PLC:Входное значение (Система.RTM_1.Каналы)
Control_status	OUT	INT		
y_ref_monitor	IN/OUT	REAL		Экран#1:1:y_ref_monitor (Система.RTM_1)
y_ref_monitor_set	IN/OUT	REAL		Экран#1:1:y_ref_monitor_set (Система.RTM_1)
control_monitor	IN/OUT	REAL		Экран#1:1:control_monitor (Система.RTM_1)
control_monitor_set	IN/OUT	REAL		Экран#1:1:control_monitor_set (Система.RTM_1)
DiscreteAlarms	IN/OUT	UINT		DiscreteAlarms:Входное значение (Система.RTM_1.Каналы)
closed	IN	BOOL		closed_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
closing	IN	BOOL		closing_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
opened	IN	BOOL		opened_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
opening	IN	BOOL		opening_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
u_impulse	OUT	REAL		Управление-импульсы:Входное значение (Система.RTM_1.Каналы)
jam	IN	BOOL		jam_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)
slip	IN	BOOL		slip_from_PLC:Значение (Источники/Приемники.OPC_1.OPC_Сервер_1.Read)

Рис. 3.99. Привязки графических элементов к аргументам экрана.

3.1.3.5. Апробация системы

На рис. 3.102 показаны графики изменения регулируемой величины, управляющего воздействия (положения исполнительного механизма) и выходного сигнала ПИ регулятора эталонной (Simulink) системы и системы на базе виртуального ПЛК. Тренды сохранены блоками Scope «y»,«u» и «u2» Simulink-диаграммы, показанной на рис. 3.90, в одноименных переменных.

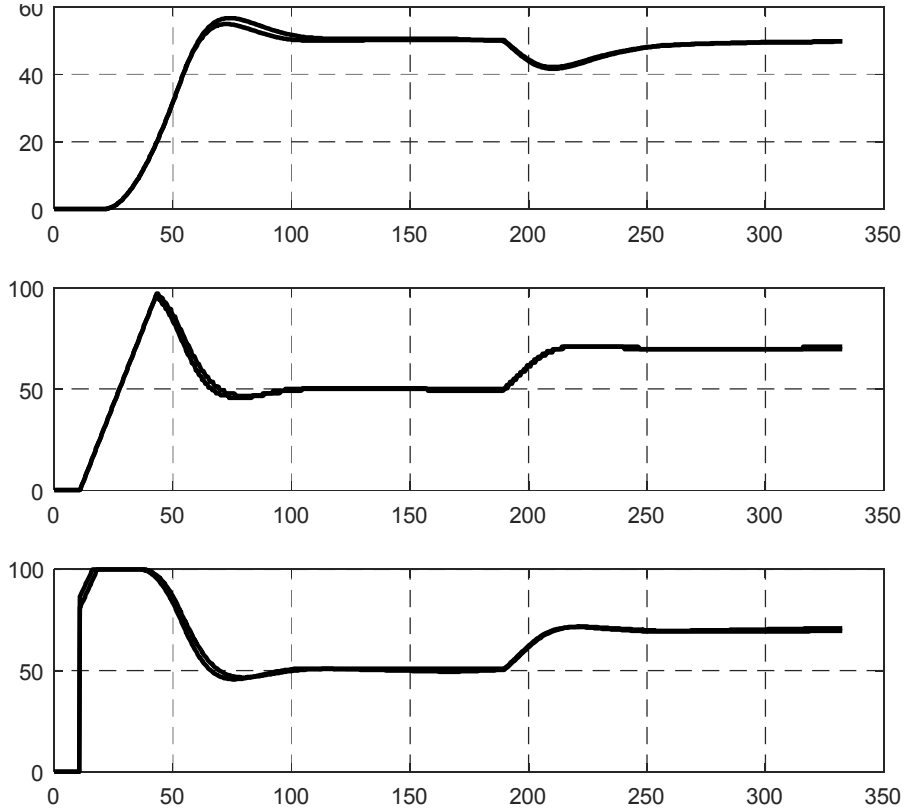


Рис. 3.102. Регулируемая величина, положение исполнительного механизма и выходной сигнал ПИ-регулятора в эталонной системе и системе на базе ПЛК.

Рис. 3.103 демонстрирует работу регулятора положения исполнительного механизма системы на базе ПЛК. Тренд сохранен блоком «u3» в переменной `u3`.

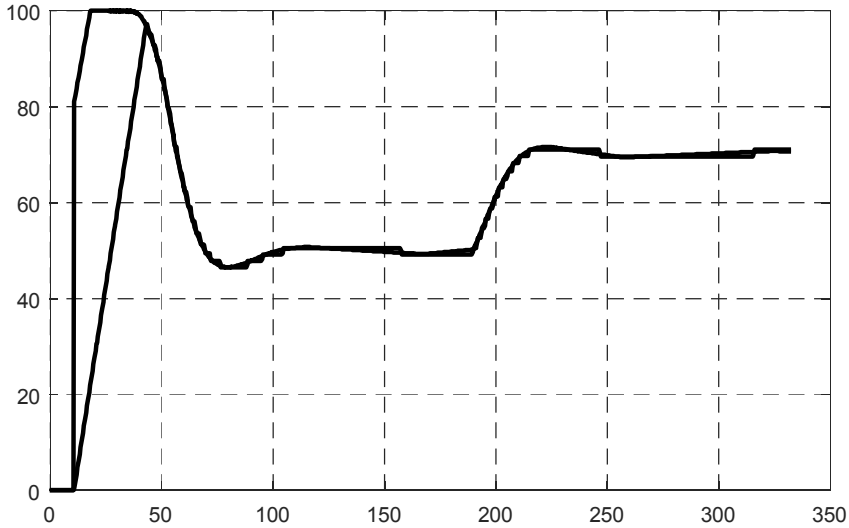


Рис. 3.103. Выходной сигнал ПИ-регулятора и положение исполнительного механизма в системе на базе ПЛК.

Графики построены с помощью кода, приведенного ниже:

```
figure(1)
subplot(3,1,1)
L = plot(y(:,1),y(:,2),y(:,1),y(:,3)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(3,1,2)
L = plot(u(:,1),u(:,2),u(:,1),u(:,3)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
subplot(3,1,3)
L = plot(u2(:,1),u2(:,2),u2(:,1),u2(:,3)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
figure(2)
L = plot(u3(:,1),u3(:,2),u3(:,1),u3(:,3)), grid
set(L(1),'LineWidth', 2);
set(L(1),'Color', [0 0 0]);
set(L(2),'LineWidth', 2);
set(L(2),'Color', [0 0 0]);
```

3.1.4. Системы с нетиповыми законами регулирования

3.1.4.1. Программная реализация нетиповых линейных законов регулирования

В предыдущих пунктах рассматривались системы регулирования на базе ПИ регулятора. В п. 3.1.2 ПИ закон регулирования был реализован совместно ПД алгоритмом контроллера и исполнительным механизмом «интегрирующего типа». ПИ и ПД законы являются частными случаями ПИД закона регулирования, применяющегося в большинстве промышленных систем автоматического регулирования, к которым предъявляются повышенные требования по точности стабилизации или воспроизведения технологических параметров. Системы программирования ПЛК предлагают готовые функциональные блоки, реализующие ПИ, ПД и ПИД законы регулирования [4].

Между тем, «с точки зрения» теории автоматического управления ПИД регулятор является довольно ограниченной по своим возможностям структурой и его применение оправдано тогда, когда мы имеем дело с «простыми» объектами управления. Зачастую практика и не противоречит этому тезису, поскольку при настройке регулятора задействуется максимально упрощенное математическое описание объекта, полученное, например, в результате обработки экспериментальных разгонных кривых. Имея передаточную функцию первого порядка с запаздыванием, нет никакого смысла применять сложные законы регулирования, тем более, как уже было сказано, есть много методов настройки ПИД регулятора для такого объекта. Существуют, конечно, и такие объекты,

для которых может быть получено адекватное описание в виде дифференциальных уравнений и передаточных функций высокого порядка. Это могут быть, например, электромеханические системы, уравнения которых выводятся аналитически, каскадные системы, когда объектом управления выступают внутренние контура регулирования и т.п. системы. В таких случаях уместно вспомнить принцип Эшби, согласно которому *сложность управления* (регулятора) *должна соответствовать сложности системы* (объекта). Определенная проблема состоит в реализации «сложных» передаточных функций на языках программирования ПЛК. В отличие от систем имитационного моделирования, таких как Simulink, библиотеки систем программирования ПЛК (по крайней мере, стандартные) не предоставляют для этого готовых функциональных блоков. В предыдущей главе было показано, что любая физически реализуемая передаточная функция может быть построена на интеграторах, сумматорах и масштабных коэффициентах. Такими средствами обладает любая среда программирования ПЛК. Здесь мы рассмотрим, как построить передаточную функцию регулятора на языках стандарта МЭК 61131.

В CoDeSys сумматоры и коэффициенты реализуются с помощью встроенных арифметических операторов ADD, SUB, MUL в языках IL, LD, FBD и соответствующих знаков операций в ST. Для интегрирования привлекается функциональный блок INTEGRAL из системной библиотеки Util.lib.

Исследовались переходные процессы, вызванные изменением задания (от 0 до 50%) и возмущения (от 0 до 20%).

Результаты апробации в целом говорят о сходности поведения систем регулирования и корректной реализации алгоритмов на ПЛК. Расхождения в графиках объясняются в первую очередь задержкой при обмене по OPC.

Пусть объект управления описывается передаточной функцией второго порядка:

$$W_{об}(p) = \frac{1}{100p^2 + 10p + 1}.$$

В п. 2.2.3.2 рассмотрен метод динамической компенсации, позволяющий в ряде случаев обеспечить эталонную динамику замкнутого контура. Настало время применить теорию «на практике».

Объект является минимально-фазовым, так как вещественные части его полюсов отрицательны: $p_{1,2} = -0{,}05 \pm 0{,}087j$, а нулей и вовсе нет. Поэтому метод динамической компенсации применим «без ограничений».

Пусть эталон задан передаточной функцией замкнутой системы вида:

$$W_{зам}(p) = \frac{1}{100p^2 + 20p + 1}.$$

Такая передаточная функция обеспечит максимально быстрые переходные процессы без колебаний (биноминальное распределение полюсов, $p_{1,2} = -0{,}1$, см. п. 2.2.3.2).

Эталонная передаточная функция разомкнутой системы находится через эталонную передаточную функцию замкнутой:

$$W_{раз}(p)=\frac{W_{зам}(p)}{1-W_{зам}(p)}=\frac{1}{100p^2+20p}.$$

Передаточная функция регулятора:

$$W_{рег}(p)=\frac{W_{раз}(p)}{W_{об}(p)}=\frac{100p^2+10p+1}{100p^2+20p}.$$

Представим передаточную функцию регулятора в виде:

$$W_{рег}(p)=1+\frac{-10p+1}{100p^2+20p}=1+\frac{-0,1p+0,01}{p^2+0,2p}.$$

По данному выражению легко составляются структурная схема и уравнения регулятора в пространстве состояний, которые и будут исходными данными для составления программ ПЛК (рис. 3.104).

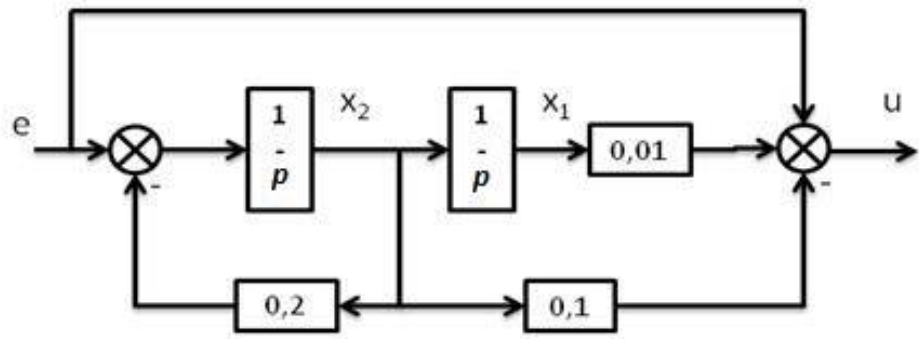


Рис. 3.104. Модель регулятора

$$\begin{cases} x_1=\int x_2; \\ x_2=\int\left(e-0,2x_2\right); \\ u=e+0,01x_1-0,1x_2. \end{cases}$$

Перед составлением Simulink-диаграммы, был разработан «зародыш» программы управления в CodeSys, в котором были определены глобальные переменные, участвующие в обмене по протоколу OPC:

```
VAR_GLOBAL
   sp, y, u:REAL;
   res: BOOL;
END_VAR
```

Назначение переменных:

`sp` (set point) – уставка (задание);

`y` – значение регулируемой величины;

`u` – управляющее воздействие;

`res` – сигнал «сброса», обнуляющий интеграторы.

Составлена Simulink-модель, позволяющая сравнивать поведение модели системы регулирования с поведением комбинированной системы, в которой для управления задействуется виртуальный контроллер CoDeSys (рис. 3.105).

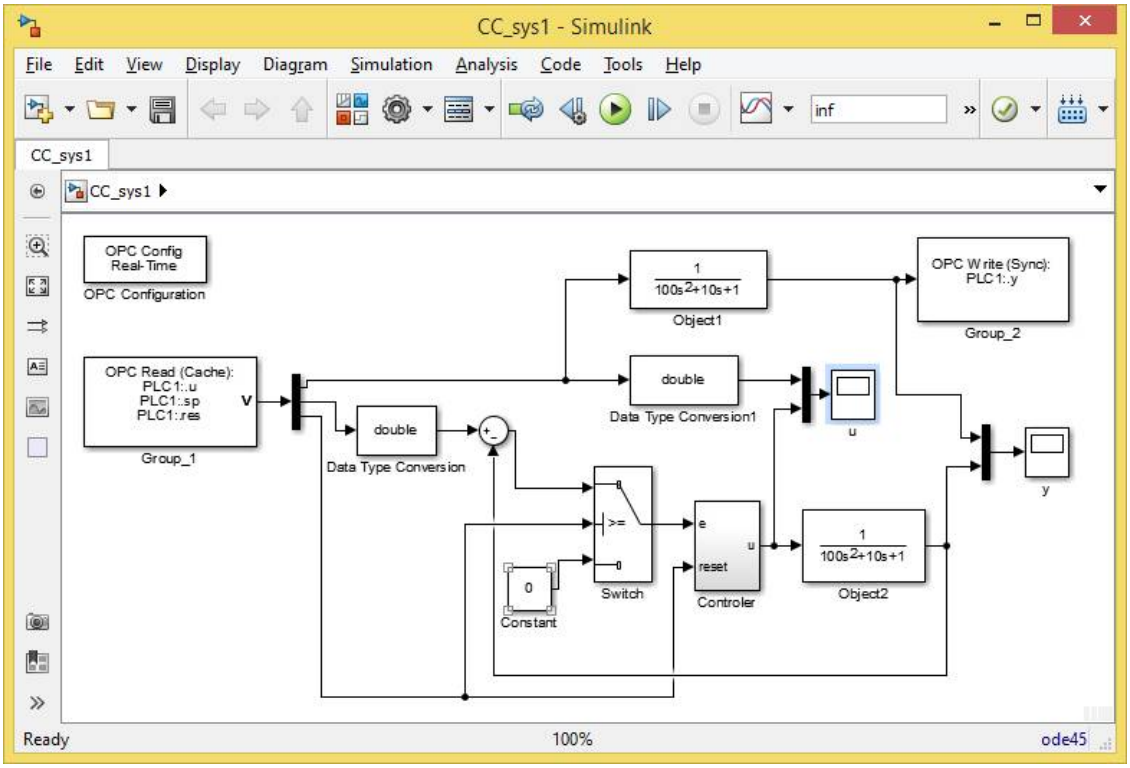


Рис. 3.105. Simulink-модель.

Для обмена данными используются блоки OPC Configuration, OPC Read и OPC Write. Временной интервал обмена был выставлен равным 0,1 сек, что примерно соответствует среднему времени задержек при приеме и передаче информации в системах регулирования на основе не самых быстрых АЦП и промышленных сетей.

«Локальный» алгоритм регулирования (рис. 3.106) представлен «развернутой» моделью, которая предусматривает возможность сброса интеграторов.

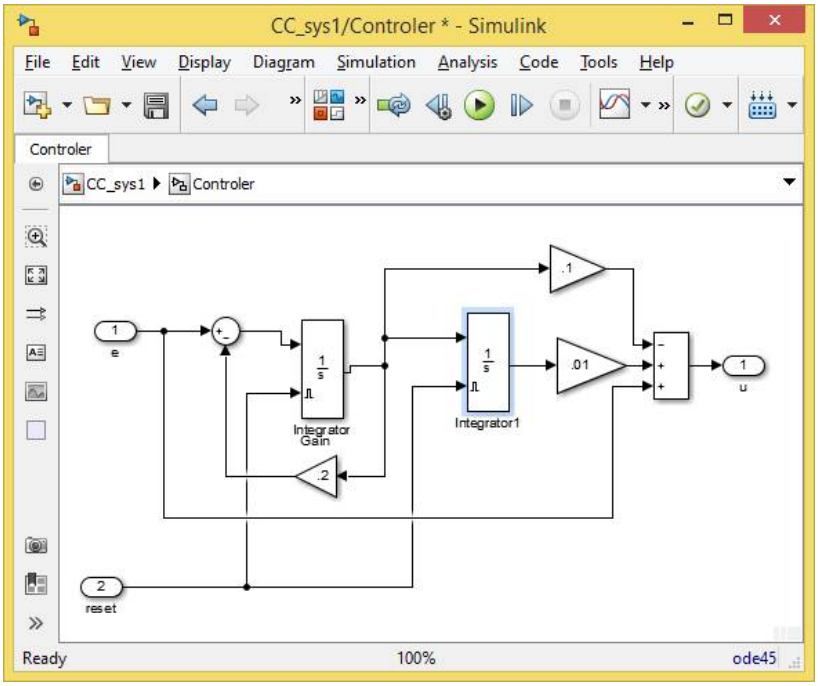


Рис. 3.106. Модель регулятора.

Программное обеспечение контроллера представлено циклически выполняемой программой PLC_PRG, которая, в свою очередь, вызывает одну из трех программ: ST_PRG, FBD_PRG или CFC_PRG, составленных на разных языках:

```
ST_PRG;
(*FBD_PRG;
CFC_PRG;*)
```

Все три программы делают одно и то же, а именно – напрямую реализуют полученный ранее закон регулирования. Программа на языке ST:

```
PROGRAM ST_PRG
VAR
   Integ1: INTEGRAL;
   Integ2: INTEGRAL;
   x1:REAL:=0;
   x2:REAL:=0;
   e:REAL;
END_VAR

e:=sp-y;
Integ1(IN:=x2, TM:=50, RESET:=res, OUT=>x1);
Integ2(IN:=e-0.2*x2, TM:=50, RESET:=res, OUT=>x2);
u:=e + 0.01*x1 - 0.1*x2;
```

Программа на FBD содержит аналогичный раздел объявлений, что и предыдущая программа. Раздел «кода» показан на рис. 3.107.

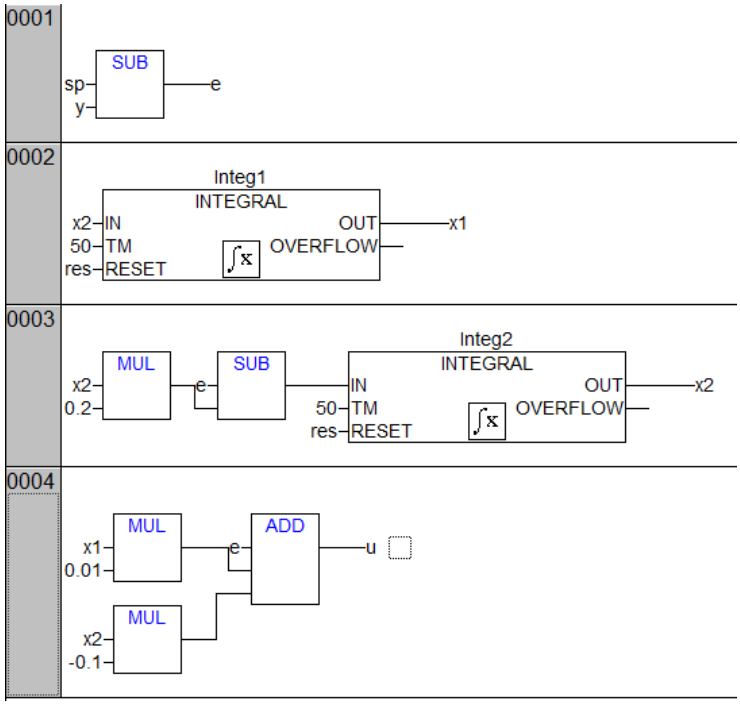


Рис. 3.107. Программа FBD_PRG.

Раздел объявлений программы на языке CFC:

```
PROGRAM CFC_PRG
VAR
   Integ1: INTEGRAL;
   Integ2: INTEGRAL;
```

```
END_VAR
```

Раздел кода показан на рис. 3.108.

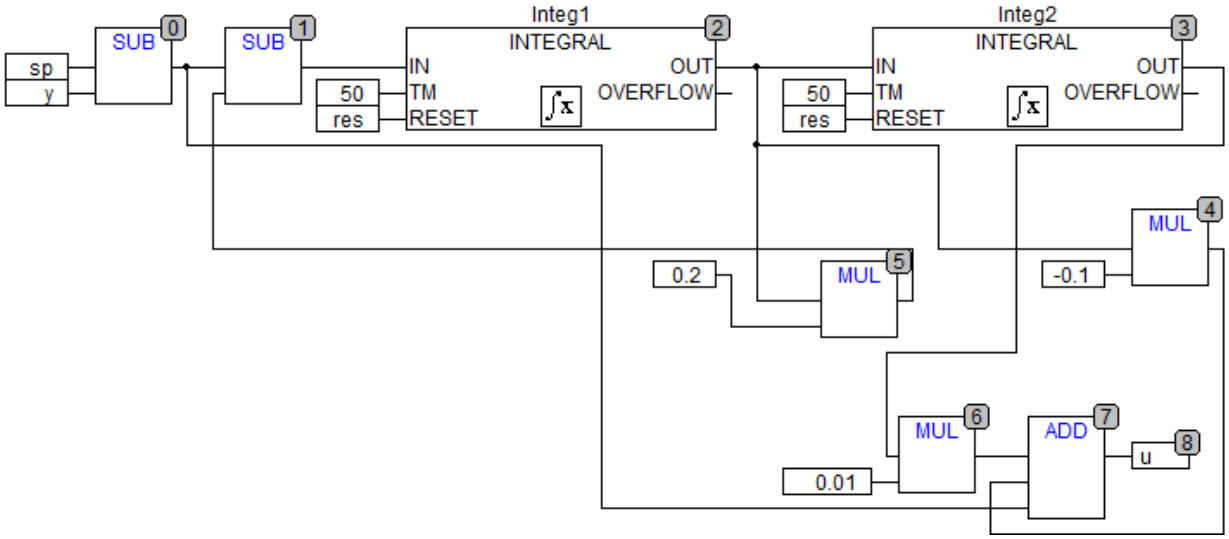


Рис. 3.108. Программа `CFC_PRG`.

Все три программы, как и следовало ожидать, продемонстрировали полностью идентичное поведение.

На рис. 3.109 показан экран визуализации с графиком отработки системой регулирования изменения задания от нуля до пятидесяти единиц. На графике представлены кривые переходных процессов управляющего и выходного сигналов.

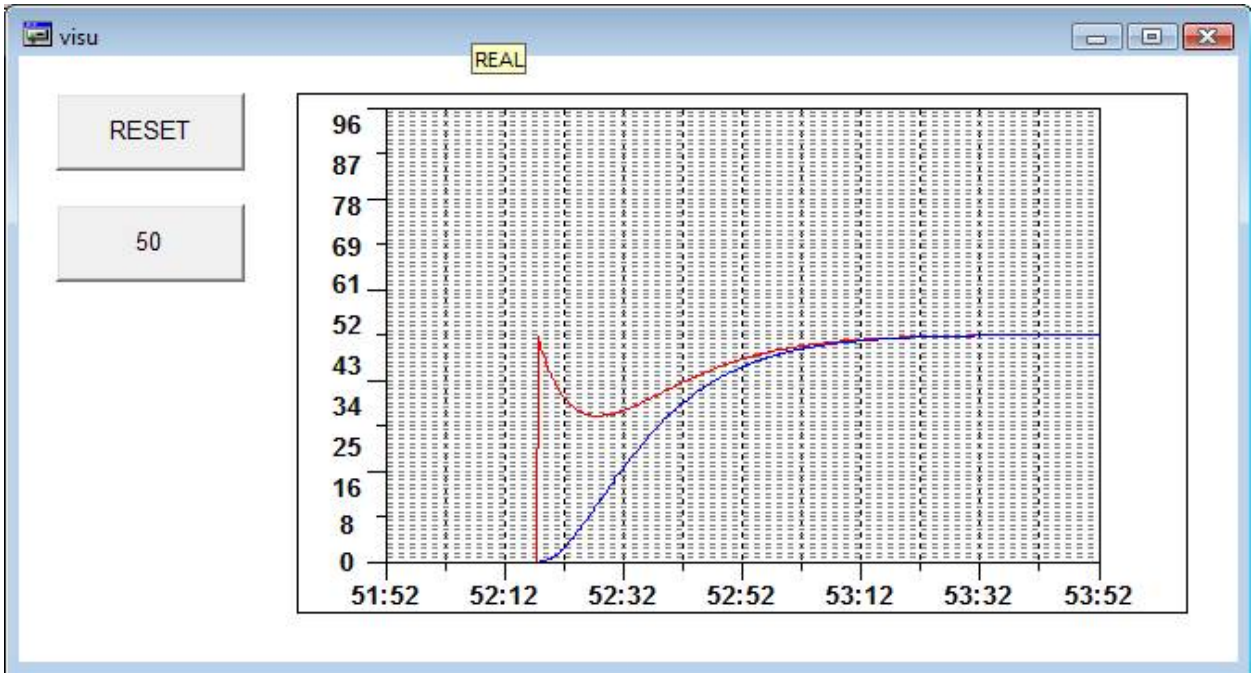


Рис. 3.109. Экран визуализации программы управления

Результаты сравнения поведения модели системы регулирования в Simulink и комбинированной системы с виртуальным контроллером приведены на рис. 3.110 и 3.111.

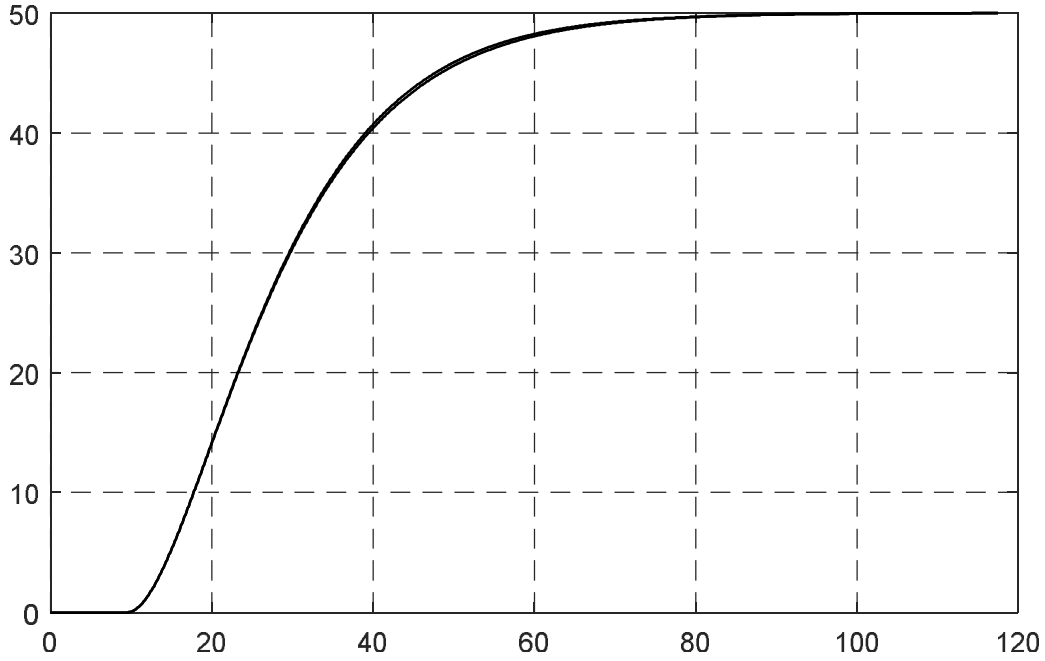


Рис. 3.110. Переходный процесс: выходная величина y(t).

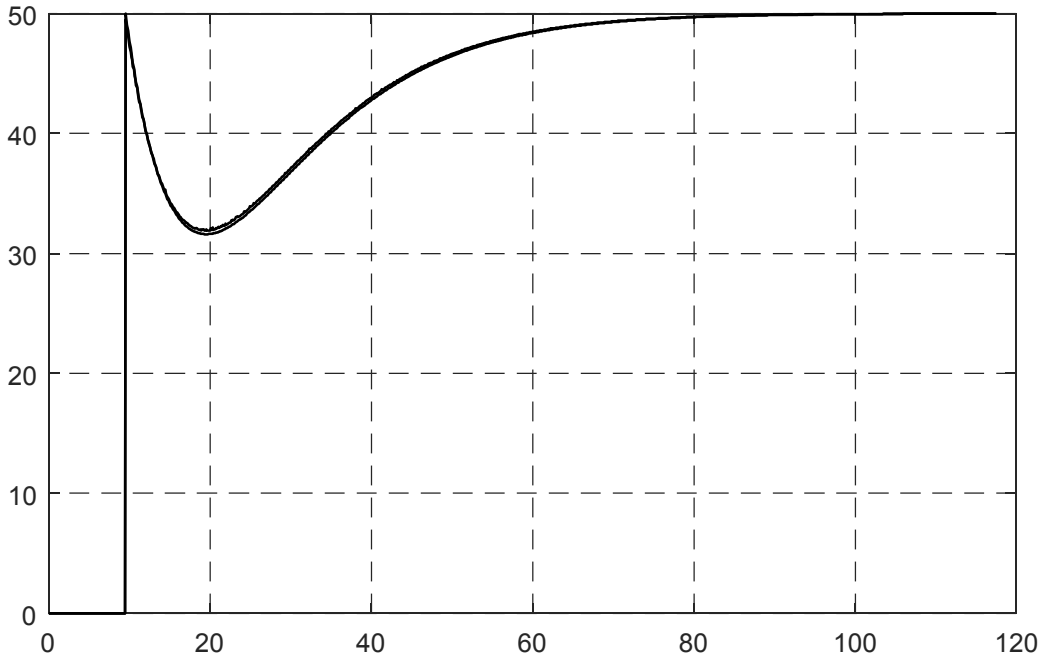


Рис. 3.111. Переходный процесс: управляющее воздействие u(t).

Как видно из рисунков, «комбинированная» система практически воспроизводит поведение имитационной модели Simulink, что свидетельствует о корректной реализации алгоритмов в программах для ПЛК. Незначительные отличия объясняются дискретным характером обмена данными между программами.

Далее интегрируем рассмотренный алгоритм регулирования в «полноценную» программу для ПЛК, для чего внесем необходимые изменения в разработку из п. 3.1.1.3. Верхний уровень (SCADA-систему) затрагивать вообще не будем, поскольку ничего нового, по сравнению с п. 3.1.1.4, здесь нас не ждет.

В состав POU программы контроллера входят:

программа CONTROL, в которой реализован алгоритм регулирования. Программа вызывается циклически;

программа PANEL, предназначенная для обслуживания операторской панели, вызывается из PLC_PRG;

программа PLC_PRG, вызывается свободно.

Из списка глобальных переменных исключены все переменные, отвечающие за связь со SCADA-системой:

```
VAR_GLOBAL
  (*Входы и выходы контроллера*)
   y:REAL; (*регулируемая величина*)
   reset_process:BOOL:=FALSE; (*сброс системы после аварии*)
  (*Выходы контроллера*)
   u:REAL; (*регулируемая величина и управление*)
   no_HMI:BOOL:=FALSE; (*потеряна связь с панелью*)
   stop:BOOL:=FALSE; (*полная остановка системы*)
   (*"Внутренние" переменные*)
    y_ref:REAL; (*задание рег., подается и в Simulink-модель*)
    u_manual:REAL:=0;(*сигнал ручного управления*)
    man:BOOL:=FALSE;(*сигнал перевода в режим ручного управ.*)
    res:BOOL:=FALSE;(*сброс рег., подается и в Simulink-модель*)
   (*Переменные обмена с панелью оператора*)
   (*входы:*)
    y_ref_from_OP:REAL:=0; (*задание с панели*)
    y_ref_update_from_OP:BOOL:=FALSE; (*обновить задание*)
    u_manual_from_OP:REAL:=0;(*сигнал ручного управления*)
    man_from_OP:BOOL:=FALSE;(*сигнал перехода на ручной режим *)
    res_from_OP:BOOL; (*сброс регулятора с панели*)
    tic_tac_from_OP:BOOL; (*тестовый сигнал наличия связи*)
   (*выходы:*)
    y_ref_to_OP, y_to_OP, u_to_OP:REAL;(*зад., выход и управл.*)
    res_to_OP:BOOL; (*сброс регулятора на панель*)
    state_to_OP:SINT; (*режим управления на панель*)
    err_gt_10_to_OP, err_lt_m10_to_OP, norma_to_OP:BOOL;
    (*-  сигналы контроля на панель*)
     tic_tac_to_OP:BOOL; (*тестовый сигнал наличия связи*)
END_VAR
```

В программе CONTROL, код которой приведен ниже, формируется переменная управления u. В ручном режиме в эту переменную заносится значение переменной u_manual, формируемое оператором панели, а интеграторы «замораживаются» путем подачи на их входы IN нулей. В режиме автоматического управления алгоритм работает в «нормальном режиме», за исключением случая, когда «активирована» переменная res (сброс регулятора), – тогда управление обнуляется, а интеграторы сбрасываются.

```
PROGRAM CONTROL
VAR
     Integ1: INTEGRAL;
     Integ2: INTEGRAL;
     x1:REAL:=0;
```

```
        x2:REAL:=0;
        e:REAL;
    END_VAR
    (*секция кода*)
    IF man THEN
        Integ1(IN:=0, TM:=50, RESET:=res, OUT=>x1);
        Integ2(IN:=0, TM:=50, RESET:=res, OUT=>x2);
        u:=u_manual;
    ELSE
        e:=y_ref-y;
        Integ1(IN:=x2, TM:=50, RESET:=res, OUT=>x1);
        Integ2(IN:=e-0.2*x2, TM:=50, RESET:=res, OUT=>x2);
        IF res THEN
            u:=0;
        ELSE
            u:=e + 0.01*x1 - 0.1*x2;
        END_IF
    END_IF
```

Окно оператора панели показано на рис. 3.112.

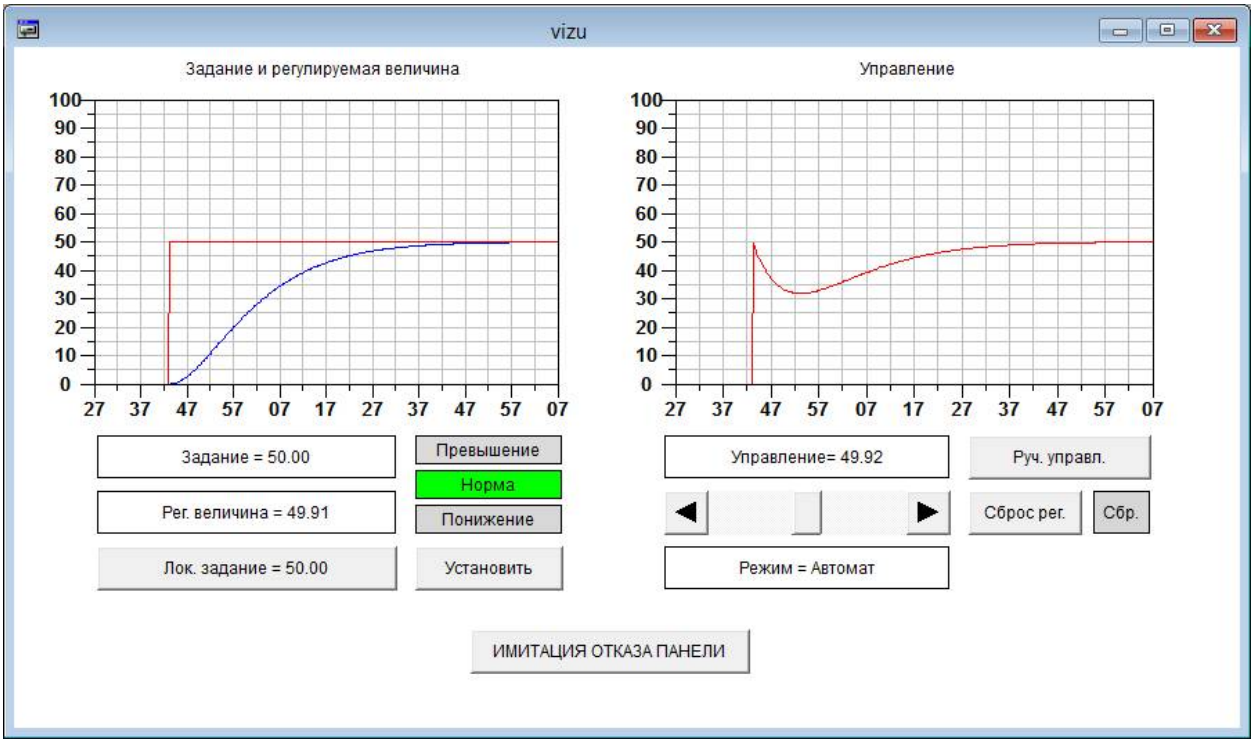


Рис. 3.112. Окно панели оператора.

Из окна визуализации, разработанного в п. 3.1.1.3, исключены элементы, связанные со SCADA-системой. Соответственно «почищены» и программы `PANEL` и `PLC_PRG`, коды которых приведены ниже.

```
PROGRAM PANEL
VAR
    (*ОТОБРАЖЕНИЕ*)
    y_ref, y, u:REAL; (*задание, выход и управление*)
    err_gt_10, err_lt_m10, norma:BOOL; (*сигналы контроля *)
    state: STRING; (*режим управления*)
    res:BOOL; (*сброс регулятора*)
```

```
     (*УПРАВЛЕНИЕ И ВВОД*)
     y_ref_local:REAL:=0; (*локальное задание*)
     y_ref_update:BOOL:=FALSE; (*обновить задание*)
     man:BOOL:=FALSE; (*сигнал перехода на ручной режим*)
     do_reset:BOOL:=FALSE; (*сброс регулятора*)
     ERROR:BOOL:=FALSE; (*отказ панели*)
END_VAR
(*секция кода*)
IF NOT ERROR THEN
     y_ref:= y_ref_to_OP;
     y:= y_to_OP;
     err_gt_10:=err_gt_10_to_OP;
     err_lt_m10:=err_lt_m10_to_OP;
     norma:=norma_to_OP;
     res:=res_to_OP;
     CASE state_to_OP OF
       -1:
          state:='Авария';
          u:=u_to_OP;
     0:
          state:='Ручной';
     1:
          state:='Автомат';
          u:=u_to_OP;
     END_CASE
     y_ref_from_OP:=y_ref_local;
     y_ref_update_from_OP:=y_ref_update;
     u_manual_from_OP:=u;
     man_from_OP:=man;
     res_from_OP:=do_reset;
     tic_tac_from_OP:=NOT tic_tac_to_OP;
ELSE
     state:='Ошибка связи';
END_IF
```

```
PROGRAM PLC_PRG
VAR
     (*режим управления*)
     state:SINT:=0; (*-1 - авария, 0 - ручн., 1 - авт.*)
     (*таймеры связи*)
     OP_EXCHANGE_WATCHDOG_1, OP_EXCHANGE_WATCHDOG_0 : TON;
     (*таймер регулирования*)
     STOP_CONTROL_TIMER:TON;
     (*флаги ошибок связи*)
     OP_FAIL:BOOL:=FALSE;
END_VAR
(*Запись по интерфейсам*)
y_to_OP:=y;
y_ref_to_OP:=y_ref;
u_to_OP:=u;
state_to_OP:=state;
res_to_OP:=res;
(*Проверка связи с панелью*)
(*ретрансляция tic_tac*)
```

```
tic_tac_to_OP:= tic_tac_from_OP;
(*запуск таймеров*)
OP_EXCHANGE_WATCHDOG_1(IN:=tic_tac_from_OP, PT:=t#5s);
OP_EXCHANGE_WATCHDOG_0(IN:=NOT tic_tac_from_OP, PT:=t#5s);
(*установка флагов*)
OP_FAIL:= OP_EXCHANGE_WATCHDOG_1.Q OR OP_EXCHANGE_WATCHDOG_0.Q;
no_HMI:= OP_FAIL;
(*Обработка ошибок связи*)
IF OP_FAIL THEN
     y_ref_from_OP:=y_ref;
     y_ref_update_from_OP:=FALSE;
END_IF
(*Обновление задания*)
IF y_ref_update_from_OP  THEN
     y_ref:=y_ref_from_OP;
END_IF
stop:=FALSE; (*по умолчанию оборудование работает*)
(*Формирование сигнала сброса регулятора*)
res:= res_from_OP;
(*Формирование сигналов контроля для панели*)
IF y_ref - y  < -10 THEN
     err_gt_10_to_OP:=TRUE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=FALSE;
ELSIF y_ref - y  > 10 THEN
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=TRUE;
     norma_to_OP:=FALSE;
ELSE
     err_gt_10_to_OP:=FALSE;
     err_lt_m10_to_OP:=FALSE;
     norma_to_OP:=TRUE;
END_IF

(*Режимы управления*)
CASE state OF
-1: (*авария*)
          y_ref:=0;
          u_manual:=0;
          res:=TRUE;
          stop:=TRUE;
          IF reset_process THEN
               state:=0;
          END_IF
0: (*ручное управление*)
     u_manual:=u_manual_from_OP;
     man:=TRUE;
     (*переходы в другие режимы*)
     IF NOT man_from_OP OR OP_FAIL THEN
          state:=1;
     END_IF
1: (*автоматическое управление*)
     u_manual:=u;
```

```
     man:=FALSE;
(*переходы в другие режимы*)
     IF man_from_OP  THEN
          state:=0;
     END_IF
     (*Если нет связи с панелью, установка "безопасного" уровня
задания*)
     IF NO_HMI THEN
          y_ref:=50;
     END_IF
     (*Если связь с HMI утеряна и процесс в течение 1 мин. не уда-
ется вернуть в норму - авария*)
     STOP_CONTROL_TIMER(IN:=no_HMI AND (err_gt_10_to_OP OR
err_lt_m10_to_OP), PT:=t#1m);
     IF STOP_CONTROL_TIMER.Q THEN
          state:=-1;
     END_IF
END_CASE
(*Панель*)
PANEL;
```

3.1.4.2. Программная реализация законов регулирования с запаздыванием

Рассмотрим вопрос реализации транспортного запаздывания в программе ПЛК. Запаздывание используется во многих алгоритмах регулирования: в комбинированных системах с компенсацией возмущающего воздействия, в многосвязных системах с компенсацией перекрестных связей объекта, в адаптивных и робастных системах [19, 20]. В качестве тестового примера взята система на основе классического предиктора Смита [21, 22].

Для отладки алгоритмов и программных компонентов задействованы «комбинированная модель» на базе Simulink-модели объекта, «идеализированной» системы регулирования и виртуального контроллера PLC WinNT.

Предиктор Смита применяется для объектов с большим транспортным запаздыванием и позволяет искусственно вынести запаздывание из контура регулирования. Тем самым обеспечивается устойчивость контура и упрощается настройка регулятора. На рис. 3.113 показана структура системы.

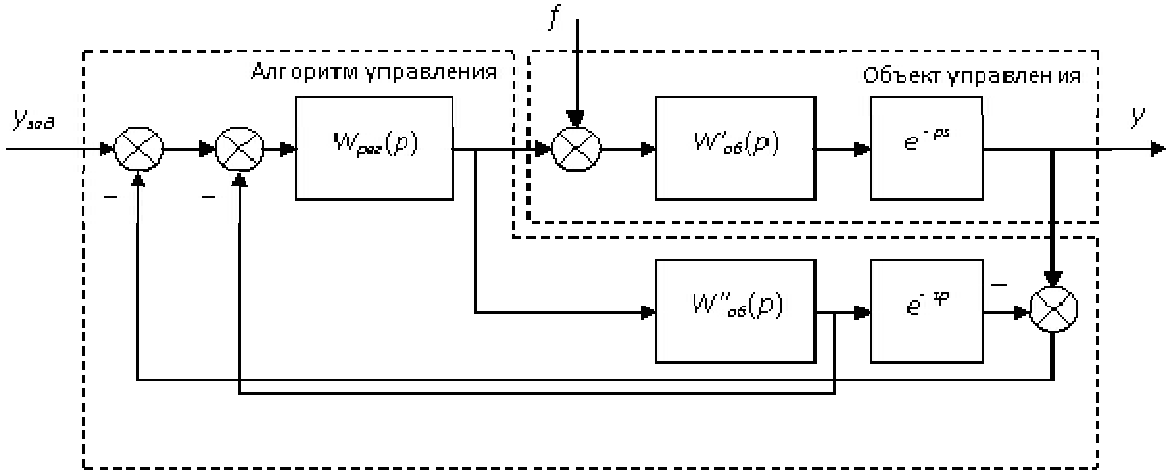


Рис. 3.113. Структура системы регулирования с предиктором Смита.

Объект управления представлен дробно-рациональной передаточной функцией $W'_{об}(p)$ и звеном запаздывания. Алгоритм управления включает модель объекта и регулятор с передаточной функцией $W_{рег}(p)$. Последний рассчитывается для объекта без учета запаздывания. Если модель объекта идеально точна ($W''_{об}(p) = W'_{об}(p)$, $\tau' = \tau$), сигнал на выходе первого сумматора равен нулю и передаточная функция замкнутой системы регулирования примет вид:

$$W_{зам}(p) = \frac{y}{y_{зад}} = \frac{W_{рег}(p)W'_{об}(p)}{1 + W_{рег}(p)W'_{об}(p)} e^{-\tau p}.$$

Таким образом, запаздывание не влияет на процесс, протекающий в контуре, а лишь сдвигает по времени отклик на задание. Аналогичный эффект может быть получен и при отработке возмущающего воздействия *f*, если в модель объекта в структуре алгоритма управления внести данный сигнал. Однако в нашем случае считается, что возмущение измерению не доступно.

Simulink-модель объекта и «идеальной» системы приведена на рис. 3.114.

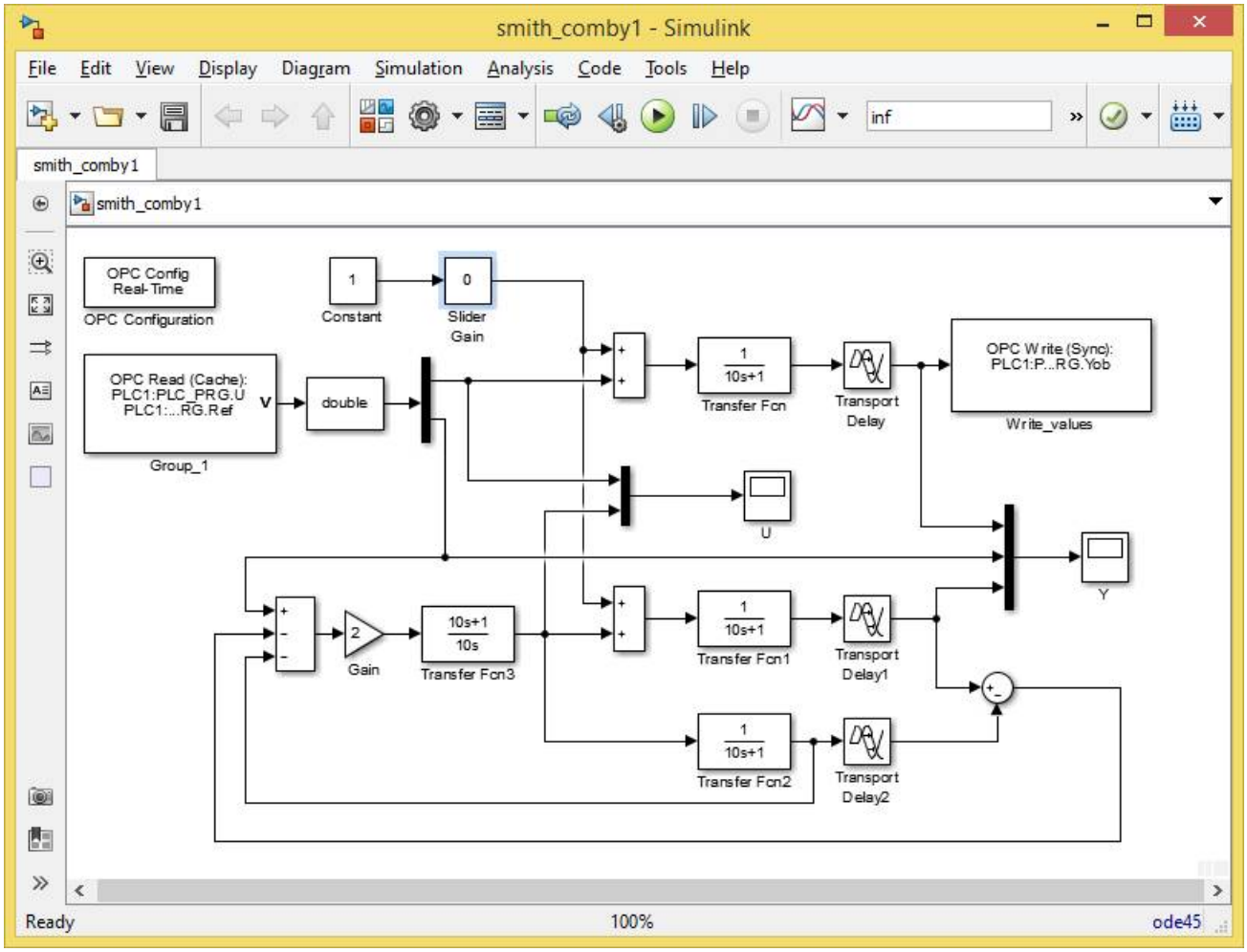


Рис. 3.114. Simulink-модель.

Передаточная функция объекта управления принята равной

$$W_{об}(p) = \frac{1}{10p + 1} e^{-10p}.$$

На «вариант» объекта в верхней части Simulink-диаграммы подается сумма сигналов возмущения и управления. Возмущение формируется блоком Slider Gain непосредственно в ходе моделирования, управление – блоком OPC Read, который считывает соответствующую переменную программы контроллера по протоколу OPC. Посредством блока OPC Write контроллеру передается значение регулируемой величины.

В нижней части диаграммы размещена «идеальная» система, поведение которой считается эталоном. От контроллера она получает сигнал задания.

Для регулятора выбран ПИ закон с коэффициентом передачи и постоянной времени интегрирования, равными 2 и 10 с соответственно:

$$W_{рег}(p) = 2\left(1 + \frac{1}{10p}\right) = 2\frac{10p+1}{10p}.$$

Передаточная функция «идеальной» системы по заданию:

$$W_{зам}(p) = \frac{W_{рег}(p)W'_{об}(p)}{1 + W_{рег}(p)W'_{об}(p)}e^{-\tau p} = \frac{1}{5p+1}e^{-10p}.$$

Первоначально программа регулирования для виртуального контроллера включала главную программу PLC_PRG, функциональный блок (ФБ) `MODEL`, представляющий модель $W''_{об}(p)$, и ФБ `DELAY`, формирующий временную задержку. `PLC_PRG` и `MODEL` составлены на языке CFC (рис. 3.115, 3.116).

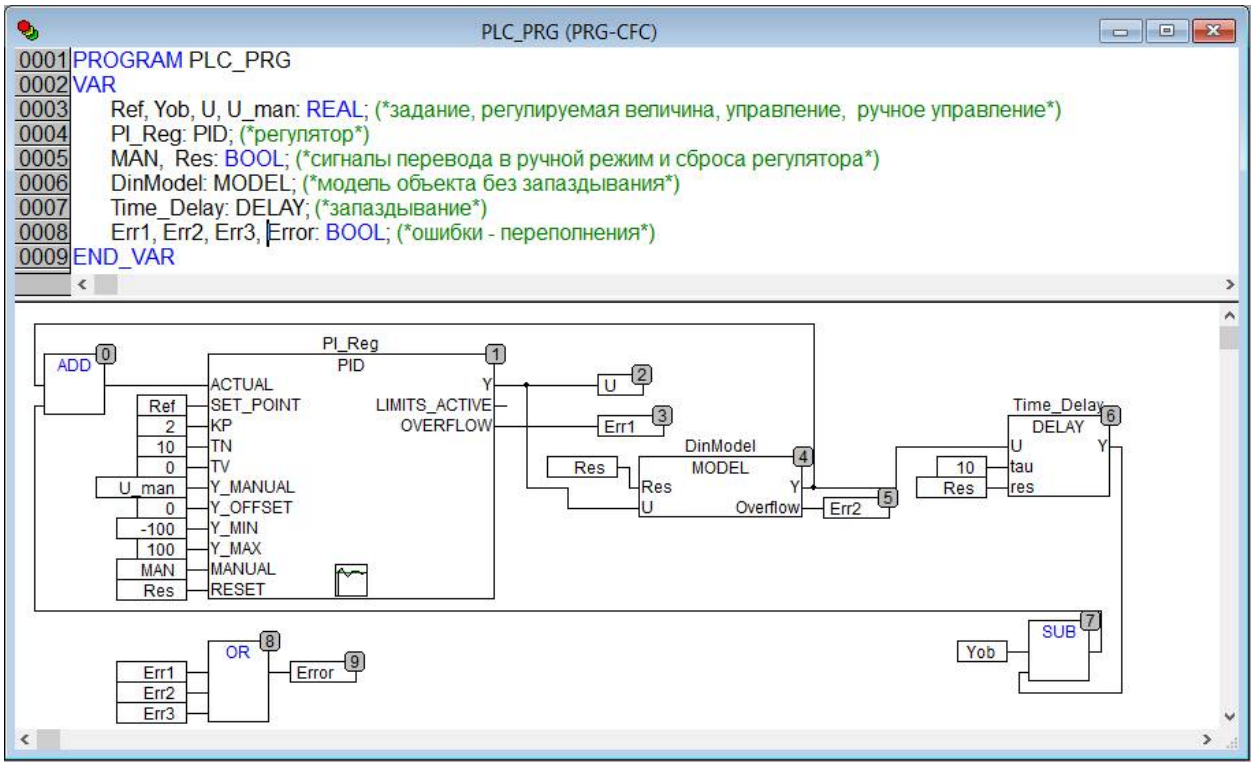


Рис. 3.115. Программа `PLC_PRG`.

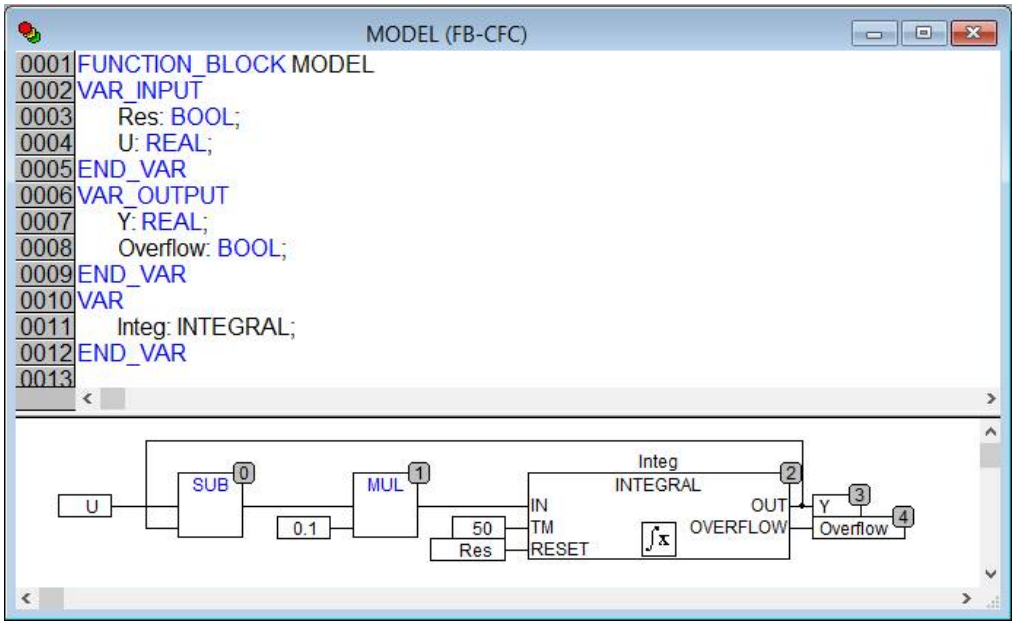


Рис. 3.116. Функциональный блок `MODEL`.

Программа `PLC_PRG` вызывается «по расписанию» каждые 50 мс.

В символьный файл (список переменных OPC-сервера) включены локальные переменные программы `PLC_PRG`: `Ref` (задание), `U` (сигнал управления) и `Yob` (регулируемая величина). Все они имеют тип REAL. Период межпрограммного обмена по OPC установлен равным 0,1 с.

Основная сложность заключается в программировании блока `DELAY`.

Принципиально возможны два способа реализации транспортного запаздывания в программе ПЛК:

1) с использованием аппроксимации запаздывания дробно-рациональной передаточной функцией (в основном применяется *аппроксимация Паде*);

2) с использованием циклического буфера для запоминания предыдущих отчетов запаздывающего сигнала.

Для *аппроксимации* запаздывания чаще всего выбираются передаточные функции второго и четвертого порядков:

$$W_2(p)=\frac{p^2-\frac{6}{\tau}p+\frac{12}{\tau^2}}{p^2+\frac{6}{\tau}p+\frac{12}{\tau^2}},\quad W_4(p)=\frac{p^4-\frac{20}{\tau}p^3+\frac{180}{\tau^2}p^2-\frac{840}{\tau^3}p+\frac{1680}{\tau^4}}{p^4+\frac{20}{\tau}p^3+\frac{180}{\tau^2}p^2+\frac{840}{\tau^3}p+\frac{1680}{\tau^4}},$$

где τ – время запаздывания.

Поскольку библиотеки сред программирования ПЛК не предоставляют готовых ФБ для реализации произвольных передаточных функций, рассмотрим построение соответствующих динамических звеньев на базе стандартных блоков интегрирования, суммирования и масштабирования. Для этого потребуется построить модели звеньев в пространстве состояний.

Передаточная функция второго порядка может быть представлена в виде:

$$W_2(p)=1-\frac{(12/\tau)p}{p^2+(6/\tau)p+(12/\tau^2)}=1-\frac{2a_1p}{p^2+a_1p+a_0}.$$

Структурная схема модели звена в пространстве состояний строится непосредственно по этому выражению, она показана на рис. 3.117.

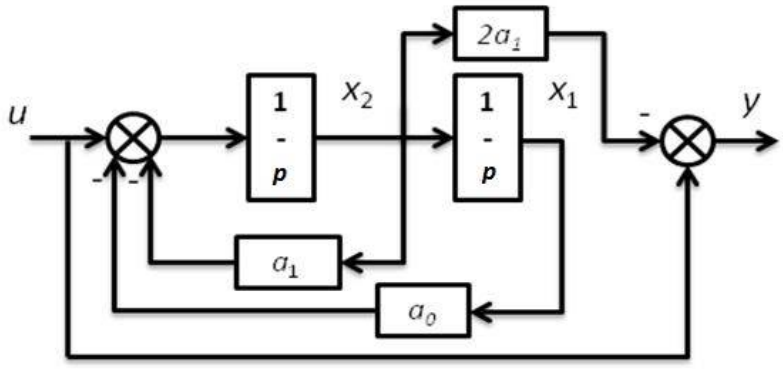


Рис. 3.117. Структурная схема аппроксимации Паде 2-го порядка.

Уравнения, связывающие переменные состояния, входной и выходной сигналы:

$$\begin{cases} x_1 = \int_0^t x_2(\tau)d\tau; \\ x_2 = \int_0^t \left(u - a_1 x_2(\tau) - a_0 x_1(\tau)\right)d\tau; \end{cases}$$
$$y = u - 2a_1 x_2.$$

На основе этих уравнений в среде CoDeSys на языке ST разработан ФБ, код которого:

```
FUNCTION_BLOCK DELAY
VAR_INPUT
     U, tau:REAL;(*входной сигнал звена и время запаздывания*)
     res:BOOL; (*сигнал сброса*)
END_VAR
VAR_OUTPUT
     Y:REAL; (*выходной сигнал*)
END_VAR
VAR
     X1:REAL:=0; X2:REAL:=0;
     a0, a1:REAL;
     isInit:BOOL:=FALSE; (*признак выполнения инициализации*)
     Integ1, Integ2: INTEGRAL;
END_VAR
-------------------------------------------------------------------
(*Секция кода*)
IF NOT isInit THEN
     a0:=12/tau/tau; a1:= 6/tau; isInit:=TRUE;
END_IF
Integ1(IN:=X2, TM:=50, RESET:=res, OUT=>X1);
Integ2(IN:=U - a0*X1 - a1*X2, TM:=50, RESET:=res, OUT=>x2);
Y:=U - 2*a1*X2;
```

На рис. 3.118. показаны результаты моделирования – графики изменения регулируемой величины и управляющего воздействия при отработке ступенчато изменяющихся задания (от 0 до 40%) и возмущения (от 0 до 20%) для эталонной и исследуемой систем.

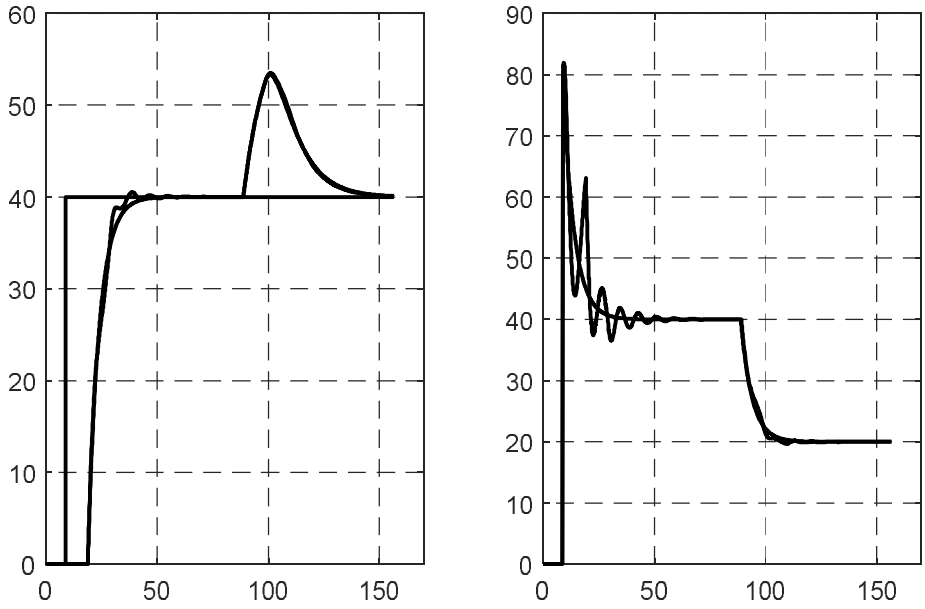


Рис. 3.118. Результаты моделирования: аппроксимация Паде 2-го порядка.

Передаточная функция четвертого порядка может быть представлена в виде:

$$W_4(s) = 1 - \frac{\frac{40}{\tau}s^3 + \frac{1680}{\tau^3}s}{s^4 + \frac{20}{\tau}s^3 + \frac{180}{\tau^2}s^2 + \frac{840}{\tau^3}s + \frac{1680}{\tau^4}} = 1 - \frac{2a_3s^3 + 2a_1s}{s^4 + a_3s^3 + a_2s^2 + a_1s + a_0}.$$

Структурная схема модели звена в пространстве состояний показана на рис. 3.119.

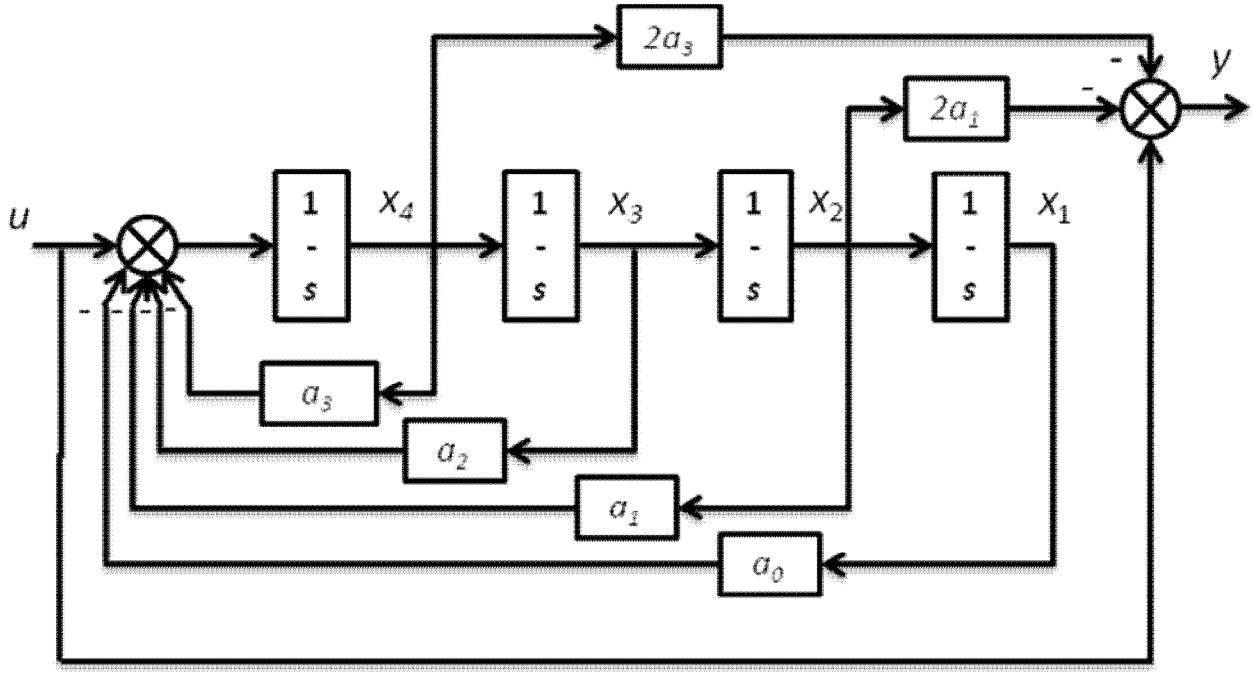


Рис. 3.119. Структурная схема аппроксимации Паде 4-го порядка.

Уравнения переменных состояния:

$$\begin{cases} x_1 = \int_0^t x_2(\tau)d\tau; \\ x_2 = \int_0^t x_3(\tau)d\tau; \\ x_3 = \int_0^t x_4(\tau)d\tau; \\ x_4 = \int_0^t \left(u - a_3 x_4(\tau) - a_2 x_3(\tau) - a_1 x_2(\tau) - a_0 x_1(\tau)\right)d\tau; \\ y = u - 2a_3 x_4 - 2a_1 x_2. \end{cases}$$

Код ФБ:

```
FUNCTION_BLOCK DELAY
VAR_INPUT
     U, tau:REAL;(*входной сигнал звена и время запаздывания*)
     res:BOOL; (*сигнал сброса*)
END_VAR
VAR_OUTPUT
     Y:REAL;
END_VAR
VAR
     X1:REAL:=0; X2:REAL:=0; X3:REAL:=0; X4:REAL:=0;
     a0,a1,a2,a3:REAL;
     isInit: BOOL:=FALSE;
     Integ1,Integ2, Integ3, Integ4: INTEGRAL;
END_VAR
-------------------------------------------------------------
(*Секция кода*)
IF NOT isInit THEN
     a0:=1680/tau/tau/tau/tau; a1:= 840/tau/tau/tau;
     a2:= 180/tau/tau; a3:= 20/tau;
     isInit:=TRUE;
END_IF
Integ1(IN:=X2, TM:=50, RESET:=res, OUT=>X1);
Integ2(IN:=X3, TM:=50, RESET:=res, OUT=>X2);
Integ3(IN:=X4, TM:=50, RESET:=res, OUT=>X3);
Integ4(IN:=U-a0*X1-a1*X2-a2*X3-a3*X4,TM:=50,RESET:=res,OUT=>X4);
Y:=U - 2*a1*X2 - 2*a3*X4;
```

Результаты моделирования показаны на рис. 3.120.

Как и следовало ожидать, аппроксимация 4-го порядка дает несколько лучшие результаты по сравнению со 2-ым порядком. Однако общим недостатком подхода, использующего аппроксимацию, является принципиальная неспособность воспроизведения транспортного запаздывания в случае действия на входе высокочастотного сигнала, в частности при ступенчато изменяющихся воздействиях.

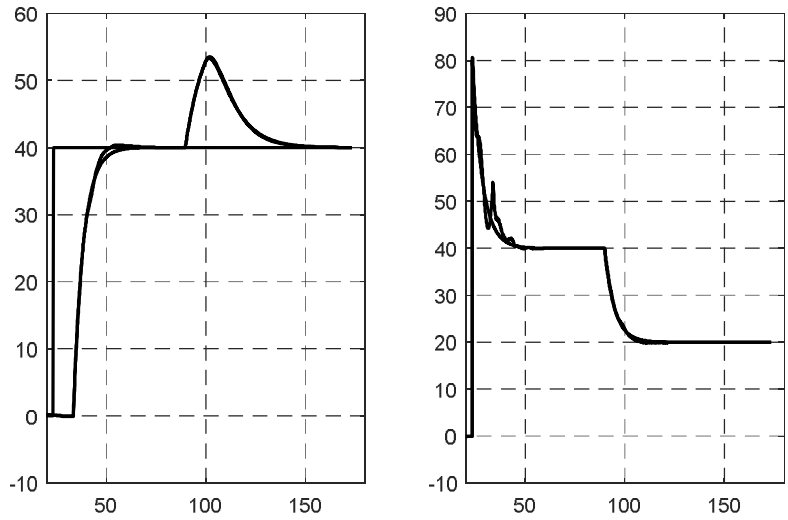


Рис. 3.120. Результаты моделирования: аппроксимация Паде 4-го порядка.

При реализации звена запаздывания с использованием *циклического буфера* могут быть применены разные подходы к организации процесса чтения и обновления буфера. Ниже рассмотрены варианты с применением таймера, с запоминанием времени в буфере, с использованием циклически вызываемой «задачи» (по сути – с прерыванием по времени).

Во всех подходах предполагается хранение «истории» изменения запаздывающей переменной с «глубиной», равной времени запаздывания, и определенным шагом по времени. Очевидно, что время запаздывания должно содержать целое число шагов. В этом смысле разумно задавать не шаг по времени, а размер буфера, тогда величину шага можно рассчитать на этапе инициализации функционального блока. Однако для пользователя, на наш взгляд, предпочтительнее все-таки задавать шаг по времени, поскольку его величина может быть легко сопоставлена с величиной самого запаздывания и скорректирована с учетом скорости протекания процесса. Поэтому в рассмотренных ниже реализациях фигурирует именно длительность шага. Число шагов (размер буфера) в таком случае вычисляется делением времени запаздывания на величину шага. Теоретически эту операцию можно выполнить при инициализации, а память для буфера выделить динамически, если это позволяет программная платформа. Однако использование динамической памяти в системах реального времени не приветствуется, а в нашем случае и не имеет особого смысла: пользователь, имея доступ к реализации функционального блока, и сам без труда может скорректировать размер буфера.

Чтение/обновление буфера по таймеру предполагает периодический запуск таймера на величину шага по времени. При срабатывании таймера:

если буфер еще не заполнен, в текущую ячейку записывается значение входного сигнала, а на выход выдается «начальное» значение звена запаздывания;

если буфер заполнен, из текущей ячейки считывается значение и выдается на выход блока, после чего в эту же ячейку записывается значение входного сигнала.

Текущий индекс в массиве-буфере изменяется «по кругу» с использованием операции «остаток от деления» на размер буфера.

Очевидный недостаток такого подхода состоит в том, что промежутки времени, необходимые для перезапуска таймера, накапливаются и увеличивают общее запаздывание. Проблему можно было бы решить уменьшением уставки таймера по сравнению с шагом по времени, но определить необходимую разницу для сколько бы то ни было сложных систем практически невозможно.

Поскольку алгоритм предполагает работу с таймером, выполнять его по расписанию было бы, по понятным причинам, неразумно. Поэтому программу, разработанную ранее, пришлось несколько модифицировать, рис. 3.121, 3.122. Модель объекта без запаздывания, ранее представленная экземпляром функционального блока `MODEL`, «вырезана» из `PLC_PRG` и оформлена в виде отдельной одноименной программы, которая по-прежнему будет вызываться циклически (потому что она содержит интеграторы). Программа `PLC_PRG` будет вызываться «свободно» в рамках второй задачи контроллера.

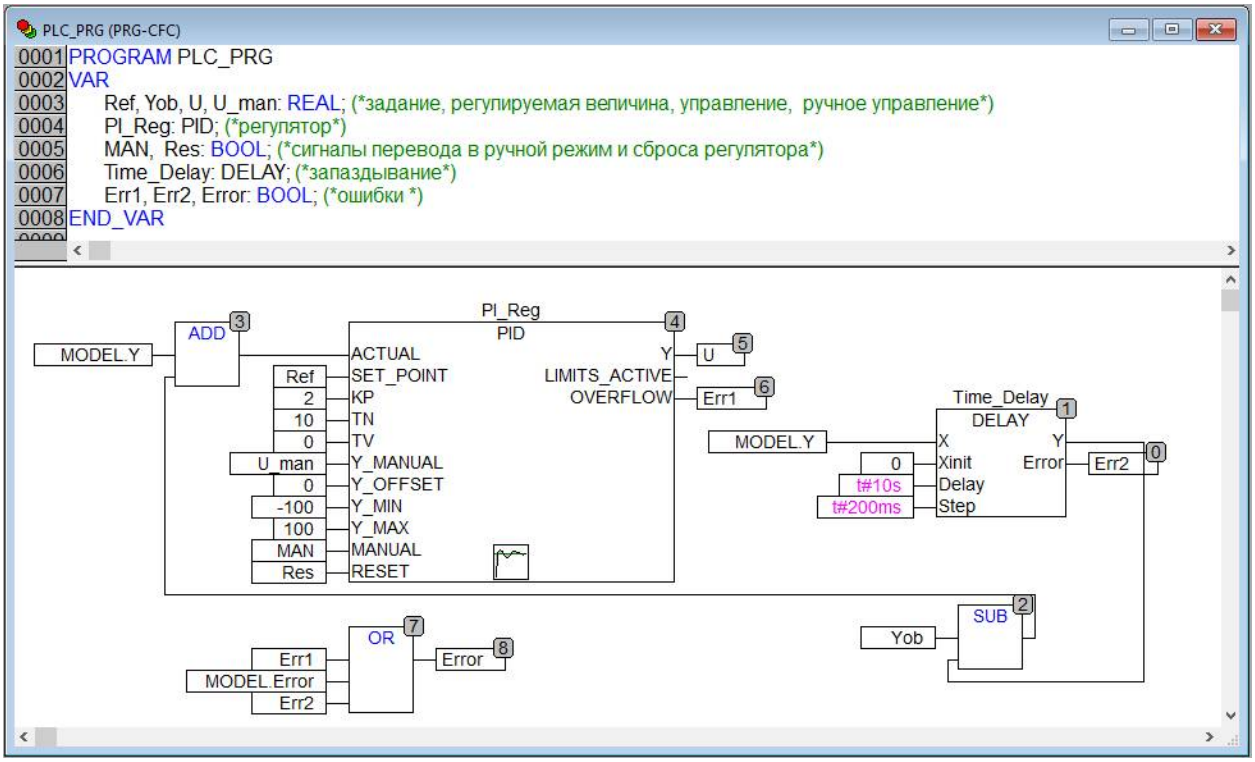


Рис. 3.121. Программа `PLC_PRG`.

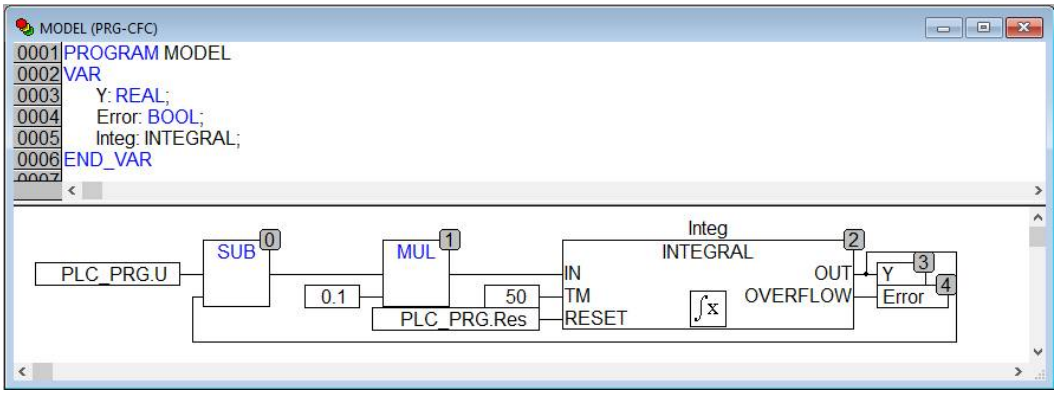


Рис. 3.122. Программа `MODEL`.

Код ФБ DELAY, в котором реализован алгоритм чтения/обновления буфера по таймеру:

```
FUNCTION_BLOCK DELAY
VAR_INPUT
     X, Xinit:REAL; (*входной сигнал и его начальное значение*)
     Delay, Step:TIME; (*время запаздывания и шаг по времени *)
END_VAR
VAR_OUTPUT
     Y:REAL; (*выходной сигнал*)
     Error:BOOL; (*признак ошибки*)
END_VAR
VAR
     buffer: ARRAY [0..100] OF REAL; (*буфер переменной*)
     current:DWORD:=0; (*место удаления/вставки*)
     count:DWORD:=1; (*количество элементов в буфере*)
     timer:TON; (*таймер*)
     bInit:BOOL; (*признак выполнения инициализации*)
     BufferSize:WORD;(*размер буфера*)
END_VAR
(*Секция кода*)
IF NOT bInit THEN
     BufferSize:=TIME_TO_WORD(Delay)/TIME_TO_WORD(Step);
     IF BufferSize<0 OR BufferSize>100 THEN
          Error:=TRUE; RETURN;
     END_IF
     bInit:=TRUE; buffer[0] := X;
END_IF
timer(IN:=TRUE,PT:=Step);
IF timer.Q THEN (*пора обновлять буфер*)
     timer(IN:=FALSE);
     IF count < BufferSize THEN (*буфер еще не заполнен*)
          Y:=XInit;
          buffer[count] := X; count:=count+1;
     ELSE
          Y:=buffer[current];
          buffer[current]:=X; current:=(current+1) MOD count;
     END_IF
END_IF
```

На рис. 3.123. показаны результаты моделирования.

Существенные отклонения поведения регулируемой величины и управляющего воздействия от эталона демонстрируют основной недостаток подхода.

Чтение/обновление буфера по времени требует наличие в системе часов реального времени и функции для работы с ними. Кроме того необходима фиксация моментов времени поступления накопленных значений входной величины, т.е., по сути, необходимо задействовать второй массив.

ФБ постоянно считывает текущее значение времени. Первоначальное заполнение буфера производится с заданным шагом. Далее разность текущего времени и времени запаздывания сравнивается со временем текущей ячейки буфера и, в случае превышения последнего, на выход посылается сохраненное значение и буфер обновляется.

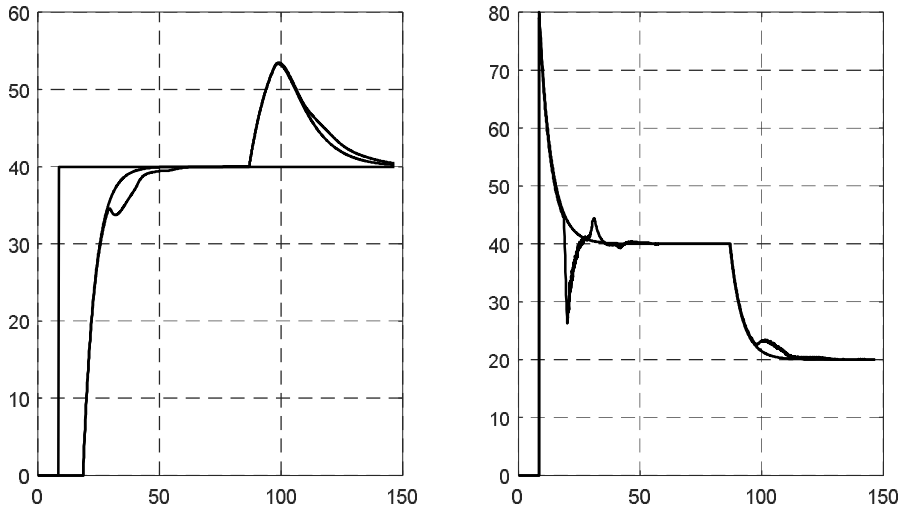


Рис. 3.123. Результаты моделирования: буфер с обновлением по таймеру.

Код ФБ:

```
FUNCTION_BLOCK DELAY
VAR_INPUT
     X,Xinit:REAL; (*входной сигнал и его начальное значение*)
     Delay, Step:TIME; (*время запаздывания и шаг по времени *)
END_VAR
VAR_OUTPUT
     Y:REAL; (*выходной сигнал*)
     Error:BOOL; (*признак ошибки*)
END_VAR
VAR
     buffer: ARRAY [0..100] OF REAL; (*буфер значений переменной*)
     t_buffer: ARRAY [0..100] OF TIME; (*буфер значений времени*)
     current:DWORD:=0; (*место удаления/вставки*)
     count:DWORD:=1; (*количество элементов в буфере*)
     bInit:BOOL; (*признак выполнения инициализации*)
     BufferSize:WORD;(*размер буфера*)
     t: TIME; (*текущее время*)
     buf_ful: BOOL:=FALSE; (*признак заполнения буфера*)
END_VAR
(*Секция кода*)
IF NOT bInit THEN
     BufferSize:=TIME_TO_WORD(Delay)/TIME_TO_WORD(Step);
     IF BufferSize<0 OR BufferSize>100 THEN
          Error:=TRUE; RETURN;
     END_IF
     bInit:=TRUE; buffer[0] := X;
     t_buffer[0]:=TIME(); Y:=XInit;
END_IF
t:=TIME();
IF NOT buf_ful THEN(*буфер еще не заполнен*)
     IF (t-t_buffer[count-1])>=Step THEN
          Y:=XInit; buffer[count] := X;
          t_buffer[count]:=t; count:=count+1;
     END_IF
END_IF
```

```
IF (t-Delay)>=t_buffer[current] THEN (*пора обновлять буфер*)
     buf_ful:=TRUE;
     Y:=buffer[current]; buffer[current]:=X;
     t_buffer[current]:=t; current:=(current+1) MOD count;
END_IF
```

Основная погрешность метода связана неравномерностью шага по времени. Вследствие накопления ошибки время постоянно «сдвигается» к концу буфера и последний шаг получается существенно короче предыдущих. Несмотря на это, результаты моделирования (рис. 3.124) оказались лучше, чем при использовании подходов, рассмотренных выше. Существенным недостатком является повышение требований к ресурсам аппаратной системы: объему памяти и времени выполнения кода.

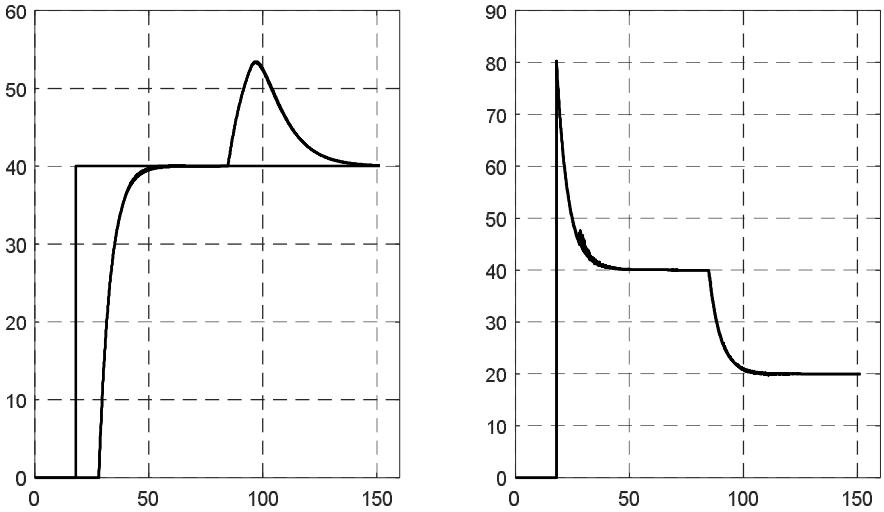


Рис. 3.124. Результаты моделирования: буфер с обновлением по времени.

Чтение/обновление буфера с использовании циклически вызываемой задачи предполагает, что алгоритм будет выполняться периодически, через строго определенное время. Для этого можно преобразовать функциональный блок `DELAY` в программу и вызывать ее циклически, как это было сделано с программой `MODEL`. Однако мы пойдем более простым путем. Объявим глобальную переменную `DelayUpdateEnable` типа `BOOL`, которая будет разрешать пересчет функционального блока `DELAY` (рис. 3.125).

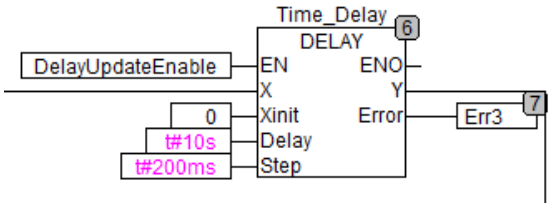


Рис. 3.125. Экземпляр `DELAY` в `PLC_PRG`.

Сама переменная `DelayUpdateEnable` будет устанавливаться в `TRUE` в специальной программе `Cyclic`, содержащей всего одну строку кода:

```
DelayUpdateEnable:=TRUE;
```

Эта программа будет вызываться циклически (третья задача контроллера) с требуемым периодом.

В `FALSE` переменную `DelayUpdateEnable` сбросит сам ФБ `DELAY`.

Описанный подход демонстрирует, конечно, не самый лучший стиль программирования, зато программа потребовала минимальной переделки.

Код ФБ DELAY:

```
FUNCTION_BLOCK DELAY
VAR_INPUT
     X,Xinit:REAL; (*входной сигнал и его начальное значение*)
     Delay, Step:TIME; (*время запаздывания и шаг по времени *)
END_VAR
VAR_OUTPUT
     Y:REAL; (*выходной сигнал*)
     Error:BOOL; (*признак ошибки*)
END_VAR
VAR
     buffer: ARRAY [0..100] OF REAL; (*буфер переменной*)
     current:DWORD:=0; (*место удаления/вставки*)
     count:DWORD:=0; (*количество элементов в буфере*)
     bInit:BOOL; (*признак выполнения инициализации*)
     BufferSize:WORD;(*размер буфера*)
END_VAR
(*Секция кода*)
IF NOT bInit THEN
     BufferSize:=TIME_TO_WORD(Delay)/TIME_TO_WORD(Step);
     IF BufferSize<0 OR BufferSize>100 THEN
          Error:=TRUE; RETURN;
     END_IF
     bInit:=TRUE; Y:=XInit;
END_IF
IF count < BufferSize THEN(*буфер еще не заполнен*)
     Y:=XInit;
     buffer[count] := X; count:=count+1;
ELSE
     Y:=buffer[current];
     buffer[current]:=X; current:=(current+1) MOD count;
END_IF
DelayUpdateEnable:=FALSE;
```

Результаты моделирования показаны на рис. 3.126.

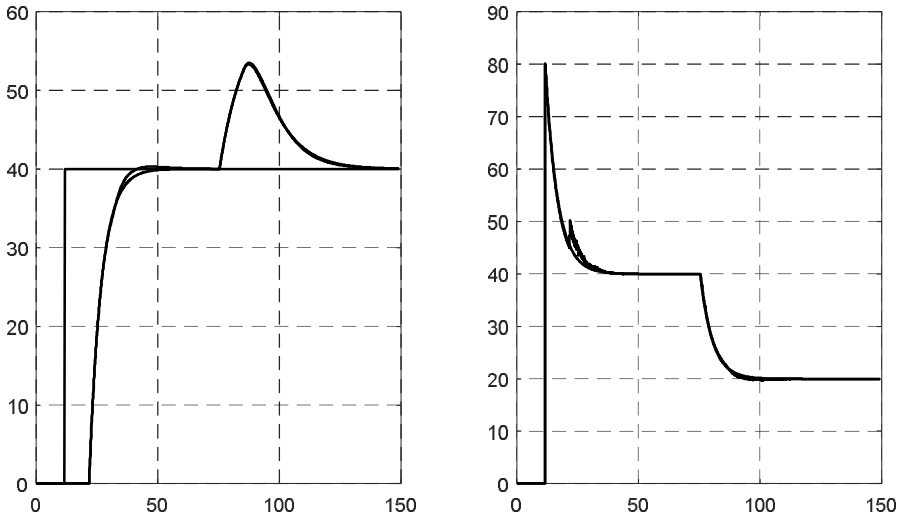


Рис. 3.126. Результаты моделирования: буфер с обновлением по прерыванию

3.2. Системы программно-логического управления

3.2.1. Конечный автомат – система управления процессом дозирования

3.2.1.1. Механизмы и электрическая схема

На рис. 3.127 показана схема системы загрузки сыпучего материала в тележку.

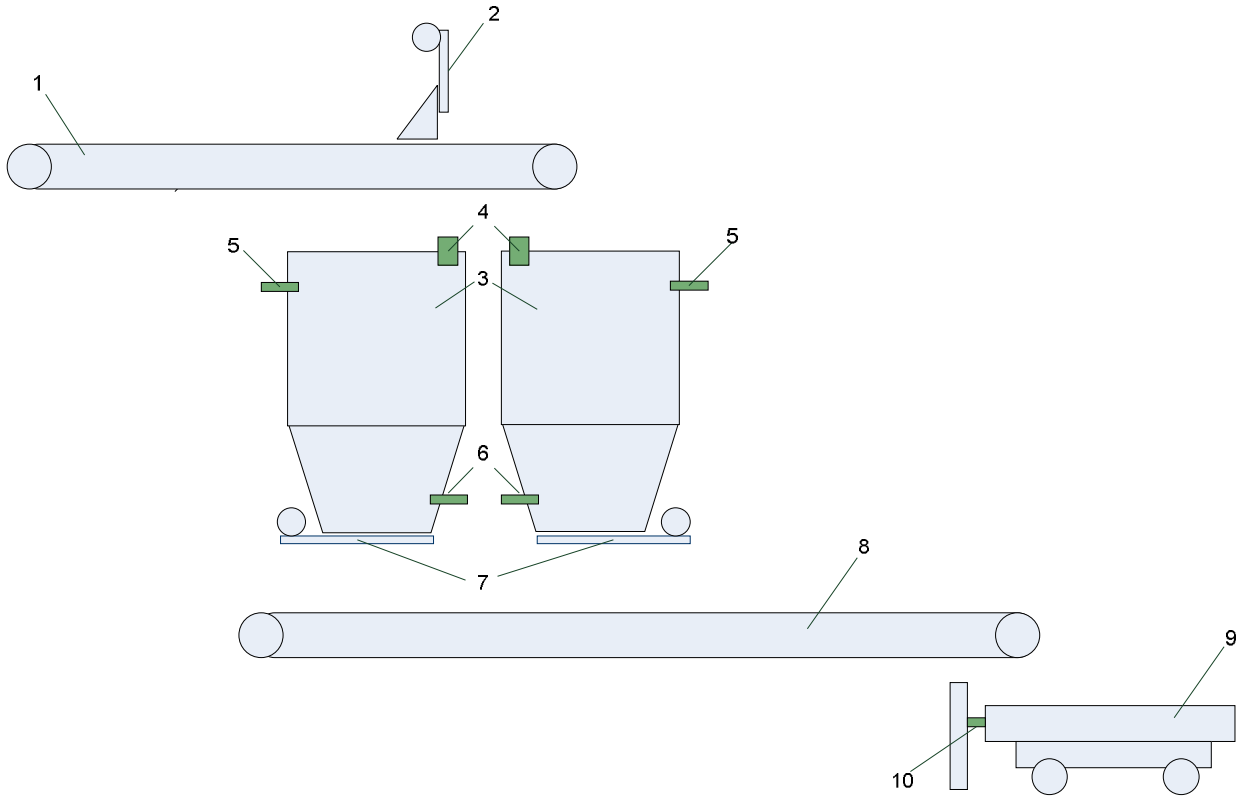


Рис. 3.127. Схема системы:

1 – подающий транспортер; 2 – плужковый сбрасыватель с электроприводом (привод снабжен концевыми выключателями, срабатывающими в положениях «плужок поднят» и «плужок опущен»); 3 – бункеры; 4 – измерительные преобразователи уровня (например, ультразвуковые или электромагнитные); 5 – контактные датчики верхнего уровня, формирующие сигналы «бункер полон»; 6 – контактные датчики нижнего уровня, формирующие сигналы «бункер пуст»; 7 – шиберные задвижки с электроприводами. Приводы снабжены концевыми выключателями, срабатывающими в положениях «задвижка закрыта» и «задвижка открыта»; 8 – разгрузочный транспортер; 9 – тележка; 10 – механизм фиксации тележки с контактным датчиком, подтверждающим фиксацию.

Загрузка сыпучего материала в тележку производится из одного из двух бункеров, причем одновременно может производиться наполнение другого бункера. Производительность подающего транспортера больше, чем разгрузочного, поэтому после окончания загрузки очередной тележки загрузка следующей может быть начата незамедлительно.

Сразу отметим два обстоятельства.

Во-первых, одного загрузочного транспортера явно недостаточно в том смысле, что требуется еще и какая-то подсистема «наполнения» этого транспортера сыпучим материалом. Мы эту подсистему не рассматриваем, считая, что одновременно с включением транспортера включается в работу и эта под-

система, а при выключении – выключается, не допуская «завала». Пусть, например, это будет бункер с шнековым разгрузчиком, привод которого одновременно получает питание с приводом транспортера. В таких условиях нам вообще незачем знать о подсистеме загрузки транспортера.

Во-вторых, ситуация длительного отсутствия тележки является *нормальной*, т.е. тележка, вообще-то говоря, не является частью системы. При отсутствии тележки, система должна просто ждать ее появления.

Рассмотрим гипотетическую принципиальную электрическую схему соединений.

На рис. 3.128 показана схема силовой части.

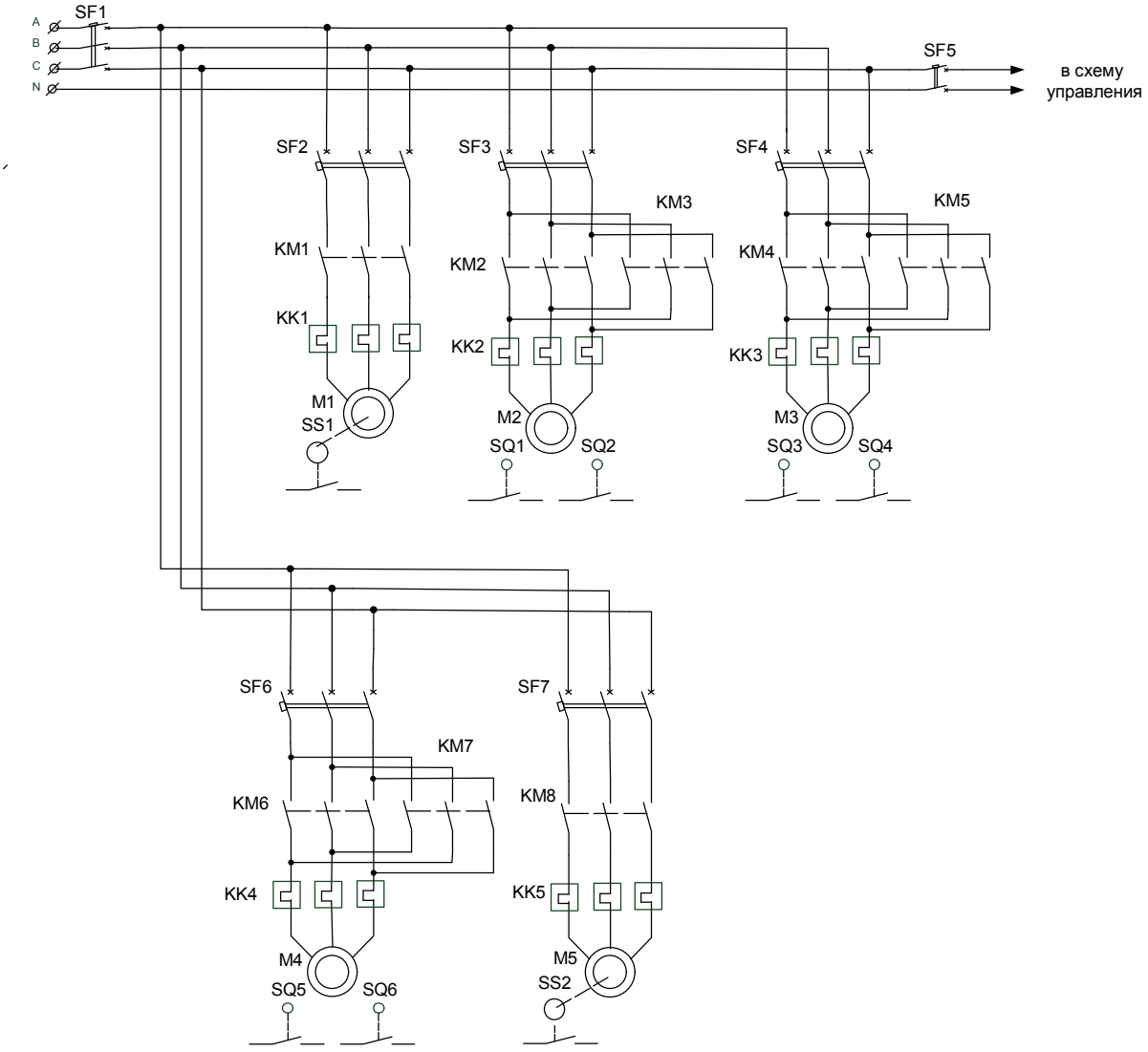


Рис. 3.128. Принципиальная электрическая схема силовой части системы.

Двигатели приводов подающего транспортера, плужкового сбрасывателя, шиберов бункеров и разгрузочного транспортера обозначены как М1, М2, М3, М4 и М5 соответственно. При этом М1 и М5 коммутируются нереверсивными пускателями КМ1 и КМ8, а М2, М3 и М4 – реверсивными, т.е. парами: КМ2 и КМ3, КМ4 и КМ5, КМ6 и КМ7. Каждый двигатель защищен «собственным» автоматическим выключателем от сверхтоков и тепловым реле – от перегрузки. Нереверсивные привода транспортеров оборудованы датчиками движения (на-

пример, датчиками контроля движения ленты) SS1 и SS2. Приводы плужкового сбрасывателя и шиберов снабжены концевыми выключателями SQ1 – SQ6.

Схема управления условно разбита на три части, показанные на рис. 3.129, 3.130 и 3.132.

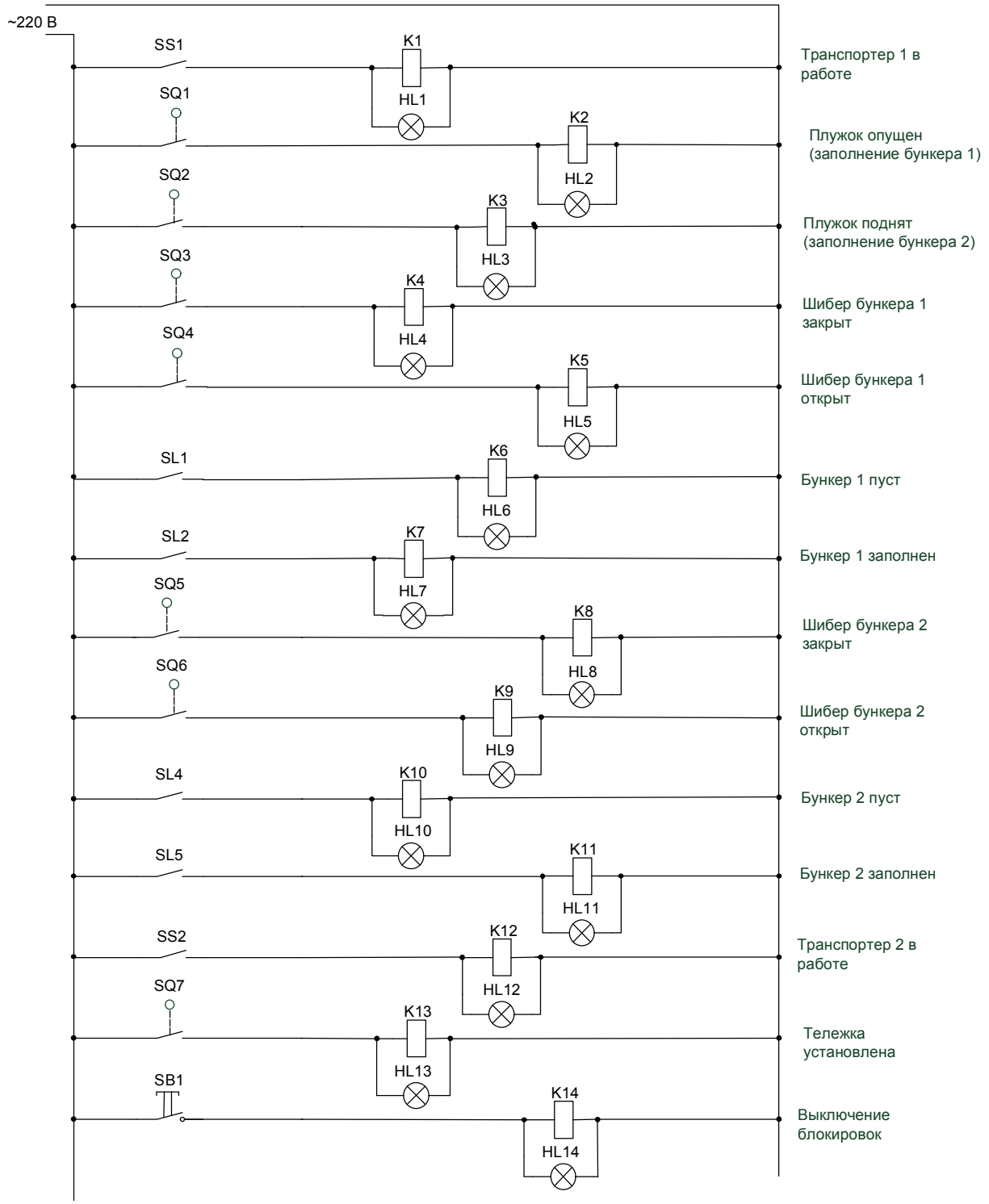


Рис. 3.129. Принципиальная электрическая схема управления, часть 1.

Первая часть занимается исключительно «размножением» дискретных сигналов датчиков с помощью промежуточных реле К1 – К14, шунтируемых сигнальными лампами HL1 – HL14, которые предполагается установить на щите управления.

Необходимость в этой части схемы чисто техническая: обычно датчики имеют только одну пару контактов – замыкающий и размыкающий (или один перекидной), изменяющих свое состояние при срабатывании датчика. Нам же для реализации различных *блокировок*, о которых речь впереди, и оповещения ПЛК потребуется больше контактов.

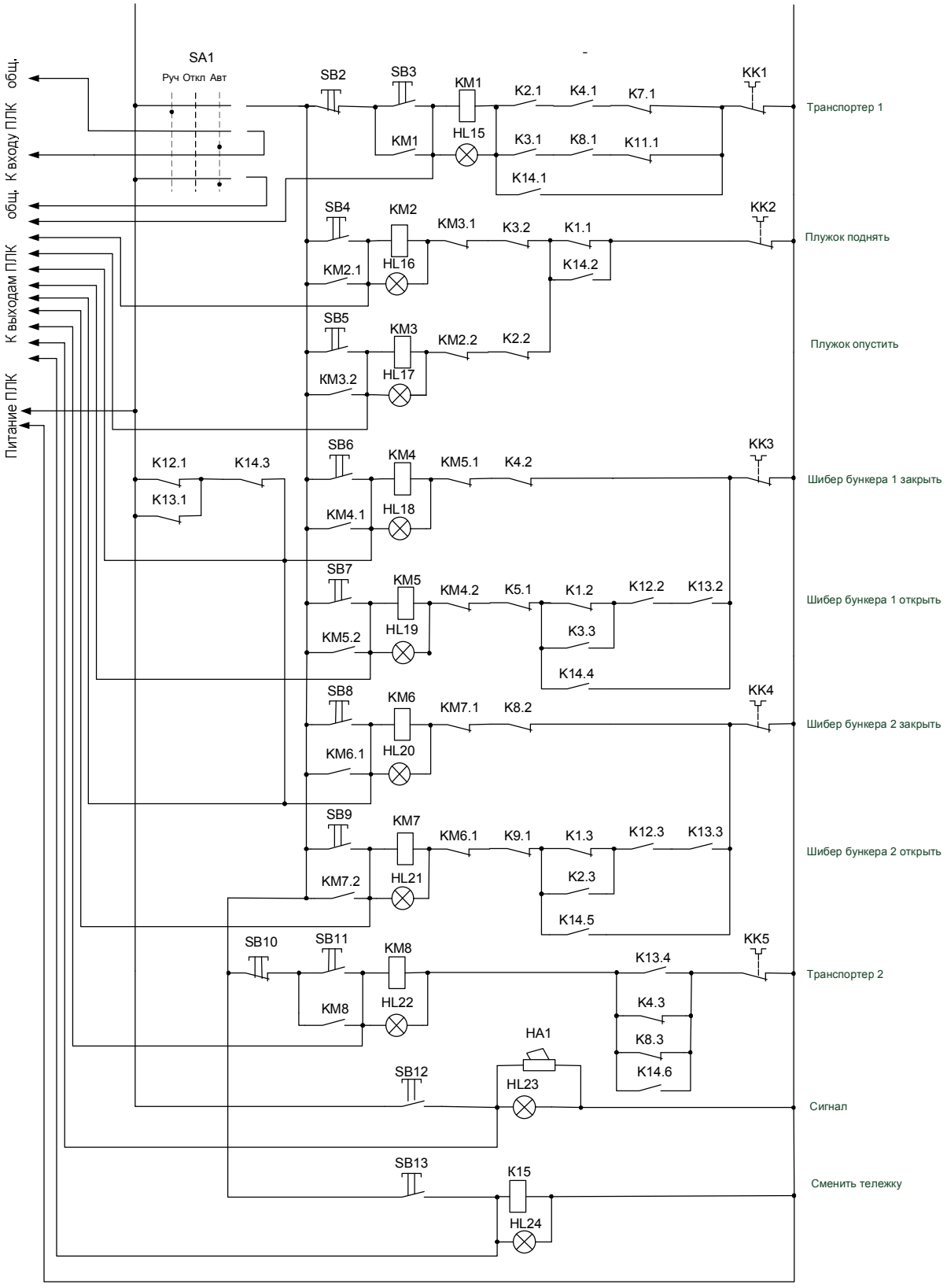


Рис. 3.130. Принципиальная электрическая схема управления, часть 2.

Требует пояснения последняя цепь этой части схемы. В ней установлена кнопка с фиксацией, отключающая блокировки. Сигнальная лампа HL24 может и не являться самостоятельным устройством, а быть «частью» этой кнопки (кнопка с подсветкой).

Блокировки выполняют защитные функции, не позволяя оператору производить потенциально опасные или просто нежелательные действия в ручном режиме управления. Помимо этого, блокировки способны самостоятельно (без оператора) решать некоторые задачи, например, приводить систему в безопасное состояние. Однако кроме ручного, часто реализуется также и т.н. *ремонтный*, или отладочный режим, когда блокировки должны быть отключены и оператору должна быть предоставлена полная свобода действий. Понятно, что такой режим не является «нормальным», и не каждому оператору можно доверить управление в таком режиме. Поэтому неплохо было бы защитить кнопку ключом (кнопка с ключом), или спрятать ее внутри шкафа управления, а сам шкаф – запирать на ключ.

Вторую часть схемы рассмотрим подробнее. В ней присутствует трехпозиционный переключатель SA1, с помощью которого систему можно переводить в ручной и автоматический режим управления, а также отключить вовсе. В ручном режиме все цепи получают питание от левой шины (верхняя цепь переключателя), в автоматическом режиме левая шина соединяется с общей клеммой дискретных выходов ПЛК, позволяя последним управлять аппаратурой (нижняя цепь переключателя).

Переключатель SA1 также занимается «информированием» ПЛК о режиме работы: в автоматическом режиме на один из входов ПЛК подается соответствующий сигнал (средняя цепь переключателя). Программа ПЛК всегда должна знать о режиме управления. В ручном режиме дискретные выходы контроллера должны быть переведены в безопасное состояние (просто «выключены»), чтобы исключить создание так называемых *ложных цепей*, которые могут возникнуть через общую клемму дискретных выходов.

Рассмотрим для примера, следующую ситуацию (рис. 3.131).

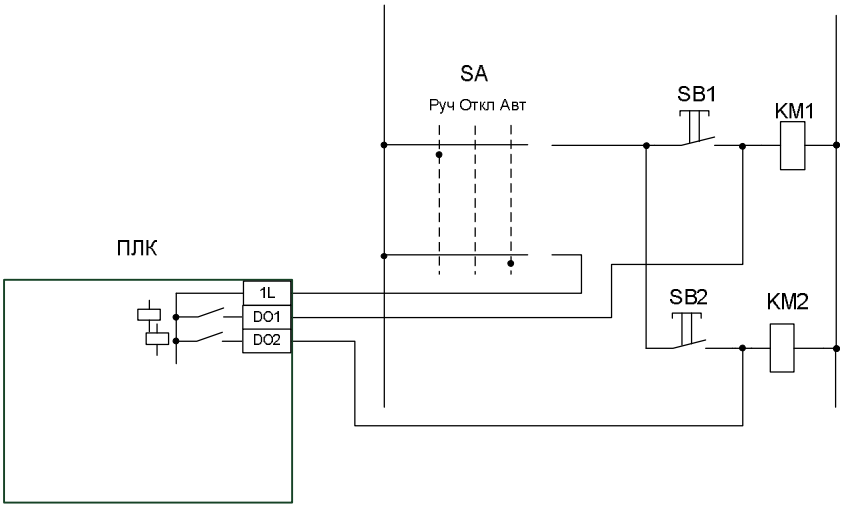


Рис. 3.131. К проблеме ложных цепей.

На рисунке упрощенно показан фрагмент некоторой схемы. В ручном режиме левая шина подключается к кнопкам SB1 и SB2, с помощью которых оператор может подать напряжение на катушки пускателей КМ1 и КМ2. В автоматическом режиме левая шина подключена к общей клемме дискретных выходов ПЛК, и управлять пускателями будут контакты релейных выходов контроллера. Как будто бы все правильно. Однако представим себе ситуацию, когда в ручном режиме контроллер, следуя логике программы автоматического управления, активирует сразу два своих выхода. В это время оператор хочет запустить первый механизм и нажимает на кнопку SB1. Катушка КМ1 получает питание. Но через внутренний общий провод выходов ПЛК напряжение попадает также и на катушку пускателя КМ2. В результате вместо одного механизма оператор запускает сразу два! Вряд ли он будет в восторге от случившегося. Если бы контроллер знал, что не он управляет процессом и выключил оба выхода, ложная цепь бы не возникла.

Кроме исключения ложных цепей информирование контроллера о режиме работы позволяет ему «синхронизироваться» в общем потоке управления, например, перейти к выполнению некоторой секции кода при переходе в автоматический режим.

Продолжим рассмотрение схемы, приведенной на рис. 3.130. Цепи катушек пускателей приводов транспортеров снабжены двумя кнопками: «Пуск» и «Стоп». Цепи катушек приводов плужкового сбрасыватели и шиберов – кнопками «Поднять»/«Опустить», «Закрыть»/«Открыть». При этом отключение этих приводов происходит при достижении конечных положений концевыми выключателями, а точнее размыкающими контактами соответствующих промежуточных реле. Все «замыкающие» кнопки оснащены подсветкой, которая на схеме показана в виде сигнальных ламп HL15 – HL22. Все магнитные пускатели при срабатывании шунтируют кнопки пуска своими замыкающими контактами.

Кнопка SB12 предназначена для подачи звукового и светового сигналов в случае аварии и в цели предупреждения о запуске механизмов. Она «действует» в любом режиме управления.

С помощью кнопки SB13 с подсветкой HL24 подается сигнал запроса на смену тележки в ручном режиме. Запрос состоит в замыкании контакта реле К15 в некой внешней цепи. Если тележка заменяется вручную, то это цепь сигнализации, если автоматически – в цепи управления механизмами смены тележки (которые мы не рассматриваем).

В схеме задействованы следующие блокировки:

1. Подающий транспортер (комплекс подающих устройств) может быть включен, если

плужок опущен (К2.1), *и* шибер бункера 1 закрыт (К4.1), *и* бункер 1 *не* заполнен (К7.1), *или*

плужок поднят (К3.1), *и* шибер бункера 2 закрыт (К8.1), *и* бункер 2 *не* заполнен (К11.1).

Блокировка может быть отключена контактом реле К14.1.

2. Блокировка от одновременного включения пускателей КМ2 и КМ3 (поднять и опустить плужок) – контактами КМ3.1 и КМ2.2. Одновременное включение привело бы к короткому замыканию в силовой цепи, поэтому данная блокировка не отключается.

3. Запрет включения привода плужкового сбрасывателя при работающем загрузочном транспортере (К1.1). Блокировка может быть отключена контактом реле К14.2.

4. «Принудительное» закрытие шиберов 1 и 2 при *не*работающем транспортере разгрузки (К12.1) *или* в отсутствие тележки (К13.1) как в ручном, так и автоматическом режиме. Блокировка может быть отключена контактом реле К14.3.

5. Блокировка от одновременного включения пускателей КМ4 и КМ5 (закрыть и открыть шибер бункера 1) – контактами КМ5.1 и КМ4.2. Одновременное включение привело бы к короткому замыканию в силовой цепи, поэтому данная блокировка не отключается;

6. Привод шибера бункера 1 может быть включен на открытие, только если

подающий транспортер *не* работает (К1.2), *или* плужок поднят (К3.3), т.е. загрузка не осуществляется вовсе или осуществляется, но в другой бункер,

и при этом разгрузочный транспортер работает (К12.2), *и* тележка на месте (К13.2). Блокировка может быть отключена контактом реле К14.4.

7. Блокировка от одновременного включения пускателей КМ6 и КМ7 (закрыть и открыть шибер бункера 2) – контактами КМ7.1 и КМ6.2. Одновременное включение привело бы к короткому замыканию в силовой цепи, поэтому данная блокировка не отключается.

8. Привод шибера бункера 2 может быть включен на открытие, только если

подающий транспортер *не* работает (К1.3), *или* плужок опущен (К2.3), т.е. загрузка не осуществляется вовсе или осуществляется, но в другой бункер,

и при этом разгрузочный транспортер работает (К12.3) и тележка на месте (К13.3). Блокировка может быть отключена контактом реле К14.5.

9. Разгрузочный транспортер может быть включен, если тележка на месте (К13.4), *или не* закрыты шиберы бункеров (К4.3, К8.3). При внезапном «исчезновении» тележки транспортер продолжит работу, пока не закроются шиберы бункеров. «Завал» транспортера считается «худшим злом», чем просыпка материала «на пол». Блокировка может быть отключена контактом реле К14.6.

На рис. 3.132 показана схема подключения ПЛК.

На дискретные входы контроллера поступают сигналы о срабатывания 14 реле и 8 пускателей. В автоматическом режиме пускателями управляет сам контроллер, поэтому может показаться излишним заводить в него информацию об их состоянии. Однако резоны для этого есть. Во-первых, в контроллере программным способом можно организовать диагностику исправности электрической схемы. Во-вторых, появляется возможность наблюдать за действиями оператора в ручном режиме.

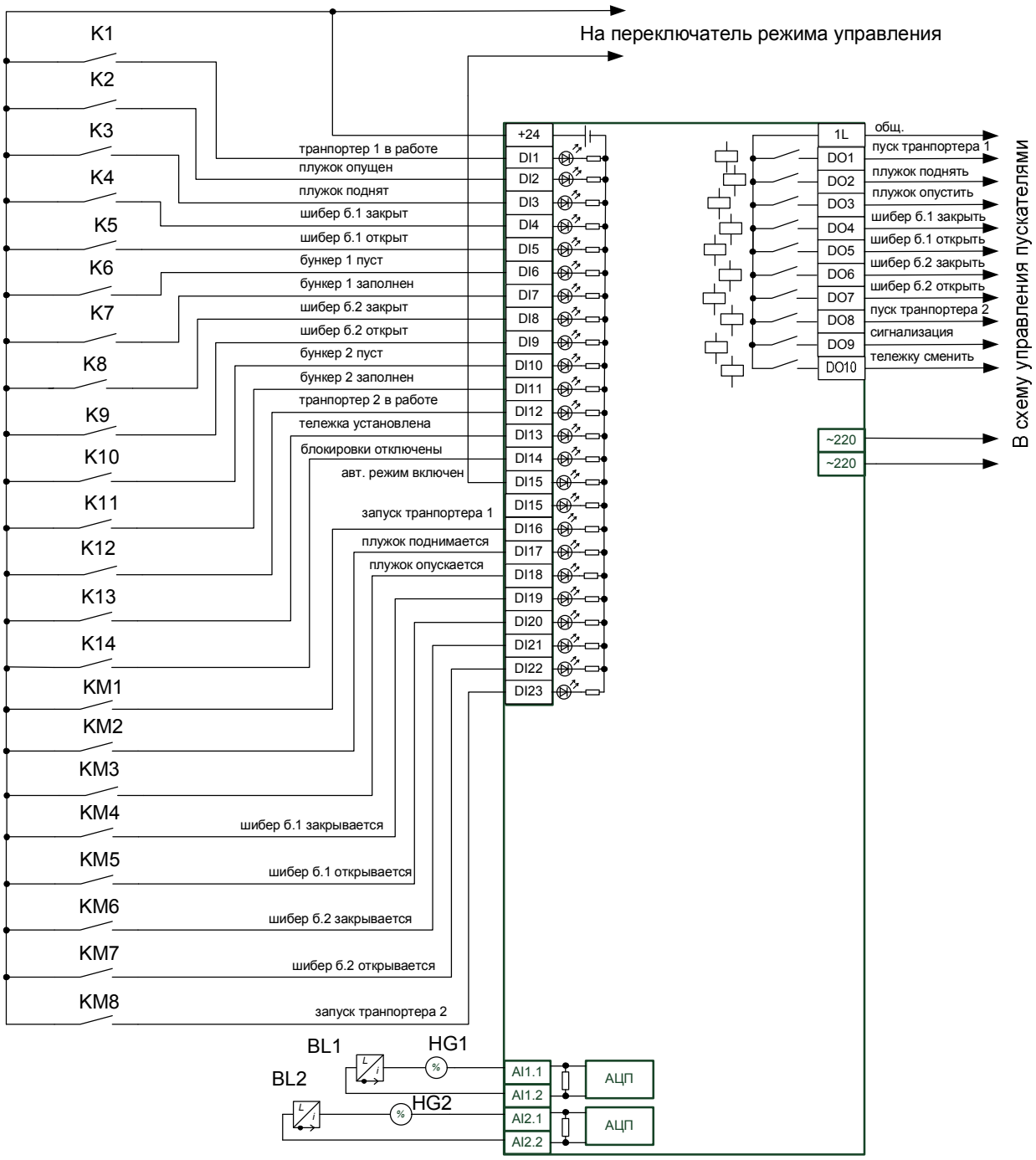


Рис. 3.132. Принципиальная электрическая схема управления, часть 3.

Питание всех дискретных входов производится от встроенного источника самого контроллера. Входы потребляют небольшой ток (порядка нескольких миллиампер), поэтому его мощности всегда достаточно.

На аналоговые входы заводятся сигналы с измерителей уровня. Здесь мы полагаем, что измерительные преобразователи самостоятельно генерируют токовые сигналы, поэтому во внешнем источнике питания нет необходимости. В цепях установлены индикаторы токовой петли, настроенные на показание уровня в %. Индикаторы монтируются на щите управления.

Все дискретные выходы контроллера – релейного типа. В нашем гипотетическом контроллере все 10 выходов сгруппированы «внутри». В реальных

случаях выходы могут не группироваться вообще (индивидуальные) или быть сгруппированы в несколько небольших групп (например, по 4 выхода). Тогда потребуется «внешняя» группировка индивидуальных выходов или их групп.

3.2.1.2. Разработка Simulink-модели объекта

В системе имитационного моделирования Simulink можно моделировать любые динамические процессы (т.е. процессы, происходящие во времени): как непрерывные, так и дискретные. Если какой-либо алгоритм преобразования входов в выходы трудно составить на графическом языке (в виде соединения библиотечных блоков), всегда можно воспользоваться инструментами из раздела User-Defined Function, например Interpreted MATLAB Function, и написать программу на «нормальном» процедурном языке. Мы так уже делали раньше, но в данном примере и этого не потребовалось. На рис. 3.133 показана полная Simulink-модель объекта.

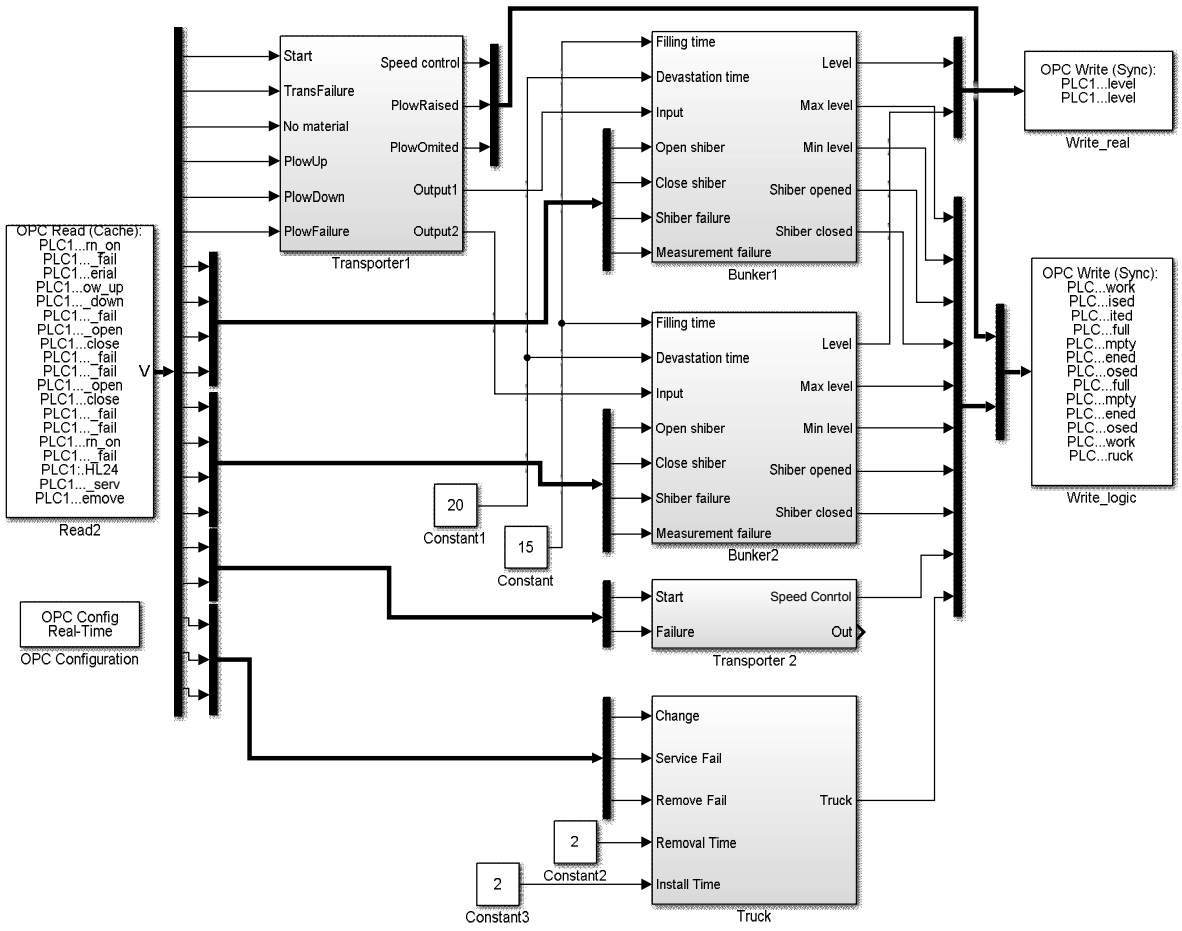


Рис. 3.133. Полная Simulink-модель объекта управления.

Прежде чем приступить к рассмотрению элементов модели, требуется определиться с составом переменных обмена по OPC.

Входными для модели будут переменные, связанные с работой магнитных пускателей и реле, формирующего запрос на замену тележки. Дополнительно введены переменные для ввода ошибок (неисправностей). Конечно, неисправности возникают на объекте, поэтому неплохо было бы формировать их

в самой Simulink-модели, как это мы делали ранее с возмущениями. Так вполне можно было и поступить, однако это усложнит модель и создаст неудобство, связанное с частыми переключениями между окнами в процессе отладки. В связи с этим принято решение формировать ошибки с помощью специального отладочного окна визуализации в CoDeSys. Если потом потребуется действительно реализовать систему и перенести программу ПЛК на «целевую платформу», мы просто удалим из проекта эти переменные вместе с отладочным окном, – ни на что больше они не влияют.

Выходные переменные будут описывать состояние технологической установки, а также имитировать сигналы датчиков и измерительных преобразователей: конечных выключателей, датчиков работы транспортеров и наличия тележки, измерителей уровней в бункерах.

На рис. 3.134 показаны настройки блоков OPC Configuration и OPC Read.

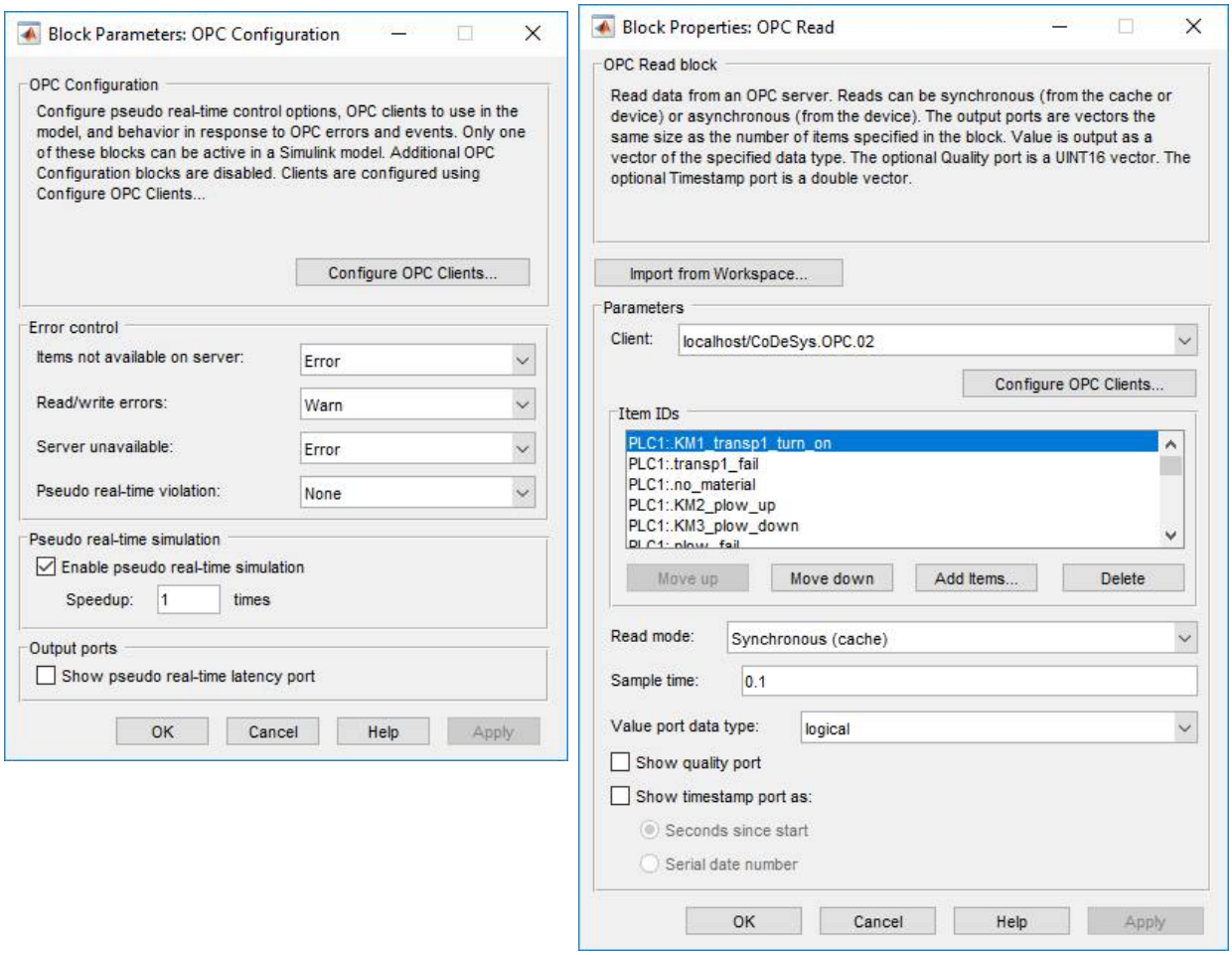


Рис. 3.134. Настройки блоков OPC Configuration и OPC Read.

В списке OPC Read входные для модели переменные перечислены в следующем порядке (порядок имеет значение для правильного понимания схемы):

```
1) KM1_transp1_turn_on:BOOL; (*МП транспортера 1 включен*)
2) transp1_fail:BOOL; (*неисправность транспортера 1*)
3) no_material:BOOL; (*отсутствие материала на подаче*)
4) KM2_plow_up:BOOL; (*МП поднятия плужка включен*)
5) KM3_plow_down:BOOL; (*МП опускания плужка включен*)
6) plow_fail:BOOL; (*неисправность плужкового сбрасывателя*)
```

– соответствующие сигналы идут на блок Transporter1, моделирующего работу подающего транспортера и плужкового сбрасывателя;

```
7) KM5_shiber1_open:BOOL; (*МП открытия шибера 1 включен*)
8) KM4_shiber1_close:BOOL; (*МП закрытия шибера 1 включен*)
9) shiber1_fail:BOOL; (*неисправность шибера бункера 1*)
10) bunker1_meas_fail:BOOL; (*неисправность измерителя бункера 1*)
```

– соответствующие сигналы идут на блок Bunker1, моделирующего процесс наполнения первого бункера и работу его шибера;

```
11) KM7_shiber2_open:BOOL; (*МП открытия шибера 2 включен*)
12) KM6_shiber2_close:BOOL; (*МП закрытия шибера 2 включен*)
13) shiber2_fail:BOOL; (*неисправность шибера бункера 2*)
14) bunker2_meas_fail:BOOL; (*неисправность измерителя бункера 2*)
```

– соответствующие сигналы идут на блок Bunker2, моделирующего процесс наполнения второго бункера и работу его шибера;

```
15) KM8_transp2_turn_on:BOOL; (*МП транспортера 2 включен*)
16) transp2_fail:BOOL; (*неисправность транспортера 2*)
```

– соответствующие сигналы идут на блок Transporter2, моделирующего работу разгрузочного транспортера;

```
17) HL24:BOOL; (*запрос на замену тележки HL24/реле K15*)
18) truck_fail_serv:BOOL; (*отказ в замене тележки*)
19) truck_fail_remove:BOOL; (*исчезновение тележки*)
```

– соответствующие сигналы идут на блок Truck, моделирующий процесс замены тележки.

Блок-схема включает также два блока OPC Write, занимающиеся записью по OPC аналоговых и дискретных выходных переменных, их настройки показаны на рис. 3.135.

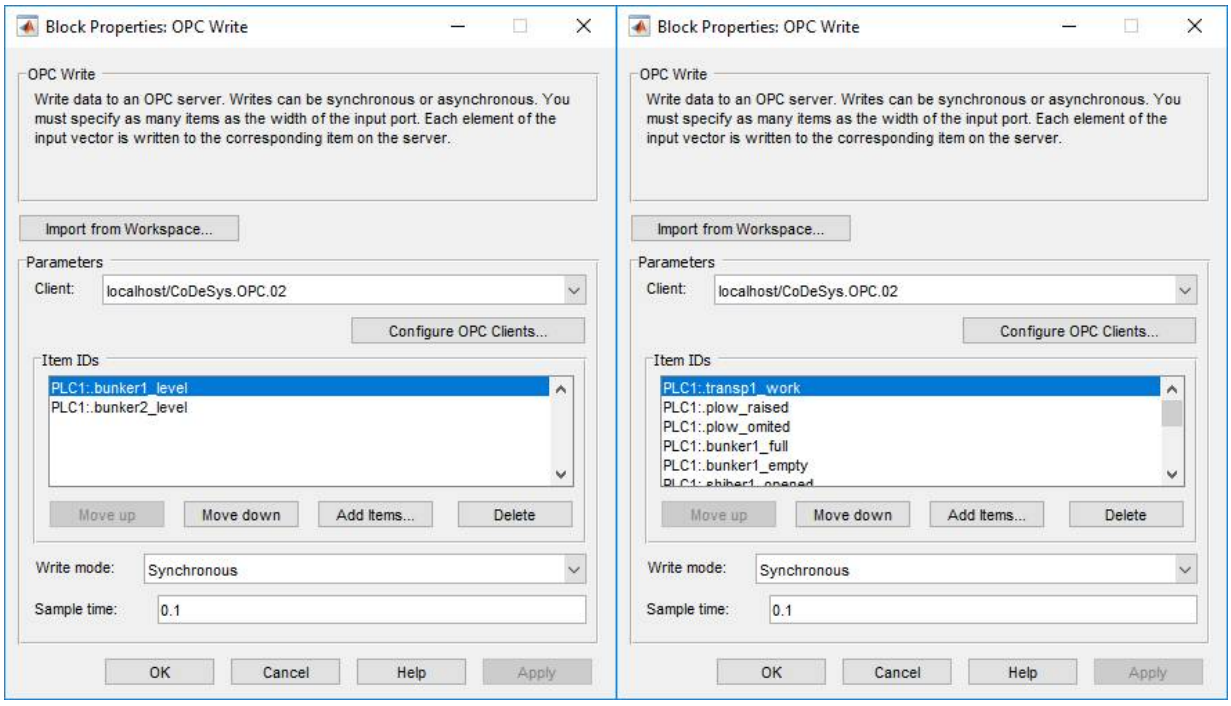


Рис. 3.135. Настройки блоков Write_real и Write_logic.

Блок Write_real записывает значения переменных:

```
1) bunker1_level:REAL; (*Уровень в бункере 1*)
```

– формируется блоком Bunker 1;

```
2) bunker2_level:REAL; (*Уровень в бункере 2*)
```

– формируется блоком Bunker 2;

Блок Write_logic записывает значения переменных:

```
1) transp1_work:BOOL; (*транспортер 1 работает*)
2) plow_raised:BOOL; (*плужок поднят*)
3) plow_omited:BOOL; (*плужок опущен*)
```

– формируются блоком Transporter 1;

```
4) bunker1_full:BOOL; (*бункер 1 полон*)
5) bunker1_empty:BOOL; (*бункер 1 пуст*)
6) shiber1_opened:BOOL; (*шибер бункера 1 открыт*)
7) shiber1_closed:BOOL; (*шибер бункера 1 закрыт*)
```

– формируются блоком Bunker 1;

```
8) bunker2_full:BOOL; (*бункер 2 полон*)
9) bunker2_empty:BOOL; (*бункер 2 пуст*)
10)  shiber2_opened:BOOL; (*шибер бункера 2 открыт*)
11)  shiber2_closed:BOOL; (*шибер бункера 2 закрыт*)
```

– формируются блоком Bunker 2;

```
12)  transp2_work:BOOL; (*транспортер 2 работает*)
```

– формируется блоком Transporter 2;

```
13)  truck:BOOL; (*тележка установлена*)
```

– формируется блоком Truck.

Перейдем к рассмотрению блоков (подсистем) модели. На рис. 3.136 показаны «внутренности» подсистемы Transporter1.

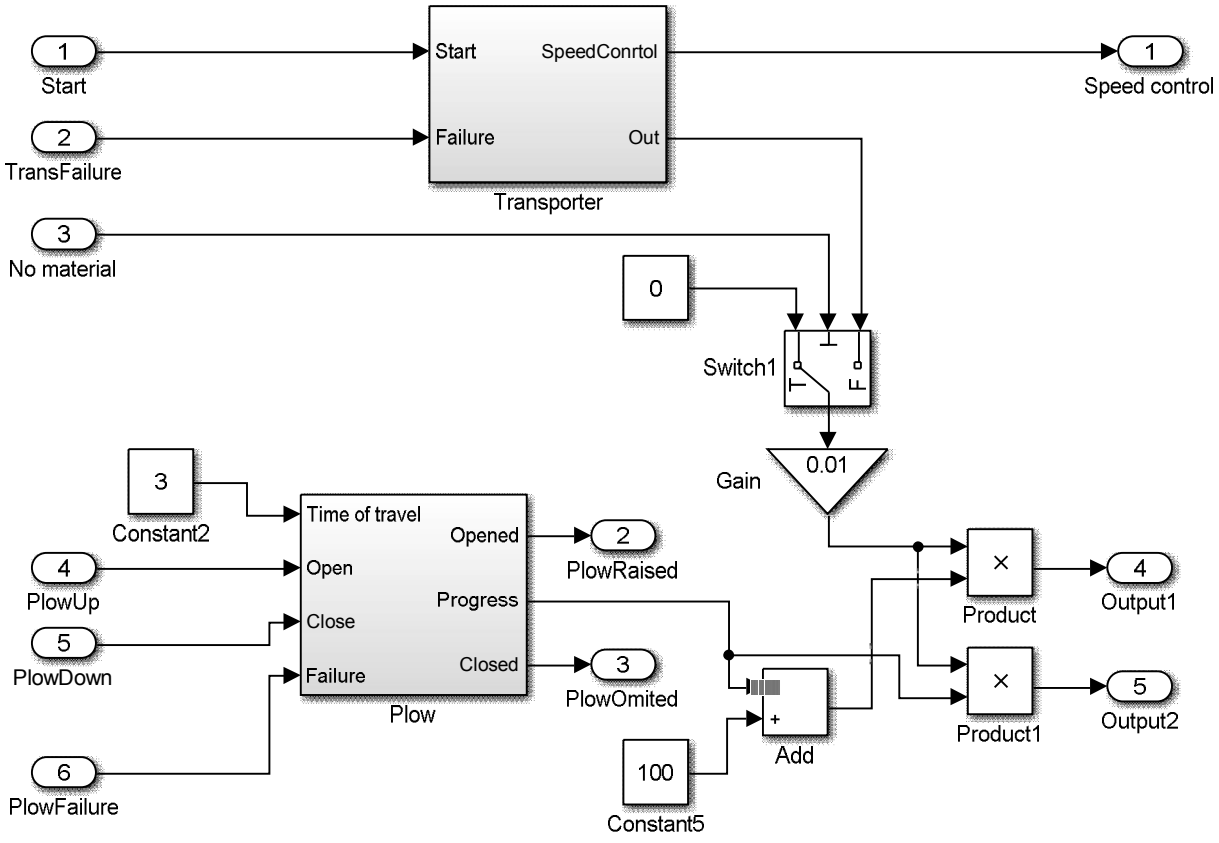


Рис. 3.136. Подсистема Transporter1.

Подсистема формирует «дискретный» сигнал срабатывания датчика работы подающего транспортера (Speed control) и два «аналоговых» сигнала Output1 и Output2, значения которых равны относительным расходам сыпучего материала на заполнение первого и второго бункера соответственно (0-100%). Последние зависят от относительной скорости движения ленты транспортера (выход Out подсистемы Transporter, 0-100%) и относительного положения (высоты поднятия) плужка (выход Progress подсистемы Plow, 0-100%). Если транспортер работает на полной скорости (100%), а плужок полностью опущен (0%), на выходе Output1 установится значение 100%, а на выходе Output2 – 0%. Если при нормальной работе транспортера плужок полностью поднят (100%), то все будет наоборот: на выходе Output1 – 0%, а на выходе Output1 – 100%. Конечно, наша система управления не должна допускать «промежуточных ситуаций», когда одновременно происходит наполнение обоих бункеров, но модель такую возможность допускает.

Активный сигнал на входе No material обнуляет подачу материала в оба бункера.

Подсистема Transporter показана на рис. 3.137.

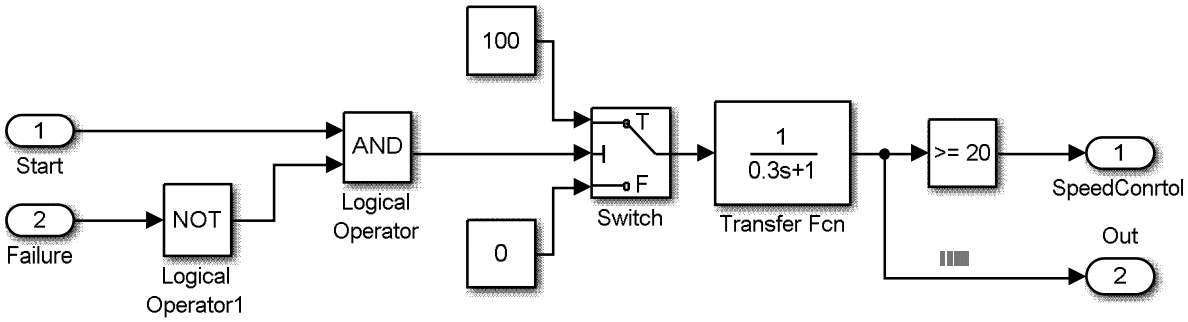


Рис. 3.137. Подсистема Transporter.

Подсистема формирует дискретный сигнал контроля скорости транспортерной ленты (TRUE/FALSE) и непрерывный сигнал по самой скорости (0-100%). Динамика набора скорости при включении сформирована с помощью простейшего звена первого порядка. Для запуска транспортера необходимо наличие сигнала на входной линии Start и отсутствие на линии Failure (неисправность). При этом переключатель Switch посылает на свой выход значение 100%. В противном случае (при отсутствии команды запуска и/или при неисправности) на выходе переключателя устанавливается 0% и «транспортер останавливается».

На рис. 3.138 показана подсистема Plow.

Входными для подсистемы являются «константа» Time of travel (время полного хода плужка в секундах), команды на поднятие и опускание плужка Open и Close и сигнал ввода неисправности Failure. На выходе формируются дискретные сигналы Opened и Closed (сигналы концевых выключателей механизма) и непрерывный сигнал Progress (относительное перемещение, 0-100%). Привод плужкового сбрасывателя не оснащен измерителем положения, поэтому сигнал Progress – это не сигнал измерителя, а переменная, описывающая «фактическое положение» плужка, необходимая для дальнейших вычислений.

Отметим также, что «странности» в именовании переменных (например, поднять – open) связаны с тем обстоятельством, что блок Plow – всего лишь «экземпляр» блока Valve, который был разработан для моделирования любой запорной арматуры. Хотя плужковый сбрасыватель, конечно, не является запорной арматурой, его модель ничем от модели арматуры не отличается.

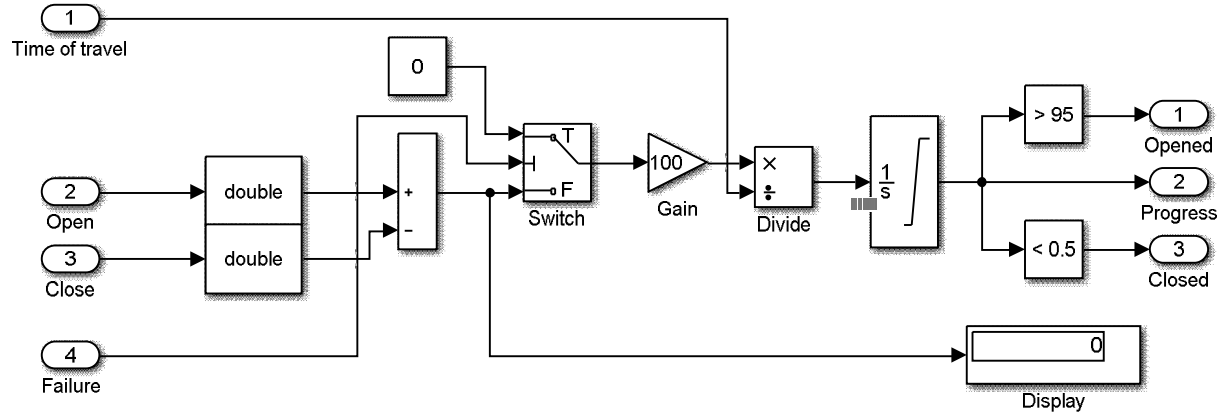


Рис. 3.138. Подсистема Plow.

Главный элемент подсистемы – это интегратор с ограничением выходного сигнала (0–100%). Постоянная времени интегрирования, равная времени полного хода, подается на второй вход (делитель) блока Divide. На первый вход (делимое) подается сигнал с выхода блока Gain, который может быть равен 0 (привод остановлен), 100 (поднятие плужка) и –100 (опускание плужка) в зависимости от комбинации входных сигналов подсистемы. Сигналы с входов Open и Close конвертируются блоками Double следующим образом: TRUE ⇒ 1, FALSE ⇒ 0. Независимо от значений на входах Open и Close активизация входа Failure приведет к подаче нуля на вход интегратора (остановке привода).

Блоки Bunker1 и Bunker2 полностью идентичны. Их устройство показано на рис. 3.139.

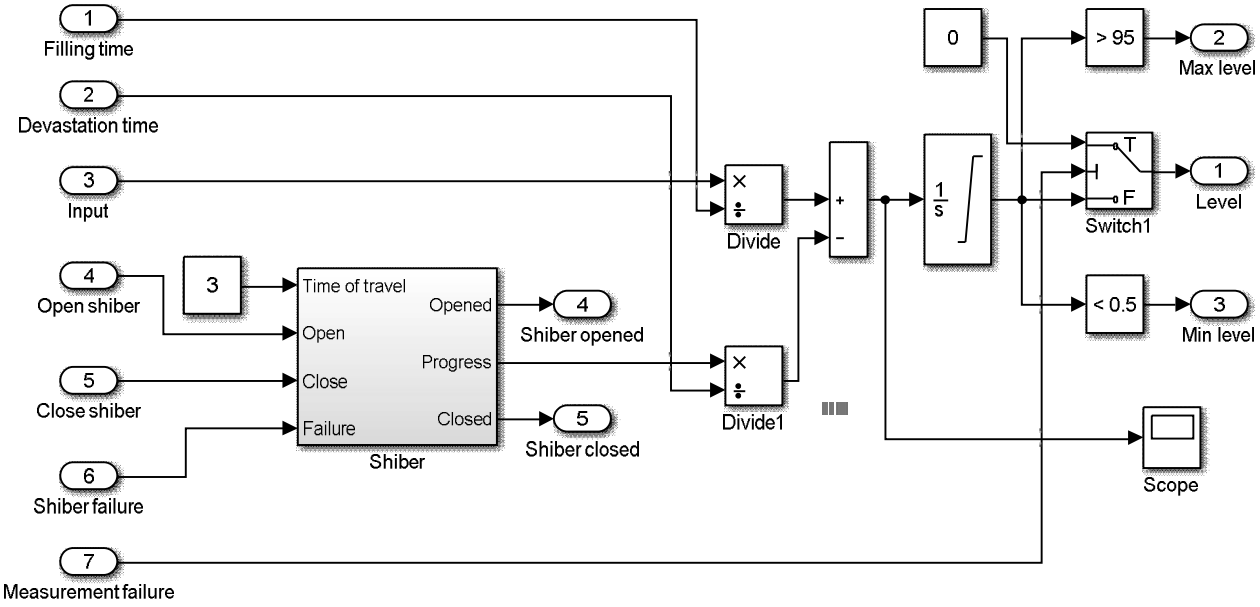


Рис. 3.139. Подсистемы Bunker (1 и 2).

Подсистема занимается формированием сигналов об уровне в бункере.

На вход подсистемы подаются следующие сигналы:

Filling Time – время полного наполнения бункера, с;

Devastation Time – время полного опустошения бункера, с;

Input – относительный расход сыпучего материала на стороне подачи, 0 – 100%;

Open shiber, Close shiber – команды на открытие и закрытие шибера;

Shiber failure – неисправность шибера;

Measurement failure – неисправность измерителя уровня.

Bunker в своем составе содержит подсистему Shiber, моделирующую шиберную заслонку. Подсистема полностью идентична ранее рассмотренной подсистеме Plow.

Основным элементом модели бункера является интегратор, вычисляющий уровень путем интегрирования разности сигналов, пропорциональных расходам материала на подаче и выгрузке. Оба сигнала измеряются в процентах. Первый сигнал подается с входа Input, второй – с выхода Progress подсистемы Shiber. Перед вычислением разности сигналы масштабируются путем деления на значения настроечных входов Filling time и Devastation time.

Активный уровень сигнала на входе Measurement failure обнуляет выход Level (отказ измерителя).

Подсистема Transporter 2 (см. рис. 3.133) является копией подсистемы Transporter, входящей в Transporter 1.

Подсистема Truck занимается заменой тележки. Ее устройство показано на рис. 3.140.

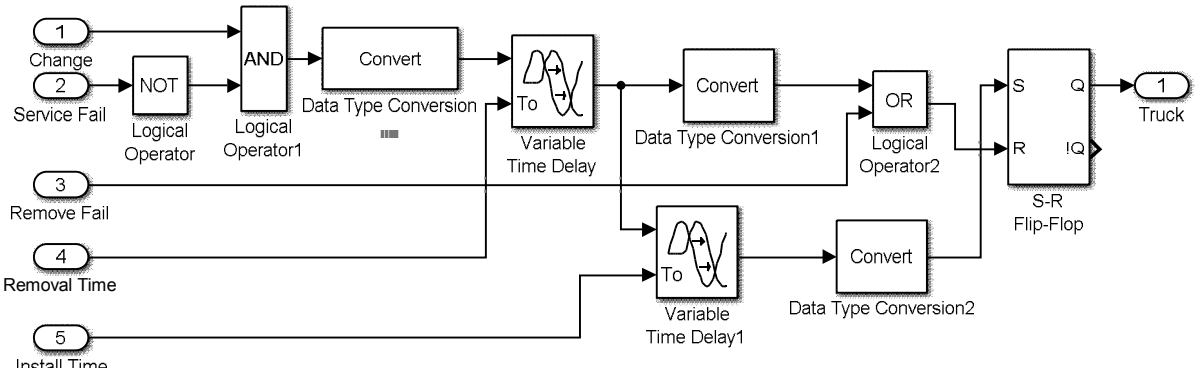


Рис. 3.140. Подсистема Truck.

Подсистема содержит два блока временной задержки с настраиваемым временем для формирования задержек при удалении и установке тележки. Их настройка производится сигналами с входов подсистемы Removal Time и Install Time. При поступлении сигнала Change (сменить тележку) и отсутствии ошибок после первой выдержки времени RS-триггер S-R Flip-Flop будет сброшен сигналом на входе R (тележка «исчезнет»). К этому моменту сигнал Change будет уже снят. После второй выдержки времени S-R Flip-Flop будет установлен сигналом на входе S (тележка «появится» вновь).

Активный уровень сигнала на входе Service Fail (отказ в обслуживании) блокирует работу алгоритма, описанного выше. Подача сигнала на вход Re-

move Fail приведет к немедленному (внеплановому) «исчезновению» тележки.

Блоки Data Type Conversion предназначены для преобразования сигналов logical (BOOL) → double и double → logical (BOOL).

3.2.1.3. Разработка программы для ПЛК

Собственно говоря, термин «программа для ПЛК» в нашем случае не является вполне корректным. «Настоящая» программа для ПЛК будет всего лишь частью того комплекса программ, который будет рассмотрен в данном разделе. Пришло время привести, наконец, полную схему работы имитационной системы, рис. 3.141.

Рис. 3.141. Структура программного комплекса.

Ниже приведен список глобальных переменных проекта.

```
VAR_GLOBAL
(*Датчики/входы контроллера*)
transp1_work:BOOL; (*транспортер 1 работает - сигнал реле K1*)
plow_omited:BOOL; (*плужок опущен  - сигнал реле K2*)
plow_raised:BOOL; (*плужок поднят  - сигнал реле K3*)
shiber1_closed:BOOL; (*шибер бункера 1 закрыт - сигнал реле K3*)
shiber1_opened:BOOL; (*шибер бункера 1 открыт - сигнал реле K4*)
bunker1_empty:BOOL; (*бункер 1 пуст  - сигнал реле K6*)
bunker1_full:BOOL; (*бункер 1 полон  - сигнал реле K7*)
shiber2_closed:BOOL; (*шибер бункера 2 закрыт - сигнал реле K8*)
shiber2_opened:BOOL; (*шибер бункера 2 открыт - сигнал реле K9*)
bunker2_empty:BOOL; (*бункер 2 пуст  - сигнал реле K10*)
bunker2_full:BOOL; (*бункер 2 полон  - сигнал реле K11*)
transp2_work:BOOL; (*транспортер 2 работает - сигнал реле K12*)
truck:BOOL; (*тележка установлена  - сигнал реле K13*)
```

```
(*Другие входы*)
K14:BOOL; (*блокировки отключены*)
SA1_auto:BOOL; (*автоматический режим*)
KM1_transp1_turn_on:BOOL; (*МП транспортера 1 включен*)
KM2_plow_up:BOOL; (*МП поднятия плужка включен*)
KM3_plow_down:BOOL; (*МП опускания плужка включен*)
KM4_shiber1_close:BOOL; (*МП закрытия шибера 1 включен*)
KM5_shiber1_open:BOOL; (*МП открытия шибера 1 включен*)
KM6_shiber2_close:BOOL; (*МП закрытия шибера 2 включен*)
KM7_shiber2_open:BOOL; (*МП открытия шибера 2 включен*)
KM8_transp2_turn_on:BOOL; (*МП транспортера 2 включен*)
(*Аналоговые входы*)
bunker1_level:REAL; (*уровень в бункере 1*)
bunker2_level:REAL; (*уровень в бункере 2*)
(*Выходы контроллера*)
transp1_turn_on:BOOL; (*транспортер 1 включить*)
plow_up:BOOL; (*плужок поднять*)
plow_down:BOOL; (*плужок опустить*)
shiber1_close:BOOL; (*шибер бункера 1 закрыть*)
shiber1_open:BOOL; (*шибер бункера 1 открыть*)
shiber2_close:BOOL; (*шибер бункера 2 закрыть*)
shiber2_open:BOOL; (*шибер бункера 2 открыть*)
transp2_turn_on:BOOL; (*транспортер 2 включить*)
signal:BOOL; (*сигнализация*)
truck_change:BOOL; (*заменить тележку*)
(*Элементы электрической схемы*)
K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12,K13:BOOL;    (*промежуточные
реле*)
HL1,HL2,HL3,HL4,HL5,HL6,HL7,HL8,HL9,HL10,HL11,HL12,HL13,
HL14,HL15,HL16,HL17,HL18,HL19,HL20,HL21,HL22:BOOL; (*лампы*)
HL23:BOOL; (*сигнализация: ревун HA1 + лампа HL23*)
HL24:BOOL; (*сигнал сменить тележку: реле K15 + лампа HL24*)
SB1,SB2,SB3,SB4,SB5,SB6,SB7,SB8,SB9,SB10,SB11,SB12,SB13:BOOL;
(* - кнопки*)
SA1_man:BOOL; (*переключатель – ручной режим*)
(*Фиктивные переменные для ввода ошибок*)
transp1_fail:BOOL; (*неисправность транспортера 1*)
no_material:BOOL; (*отсутствие материала на подаче*)
transp2_fail:BOOL; (*неисправность транспортера 2*)
plow_fail:BOOL; (*неисправность плужкового сбрасывателя*)
shiber1_fail:BOOL; (*неисправность шибера бункера 1*)
shiber2_fail:BOOL; (*неисправность шибера бункера 2*)
bunker1_meas_fail:BOOL; (*неисправность измерителя бункера 1*)
bunker2_meas_fail:BOOL; (*неисправность измерителя бункера 2*)
truck_fail_remove:BOOL; (*исчезновение тележки*)
truck_fail_serv:BOOL; (*незамена тележки*)
(*Переменные для управления программой в автоматическом режиме*)
current_task:REAL:=50.55; (*текущее задание - процент наполнения
бункера*)
start_prog:BOOL:=FALSE; (*пуск программы*)
reset:BOOL:=FALSE; (*сброс после аварии*)
END_VAR
```

С частью переменных мы уже знакомы, поскольку они участвуют в обмене с Simulink-моделью. Это все переменные секций «Датчики/входы контроллера», «Аналоговые входы» и «Фиктивные переменные для ввода ошибок», а также переменные секции «Другие входы», за исключением первых двух, которые в обмене по OPC участия не принимают. «Незнакомые» пока переменные будут задействованы в дальнейшем при разработке программных единиц проекта.

При переносе программы на «целевую платформу» из списка следует исключить секции «Элементы электрической схемы» и «Фиктивные переменные для ввода ошибок», так как электрическая схема будет реализована «живьем», а об ошибках позаботится сам объект управления.

Рассмотрение программных единиц начнем с программы, имитирующей работу релейной схемы управления. Абсолютно естественным выбором для составления программы является язык релейной логики LAD (Ladder Diagram).

Программа представлена на рис. 3.142 – 3.145 четырьмя фрагментами.

По сравнению с электрической схемой, приведенной на рис. 3.129, 3.130, в программе нет ничего нового. Другими словами, программа практически полностью соответствует схеме. Различия носят чисто технический характер:

1) в программе отсутствуют элементы тепловой защиты двигателей (контакты тепловых реле). Так как сигналы о срабатывании тепловых реле на входы контроллера не подаются (что, в принципе, можно было сделать), имитировать перегрузки не имеет смысла. На практике срабатывание какого-либо теплового реле приведет к остановке защищаемого им двигателя, что будет «замечено» соответствующим датчиком и системой управления в целом. Система сообщит оператору о проблеме, и он, выясняя причины отказа, обнаружит факт срабатывания реле. Для предотвращения «самозапуска» механизма после «остывания» теплового реле в автоматическом режиме управления достаточно поставить его на «ручной взвод» (в режиме ручного управления «самозапуск» невозможен в принципе). Дополнительно желательно предусмотреть перевод всех выходов контроллера в безопасное состояние (выключено) при возникновении аварии до тех пор, пока оператор не разрешит дальнейшее выполнение программы автоматического управления;

2) в программе, в отличие от электрической схемы, все «катушки», включая «катушки» «пускателей», размещены в правой части цепей, так как это требует редактор LAD. Однако логика работы цепей при этом ничуть не изменилась;

3) переключатель SA1 в программе представлен двумя «контактами» (булевыми переменными) SA1_man и SA1_auto, «замыкающимися» в ручном и автоматическом режимах соответственно. Очевидно, что ситуация SA1_man = SA1_auto = TRUE недопустима, что следует учесть в дальнейшем;

4) в программе отсутствуют ревун HA1 и реле K15, поскольку для имитационного моделирования вполне достаточно ламп HL23 и HL24, срабатывающих одновременно с ревуном и реле.

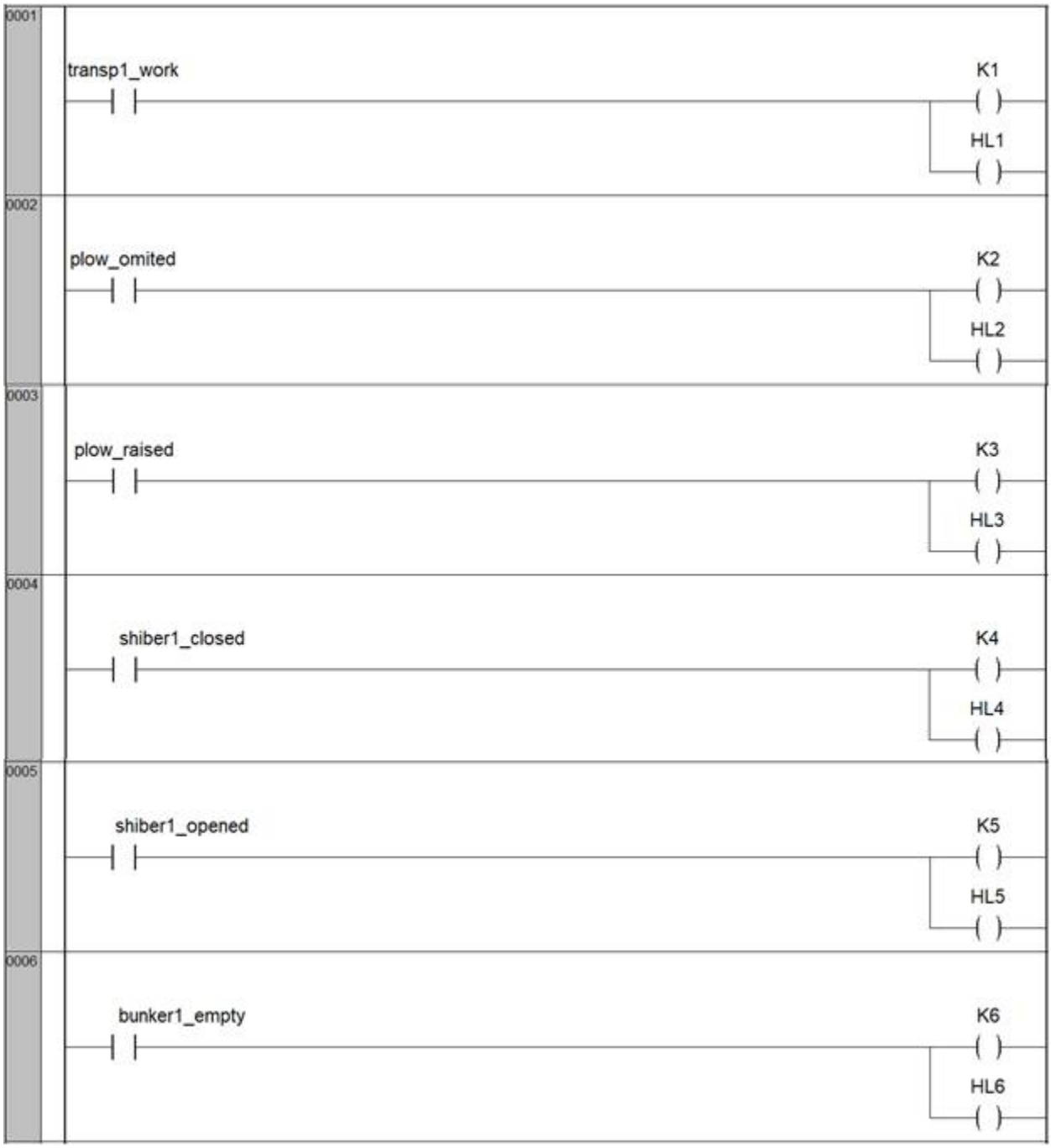


Рис. 3.142. Программа `Electrical_Circuits`, фрагмент 1.

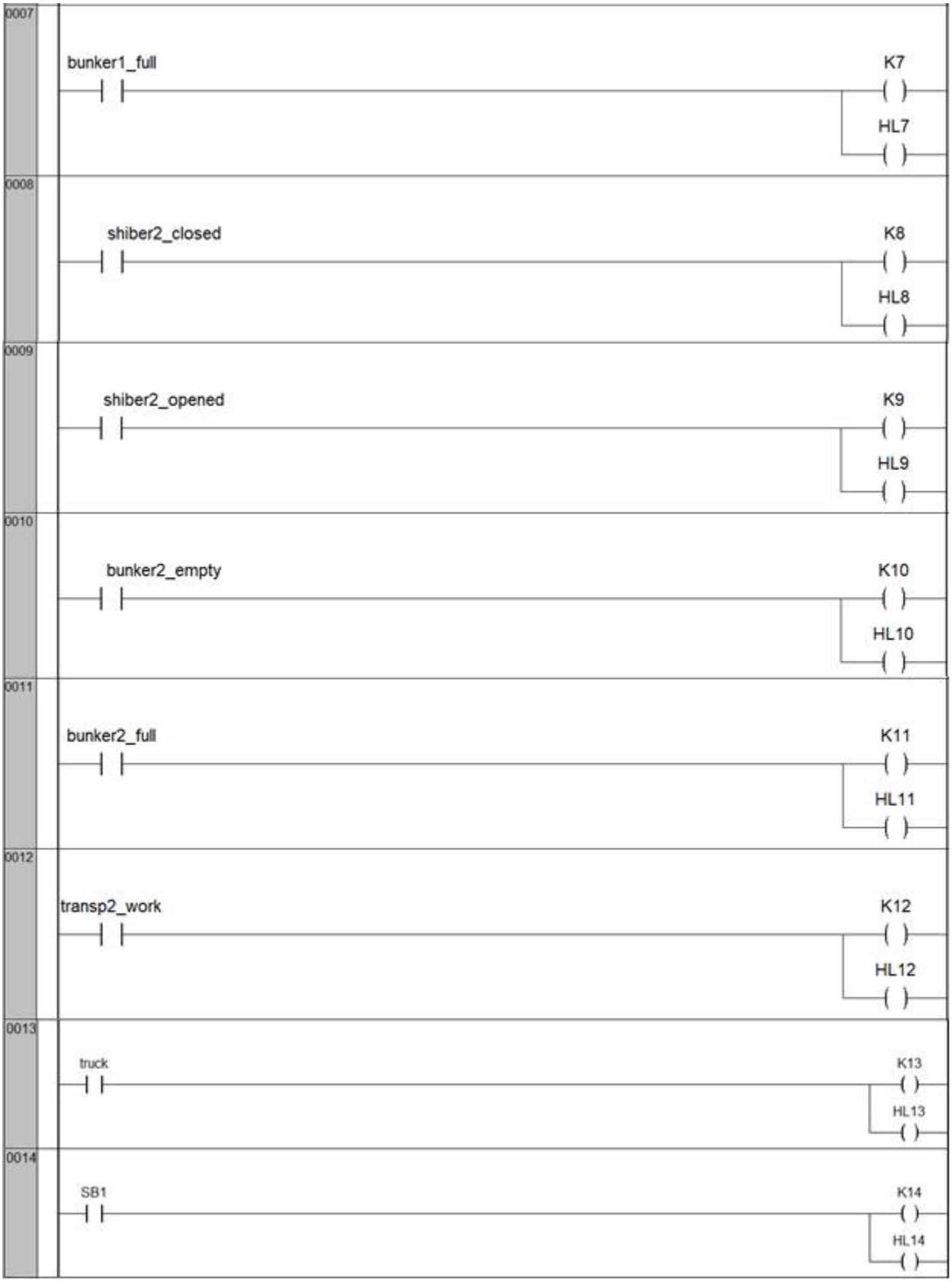


Рис. 3.143. Программа `Electrical_Circuits`, фрагмент 2.

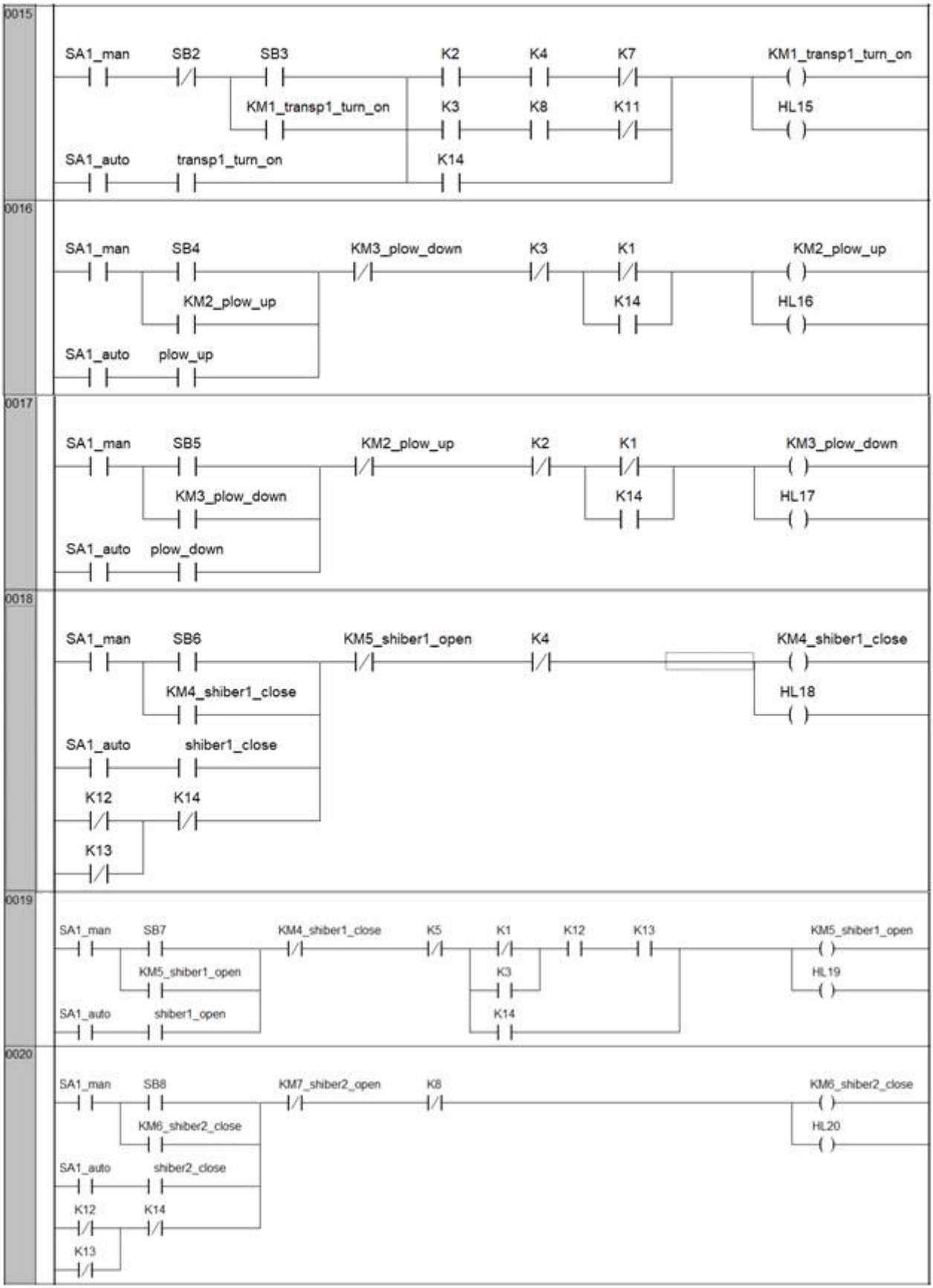


Рис. 3.144. Программа `Electrical_Circuits`, фрагмент 3.

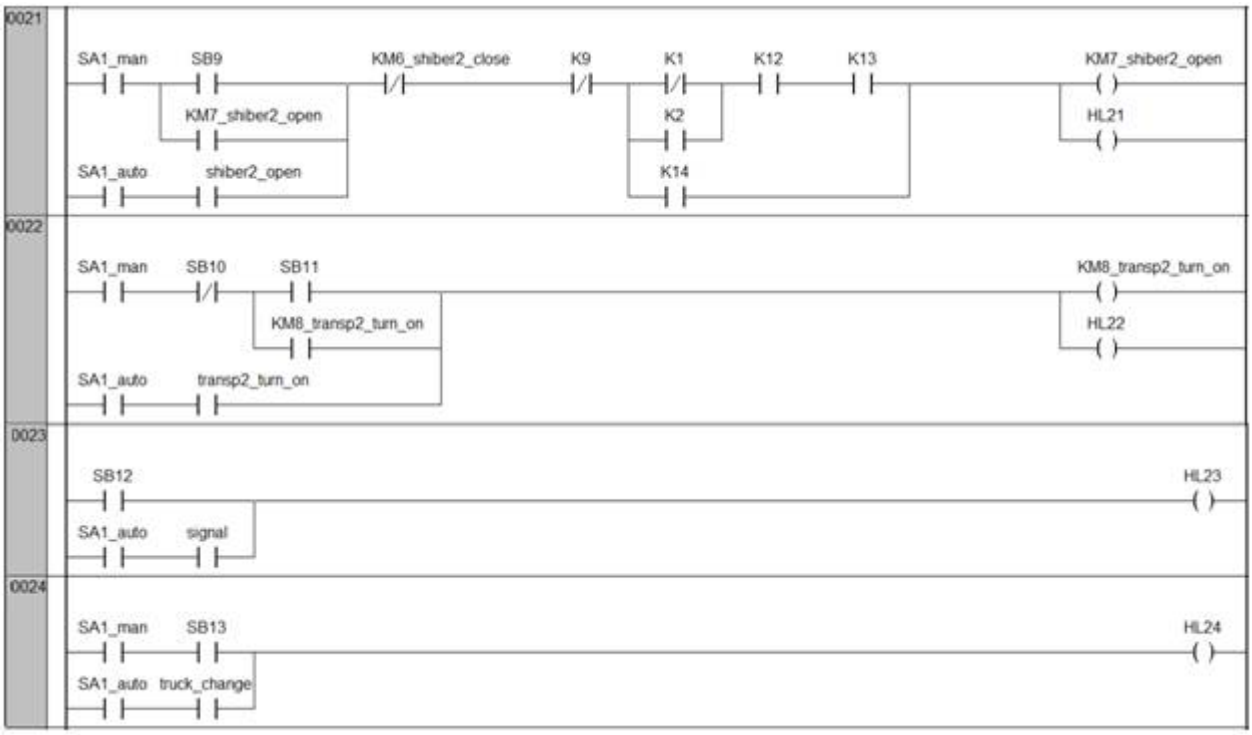


Рис.3.145. Программа `Electrical_Circuits`, фрагмент 4.

В ручном режиме управление осуществляется с помощью органов *щита управления*. На щите установлены индикаторы уровня в бункерах, кнопки, сигнальные лампы, переключатель режимов управления.

В нашей системе щит будет имитироваться экраном визуализации CoDeSys, показанным на рис. 3.146 «в работе».

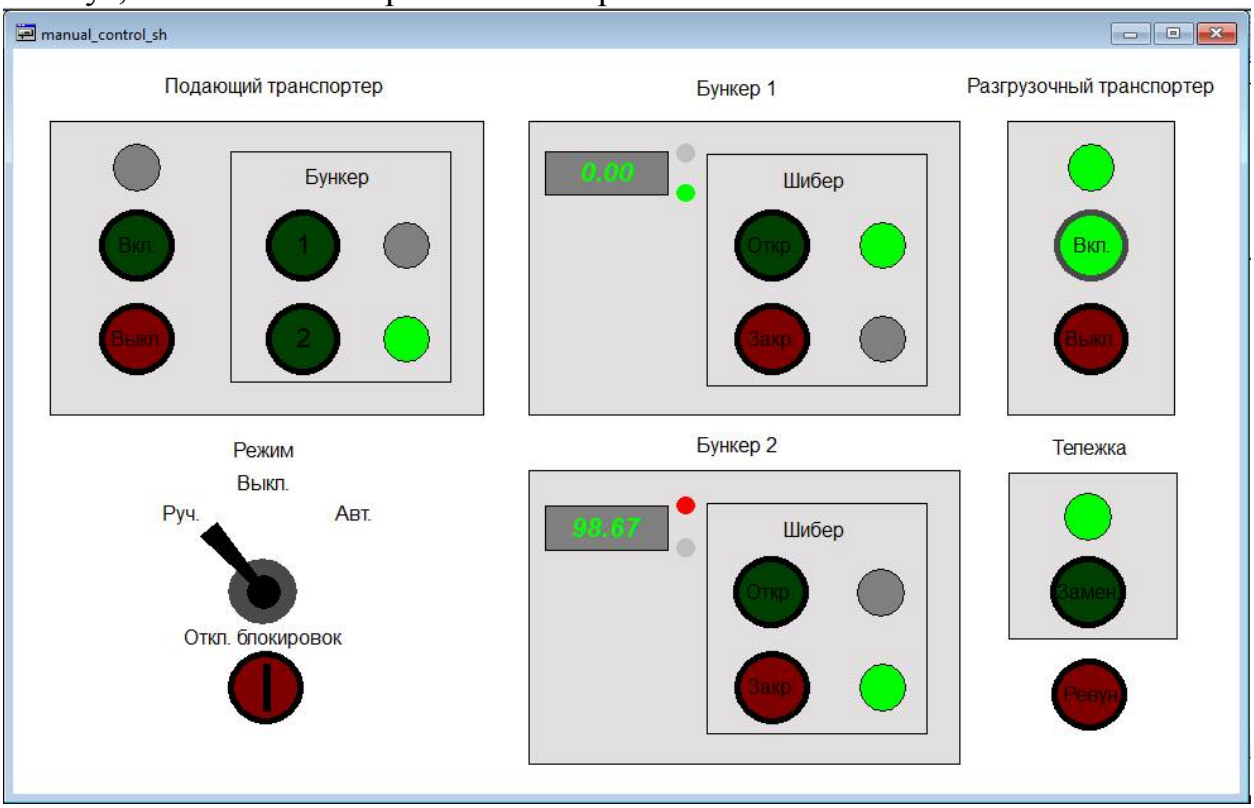


Рис. 3.146. Экран визуализации для имитации щита управления.

В контуре «Подающий транспортер» расположены:

лампа, сигнализирующая о работе транспортера (HL1);

кнопка с подсветкой включения транспортера «Вкл» (SB3/HL15);

кнопка остановки транспортера «Выкл» (SB2);

кнопка выбора первого бункера «1» (плужок опустить) с подсветкой (SB5/HL17);

кнопка выбора второго бункера «2» (плужок поднять) с подсветкой (SB4/HL16);

лампа сигнализации выбора бункера 1 (плужок опущен, HL2);

лампа сигнализации выбора бункера 2 (плужок поднят,HL3).

В контуре «Бункер 1» расположены:

прибор индикации уровня (HG1, рис. рис.3.127);

лампы сигнализации о достижении верхнего и нижнего уровней (HL7, HL6);

кнопка с подсветкой открытия шибера «Откр» (SB7/HL19);

кнопка с подсветкой закрытия шибера «Закр» (SB6/HL18);

лампа сигнализации полного открытия шибера (HL5);

лампа сигнализации полного закрытия шибера (HL4).

В контуре «Бункер 2» расположены:

прибор индикации уровня (HG2, см. рис. 3.132);

лампы сигнализации о достижении верхнего и нижнего уровней (HL11, HL10);

кнопка с подсветкой открытия шибера «Откр» (SB9/HL21);

кнопка с подсветкой закрытия шибера «Закр» (SB8/HL20);

лампа сигнализации полного открытия шибера (HL9);

лампа сигнализации полного закрытия шибера (HL8).

В контуре «Разгрузочный транспортер» расположены:

лампа, сигнализирующая о работе транспортера (HL12);

кнопка с подсветкой для включения транспортера «Вкл» (SB11/HL22);

кнопка остановки транспортера «Выкл» (SB10).

В контуре «Тележка» расположены:

лампа, сигнализирующая о наличии тележки (HL13);

кнопка с подсветкой замены тележки «Замен» (SB13/HL24).

Вне контуров расположены:

переключатель режима управления (SA1, см. рис. 3.130);

кнопка с ключом отключения блокировок (SB1/HL14);

кнопка с подсветкой подачи звукового сигнала «Ревун» (SB12/HL23).

Привязки элементов экрана показаны на рис. 3.147.

Комментариев требуют привязки переключателя режима управления SA1 и кнопки с ключом отключения блокировок SB1.

«Самодельный» переключатель SA1 имеет три «сектора», щелчком мыши по которым имеется возможность перевести систему в ручной или автоматический режимы или выключить ее вовсе. Щелчок мыши приводит к выполнению программ (см. окно «Конфигурирование полигона» категории «Ввод»), тексты которых показаны на рис. 3.147, а также к изменению цвета сектора с белого

(невидимый сектор) на черный (появление «рукоятки» переключателя в нужной позиции).

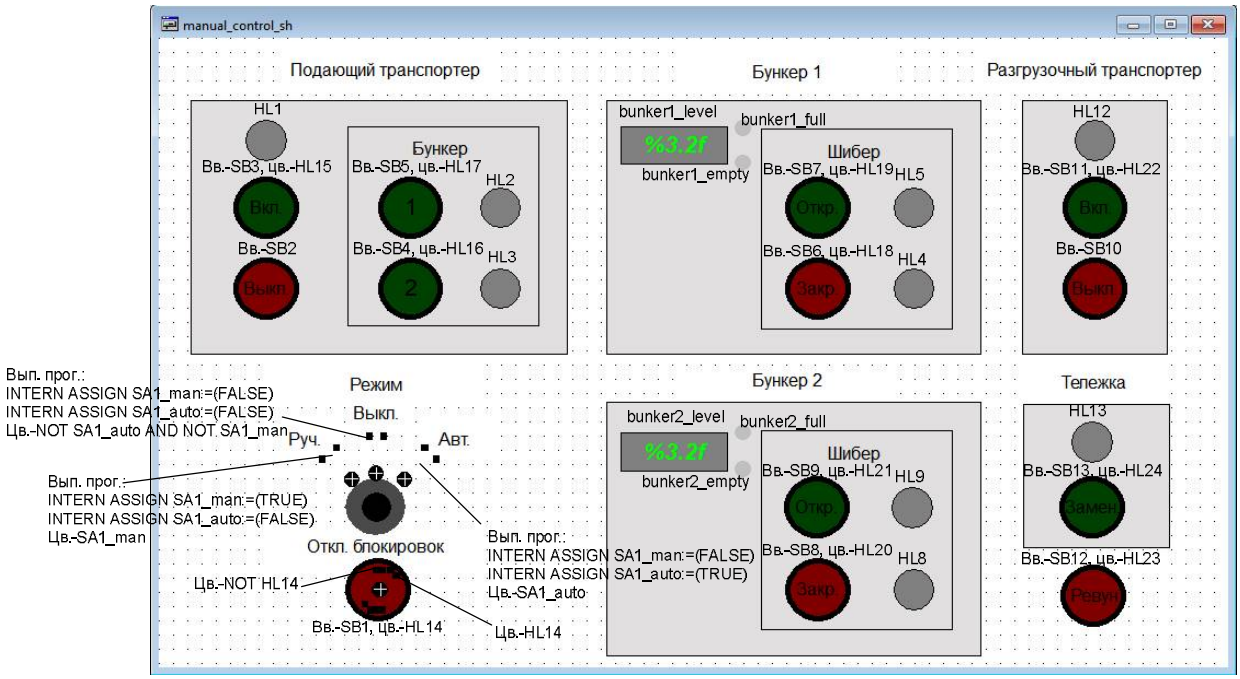


Рис. 3.147. Привязки экрана визуализации имитации щита управления.

Черная линия на кнопке SB1 («Откл. блокировок») располагается вертикально при включенных блокировках и под наклоном – при выключенных (поворот ключа). На самом деле линий две: первая «проявляется», когда блокировки отключены, и лампа HL14 кнопки «не горит», вторая – когда блокировки включены и лампа HL14 «горит».

Панель оператора предназначена в основном для визуализации технологического процесса, причем в обоих режимах управления: и в ручном и в автоматическом. В автоматическом режиме панель дополнительно позволяет запускать программу, задавать процент наполнения бункеров и производить «сброс» системы при возникновении аварии. Экран визуализации панели в работе показан на рис. 3.148. Все элементы визуализации привязаны к «реальным» (не фиктивным) переменным контроллера (рис. 3.149). Большая часть элементов привязана к входам, однако имеются и элементы, управляемые другими переменными:

положение «плужка» (сдвиг по оси Y) задается вычисляемой в программе `PLC_PRG` переменной `plow_position` (см. ниже);

положение «шиберов» (сдвиги по оси X) задаются вычисляемыми в программе `PLC_PRG` переменными `shiber1_position` и `shiber2_position` (см. ниже);

кнопки и поле ввода задания привязаны к глобальным, но не входным, переменным `start_prog`, `reset` и `current_task`;

в текстовое поле выводится строка сообщения об аварии `alarm_string`, формируемая программой автоматического управления `automat` (см. ниже).

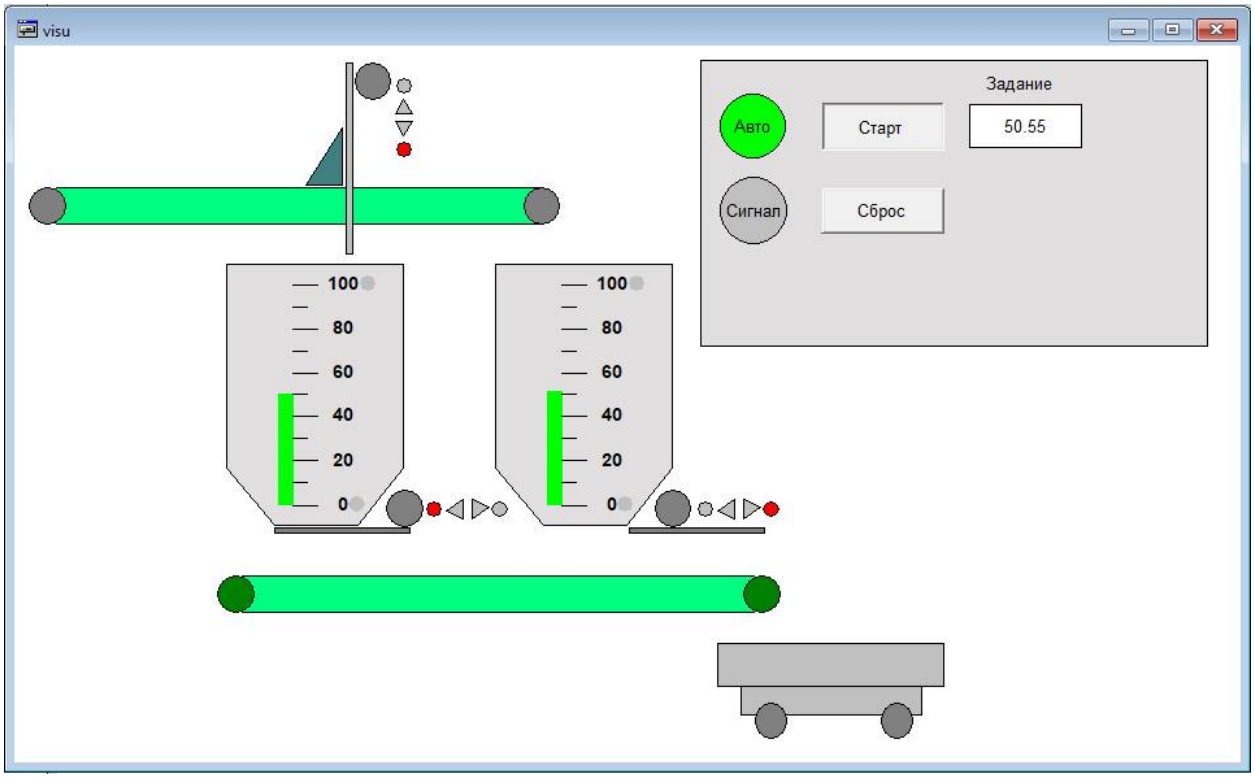


Рис.3.148. Экран визуализации операторской панели.

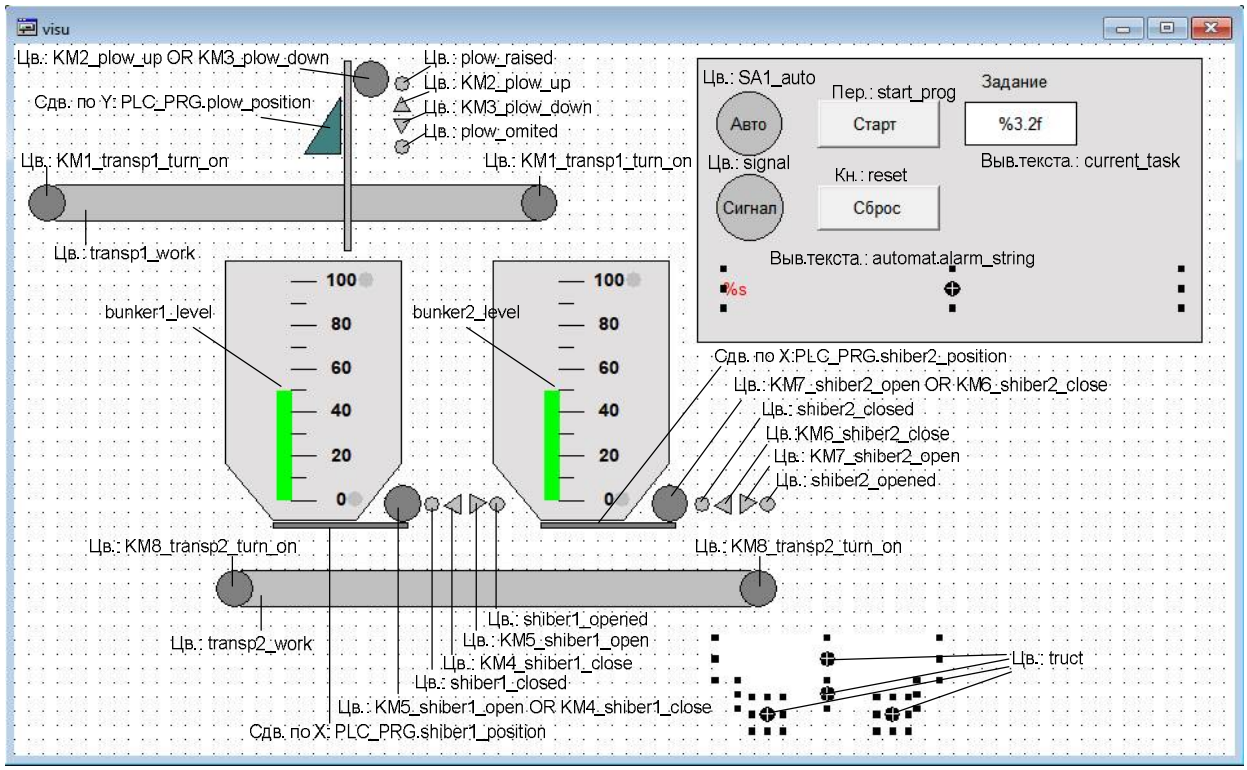


Рис. 3.149. Привязки экрана визуализации операторской панели.

На рис. 3.150 показан виртуальный экран ввода неисправностей с привязками. На экране располагаются кнопки с фиксацией, привязанные к соответствующим глобальным переменным.

Программа PLC_PRG:

```
PROGRAM PLC_PRG
VAR
     plow_position:BYTE;
     shiber1_position:BYTE;
     shiber2_position:BYTE;
END_VAR
-------------------------
IF plow_raised THEN
     plow_position:= 0;
ELSIF plow_omited THEN
     plow_position:=20;
ELSE
     plow_position:=10;
END_IF
IF shiber2_closed THEN
     shiber2_position:= 0;
ELSIF shiber2_opened THEN
     shiber2_position:=60;
ELSE
     shiber2_position:=30;
END_IF

IF shiber1_closed THEN
     shiber1_position:= 0;
ELSIF shiber1_opened THEN
     shiber1_position:=60;
ELSE
     shiber1_position:=30;
END_IF

Electrical_circuits;
automat;
```

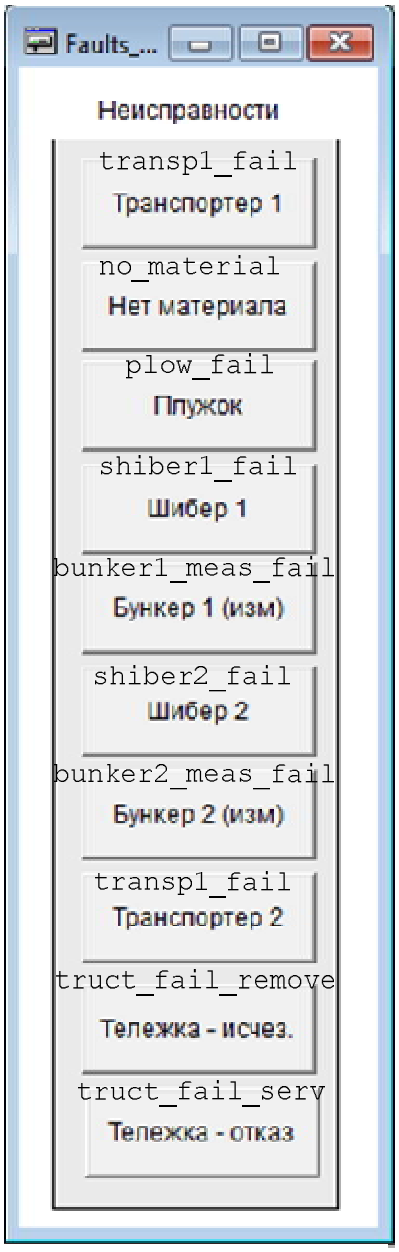


Рис. 3.150. Привязки экрана `Faults and truck`.

Программа занимается формированием своих локальных переменных, управляющих положением подвижных элементов экрана визуализации лицевой панели по сигналам «датчиков» положения плужка и шиберов, а также вызывает программы `Electrical_circuits` (имитация релейной схемы управления) и `automat` (программа автоматического управления).

Программа `automat`:

```
PROGRAM automat
VAR
     status:BYTE:=0;
     (*0 - ожидание,
       1 -  наполнение бункера №1, опустошение бункера №2,
       2 -  наполнение бункера №2, опустошение бункера №1,
       3 - смена тележки,
       4 - авария*)
```

```
     timer_transp2: TOF; (*таймер для отключения разгрузочного
                           транспортера*)
     timer_truck:TON; (*таймер для вызова тележки*)
     timer_load:TON; (*таймер для контроля времени при загрузке*)
     timer_unload:TON; (*таймер для контроля времени при
                         выгрузке*)
     alarm_string:STRING(50); (*сообщение об ошибке*)
     alarm:BOOL:=FALSE; (*произошла авария*)
     signal1: BOOL:=FALSE; (*предупр. сигнализация перед запуском
                             загрузочного транспортера*)
     unload_enable:BOOL:=FALSE; (*разрешение на разгрузку*)
     signal2: BOOL:=FALSE; (*предупр. сигнализация перед запуском
                             разгрузочного транспортера*)
     unload_success: BOOL; (*разгрузка прошла успешно*)
     load_success: BOOL; (*загрузка прошла успешно*)
     change_status: BOOL:=FALSE; (*фаза смены тележки: TRUE -
                                   тележку забрали*)
END_VAR

CASE status OF
0: (*ожидание*)
     transp1_turn_on:=shiber1_open:=shiber2_open:=truck_change:=
     signal:=FALSE; (*выкл. выходов*)
     alarm_string:=''; alarm:=FALSE; (*сброс аварии*)
     (*Сброс таймеров*)
     timer_truck(IN:=FALSE);
     timer_load(IN:=FALSE);

     (*Не автоматическое управление - выключение всех оставшихся
      выходов, останов таймеров*)
     IF NOT SA1_auto THEN
          shiber1_close:=shiber2_close:=transp2_turn_on:=FALSE;
          timer_transp2(IN:=FALSE);
          timer_unload(IN:=FALSE);
          RETURN;
     END_IF

     (*Автоматическое управление - безопасная "парковка"*)

     timer_unload(IN:=TRUE,PT:=t#10s);
     (* закрытие шиберов*)
     shiber1_close:=NOT shiber1_closed;
     shiber2_close:=NOT shiber2_closed;
     (*остановка разгрузочного транспортера спустя 5 сек. после
      закрытия шиберов непустых бункеров*)
     timer_transp2(IN:=(NOT bunker1_empty AND NOT shiber1_closed)
     OR (NOT bunker2_empty AND NOT
     shiber2_closed), PT:=t#5s,Q=>transp2_turn_on);

     (*Обработка ошибок*)
     (*отсутствие тележки при продолжении разгрузки*)
     IF ((NOT bunker1_empty  AND NOT shiber1_closed) OR
     (NOT bunker2_empty AND NOT shiber2_closed))
```

```
     AND NOT truck THEN
          alarm_string:='Тележка не подана';
          alarm:=TRUE;
     END_IF

     IF timer_unload.Q THEN (* в течение заданного времени*)
          IF NOT shiber1_closed THEN (*шибер 1 не закрылся*)
               alarm_string:='Неисправность при закрытии шибера
                              бункера 1';
               alarm:=TRUE;
          END_IF
          IF NOT shiber2_closed THEN (*шибер 1 не закрылся*)
               alarm_string:='Неисправность при закрытии шибера
                              бункера 2';
               alarm:=TRUE;
          END_IF
     END_IF

     IF alarm THEN (*переход в режим "Авария"*)
          timer_unload(IN:=FALSE);
          timer_transp2(IN:=FALSE);
          status:=4;
          RETURN;
     END_IF

     IF start_prog THEN (*переходы в режимы 1,2,3*)
          timer_unload(IN:=FALSE);
          timer_transp2(IN:=FALSE);
          IF truck THEN
               IF bunker1_empty THEN status:=1;
               ELSE status:=2;
               END_IF
          ELSE
               status:=3;
          END_IF
     END_IF

1: (*Режим 1: наполнение бункера №1, опустошение бункера №2*)

     alarm:=FALSE;  alarm_string:=''; (*пока все хорошо*)

     (*Загрузка бункера 1*)
     timer_load(IN:=bunker1_level< current_task,PT:=t#40s);(*конт-
                                              рольное время загрузки*)
     signal1:= bunker1_level< current_task AND
               timer_load.ET < t#2s; (*предупр. сигнализация*)
     (*управление загрузочным транспортером, плужком и шибером
       бункера 1*)
     transp1_turn_on:=plow_omited AND shiber1_closed AND
                     (bunker1_level < current_task) AND NOT
                      bunker1_full AND NOT signal1;
     plow_up:=FALSE;
     plow_down:=NOT plow_omited AND NOT signal1;
```

```
shiber1_open:=FALSE;
shiber1_close:=NOT shiber1_closed AND NOT signal1;

load_success:=bunker1_level > current_task AND
              NOT transp1_work; (*загрузка завершена*)

(*обработка ошибок при загрузке*)
IF NOT load_success THEN
     IF timer_load.ET>t#7s THEN (*спустя 7 сек. после начала
                                  процесса загрузки*)
          IF NOT plow_omited THEN (*плужок не опущен*)
               alarm_string:='Неисправность плужкового
                              сбрасывателя';
               alarm:=TRUE;
          END_IF
          IF   NOT shiber1_closed THEN (*шибер первого бункера
                                         не закрыт*)
               alarm_string:='Неисправность при закрытии шибе-
                              ра бункера 1';
               alarm:=TRUE;
          END_IF
          (*неисправность загрузочного транспортера*)
          IF (bunker1_level < current_task) AND
              plow_omited AND shiber1_closed AND NOT
              transp1_work THEN
               alarm_string:='Неисправность  транспортера
                              загрузки';
               alarm:=TRUE;
          END_IF
     END_IF
     (*бункер не наполнен в течение контрольного времени*)
     IF timer_load.Q AND  (bunker1_level < current_task)  THEN
      (*спустя 40 сек. после начала процесса загрузки*)
          alarm_string:='Отсут. подачи/ неисправн. измер.
                         уровня бункера 1';
          alarm:=TRUE;
     END_IF
END_IF

(*Выгрузка из бункера 2*)
IF truck AND NOT bunker2_empty THEN
     unload_enable:=TRUE;
END_IF


timer_unload(IN:=unload_enable,PT:=t#40s); (*контрольное
                                             время выгрузки*)
signal2:=unload_enable AND timer_unload.ET < t#2s; (*предупр.
                                                  сигнализация*)
(*управление разгрузочным транспортером и шибером бункера 2*)
shiber2_open:=NOT shiber2_opened AND truck AND
              transp2_work AND NOT signal2;
shiber2_close:=FALSE;
```

```
timer_transp2(IN:= unload_enable AND NOT bunker2_empty  AND
             NOT signal2, PT:=t#5s,Q=>transp2_turn_on);

unload_success:=bunker2_empty AND NOT transp2_work; (*выг-
                                                  рузка завершена*)

(*обработка ошибок при выгрузке*)
IF NOT unload_success THEN

     IF unload_enable AND NOT truck THEN (*тележка исчезла!*)
          alarm_string:='Тележка удалена';
          alarm:=TRUE;
     END_IF

     IF  (timer_unload.ET>t#7s)  THEN (*спустя 7 сек. после
                                    начала процесса выгрузки*)
          IF NOT bunker2_empty AND NOT transp2_work THEN
               (*неисправность разгрузочного транспортера*)
               alarm_string:='Неисправность транспортера
                              разгрузки';
               alarm:=TRUE;
          END_IF
          IF NOT shiber2_opened AND transp2_work THEN (*шибер
                                  второго бункера не открыт*)
               alarm_string:='Неисправность при открытии
                              шибера бункера 2';
               alarm:=TRUE;
          END_IF
     END_IF

     (*бункер не опустошен в течение контрольного времени*)
     IF timer_unload.Q AND  NOT bunker2_empty  THEN
          alarm_string:='Неисправность измерителя уровня
                         бункера 2';
          alarm:=TRUE;
     END_IF

END_IF

signal:=signal1 OR signal2; (*управление сигнализацией*)

(*Переходы в другие режимы*)
IF NOT (SA1_auto AND start_prog) THEN (* переход в состояние
                                       0*)
     timer_transp2(IN:=FALSE);
     timer_load(IN:=FALSE);
     timer_unload(IN:=FALSE);
     start_prog:=FALSE;
     load_success:=FALSE;
     unload_success:= FALSE;
     unload_enable:=FALSE;
     status:=0;
```

```
          RETURN;
     END_IF

     IF alarm OR (load_success AND unload_success) THEN (*общее
                                 условие перехода в режимы 2,3,4*)
          timer_transp2(IN:=FALSE);
          timer_load(IN:=FALSE);
          timer_unload(IN:=FALSE);
          load_success:=FALSE;
          unload_success:= FALSE;

          IF alarm THEN (*переход в режим 4 (авария)*)
               unload_enable:=FALSE;
               status:=4;
               RETURN;
          ELSIF NOT unload_enable AND truck THEN (*выгрузка практи-
                    чески не состоялась и тележка осталась пустой*)
               status:=2;
               RETURN;
          ELSE (*все было "правильно" или нет тележки*)
               unload_enable:=FALSE;
               status:=3;
               RETURN;
          END_IF

     END_IF

2: (*Режим 2: наполнение бункера №2, опустошение бункера №1*)

     alarm:=FALSE;  alarm_string:=''; (*пока все хорошо*)

     (*Загрузка бункера 2*)
     timer_load(IN:=bunker2_level< current_task,PT:=t#40s);(*конт-
                                              рольное время загрузки*)
     signal1:=bunker2_level< current_task AND
              timer_load.ET < t#2s;  (*предупр. сигнализация*)

     (*управление загрузочным транспортером, плужком и шибером
      бункера 1*)
     transp1_turn_on:=plow_raised AND shiber2_closed AND
                      (bunker2_level < current_task) AND NOT
                     bunker2_full AND NOT signal1;
     plow_down:=FALSE;
     plow_up:=NOT plow_raised AND NOT signal1;
     shiber2_open:=FALSE;
     shiber2_close:=NOT shiber2_closed AND NOT signal1;
     load_success:=bunker2_level > current_task AND
                   NOT transp1_work;  (*загрузка завершена*)

     (*обработка ошибок при загрузке*)
     IF NOT load_success THEN
```

```
     IF timer_load.ET>t#7s THEN (*спустя 7 сек. после начала
                                  процесса загрузки*)
          IF NOT plow_raised THEN (*плужок не опущен*)
               alarm_string:='Неисправность плужкового
                              сбрасывателя';
               alarm:=TRUE;
          END_IF

          IF   NOT shiber2_closed THEN (*шибер второго бункера
                                         не закрыт*)
               alarm_string:='Неисправность при закрытии шибе-
                              ра бункера 2';
               alarm:=TRUE;
          END_IF

          (*неисправность загрузочного транспортера*)
          IF (bunker2_level < current_task) AND plow_raised
              AND shiber2_closed AND NOT transp1_work THEN
               alarm_string:='Неисправность  транспортера
                              загрузки';
               alarm:=TRUE;
          END_IF

     END_IF

     (*бункер не наполнен в течение контрольного времени*)
     IF timer_load.Q AND  (bunker2_level < current_task)  THEN
          (*спустя 40 сек. после начала процесса загрузки*)
          alarm_string:='Отсут. подачи/ неисправн. измер.
                         уровня бункера 2';
          alarm:=TRUE;
     END_IF
END_IF

(*Выгрузка из бункера 1*)
IF truck AND NOT bunker1_empty THEN
     unload_enable:=TRUE;
END_IF

timer_unload(IN:=unload_enable,PT:=t#40s);  (*контрольное
                                              время выгрузки*)
signal2:=unload_enable AND timer_unload.ET < t#2s;
                                    (*предупр. сигнализация*)

(*управление разгрузочным транспортером и шибером бункера 1*)
shiber1_open:=NOT shiber1_opened AND truck AND
              transp2_work AND NOT signal2;
shiber1_close:=FALSE;
timer_transp2(IN:=unload_enable AND NOT bunker1_empty  AND
              NOT signal2, PT:=t#5s,Q=>transp2_turn_on);
```

```
unload_success:=bunker1_empty AND NOT transp2_work; (*выг-
                                                  рузка завершена*)
(*обработка ошибок при выгрузке*)
IF NOT unload_success THEN

     IF unload_enable AND NOT truck THEN (*тележка исчезла!*)
          alarm_string:='Тележка удалена';
          alarm:=TRUE;
     END_IF

     IF  (timer_unload.ET>t#7s)  THEN (*спустя 7 сек. после
                                   начала процесса выгрузки*)
          IF NOT bunker1_empty AND NOT transp2_work THEN
               (*неисправность разгрузочного транспортера*)
               alarm_string:='Неисправность транспортера
                              разгрузки';
               alarm:=TRUE;
          END_IF

          IF NOT shiber1_opened AND transp2_work THEN (*шибер
                                    первого бункера не открыт*)
               alarm_string:='Неисправность при открытии
                              шибера бункера 1';
               alarm:=TRUE;
          END_IF
     END_IF

     (*бункер не опустошен в течение контрольного времени*)
     IF timer_unload.Q AND  NOT bunker1_empty  THEN (*спустя
                     40 сек. после начала процесса выгрузки*)
          alarm_string:='Неисправность измерителя уровня
                         бункера 1';
          alarm:=TRUE;
     END_IF
END_IF
signal:=signal1 OR signal2; (*управление сигнализацией*)

(*переходы в другие режимы*)
IF NOT (SA1_auto AND start_prog) THEN (*переход в состояние
                                        0*)
     timer_transp2(IN:=FALSE);
     timer_load(IN:=FALSE);
     timer_unload(IN:=FALSE);
     start_prog:=FALSE;
     load_success:=FALSE;
     unload_success:= FALSE;
     unload_enable:=FALSE;
     status:=0;
     RETURN;
END_IF

IF alarm OR (load_success AND unload_success) THEN (*общее
                           условие перехода в режимы 2,3,4*)
```

```
          timer_transp2(IN:=FALSE);
          timer_load(IN:=FALSE);
          timer_unload(IN:=FALSE);
          load_success:=FALSE;
          unload_success:= FALSE;
          IF alarm THEN (*переход в режим 4 (авария)*)
               unload_enable:=FALSE;
               status:=4;
               RETURN;
          ELSIF NOT unload_enable AND truck THEN (*выгрузка практи-
                    чески не состоялась, и тележка осталась пустой*)
               status:=1;
               RETURN;
          ELSE (*все было "правильно" или нет тележки*)
               unload_enable:=FALSE;
               status:=3;
               RETURN;
          END_IF
     END_IF

3: (*Режим 3: смена тележки*)
     (*не автоматический режим - переход в состояние 0*)
     IF NOT SA1_auto THEN
          timer_transp2(IN:=FALSE);
          timer_load(IN:=FALSE);
          timer_unload(IN:=FALSE);
          start_prog:=FALSE;
          status:=0;
          RETURN;
     END_IF

     IF NOT change_status THEN
          IF NOT truck THEN (*тележку забрали или ее и не было*)
               change_status:=1;
          ELSE (*тележка на месте - подача сигнала на замену
                  тележки*)
               timer_truck(IN:=TRUE,PT:=t#7s);
               truck_change:= timer_truck.ET < t#2s ;
               IF timer_truck.Q THEN timer_truck(IN:=FALSE); END_IF
          END_IF
     ELSE
          IF truck THEN (*тележка появилась - можно начинать новый
                         цикл*)
               change_status:=0;
               timer_truck(IN:=FALSE);
               IF bunker1_empty THEN status:=1;
               ELSE status:=2;
               END_IF
          ELSE (*тележка на месте - подача сигнала на замену
                  тележки*)
               timer_truck(IN:=TRUE,PT:=t#7s);
               truck_change:= timer_truck.ET < t#2s ;
               IF timer_truck.Q THEN timer_truck(IN:=FALSE); END_IF
```

```
          END_IF
     END_IF
4: (*Режим 4: авария*)

     (*не автоматический режим - переход в состояние 0*)
     IF NOT SA1_auto THEN
          timer_transp2(IN:=FALSE);
          timer_load(IN:=FALSE);
          timer_unload(IN:=FALSE);
          start_prog:=FALSE;
          status:=0;
          RETURN;
     END_IF

     (*выключение всего, что можно выключить*)
     transp1_turn_on:=shiber1_open:=shiber2_open:=plow_up:=
                     plow_down:=FALSE;

     IF alarm_string<>'Неисправность при закрытии шибера
                       бункера 1' THEN
          shiber1_close:=NOT shiber1_closed;
     ELSE
          shiber1_close:=FALSE;
     END_IF

     IF alarm_string<>'Неисправность при закрытии шибера
                       бункера 2' THEN
          shiber2_close:=NOT shiber2_closed;
     ELSE
          shiber2_close:=FALSE;
     END_IF

     (*безопасный останов транспортера разгрузки*)
     IF alarm_string<>'Неисправность транспортера разгрузки' THEN
          timer_transp2(IN:=(NOT bunker1_empty  AND NOT
                      shiber1_closed) OR (NOT bunker2_empty AND NOT
                      shiber2_closed),PT:=t#5s,Q=>transp2_turn_on);
     ELSE
          transp2_turn_on:=FALSE;
     END_IF
     signal:=TRUE; (*сигнализация*)
     (*сброс аварии*)
     IF reset THEN
          start_prog:=FALSE;
          status:=0;
     END_IF

END_CASE
```

Программа может находиться в одном из пяти состояний:

0 – ожидание;

1 – наполнение бункера 1, опустошение бункера 2;

2 – наполнение бункера 2, опустошение бункера 1;
3 – смена тележки;
4 – авария.
Схема смены состояний показана на рис. 3.151.

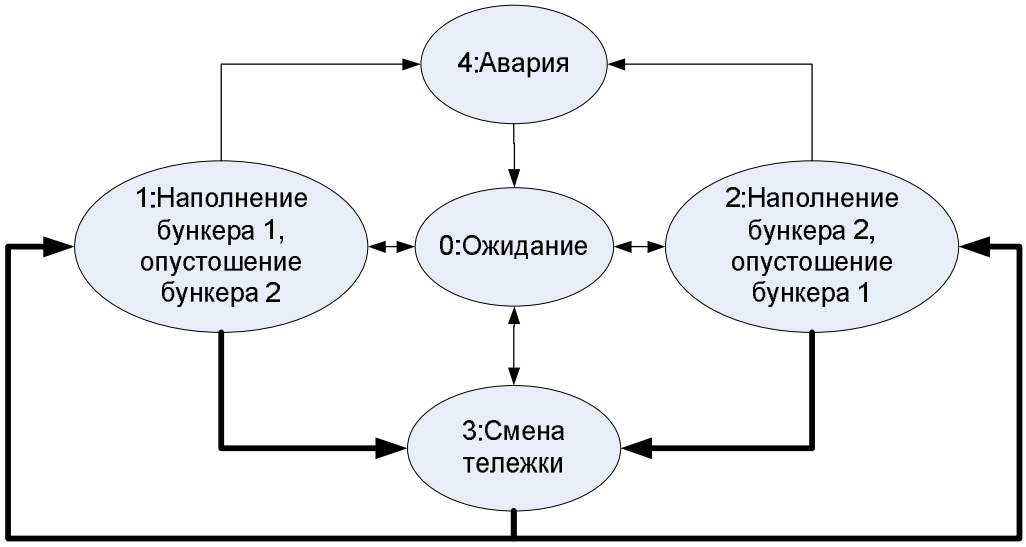


Рис. 3.151. Состояния программы `automat`.

При запуске программы она автоматически оказывается в режиме ожидания. В этом режиме все выходы контроллера переводятся в безопасное состояние (выключено). При переходе в режим автоматического управления программа в зависимости от наличия материала в бункерах и наличия тележки выбирает одно из трех состояний: 1, 2 или 3.

Основной цикл изменений состояния в автоматическом режиме показан жирными стрелками: $1 \Rightarrow 3 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1 \Rightarrow 3 \Rightarrow \ldots$

Переход в состояние 4 («авария») производится из состояний 1 и 2 при неисправностях аппаратуры и «внезапном» исчезновении тележки. Неисправности аппаратуры фиксируются в случаях, когда спустя определенное время после подачи команд управления отсутствуют ожидаемые «сигналы подтверждения» от соответствующих датчиков. В состоянии 3 («смена тележки») переход в состояние 4 не предусмотрен: заполненная тележка может сколь угодно долго оставаться на месте, а «новая» (пустая) – сколь угодно долго не поставляться. В таких случаях программа периодически формирует сигнал запроса на замену тележки «внешнему миру».

3.2.1.4. Подсистема архивирования

Разработка имитационной модели в предыдущем пункте в целом была завершена. Подсистема архивирования может рассматриваться как опциональная надстройка, тем более, что действовать она будет совершенно независимо от основного программного комплекса.

Задачи системы:

1) сохранение в бинарном файле специального формата системной информации: даты и времени пуска системы в эксплуатацию, общего количества

отгруженных тележек, суммарной массы отгруженного материала, а также даты последнего запуска системы, количества отгруженных тележек и суммарной массы отгруженного материала за последнюю смену (далее будем называть это «общим архивом», хотя слово «архив» здесь и не совсем уместно);

2) сохранение в текстовых файлах детальной информации о каждой отгруженной тележке: ее порядкового номера, даты и времени, массы материала, режима работы системы при загрузке (ручной / автоматический), а также о том, фиксировались ли аварии при работе системы. На каждую смену (дату) будет формироваться отдельный файл, который далее мы будем называть «текущим архивом».

Предполагается, что ПЛК имеет файловую систему, «разрешает» программе создавать и удалять файлы, читать их содержимое и записывать в них информацию. Применительно к контроллерам, программируемым с помощью CoDeSys, это означает, что имеется реализация системной библиотеки SysLibFile.lib для целевой платформы. В дополнение к этому, поскольку на каждую дату будет создаваться отдельный текстовый файл, потребуется средство просмотра директории на предмет поиска «старых» файлов, которые необходимо удалить, чтобы не «замусоривать» файловую систему. Эту задачу решает системная библиотека SysLibDir.lib, которая также должна быть реализована для конкретного ПЛК.

Поскольку в архивах, естественно, будут фиксироваться даты и время, необходимо, чтобы ПЛК имел на борту энергонезависимые часы. Запрос текущей даты и времени в программе CoDeSys производится с помощью функции `SysRtcGetTime` из системной библиотеки SysLibRtc.lib.

Для виртуального контроллера PLC WinNT все указанные выше библиотеки имеются и поставляются бесплатно вместе со средой разработки CoDeSys. Для того чтобы воспользоваться их средствами, необходимо подключить их к проекту в «Менеджере библиотек».

Определимся с именами и форматами файлов.

Бинарный файл *«общего архива»* будет называться sysinfo (без расширения) и содержать одну единственную одноименную структуру, тип которой необходимо объявить во вкладке «Типы данных» (рис. 3.152).

Каждый текстовый файл *текущего архива* будет иметь имя, содержащее дату, и расширение .txt, например, 2018-10-27.txt – текущий архив на 27 октября 2018 года. Файлы будут содержать строки вида:

```
1    2018-10-27-09:07:56 53.4      автомат
2    2018-10-27-09:18:21 60.9      ручное
3    2018-10-27-09:25:46 25.4      руч,авт
4    2018-10-27-09:49:11 25.9      автомат   авария
```

Из приведенного фрагмента, например, видно, что тележка №1 была отправлена в 9.07 с массой материала 53.4, и ее загрузка происходила полностью в автоматическом режиме и без аварий. Тележка №2 загружалась в ручном режиме, а тележка №3 – частично в ручном, частично в автоматическом. При загрузке тележки №4 произошла авария (она была устранена, а загрузка возобновлена по-прежнему в автоматическом режиме).

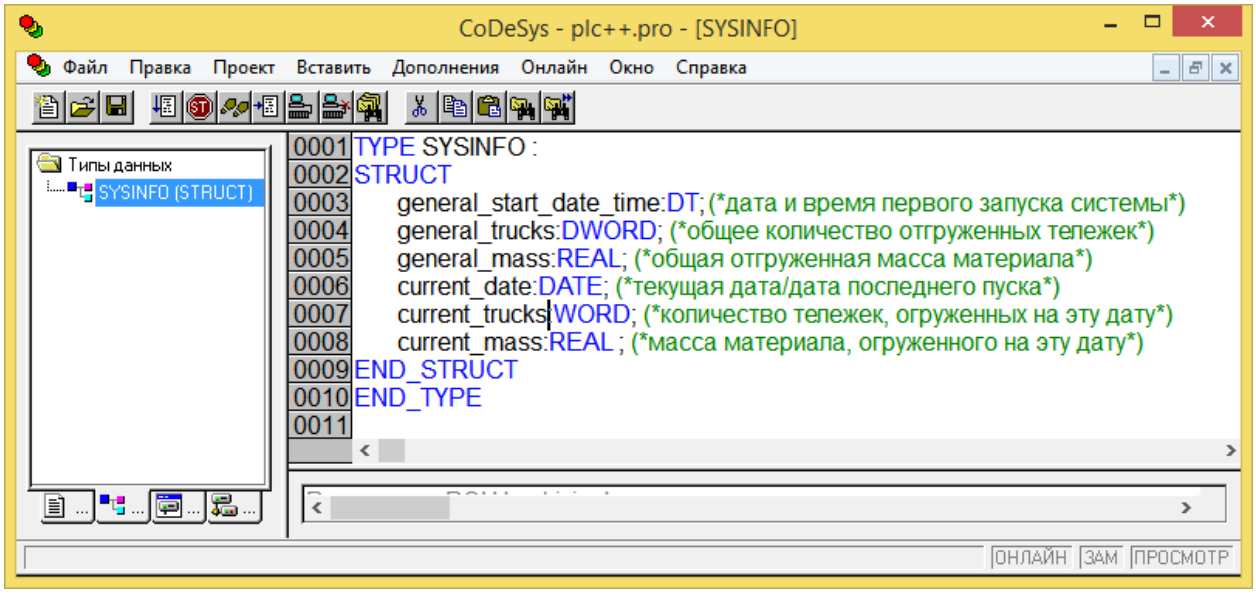


Рис. 3.152. Объявление типа `SYSINFO`.

В список глобальных переменных добавлены следующие переменные:

```
current_date_time:DT; (*текущие дата и время*)
current_date:DATE; (*текущая дата*)
DIR_PATH:STRING:='C:\Users\1\Desktop\Archive\';
(* - путь к директории архива*)
```

Строго говоря, `DIR_PATH` является не переменной, а константой, введенной лишь для того, чтобы упростить редактирование программы при изменении директории архива (это может потребоваться, например, при запуске проекта на другом компьютере).

Ниже приведены листинги основной программы архивирования `archiving` и вспомогательной программы `archive_clearning`, предназначенной для очистки архива от ненужных и устаревших файлов.

```
PROGRAM archiving
VAR
     bunker1_level_max, bunker2_level_max:REAL; (*максимальные
                                   фиксируемые уровни в бункерах, %*)
     truck_mass:REAL:=0; (*масса тележки, %*)
     bunker1_unload, bunker2_unload:BOOL; (*флаги разгрузки
                                          бункеров*)
     auto, manual:BOOL:=FALSE; (*флаги режимов работы*)
     normal:BOOL:=TRUE; (*флаг нормальности (безаварийности)
                         загрузки*)
     FileD:DWORD; (*дескриптор файла*)
     FileName, str:STRING(255); (*строка с именем файла, строка
                                  самого файла текущего архива*)
     si:SYSINFO; (*структура "Системная информация", сохраняемая в
                  общем архиве*)
     n: DWORD; (*количество считанных/записанных байт при работе с
                файлом*)
     alarm_string:STRING(50); (*сообщение об ошибке*)
END_VAR
```

```
-----------------------------------------------------------------
IF truck THEN (*Тележка на месте ведем мониторинг*)
     (*Фиксация начала разгрузки бункера 1*)
     IF NOT bunker1_unload AND NOT shiber1_closed  AND
        NOT bunker1_empty  THEN
          bunker1_unload:=TRUE;
          bunker1_level_max:=bunker1_level;
     END_IF
     (*Фиксация начала разгрузки бункера 2*)
     IF NOT bunker2_unload AND NOT shiber2_closed  AND
        NOT bunker2_empty THEN
          bunker2_unload:=TRUE;
          bunker2_level_max:=bunker2_level;
     END_IF
     (*Фиксация конца разгрузки бункера 1*)
     IF  bunker1_unload AND  (shiber1_closed OR bunker1_empty)
     THEN
          bunker1_unload:=FALSE;
          IF bunker1_empty THEN
               truck_mass:=truck_mass+bunker1_level_max;
          ELSE
               truck_mass:=truck_mass+bunker1_level_max -
                          bunker1_level;
          END_IF
     END_IF
     (*Фиксация конца разгрузки бункера 2*)
     IF  bunker2_unload AND  (shiber2_closed OR bunker2_empty)
     THEN
          bunker2_unload:=FALSE;
          IF bunker2_empty THEN
               truck_mass:=truck_mass+bunker2_level_max;
          ELSE
               truck_mass:=truck_mass+bunker2_level_max -
                          bunker2_level;
          END_IF
     END_IF
     (*Фиксация флагов*)
     IF SA1_auto THEN auto:=TRUE; END_IF
     IF NOT SA1_auto THEN manual:=TRUE; END_IF
     IF automat.alarm THEN normal:=FALSE; END_IF

ELSE (*Тележки нет, возможно, пора обновлять архивы*)
     IF truck_mass>0 THEN (*пора обновить архивы - тележки нет,
                            а масса есть*)
          current_date_time:=SysRtcGetTime(TRUE);
          current_date:=DT_TO_DATE(current_date_time);
          (*ОБНОВЛЕНИЕ ОБЩЕГО АРХИВА*)
          FileD:=SysFileOpen(CONCAT(DIR_PATH,'sysinfo'),'r'); (*по-
                                    пытка открыть файл для чтения*)
          n := SysFileRead(FileD,ADR(si),SIZEOF(si)); (*попытка
                                                        чтения*)
          SysFileClose(FileD);
```

```
IF n=0 THEN (*файл не был открыт или пуст – формирование
              первой записи в новый файл*)
     si.general_start_date_time:=current_date_time;
     si.general_trucks:=1;
     si.general_mass:=truck_mass;
     si.current_date:=current_date;
     si.current_trucks:=1;
     si.current_mass:=truck_mass;
     alarm_string:='Ошибка чтения системного файла';
ELSE (*обновление файла*)
     si.general_trucks:=si.general_trucks+1;
     si.general_mass:=si.general_mass+truck_mass;
     IF si.current_date<>current_date THEN (*начинается
                                            новый день*)
          archive_clearning; (*очистка архива от ненужных
                              и старых файлов*)
          si.current_date:=current_date;
          si.current_trucks:=1;
          si.current_mass:=truck_mass;
     ELSE (*день продолжается*)
          si.current_trucks:=si.current_trucks+1;
          si.current_mass:=si.current_mass+truck_mass;
     END_IF
END_IF
     FileD:=SysFileOpen(CONCAT(DIR_PATH,'sysinfo'),'w');
                    (*- попытка открыть файл для записи*)
     n := SysFileWrite(FileD,ADR(si),SIZEOF(si)); (*по-
                                          пытка  записи*)
     SysFileClose(FileD);
     IF n=0 THEN  alarm_string:='Ошибка записи системного
                              файла'; END_IF

(*ОБНОВЛЕНИЕ ТЕКУЩЕГО АРХИВА*)
(*Формирование имени полного имени файла*)
FileName:=CONCAT(CONCAT(DIR_PATH,
         RIGHT(DATE_TO_STRING(current_date),10)),'.txt');
FileD:=SysFileOpen(FileName,'a'); (*попытка открыть файл
                                    для дополнения*)
(*Формирование строки в текстовом файле*)
str:=WORD_TO_STRING(si.current_trucks);
str:=CONCAT(str, '$t');
str:=CONCAT(str, RIGHT(DT_TO_STRING(current_date_time),
           19));
str:=CONCAT(str, '$t');
str:=CONCAT(str, LEFT(REAL_TO_STRING(truck_mass),6));
str:=CONCAT(str, '$t');

IF auto AND NOT manual THEN
     str:=CONCAT(str, 'автомат');
ELSIF NOT auto AND  manual THEN
     str:=CONCAT(str, 'ручное');
ELSE
     str:=CONCAT(str, 'руч,авт');
```

```
          END_IF
          IF NOT normal THEN
               str:=CONCAT(str, '$t');
               str:=CONCAT(str, 'авария');
          END_IF
          str:=CONCAT(str, '$R$N');
          n:=SysFileWrite(fileD,ADR(str),LEN(str));  (*попытка
                                                       записи*)
          SysFileClose(fileD);
          IF n=0 THEN  alarm_string:='Ошибка записи файла текущего
                                     архива'; END_IF
     END_IF
     (*Сброс накопленных данных*)
     truck_mass:=0;
     bunker1_unload:=FALSE;
     bunker2_unload:=FALSE;
     auto:= manual:=FALSE;
     normal:=TRUE;
END_IF
IF reset THEN alarm_string:=''; END_IF (*сброс сообщения об
                                        ошибке*)
-------------------------------------------------------------------

PROGRAM archive_clearning
VAR
     dirD:DWORD; (*дескриптор директории*)
     stDirEntry : STRING(80); (*строка для записи имени файла*)
     pDirInfo : POINTER TO DIRECTORY_INFO; (*указатель на структу-
                                    ру с информацией о директории*)
     filelist : ARRAY [1..10] OF STRING(80); (*масиив имен
                                              файлов*)
     n:WORD:=1; (*счетчик цикла*)
     filedate: DATE; (*дата из имени файла*)
END_VAR

-------------------------------------------------------------------

(*Очистка массива*)
FOR n:=1 TO 10 DO
     filelist[n]:='';
END_FOR
n:=1;
dirD:=SysDirOpen(DIR_PATH);
(*Чтение директории*)
WHILE SysDirRead(dirD,stDirEntry, pDirInfo) <>0 DO
     IF stDirEntry<>'.' AND stDirEntry<>'..' AND
        stDirEntry<>'sysinfo' THEN
          filelist[n]:=stDirEntry;
          n:=n+1;
     END_IF
     IF n>10 THEN
          EXIT;
     END_IF
```

```
END_WHILE

(*Удаление ненужных файлов*)
FOR n:=1 TO 10 DO
     IF  filelist[n]='' THEN (*список файлов закончен*)
          EXIT;
     ELSIF  LEN(filelist[n])<>14  OR   RIGHT(filelist[n],4)<>'.txt'
THEN
(* - длина имени файла или его расширение неправильные*)
          SysFileDelete(CONCAT(DIR_PATH, filelist[n]));
     ELSE
          filedate := STRING_TO_DATE(CONCAT ('D#',
                      LEFT(filelist[n],10)));
     (* - формирование даты из имени файла*)
          IF filedate=D#1970-01-01 OR DATE_TO_TIME (current_date) -
             DATE_TO_TIME(filedate) >= T#240h
     THEN
               SysFileDelete(CONCAT(DIR_PATH,filelist[n]));
          END_IF
     END_IF
END_FOR
```

Программа `archiving` работает «параллельно» с основной программой `PLC_PRG`. Обе они запускаются теперь как *свободные задачи*, для каждой из которых можно независимо настроить свой сторожевой таймер (watch dog). И хотя в нашем проекте для PLC WinNT сторожевые таймеры не были активированы вовсе, для реальных контроллеров такая возможность может оказаться весьма полезной. Дело в том, что операции с файлами занимают намного больше времени, чем обычные вычисления и «общий» сторожевой таймер может не справиться: при обновлении архива выполнение одной большой программы существенно затягивается. Так, первая версия программы для PLC WinNT без отдельных задач, с вызовом `archiving` из `PLC_PRG`, при попытке создания нового файла прекращала работу с выводом сообщения от сторожевого таймера (хотя создать файл и успевала).

Конечно, такое решение по сути всего лишь обман системы контроля времени исполнения, ведь при обновлении архивов выполнение основной программы просто откладывается и выходы контроллера не обновляются (если только ОС контроллера не поддерживает многозадачность, что маловероятно). Однако в нашем случае все не так плохо. Во-первых, работа с архивами будет происходить при отсутствии тележки, когда программа автоматического управления ничем другим, кроме ожидания новой, не занимается, и, следовательно, может и подождать, пока не закончится работа с архивами (в нашей имитационной системе «подождут» и «электрические схемы» с «щитом управления»). Во-вторых, работа с файлами не потребует слишком уж много времени даже в случае, когда все операции проходят успешно. В случае неудачи программа архивирования, вероятно, будет выполняться еще быстрее. Если, например, не удалось открыть файл, библиотечная функция открытия файла возвращает нулевой дескриптор, получив который, функция чтения/записи сразу же завершает свою работу и возвращает нуль (количество считанных/записанных байт).

Таким образом, архивирование не должно нарушить ход выполнения основной программы.

Кратко прокомментируем код программ.

Программа `archiving` при наличии тележки занимается мониторингом: фиксирует максимальные уровни в бункерах, которые имеют место в моменты времени, когда бункеры начинают разгружаться. В моменты окончания разгрузки вычисляется масса отгруженного материала. Помимо этого фиксируются «дополнительные условия»: режим работы системы и факты аварий.

При отсутствии тележки, если рассчитанная при мониторинге масса отлична от нуля, происходит обновление архивов. Открывается и обновляется общий архив с системной информацией. При изменении даты (новая смена) запускается программа очистки директории `archive_clearning` и начинается новый отчет тележек и общей массы на дату. Если файл sysinfo не удалось открыть или прочитать, этот файл создается заново, и общий архив «начинается сначала».

Далее обновляется (или создается новый, если его еще нет) текущий архив. По текущей дате формируется имя текстового файла, и он открывается для дополнения. С помощью функции объединения строк `CONCAT` формируется новая строка, которая и записывается в конец файла.

После этого вся накопленная на этапе мониторинга информация «очищается» путем записи в соответствующие переменные начальных значений.

Рис. 3.153. Модификация панели оператора для вывода сообщений об ошибках архивирования.

При выполнении всех операций с файлами фиксируются ошибки. Описание ошибки, если она произошла, заносится в локальную строковую переменную `alarm_string`. Оно будет отображаться на панели оператора, для чего на панель требуется добавить еще одну строку вывода (ниже строки вывода сообщений об ошибках управления), см. рис. 3.153.

Сброс сообщения об ошибке архивирования, также как и для ошибки управления выполняется при помощи кнопки «Сброс».

Программа `archive_clearning` очищает директорию архива от всех файлов с неправильными именами, а также от архивов десятидневной и более давности.

В предварительно очищенный строковый массив заносятся имена всех файлов директории, кроме sysinfo, «.» (текущая директория), «..» (родительская директория). Далее массив просматривается до последней записи. Если обнаружилась строка, длина которой не равна 14 или не оканчивающаяся на «.txt», файл с соответствующим именем удаляется. После этого предпринимается попытка создать дату из имени файла. Если попытка не удается или сформиро-

ванная дата «старше» текущей на 10 и более дней (240 часов), файл также удаляется.

Отметим, что в программе не выполняется проверка доступности директории. В этом нет необходимости, так как в случае отсутствия директории до вызова `archive_clearning` дело даже не дойдет. Сообщение же об ошибке сформирует программа `archiving`, когда не сможет открыть файлы.

3.2.1.5. Подсистема мониторинга

Данная подсистема будет имитировать работу удаленной инженерной станции и осуществлять мониторинг технологического процесса, включая и работу с архивными файлами. Для решения этих задач будет задействована SCADA-система Trace Mode, однако, в отличие от систем регулирования, рассмотренных ранее, экран визуализации не будет содержать органов воздействия на процесс. Функции пользователя будут ограничены наблюдением и работой с архивами. Поэтому имитация обмена данными между SCADA и ПЛК будет предельно упрощена (фактически об обмене говорить не приходится, так как имеет место однонаправленный поток информации от ПЛК к SCADA). В частности, мы не будем объявлять дополнительные переменные в программе контроллера, предназначенные исключительно для «сетевого» обмена, как это было сделано в п. 3.1. Вместо этого будем считать, что те переменные, которые уже есть, вполне доступны SCADA-системе. Кроме того, предположим, что имеется возможность копирования архивных файлов из ПЛК на жесткий диск компьютера по команде с экрана визуализации. В нашей имитационной системе это будет означать всего лишь копирование файлов из одной папки в другую.

Экран визуализации в работе показан на рис. 3.154.

На экране размещены:

1) упрощенная анимация технологического процесса. В отличие от экрана операторской панели здесь представлена только информация от датчиков и измерительных преобразователей. «Команды» системы управления (например, на пуск транспортера) не отображаются;

2) информация о режиме управления;

3) содержимое «общего архива» ПЛК и кнопки загрузки архивных файлов и открытия папки.

На рис. 3.155 показаны источники сигналов для SCADA (вся информация поступает от виртуального ПЛК по OPC).

Все переменные кроме первых шести имеют те же самые имена, что и переменные ПЛК, из которых считываются данные. Первые шесть переменных получают свои значения из структуры `si` типа `SYSINFO` программы `archiving`.

Все «аналоговые» и «временные» сигналы пропускаются через каналы классов FLOAT и HEX32 соответственно, рис. 3.156. Помимо «сигнальных» список каналов содержит два канала класса CALL с типом вызова Exec: «C:\Windows\explorer» и «copyfromPLC».

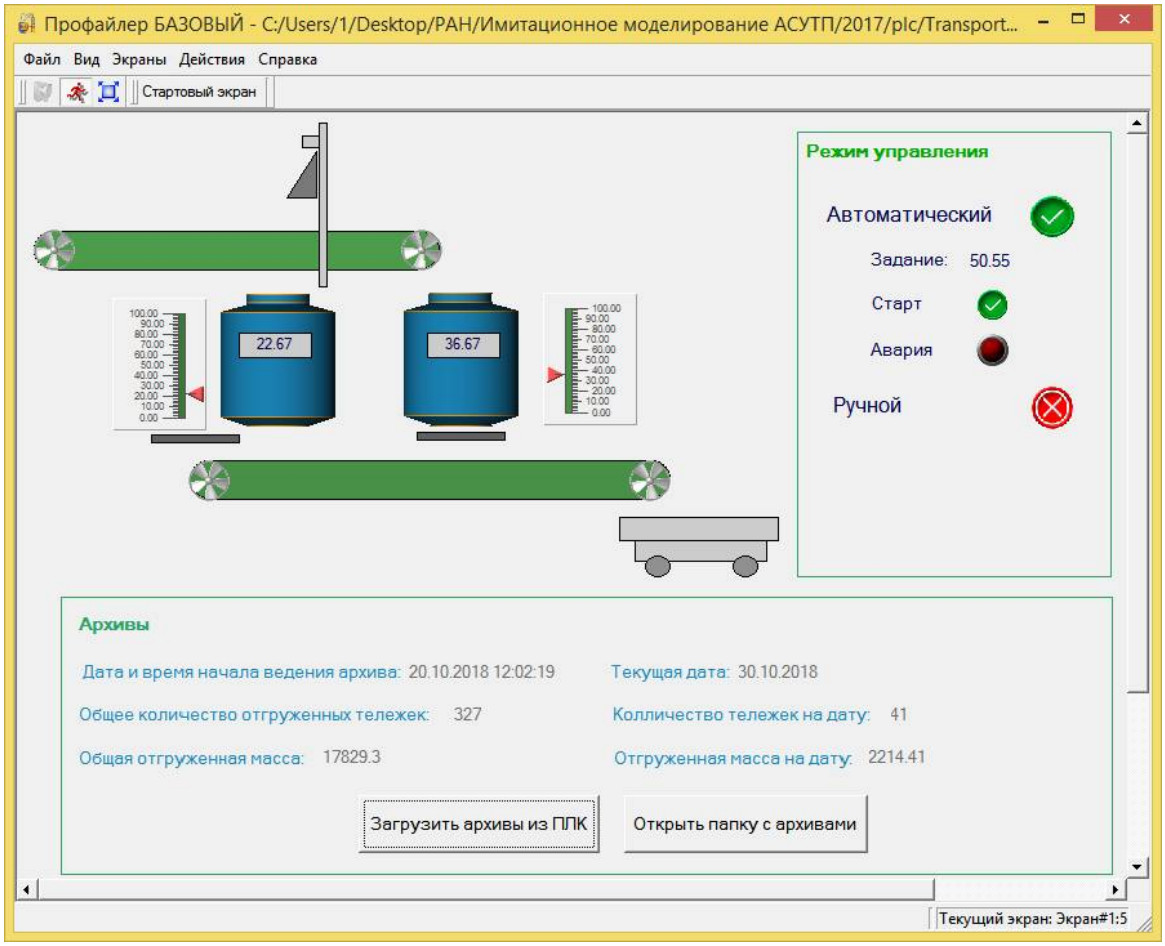


Рис. 3.154. Экран визуализации в работе.

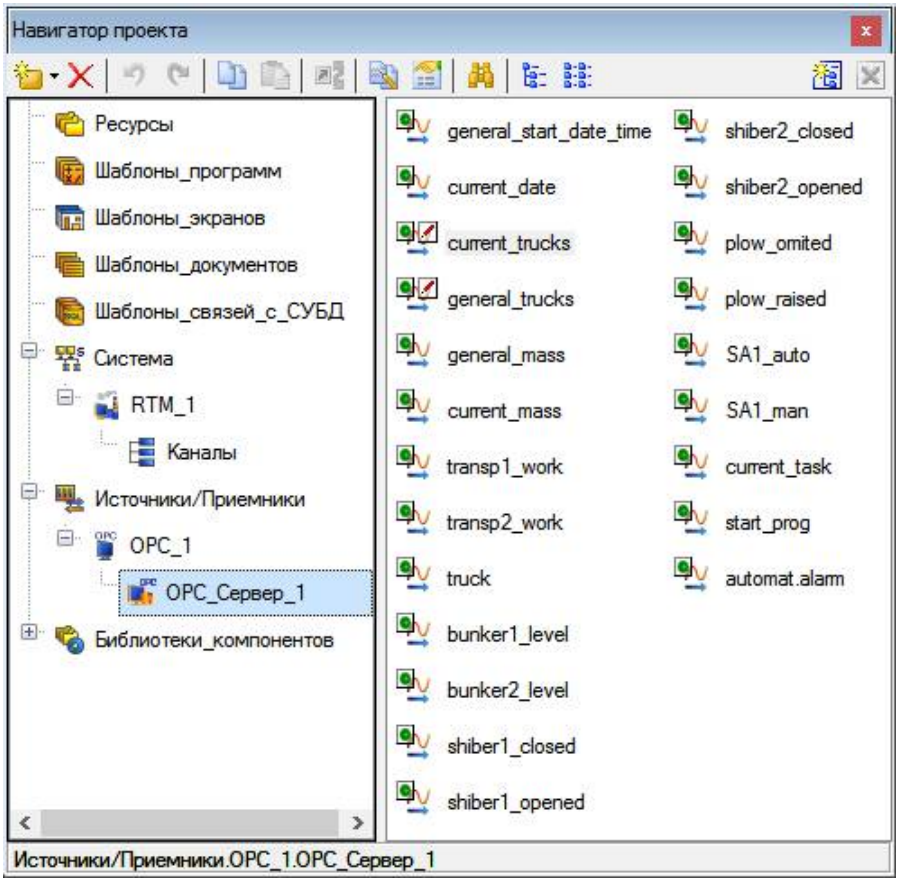


Рис. 3.155. Источники-приемники.

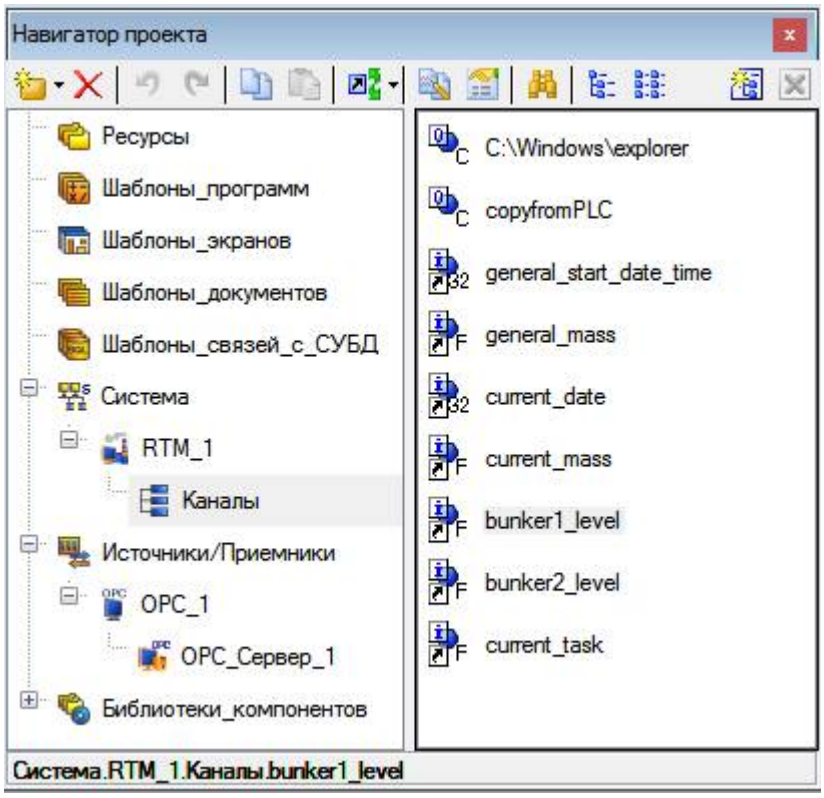


Рис. 3.156. Каналы.

Эти каналы предназначены для запуска внешних приложений: «Проводника» для просмотра папки с архивами и пакетного файла для копирования архивов из папки виртуального контроллера в папку SCADA-системы (фактически обе папки в нашем проекте размещаются на Рабочем столе). Настройки каналов показаны на рис. 3.157 и 3.158.

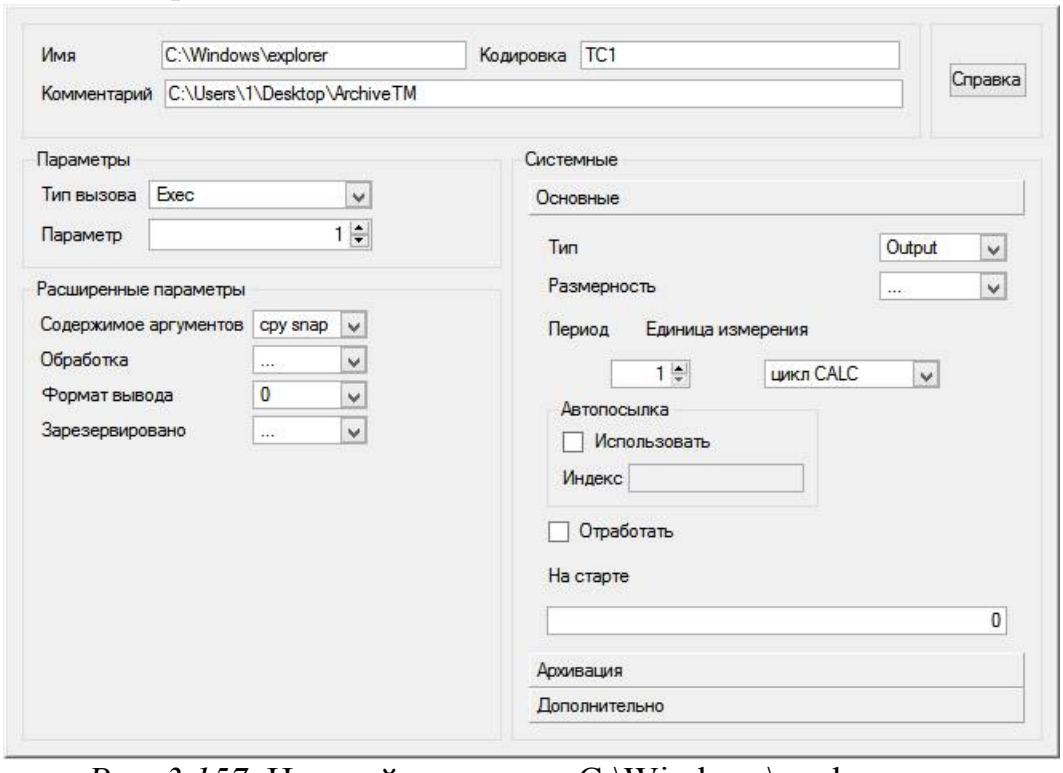


Рис. 3.157. Настройка канала «C:\Windows\explorer».

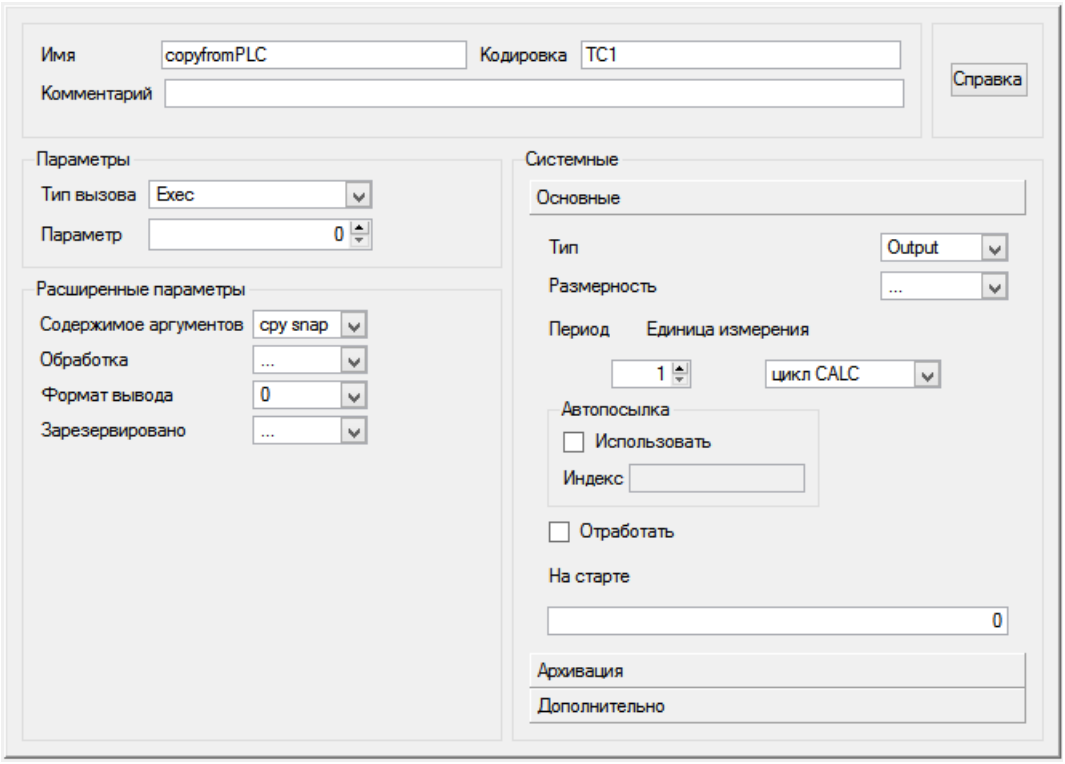


Рис. 3.158. Настройка канала «copyfromPLC».

Сам пакетный файл copyfromPLC.cmd размещается в папке RTM проекта и содержит строку вызова программы xcopy, которая копирует все файлы с расширением txt из «папки ПЛК» в «папку SCADA» игнорируя все «сомнения» и не задавая «вопросов»:

```
@start xcopy C:\Users\1\Desktop\Archive\*.txt C:\Users\1\Desktop\ArchiveTM /I/Y
```

Необходимость в командном файле связана с ограничением на максимальное количество символов в выполняемой строке канала.

Аргументы экрана показаны на рис. 3.159. Все они, за исключением последних двух, являются входными. Большинство аргументов прямо или через каналы получают информацию от одноименных источников. Три аргумента получают значения от программы:

```
Программа_1_8_shiber1_position;
Программа_1_8_shiber2_position;
Программа_1_8_plow_position.
```

Программа «рассчитывает» положение элементов визуализации, изображающих шиберы и плужок на основании сигналов их концевых выключателей примерно так же, как это делает программа PLC_PRG для панели оператора. Привязки аргументов программы показаны на рис. 3.160, а код программы приведен ниже.

Два выходных аргумента экрана записывают свои значения в описанные выше каналы класса CALL. Отметим, что для того, чтобы запустить программу на выполнение, достаточно записать любое ненулевое число в атрибут канала «Входное значение».

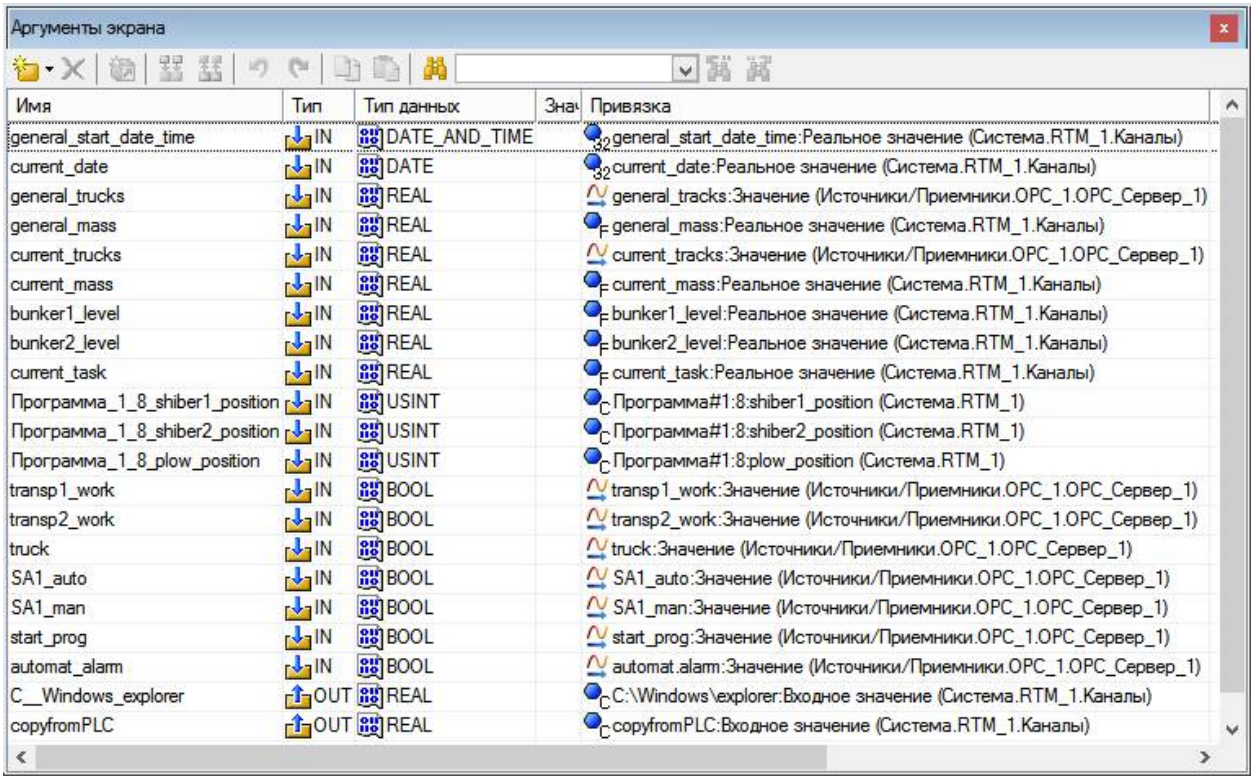


Рис. 3.159. Аргументы экрана.

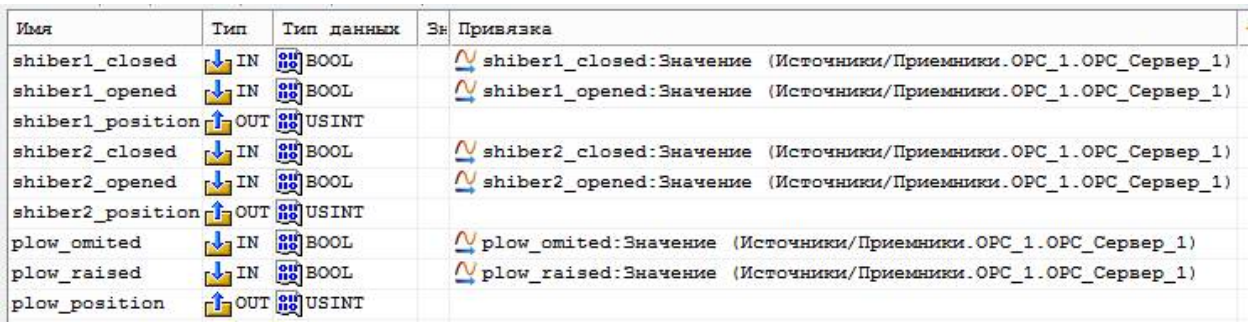


Рис. 3.160. Аргументы программы.

```
PROGRAM
     VAR_INPUT shiber1_closed : BOOL; END_VAR
     VAR_INPUT shiber1_opened : BOOL; END_VAR
     VAR_OUTPUT shiber1_position : USINT; END_VAR
     VAR_INPUT shiber2_closed : BOOL; END_VAR
     VAR_INPUT shiber2_opened : BOOL; END_VAR
     VAR_OUTPUT shiber2_position : USINT; END_VAR
     VAR_INPUT plow_omited : BOOL; END_VAR
     VAR_INPUT plow_raised : BOOL; END_VAR
     VAR_OUTPUT plow_position : USINT; END_VAR

     IF shiber1_closed THEN
          shiber1_position:=0;
     ELSIF shiber1_opened THEN
          shiber1_position:=100;
     ELSE
          shiber1_position:=50;
     END_IF;
```

```
    IF shiber2_closed THEN
         shiber2_position:=0;
    ELSIF shiber2_opened THEN
         shiber2_position:=100;
    ELSE
         shiber2_position:=50;
    END_IF;
    IF plow_omited THEN
         plow_position:=0;
    ELSIF plow_raised THEN
         plow_position:=100;
    ELSE
         plow_position:=50;
    END_IF;

END_PROGRAM
```

На рис. 3.161 показаны привязки графических элементов к аргументам экрана.

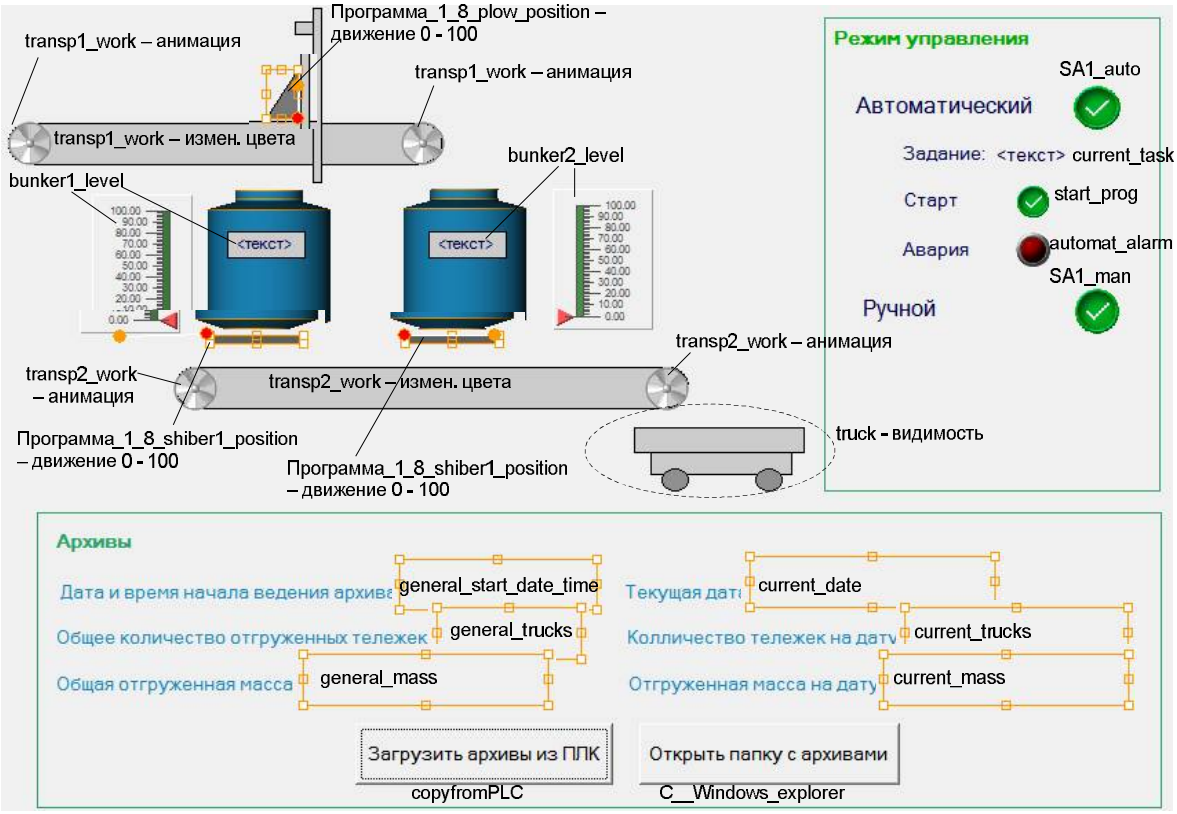


Рис. 3.161. Привязки графических элементов к аргументам экрана.

3.2.2. Обеспечение равномерного износа однотипных агрегатов

3.2.2.1. Постановка задачи

На практике часто требуется, чтобы однотипные агрегаты, совместно или по очереди решающие одну и ту же задачу, изнашивались равномерно.

Пусть, например, речь идет об откачке воды из некоторого резервуара, где она собирается из разных источников, причем предсказать заранее процесс наполнения резервуара невозможно. Откачку воды выполняют два насоса: один

рабочий, а другой *резервный*. Рабочий насос включается, если уровень в резервуаре достиг первой отметки. Резервный насос включается в случае отказа рабочего, а также в случае, если уровень достиг второй отметки. Было бы неразумным в качестве рабочего использовать всегда один и тот же насос: очевидно, что он будет изнашиваться намного быстрее, чем второй – резервный. Поэтому насосы должны периодически меняться ролями.

Усложним ситуацию. Пусть у нас имеется три насоса: два рабочих и один резервный. При достижении уровнем первой отметки должен быть запущен первый рабочий насос, при достижении второй отметки – второй. Резервный насос заменяет вышедший из строя рабочий, а кроме того включается, если несмотря на «старания» двух рабочих уровень все же достиг третьей отметки. Здесь должны меняться ролями не только рабочие и резервный насосы и два рабочих между собой, и мы имеем уже что-то вроде «первого рабочего» и «второго рабочего».

Возможно, стоит вообще отказаться от таких понятий как «резервный насос» и «меняться местами» и перейти к другим, более способствующим формализации задачи. Каждый насос может находиться в трех состояниях: «работа», «готов к работе» и «авария/ремонт». Кроме того, для каждого насоса фиксируется суммарная наработка за какой-то достаточно длительный интервал, например, год, или вообще с начала эксплуатации. Тогда алгоритм управления насосами сводится к следующему.

1. По сигналам с датчиков уровня и сигналам об отказах ранее работающих насосов система определяет, сколько насосов должно работать в настоящее время, и принимает решение о запуске или остановке одного или более насосов, но не каких-то конкретных, а пока что «абстрактных».

2. Если требуется запустить очередной насос, система выбирает его из списка готовых к работе, причем самым очевидным образом: выбирается тот, у которого суммарная наработка минимальна. Если требуется остановить очередной насос, то он выбирается из списка работающих, и, конечно же, это должен насос с максимальной суммарной наработкой.

Понятно, что такой простой алгоритм будет работать при любом количестве насосов (просто потому, что он не зависит от количества). Однако у него имеется один недостаток. А что будет, если в системе установится некоторый баланс и в течение длительного времени потребности в отключении работающего насоса не возникает, например, уровень не падает ниже отметки, на которой он включился? Очевидно, алгоритм должен быть скорректирован таким образом, чтобы насос принудительно отключался, что бы дать возможность поработать другим. Условием такого отключения может быть, например, достижение его *текущей наработки* определенного значения («долго уже работает»). Другой вариант – значительное превышение суммарной наработки над средней среди всех насосов. При этом, конечно, принудительно отключать насос ни в коем случае не следует, если список готовых к работе пуст.

Есть и другие «тонкости», которые должны быть учтены при детальной разработке алгоритма.

3.2.2.2. Разработка алгоритма и его программная реализация

Оформим новую сущность «насос» в виде функционального блока. Это даст нам возможность легко манипулировать несколькими экземплярами этой сущности. Функциональный блок будет запоминать свое состояние, по внешним командам переходить из одного состояния в другое, а также вести подсчет временных величин: общей и текущей наработки и времени простоя в состоянии готовности.

Код функционального блока приведен ниже. Использование комментариев и «осмысленных» имен позволяет легко его понять.

```
FUNCTION_BLOCK PUMP
VAR_INPUT
     RUN:BOOL; (*команда пуска*)
     REP:BOOL; (*перевод в состояние "ремонт"*)
END_VAR
VAR_OUTPUT
     STATE:BYTE; (*0 - ремонт, 1 - готов, 2 - работает*)
     START_DT, STOP_DT: DT; (*дата-время пуска и останова*)
     GENERAL_WORK_TIME:TIME:=t#0s; (*общая наработка*)
     CURRENT_WORK_TIME:TIME:=t#0s; (*текущая наработка*)
     CURRENT_REST_TIME:TIME:=t#0s; (*текущий простой*)
END_VAR
VAR
     CURRENT_DT:DT; (*текущие дата-время*)
     OLD_DT:DT;(*"старые" дата-время, зафиксированные при прошлом
                 запуске *)
END_VAR
-----------------------------------------------------------------
CURRENT_DT:=SysRtcGetTime(TRUE); (*определение текущих
                                   даты-времени*)
CASE STATE OF
0: (*ремонт*)
     IF  NOT REP THEN
          IF NOT RUN  THEN
               STATE:=1;
          ELSE
               STATE:=2;
               START_DT:= OLD_DT:= CURRENT_DT;
          END_IF
     END_IF
1: (*готов*)
     CURRENT_REST_TIME:=CURRENT_DT - STOP_DT;
     IF REP THEN
          STATE:=0;
     ELSIF RUN THEN
          STATE:=2;
          START_DT:= OLD_DT:= CURRENT_DT;
     END_IF
2: (*работа*)
     CURRENT_WORK_TIME:=CURRENT_DT - START_DT;
     GENERAL_WORK_TIME:=GENERAL_WORK_TIME + (CURRENT_DT -
                                                    OLD_DT);
```

```
     IF REP THEN
          STATE:=0;
          STOP_DT:= CURRENT_DT;
     ELSIF NOT RUN THEN
          STATE:=1;
          STOP_DT:= CURRENT_DT;
     END_IF
     OLD_DT:=CURRENT_DT;
END_CASE
```

Сами насосы (экземпляры функционального блока `PUMP`) в количестве 5 шт. помещены в массив `PUMPS`, объявленный в списке глобальных переменных:

```
VAR_GLOBAL
     level:REAL:=0; (*уровень в резервуаре*)
     PUMPS:ARRAY[1..5] OF PUMP; (*массив насосов*)
     (*Максимальное время работы насоса в нормальном режиме –
     рекомендуемое*)
     Normal_work_time:TIME:=t#20s;
     (*Минимальное время "отдыха" насоса в нормальном режиме –
     рекомендуемое*)
     Normal_rest_time:TIME:=t#10s;
     (*Авария*)
     alarm: BOOL;
END_VAR
```

Помимо массива в списке имеется переменная для хранения значения уровня в резервуаре `level`, флаг аварии `alarm` и две «константы»: `Normal_work_time` и `Normal_rest_time`.

Первая константа задает время работы насоса, по истечении которого его желательно выключить и запустить другой, готовый к работе, насос. Эта рекомендация может быть связана с чисто техническими обстоятельствами, однако в нашей программе все подчинено необходимости обеспечить равномерный износ агрегатов. Константа фактически будет задавать максимальное время работы насоса при условии, что его можно заменить каким-то другим. С константой `Normal_rest_time` дело обстоит аналогичным образом: по возможности, программа не будет запускать готовый к работе насос, если с момента его остановки прошло время, меньшее, чем значение этой константы. Значения констант `Normal_work_time` и `Normal_rest_time` выбраны малыми: 20 и 10 сек соответственно. Это сделано специально, чтобы ускорить процесс апробации программы (не будем же гонять нашу программу часами и сутками!).

Для наблюдения за процессом создана визуализация, окно которой в работе показано на рис. 3.162.

Визуализация предоставляет детальную информацию о работе насосов, позволяет переводить их состояние «Ремонт», изменять и наблюдать значение уровня в резервуаре. Кроме того, выводятся требуемое число и число реально работающих насосов, а также оповещение о тревоге, формируемое системой, в случае, когда она не в состоянии справиться с ситуацией.

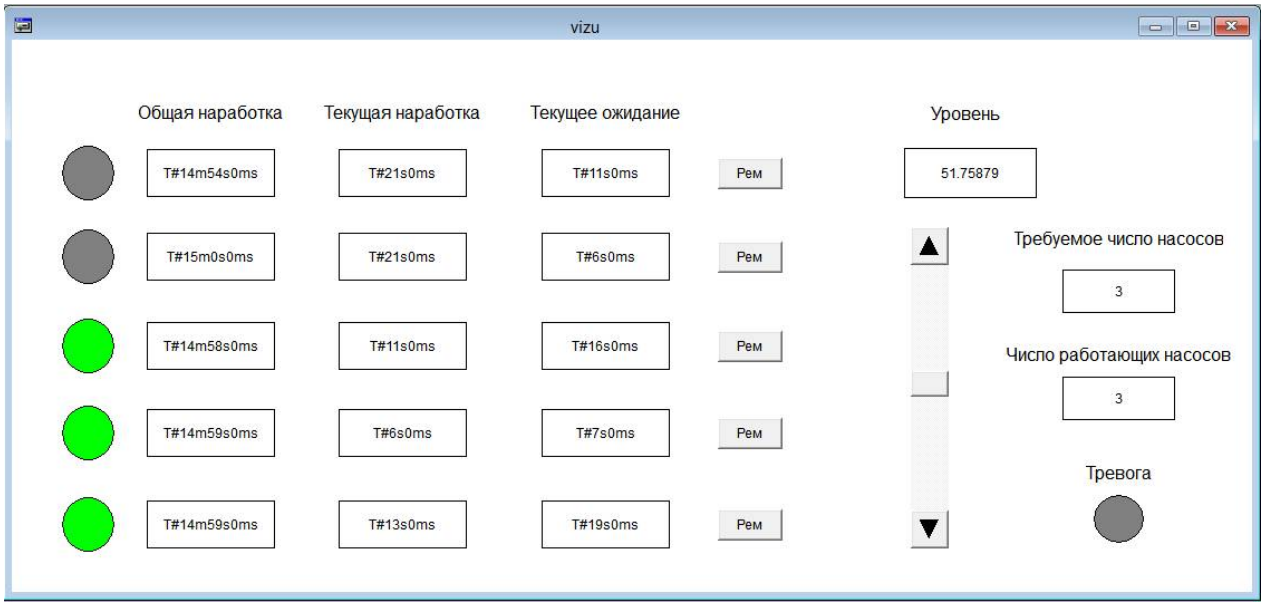


Рис. 3.162. Отладочное окно визуализации.

Привязки элементов визуализации:

ряда индикаторов включения насосов: изменение цвета, `PUMPS[i].STATE=2`, где `i` – номер насоса 1…5;

ряда «Общая наработка»: вывод текста, `PUMPS[i].GENERAL_WORK_ TIME`;

ряда «Текущая наработка»: вывод текста, `PUMPS[i].CURRENT_WORK_ TIME`;

ряда «Текущее ожидание»: вывод текста, `PUMPS[i].CURRENT_REST_ TIME`;

ряда кнопок «Рем»: переменная переключения `PUMPS[i].REP`;

«Уровень»: вывод текста, `level`;

полоса прокрутки: минимум – 0, ползунок – `level`, максимум – 100;

«Требуемое число насосов»: вывод текста, `PLC_PRG.Need_number`;

«Число работающих насосов»: вывод текста, `PLC_PRG.Real_number`;

«Тревога»: изменение цвета, `alarm`.

Код программы `PLC_PRG` приведен ниже. Программа реализует алгоритм, в общих чертах описанный в предыдущем пункте.

```
PROGRAM PLC_PRG
VAR
     (*Число необходимых, число реально работающих, число готовых
       к работе насосов*)
     Need_number, Real_number, Ready_number:BYTE;
     (*Номер насоса с максимальной общей наработкой*)
     PUMP_max:BYTE;
     (*Номер насоса с максимальной текущей наработкой*)
     PUMP_curr_max:BYTE;
     (*Номер насоса минимальной общей наработкой или
       с максимальным простоем в текущем состоянии готовности
       в зависимости от режима поиска*)
     PUMP_min:BYTE;
     (*Максимальная общая и текущая наработки*)
     Max_Time, Max_Current_Time:TIME;
```

```
    (* Минимальная общая наработка или максимальное время простоя
       в текущем состоянии готовности
    - в зависимости от режима поиска*)
    Min_Time:TIME;
    (*Счетчик цикла*)
    i:BYTE;
    (*Подходящий насос еще не нашли/ найти не удалось*)
    Not_find_free: BOOL;
    (*"Форсировать" поиск насоса - искать насос с максимальным
       простоем в текущем состоянии готовности*)
    FORCE: BOOL:=FALSE;
END_VAR
-----------------------------------------------------------------

Need_number:= REAL_TO_BYTE(level/16); (*определение требуемого
                                        числа насосов*)
Real_number:=Ready_number:=0; (*очистка счетчиков*)
Max_Time:=Max_Current_Time:=t#0s; (*обнуление максимальных
                                    времен*)
Not_find_free:=TRUE; (*насос еще не нашли*)


(*Опрос насосов*)
FOR i:=1 TO 5 BY 1 DO
  PUMPS[i]; (*вызов экземпляра ФБ из массива*)
  IF PUMPS[i].STATE=2 THEN (*если насос работает*)
     Real_number:=Real_Number+1; (*увеличиваем счетчик работающих
                                   насосов*)
     IF PUMPS[i].GENERAL_WORK_TIME > Max_Time THEN (*нашли
                                          кандидата на PUMP_max*)
        Max_Time:=PUMPS[i].GENERAL_WORK_TIME; (*сохраняем
                                    максимальную общую наработку*)
        PUMP_max:=i; (*сохраняем номер насоса*)
     END_IF
     IF PUMPS[i].CURRENT_WORK_TIME > Max_Current_Time THEN (*нашли
                                      кандидата на PUMP_curr_max*)
        Max_Current_Time:=PUMPS[i].CURRENT_WORK_TIME; (*сохраняем
                                   максимальную текущую наработку*)
        PUMP_curr_max:=i; (*сохраняем номер насоса*)
     END_IF
  ELSIF  PUMPS[i].STATE=1 THEN (*насос в состоянии готовности
                                  к пуску*)
     Ready_number:=Ready_number+1;  (*увеличиваем счетчик готовых
                                     насосов*)
     IF NOT FORCE THEN (*"щадащий" поиск среди готовых и
                         отдохнувших насосов*)
        (*Первый найденный неработающий или другой кандидат на
          PUMP_min*)
        IF ((Not_find_free) OR (PUMPS[i].GENERAL_WORK_TIME <
             Min_Time)) AND (PUMPS[i].CURRENT_REST_TIME >
             Normal_rest_time) THEN
           Not_find_free:=FALSE; (*нашли подходящий насос*)
```

```
          Min_Time:=PUMPS[i].GENERAL_WORK_TIME; (*сохраняем
                                        минимальную общую наработку*)
           PUMP_min:=i; (*сохраняем номер насоса*)
        END_IF
     ELSE (*"жесткий" поиск среди наиболее отдохнувших*)
        (*Первый найденный неработающий или другой кандидат на
          PUMP_min*)
        IF (Not_find_free) OR (PUMPS[i].CURRENT_REST_TIME >
            Min_Time) THEN
           Not_find_free:=FALSE; (*нашли подходящий насос*)
           Min_Time:=PUMPS[i].CURRENT_REST_TIME; (*сохраняем
                                        максимальное время отдыха*)
           PUMP_min:=i; (*сохраняем номер насоса*)
        END_IF
     END_IF
  ELSE (*насос в ремонте*)
     PUMPS[i].RUN:=FALSE; (*снять команду пуска, если она была*)
  END_IF
END_FOR

IF Real_number < Need_number THEN (*нужно включить очередной
                                        насос*)
  IF Ready_Number = 0 THEN (*Полный провал - готовых насосов
                                больше нет!*)
     alarm:=TRUE; (*Спасайся, кто может!*)
  ELSIF Not_find_free THEN (*полностью отдохнувших насосов не
                                нашли*)
     FORCE:=TRUE; (*ужесточить условия поиска*)
  ELSE
     PUMPS[Pump_min](RUN:=TRUE); (*включить найденного*)
     FORCE:=FALSE; (*сбросить форсированный поиск*)
     alarm:=FALSE; (*все в порядке*)
  END_IF
ELSIF Real_number > Need_number THEN (*нужно выключить очередной
                                        насос*)
  PUMPS[Pump_max](RUN:=FALSE); (*выключить самого усталого*)
  alarm:=FALSE; (*все в порядке*)
ELSIF Max_Current_Time > Normal_work_time AND Ready_number > 0
THEN (*нужно заменить проработавшего
     Normal_work_time*)
  PUMPS[Pump_curr_max](RUN:=FALSE); (*выключить проработавшего
                                        более Normal_work_time*)
END_IF
```

Сначала программа определяет требуемое количество работающих насосов для данного уровня в резервуаре. Делается это путем округления до ближайшего целого результата деления значения уровня на 16. В результате имеем следующее:

$$\texttt{level} < 8 - \texttt{Need\_number} = 0;$$
$$8 \le \texttt{level} < 24 - \texttt{Need\_number} = 1;$$
$$24 \le \texttt{level} < 40 - \texttt{Need\_number} = 2;$$
$$40 \le \texttt{level} < 56 - \texttt{Need\_number} = 3;$$

$56 \le$ level < 72 – Need_number = 4;

$72 \le$ level – Need_number = 5;

После инициализации переменных начинается опрос всех насосов.

В результате опроса определяются:

1) число работающих насосов (Real_number) и насосов, готовых к работе (Ready_Number);

2) номера насосов с максимальной общей и максимальной текущей наработками. Оба они будут рассматриваться далее как кандидаты на отключение;

3) найден или не найден в случае «щадящего» поиска насос, который нужно и можно запустить «по всем правилам»: он имеет минимальную среди всех готовых общую наработку и прошло достаточно времени с момента его остановки. Если такой насос найден, становится известным его номер;

4) в случае «жесткого» поиска – номер готового насоса с максимальным значением времени текущего простоя.

На основе собранной информации на втором этапе программа принимает решение о запуске или останове определенного насоса.

Если число работающих насосов меньше требуемого, нужно включить еще один насос. Если при этом готовых насосов нет, возможности управления исчерпаны и единственное, что может сделать программа – это включить сигнализацию. Если не все так плохо, проверяется значение переменной Not_find_free. Равенство ее TRUE означает, что обычный, «щадящий» поиск не увенчался успехом. Требуется повторить процедуру с установленным флагом FORCE, что означает поиск готового насоса с максимальным значением времени текущего простоя. Если Not_find_free = FALSE, насос найден, и он запускается.

Если число работающих насосов больше требуемого, нужно отключить один насос. Естественно, отключается насос с максимальной общей наработкой, номер которого был определен ранее.

При равенстве числа работающих насосов требуемому проверяется значение максимальной текущей наработки среди всех работающих насосов. Если оно больше Normal_work_time и при этом в системе есть хотя бы один готовый к запуску насос, насос с максимальной текущей наработкой останавливается (это в ближайшем будущем приведет к запуску другого насоса, т.е. произойдет замещение).

Многочисленные прогоны с различными манипуляциями над уровнем и переводом части насосов в ремонт показали работоспособность программы и алгоритма, на котором она основана. Об этом, в частности, свидетельствует «состояние дел», отраженное на визуализации (см. рис. 3.162). По истечении 15 минут работы общая наработка у каждого из пяти насосов практически одинакова. И это притом, что часть насосов первое время находилась в ремонте и вследствие этого «отстала» от основной группы на несколько минут. Как только эти насосы вступили в работу, система стала отдавать им «больший приоритет», чем другим, что и позволило «отставшим» «наверстать упущенное».

3.2.2.3. Разработка имитационной модели объекта

Под объектом управления мы будем понимать собственно резервуар с водой (вместе с установленными в нем датчиками уровня) и насосы. Верхний уровень Simulink-диаграммы представлен на рис. 3.163.

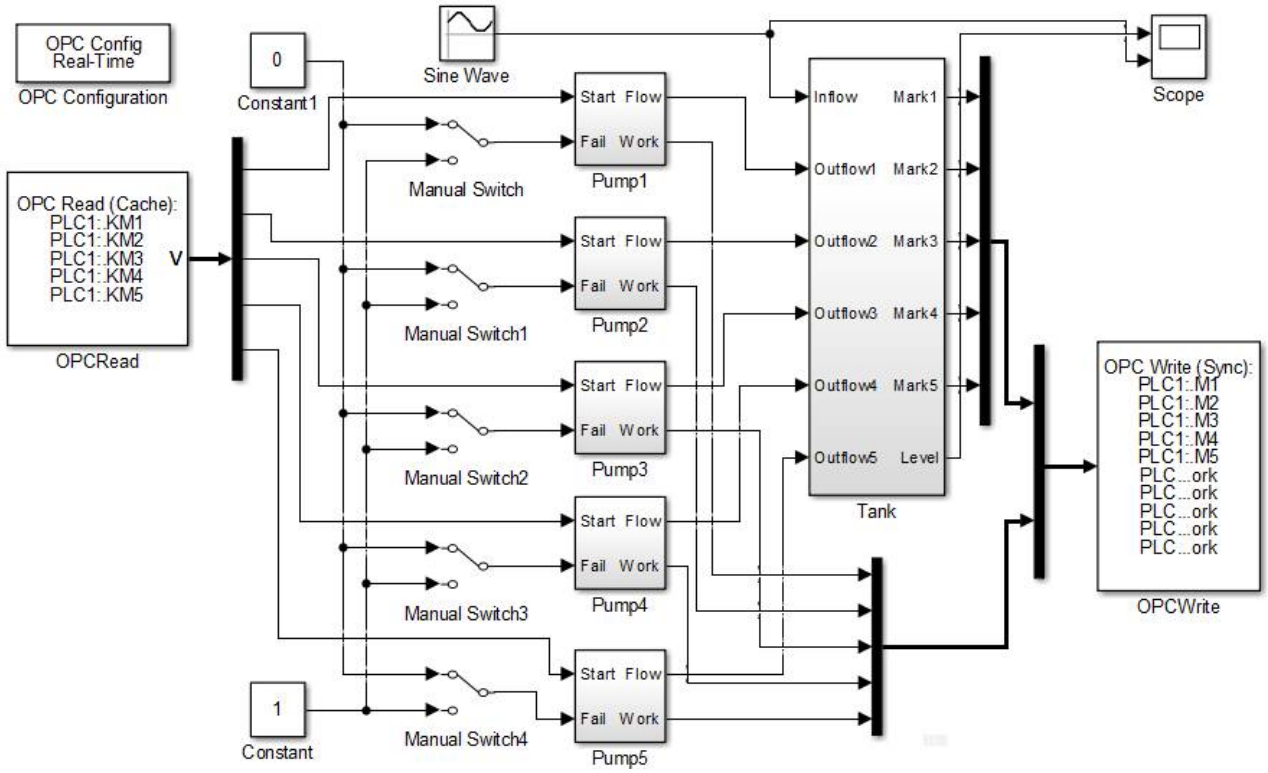


Рис. 3.163. Simulink-модель объекта.

Резервуар представлен подсистемой Tank, на входы которой подаются «расход на притоке» (0…1), формируемый блоком Sine Wave (смещенная синусоида), и пять «расходов на откачке» (0…0,2), формируемых подсистемами Pump1…Pump5, имитирующими насосы. Подсистема Tank выдает дискретные сигналы датчиков уровня Mark1…Mark5 и непрерывный сигнал Level, который хотя и не используется в системе управления, но может наблюдаться в блоке Scope (вместе с расходом на притоке).

Содержание подсистемы Tank показано на рис. 3.164. Основу подсистемы представляет обычный интегратор с ограничением выходного сигнала 0…100(%). Сигналы датчиков уровня вырабатывают релейные элементы, настроенные на срабатывания на соответствующих отметках с гистерезисом 0,5(%).

Коэффициент на входе интегратора, равный двум, устанавливает скорость наполнения резервуара: при максимальном расходе на притоке (InFlow = 1) и отсутствии откачки наполнение резервуара от нулевого уровня до максимального (100%) происходит за 50 с.

На рис. 3.165 показана модель насоса. При включении (подаче на вход Start единицы) расход достигает установившегося значения (0,2) за время, примерно равное 1 с. Релейный блок формирует дискретный сигнал при достижении расходом значения 0,1 («насос в работе»).

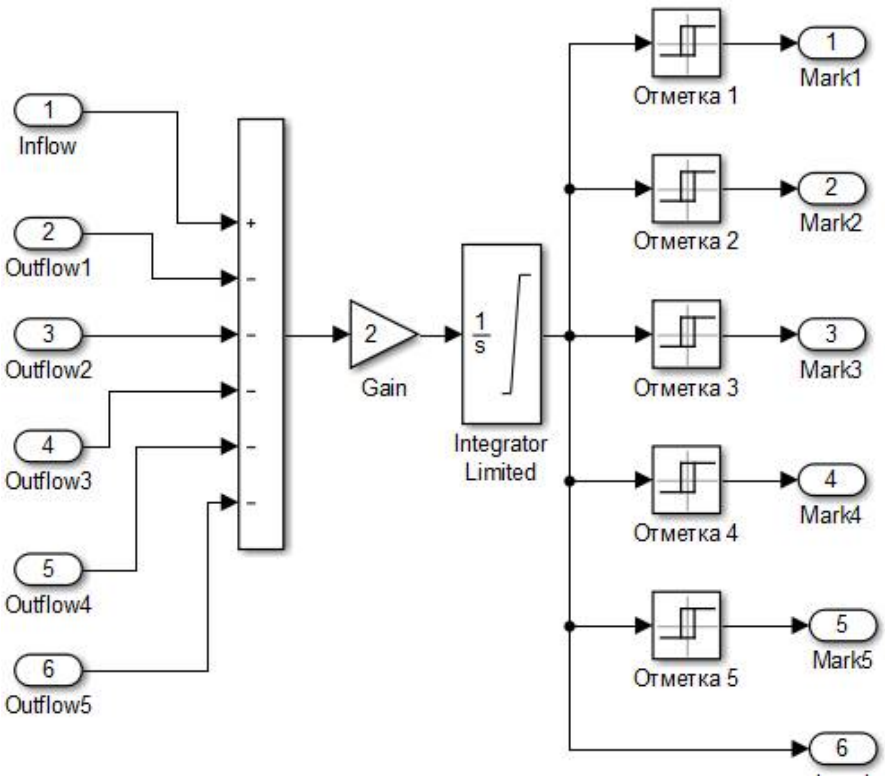


Рис. 3.164. Подсистема Tank.

При подаче на вход Fail логической единицы насос «останавливается» (если до этого он работал). Это происходит благодаря переключению выхода блока Switch с третьего («рабочего») входа на первый вход, на который подается нуль.

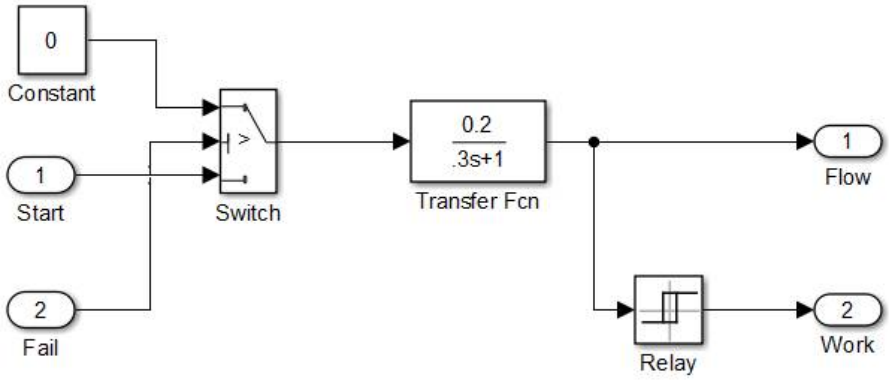


Рис. 3.165. Подсистемы Pump1…Pump5.

Сигналы Fail на подсистемы Pump1…Pump5 подаются «вручную» с помощью блоков Manual Switch (см. рис. 3.163) с целью имитации отказа насосов непосредственно в ходе работы системы. Сигналы Start приходят от блока OPCRead (из программы управления). Блок читает их из глобальных переменных виртуального контроллера `KM1…KM5` (срабатывание магнитных пускателей).

Выходные сигналы имитационной модели объекта подаются на блок OPCWrite и в дальнейшем записываются в глобальные переменные виртуального контроллера `M1…M5` и `P1_work…P5_work`.

3.2.2.4. Разработка программы для ПЛК

В отличие от предыдущей разработки здесь мы не будем отталкиваться от электрической схемы соединений, – она достаточна простая, – а сразу же приступим к разработке программного обеспечения контроллера. При необходимости будут даны краткие разъяснения по поводу того, как формируются значения тех или иных переменных. Разработку, как и ранее, начнем «с конца». На рис. 3.166 показан в работе экран визуализации, имитирующий щит управления насосами.

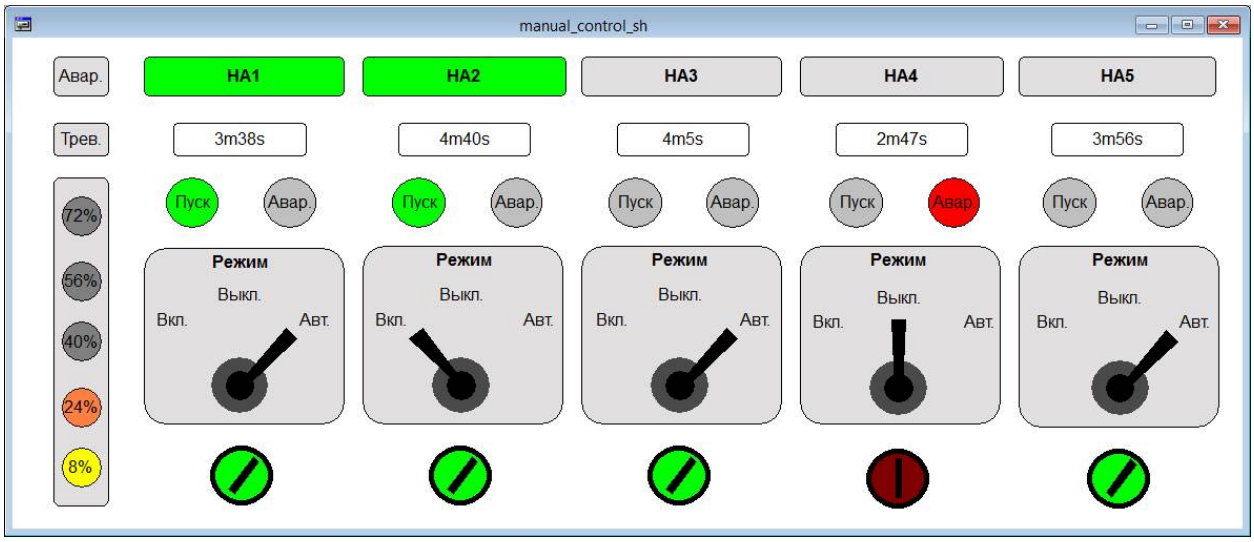


Рис. 3.166. Экран визуализации в работе.

На щите расположены:

аварийный и тревожный сигнализаторы, а также сигнализаторы достижения уровнем пяти отметок. Тревожный сигнализатор срабатывает, как и ранее, при недостаточном количестве готовых к работе насосов, а аварийный – если, помимо этого, уровень превысил максимальную отметку;

сигнальные лампы «Насос в работе» (прямоугольные), загорающиеся по сигналам датчиков работы насосов;

цифровые табло, выводящие значения общего времени наработки каждого насоса. Информация поступает от ПЛК по цифровому каналу. (На практике для упрощения эти элементы могут быть исключены, но для нашей имитационной системы весьма желательны);

сигнализаторы пуска и аварии насосов. Сигнал аварии может быть сформирован ПЛК при невозможности пуска или вручную путем снятия разрешения на работу;

переключатели режимов управления насосами на три позиции: включить/выключить/автоматический режим;

кнопки-переключатели, защищенные ключами, разрешающие запуск насосов (как в автоматическом, так и в ручном режимах).

Список глобальных переменных виртуального ПЛК приведен ниже.

В секции датчики/входы контроллера присутствуют переменные `KM1`…`KM5`. Это дискретные сигналы о срабатывании магнитных пускателей насосов.

Практически в формировании входных цепей задействуются дополнительные контакты пускателей.

Сигналы SA1_auto…SA5_auto и SA1_man…SA5_man предназначены для информирования ПЛК о положении трехпозиционных переключателей, управляющих режимами работы насосов (ручной пуск, останов, автоматическое управление).

Через входы, связанные с переменными P1_en…P5_en, в контроллер поступают сигналы разрешения на запуск насосов. Эти сигналы формируются посредством кнопок с ключами.

Выходы KM1_start…KM5_start управляют магнитными пускателями насосов в автоматическом режиме, а P1_fail…P5_fail – отключают пускатели при возникновении неисправности. Они используются для программного отключения отказавших насосов, запущенных вручную. Отметим, что помимо контроля исправности в этом режиме ПЛК продолжает заниматься фиксацией моментов пуска и останова и сохранении времени наработки.

Переменные alarm и crash управляют соответствующими сигнальными лампами (на практике неплохо было бы использовать и звуковую сигнализацию, по крайней мере, в дубле с аварийной лампой).

Переменные time1…time5 типа STRING предназначены для строкового представления выводимых на табло времен наработки каждого насоса.

«Электрическая схема» представлена вспомогательными реле K1…K5. К ней также относятся и пускатели KM1…KM5, но эти переменные были объявлены ранее.

В списке глобальных переменных присутствуют также массив виртуальных насосов PUMP и массив их общих наработок GWT. Отдельное хранение последнего связано с необходимостью сохранения его в файл. К условным константам относятся уже известные нам Normal_work_time и Normal_rest_time, а также строка с путем к папке архивного файла.

```
VAR_GLOBAL

(*Датчики/входы контроллера*)
P1_work,P2_work, P3_work, P4_work,P5_work:BOOL; (*датчики давле-
                                                  ния/ работы насосов*)
M1,M2,M3,M4,M5:BOOL; (*датчики уровня*)
KM1, KM2, KM3, KM4, KM5:BOOL; (*сигналы срабатывания пускателей*)
SA1_auto, SA2_auto, SA3_auto, SA4_auto, SA5_auto: BOOL; (*пере-
                                ключатели режимов в положении Авт.*)
SA1_man, SA2_man, SA3_man, SA4_man, SA5_man : BOOL; (*пере-
                                ключатели режимов в положении Вкл.*)
P1_en, P2_en, P3_en, P4_en, P5_en: BOOL; (*разрешения на пуск -
                                              переключатели*)

(*Выходы контроллера*)
KM1_start, KM2_start, KM3_start, KM4_start, KM5_start: BOOL;
                                        (*- управление пускателями*)
(*Сигналы аварии насосов, зафиксированные программно:*)
P1_fail, P2_fail, P3_fail, P4_fail, P5_fail: BOOL;
```

```
alarm: BOOL; (*тревога - недостаточное количество готовых к работе
               насосов*)
crash: BOOL; (*...    + уровень выше верхней отметки*)
time1,time2,time3,time4,time5: STRING; (*сигналы управления цифро-
                                          выми индикаторами*)

(*Элементы электрической схемы*)
K1,  K2,  K3,  K4,  K5: BOOL; (*реле разрешения запуска насосов*)
(*KM1, KM2, KM3, KM4, KM5:BOOL; - пускатели*)

(*Другие переменные*)
PUMPS:ARRAY[1..5] OF PUMP; (*массив насосов*)
GWT:ARRAY[1..5] OF TIME; (*массив времен наработок насосов*)

(*"Константы": *)
(*Максимальное время работы насоса в нормальном режиме -
                                                  рекомендуемое:*)
Normal_work_time:TIME:=t#20s;
(*Минимальное время "отдыха" насоса в нормальном режиме -
                                                  рекомендуемое:*)
Normal_rest_time:TIME:=t#10s;
DIR_PATH:STRING:='C:\Users\1\Desktop\Archive\'; (*путь к папке
                                                   архива*)
END_VAR
```

Привязки переменных к элементам экрана визуализации показаны на рис. 3.167. Поскольку органы управления и сигнализации для всех пяти насосов идентичны, привязки показаны только для первого насоса.

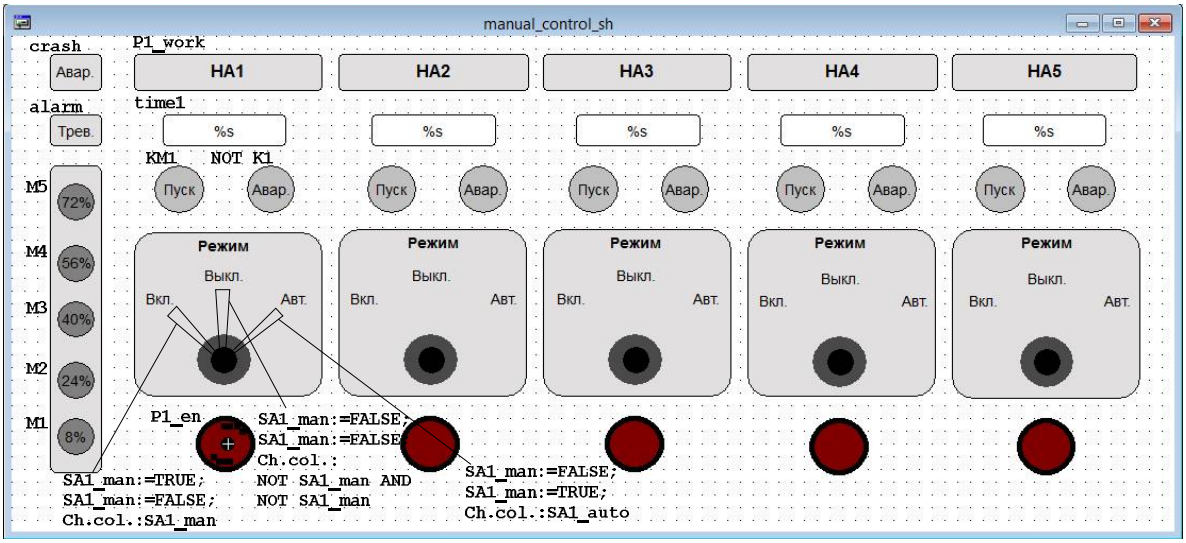


Рис.3.167. Привязки переменных к элементам экрана визуализации.

На рис. 3.168, 3.169 показаны фрагменты программы `Electrical_circuit`, моделирующей релейную схему управления.

«Реле» `K1…K5` при срабатывании дают разрешение на запуск насосов. Снятие разрешения производится кнопками-переключателями `P1_en…P5_en` вручную или путем активации выходов контроллера `P1_fail…P5_fail`. Отметим, что в схеме используются нормально закрытые контакты. Это сделано для того, чтобы в случае отказа контроллера обеспечить возможность ручного

управления. Большинство реальных ПЛК имеют релейные выходы с нормально открытыми контактами. Поэтому могут потребоваться дополнительные реле для «инверсии» сигналов. Другое возможное решение – установка «безопасного состояния» выходных реле в положение «замкнуто».

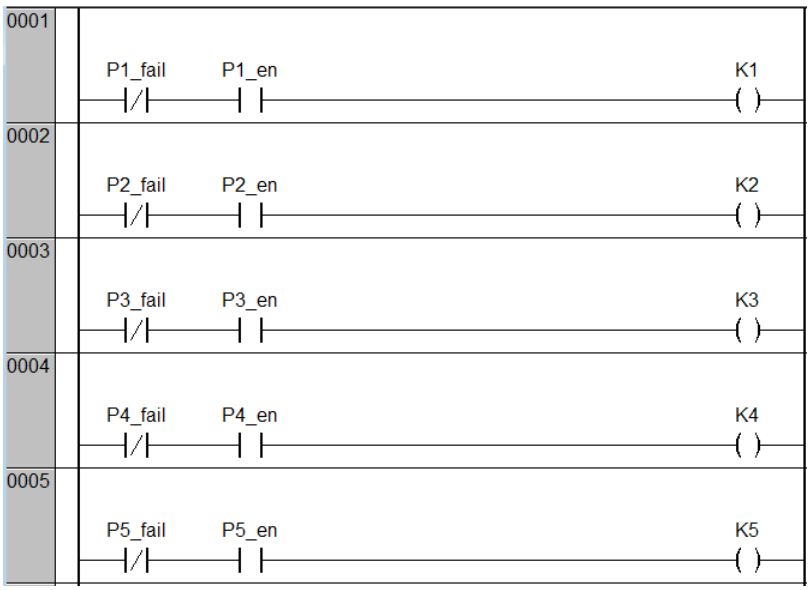


Рис. 3.168. Программа `Electrical_circuit` (фрагмент 1).

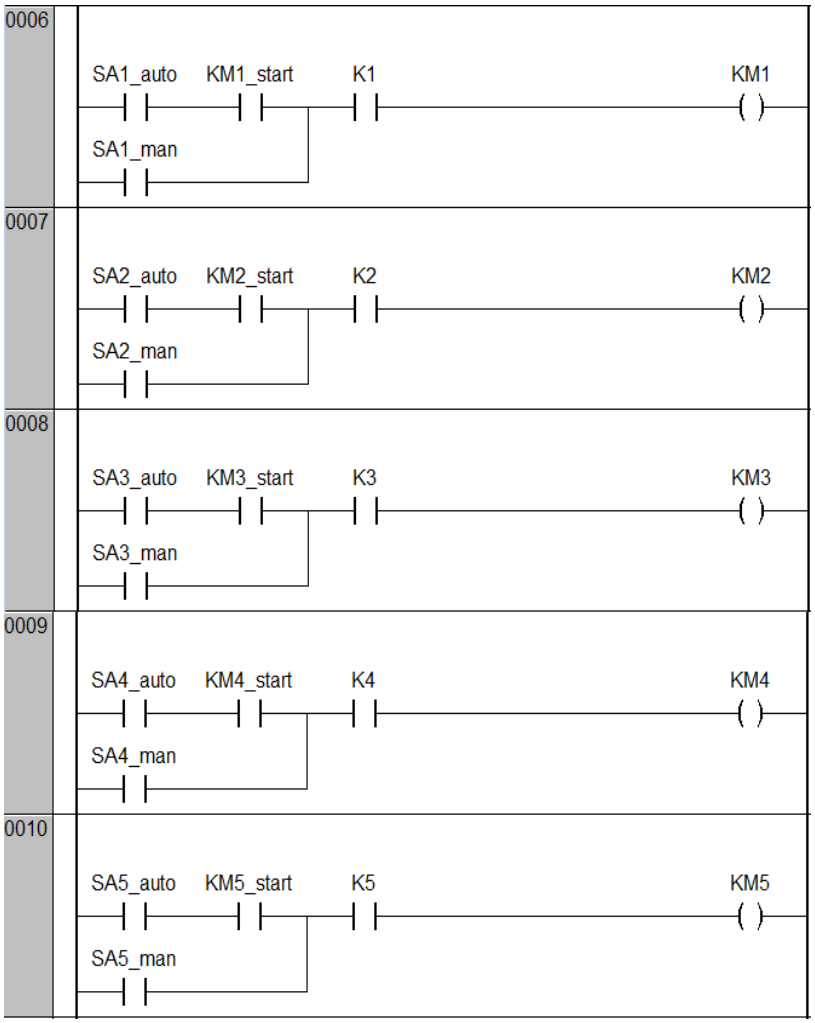


Рис. 3.169. Программа `Electrical_circuit` (фрагмент 2).

В п. 3.2.2.2. разработан функциональный блок PUMP (насос), фиксирующий состояние насоса и ведущий подсчет временных величин. Внесем в него небольшие изменения, позволяющие устанавливать начальное значение общего времени наработки насоса. В список входных переменных добавим переменные-команды перехода в автоматический режим и на загрузку начального значения, а также само начальное значение:

```
VAR_INPUT
     RUN:BOOL; (*команда пуска*)
     REP:BOOL; (*перевод в состояние "ремонт"*)
     AUTO:BOOL; (*автоматическое управление*)
     INIT:BOOL; (*загрузка начального значения GWT*)
     GWT_VAL:TIME; (*начальное значение GWT*)
END_VAR
```

В начало (или в конец) секции кода добавим следующие операторы:

```
IF INIT THEN
     GENERAL_WORK_TIME:=GWT_VAL;
END_IF
```

Всю логику работы с виртуальными насосами разместим в отдельной программе PUMPS_SELECT, листинг которой приведен ниже.

На самом деле программа PUMPS_SELECT есть не что иное, как немного переделанная PLC_PRG из п. 3.2.2.2. Исправления выделены жирным.

Во-первых, требуемое число работающих насосов теперь будет определяться немного по-другому. При достижении уровнем в резервуаре i-ой отметки должны быть включены, как и ранее, i штук насосов. Однако выключиться все они должны только при падении уровня ниже отметки 1. Такой вариант управления ближе к практике. Действительно, если цель работы системы – поддерживать минимальный уровень в резервуаре, то насосы должны работать, пока эта цель не достигнута. Иначе при больших притоках мы рискуем получить колебания уровня на верхних отметках.

Во-вторых, все манипуляции по поиску кандидатов на включение и отключение должны касаться только насосов, находящихся в режиме автоматического управления. С этой целью и вводится дополнительная проверка входной переменной AUTO экземпляров функциональных блоков из массива PUMPS. Собственно говоря, только для этого эта переменная и была введена, – больше нигде она и не используется.

Последний вид изменений связан с управлением вновь введенной переменной crash.

Листинг «новой» программы PLC_PRG приведен ниже. Программа занимается решением следующих задач:

1) загрузка начальных значений общей наработки в виртуальные насосы из файла *pumps*;

2) формирование строковых переменных, значения которых (наработки насосов) выводятся на табло щита управления;

3) обеспечение взаимодействия «мира» реальных насосов с «миром» виртуальных насосов из массива PUMPS. Все изменения в виртуальном мире отра-

жаются в реальном и наоборот. Работа реальных насосов контролируется с помощью таймеров T1…T5. Если по истечении контрольного времени (в программе – 2с) после срабатывания пускателя не приходит подтверждающего сигнала с датчика, виртуальный насос переводится в состояние «неисправность», а реальный – отключается. Сброс состояния производится снятием и последующей подачей разрешения на запуск с помощью кнопки-переключателя с ключом.

Для распознавания этой операции программа сравнивает между собой текущие и «старые» значения переменных `P1_en…P5_en`;

4) запуск программ `PUMPS_SELECT` и `Electrical_circut`.

```
PROGRAM PUMPS_SELECT
VAR
     (*Число необходимых, число реально работающих, число готовых
       к работе насосов*)
     Need_number, Real_number, Ready_number:BYTE;
     (*Номер насоса с максимальной общей наработкой*)
     PUMP_max:BYTE;
     (*Номер насоса с максимальной текущей наработкой*)
     PUMP_curr_max:BYTE;
     (*Номер насоса минимальной общей наработкой или
     с максимальным простоем в текущем состоянии готовности в за-
     висимости от режима поиска*)
     PUMP_min:BYTE;
     (*Максимальная общая и текущая наработки*)
     Max_Time, Max_Current_Time:TIME;
     (* Минимальная общая наработка или максимальное время простоя
       в текущем состоянии готовности  - в зависимости от режима
       поиска*)
     Min_Time:TIME;
     (*Счетчик цикла*)
     i:BYTE;
     (*Подходящий насос еще не нашли/ найти не удалось*)
     Not_find_free: BOOL;
     (*"Форсировать" поиск насоса - искать насос с максимальным
        простоем в текущем состоянии готовности*)
     FORCE: BOOL:=FALSE;
END_VAR

-----------------------------------------------------------------------

(*Определение требуемого числа насосов*)
CASE Need_number OF
0:   IF M1 THEN Need_number:=1; END_IF
1:   IF M2 THEN Need_number:=2; END_IF
2:   IF M3 THEN Need_number:=3; END_IF
3:   IF M4 THEN Need_number:=4; END_IF
4:   IF M5 THEN Need_number:=5; END_IF
END_CASE
IF NOT M1 THEN Need_number:=0; END_IF
```

```
Real_number:=Ready_number:=0; (*очистка счетчиков*)
Max_Time:=Max_Current_Time:=t#0s; (*обнуление максимальных
                                    времен*)
Not_find_free:=TRUE; (*насос еще не нашли*)
(*Опрос насосов*)
FOR i:=1 TO 5 BY 1 DO
  PUMPS[i]; (*вызов экземпляра ФБ из массива*)
  IF PUMPS[i].STATE=2 THEN (*если насос работает*)
     Real_number:=Real_Number+1; (*увеличиваем счетчик работающих
                                   насосов*)
     IF PUMPS[i].AUTO AND (PUMPS[i].GENERAL_WORK_TIME > Max_Time)
     THEN
        (*нашли кандидата на PUMP_max*)
        Max_Time:=PUMPS[i].GENERAL_WORK_TIME; (*сохраняем макси-
                                              мальную общую наработку*)
        PUMP_max:=i; (*сохраняем номер насоса*)
     END_IF
     IF  PUMPS[i].AUTO AND (PUMPS[i].CURRENT_WORK_TIME >
         Max_Current_Time) THEN
        (*нашли кандидата на PUMP_curr_max*)
        Max_Current_Time:=PUMPS[i].CURRENT_WORK_TIME; (*сохраняем
                                    максимальную текущую наработку*)
        PUMP_curr_max:=i; (*сохраняем номер насоса*)
     END_IF
  ELSIF PUMPS[i].AUTO AND (PUMPS[i].STATE=1) THEN (*насос в
                                    состоянии готовности к пуску*)
     Ready_number:=Ready_number+1;  (*увеличиваем счетчик готовых
                                     насосов*)
     IF NOT FORCE THEN (*"щадащий" поиск среди готовых и отдохнув-
                         ших насосов*)
        (*Первый найденный неработающий или другой кандидат на
          PUMP_min*)
       IF ((Not_find_free) OR (PUMPS[i].GENERAL_WORK_TIME <
          Min_Time))
          AND (PUMPS[i].CURRENT_REST_TIME > Normal_rest_time) THEN
          Not_find_free:=FALSE; (*нашли подходящий насос*)
          Min_Time:=PUMPS[i].GENERAL_WORK_TIME; (*сохраняем
                                    минимальную общую наработку*)
          PUMP_min:=i; (*сохраняем номер насоса*)
       END_IF
     ELSE (*"жесткий" поиск среди наиболее отдохнувших*)
       (*Первый найденный неработающий или другой кандидат на
         PUMP_min*)
       IF (Not_find_free) OR (PUMPS[i].CURRENT_REST_TIME >
          Min_Time) THEN
          Not_find_free:=FALSE; (*нашли подходящий насос*)
          Min_Time:=PUMPS[i].CURRENT_REST_TIME; (*сохраняем
                                    максимальное время отдыха*)
          PUMP_min:=i; (*сохраняем номер насоса*)
       END_IF
     END_IF
  ELSE (*насос в ремонте*)
     PUMPS[i].RUN:=FALSE; (*снять команду пуска, если она была*)
```

```
  END_IF
END_FOR

IF Real_number < Need_number THEN (*нужно включить очередной
                                    насос*)
   IF Ready_Number = 0 THEN (*Полный провал - готовых насосов
                              больше нет!*)
     alarm:=TRUE; (*Тревога!*)
     crash:=M5; (*Спасайся, кто может!*)
   ELSIF Not_find_free THEN (*полностью отдохнувших насосов не
                              нашли*)
     FORCE:=TRUE; (*ужесточить условия поиска*)
     alarm:=FALSE;  (*все в порядке*)
     crash:=FALSE;
   ELSE
     PUMPS[Pump_min](RUN:=TRUE); (*включить найденного*)
     FORCE:=FALSE; (*сбросить форсированный поиск*)
     alarm:=FALSE; (*все в порядке*)
     crash:=FALSE;
   END_IF
ELSIF Real_number >Need_number THEN (*нужно выключить очередной
                                     насос*)
   PUMPS[Pump_max](RUN:=FALSE); (*выключить самого усталого*)
   alarm:=FALSE; (*все в порядке*)
   crash:=FALSE;
ELSIF Max_Current_Time > Normal_work_time AND Ready_number > 0
THEN
(*нужно заменить проработавшего Normal_work_time*)
   PUMPS[Pump_curr_max](RUN:=FALSE); (*выключить проработавшего
                                      Normal_work_time*)
   alarm:=FALSE; (*все в порядке*)
   crash:=FALSE;
END_IF
```

Последняя программа, `archiving`, листинг которой также приведен ниже, занимается записью в файл времен общих наработок насосов. Для этого сначала опрашиваются все виртуальные насосы, и формируется массив `GWT`. После этого массив сбрасывается в файл *pumps*.

Программа `archiving` вызывается в рамках циклически выполняемой задачи (в нашей реализации – через каждые 10 мин.).

`PLC_PRG` вызывается в рамках свободно выполняемой задачи.

Программа занимается:

загрузкой начальных значений общей наработки насосов из файла;

формированием строк с временами общей наработки для вывода на табло;

переводом виртуальных насосов в автоматический режим в зависимости от положения переключателей;

запуском виртуальных насосов при ручном включении;

выключение виртуальных насосов;

управлением насосами, находящимися в автоматическом режиме;

контролем запуска насосов и переводо виртуальных насосов в режим «Неисправность»;

возврат ранее «неисправных» виртуальных насосов в работу;

управлением выходами «авария насоса»;

вызовами программ PUMPS_SELECT, Electrical_circut.

Код программы приведен ниже.

```
PROGRAM archiving
VAR
     FileD:DWORD; (*дескриптор файла*)
     n: DWORD; (*количество считанных/записанных байт при работе с
                файлом*)
     i:BYTE; (*счетчик цикла*)
END_VAR
-------------------------------------------------------------------
FOR i:=1 TO 5 BY 1 DO
     GWT[i]:=PUMPS[i].GENERAL_WORK_TIME; (*формирование массива
                                          GWT*)
END_FOR

FileD:=SysFileOpen(CONCAT(DIR_PATH,'pumps'),'w'); (*попытка
                                        открыть файл для записи*)
n := SysFileWrite(FileD,ADR(GWT),SIZEOF(GWT)); (*попытка записи*)
SysFileClose(FileD); (*закрытие файла*)
-------------------------------------------------------------------
PROGRAM PLC_PRG
VAR
     T1, T2, T3, T4, T5: TON; (*таймеры контроля запуска насосов*)
     P1_en_old, P2_en_old, P3_en_old, P4_en_old, P5_en_old: BOOL;
     (* - старые значения Pi_en*)
     archive_loaded:BOOL:=FALSE; (*признак начальной загрузки
                                  архива*)
     FileD:DWORD; (*дескриптор файла*)
     n: DWORD:=0; (*количество считанных/записанных байт при
                   работе с файлом*)
     i: BYTE; (*счетчик цикла*)
END_VAR
-------------------------------------------------------------------
(*Загрузка начальных значений общей наработки*)
IF NOT archive_loaded THEN
     FileD:=SysFileOpen(CONCAT(DIR_PATH,'pumps'),'r'); (*попытка
                                       открыть файл для чтения*)
     n := SysFileRead(FileD,ADR(GWT),SIZEOF(GWT)); (*попытка
                                                    чтения*)
     SysFileClose(FileD); (*закрыть файл*)
     archive_loaded:=TRUE; (*архив загружен*)
     IF n<>0 THEN (*попытка чтения удалась*)
          FOR i:=1 TO 5 BY 1 DO
               PUMPS[i](INIT:=TRUE, GWT_VAL:=GWT[i]); (*запись
                                              начального значения*)
               PUMPS[i].INIT:=FALSE; (*сброс приказа*)
          END_FOR
     END_IF
```

```
END_IF
(*Формирование строк с временами общей наработки для вывода на
табло*)
time1:=TIME_TO_STRING(PUMPS[1].GENERAL_WORK_TIME);
time1:=RIGHT(time1,LEN(time1)-2);
time2:=TIME_TO_STRING(PUMPS[2].GENERAL_WORK_TIME);
time2:=RIGHT(time2,LEN(time2)-2);
time3:=TIME_TO_STRING(PUMPS[3].GENERAL_WORK_TIME);
time3:=RIGHT(time3,LEN(time3)-2);
time4:=TIME_TO_STRING(PUMPS[4].GENERAL_WORK_TIME);
time4:=RIGHT(time4,LEN(time4)-2);
time5:=TIME_TO_STRING(PUMPS[5].GENERAL_WORK_TIME);
time5:=RIGHT(time5,LEN(time5)-2);
(*Перевод виртуальных насосов в автоматический режим в зависимости
от положения переключателей*)
PUMPS[1].AUTO:=SA1_auto;
PUMPS[2].AUTO:=SA2_auto;
PUMPS[3].AUTO:=SA3_auto;
PUMPS[4].AUTO:=SA4_auto;
PUMPS[5].AUTO:=SA5_auto;
(*Запуск виртуальных насосов при ручном включении*)
IF SA1_man THEN PUMPS[1].RUN:=TRUE; END_IF
IF SA2_man THEN PUMPS[2].RUN:=TRUE; END_IF
IF SA3_man THEN PUMPS[3].RUN:=TRUE; END_IF
IF SA4_man THEN PUMPS[4].RUN:=TRUE; END_IF
IF SA5_man THEN PUMPS[5].RUN:=TRUE; END_IF
(*Выключение виртуальных насосов*)
IF NOT SA1_man  AND NOT SA1_auto THEN PUMPS[1].RUN:=FALSE; END_IF
IF NOT SA2_man  AND NOT SA2_auto THEN PUMPS[2].RUN:=FALSE; END_IF
IF NOT SA3_man  AND NOT SA3_auto THEN PUMPS[3].RUN:=FALSE; END_IF
IF NOT SA4_man  AND NOT SA4_auto THEN PUMPS[4].RUN:=FALSE; END_IF
IF NOT SA5_man  AND NOT SA5_auto THEN PUMPS[5].RUN:=FALSE; END_IF
(*Управление насосами, находящимися в автоматическом режиме*)
KM1_start:= PUMPS[1].STATE=2 AND  PUMPS[1].AUTO;
KM2_start:= PUMPS[2].STATE=2 AND  PUMPS[2].AUTO;
KM3_start:= PUMPS[3].STATE=2 AND  PUMPS[3].AUTO;
KM4_start:= PUMPS[4].STATE=2 AND  PUMPS[4].AUTO;
KM5_start:= PUMPS[5].STATE=2 AND  PUMPS[5].AUTO;
(*Запуск таймеров контроля пуска*)
T1(IN:=KM1, PT:=t#2s);
T2(IN:=KM2, PT:=t#2s);
T3(IN:=KM3, PT:=t#2s);
T4(IN:=KM4, PT:=t#2s);
T5(IN:=KM5, PT:=t#2s);
 (*Перевод виртуальных насосов в режим "Неисправность"*)
IF (T1.Q AND NOT P1_work) OR  NOT P1_en THEN  PUMPS[1].REP:=TRUE;
END_IF
IF (T2.Q AND NOT P2_work) OR  NOT P2_en THEN  PUMPS[2].REP:=TRUE;
END_IF
IF (T3.Q AND NOT P3_work) OR  NOT P3_en THEN  PUMPS[3].REP:=TRUE;
END_IF
IF (T4.Q AND NOT P4_work) OR  NOT P4_en THEN  PUMPS[4].REP:=TRUE;
END_IF
```

```
IF (T5.Q AND NOT P5_work) OR  NOT P5_en THEN  PUMPS[5].REP:=TRUE;
END_IF
(*Возврат "неисправных" виртуальных насосов в работу*)
IF  P1_en AND NOT P1_en_old THEN  PUMPS[1].REP:=FALSE; END_IF
IF  P2_en AND NOT P2_en_old THEN  PUMPS[2].REP:=FALSE; END_IF
IF  P3_en AND NOT P3_en_old THEN  PUMPS[3].REP:=FALSE; END_IF
IF  P4_en AND NOT P4_en_old THEN  PUMPS[4].REP:=FALSE; END_IF
IF  P5_en AND NOT P5_en_old THEN  PUMPS[5].REP:=FALSE; END_IF
P1_en_old:=P1_en;
P2_en_old:=P2_en;
P3_en_old:=P3_en;
P4_en_old:=P4_en;
P5_en_old:=P5_en;
(*Управление выходами "авария насоса"*)
P1_fail:=PUMPS[1].STATE=0;
P2_fail:=PUMPS[2].STATE=0;
P3_fail:=PUMPS[3].STATE=0;
P4_fail:=PUMPS[4].STATE=0;
P5_fail:=PUMPS[5].STATE=0;
(*Вызов программ*)
PUMPS_SELECT;
Electrical_circut;
```

3.2.3. Система управления лифтом

3.2.3.1. Физическая модель лифта

В отличие от ранее рассмотренных имитационных систем эта система имеет реальный прототип: действующую физическую модель лифта с настоящей системой управления. Результаты, полученные при разработке имитационной системы, были использованы в ходе создания программного обеспечения установки. Поэтому перед тем, как перейти к рассмотрению имитационной модели, кратко опишем физическую модель и систему управления.

Модель лифта показана на рис. 3.170.

На рисунке обозначены:

1 – шахта лифта; 2 – трос; 3 – кабина лифта; 4 – двери кабины; 5 – привод дверей; 6 – направляющие дверей; 7,8 – направляющие дверей; 9 – блок питания;

Рис. 3.170. Физическая модель лифта.

10 – шкив; 11 – редуктор; 12 –двигатель главного привода; 13 – аварийный выключатель; 14 – основание; 15 – пускатель аварийный; 16 – фотодиоды (этажные и межэтажные датчики); 17 – светодиод; 18 – шина проводов.

Размеры установки были определены исходя из возможностей лаборатории. Высота шахты равняется 2,9 м (при высоте потолка в 3,2 м). Шахта представляет собой две параллельные направляющие, выполненные из стального уголка. Шахта прикрепляется как к основанию, на котором размещены другие элементы установки, так и к потолку лаборатории.

Количество этажей принято равным четырем. Межэтажное расстояние составляет 0,65 м, высота кабины – 0,45 м. Кабина снабжена боковыми конструкциями, включающими подшипники качения, посредством которых она удерживается и передвигается по направляющим шахты.

Двигатель главного привода 4АА50В4УЗ имеет синхронную скорость вращения 1500 об/мин, номинальную мощность 0,29 кВт. Передаточное число червячно-цилиндрического редуктора равно 167. На выходной вал редуктора насажен шкив, наматывающий трос, соединенный с кабиной лифта через блоки, установленные на верхних перекрытиях шахты.

На максимальной скорости кабина проходит от первого этажа до четвертого за 15 с. Двигатель питается от преобразователя частоты ASC 300, с помощью которого осуществляется регулирование скорости подъема/опускания лифта.

Двери кабины имеют собственный привод на основе двигателя постоянного тока. Для определения состояния дверей предусмотрены три датчика (двери открыты, закрыты или находятся под внешним воздействием).

Положение кабины контролируется оптическими датчиками. Передатчик всех датчиков установлен на крыше кабины, а приемники расположены на шахте лифта. Этажные приемники отвечают за точную остановку кабины на этаже. Промежуточные – за снижение частоты вращения привода при подходе к требуемому этажу. Сигналы датчиков усиливаются транзисторными усилителями. Отдельная оптическая пара установлена на валу двигателя и формирует два импульса на один его оборот. Этот сигнал может использоваться вместо сигналов межэтажных датчиков.

В конструкции установки также предусмотрено автоматическое отключение привода в тех случаях, когда по каким-либо причинам кабина лифта опустится ниже первого этажа или поднимется выше четвертого. Аварийное отключение осуществляют конечные выключатели, расположенные в верхней и нижней частях одной из направляющих шахты.

На пульте управления расположены кнопки *приказов* и *вызовов*, а также светодиоды для фиксации фактов их принятия системой управления. Для второго и третьего этажа предусмотрены по две кнопки вызова: вниз и вверх.

3.2.3.2. Система управления

Структура системы управления показана на рис. 3.171.

Основной привод стенда управляется частотным преобразователем ACS 300, который обеспечивает плавный разгон и плавное торможение кабины.

Общее управление установкой осуществляет программируемый логический контроллер Siemens S7-226.

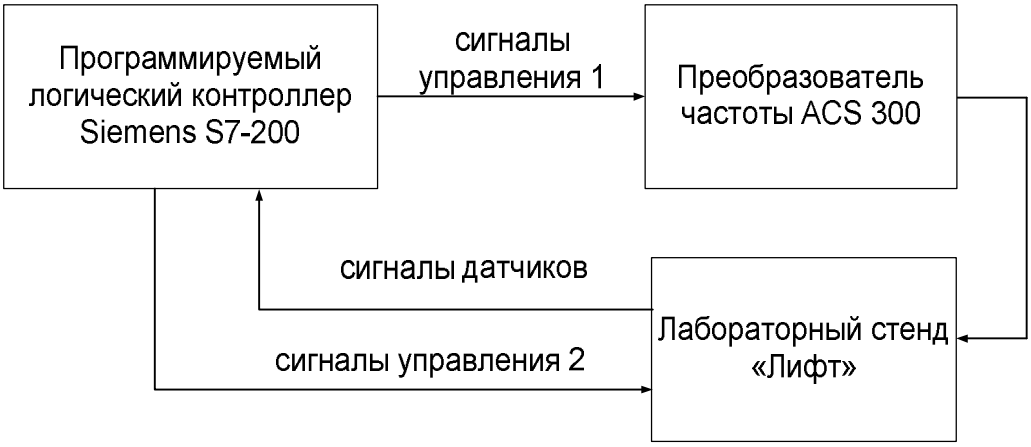


Рис. 3.171. Структура системы управления.

Контроллер получает сигналы следующих датчиков:

1) кнопки пульта;

2) концевые выключатели дверей и датчик перегрузки;

3) этажные и межэтажные датчики положения кабины;

4) датчик на валу двигателя.

Выполняя программу, Siemens S7-226 формирует сигналы управления:

сигналы управления 1 (преобразователем частоты):

1) пуск;

2) реверс;

3) переход на пониженную скорость;

сигналы управления 2 (непосредственно моделью лифта):

1) включение привода дверей кабины на их открытие;

2) включение привода дверей кабины на их закрытие;

3) сигналы управления светодиодной индикацией пульта.

Принципиальная схема соединений входных цепей контроллера показана на рис. 3.172.

На рис. 3.172 обозначены:

SB1–SB4 – кнопки приказов на пульте управления;

SB5–SB10 – кнопки вызовов кабины на пульте управления;

SQ1–SQ3 –концевые выключатели, использующиеся в качестве датчиков состояния дверей (дверь закрыта, находится под внешней нагрузкой, закрыта);

VD1 –VD10 – фотодиоды датчиков положения кабины в шахте: VD1, VD4, VD7, VD10 – этажные датчики, остальные –межэтажные;

VD13 – светодиод датчиков положения кабины в шахте;

VD11, VD12 – оптическая пара датчика оборотов двигателя главного привода.

Для усиления сигналов всех оптических датчиков используются одинаковые блоки усиления на биполярных транзисторах.

Принципиальная схема соединений выходных цепей контроллера показана на рис. 3.173.

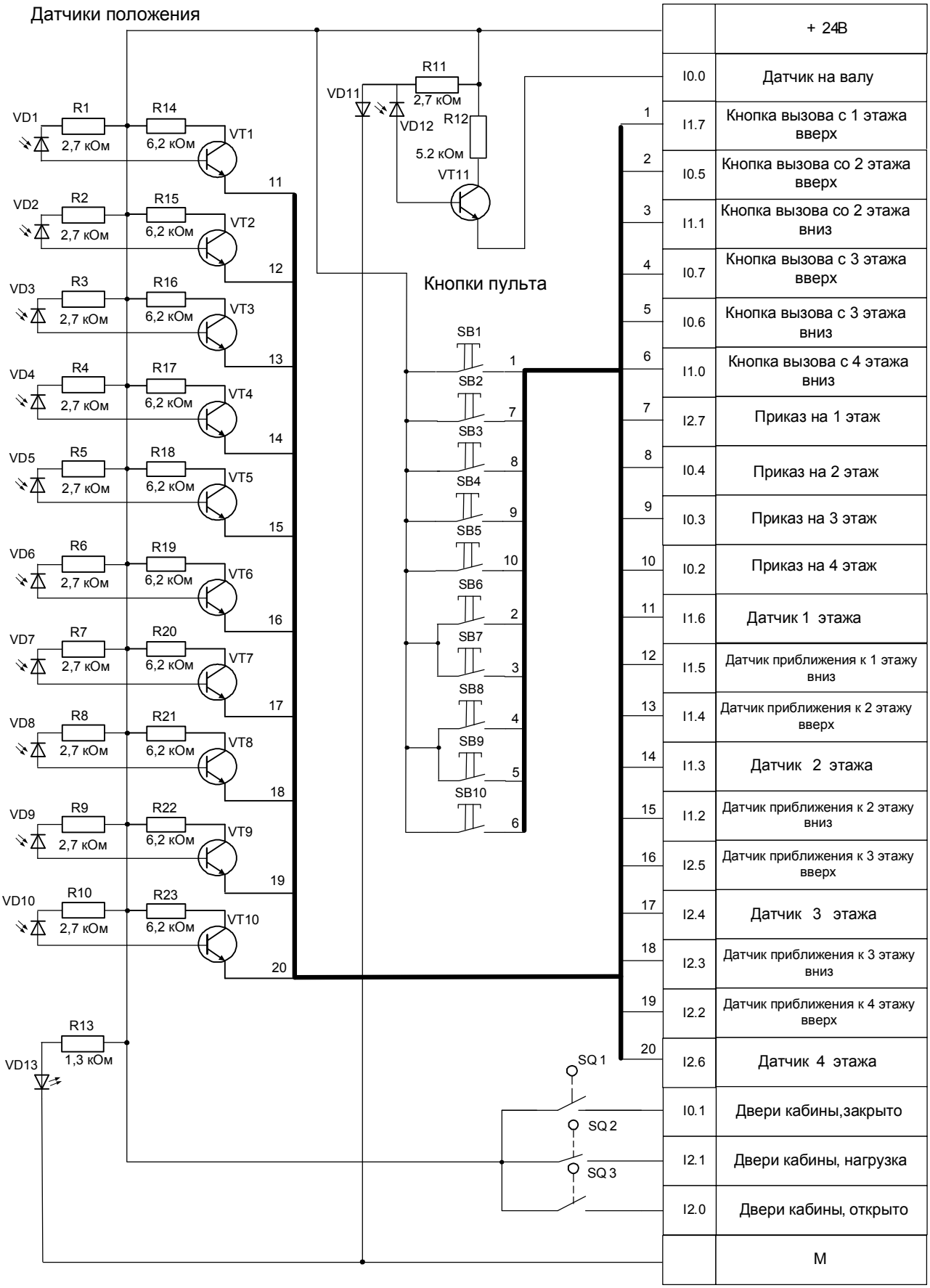


Рис. 3.172. Схема соединений. Входные цепи контроллера.

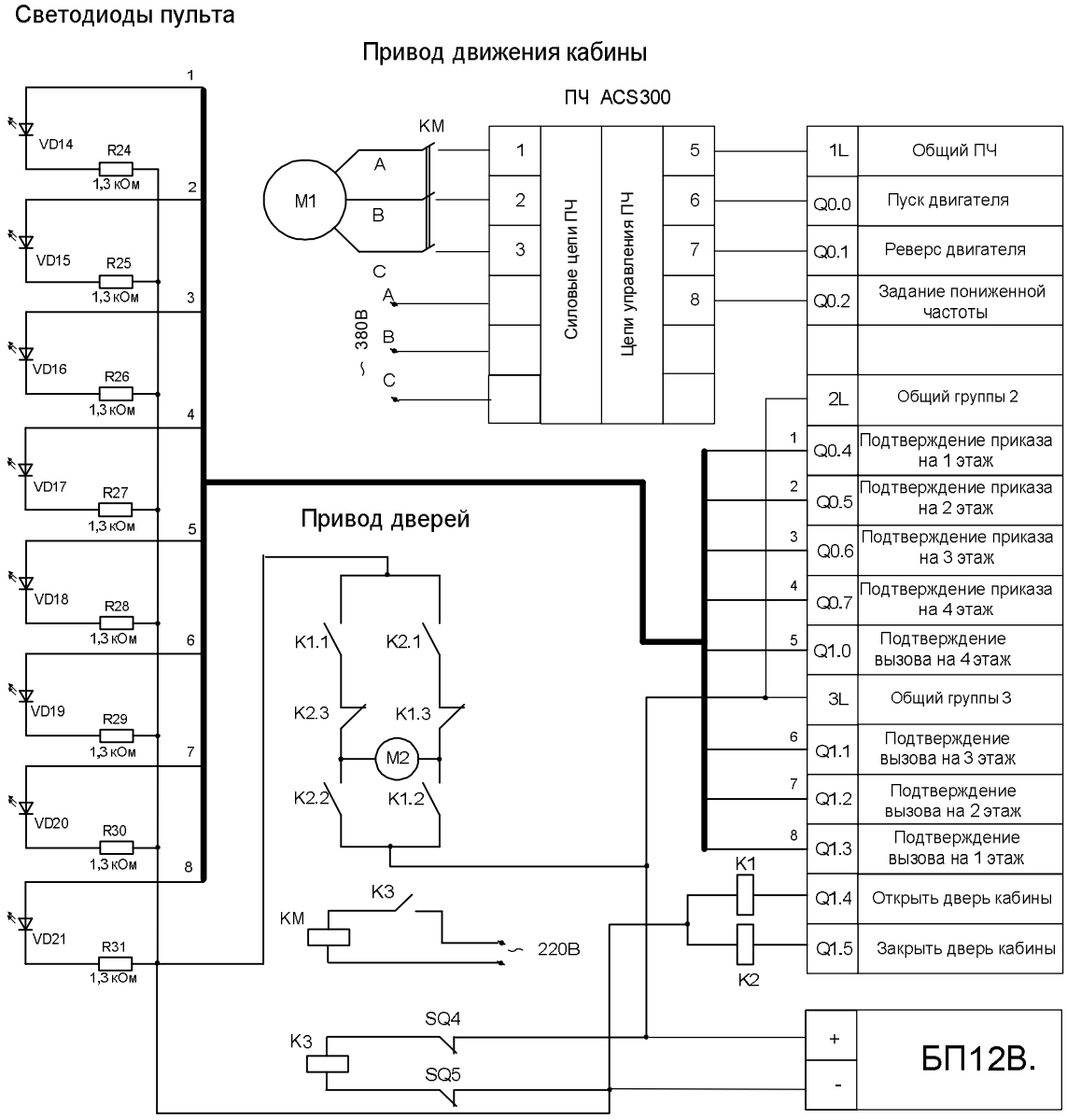


Рис. 3.173. Схема соединений. Выходные цепи контроллера.

На рис. 3.173 обозначены:

VD14 – VD21 – светодиоды на пульте управления;

R24 –R31 – токоограничивающие сопротивления;

M1 – двигатель главного привода (асинхронный);

M2 – двигатель привода дверей кабины (постоянного тока);

KM – магнитный пускатель, отключающий главный привод при выходе кабины за пределы рабочей зоны;

K1, K2 – реле управления приводом дверей кабины;

K3 – реле аварийного отключения главного привода, управляется концевыми выключателями SQ4 и SQ5, расположенными ниже первого этажа и выше четвертого этажа на направляющей шахты.

Выходы контроллера объединены в 3 группы: 1L, 2L, 3L. Под управление преобразователем частоты выделена группа 1L, для управления светодиодами пульта управления и приводом открытия дверей – группы 2L и 3L.

Цепи группы 1L питаются от блока питания преобразователя напряжением 24 В, цепи групп 2L и 3L – от отдельного источника напряжением 12В.

3.2.3.3. Имитационная модель лифтового механизма

Задача модели состоит в определении положения кабины в шахте, положения дверей, состояния этажных и межэтажных датчиков.

Входами модели являются:

команды закрытия и открытия для привода дверей;

команды пуска, реверса и перехода на пониженную скорость для главного привода кабины.

Все команды формируются контроллером и передаются в модель через блок OPC Read. Рассчитанные моделью значения сигналов передаются контроллеру через два блока OPC Write (один для дискретных сигналов, другой – для аналоговых). Ниже приведен список глобальных переменных контроллера, в котором жирным выделены объявления переменных, участвующих в обмене.

```
VAR_GLOBAL
(*Управление моделью*)
Start:BOOL:=FALSE; (*пуск главного привода, выход*)
Reverse:BOOL:= FALSE; (*реверс главного привода (вверх), выход*)
LowSpeed:BOOL:=FALSE; (*включение пониженной скорости, выход*)
Open:BOOL:=FALSE; (*двери кабины открыть, выход*)
Close:BOOL:=FALSE; (*двери кабины закрыть, выход*)
(*Выходы модели*)
Floor1:BOOL:=FALSE; (*датчик первого этажа, вход*)
Floor2:BOOL:=FALSE; (*датчик второго этажа, вход*)
Floor3:BOOL:=FALSE; (*датчик третьего этажа, вход*)
Floor4:BOOL:=FALSE; (*датчик четвертого этажа, вход*)
UpFloor2:BOOL:=FALSE; (*датчик подхода ко 2 этажу снизу, вход*)
UpFloor3:BOOL:=FALSE; (*датчик подхода к 3 этажу снизу, вход*)
UpFloor4:BOOL:=FALSE;  (*датчик подхода к 4 этажу снизу, вход*)
DownFloor1:BOOL:=FALSE;(*датчик подхода к 1 этажу сверху, вход*)
DownFloor2:BOOL:=FALSE;(*датчик подхода к 2 этажу сверху, вход*)
DownFloor3:BOOL:=FALSE;(*датчик подхода к 3 этажу сверху, вход*)
Opened:BOOL:=FALSE;  (*двери кабины открыты, вход*)
Closed:BOOL:=FALSE;  (*двери кабины закрыты, вход*)
y:REAL:=10; (*положение кабины, вход*)
x:REAL:=0; (*положение дверей, вход*)
(*Связь с Trace Mode*)
CallsUp:BYTE:=0; (*вызовы вверх, выход*)
CallsDown:BYTE:=0; (*вызовы вниз, выход *)
Orders:BYTE:=0; (*приказы, выход*)
(*y:REAL:=10; - положение кабины, выход*)
(*x:REAL:=0; - положение дверей, выход*)
CallsUpForm:BYTE:=0; (*вызовы вверх, вход*)
CallsDownForm:BYTE:=0; (*вызовы вниз, вход*)
OrdersForm:BYTE:=0; (*приказы, вход*)
(*"Внутренние" переменные*)
(*CallsUp:BYTE:=0; - вызовы вверх *)
(*CallsDown:BYTE:=0;  - вызовы вниз*)
(*Orders:BYTE:=0;  - приказы *)
Floors:BYTE:=0; (*этажные датчики*)
Appr_Floors_Up:BYTE:=0; (*межэтажные датчика подхода снизу*)
Appr_Floors_Down:BYTE:=0; (*межэтажные датчика подхода сверху*)
State:BYTE:=1; (*состояние автомата*)
END_VAR
```

Simulink-диаграмма верхнего уровня модели показана на рис. 3.174.

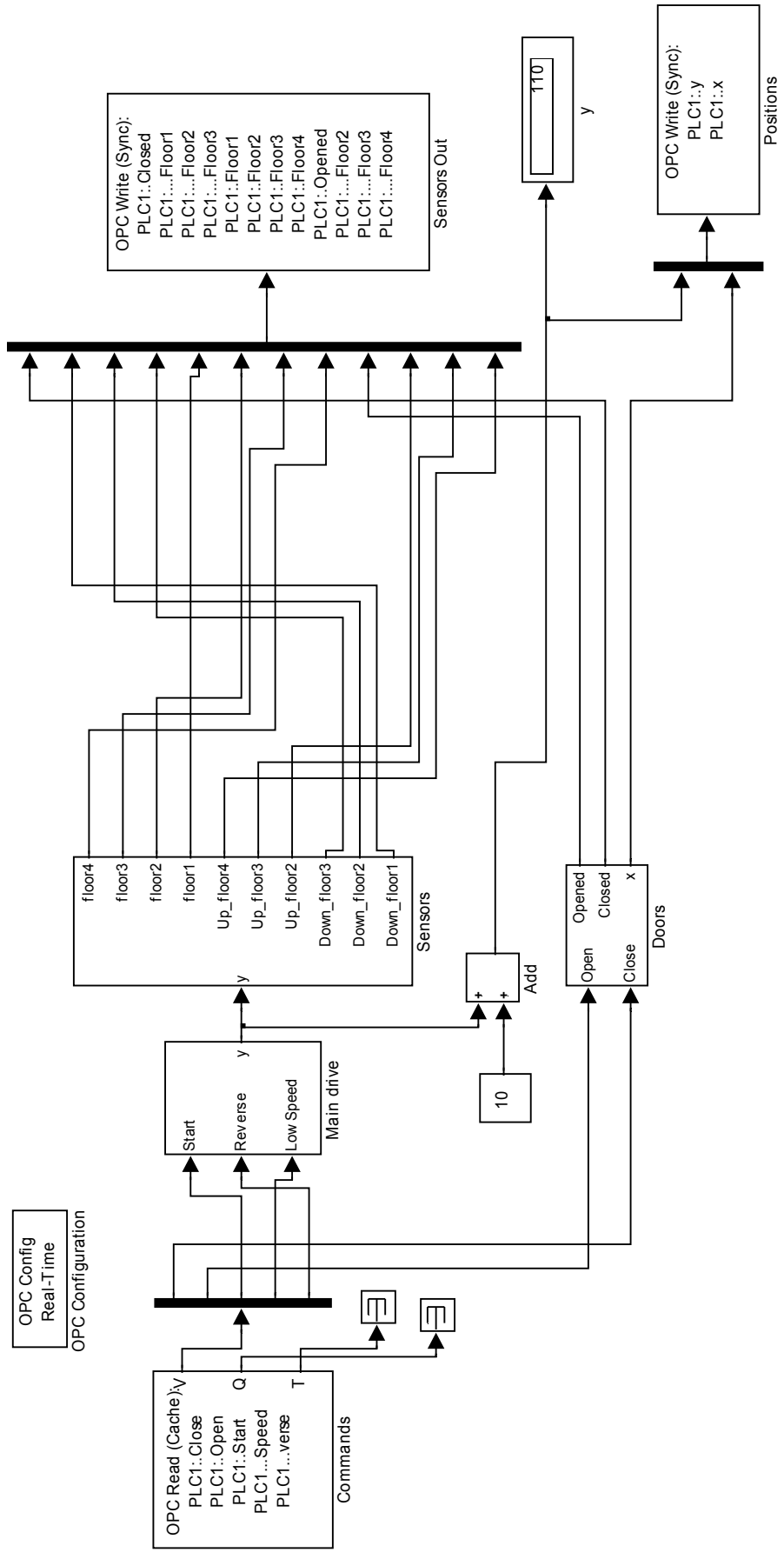


Рис. 3.174. Simulink-модель лифтового механизма.

Подсистема Main drive (рис. 3.175) представляет собой модель главного привода, включающего преобразователь частоты, двигатель, редуктор и другие элементы, приводящие в движение кабину лифта.

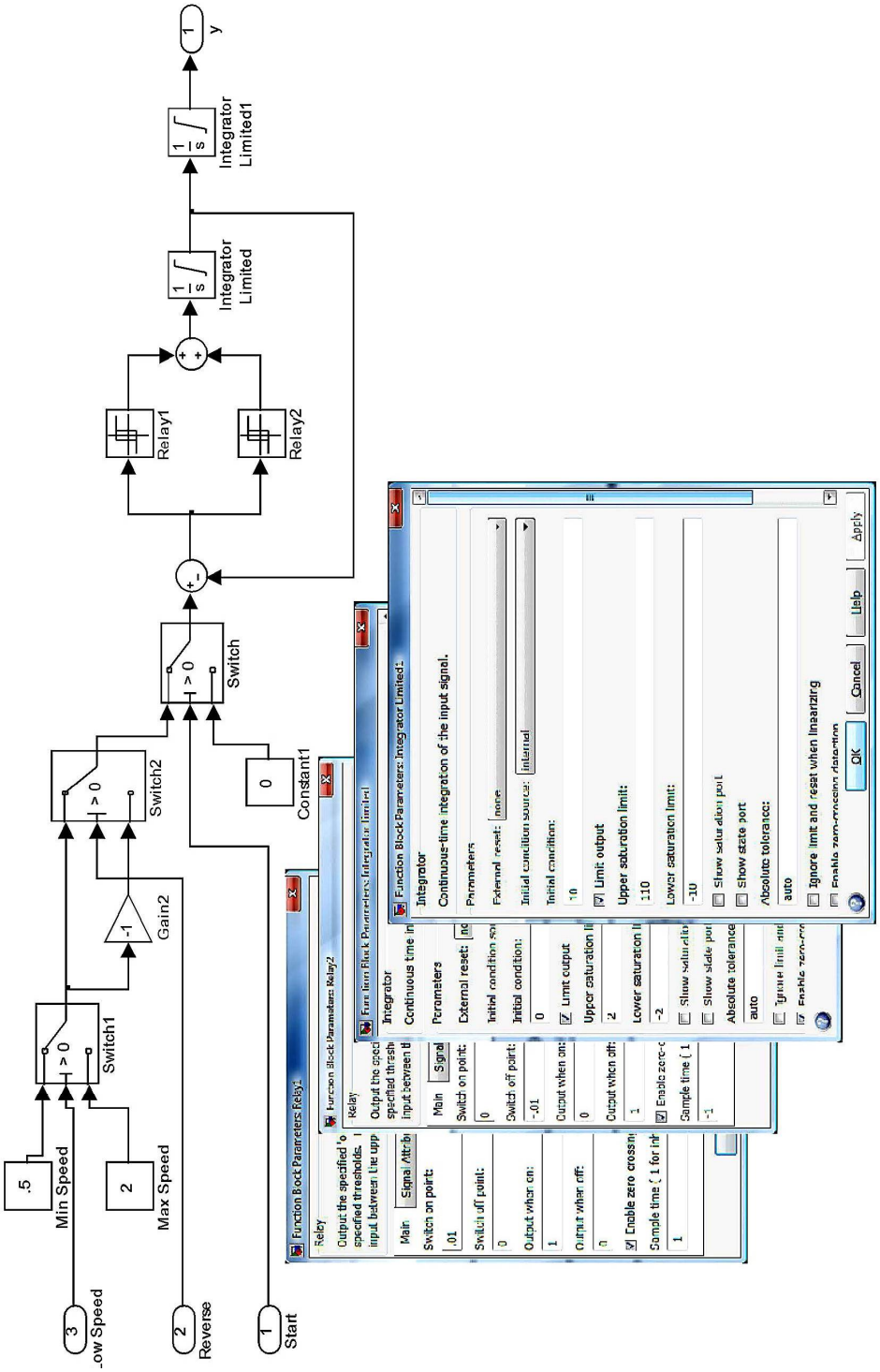


Рис. 3.175. Подсистема Main Drive и ее настройки.

Модель построена на двух интеграторах, на выходе первого из которых формируется вертикальная скорость кабины, на выходе второго – ее положение в шахте. Релейный регулятор скорости обеспечивает постоянное ускорение разгона и торможения, имитируя тем самым реальное поведение преобразователя частоты. Задание по скорости формируется с помощью трех переключателей, управляемых входными сигналами блока. На рис. 3.175 показаны также окна настроек релейных элементов и интеграторов.

На рис. 3.176 представлена подсистема Sensors, формирующая модельные эквиваленты сигналов этажных и межэтажных датчиков. Подсистема построена на блоках Interval Test, фиксирующих нахождение входного сигнала (положения кабины) в заданном диапазоне.

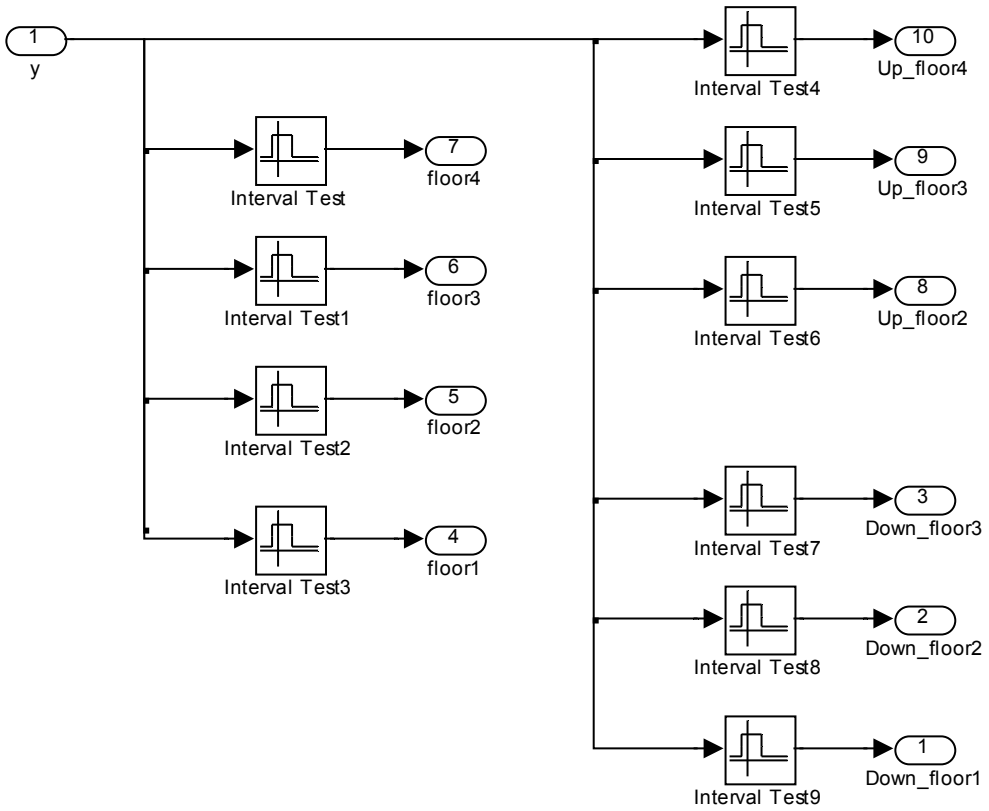


Рис. 3.176. Подсистема Sensors.

Пределы диапазонов блоков приведены в табл. 3.1.

Таблица 3.1

Параметры блоков Interval Test

Датчик	Upper Limit	Lower Limit
Interval Test, floor4	101	99
Interval Test1, floor3	67	65
Interval Test2, floor2	34	32
Interval Test3, floor1	1	-1
Interval Test4, Up_floor4	96	94
Interval Test5, Up_floor3	62	60
Interval Test6, Up_floor2	29	27
Interval Test7, Down_floor3	71	69
Interval Test8, Down_floor2	39	37
Interval Test9, Down_floor1	6	4

Подсистема Doors представлена на рис. 3.177. Она моделирует работу привода дверей и формирует сигналы положения дверей и состояния концевых выключателей.

Некоторые моменты требуют дополнительных пояснений:

1) выходной сигнал блока Main Drive согласно настройкам выходного интегратора изменяется в диапазоне от –10 до 110. Нулевое значение означает на-

хождение кабины на первом этаже, значение 100 – на четвертом. В системе визуализации, рассмотренной ниже, соответствующая переменная изменяется от 0 до 120. В связи с этим потребовалось увеличить выходной сигнал Main Drive на 10. Отметим, что и модель и система визуализации допускают выход кабины за разрешенные пределы (ниже первого и выше четвертого этажей);

2) «аналоговые» сигналы x и y являются фиктивными в том смысле, что в реальной системе таких сигналов нет. Они потребовались исключительно для задач визуализации и программе управления не задействуются (на самом деле сигнал y может быть сформирован подсчетом импульсов датчика на валу главного привода, но в дальнейшем этот вариант не рассматривается).

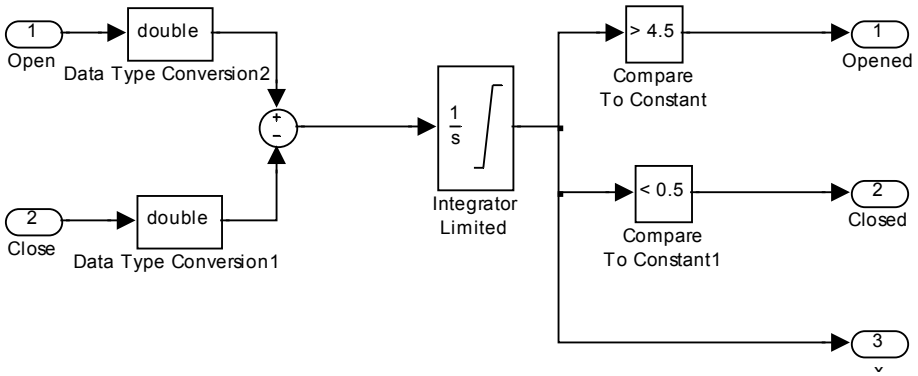


Рис. 3.177. Подсистема Doors.

3.2.3.4. Разработка управляющей программы

Сигналы, которые задают программу движения лифтом, могут поступать как из кабины, так и с лестничных площадок. Схема их обработки может быть раздельной или же собирательной. В первом случае система реагирует на первый поступивший сигнал, и игнорирует во время его выполнения на все последующие. Собирательный же принцип предполагает восприятие уже нескольких команд, и их выполнение в оптимальной последовательности.

Собирательный принцип получил широкое распространение. На вызывном аппарате лифтов с такой системой управления располагаются две кнопки – вверх и вниз, и при вызове системе управления задается не только этаж, на котором находится вызывающий, но и требуемое направление движения.

Лифт, движущийся вниз, делает остановки только в соответствии с вызовами вниз, собирая, таким образом, пассажиров. Вызовы вверх при этом игнорируются (хотя и запоминаются системой управления). При движении вверх игнорируются вызовы вниз.

Решение о реверсе (изменении направления) принимается всегда в «крайних точках»: при движении вверх на самом высоком этаже из всех запрошенных целей движения, по вызову вниз или по приказу; при движении вниз на самом низком этаже из всех запрошенных целей движения, по вызову вверх или по приказу. Собирательный принцип упорядочивает движение лифта, сокращая до минимума количество челночных движений кабины и экономя тем самым электрическую энергию. Ниже будет рассмотрен только собирательный принцип.

Для того чтобы сделать программу управления универсальной, т.е. практически независимой от этажности здания, и использовать в ней «мощные» команды, оперирующие байтами и словами, введем следующие переменные (рис. 3.178).

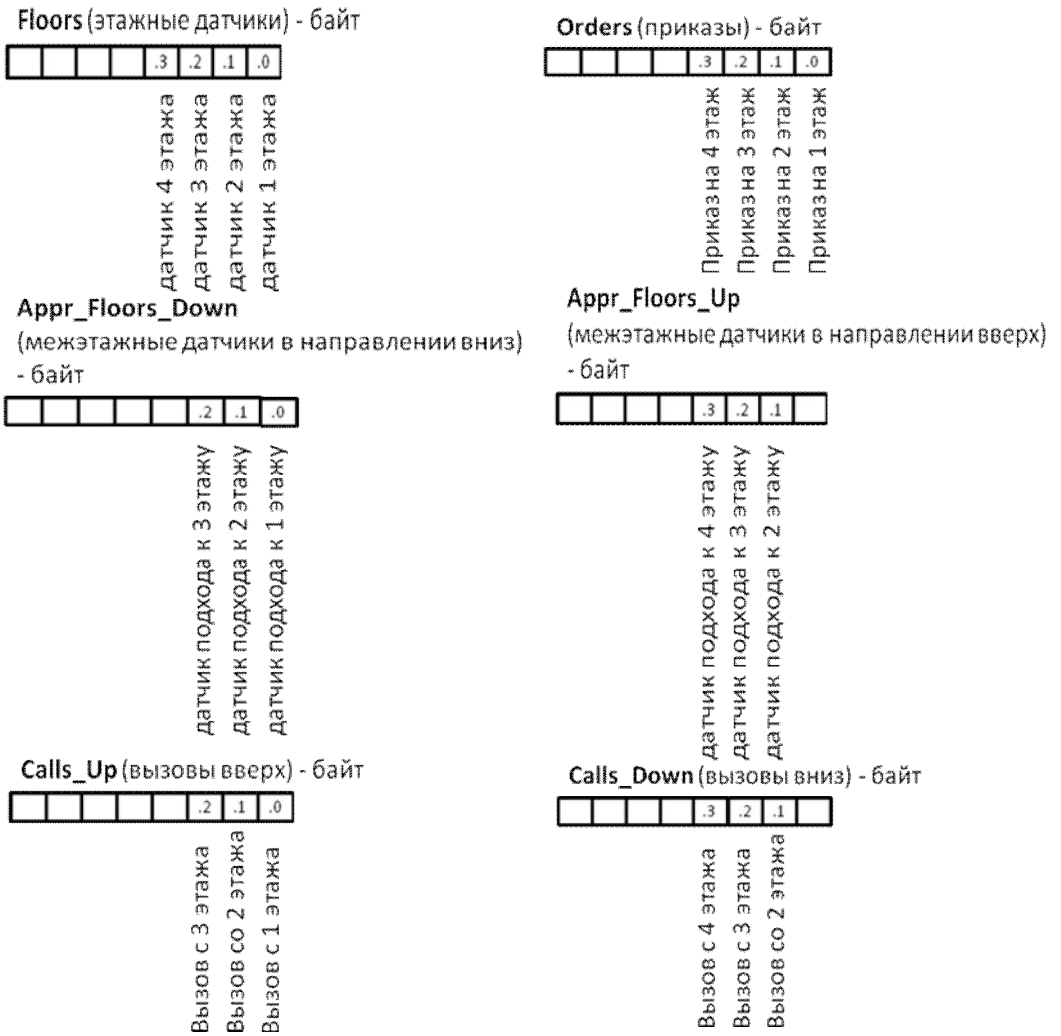


Рис. 3.178. Формирование переменных

Эти переменные будут «заполняться» в начале каждого цикла контроллера после опроса входов, связанных с этажными и межэтажными датчиками и кнопками пульта управления.

В нашей реализации используются байты, поэтому максимальная этажность здания, обслуживаемого системой равна восьми. Однако программу можно будет достаточно просто передать и для большего числа этажей, если вместо байтов использовать слова (два байта, 16 этажей) или двойные слова (4 байта, 32 этажа).

Концепция программы представлена на рис. 3.179.

Главная программа:

1) производит опрос этажных и межэтажных датчиков и *копирует* биты соответствующих входов в байтовые переменные `Floors`, `Appr_Floors_Up`, `Appr_Floors_Down`;

2) производит опрос входов, связанных с кнопками пульта и *запоминает* вызовы и приказы, *устанавливая* биты байтовых переменных `Calls_Up`, `Calls_Down`, `Orders`. Сбрасываться эти биты будут только после выполнения соответствующих вызовов и приказов;

3) управляет светодиодами пульта простым *копированием* битов из байтовых переменных `Calls_Up`, `Calls_Down`, `Orders` в соответствующие выходы контроллера;

4) организует работу конечного автомата, вызывая подпрограммы в зависимости от его состояния.

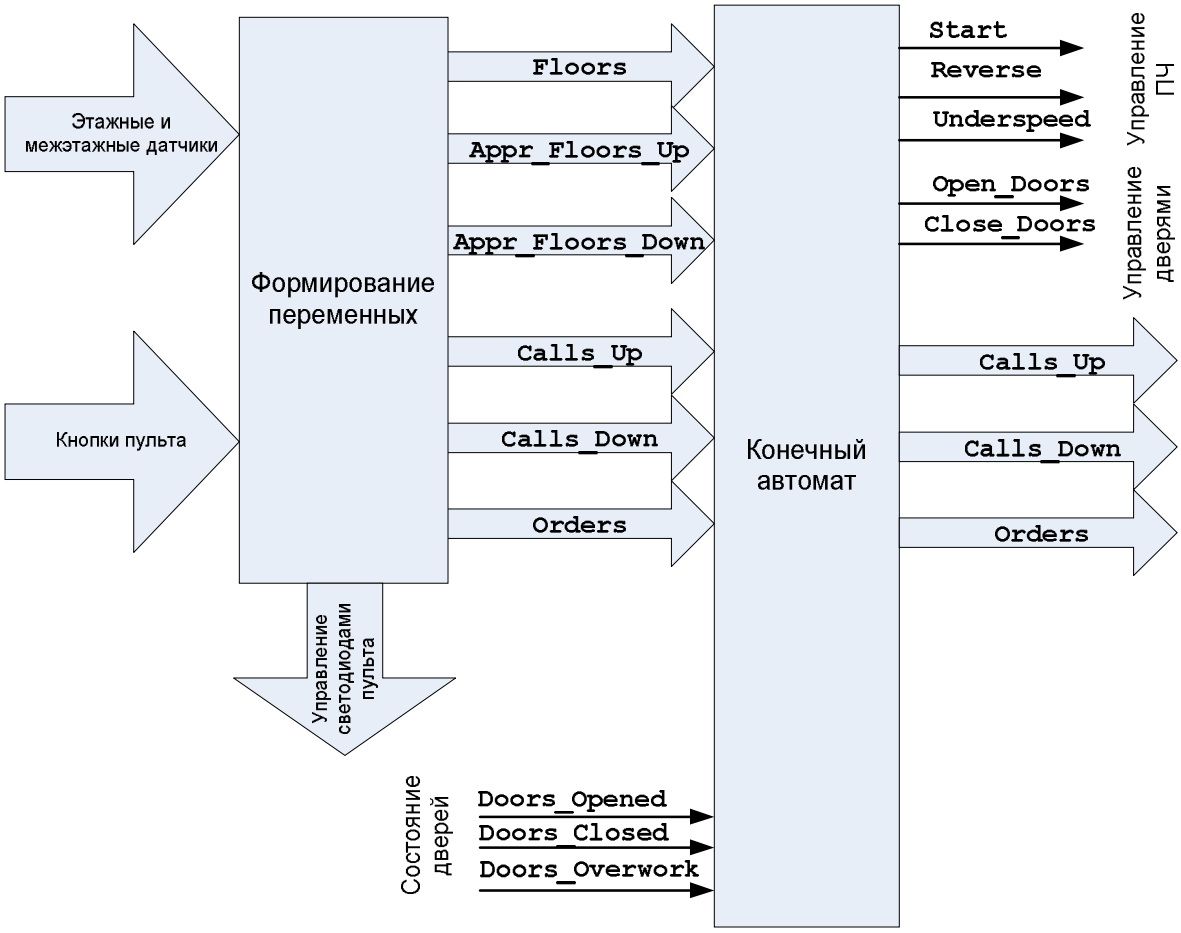


Рис. 3.179. Структура программы.

Конечный автомат формирует команды управления преобразователем частоты (пуск, реверс, переход на пониженную скорость) и приводом дверей кабины (открыть, закрыть) в зависимости от:

1) своего состояния;

2) сигналов этажных и межэтажных датчиков, сгруппированных в переменных `Floors`, `Appr_Floors_Up`, `Appr_Floors_Down`;

3) действующих вызовов и приказов, сгруппированных в переменных `Calls_Up`, `Calls_Down`, `Orders`;

4) сигналов с концевых выключателей дверей `Doors_Opened`, `Doors_Closed` и датчика их перегрузки `Doors_Overwork`.

Помимо этого, конечный автомат сбрасывает биты выполненных вызовов и приказов в байтовых переменных `Calls_Up`, `Calls_Down`, `Orders`.

Структура конечного автомата показана на рис. 3.180.

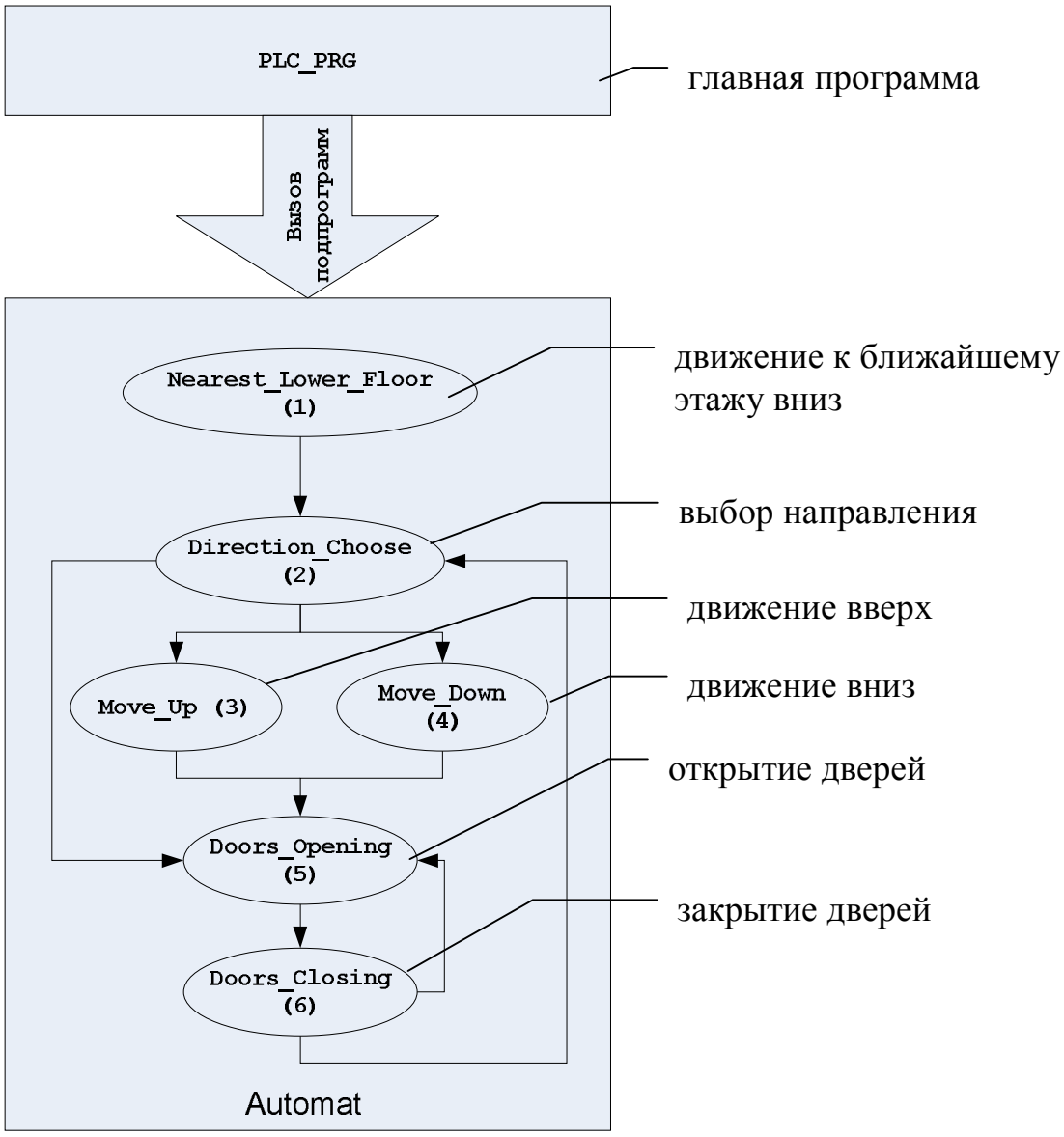


*Рис. 3.180.*Структура конечного автомата.

Прокомментируем ряд выражений, которые могут оказаться трудными для понимания.

«Достигнут какой бы то ни было этаж?» (`Nearast_Lower_Floor`):

```
If (Floors==0)
```

переменная `Floors` равна нулю, если все ее биты равны нулю.

«Формирование переменной «Цели» (`Direction_Choose`):

```
unsigned Goals = Calls_Up | Calls_Down | Orders;
```

переменная `Goals` (цели) производится «побитовым сложением» (операция ИЛИ) переменных `Calls_Up` (вызовы вверх), `Calls_Down` (вызовы вниз), `Orders` (приказы). В результате все установленные в этих трех переменных биты попадут в переменную `Goals`.

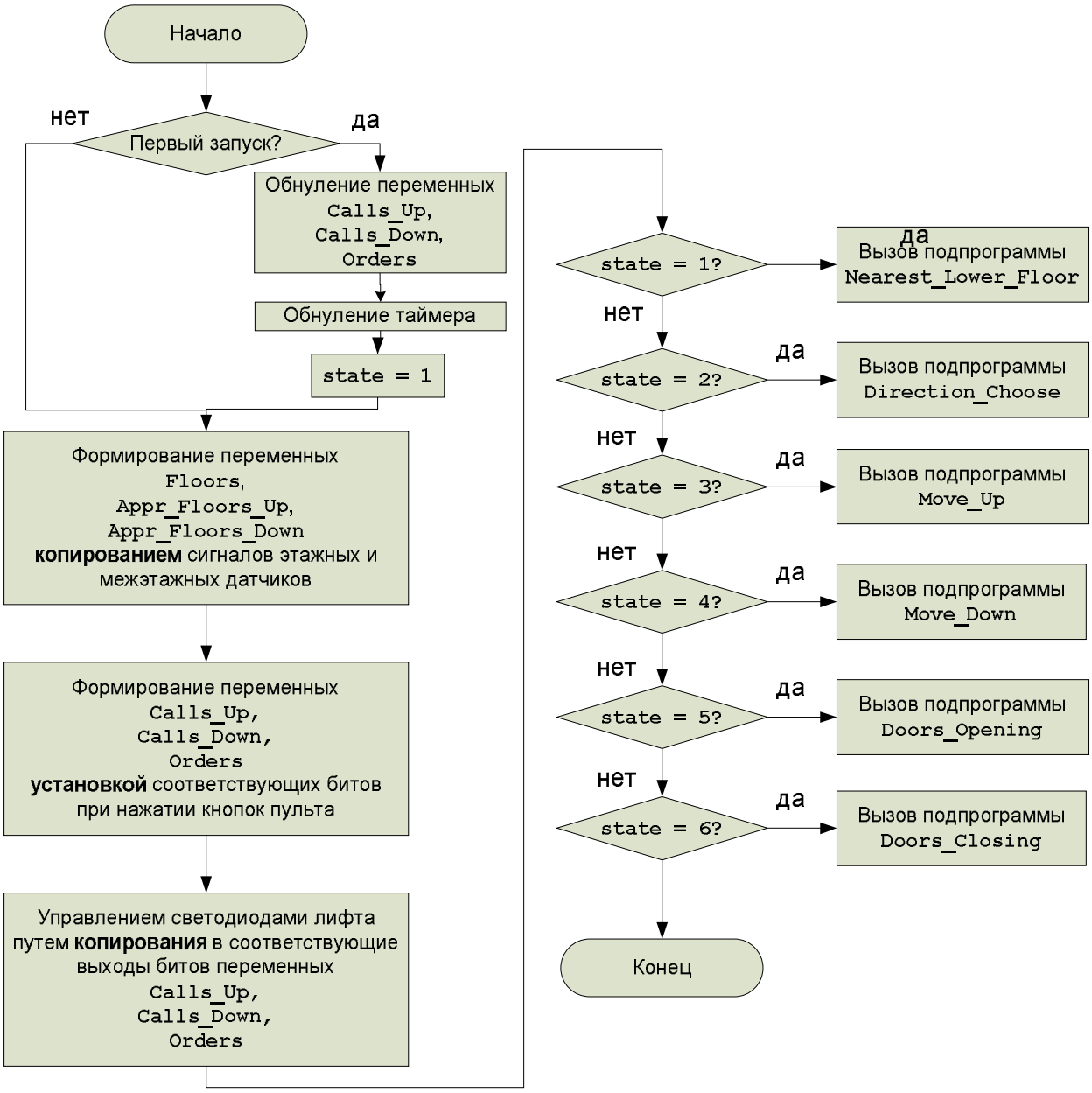


Рис. 3.181. Структура главной программы `PLC_PRG`.

«Цели есть?» (`Direction_Choose`):

```
If(! Goals)
```

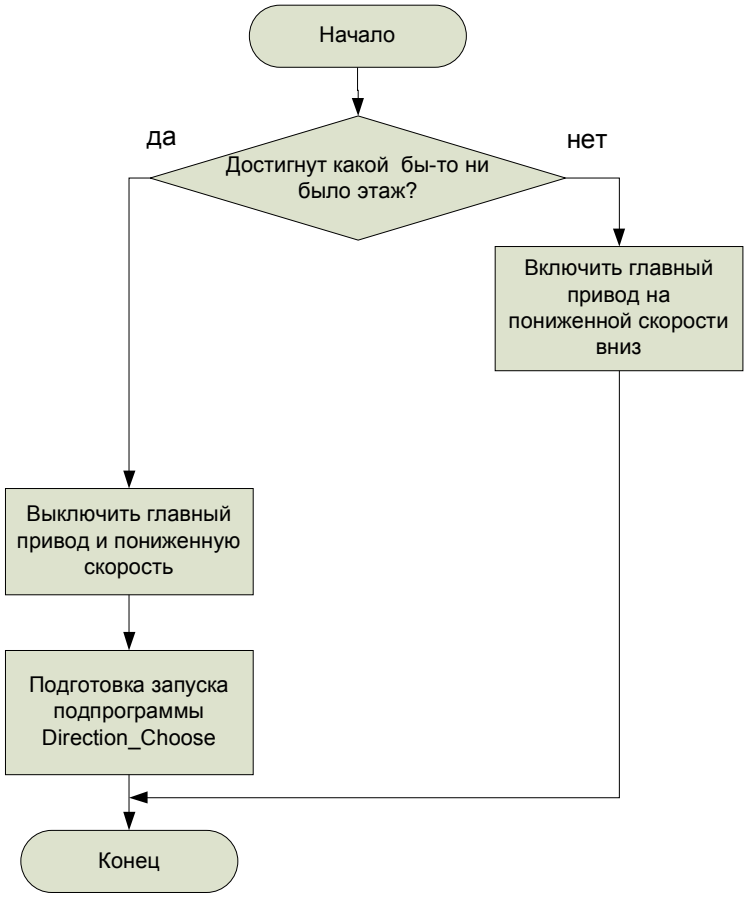
логическое отрицание (`!`) делает из любого ненулевого значения нулевое («ложь»), а из нулевого – единицу («истину»). Таким образом, если условие «сработает» если целей нет (переменная `Goals` равна нулю).

«Цель достигнута?» (`Direction_Choose`):

```
If(Goals&Floors)
```

логическое умножение (`&`, операция «И») в своем результате устанавливает биты в единицу, только если в соответствующих позициях у каждого операнда эти биты установлены. Переменная `Floors`, на самом деле, может иметь только один установленный в единицу бит или ни одного (кабина находится на некотором этаже или она находится между этажами). Если бит установлен (ка-

бина на этаже), и его номер (номер этажа) совпадает с одиним из номеров целевых этажей, это означает, что очередная цель достигнута: результат выражения в скобках отличен от нуля и воспринимается как `TRUE`.




```
If(Floors==0)
{
      Start = UnderSpeed = 1;
      Reverse = 0;
 }
else
{
     Start = UnderSpeed = 0;
     state = 2
}
```

Рис. 3.182. Структура программы `Nearast_Lower_Floor`.

«Внизу есть цели?» (`Direction_Choose`):

```
(Floors-1)&Goals
```

переменная `Floors`, как уже было сказано, в случае нахождения кабины на этаже содержит одну единицу в соответствующей позиции. Вычитая из ее единицу, мы, тем самым, получаем слово, в котором единицы установлены левее этой позиции, а остальные биты раны нулю, например:

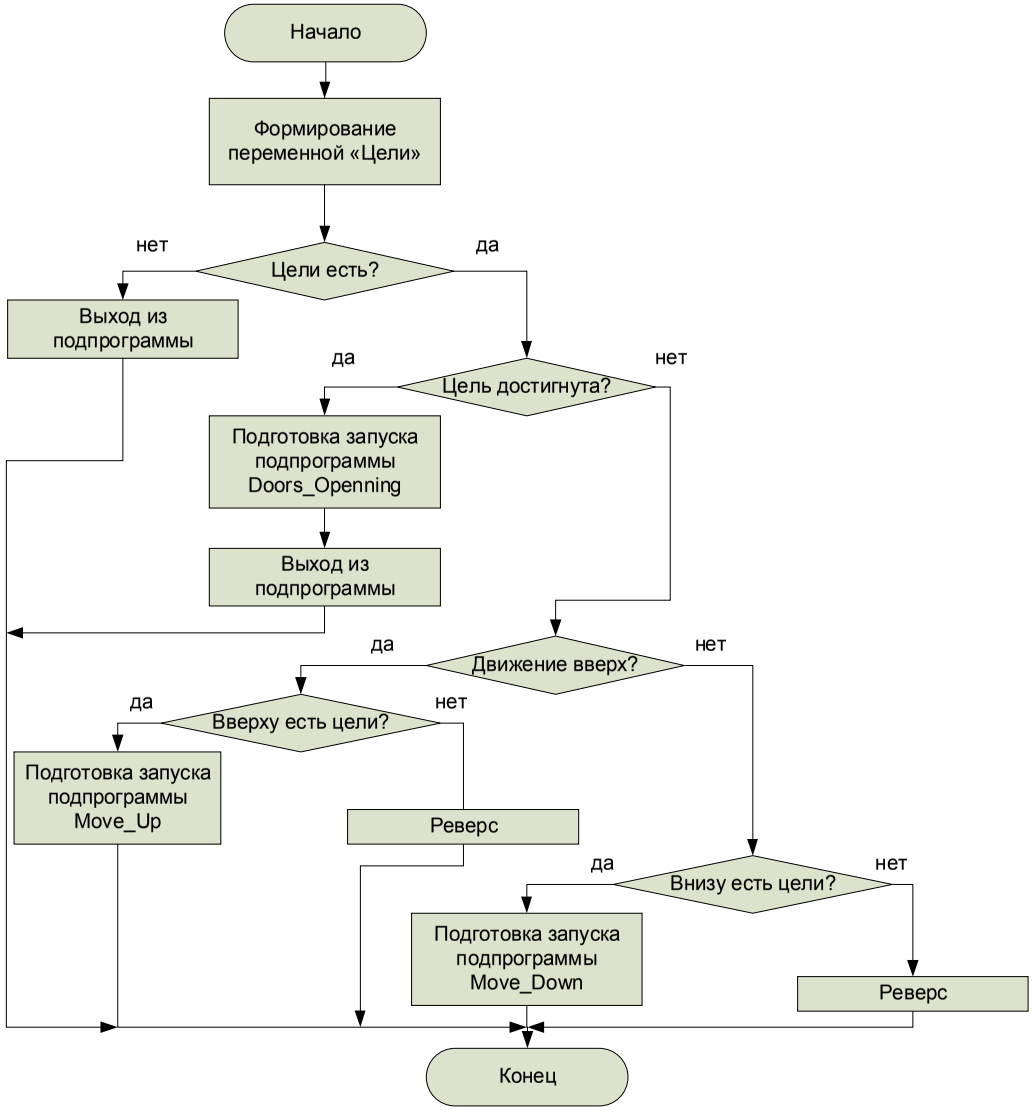
000001000 – 1 = 000000111.

Таким образом, мы получаем «маску», накладывая которую на переменную `Goals` с помощью операции «И», можно выяснить наличие целей ниже текущего этажа. Если результат операции отличен от нуля, такие цели есть.

«Кабина подходит к этажу, заказанному приказом или вызовом вверх?» (`Move_Up`):

```
If(Appr_Floors_Up & Orders || Appr_Floors_Up & Calls_Up))
```

переменная `Appr_Floors_Up` содержит один единственный установленный бит, позиция которого соответствует номеру этажа, к которому кабина подходит снизу. Если в этой же позиции установлен бит в переменной `Orders` (приказы) ИЛИ (`||`) `Calls_Up` (вызовы вверх), результат выражения будет ненулевым (`TRUE`) и условие сработает.




```
unsigned Goals = Calls_Up | Calls_I
Orders;


If(!Goals)
   return;


If(Goals&Floors)
{

 state = 5;



   return;
}

If(Reverse)

   If(Goals>Floors)
     state = 3;
   else
     Reverse = 0;

else if((Floors-1)&Goals)

      state = 4;
    else
      Reverse = 1;
```

Рис. 3.183. Структура программы `Direction_Choose`.

«Кабина подходит к этажу, заказанному вызовом вниз, и выше нет целей?» (`Move_Up`):

```
If((Appr_Floors_Up & Calls_Down) &&
   (Appr_Floors_Up > Appr_Floors_Up^Goals))
```

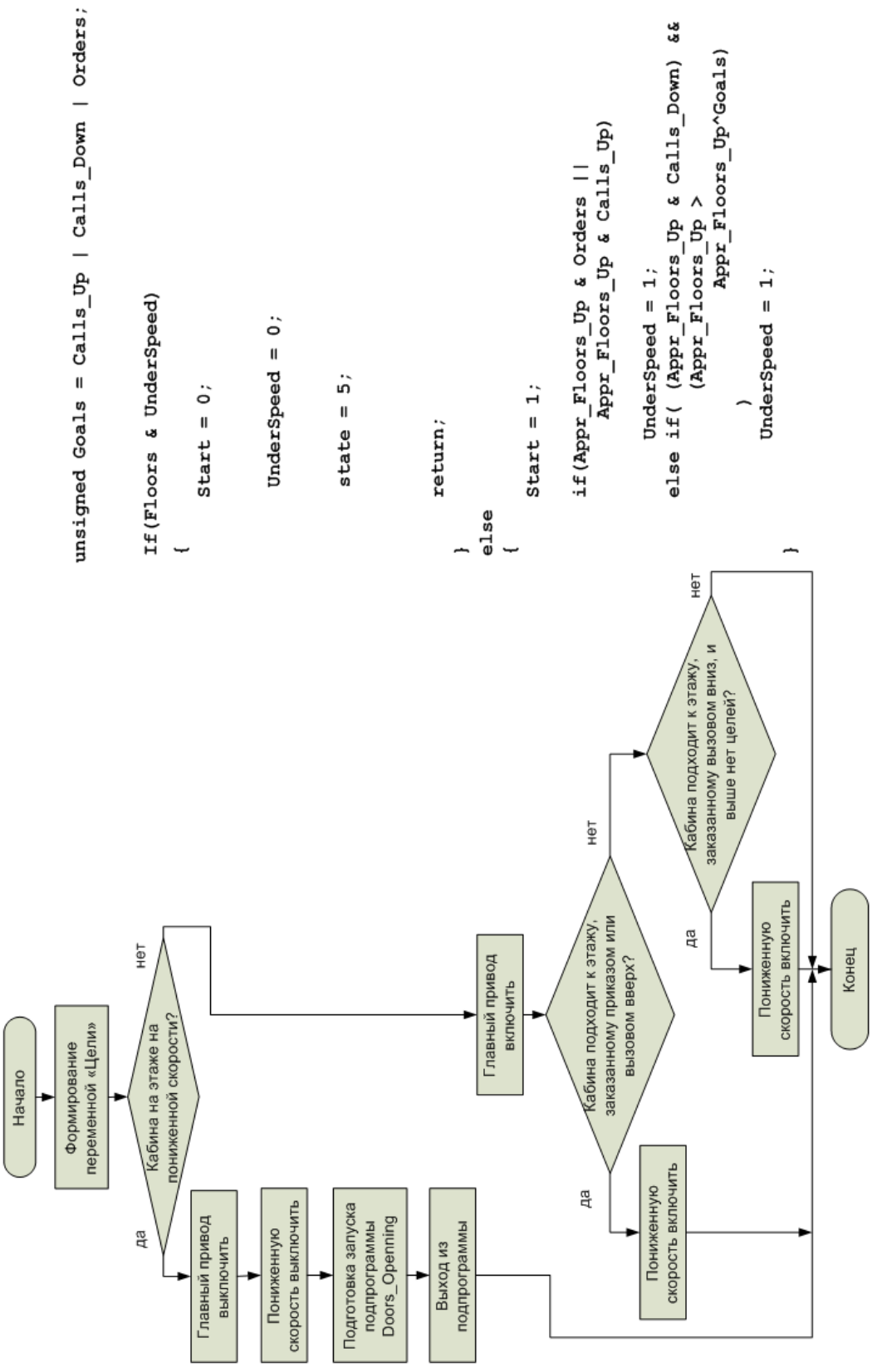


Рис. 3.184. Структура программы `Move_Up`.

`Appr_Floors_Up & Calls_Down`, очевидно, есть ответ на первую часть вопроса: «*Кабина подходит к этажу, заказанному вызовом вниз?*». `&&` – операция логического «И», результат которой – истина (`TRUE`), если оба операнда отличны от нуля.

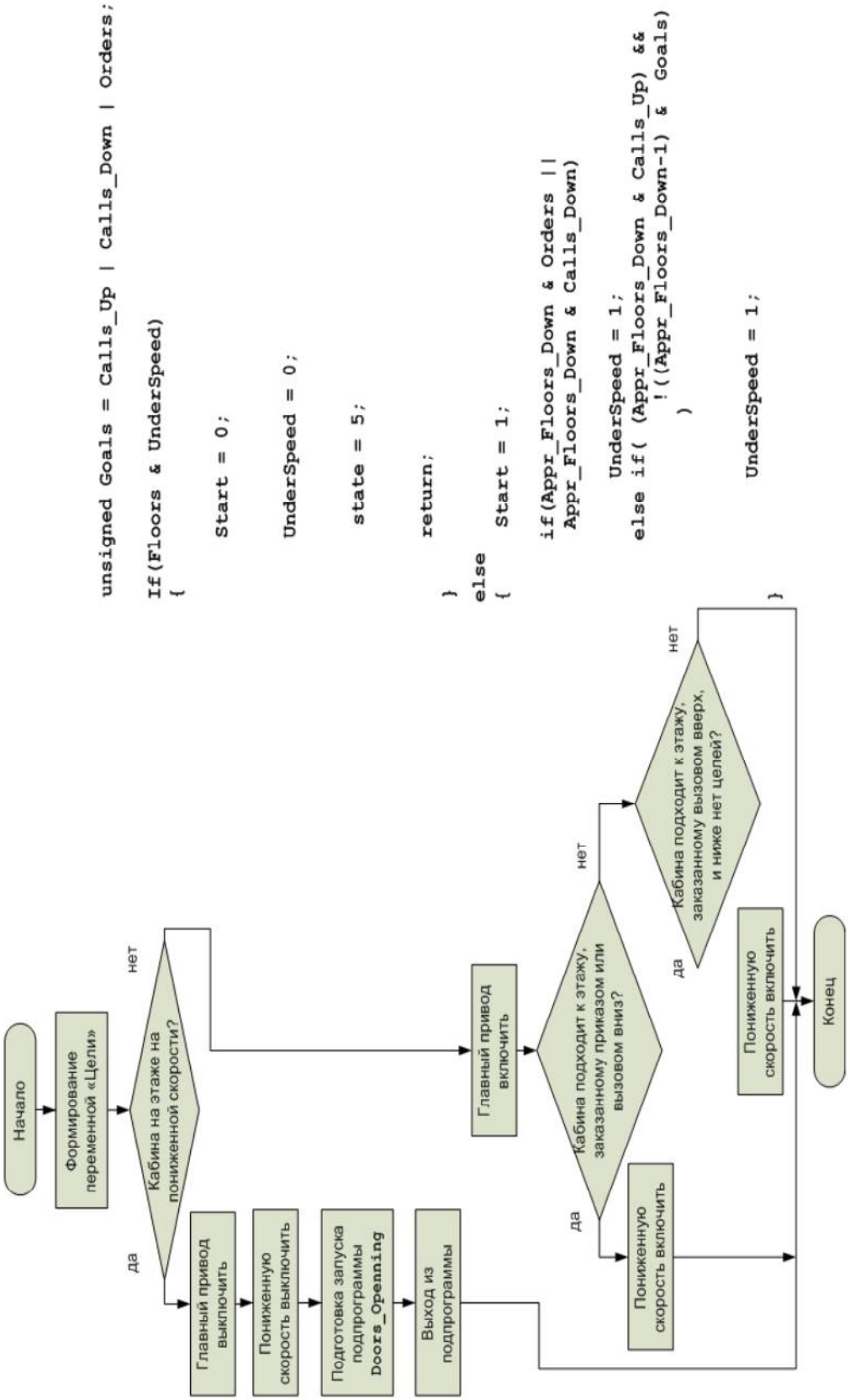


Рис. 3.185. Структура программы `Move_Down`.

Попробуем разобраться со второй частью: «*…выше нет целей?*».

В случае если кабина действительно подходит к этажу, заказанному вызовом вниз, `Appr_Floors_Up` – слово, у которого установлен единственный бит в соответствующей позиции. Исключающее ИЛИ (^) исключит из целей (`Goals`) текущую цель. Если оставшиеся цели «меньше» текущей, это, очевидно, означает, что *выше нет целей*.

«Кабина подходит к этажу, заказанному приказом или вызовом вниз?» (`Move_Down`):

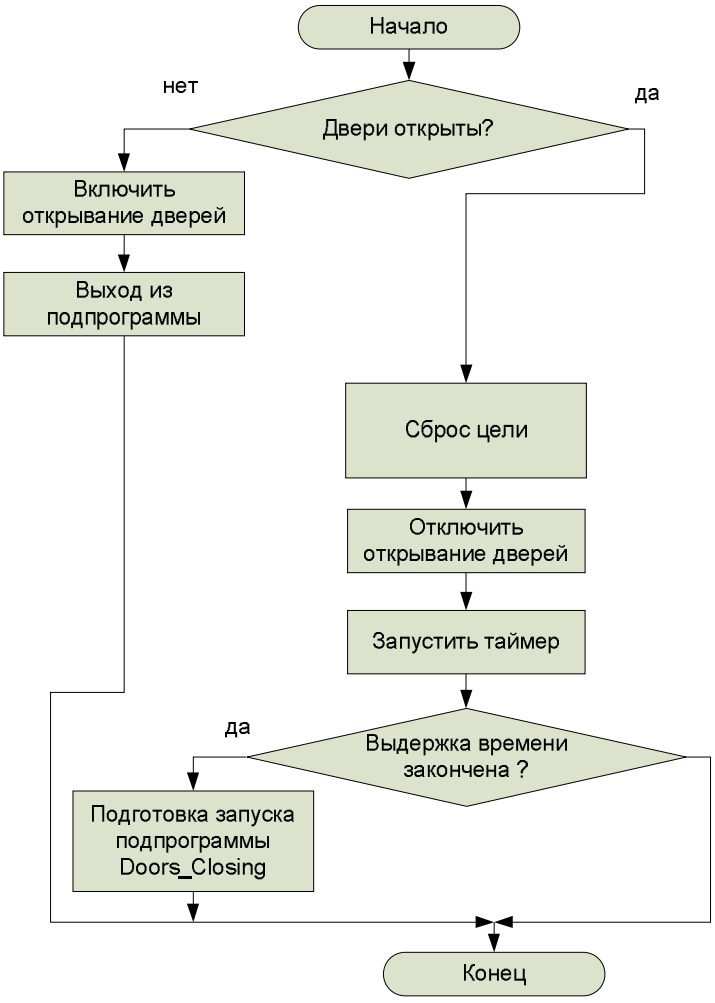
```
If(Appr_Floors_Down & Orders ||
          Appr_Floors_Down & Calls_Down)
```

совершенно аналогично рассмотренному ранее
`If(Appr_Floors_Up & Orders || Appr_Floors_Up & Calls_Up))`.

«Кабина подходит к этажу, заказанному вызовом вверх, и ниже нет целей?» (`Move_Down`):

```
(Appr_Floors_Down & Calls_Up) &&
                !((Appr_Floors_Down-1) &  Goals)
```

`Appr_Floors_Down & Calls_Up` – это ответ на первую часть вопроса: «*Кабина подходит к этажу, заказанному вызовом вверх?*».




```
if (!Doors_Opened)
{
     Open_Doors = 1;

     return;
}
else
{
     Calls_Up&=~Floors;
     Calls_Down&=~Floors;
     Orders&=~Floors;

      Open_Doors = 0;

     Timer(выдержка);
}

if (Timer)

      state = 6;
```

Рис. 3.186. Структура программы `Doors_Opening`.

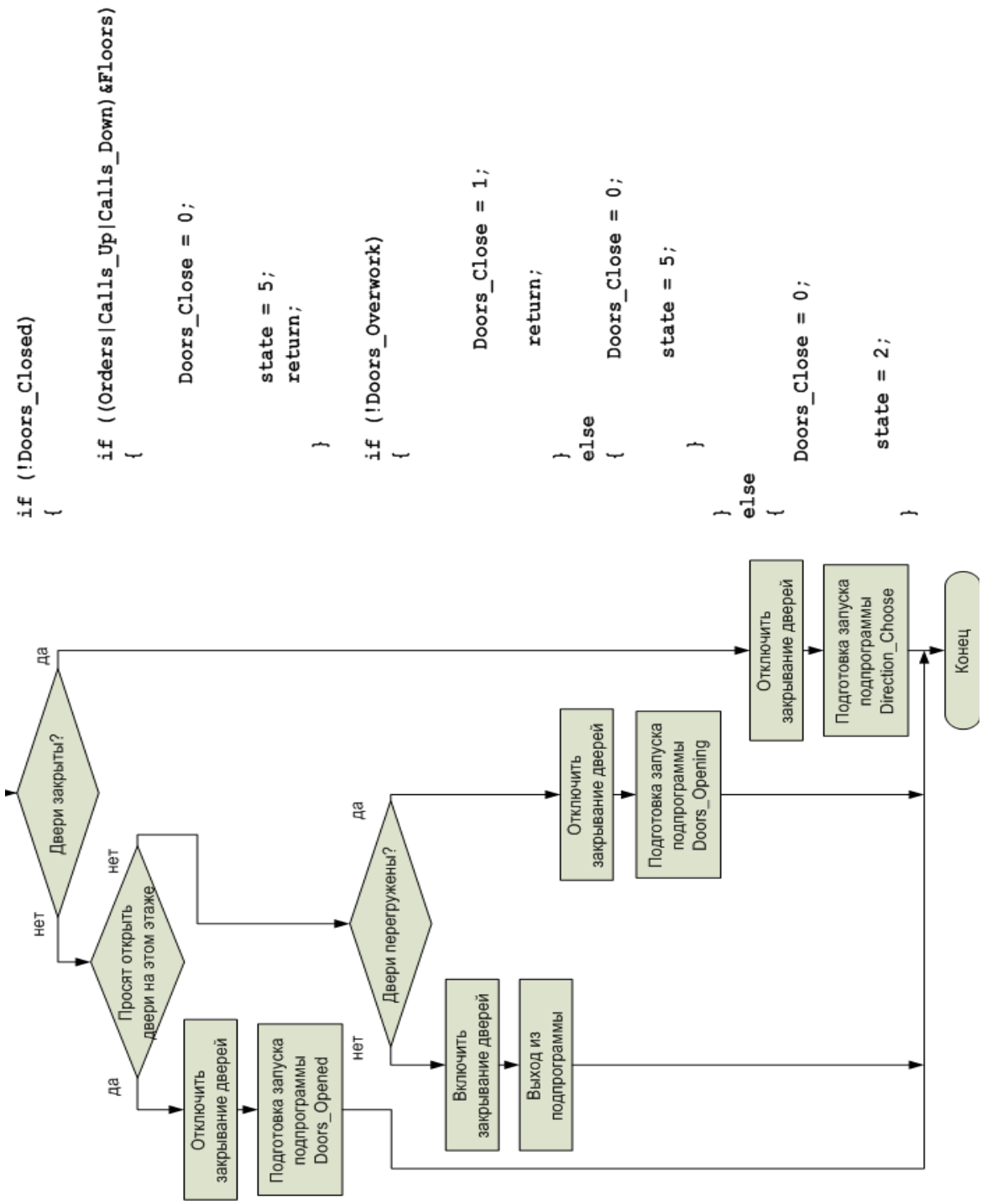


Рис. 3.187. Структура программы `Doors_Closing`.

`Appr_Floors_Down-1` – делаем «маску из ниже лежащих этажей».

`(Appr_Floors_Down-1) & Goals` – не нуль (истина), если снизу есть цели.

`!((Appr_Floors_Down-1) & Goals)` – логическая инверсия, возвращает `TRUE`, если ниже целей нет.

«Вверху есть цели?» (`Direction_Choose`):

```
If(Goals>Floors)
```

номера целевых этажей больше номера текущего этажа.

«Сброс цели» (`Doors_Opening`):

```
Calls_Up&=~Floors;
Calls_Down&=~Floors;
Orders&=~Floors;
```

– это – то же самое, что

```
Calls_Up = Calls_Up & ~Floors;
Calls_Down = Calls_Down &~Floors;
Orders= Orders & ~Floors;
```

Операция «~» (побитовая инверсия), примененная к переменной `Floors` формирует маску, в которой установлены все единицы, за исключением позиции, соответствующей текущему этажу. Прикладывая эту маску к переменным `Calls_Up` (вызовы вверх), `Calls_Down` (вызовы вниз), `Orders` (приказы), мы тем самым сбрасываем соответствующие биты в этих переменных, исключая текущий этаж из «целевых списков».

Программа для виртуального контроллера написана на языке ST в CoDeSys. Она практически полностью реализует логику управления и алгоритмы, описанные выше. Исключение составляет подпрограмма `Doors_Closing`: в ней отсутствует обработка сигнала о заклинивании дверей. Этот сигнал, как и ситуация с заклиниванием, в данной модели не предусмотрены в целях упрощения.

Глобальные переменные были объявлены ранее. Главная программа `PLC_PRG` дополнена вызовом подпрограмм (рис. 3.188).

PLC_PRG (PRG-ST)

```
0001 PROGRAM PLC_PRG
0002 VAR
0003 END_VAR
```

```
0001 (*Формирование переменных Floors,Appr_Floors_Up,
0002 Appr_Floors_Down копированием сигналов этажных
0003 и межэтажныхдатчиков*)
0004 Floors.0:=Floor1;
0005 Floors.1:=Floor2;
0006 Floors.2:=Floor3;
0007 Floors.3:=Floor4;
0008
0009 Appr_Floors_Up.1:=UpFloor2;
0010 Appr_Floors_Up.2:=UpFloor3;
0011 Appr_Floors_Up.3:=UpFloor4;
0012
0013 Appr_Floors_Down.0:=DownFloor1;
0014 Appr_Floors_Down.1:=DownFloor2;
0015 Appr_Floors_Down.2:=DownFloor3;
0016
0017 (*Формирование переменных CallsUp,CallsDown,Orders
0018 установкой соответствующих битов при нажатии кнопок
0019 в Trace Mode*)
0020 CallsUp:=CallsUp OR CallsUpForm;
0021 CallsDown:=CallsDown OR CallsDownForm;
0022 Orders:=Orders OR OrdersForm;
0023
0024 CASE State OF
0025 1: Nearest_Lower_Floor;
0026 2: Direction_Choose;
0027 3: Move_Up;
0028 4: Move_Down;
0029 5: Doors_Openning;
0030 6: Doors_Closing;
0031 END_CASE;
```

Рис. 3.188. Программа `PLC_PRG`.

Подпрограммы представлены на рис. 3.189 – 3.194.

Nearest_Lower_Floor (PRG-ST)

```
0001 PROGRAM Nearest_Lower_Floor
0002 VAR
0003 END_VAR
```

```
0001 IF Floors=0 THEN
0002     Start:=TRUE;
0003     Reverse:=FALSE;
0004     LowSpeed:=TRUE;
0005 ELSE
0006     Start:=FALSE;
0007     LowSpeed:=FALSE;
0008     State:=2;
0009 END_IF
0010
```

Рис. 3.189. Подпрограмма `Nearest_Lower_Floor`.

Direction_Choose (PRG-ST)

```
0001 PROGRAM Direction_Choose
0002 VAR
0003     Goals:BYTE;
0004 END_VAR
```

```
0001 Goals:=CallsUp OR CallsDown OR Orders;
0002 IF Goals=0 THEN
0003     RETURN;
0004 END_IF
0005 IF (Goals AND Floors) <> 0 THEN
0006     State:=5;
0007     RETURN;
0008 END_IF
0009 IF Reverse = TRUE THEN
0010     IF Goals > Floors THEN
0011         State:=3;
0012     ELSE
0013         Reverse:=FALSE;
0014     END_IF
0015 ELSE
0016     IF ((Floors-1) AND Goals) <> 0 THEN
0017         State:=4;
0018     ELSE
0019         Reverse:=TRUE;
0020     END_IF
0021 END_IF
```

Рис. 3.190. Подпрограмма `Direction_Choose`.

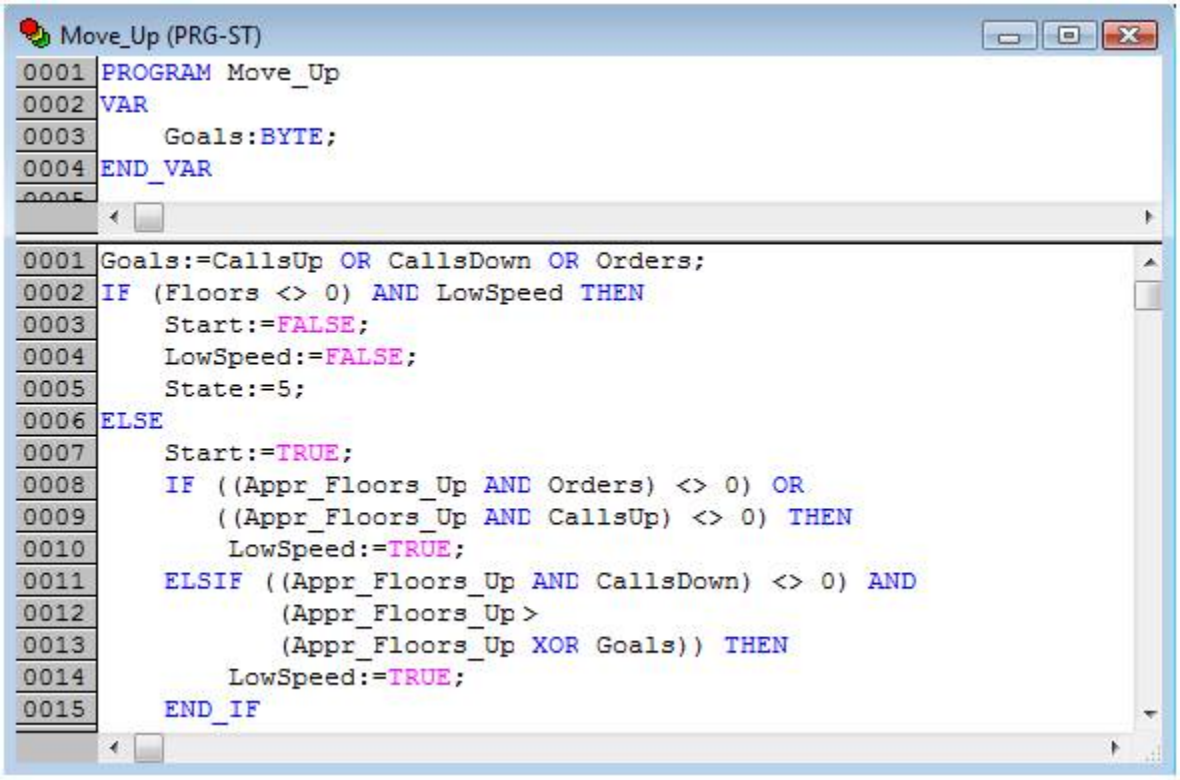


Рис. 3.191. Подпрограмма `Move_Up`.

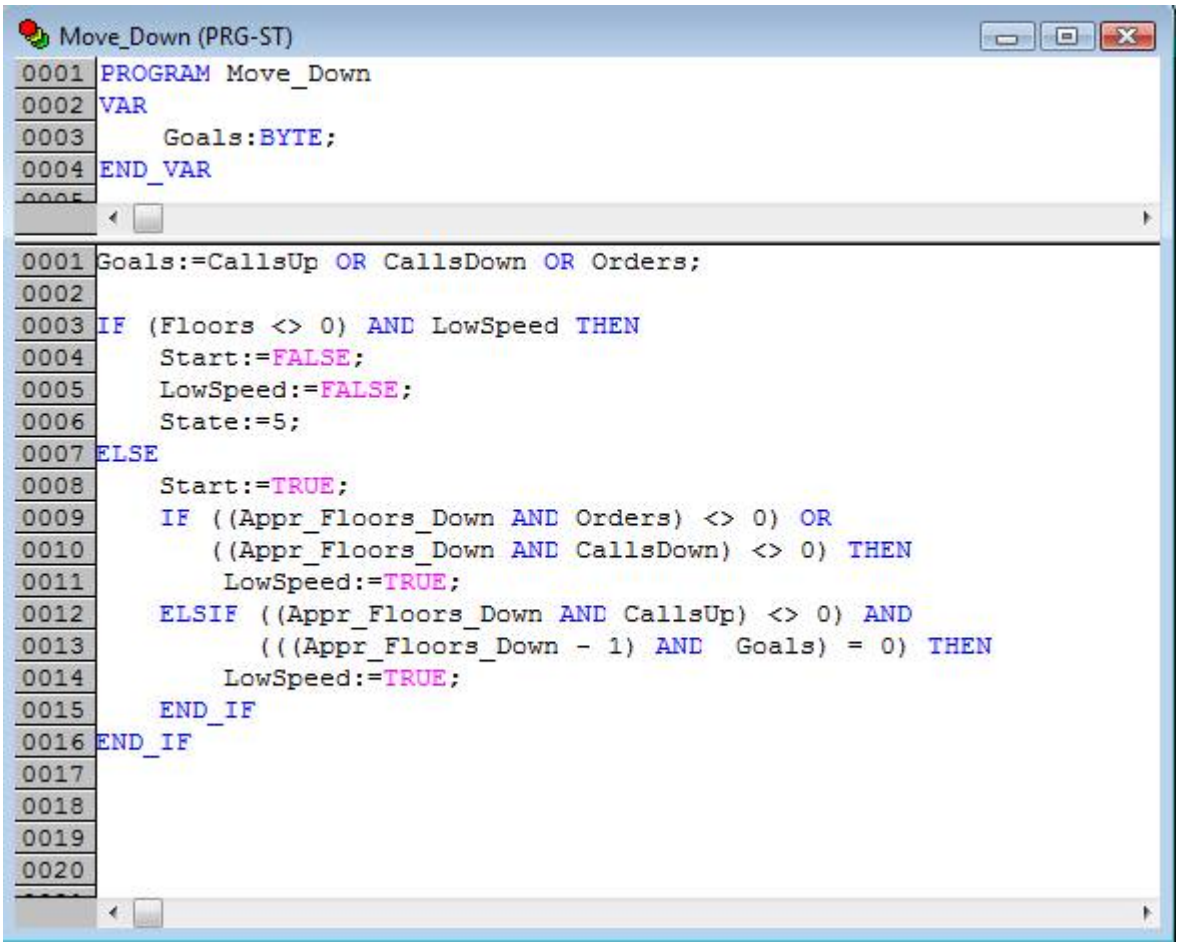


Рис. 3.192. Подпрограмма `Move_Down`.

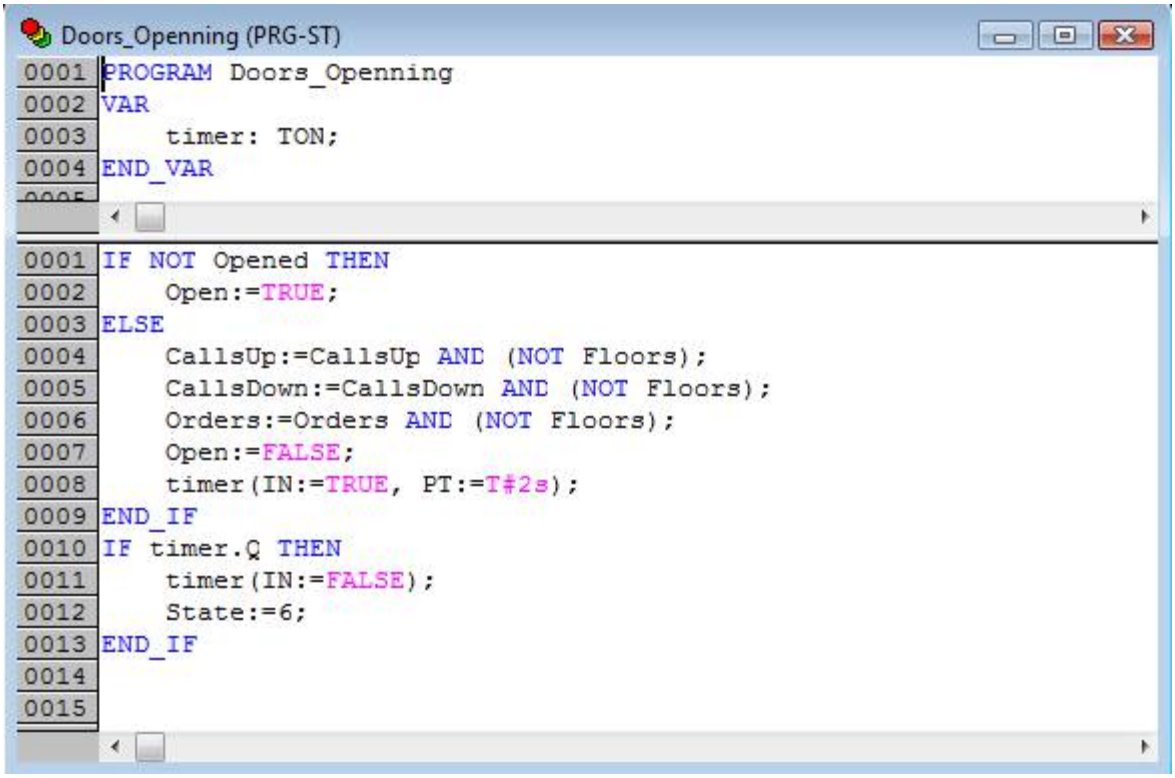


Рис. 3.193. Подпрограмма `Doors_Opening`.

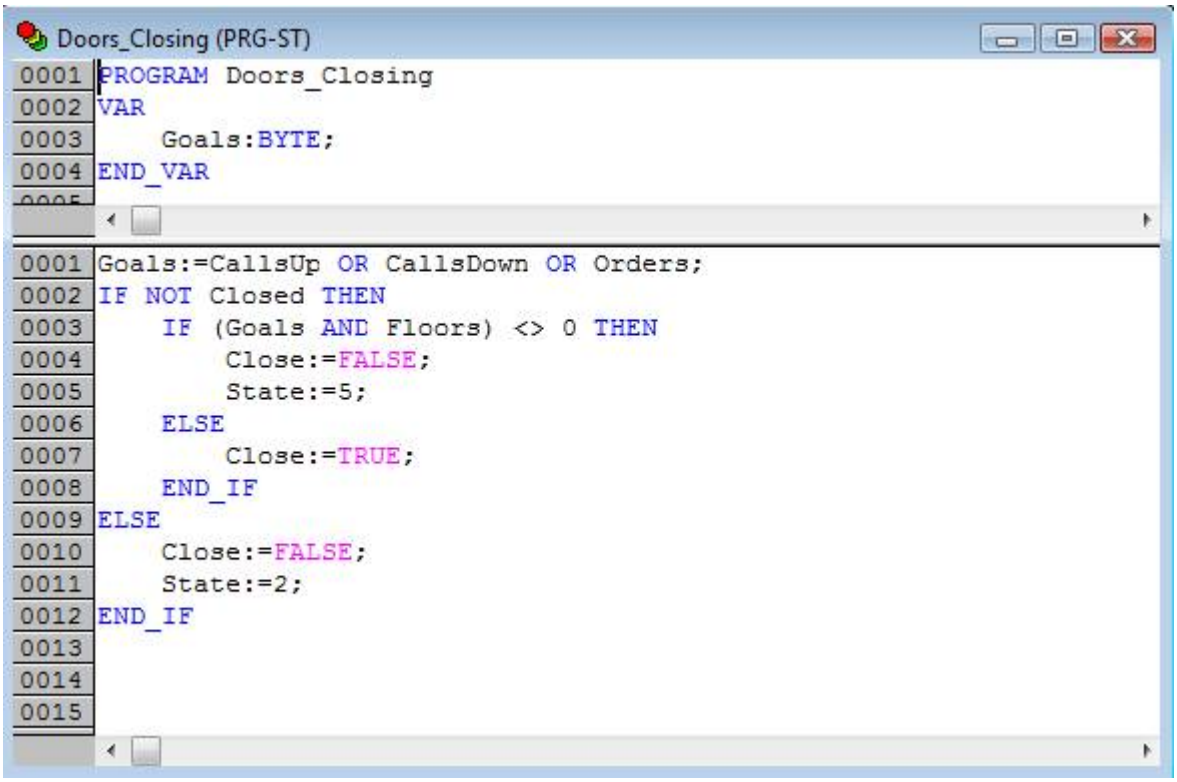


Рис. 3.194. Подпрограмма `Doors_Closing`.

3.2.3.5. Визуализация системы

Задачами визуализации являются отображение положения кабины и ее дверей и имитация этажных панелей вызовов и панели приказов в кабине.

Экран визуализации и необходимые каналы ввода-вывода созданы в системе Trace Mode 6.

На рис. 3.195 показано создание каналов Источники/Приемники посредством которых будет осуществляться обмен с OPC сервером.

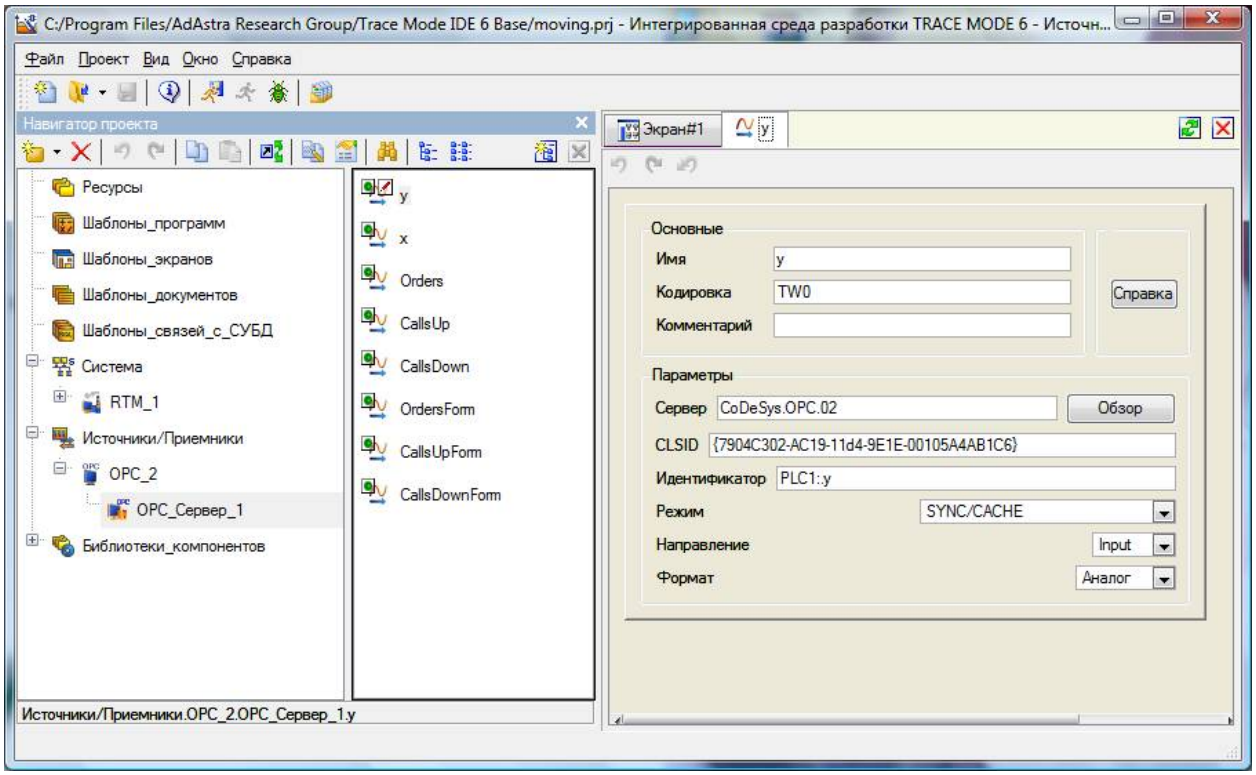


Рис. 3.195. Создание каналов Источники/Приемники.

В обмене с контроллером задействовано 8 переменных. Переменные `y`, `x`, `Orders`, `CallsUp`, `CallsDown` являются входными, `OrdersForm`, `CallsUpForm`, `CallsDownForm` – выходными.

Экран визуализации показан на рис. 3.196. Экран содержит схематичные изображения шахты с кабиной лифта, этажных панелей вызовов и панели приказов в кабине с кнопками и индикаторами.

Кабина перемещается в шахте, принимая положение, вычисленное имитационной моделью. Двери лифта «открываются» и «закрываются».

С помощью кнопок формируются вызовы и приказы. Индикаторы фиксируют еще неотработанные системой вызовы и приказы.

Аргументы экрана с привязками показаны на рис. 3.197.

Движение кабины вдоль шахты происходит под управлением аргумента экрана y_VALUE, привязанной к OPC переменной `y`.

На рис. 3.198 показано формирование траектории для левой половины дверей.

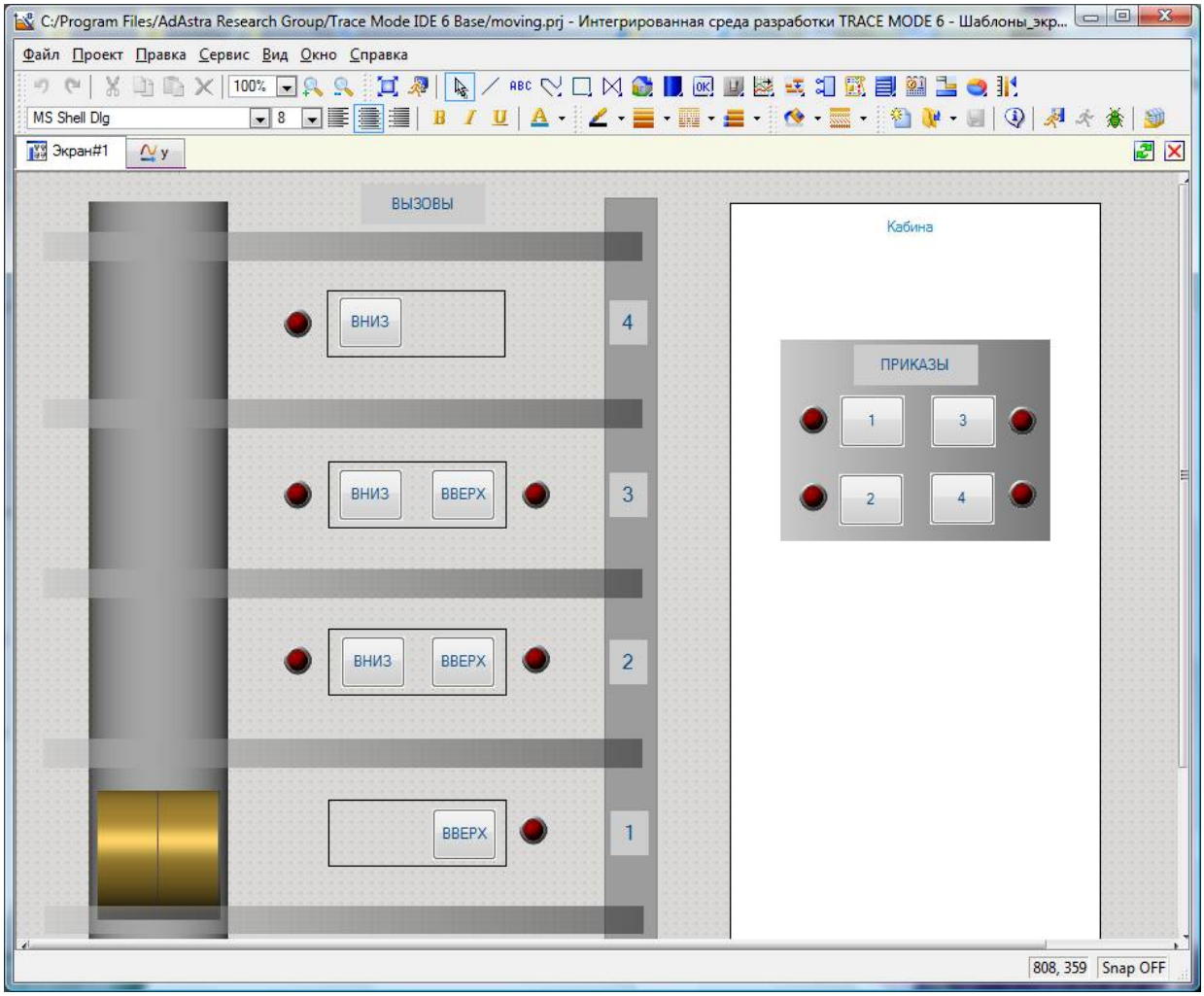


Рис. 3.196. Экран визуализации.

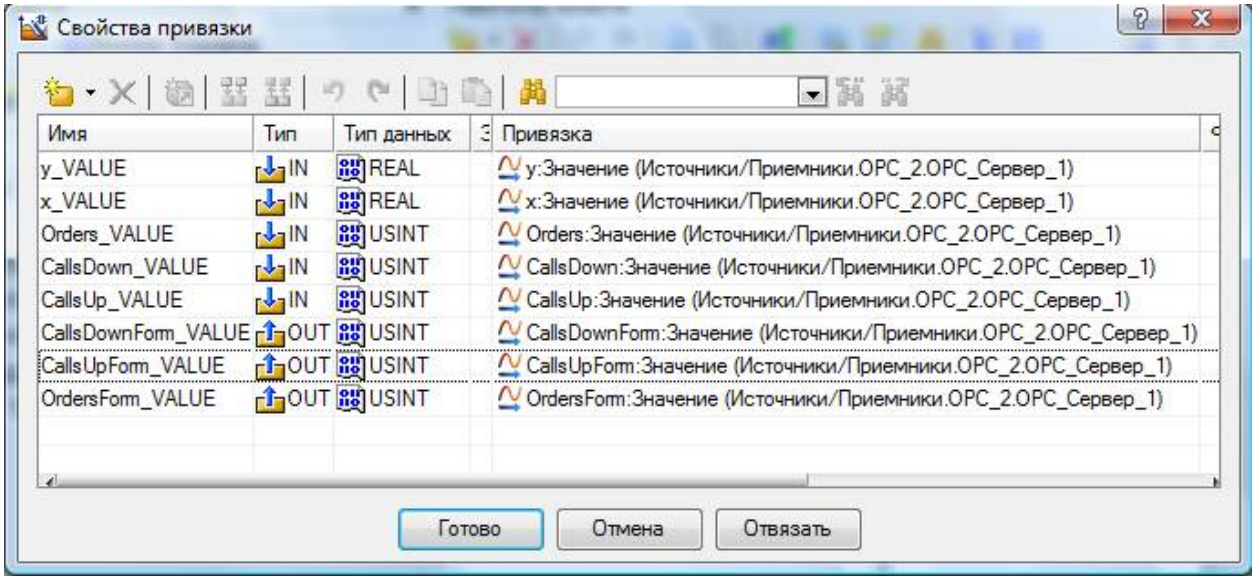


Рис. 3.197. Аргументы экрана.

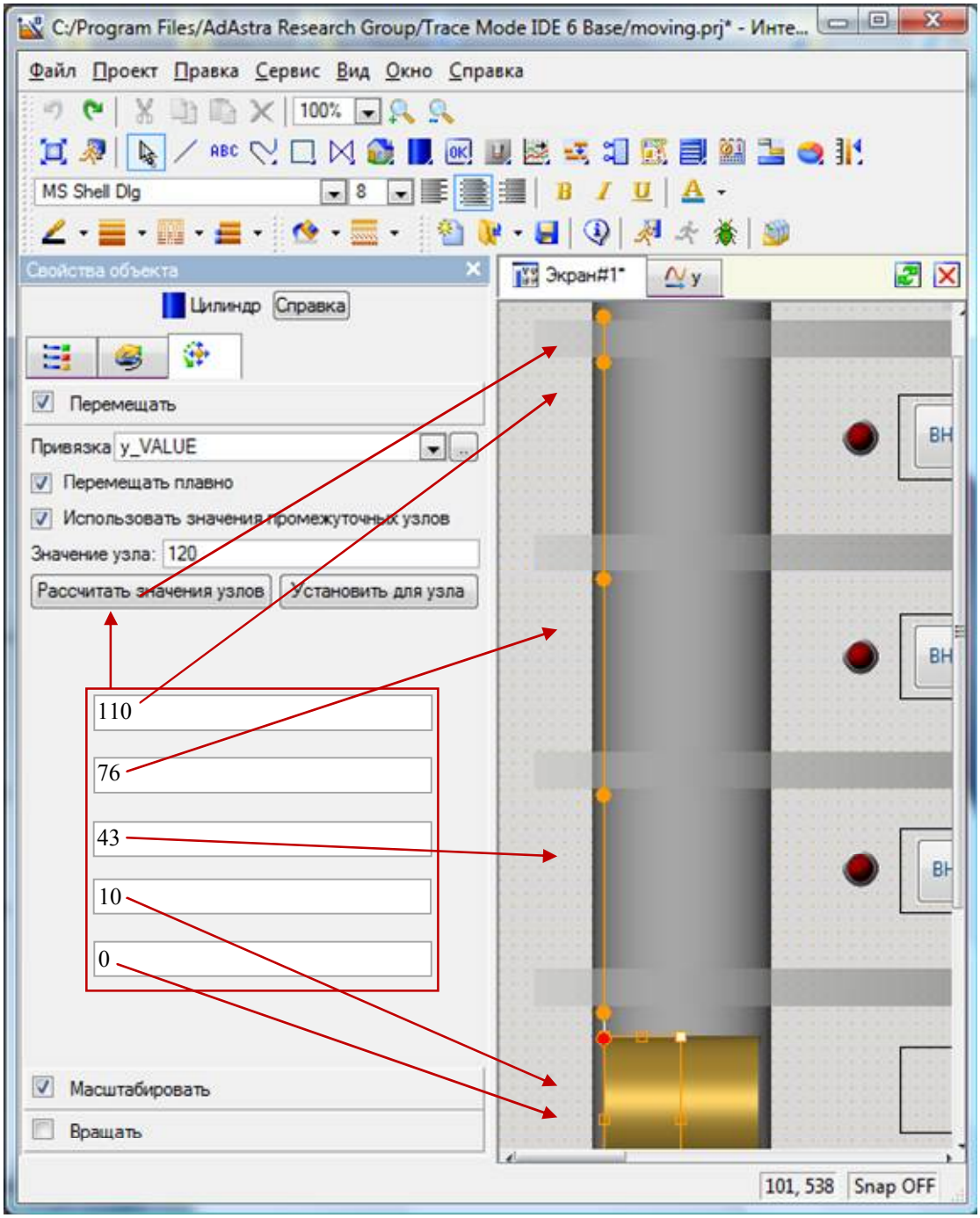


Рис. 3.198. Формирование траекторий движения левой половины дверей.

К сожалению, в Trace Mode 6 отсутствует возможность группирования графических элементов, поэтому траектории движения согласованно двигающихся объектов нужно задавать для каждого из них в отдельности. Практически проще сначала создать и полностью настроить один графический объект, потом сделать и отредактировать его копии. Именно таким способом и были сформированы правая половина дверей и рамка обрамления кабины (она становится видимой при открытых дверях).

Открытие дверей имитируется горизонтальным масштабированием «половинок» под управлением аргумента экрана `x_VALUE`, привязанной к OPC переменной `x`.

На рис. 3.199 показана настройка масштабирования для левой половины дверей.

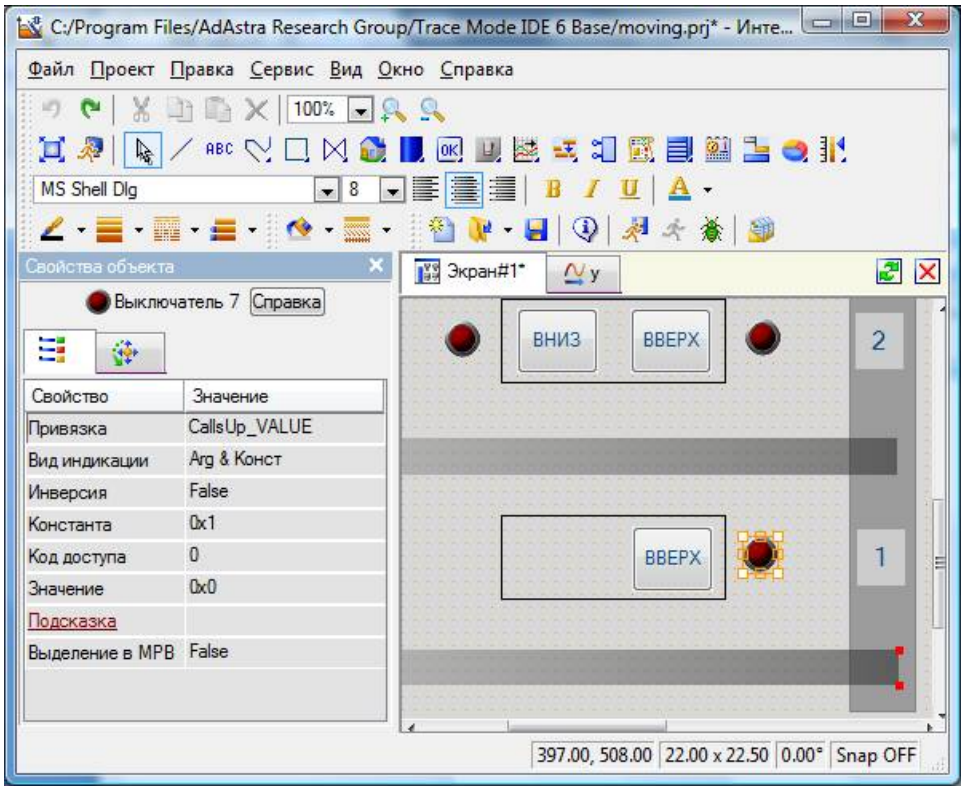


Рис. 3.199. Настройка «открытия» левой половины дверей.

Настройка «открытия» правой половины дверей аналогична за исключением положения центра масштабирования.

Настройка индикаторов неотработанных вызовов и приказов демонстрируется на рис. 3.200 на примере индикатора вызова вверх с первого этажа. Другие индикаторы настраиваются аналогично (табл. 3.2).

Таблица 3.2

Настройка индикаторов вызовов и приказов

Индикатор	Привязка	Константа
Вызов вверх 1 этаж	CallsUp_VALUE	0x1
Вызов вверх 2 этаж	CallsUp_VALUE	0x2
Вызов вверх 3 этаж	CallsUp_VALUE	0x4
Вызов вниз 2 этаж	CallsDown_VALUE	0x2
Вызов вниз 3 этаж	CallsDown_VALUE	0x4
Вызов вниз 4 этаж	CallsDown_VALUE	0x8
Приказ 1 этаж	Orders_VALUE	0x1
Приказ 2 этаж	Orders_VALUE	0x2
Приказ 3 этаж	Orders_VALUE	0x4
Приказ 4 этаж	Orders_VALUE	0x8

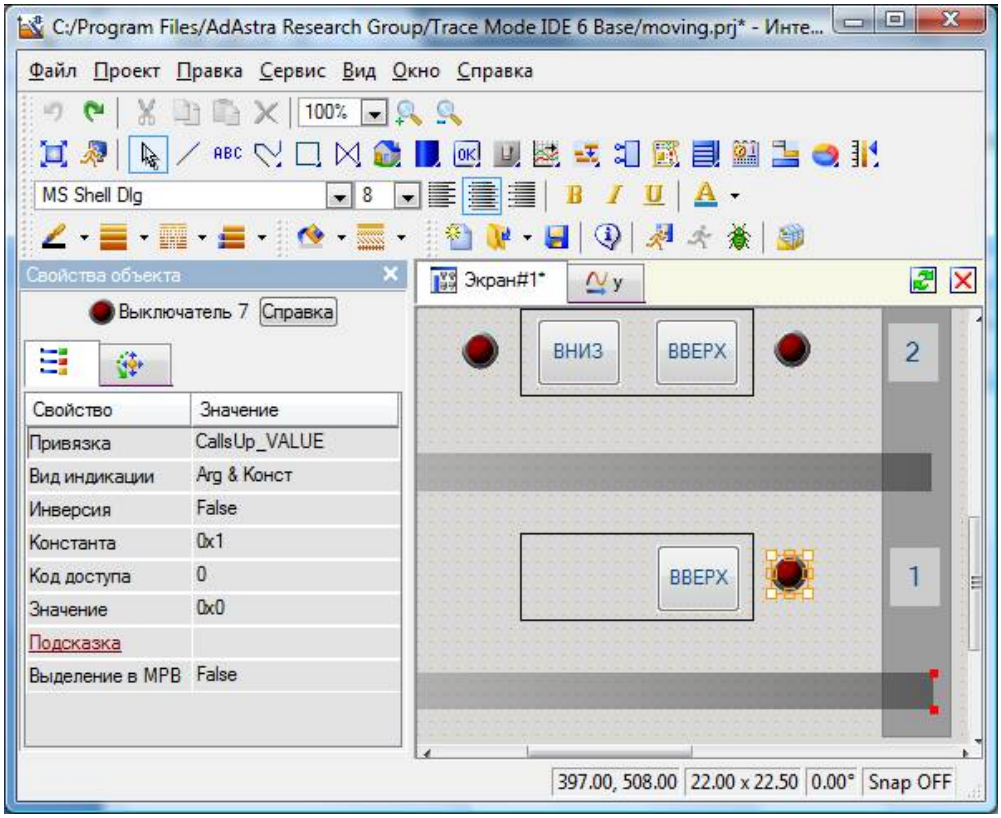


Рис. 3.200. Настройка индикатора вызова вверх с первого этажа.

Настройка кнопок вызовов и приказов демонстрируется на рис. 3.201 на примере кнопки вызова вверх с первого этажа.

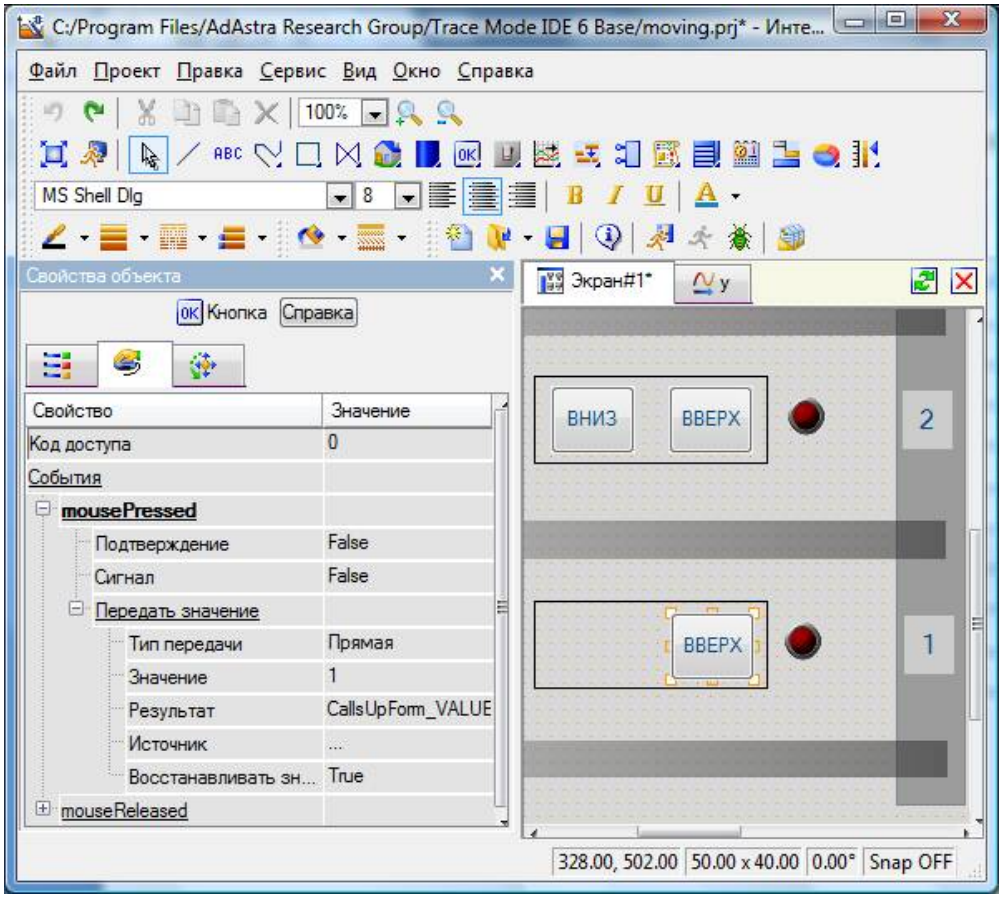


Рис. 3.201. Настройка кнопки вызова вверх с первого этажа.

Другие кнопки настраиваются аналогично (табл. 3.3).

Таблица 3.3

Настройка кнопок вызовов и приказов

Кнопка	Привязка	Значение
Вызов вверх 1 этаж	CallsUpForm_VALUE	1
Вызов вверх 2 этаж	CallsUpForm_VALUE	2
Вызов вверх 3 этаж	CallsUpForm_VALUE	4
Вызов вниз 2 этаж	CallsDownForm_VALUE	2
Вызов вниз 3 этаж	CallsDownForm_VALUE	4
Вызов вниз 4 этаж	CallsDownForm_VALUE	8
Приказ 1 этаж	OrdersForm_VALUE	1
Приказ 2 этаж	OrdersForm_VALUE	2
Приказ 3 этаж	OrdersForm_VALUE	4
Приказ 4 этаж	OrdersForm_VALUE	8

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Безверхов, М. Архив статей «Что такое «технология COM». Клуб программистов DEVELOPING. – Режим доступа: https://www.developing.ru/com/ index.ru.

2. GUID. Материал из Википедии – свободной энциклопедии. – Режим доступа: https://ru.wikipedia.org/wiki/GUID.

3. Руководство пользователя по программированию ПЛК в CoDeSys 2.3. – Режим доступа: https://www.owen.ru/product/ codesys_v2 /768

4. User documentation: The CoDeSys Gateway Server Manual. Document Version 2.0. –Режим доступа: http://www.5ichai.com/pdf/ Gateway%20Manual.pdf

5. CoDeSys OPC-Server V2.0 Установка и использование. Версия документа 1.8. – Режим доступа: https://forum-ru.codesys.com/ download/ file.php?id= 129

6. Использование OPC-сервера 3S Software для подключения контроллеров системы CoDeSys к компьютеру. Методическое пособие. – Режим доступа: https://forum-ru.codesys.com/download/file.php?id=130

7. Солодов, А.В., Петров, Ф.С. Линейные автоматические системы с переменными параметрами. – М.: Наука, 1974.

8. Методы классической и современной теории автоматического управления: учебник. – В 5 т. – Изд. 2-е, перераб. и доп. – Т. 1. Математические модели, динамические характеристики и анализ систем автоматического управления / под ред. К.А. Пупкова, Н.Д. Егупова. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 656 с., ил.

9. Там же. – Т. 3. Синтез регуляторов систем автоматического управления / под ред. К.А. Пупкова, Н.Д. Егупова. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. –616 с; ил.

10. Денисенко, В. ПИД-регуляторы: вопросы реализации // Современные технологии автоматизации. – 2008. – № 1. – С. 86-99.

11. Рыбалев, А.Н., Усенко, В.И, Русинов, В.Л. Теория автоматического управления. Часть 4. Импульсные, цифровые и нелинейные САР. Учебное пособие. – Благовещенск: Амурский гос. ун-т, 2017.

12. Литюга, А.М., Клиначёв, Н.В., Мазуров, В.М. Теоретические основы построения эффективных АСУ ТП. – Режим доступа: http://www.studmed.ru/ lityuga-am-klinachev-nv-mazurov-vm-teoreticheskie-osnovy-postroeniya-effektiv-nyh-asu-tp_ee882a7cc92.html

13. Спасибов, В.М. Идентификация промышленных объектов и систем управления: учебное пособие. Ч. 1 / В. М. Спасибов, И.А. Каменских, Ю.А. Ведерникова. – Тюмень: ТюмГНГУ, 2010. – 104 с.

14. Рыбалев, А.Н.. Теория автоматического управления. Пособие к курсовому проектированию. Учебное пособие. – Благовещенск: Амурский гос. ун-т, 2004. – 145 с.

15. Магергут, В.З., Вент, Д.П., Кацер, И.А. Инженерные методы выбора и расчета оптимальных настроек промышленных регуляторов. – Новомосковск: НФ РХТУ им. Д.И. Менделеева, 1994. – 158 с.

16. Бармин, А. Устройства локальной автоматики. Микроконтроллеры // Современные технологии автоматизации. – 2003. – № 4. – С. 38-42.

17. Лупал, А.М. Теория автоматов: учеб. пособие. – СПб.: СПбГУАП, 2000. – 119 с.

18. Бобриков, С.А. Математические основы теории систем: конспект лекций для студентов специальности 7.091401. – Режим доступа: https:// studfiles.net/preview/718684

19. Еремин, Е.Л., Теличенко, Д.А. Адаптивное и робастное управление объектами теплоэнергетики. – Благовещенск: Амурский гос. ун-т, 2009. – 228 с.

20. Еремин, Е.Л., Теличенко, Д.А., Семичевская, Н.П., Чепак Л.В, Шеленок Е.А. Управление техническими системами в условиях неопределенности. – Благовещенск: Амурский гос. ун-т, 2014. – 211 с.

21. Smith, O.J.M. Close control of loops with dead time // Chemical Engineering Progress. – 1957. – Vol. 53. – P. 217–235.

22. Денисенко, В.В. ПИД-регуляторы: принципы построения и модификации. Часть 2// Современные технологии автоматизации. – 2007. – № 1. – С. 78 – 88.

Адрес редакции и издателя:
675027, г. Благовещенск, Игнатьевское шоссе, 21, комн. 406.

Адрес типографии:
675000, г. Благовещенск, ул. Мухина, 150а

Адрес учредителя:
ФБГОУ ВО «Амурский государственный университет»
675027, г. Благовещенск, Игнатьевское шоссе, 21.

Андрей Николаевич Рыбалев,
доцент кафедры АППиЭ АмГУ, канд. техн. наук.

Имитационное моделирование АСУ ТП. *Монография.*

Издательство АмГУ. Подписано к печати __.09.2019. Дата выхода журнала в свет: __.09.2019. Редактор – *О.К. Мамонтова.* Компьютерная верстка – *Л.М. Пейзель*. Формат 60 х 84/16. Усл. печ. л. 23,72. Тираж __. Заказ ___. Бесплатно.

Отпечатано в типографии АмГУ.