

**Министерство образования Республики Беларусь**

**Учреждение образования  
«Гомельский государственный технический  
университет имени П. О. Сухого»**

**Кафедра «Промышленная электроника»**

**Д. А. Литвинов, А. В. Ковалев**

**ПРОГРАММИРОВАНИЕ ПРОМЫШЛЕННЫХ  
МИКРОКОНТРОЛЛЕРОВ НА ЯЗЫКАХ  
СТАНДАРТА IEC 61131-3**

**ПРАКТИКУМ**

**по дисциплине «Локальные информационные системы»  
для студентов специальности 1-36 04 02  
«Промышленная электроника»  
специализации 1-36 04 02 02  
«Техника и средства электронной связи»  
дневной формы обучения**

**Часть 2**

**Гомель 2016**

УДК 004(075.8)  
ББК 32.965я73  
Л64

*Рекомендовано научно-методическим советом  
факультета автоматизированных систем ГГТУ им. П. О. Сухого  
(протокол № 10 от 25.05.2015 г.)*

Рецензент: зав. каф. «Информационные технологии» ГГТУ им. П. О. Сухого  
канд. техн. наук, доц. *К. С. Курочка*

**Литвинов, Д. А.**

Л64

Программирование промышленных микроконтроллеров на языках стандарта IEC 61131-3 : практикум по дисциплине «Локальные информационные системы» для студентов специальности 1-36 04 02 «Промышленная электроника» специализации 1-36 04 02 02 «Техника и средства электронной связи» днев. формы обучения. Ч. 2 / Д. А. Литвинов, А. В. Ковалев. – Гомель : ГГТУ им. П. О. Сухого, 2016. – 66 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: <http://elib.gstu.by>. – Загл. с титул. экрана.

Посвящен ознакомлению с языками программирования стандарта IEC 61131-3. Дается общее описание языков программирования промышленных контроллеров. Рассматривается процесс разработки программ в SCADA-системе TRACE MODE с использованием наиболее распространенных языков Техно FBD и Техно ST.

Для студентов специальности 1-36 04 02 «Промышленная электроника» специализации 1-36 04 02 02 «Техника и средства электронной связи» дневной формы обучения.

**УДК 004(075.8)  
ББК 32.965я73**

© Учреждение образования «Гомельский  
государственный технический университет  
имени П. О. Сухого», 2016

## СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	3
ВВЕДЕНИЕ	4
Языки программирования ПЛК. Стандарт МЭК 61131-3	4
Средства разработки программного обеспечения на языках МЭК 61131-3. SOFTLOGIC	6
Лабораторная работа №1	11
Лабораторная работа №2	31
Лабораторная работа № 3	45
Список литературы	65

## **ВВЕДЕНИЕ**

### **Языки программирования ПЛК. Стандарт МЭК 61131-3**

Стандарт МЭК 61131-3 устанавливает пять языков программирования ПЛК, три графических и два текстовых. Основной целью стандарта было повышение скорости и качества разработки программ для ПЛК, а также создание языков программирования, ориентированных на технологов. Системы программирования, основанные на МЭК 61131-3, характеризуются следующими показателями:

- надежность создаваемого программного обеспечения. Надежность обеспечивается специализированной средой разработки, которая содержит все необходимые средства для написания, тестирования и отладки программ с помощью эмуляторов и реальных ПЛК, а также библиотеки готовых компонентов;
- простота языка и модификации программы;
- переносимость проекта с одного ПЛК на другой;
- возможность повторного использования отработанных фрагментов программы.

Языки МЭК 61131-3 появились как результат анализа множества языков, уже используемых на практике и предлагаемых рынку производителями ПЛК. Стандарт устанавливает пять языков программирования со следующими названиями:

- структурированный текст (ST - Structured Text);
- последовательные функциональные схемы (SFC - "Sequential Function Chart");
- диаграммы функциональных блоков (FBD - Function Block Diagram);
- релейно-контактные схемы, или релейные диаграммы (LD - Ladder Diagram);
- список инструкций (IL - Instruction List).

Графическими языками являются SFC, FBD, LD. Языки IL и ST являются текстовыми.

Выбор одного из пяти языков определяется не только предпочтениями пользователя, но и смыслом решаемой задачи. Программисты чаще выбирают язык IL (похожий на ассемблер) или ST, похожий на язык высокого уровня Паскаль; специалисты, имеющие опыт работы с релейной логикой, выбирают язык LD, специалисты по системам ав-

томатического управления (САУ) и схемотехники выбирают привычный для них язык FBD.

Языки МЭК 61131-3 базируются на следующих принципах:

- вся программа разбивается на множество функциональных элементов - Program Organization Units (POU);
- строгая типизация данных. Указание типов данных позволяет легко обнаруживать большинство ошибок в программе до ее исполнения;
- средства для исполнения разных фрагментов программы в разное время, с разной скоростью, а также параллельно. Например, один фрагмент программы может сканировать концевой датчик с частотой 100 раз в секунду, в то время как второй фрагмент будет сканировать датчик температуры с частотой один раз в 10 сек;
- для выполнения операций в определенной последовательности, которая задается **моментами времени или событиями**, используется специальный язык последовательных функциональных схем (SFC);
- стандарт поддерживает структуры для описания разнородных данных. Например, температуру подшипников насоса, давление и состояние "включено-выключено" можно описать с помощью единой структуры "Poup" и передавать ее внутри программы как единый элемент данных;
- стандарт обеспечивает совместное использование всех пяти языков, поэтому для каждого фрагмента задачи может быть выбран любой, наиболее удобный, язык;
- программа, написанная для одного контроллера, может быть перенесена на любой контроллер, совместимый со стандартом МЭК 61131-3.

Любой ПЛК работает в циклическом режиме. Цикл начинается со сбора данных с модулей ввода, затем исполняется программа ПЛК и оканчивается цикл выводом данных в устройства вывода. Поэтому величина контроллерного цикла зависит от времени исполнения программы и быстродействия процессорного модуля.

Цели создания МЭК 61131-3 в полном объеме достигнуты не были поскольку, каждый производитель ПЛК сопровождает свой продукт собственной средой программирования, которая, как правило, не совместима с другими, да и о кросс-платформенности программного кода можно забыть. Тем не менее, в части описания языков програм-

мирования стандарт IEC 61131 остается чрезвычайно актуальным и является ориентиром для большинства разработчиков ПЛК.

### **Средства разработки программного обеспечения на языках МЭК 61131-3. SOFTLOGIC**

SOFTLOGIC – это система программирования промышленных контроллеров интегрируемая со SCADA/HMI. Программирование ПЛК на описанных выше языках МЭК 61131-3 осуществляется с помощью специализированного программного обеспечения, которое разрабатывается производителями ПЛК или фирмами, специализирующимися на создании ПО для систем автоматизации. Наиболее известными в мире являются системы CoDeSys фирмы 3S, ISaGRAF фирмы ICS Triplex, SIMATIC STEP 7 фирмы Siemens, и др. Среди отечественных разработок TRACE MODE компании AdAstra Research Group, MasterSCADA компанией ИнСАТ (InSAT) и др.

#### **CoDeSys**

CoDeSys (**C**ontroller **D**evelopment **S**ystem) – система программирования контроллеров и встроенных систем стандарта технологических языков программирования IEC 61131-3. Этот комплекс не является привязанным, к какой либо аппаратной платформе. Более 250 компаний по всему миру поддерживают программирование в среде CoDeSys. Его главными особенностями являются:

- Прямая генерация машинного кода – классическим компилятором, что обеспечивает высокое быстродействие;
- Полноценная + расширенная реализация всех технологических языков программирования стандарта IEC 61131;
- Встроенный симулятор контроллера CoDeSys позволяет совершать отладку проекта без аппаратных средств. При этом эмулируется конкретный ПЛК с учетом аппаратных особенностей. Также возможна отладка и реального контроллера в режиме on-line;
- Встроенные инструменты визуализации CoDeSys. В большинстве простых случаев нет необходимости покупать SCADA программу.

Основными частями системы являются среда разработки программы и среда ее исполнения (CoDeSys SP), которая находится в

ПЛК. Она выполняет загрузку кода в память процессора, управление задачами, функции мониторинга, просмотр и фиксацию переменных, аккумулярование данных трассировки и трендов, содержит оптимизированный код стандартных библиотек и т.д. Большая по объему часть кода системы исполнения работает только при подготовке программы. Ядро, управляющее прикладными задачами, исключительно компактно. В совокупности с компилятором это обеспечивает высокое быстродействие прикладного ПО в CoDeSys.

## **SIMATIC STEP**

Simatic Step 7 — программное обеспечение фирмы Siemens для разработки систем автоматизации на основе программируемых логических контроллеров Simatic S7-300/S7-400/M7/C7 и систем компьютерного управления SIMATIC WinAC. Промышленное программное обеспечение SIMATIC содержат полный набор средств, необходимых для всех этапов разработки и эксплуатации систем автоматического управления.

Инструментальные средства проектирования включают:

- Языки программирования высокого уровня.
- Графические языки для специалистов в области технологии.
- Сопутствующее программное обеспечение для диагностирования, имитации, дистанционного обслуживания, разработки заводской документации и т.д.

С помощью этой программы выполняется комплекс работ по созданию и обслуживанию систем автоматизации. В первую очередь это работы по программированию контроллеров. Программирование контроллеров производится редактором программ, обеспечивающим написание программ на трех языках:

- LAD — язык релейно-контактной логики;
- FBD — язык функциональных блочных диаграмм;
- STL — язык списка инструкций.

В дополнение к трем основным языкам могут быть добавлены четыре дополнительных языка, поставляемые отдельно:

- SCL — структурированный язык управления, по синтаксису близкий к Pascal;
- GRAPH 7 — язык управления последовательными технологическими процессами;

- HiGraph 7 — язык управления на основе графа состояний системы;
- SFC — язык диаграмм состояния.

В проект Step 7 могут быть, включены системы человеко-машинного интерфейса – операторские панели, конфигурируемые с помощью производимого Siemens программного обеспечения ProTool, WinCC Flexible или персональный компьютер с программным обеспечением WinCC.

## **ISaGRAF**

ISaGRAF – это встраиваемая, масштабируемая технология программирования контроллеров, позволяющая создавать как приложения для автономных контроллеров, так и распределенные приложения для нескольких обменивающихся данными по сети контроллеров.

Интегрированная среда разработки программ для программируемых логических контроллеров на языках стандарта IEC 61131-3 и IEC 61499, который позволяет создавать локальные или распределенные системы управления. Основа технологии — среда разработки приложений (ISaGRAF Workbench) и адаптируемая под различные аппаратно-программные платформы исполнительная система (ISaGRAF Runtime). В ISaGRAF поддерживаются все пять языков стандарта IEC 61131-3.

## **TRACE MODE**

TRACE MODE® 6 предоставляет широкий набор средств программирования задач АСУТП и бизнес-приложений (АСУП), ориентированный на специалистов разной квалификации и профессиональной подготовки. В систему TRACE MODE® 6 включены 5 языков программирования – Techno SFC, Techno LD, Techno FBD, Techno ST, и Techno IL, которые являются расширением языков стандарта IEC 61131-3.

Для всех 5 языков существует единый механизм связи с базой данных реального времени TRACE MODE® 6. Каждая программа обладает набором аргументов, исходные данные передаются в программу через входные аргументы, а результаты вычислений возвращаются в выходных аргументах. Аргументы связываются с атрибутами кана-



лов TRACE MODE 6, т.е. с реальными входами и выходами контроллеров и УСО, ячейками баз данных, либо с внутренними переменными. Таким образом, одна и та же программа может вызываться несколько раз за цикл для обработки разных потоков данных.

Программирование контроллеров осуществляется в инструментальной системе, откуда SOFTLOGIC-программа копируется в память контроллера, а ее выполнение обеспечивается исполнительными модулями TRACE MODE - Микро MPB или Embedded MPB или др.

Программирование и отладка программ на языках МЭК 6-1131/3 в TRACE MODE 6 производится в интегрированной среде разработки, включающей в себя редакторы для каждого языка программирования. Программы на разных языках стандарта МЭК 6-1131/3 в TRACE MODE® 6 могут взаимодействовать между собой. Например, программа на Techno FBD может вызывать функциональный блок, написанный на языке Techno ST, а внутри этого блока может вызываться подпрограмма на Techno LD и т.д.

Система Trace Mode поддерживает модифицированные языки стандарта IEC6113-3:

1. Techno ST (Structured Text);
2. Techno SFC (Sequential Function Chart);
3. Techno FBD (Function Block Diagram);
4. Techno LD (Ladder Diagram);
5. Techno IL (Instruction List).

Для обеспечения обмена данными между программой и каналами узла служат аргументы программы. Тип каждого аргумента определяет направление передачи данных. Так если аргумент служит для передачи значения в программу, следует выбрать тип IN, если для передачи из программы – OUT. В ряде случаев нужен аргумент, который будет передавать данный как в программу, так и из нее, тогда следует выбрать тип IN/OUT. Для обмена данными программой необходимо произвести привязку каждого аргумента программы к тому или иному каналу (аргументу канала). Помимо аргументов можно создать локальные и глобальные переменные.

Этапы разработки шаблона программ в пакете TRACE MODE:

- создание шаблона программы;

1. создание аргументов шаблона программы;
2. выбор языка программирования и создание программы на выбранном языке программирования;
3. компиляция и отладка программы.

Данные этапы могут выполняться в любой последовательности.  
Также в любой момент можно вернуться к любому этапу.

Библиотека ГГТУ им. П.О.Сухого

# Лабораторная работа №1

## Реализация логического управления в SCADA TRACE MODE 6 с использованием языка Техно FBD

**Цель работы:** освоить методику программирования логических функций при помощи SCADA–системы TRACE MODE на языке Техно FBD.

### 1. Теоретические сведения

Язык FBD (Functional Block Diagram, диаграмма функциональных блоков) является графическим языком и наиболее удобен для программирования процессов прохождения сигналов через функциональные блоки. Программа на языке FBD представляет собой совокупность функциональных блоков (functional blocks, FBs), которые соединяются линиями связи (connections). Каждый блок представляет собой математическую операцию (сложение, умножение, триггер, логическое “или” и т.д.). Начальные значения переменных задаются с помощью специальных блоков – входов или констант, выходные цепи могут быть связаны либо с физическими выходами контроллера, либо с глобальными переменными программы.

Типичным применением языка FBD является описание "жесткой логики" и замкнутых контуров систем управления. Язык функциональных блоков является удобным средством при программировании задач промышленной автоматизации. Практика показывает, что FBD является наиболее распространенным языком стандарта IEC. Графическая форма представления алгоритма, простота в применении и повторном использовании, библиотеки функциональных блоков делают язык FBD незаменимым при разработке программного обеспечения ПЛК. К недостаткам можно отнести, неудобство (не очевидность) реализации релейной логики.

Дальнейшим развитием языка FBD, является CFC (Continuous Flow Chart). Язык был специально создан для проектирования систем управления непрерывными технологическими процессами. Проектирование сводится к выбору из библиотек готовых функциональных блоков, установке связей между ними, а также настройке параметров выбранных блоков. В отличие от FBD, функциональные блоки языка CFC выполняют не только простые математические операции, а ориентированы на управление целыми технологическими единицами. Так

в типовой библиотеке CFC блоков находятся комплексные функциональные блоки, реализующие управление клапанами, моторами, насосами; блоки, генерирующие аварийные сигнализации; блоки PID-регулирования и т.д. Вместе с тем доступны и стандартные блоки FBD. Унаследовав от FBD саму концепцию программирования, язык CFC в наибольшей степени ориентирован на сам технологический процесс, позволяя разработчику абстрагироваться от сложного математического аппарата.

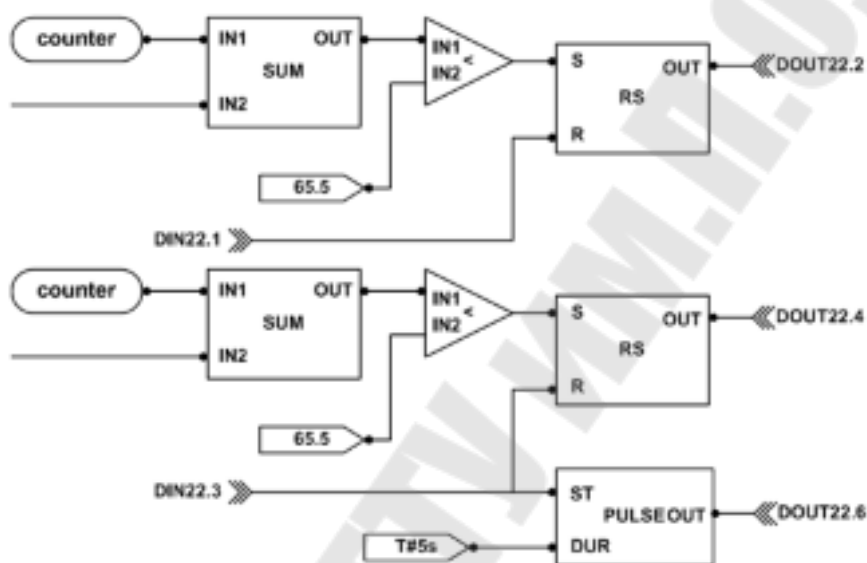


Рисунок 1.1 – Пример программы на языке FBD

**Функциональный блок** – это графическое изображение вызова встроенной функции Техно FBD (FBD-блока). В качестве примера рассмотрим функциональный блок, производящий сложение. Изображение его приведено на рисунке 1.2.

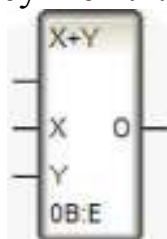



Рисунок 1.2 – Функциональный блок сложения на языке Техно ST

В верхней части блока указывается обозначение блока. Внизу выводится его номер. Номера блоком приписываются автоматически при их размещении в рабочем поле редактора программы. После

двоеточия указывается номер функционального блока, который будет выполняться следующим.

Горизонтальные линии, расположенные слева, выступают в качестве входов, на которые подается та или иная описанная локальная или глобальная переменная, аргумент программы, выходы с других функциональных блоков. На вход можно подать аргументы, тип которых **In** или **In/Out**. У каждого входа указывается его название. В указанном примере названия: X, Y. Безымянный вход, расположенный сверху, управляет выполнением блока: блок выполняет действие (в данном случае сложение) в том случае, если подается 0 или вход не подключен, в противном случае функциональный блок не выполняется.

Горизонтальная линия справа обозначает выход, содержащий результат выполнения функционального блока. Выход можно соединить с входом другого функционального блока. Выход функционального блока можно привязать к описанной глобальной или локальной переменной, аргументу, тип которого **Out** или **In/Out**.

Для размещения функционального блока следует открыть окно – «**FBD блоки**», для чего следует щелкнуть левой клавишей мыши по иконке  или выбрать – «палитра **FBD** блоков» в меню «Вид». Появится окно FBD блоки (рисунок 1.3).

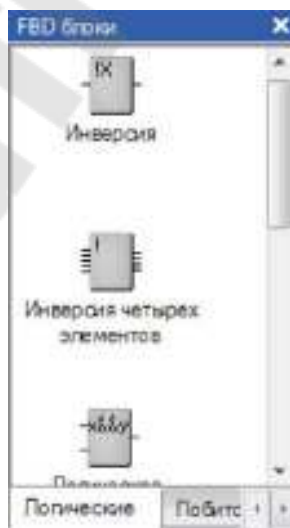


Рисунок 1.3 – Окно **FBD** блоки

В данном окне выбирается нужная закладка (логические, побитовые, арифметические) в соответствии с тематикой нужного блока. В окне отображается ряд блоков, относящихся к данной тематике. Среди этих блоков отыскивается нужный блок и перетаскивается на поле

редактора программы с использованием механизма drag-and-drop. То есть, курсор наводится на необходимый функциональный блок. Нажимается левая клавиша мыши. Блок «перетаскивается» на рабочее поле в нужное место при нажатой левой клавише мыши. Когда курсор переведен в положение, где должен располагаться функциональный блок левая клавиша отпускается.

Для соединения выхода одного функционального блока с входом другого следует навести курсор на выход функционального блока и нажать левую клавишу мыши. Блок станет синим, а имя выделенного выхода будет зеленым. Не отпуская клавиши мыши, наведем курсор на нужный вход функционального блока и отпустим клавишу мыши. Соответствующий вход и выход будут соединены линией, если все было правильно сделано. Аналогично можно нажать левую клавишу мыши, наведя на нужный вход функционального блока, и отпустить, наведя курсор на требуемый выход функционального блока.

Для привязки входа или выхода функционального блока следует выделить вход (выход) щелчком левой клавиши мыши. Появится контекстное меню, в котором следует выбрать – «**привязать**». Появится окно со списком аргументов и переменных, к которым можно привязать вход (выход) функционального блока. В этом списке следует выбрать переменную или аргумент (рисунок 1.4).



Рисунок 1.4 – Привязка входа (выхода) функционального блока

Создание константы рассмотрим на примере создания константы равной –  $1e-9$ . Для этого следует выделить вход, на который необходимо подать константу, вызвать контекстное меню и выбрать – **привязать**. Будет выведено окно, как указано на рисунке 1.4. В поя-

вившемся окне вместо выбора аргумента или переменной вводится значение константы. В данном случае вводится – «1e-9».

В качестве примера приведена реализация примера программы, написанной на языке Техно ST, используя язык Техно FBD (рисунок 1.5).

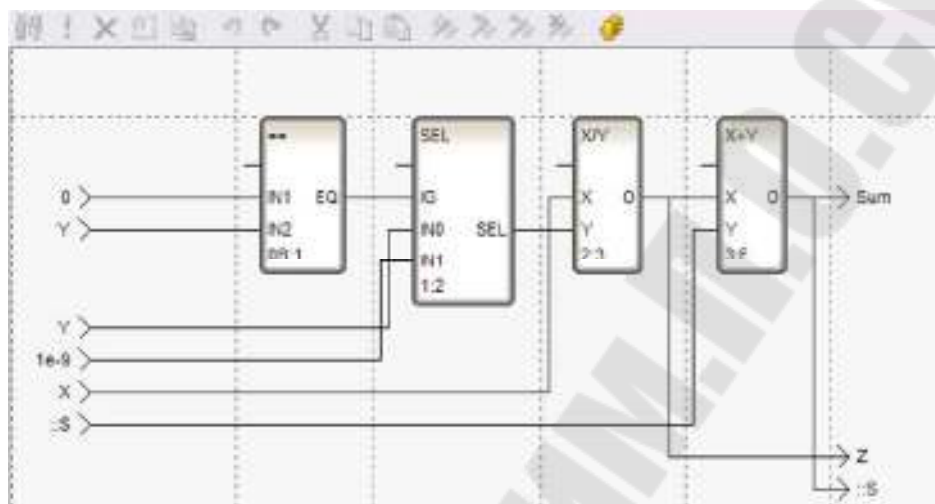


Рисунок 1.5 – Пример FBD программы

## 2. Ход работы

Для изучения средств разработки и основных элементов проекта, рассмотрим процесс его создания на примере.

### 2.1 Создание узла АРМ

Рассмотрим реализацию логической функции  $f = x_1x_2x_4 \cup x_1x_2 \cup x_1x_3 \cup x_2x_4$  при помощи SCADA-системы TRACE MODE на языке Техно FBD.

Разработка любого проекта автоматизации всегда начинается с запуска Интегрированной среды разработки (ИСР).

После запуска ИСР в меню «Файл» выбрать команду «Настройки ИС...». В появившемся окне в закладке «Уровень сложности» настроить проект как показано на рисунке 2.1, а затем выбрать закладку «Отладка» и настроить проект как показано на рисунке 2.2.



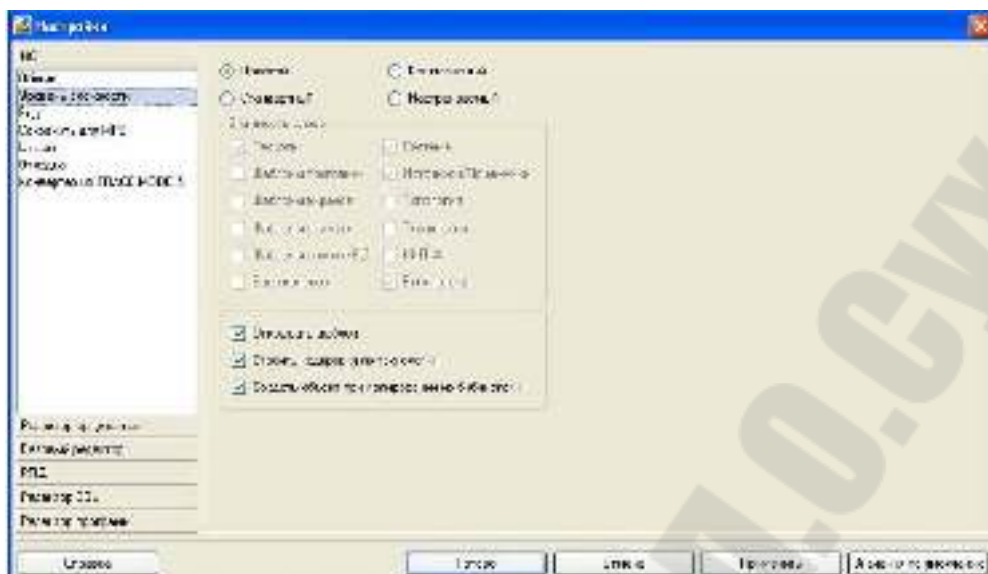



Рисунок 2.1 – Настройки «Уровень сложности»



Рисунок 2.2 – Настройки «Отладка»

После проведенных настроек ИСР нажать кнопку «**Готово**». С помощью иконки  инструментальной панели создадим новый проект при этом в открывшемся на экране диалоге (рисунок 2.3).



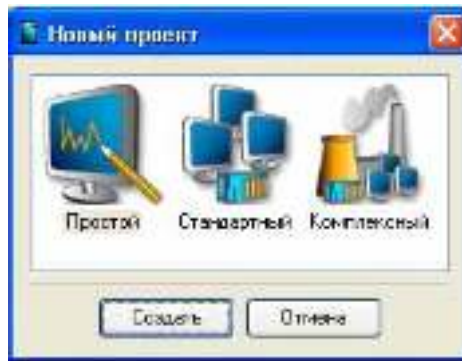


Рисунок 2.3 – Выбор типа проекта

Выберем «**Простой**» стиль разработки. После нажатия левой клавиши мыши (ЛК) на экранной кнопке «**Создать**», в левом окне Навигатора проекта появится дерево проекта с созданным узлом **АРМ RTM\_1**. Откроем узел **RTM\_1** двойным щелчком ЛК, в правом окне «**Навигатора проекта**» отобразится содержимое узла – пустая группа «**Каналы**» и один канал класса «**Вызов**» **Экран#1**, предназначенный для отображения на узле **АРМ** графического экрана (рисунок 2.4).

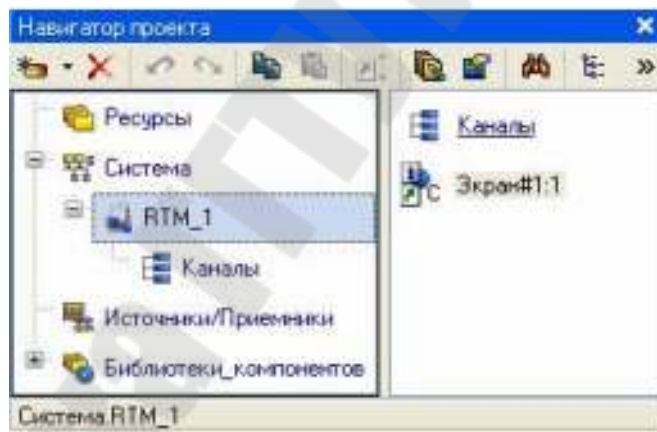



Рисунок 2.4 – Навигатор проекта

## 2.2 Создание графического интерфейса

Двойным щелчком ЛК по объекту **Экран#1** откроем окно графического редактора. Подготовим на экране вывод динамического текста для отображения численного значения входных переменных  $X_1$ ,  $X_2$ ,  $X_3$ ,  $X_4$  и выходного значения  $Y$  путем указания динамизации атрибута графического элемента (ГЭ). Определим назначение аргумента шаблона экрана. Создадим и разместим четыре ГЭ «**Текст**»  как показано на рисунке 2.5.

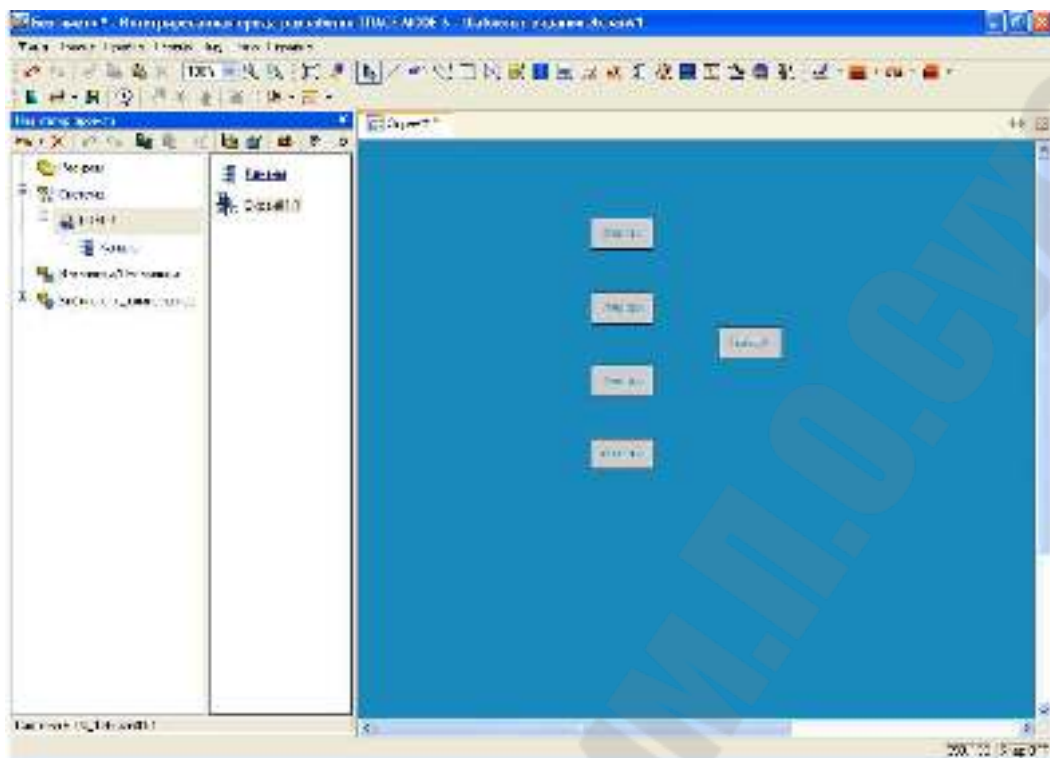


Рисунок 2.5 – Создание графических элементов

Откроем окно свойств ГЭ «Текст». Двойным щелчком ЛК на строке «Текст» вызовем меню «Вид индикации» (рисунок 2.6).

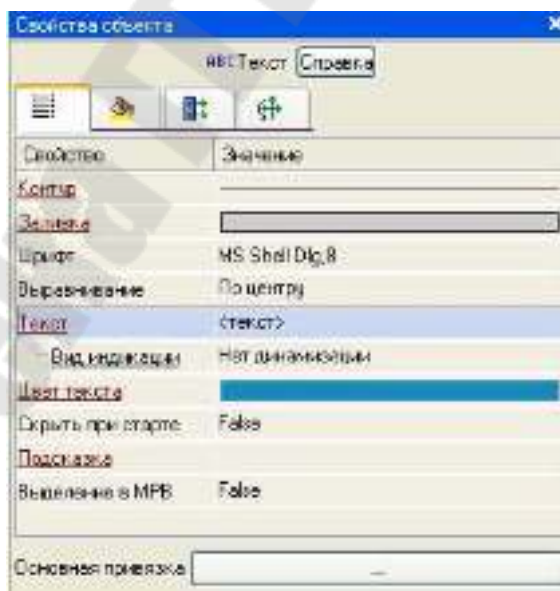


Рисунок 2.6 – Настройка индикации

В правом поле строки щелчком ЛК, вызвать список доступных типов. Выбрать тип **Значение** (рисунок 2.7).

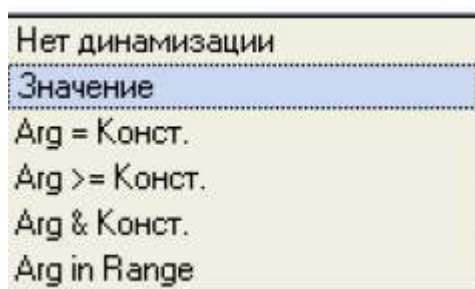



Рисунок 2.7 – Выбор типа динамизации

В открывшемся меню настройки параметров динамизации, выбрать свойство «Привязка» (рисунок 2.8).



Рисунок 2.8 – Окно привязки

В открывшемся окне «Свойства привязки», нажав кнопку  на его панели инструментов, создать пять аргументов экрана 4 входных и один выходной рисунок 2.9.

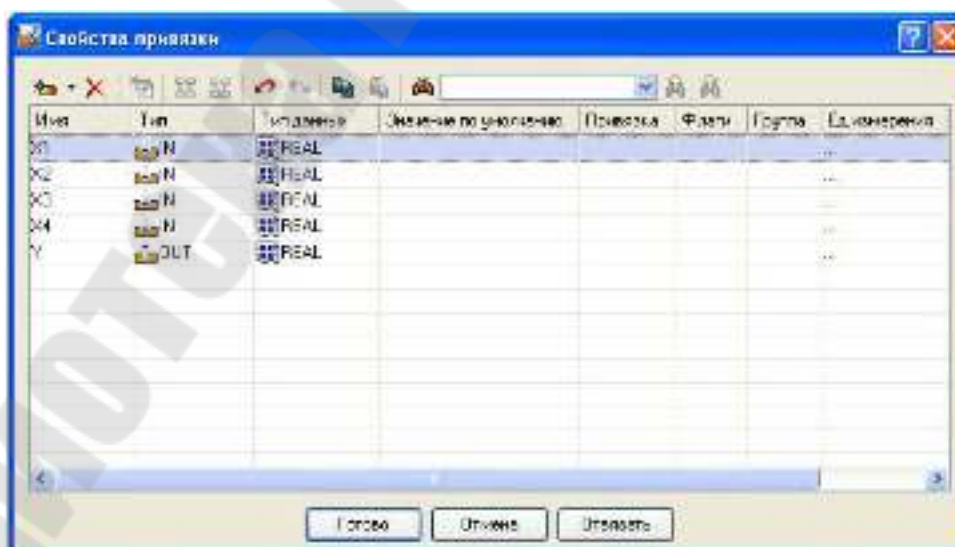



Рисунок 2.9 – Создание аргументов экрана

Двойным щелчком ЛК выделить имя аргумента и изменить его, введя с клавиатуры «X1» (завершить ввод нажатием клавиши Enter). Подтвердить связь с этим аргументом нажатием кнопки «Готово». Аналогичным образом настроить все **входные** и **выходные** аргументы.

Введем в состав графического экрана средство, позволяющее реализовать ввод числовых значений с клавиатуры. Создадим новый аргумент шаблона экрана для их приема.

Для этого, выбрать на инструментальной панели графического редактора иконку ГЭ Кнопка – . С помощью мыши разместить его в поле экрана как показано на рисунке 2.10.

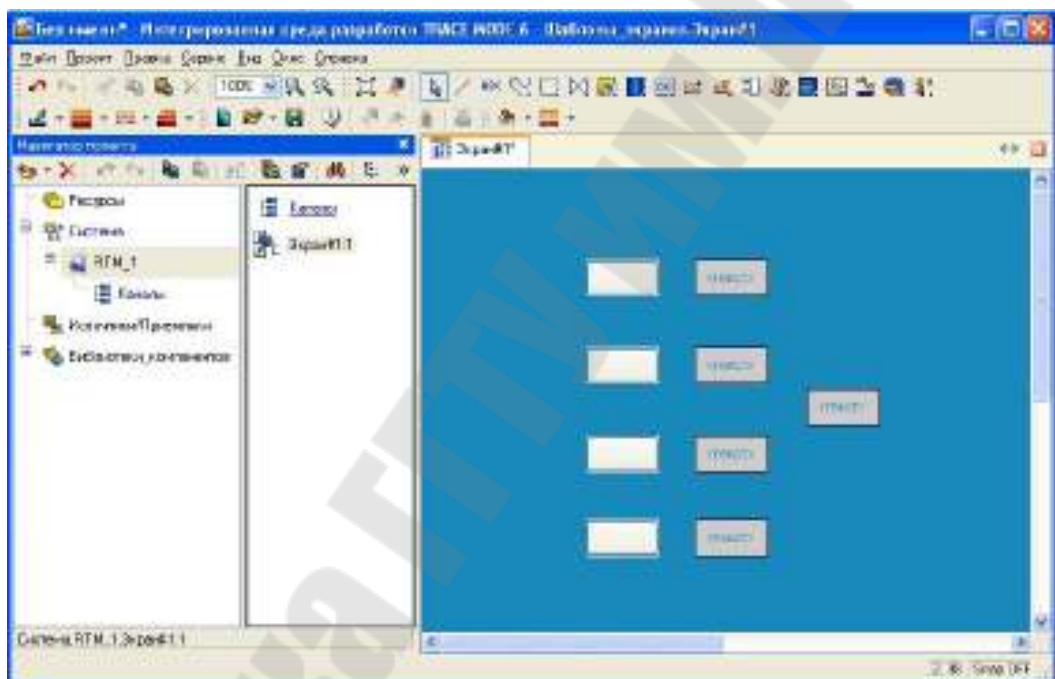



Рисунок 2.10 – Вид экрана

Для перехода в режим редактирования атрибутов размещенного ГЭ выделим ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ откроем окно его свойств (рисунок 2.11).

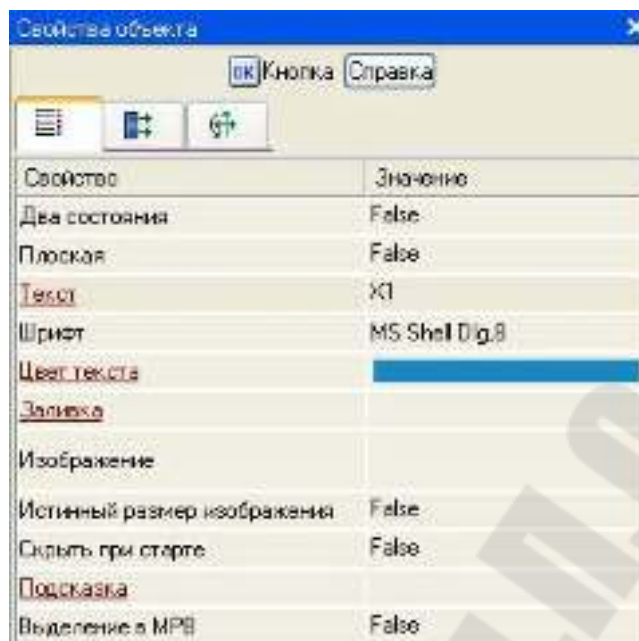


Рисунок 2.11 – Свойства графического элемента – «Кнопка»

В поле «Текст» ввести «X1». Откроем закладку «События» и раскроем меню «По нажатию» (mousePressed). Выберем из списка команду «Добавить Send Value» («Добавить Передать значение») (рисунок 2.12).

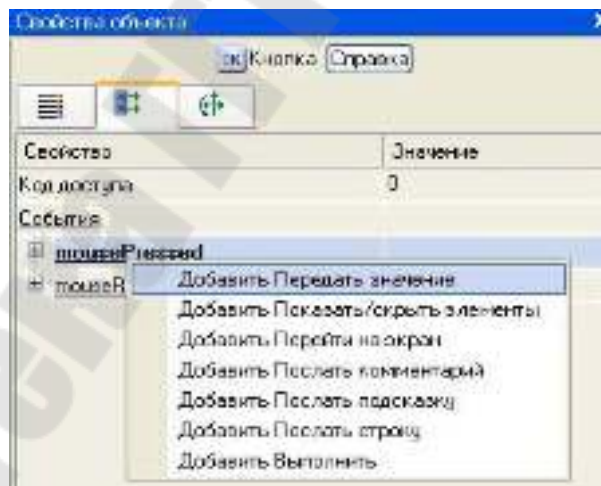


Рисунок 2.12 – Настройка типа передачи

В раскрывшемся меню настроек выбранной команды в поле «Тип передачи (Send Type)» выберем из списка «Ввести и передать (Enter & Send)» (рисунок 2.13).



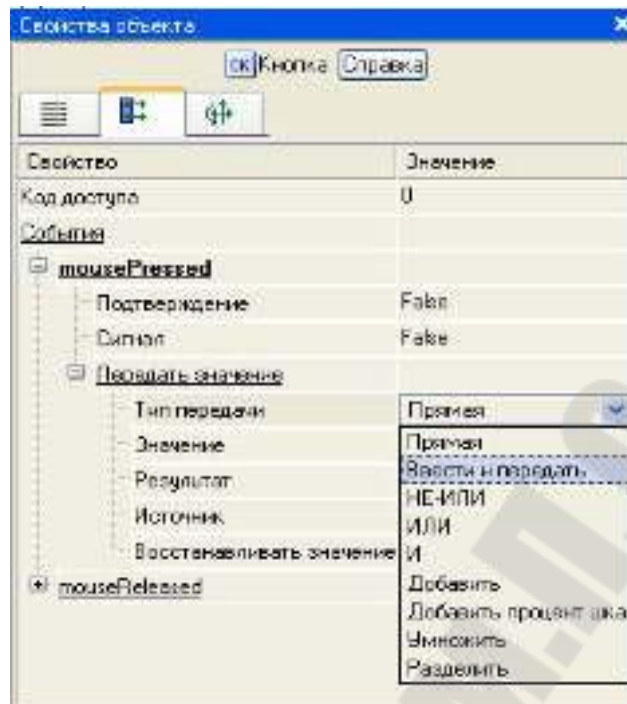


Рисунок 2.13 – Окно настройки событий элемента кнопка

Щелчком ЛК в поле «**Результат (Destination)**» вызовем табличный редактор аргументов и произведем привязку к аргументу **X1**. После привязки окно «**Свойства объекта**» выглядит следующим образом рисунок 2.14.

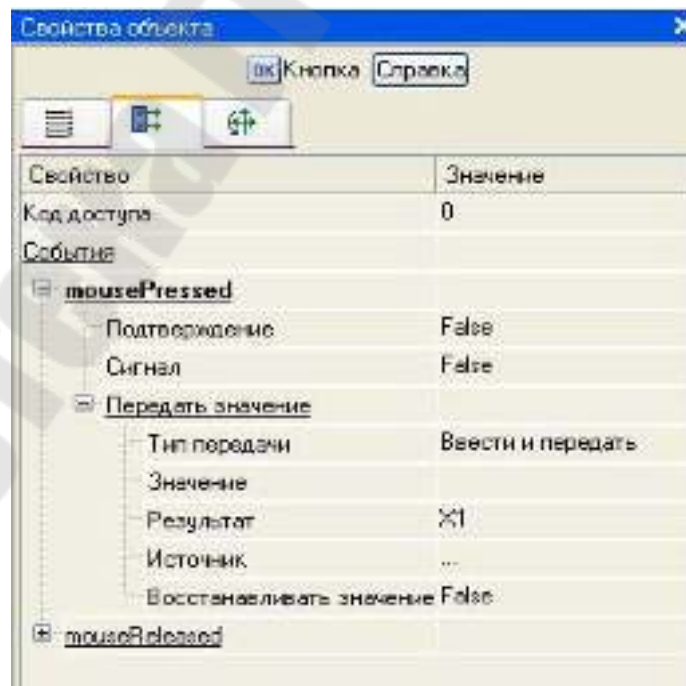



Рисунок 2.14 – Окончательный вид окна свойств графического элемента

Аналогичным образом настроить остальные кнопки **X2**, **X3**, **X4**.

### 2.3 Привязка аргумента экрана к каналу

Создадим по аргументам **X1**, **X2**, **X3**, **X4** и **Y** шаблона экрана новые каналы и отредактируем их привязку. В слое «Система» открыть узел **RTM\_1**. С помощью ПК мыши, вызвать через контекстное меню свойства компонента **Экран#1**.

Выбрать вкладку «Аргументы», ЛК мыши выделить аргументы **X1**, **X2**, **X3**, **X4** и **Y** и с помощью иконки  создать новый канал. В результате, в узле **RTM\_1** будут автопостроены следующие каналы **X1**, **X2**, **X3**, **X4** и **Y** (рисунок 2.15).

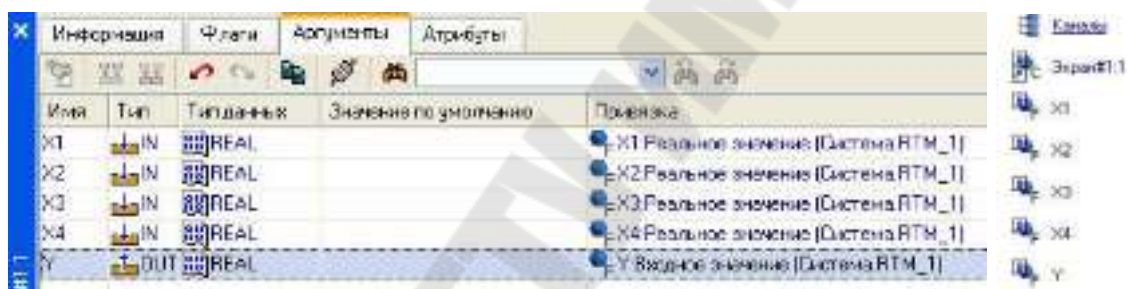


Рисунок 2.15 – Окно свойств экрана и созданные каналы

### 2.4 Создание программы на языке Техно FBD

Создадим программу, которая будет реализовывать заданную логическую функцию (см раздел «Контрольное задание»). Двойным щелчком ЛК открыть узел **RTM\_1** и создать в нем компонент «Программа» (рисунок 2.16).

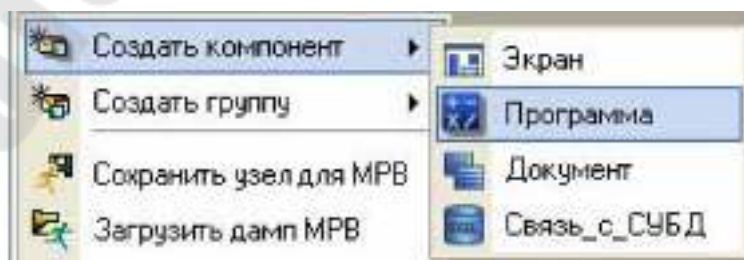


Рисунок 2.16 – Создание компонента Программа

Выделить компонент Программа#1 и ПК мыши вызвать контекстное меню, выбрав в котором ЛК мыши, пункт «**Редактировать шаблон**», перейти в режим редактирования программы (рисунок 2.17).

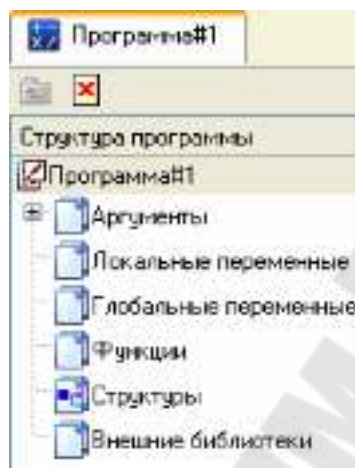



Рисунок 2.17 – Редактирование аргументов «Программы»

Выделением ЛК мыши в дереве шаблона **Программа#1** строку «**Аргументы**», вызвать табличный редактор аргументов. Кнопкой  создать в редакторе аргументов четыре аргумента X1, X2, X3, X4 и выход Y. При этом первые 4 аргумента должны быть типа **IN**, а последний – **OUT** (рисунок 2.18).

Имя	Тип	Тип данных	Значение по умолчанию
X1	IN	REAL	
X2	IN	REAL	
X3	IN	REAL	
X4	IN	REAL	
Y	OUT	REAL	

Рисунок 2.18 – Аргументы программы

Выделить в дереве шаблона строку **Программа#1** и в открывшемся диалоге «**Выбор языка**» выбрать язык **FBD** (рисунок 2.19).



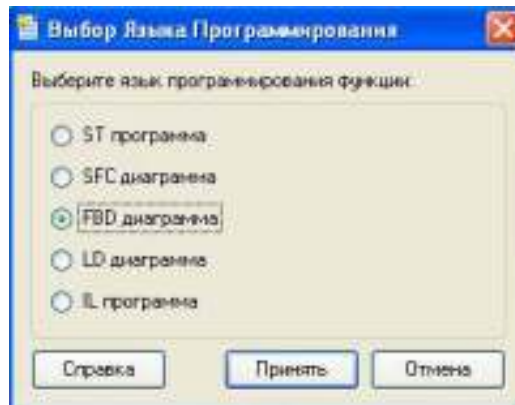


Рисунок 2.19 – Выбор языка программирования


По нажатию кнопки «**Принять**» в открывшемся окне редактора программ с объявленными переменными создать программу в соответствии с заданием (см. раздел «**Контрольное задание**»). Для выбора палитры FBD блоков необходимо ЛК мыши нажать на кнопку  после чего появится окно выбора FBD блоков (рисунок 2.20). При разработке программы верхние входы FBD блоков не используются т.к. они предназначены для изменения порядка пересчета блоков, а информационными входами, являются входы начиная со второго.



Рисунок 2.20 – Палитра FBD блоков

Для реализации заданной логической функции, выберем следующие блоки: из раздела «**Логические Функции**» **FBD** блоки инверсия (!X), логическое умножение (X&&Y и &&), логическое сложение (||). После размещения всех блоков для рассматриваемого примера программа будет выглядеть следующим образом – рисунок 2.21.

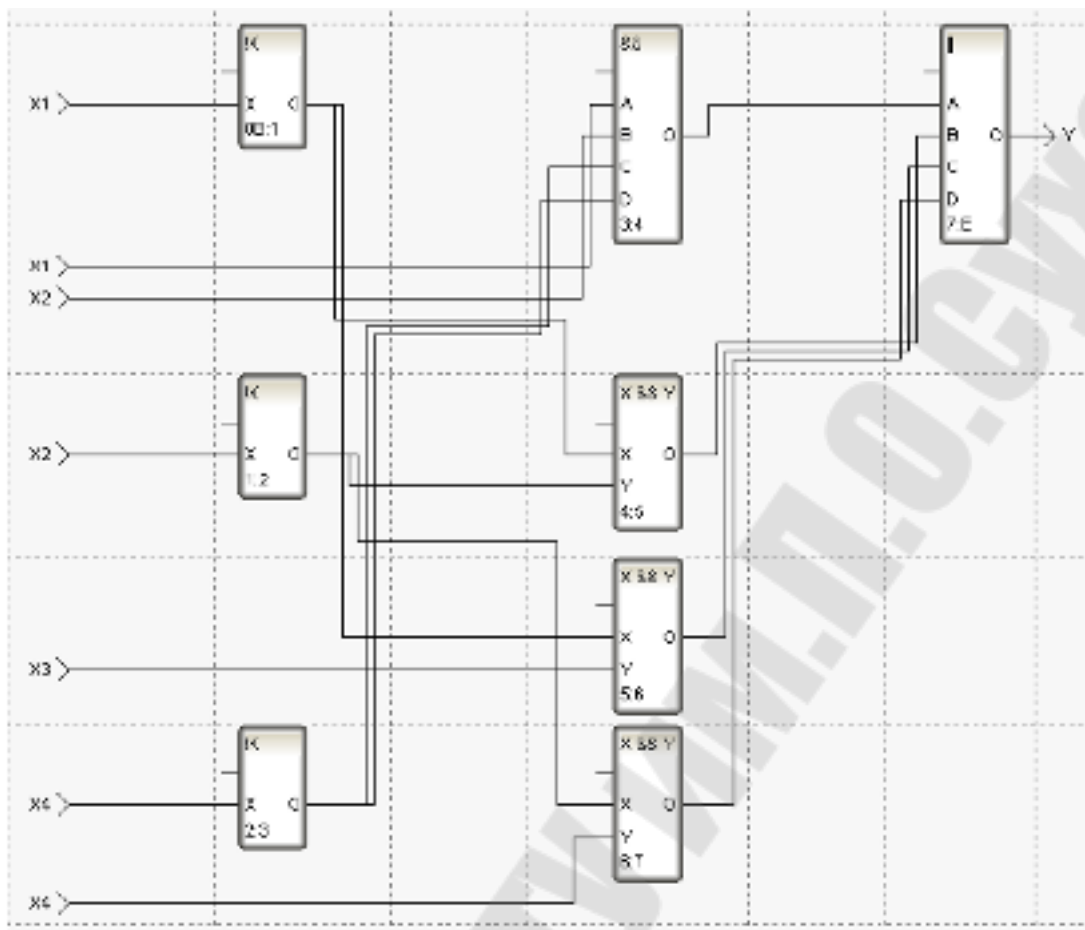




Рисунок 2.21 – Программа на языке Техно FBD

С помощью иконки  на инструментальной панели редактора или «горячей клавиши» F7, скомпилировать программу и убедиться в успешной компиляции в окне «Выход» (Output), вызываемого из инструментальной панели с помощью иконки  (рисунок 2.22).

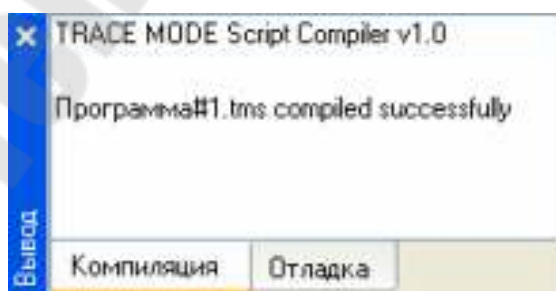


Рисунок 2.22 – Результат успешной компиляции программы

## 2.5 Привязка аргументов программы

Выполним привязку аргументов программы к атрибутам каналов. Для этого, вызвать свойства компонента **Программа#1** через контекстное меню. Выбрать вкладку «**Аргументы**». Двойным нажатием в поле «**Привязка**» привязать аргументы программы к атрибутам каналов – аргументы X1, X2, X3, X4 к **реальному значению** каналов X1, X2, X3, X4 (рисунок 2.23).

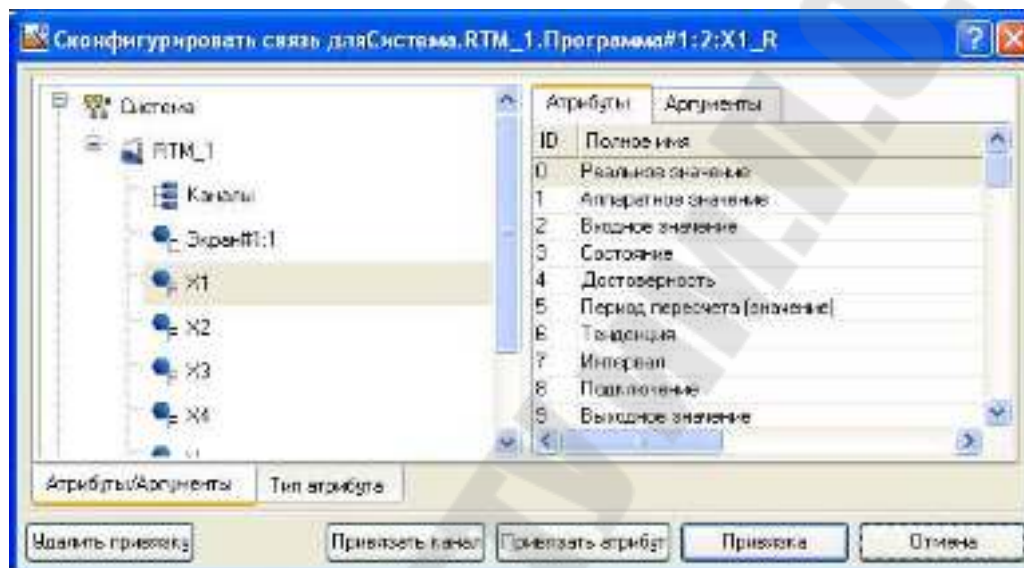


Рисунок 2.23 – Связь аргументов программы с аргументами экрана

Двойным щелчком в поле **Привязка** аргумента программы Y вызвать окно настройки связи, выбрать в левом окне канал класса «**Вызов**» **Экран#1**, а в правом окне выбрать вкладку «**Аргументы**» и указать в ней аргумент Y и кнопкой «**Привязка**» подтвердить связь (рисунок 2.24).

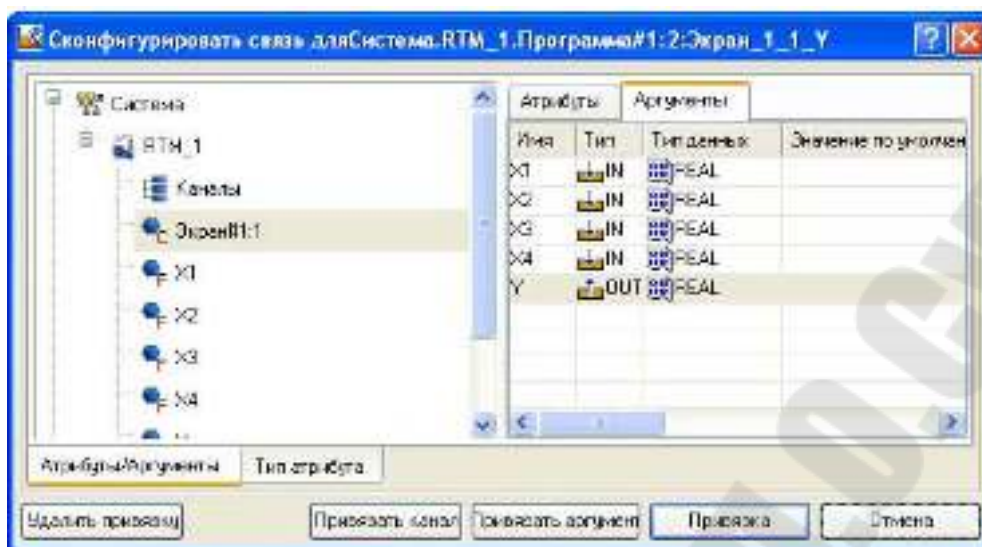


Рисунок 2.24 – Связь выходных аргументов

В результате, получим следующие привязки – рисунок 2.25.

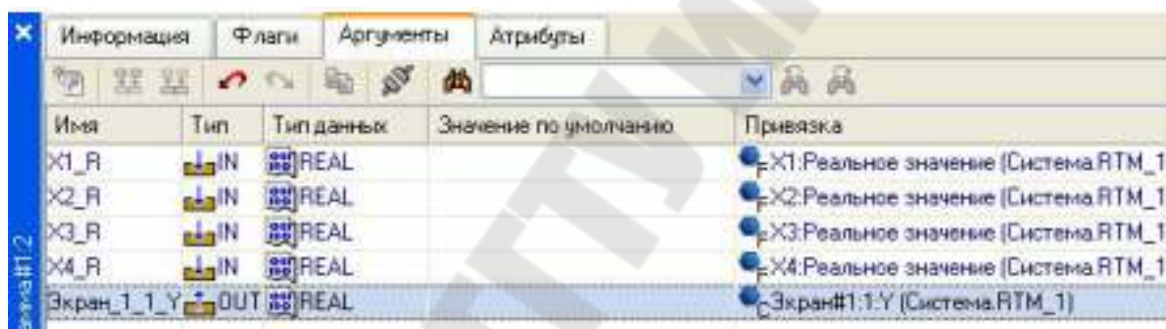






Рисунок 2.25 – Окончательная настройка связи

После закрыть окно свойств компонента **Программа#1**.

## 2.6 Запуск проекта

Сохранить проект с помощью кнопки . На инструментальной панели выберем иконку  и подготовим проект для запуска в реальном времени. ЛК выделим в слое «Система» узел **RTM\_1** (рисунок 2.26), а после, нажав ЛК иконку  на инструментальной панели, запустим профайлер. Запуск/останов профайлера осуществляется с помощью иконки  на его инструментальной панели или клавишной комбинации **Ctrl+R**.

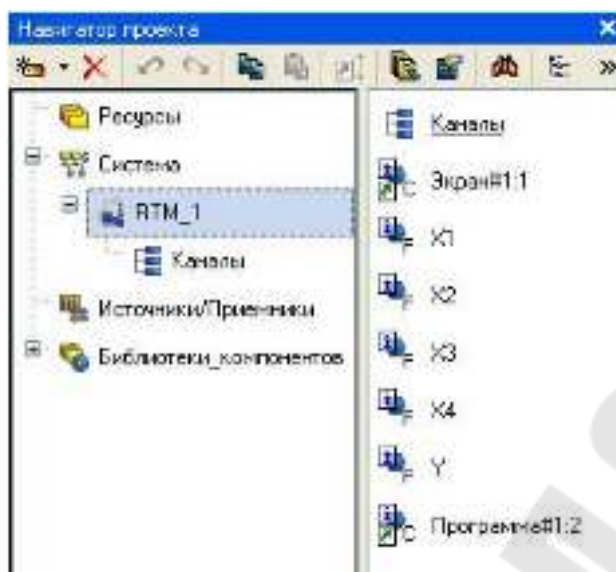


Рисунок 2.26 – Навигатор проекта

Для составления отчета необходимо в меню **Файл** выполнить команду «**Документировать проект**» полученный \*.html файл необходимо внести в отчет.

### 3 Контрольные задания

Варианты заданий выбираются по порядковому номеру в списке группы.

$$1. f = x_1 x_2 \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_4}$$

$$2. f = x_1 \overline{x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_4}$$

$$3. f = \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup x_4$$

$$4. f = \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3} \cup \overline{x_2 x_4}$$

$$5. f = \overline{x_1 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_4} \cup \overline{x_2 x_4}$$

$$6. f = x_1 x_4 \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_3 x_4} \cup \overline{x_2 x_3}$$

$$7. f = x_1 x_2 \cup \overline{x_1 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_2 x_4} \cup \overline{x_3 x_4}$$

$$8. f = \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3} \cup \overline{x_2 x_3 x_4}$$

$$9. f = x_1 x_4 \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_3 x_4} \cup \overline{x_2 x_3}$$

$$10. f = \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3 x_4}$$

11.  $f = x_1x_2x_4 \cup x_1x_2x_3 \cup x_1x_3 \cup x_2x_3 \cup x_1x_2x_3 \cup x_3x_4$
12.  $f = x_1x_3 \cup x_1x_4 \cup x_1x_2x_3 \cup x_1x_3x_4 \cup x_2x_3x_4$
13.  $f = x_1x_3 \cup x_1x_2x_4 \cup x_1x_2x_3 \cup x_1x_4$
14.  $f = x_1x_2x_3 \cup x_1x_2x_3 \cup x_1x_2x_3 \cup x_1x_2x_3 \cup x_3x_4 \cup x_3x_4$
15.  $f = x_1x_3 \cup x_1x_3x_4 \cup x_1x_2 \cup x_2x_4 \cup x_3x_4$
16.  $f = x_1x_2x_3x_4 \cup x_1x_3x_4 \cup x_1x_2x_3 \cup x_1x_2x_3 \cup x_1x_3x_4 \cup x_1x_3x_4 \cup x_2x_3x_4$
17.  $f = x_1x_3 \cup x_1x_2x_4 \cup x_1x_2x_4 \cup x_2x_3 \cup x_3x_4$
18.  $f = x_1x_3 \cup x_1x_2 \cup x_1x_4 \cup x_2x_4 \cup x_2x_3x_4$
19.  $f = x_1x_3 \cup x_1x_2x_3 \cup x_2x_3x_4$
20.  $f = x_1x_2x_4 \cup x_1x_3x_4 \cup x_1x_4 \cup x_1x_2x_3 \cup x_2x_3x_4$
21.  $f = x_1x_3 \cup x_1x_2x_3 \cup x_1x_3x_4 \cup x_2x_4$
22.  $f = x_1x_2x_4 \cup x_1x_3x_4 \cup x_1x_2x_3 \cup x_1x_2x_4 \cup x_2x_3x_4 \cup x_2x_3x_4$
23.  $f = x_1x_2x_3 \cup x_1x_2 \cup x_1x_4 \cup x_2x_3x_4 \cup x_3x_4$
24.  $f = x_1x_2x_3 \cup x_1x_3x_4 \cup x_1x_4 \cup x_2x_3$
25.  $f = x_1x_2x_4 \cup x_1x_2 \cup x_1x_3 \cup x_2x_4$

#### 4 Содержимое отчета

В отчете описать основные элементы проекта, аргументы элементов операторского интерфейса, программы, созданные каналы и привязки. Привести скриншоты основного окна операторского интерфейса в режиме симуляции и **FBD** программы.

#### 5 Контрольные вопросы

1. Языки МЭК 61131-3. Общие характеристики;
2. Принципы построения программ на языках МЭК 61131-3;
3. Язык FBD.
4. Порядок написания FBD-программ.
5. Понятие функционального блока.



## Лабораторная работа №2

### Реализация логического управления в SCADA TRACE MODE 6 с использованием языка Техно ST

**Цель работы:** освоить методику программирования логических функций при помощи SCADA–системы TRACE MODE на языке Техно ST.

#### 1. Теоретические сведения

Язык ST (Structured Text, Структурированный Текст) является текстовым языком высокого уровня, имеющий черты языков Pascal и Basic. Язык адаптирован специально для программирования ПЛК. С помощью ST удобно реализовывать арифметические и логические операции (в том числе, побитовые), безусловные и условные переходы, циклические вычисления; возможно использование как библиотечных, так и пользовательских функций.

Этот язык предназначен в основном для выполнения сложных математических вычислений, описания сложных функций, функциональных блоков и программ.

Основным недостатком является сложность использования языка не подготовленным разработчиком и отсутствие наглядного представления о алгоритме программы в целом и происходящих в ней процессах.

#### 1.1 Описание языка Техно ST

В алфавит языка входят:

1. прописные и строчные буквы латинского алфавита;
2. цифры;
3. специальные знаки: + - \* / < = > ! : & | ^ ~ % ( ) [ ] , ; #.

Идентификаторы могут состоять из прописных и строчных букв латинского алфавита, знака подчеркивания «\_», цифр. Идентификаторы не чувствительны к регистру.

Для данного языка характерны ключевые слова: *and, array, bool, break, by, byte, case, constant, continue, date, date\_and\_time, dint, do, dt, dword, else, elsif, endcase, endfor, endfunction, end\_function\_block, end\_if, end\_program, endrepeat, endstruct, endtype, endvar, endwhile, ex-*

*it, false, for, function, functionblock, goto, handle, if, int, lreal, mod, not, of, or, program, real, repeat, return, rol, ror, shl, shr, sint, string, struct, time, time\_of\_day, to, tod, true, type, udint, uint, until, usint, var, var\_arg, var\_global, var\_inout, var\_input, var\_output, while, word, xor.*

Строчный комментарий начинается с «//» и продолжается до конца строки. Блочный комментарий начинается с «/\*» и продолжается до «\*/».

Для описания структуры программы и операторов в приняты следующие терминологические соглашения:

- **выражение** – последовательность операндов, разделителей и символьных операторов, задающая вычисление без присвоения результата;
- **предложение** – последовательность лексем, определяющая выполнение логически законченного промежуточного действия. Таким действием может быть присвоение переменной результата вычислений, вызов функции-блока и т.п. Операторы (кроме символьных) также образуют предложения.

На основании этих соглашений программа или ее компонент на языке Техно ST определяется как последовательность предложений.

Каждое предложение должно **завершаться точкой с запятой**. Исключением из этого правила являются операторы определения переменных, для завершения которых точка с запятой не используется.

Длина строки программы не ограничивается, лексемы разделяются произвольным числом пробелов, знаков табуляции или символов перевода строки.

Основная точка входа в программу определяется следующей конструкцией:

```
program  
    {определение аргументов}  
    {список предложений}  
end_program
```

Необязательное выражение {определение аргументов} задается аналогично выражению {определение переменной} для операторов определения переменной. В дальнейшем конструкция **program...end\_program** называется основной программой.

Функции, глобальные переменные и структурные типы не могут быть определены в основной программе.



Переменные можно задать, заполняя таблицу аргументов, локальных и глобальных переменных. Переменные определяются в разделе описания аргументов **автоматически** в соответствии с заполненными таблицами аргументов и переменных.

Язык Turbo ST позволяет создавать числовые и строковые константы. Рассмотрим их представление. Целочисленная десятичная константа начинается с цифры, отличной от нуля, после которой располагаются любые цифры. Можно привести следующие примеры целочисленных десятичных констант: 123, 350, 498.

Двоичная целочисленная константа начинается с префикса «2#», после которого приводится двоичное представление целого числа. Примеры:

**2#1011, 2#0111, 2#1001.**

Восьмеричные целочисленные константы начинаются с префикса «8#», после которого записывается восьмеричное представление числа. Примеры:

**8#145, 8#0277, 8#756.**

Шестнадцатеричные целочисленные константы начинаются с префикса «16#», после которого приводятся шестнадцатеричные представления чисел. При записи шестнадцатеричного представления числа можно использовать как строчные символы а.. f, так и прописные А... F. Примеры:

**16#149, 16#A145E, 16#a145e.**

Вещественная константа состоит из целой и дробной части. Допустимо наличие только целочисленной или дробной части. Примеры: .123, 0.456, 489. . Возможно представление в формате с плавающей точкой (используется префикс e или E с указанием порядка). Примеры:

**1.23E-6, 6.7504E4, 6.798e-5.**

Частным случаем числовой константы является временной интервал, дата, время дня. Временной интервал записывается в виде:  
**t#<дни>d<часы>h<минуты>m<секунды>s<миллисекунды>ms**

Возможна также запись в виде:

**time#<дни>d<часы>h<минуты>m<секунды>s<миллисекунды>ms**

Любая составляющая в приведенных представлениях временного интервала может быть опущена.

**Пример:** Временной интервал равной 2 часам, 31 минута, 25 секундам и 10 миллисекунд может быть записан как **t#2h31m25s10ms** или в виде **time#2h31m25s10ms**.

Дата записывается в виде **d#<год>-<месяц>-<день>**, возможна также запись в виде **date#<год>-<месяц>-<день>**.

**Пример:** 25 сентября 2001 года может быть записано в виде: **d#2001-9-2001** или **date#2001-9-2001**.

Время дня можно записать в формате: **tod#<час>:<минута>:<секунда>** или **time\_of\_day#<час>:<минута>:<секунда>**.

**Пример:** Время 19 часов 15 минут 42 секунды может быть записано как: **tod#19:15:42**, так и **time\_of\_day#19:15:42**.

Константа «дата и время» может быть записана как: **dt#<год>-<месяц>-<день>-<час>:<минута>:<секунда>**, так и **date\_and\_time#<год>-<месяц>-<день>-<час>:<минута>:<секунда>**.

**Пример:** 12 февраля 1995 года 13 часов 47 минут и 13 секунд можно записать в виде: **dt#1995-2-12-13:47:13** или **date\_and\_time# 1995-2-12-13:47:13**.

Помимо числовых констант часто используются строковые константы, которые представляют собой набор символов заключенных в одинарные или двойные кавычки.

**Пример:** "Первая строка символов", Вторая строка символов'.

В строках не могут присутствовать управляющие символы, кавычки и символ \$. Для размещения в строке произвольного символа, включая управляющие, используется механизм эскейп-последовательностей. Данный механизм позволяет разместить в строке следующие последовательности:

- \$r— возврат каретки;
- \$n— перевод строки;

- \$t— табуляция;
- \$uXXXX— UNICODE символ, где XXXX— шестнадцатеричный символ;
- \$ x— символ x («x»— любой символ).

Рассмотрим символьные операторы. Под символьными операторами понимают знаки операций, выполняемых над операндами. В качестве операндов могут выступать:

1. имена констант;
2. имена переменных;
3. имена массивов с указанием индекса отдельного элемента;
4. вызов пользовательских функций;
5. вызов библиотечных функций;
6. выражения, заключенные в скобки;
7. имена элементов структур.

Арифметические операторы приведены в таблице 1.1, побитовые – в таблице 1.2, операторы сравнения – в таблице 1.3, логические – в таблице 1.4.

Таблица 1.1 – Арифметические операторы

Оператор	Действие
Унарный «-»	Смена знака
Унарный «+»	Пустая операция
«+»	Сложение чисел или конкатенация строк
«-»	Вычитание
«*»	Умножение
«/»	Деление
«%»	Получение остатка от деления
«**»	Возведение в степень

Таблица 1.2 – Побитовые операторы

Оператор	Действие
«&»	Побитовое «И»
« »	Побитовое «ИЛИ»
«^» или xor	Побитовое «исключающее ИЛИ»
Унарная «-»	Поразрядная инверсия
«<<» или shl	Сдвиг влево на указанное число разрядов
«>>» или shr	Сдвиг вправо на указанное число разрядов

rol	Циклический сдвиг влево на указанное число разрядов
ror	Циклический сдвиг вправо на указанное число разрядов

Таблица 1.3 – Операторы сравнения

Оператор	Проверяемое условие
«==»	Равенство
«!=» или «<>»	Неравенство
«<»	Меньше
«>»	Больше
«<=»	Меньше или равно
«>=»	Больше или равно

Таблица 1.4 – Логические операторы

Оператор	Действие
«&&» или and	Логическое «И»
«  » или or	Логическое «ИЛИ»
«!» или not	Логическое отрицание

Оператор присваивания позволяет произвести присваивание значения переменной. Есть два синтаксиса оператора присваивания:

{ операнд } = { выражение }  
 { операнд } := { выражение }

В качестве операнда может использоваться имя переменной, массива, уточненное имя переменной объекта. В таблице 1.5 приводится приоритет символьных операторов. Первыми будут выполняться выражения, заключенные в скобках, а затем в порядке возрастания номера строки в таблице.

Таблица 1.5 – Приоритет символьных операторов

Приоритет	Операция
1	**
2	!, not, -, унарный -, унарный +
3	<<, >>, shl, shr, rol, ror
4	&,  , ^, xor
5	*, /, %, mod

6	+, -
7	== != <> < <= > >=
8	&&, and
9	, or
10	

В ST – программе можно использовать стандартные функции языка Си: sin, cos, tan, asin, exp, log. Помимо символьных операторов в языке имеются и управляющие:

1. return
2. if
3. case
4. while
5. repeat
6. for
7. break
8. exit
9. continue
10. операторы определения переменных
11. операторы индексирования элементов массива
12. got.

Для разветвления алгоритма используется оператор if. Данный оператор всегда начинается с ключевого слова if и заканчивается ключевым словом end\_if Существует три варианта задания данного оператора. Первый вариант:

```
if {выражение} then {последовательность предложений};  
end_if;
```

В данном варианте последовательность предложений выполняется только в том случае, если выражение истинно. Второй вариант задания оператора if имеет вид:

```
if { выражение} then { последовательность предложений 1};  
else { последовательность предложений 2} ;  
end_if;
```

Во втором варианте задания оператора if проверяется выражение. Если оно истинно, то выполняется последовательность предложений 1, в противном случае— последовательность предложений 2. Третий вариант оператора if имеет вид:

```
if { выражение 1} then { последовательность предложений 1};  
elseif { выражение 2} then { последовательность предложений 2};  
...  
elseif { выражение N} then { последовательность предложений N};  
else { последовательность предложений N+1};  
end_if;
```

В последнем варианте оператора if выполняется i-ая последовательность предложений в том случае, если i-ое выражение истинно. Если все выражения ложны, то выполняется последовательность предложений, которая идет после ключевого слова else.

Язык Turbo ST содержит оператор выбора case. Оператор начинается с ключевого слова case и заканчивается ключевым словом end\_case. Первый вариант оператора case можно представить следующим образом:

```
case { выражение} of  
    { список значений}:{ последовательность предложений};  
    ...  
    { список значений}:{ последовательность предложений};  
end_case;
```

В данном случае вычисляется выражение, производится поиск результата вычисления в списках значений. Выполняется та последовательность предложений, в списках значений которой найден результат вычислений.

Второй вариант оператора case можно представить следующим образом:

```
case { выражение} of  
    { список значений}:{ последовательность предложений};  
    ...  
    { список значений}:{ последовательность предложений};  
else { последовательность предложений};  
end_case;
```

Отличие второй формы записи оператора case от первой заключается в том, что если результат вычисления выражения не найден ни в одном из списков значений, то выполняется последовательность предложений после ключевого слова else. В первом случае при отсут-

ствии результата вычислений в списках значений приведенные в операторе последовательности предложений не выполняются.

В обоих представлениях оператора case список значений – набор целых чисел или диапазонов целых чисел, разделенных запятой. Диапазон указывается в виде:

**{нижняя граница диапазона} .. {верхняя граница диапазона}**

Язык Техно ST позволяет создавать циклы используя операторы while, repeat, for. Синтаксис оператора while имеет вид:

```
while { выражение} do  
  { последовательность предложений};  
end_while;
```

В данном операторе последовательность выражений выполняется пока выражение истинно. Каждый раз перед выполнением последовательности предложений производится проверка выражения.

Оператор repeat можно представить в виде:

```
repeat  
  { последовательность предложений};  
until { выражение} end_repeat;
```

В операторе repeat последовательность выполняется, проверяется выражение, если оно истинно, то последовательность выражений повторяется снова, в противном случае происходит выход из цикла.

Цикл for можно представить в виде:

```
for {имя переменной} := {выражение 1} to {выражение 2} by {выражение 3} do  
  {последовательность предложений};
```

```
end_for;
```

Данный оператор сначала присваивает переменной цикла с указанным именем результат вычисления выражения 1, выполняет последовательность предложений, если вычисленная переменная цикла не превысит значение выражения 2. Затем к переменной цикла прибавляется выражение 3, выполняется последовательность предложений, если вычисленное значение не превышает выражение 2. Выполнение последовательности предложений и увеличение значения переменной на выражение 3 повторяется до тех пор, пока значение пере-

менной цикла не превышает выражения 2. Для цикла for характерно то, что он не позволяет создавать цикл с отрицательным шагом.

Операторы break и exit позволяют выйти из текущего цикла. Оператор continue служит для перехода в конец цикла. При его вызове все следующие за ним до конца цикла предложения не выполняются.

## 1.2 Создание программы на языке Техно ST

Для создания программы, необходимо двойным щелчком ЛК открыть узел **RTM\_1** и создать в нем компонент «**Программа**» (рисунок 1.1).

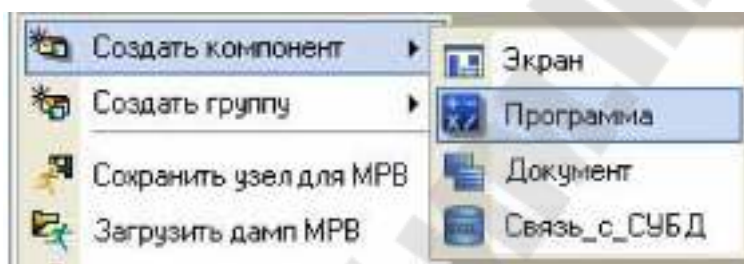


Рисунок 1.1 – Создание компонента Программа

Выделить компонент Программа#1 и ПК мыши вызвать контекстное меню, выбрав в котором ЛК мыши, пункт «**Редактировать шаблон**», перейти в режим редактирования программы (рисунок 1.2).

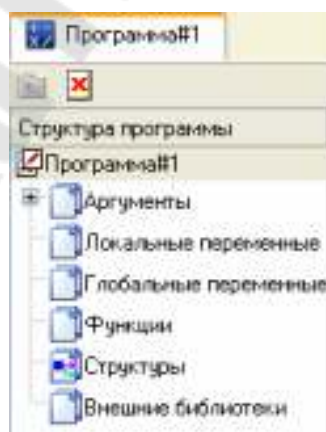



Рисунок 1.2 – Редактирование аргументов «Программы»

Выделением ЛК мыши в дереве шаблона **Программа#1** строку «**Аргументы**», вызвать табличный редактор аргументов. Кнопкой  создать в редакторе аргументов четыре аргумента X1, X2, X3, X4



и выход Y. При этом первые 4 аргумента должны быть типа **IN**, а последний – **OUT** (рисунок 1.3).

Имя	Тип	Тип данных	Значение по умолчанию	Ссылка
K	IN	REAL		X: Реальное значение (Система, РТН_1, Каналы)
Y	IN	REAL		Y: Реальное значение (Система, РТН_1, Каналы)
Z	OUT	REAL		Z: Выходное значение (Система, РТН_1, Каналы)
Sum	OUT	REAL		Sum: Выходное значение (Система, РТН_1, Каналы)

Рисунок 1.3 – Аргументы программы

Аналогично задаются и глобальные переменные/

Имя	Тип данных	[ ]	Начальное значение	Комментарий
S	REAL		0	

Рисунок 1.4 – Глобальные переменные программы

Выделить в дереве шаблона строку **Программа#1** и в открывшемся диалоге «**Выбор языка**» выбрать язык **ST** (рисунок 1.5).

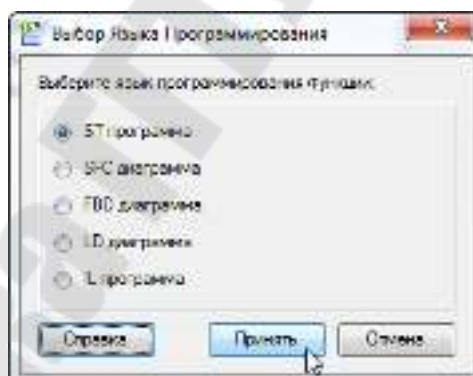


Рисунок 1.5 – Выбор языка программирования

После выбора языка откроется окно редактора, изображенное на рис. 1.6. Все созданные аргументы, и локальные переменные будут описаны в начале программы. **Глобальные переменные** в отличие от локальных, в самой программе не описываются, но они могут использоваться как операнды.

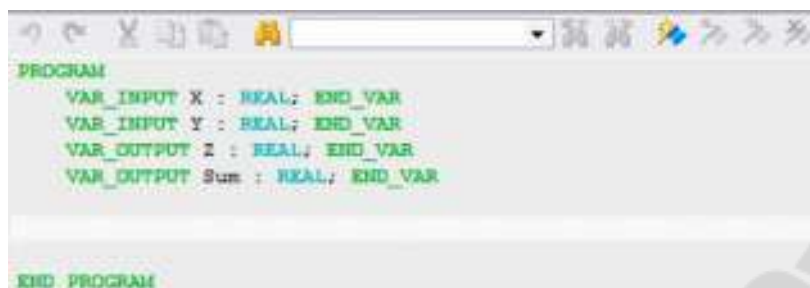




Рисунок 1.6 – Окно редактора для создания программы

Составим программу вычисления деления аргумента  $X$  на аргумент  $Y$ , произведя проверку на равенство  $Y$  нулю, результат присвоим аргументу  $Z$ . Аргументу  $Sum$  присваивается сумма всех результатов деления. Листинг программы:

```
PROGRAM  
  VARINPUT X : REAL; ENDVAR  
  VARINPUT Y : REAL; ENDVAR  
  VAR_OUTPUT Z : REAL; END_VAR  
  VARINOUT Sum : REAL; ENDVAR  
  if Y == 0 then Z = X / 1e-9;  
  else Z = X / Y;  
  end_if;  
  Sum = S+ Z;    // накапливаем сумму  
  S = Sum;    // сохраняем ее в глобальной переменной до следующего вызова программы  
END_PROGRAM
```

После того, как написали текст программы необходимо проверить ее. С помощью иконки  на инструментальной панели редактора или «горячей клавиши» F7, скомпилировать программу и убедиться в успешной компиляции в окне «Выход» (Output), вызываемого из инструментальной панели с помощью иконки .

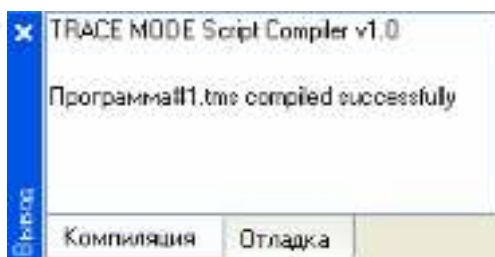


Рисунок 1.7 – Результат успешной компиляции программы

## 2 Ход работы

## 2.1 Задание

На основе проекта, реализации логической функции на языке Техно FBD, выполнить реализацию той же функции на языке Техно ST, следующими способами.

1. Арифметическими операторами;
2. Побитовыми операторами;

## 2.2 Варианты заданий

Варианты заданий выбираются по номеру в списке группы.

1.  $f = x_1 x_2 \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_4}$
2.  $f = \overline{x_1 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_4}$
3.  $f = \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup x_4$
4.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3} \cup \overline{x_2 x_4}$
5.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_4} \cup \overline{x_2 x_4}$
6.  $f = \overline{x_1 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_3 x_4} \cup \overline{x_2 x_3}$
7.  $f = \overline{x_1 x_2} \cup \overline{x_1 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_2 x_4} \cup \overline{x_3 x_4}$
8.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3} \cup \overline{x_2 x_3 x_4}$
9.  $f = \overline{x_1 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_3 x_4} \cup \overline{x_2 x_3}$
10.  $f = \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3 x_4}$
11.  $f = \overline{x_1 x_2 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_3} \cup \overline{x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_3 x_4}$
12.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_3 x_4} \cup \overline{x_2 x_3 x_4}$
13.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_4}$
14.  $f = \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_3 x_4} \cup \overline{x_3 x_4}$
15.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_2} \cup \overline{x_2 x_4} \cup \overline{x_3 x_4}$
16.  $f = \overline{x_1 x_2 x_3 x_4} \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_3 x_4} \cup \overline{x_2 x_3 x_4}$
17.  $f = \overline{x_1 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_1 x_2 x_4} \cup \overline{x_2 x_3} \cup \overline{x_3 x_4}$

$$18. f = \overline{x_1 x_3} \cup \overline{x_1 x_2} \cup \overline{x_1 x_4} \cup \overline{x_2 x_4} \cup \overline{x_2 x_3 x_4}$$

$$19. f = x_1 x_3 \cup \overline{x_1 x_2 x_3} \cup \overline{x_2 x_3 x_4}$$

$$20. f = x_1 x_2 x_4 \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_2 x_3 x_4}$$

$$21. f = x_1 x_3 \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_3 x_4} \cup \overline{x_2 x_4}$$

$$22. f = x_1 x_2 x_4 \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_2 x_3} \cup \overline{x_1 x_2 x_4} \cup \overline{x_2 x_3 x_4} \cup \overline{x_2 x_3 x_4}$$

$$23. f = x_1 x_2 x_3 \cup \overline{x_1 x_2} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3 x_4} \cup \overline{x_3 x_4}$$

$$24. f = x_1 \overline{x_2 x_3} \cup \overline{x_1 x_3 x_4} \cup \overline{x_1 x_4} \cup \overline{x_2 x_3}$$

$$25. f = x_1 x_2 \overline{x_4} \cup \overline{x_1 x_2} \cup \overline{x_1 x_3} \cup \overline{x_2 x_4}$$

### 3 Содержимое отчета

В отчете описать основные элементы проекта, аргументы элементов операторского интерфейса, программы, созданные каналы и привязки. Привести скриншоты основного окна операторского интерфейса в режиме симуляции и ST программы.

### 4 Контрольные вопросы

1. Языки МЭК 61131-3. Общие характеристики;
2. Принципы построения программ на языках МЭК 61131-3;
3. Язык ST. Описание языка.
4. Язык ST. Структура программы.

### Лабораторная работа № 3

#### Реализация системы автоматического регулирования при помощи в SCADA TRACE MODE 6 с использованием языка Техно FBD

**Цель:** освоить методику программирования системы автоматического регулирования при помощи SCADA-системы TRACE MODE на языке Техно FBD

#### 1. Теоретические сведения

Для операций моделирования управления объектами необходимо использование блоков задающих (описывающих) объект управления. В TRACE MODE для этого используется блок **OBJ** (рисунок 1.1).

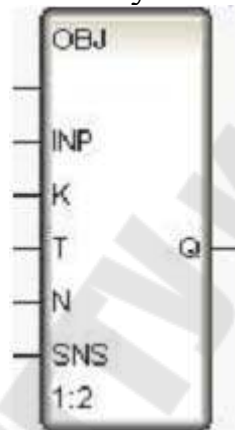


Рисунок 1.1 – FBD блок – модель объекта (OBJ)

Данный блок моделирует объект управления для отладки алгоритмов регулирования или подготовки демонстрационных проектов. Он представляет собой комбинацию периодического (инерционного) звена первого порядка и звена запаздывания. Передаточная функция блока имеет вид:

$$\Phi_o(s) = \frac{k}{Ts + 1} e^{-\lambda s}$$

где

- **k** и **T** – соответственно коэффициент усиления и постоянная времени инерционного звена первого порядка;
- **λ > 0** – время запаздывания.

Кроме того, на выходной сигнал блока можно наложить помеху в виде случайной составляющей, синусоидального сигнала или случай-

ных бросков. Здесь же можно задать случайное колебание динамических характеристик объекта. Входным по отношению к моделируемому объекту является вход **INP**. Входы **K**, **T** и **N** используются для задания соответственно коэффициента усиления, постоянной времени и времени запаздывания. Последние два параметра задаются в тактах пересчета, максимальное значение времени запаздывания – 4. Вход **SNS** предназначен для управления случайными помехами, вносимыми в работу объекта. Значение отдельных битов этого входа включает следующие помехи:

- 1 бит – добавление к выходному сигналу случайной величины в диапазоне от 0 до 1%;
- 2 бит – формирование пика величиной 25% от значения выхода с вероятностью 0,01;
- 3 бит – добавление к выходу синусоидального сигнала с амплитудой 2% от значения выхода;
- 5 бит – случайное увеличение коэффициента усиления в диапазоне от 0 до 2%;
- 6 бит – случайное увеличение постоянной времени в диапазоне от 0 до 2%;
- 7 бит – случайное изменение на 1 запаздывание.

Первые три помехи добавляются к выходу блока после формирования его нового значения. Динамические характеристики объекта (последние три помехи) корректируются до пересчета блока.

Для реализации регулирующих функций в TRACE MODE реализованы типовые алгоритмы управления. Наиболее распространенным алгоритмом автоматического управления является ПИД регулирование. В TRACE MODE для этого используется FBD блок – ПИД регулятора **PID** (рисунок 1.2).

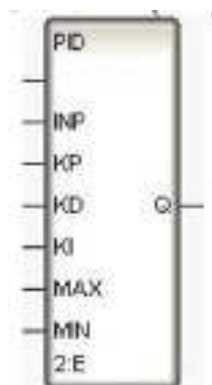


Рисунок 1.2 – FBD блок – ПИД регулятор

Этот блок формирует выходное значение по ПИД – закону от величины, поданной на вход **INP**:

$$Q_i = KP \times INP_i + \frac{KD \times (INP_i - INP_{i-1})}{\Delta t} + KI \times \Delta t \times \sum_{k=1}^i INP_k$$

где

**i** – текущий такт пересчета; **KP**, **KD** и **KI** – соответственно коэффициенты при пропорциональной, дифференциальной и интегральной составляющих; **Δt** – период пересчета блока в секундах (длительность такта).

Модуль подаваемого на вход **KI** отрицательного значения передается на выход. При подаче на вход **KI** неотрицательного значения регулирование начинается с установленной величины. Для ограничения величины управляющего воздействия используются входы блока **MIN** и **MAX**. Если величина управления меньше **MIN**, то **Q = MIN**, если величина управления больше **MAX**, то **Q = MAX**, при этом в обоих случаях накопление интегральной составляющей закона регулирования прекращается. Данный блок вычисляет величину управляющего воздействия по значению рассогласования регулируемой величины и задания, которое предварительно надо вычислять с помощью блока **X-Y**. Введение в алгоритм параметра **Δt** исключает необходимость пересчета настроек регулятора при смене периода пересчета.

## 2 Ход работы

### 2.1 Создание проекта Trace Mode

Рассмотрим создание системы автоматического регулирования при помощи SCADA-системы TRACE MODE на языке Техно FBD. Необходимо разработать систему с ПИД регулированием технологическим объектом, с заданной передаточной функцией (параметры по варианту):

$$W(s) = \frac{K \cdot e^{-\tau s}}{T \cdot s + 1}$$

где **K** – коэффициент усиления объекта, **T** – постоянная времени; **τ** – время запаздывания.

Разработка любого проекта автоматизации всегда начинается с запуска Интегрированной среды разработки (ИСР).



После запуска ИСР в меню «**Файл**» выбрать команду «**Настройки ИС...**». В появившемся окне в закладке «**Уровень сложности**» настроить проект как показано на рисунке 2.1, а затем выбрать закладку «**Отладка**» и настроить проект как показано на рисунке 2.2.

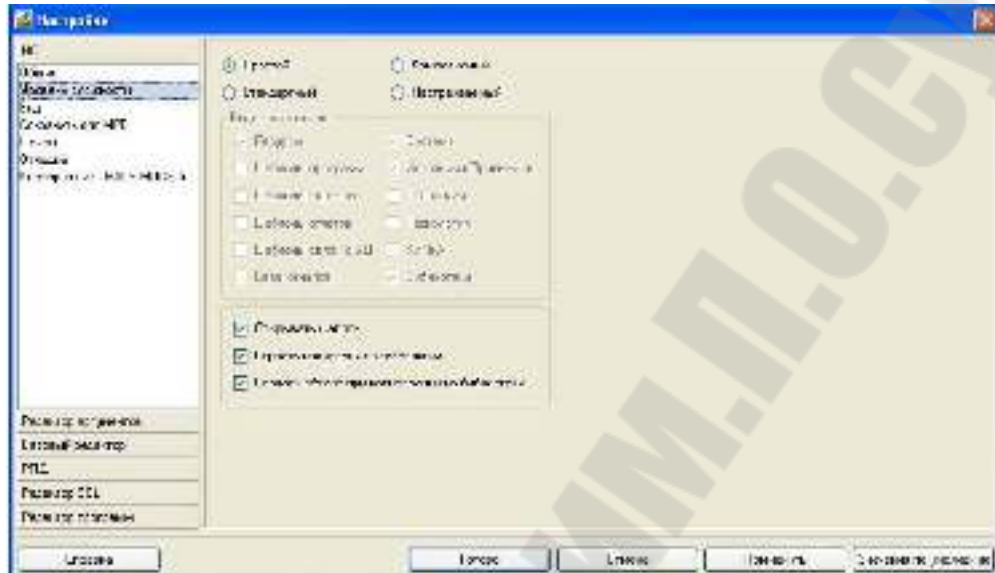


Рисунок 2.1 – Настройки «Уровень сложности»

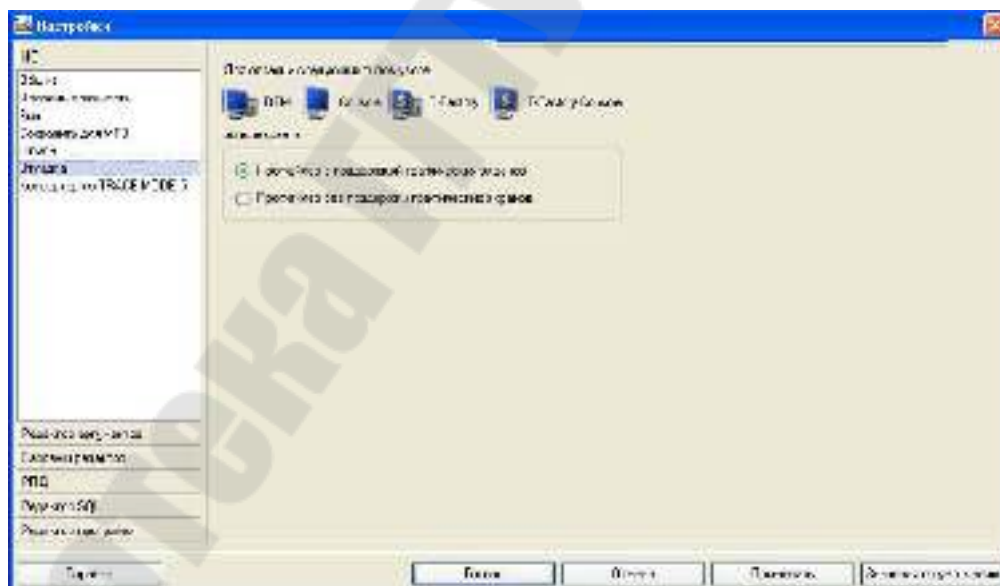



Рисунок 2.2 – Настройки «Отладка»

После проведенных настроек ИСР нажать кнопку «**Готово**». С помощью иконки  инструментальной панели создадим новый проект при этом в открывшемся на экране диалоге (рисунок 1.3).



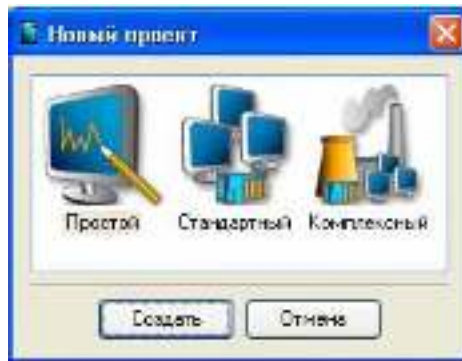


Рисунок 2.3 – Выбор типа проекта

Выберем «**Простой**» стиль разработки. После нажатия левой клавиши мыши (ЛК) на экранной кнопке «**Создать**», в левом окне Навигатора проекта появится дерево проекта с созданным узлом **АРМ RTM\_1**. Откроем узел **RTM\_1** двойным щелчком ЛК, в правом окне «**Навигатора проекта**» отобразится содержимое узла – пустая группа «**Каналы**» и один канал класса «**Вызов**» **Экран#1**, предназначенный для отображения на узле **АРМ** графического экрана (рисунок 2.4).

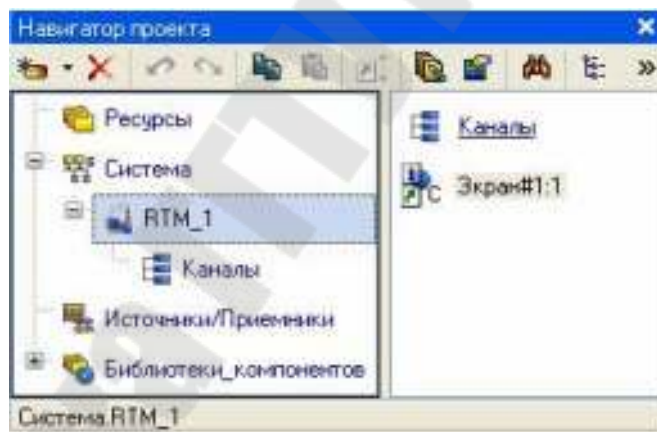



Рисунок 2.4 – Навигатор проекта

## 2.2 Создание графического интерфейса

Двойным щелчком ЛК на компоненте **Экран#1** открываем окно графического редактора. Подготовим на экране вывод динамического текста для отображения численного значения входных переменных апериодического звена первого порядка: коэффициента усиления объекта ( $K_{об}$ ), постоянной времени объекта ( $T_{об}$ ), времени запаздывания ( $T_{за}$ ); ПИД регулятора: коэффициент усиления регулятора ( $K_r$ ), времени интегрирования ( $T_i$ ), времени дифференцирования ( $T_d$ ); и величины задания ( $Z_{ад}$ ).

Выберем на инструментальной панели графического редактора иконку ГЭ «Кнопка» -  С помощью мыши разместим семь элементов в поле экрана как показано на рисунке 2.5.

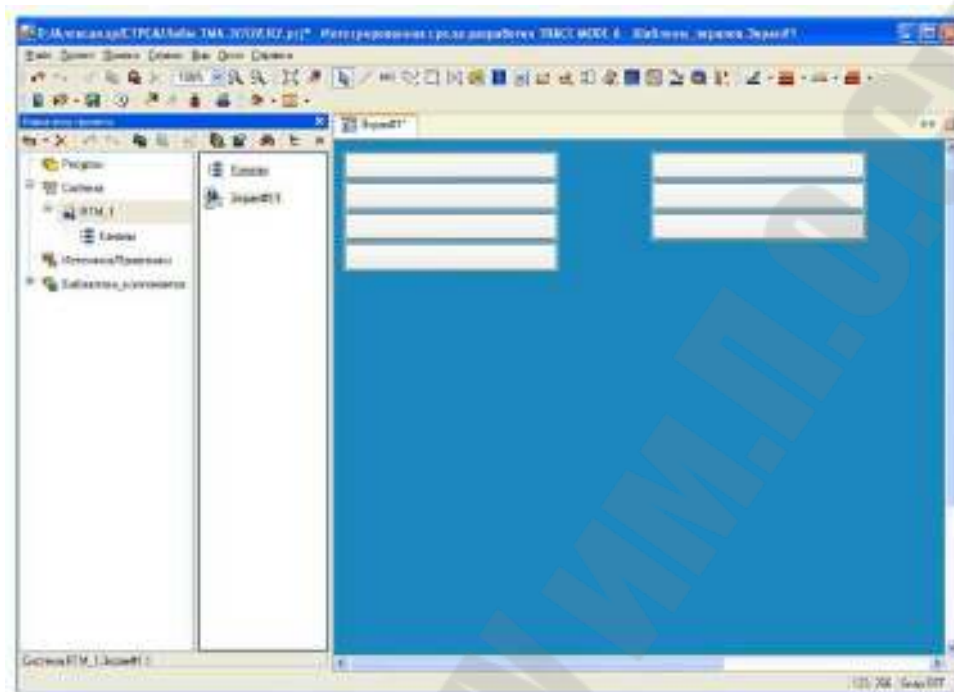




Рисунок 2.5 – Вид экрана оператора

Перейти в режим редактирования нажатием  кнопки, затем двойным щелчком ЛК вызвать окно свойств первого ГЭ  рисунок 2.6.

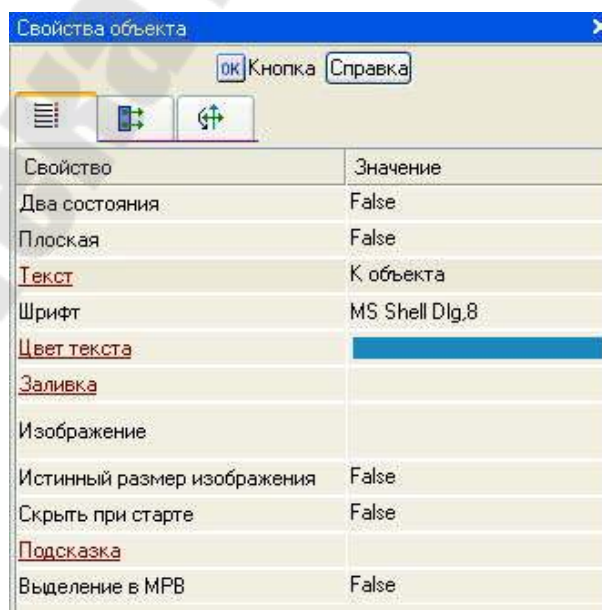


Рисунок 2.6 – Свойства графического элемента «Кнопка»

В поле «Текст» ввести «**К объекта**». Открыть закладку «**Действия**» и (правой кнопкой мыши) ПК раскрыть меню «**События По нажатию (mousePressed)**». Выбрать из списка команду «**Добавить Передать значение**», раскрыть меню настроек выбранной команды рисунок 2.7.

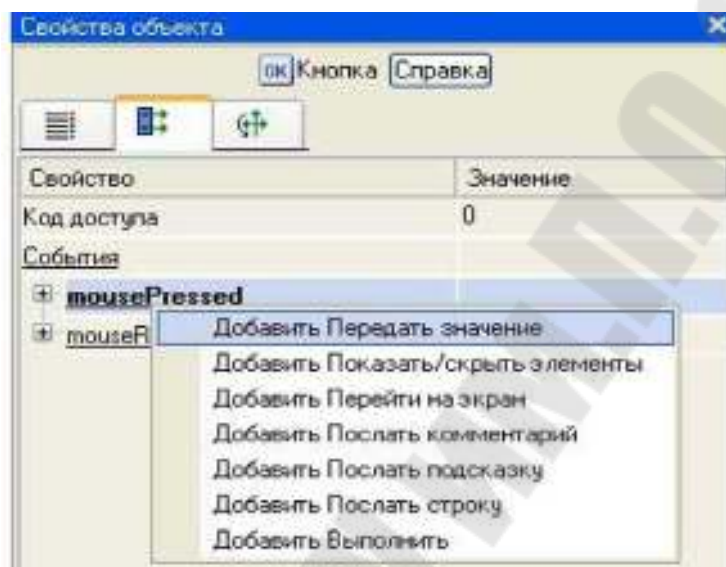


Рисунок 2.7 – Настройка типа передачи

В поле «**Тип передачи (Send Type)**» выбрать из списка «**Ввести и передать (Enter & Send)**» (рисунок 2.8).

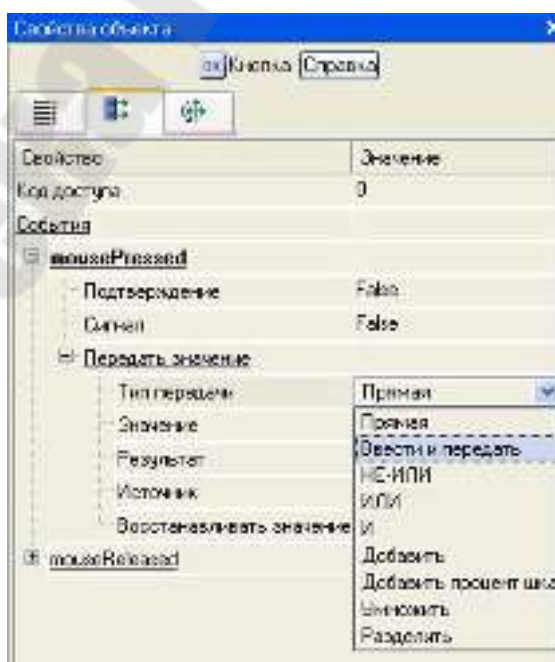



Рисунок 2.8 – Настройка типа передачи

ЛК в поле «**Результат**» вызвать табличный редактор аргументов. В открывшемся окне «**Свойство привязки**», нажав кнопку  на его панели инструментов, создать восемь аргументов экрана 7 входных и один выходной. «**Входной (IN)**» или «**Выходной (OUT)**» будет аргумент, выбирается из низпадающего меню, появляющегося при двойном щелчке ЛК по типу аргумента (рисунок 2.9). Двойным щелчком ЛК выделить имя аргумента и изменить его, введя с клавиатуры «**Коб**» (завершить ввод нажатием клавиши Enter). Подтвердить связь с этим аргументом нажатием кнопки «**Готово**» Аналогичным образом настроить все остальные аргументы.

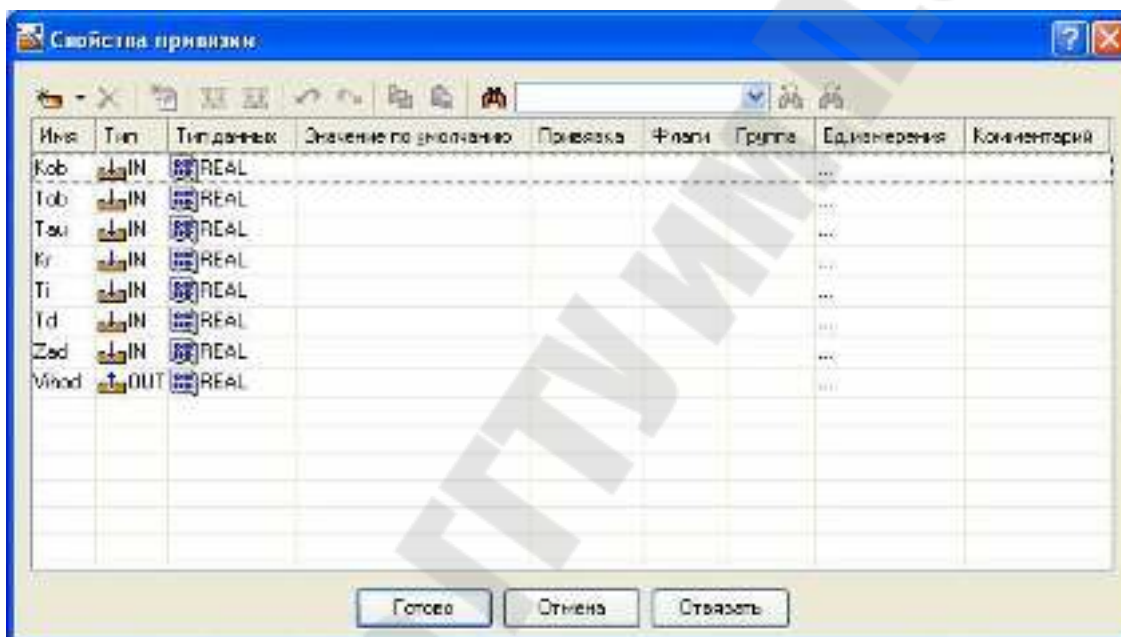


Рисунок 2.9 – Создание аргументов экрана

После привязки окно «**Свойства объекта**» будет выглядеть следующим образом – рисунок 2.10.

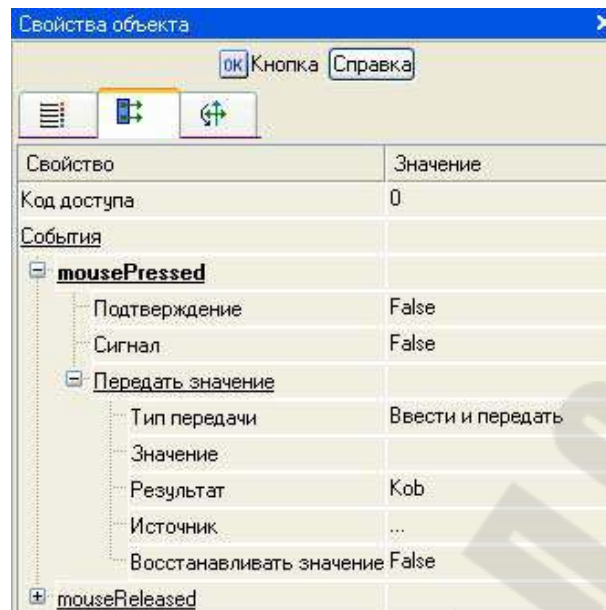


Рисунок 2.10 – Окончательный вид окна свойств графического элемента

Аналогичным образом настроить остальные кнопки  $T_{ob}$ ,  $T_{au}$ ,  $K_r$ ,  $T_i$ ,  $T_d$ ,  $Z_{ad}$ .

Для отображения числового значения входных аргументов создадим и разместим семь ГЭ «Текст»  $ABC$ , как показано на рисунке 2.11. Для чего, активировав ГЭ  $ABC$  нажатием ЛК, рисуем области вывода динамического текста, задавая левый верхний и правый нижний углы щелчком ЛК.

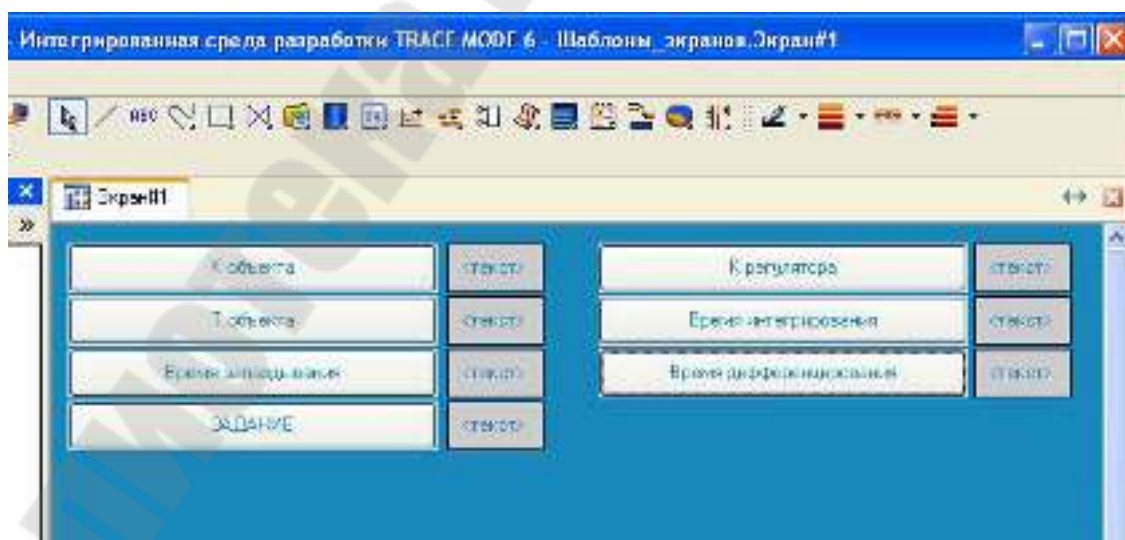



Рисунок 2.11 – Создание текстовых графических элементов



После этого необходимо переключиться в режим редактирования активацией ГЭ . Двойным щелчком ЛК на строке «Текст» вызвать меню «Вид индикации» (рисунок 2.12.)

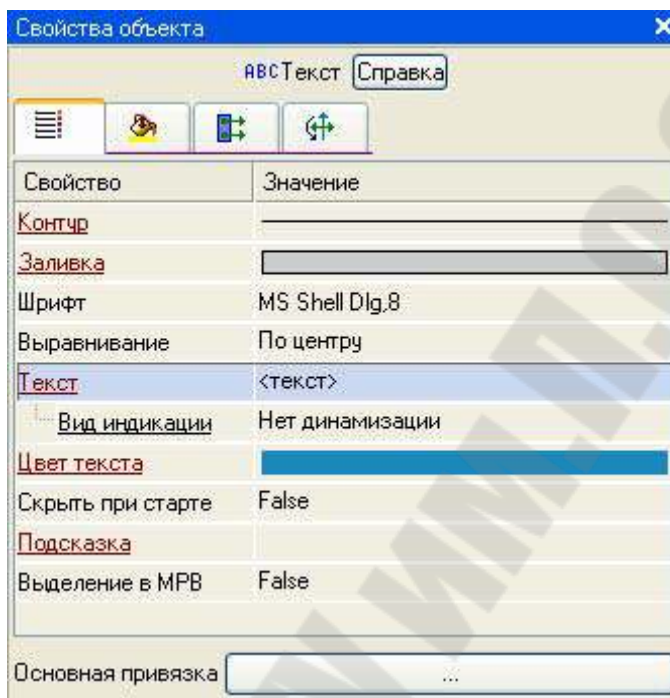


Рисунок 2.12 – Настройка индикации

В окне «Свойства объекта», двойным щелчком ЛК по свойству «Текст» раскрыть его элементы. В появившемся подпункте «Вид индикации», щелкнув по значению «Нет динамизации» выбрать тип «Значение» (рисунок 2.13).

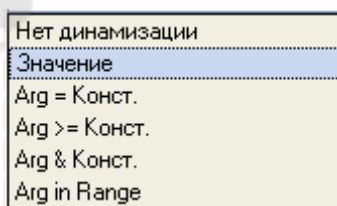




Рисунок 2.13 – Выбор типа динамизации

В открывшемся меню настройки параметров динамизации, выбрать свойство «Привязка» и связать с **Kob** (рисунок 2.14).

<b>Текст</b>	<текст>
[-] Вид индикации	Значение
Привязка	...
[-] Формат	Generic
Generic	%g

Рисунок 2.14 – Настройка привязки

Дополним созданный экран новым ГЭ для совместного просмотра изменений значений каналов узла во времени и отслеживании предыстории – трендом.

На экрана разместим ГЭ «Тренд»  для вывода значений **Zad** и **Vihod** (рисунок 2.15). Основные свойства ГЭ  оставим заданными по умолчанию.

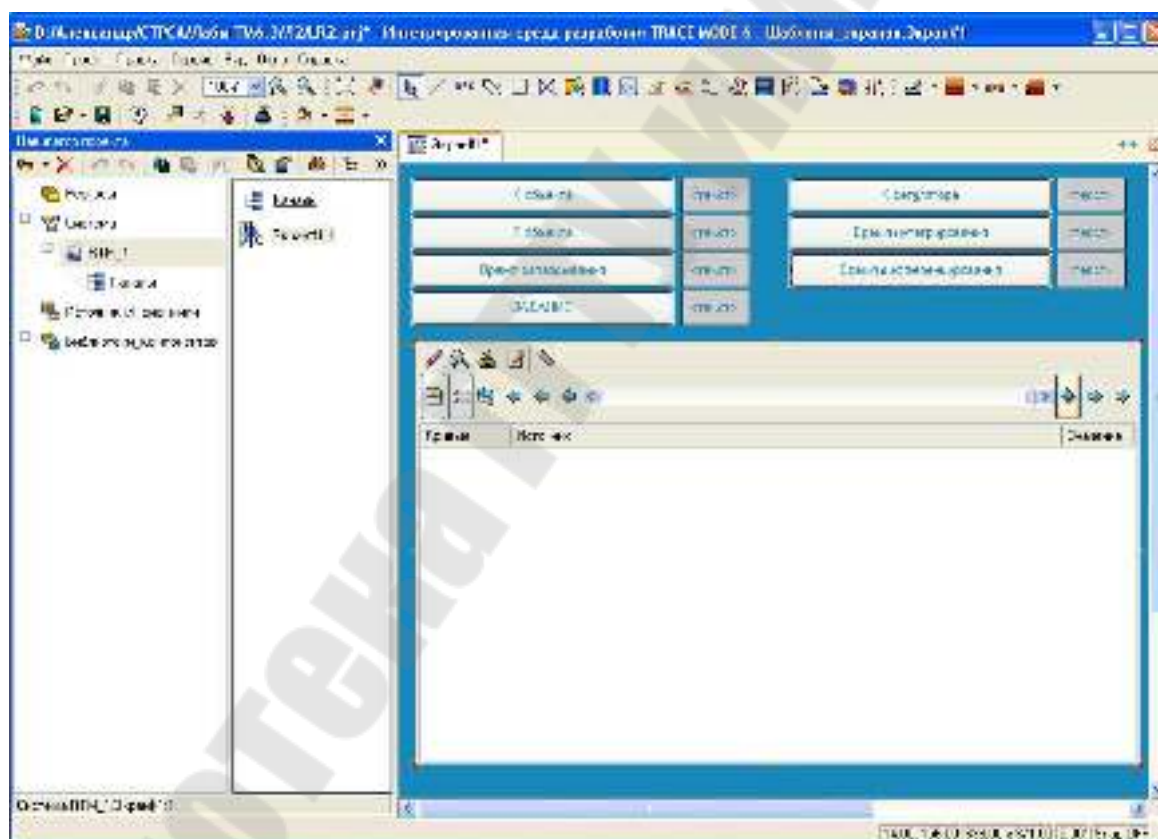



Рисунок 2.15 – Размещение тренда

Перейдем во вкладку , выделив ЛК строку «**Кривые**», с помощью ПК создадим две новые кривые. Настроим их привязки к аргументам, толщину и цвет линий. Двойным щелчком ЛК мыши в поле тренда открываем окно «**Свойства тренда**» и переходим на вкладку «**Кривые**». Нажимая ПК мыши в поле «**Кривые**» задаем две кривые

**Zad** и **Vihod**, а в поле привязка привязываем созданные кривые к заданию и выходу как показано на рисунке 2.16.

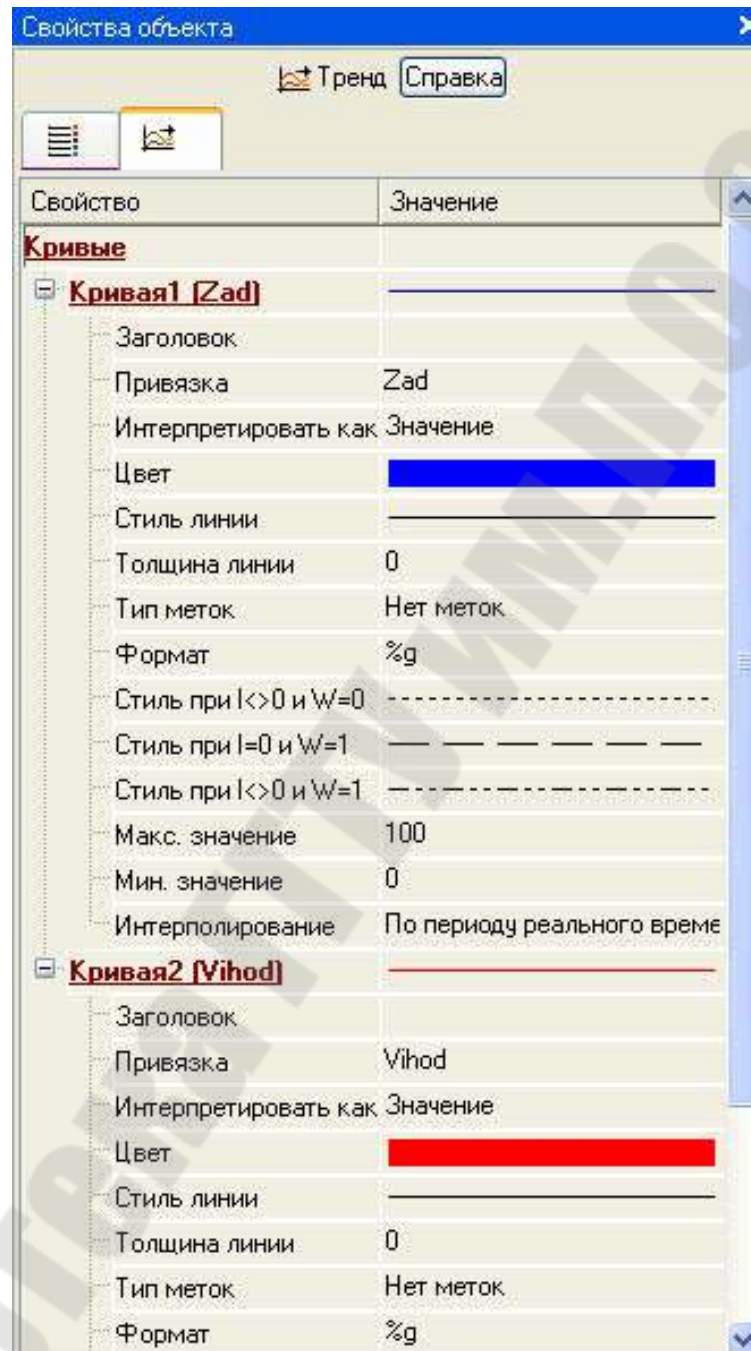



Рисунок 2.16 – Свойства тренда

### 2.3 Привязка аргумента экрана к каналу

Создадим по аргументам Kob, Tob, Tau, Kr, Ti, Td, Zad, Vihod, шаблона экрана новые каналы и отредактируем их привязку. В слое «Система» открыть узел **RTM\_1**. С помощью ПК вызвать контекст-



ное меню свойства компонента **Экран#1** и выбрать команду «**Свойства**» (рисунок 2.17). Выбрать вкладку «**Аргументы**», ЛК, при нажатой клавише **Ctrl**, выделить аргументы **Kob**, **Tob**, **Tau**, **Kr**, **Ti**, **Td**, **Zad**, **Vihod** и с помощью иконки  создать новые каналы.

Имя	Тип	Тип данных	Значение по умолчанию	Привязка
Kob	IN	REAL		F Kob:Реальное значение (Система.RTM_1)
Tob	IN	REAL		F Tob:Реальное значение (Система.RTM_1)
Tau	IN	REAL		F Tau:Реальное значение (Система.RTM_1)
Kr	IN	REAL		F Kr:Реальное значение (Система.RTM_1)
Ti	IN	REAL		F Ti:Реальное значение (Система.RTM_1)
Td	IN	REAL		F Td:Реальное значение (Система.RTM_1)
Zad	IN	REAL		F Zad:Реальное значение (Система.RTM_1)
Vihod	OUT	REAL		F Vihod:Входное значение (Система.RTM_1)

Рисунок 2.17 – Окно свойств экрана

В результате, в узле **RTM\_1**, будут автопостроены следующие каналы – рисунок 2.18.



Рисунок 2.18 – Автопостроенные каналы

## 2.4 Создание программы на языке Техно FBD

Для создания FBD-программы и подключения ее к проекту нужно выполнить следующие операции:

1. разместить необходимые функциональные блоки в рабочем поле FBD-редактора;

2. соединить нужные входы и выходы блоков, образовав единую диаграмму;
3. задать аргументы, переменные и константы программы;
4. привязать входы/выходы FBD-диаграммы к аргументам, переменным и константам программы;
5. скомпилировать программу

Создадим программу, которая будет реализовывать одноконтурную систему автоматического регулирования. Для этого ПК по узлу RTM\_1 вызвать контекстное меню выбрать подменю «Создать компонент» и выбрать ЛК в нем компонент «Программа» (рисунок 2.19).

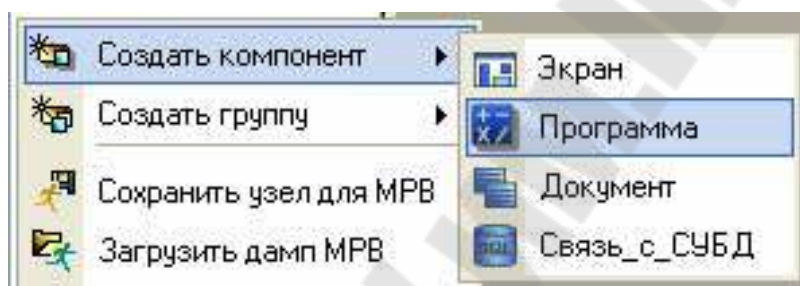


Рисунок 2.19 – Создание компонента «Программа»

Выделить компонент **Программа#1** и ПК вызвать контекстное меню, выбрав в котором ЛК пункт «**Редактировать шаблон**», перейти в режим редактирования программы (рисунок 2.20).

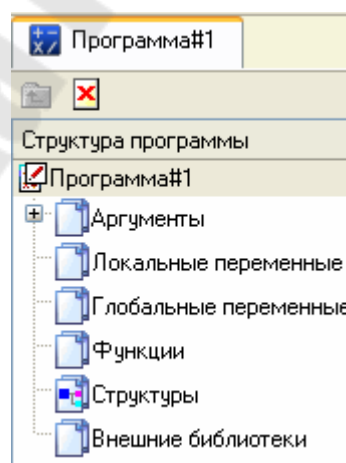

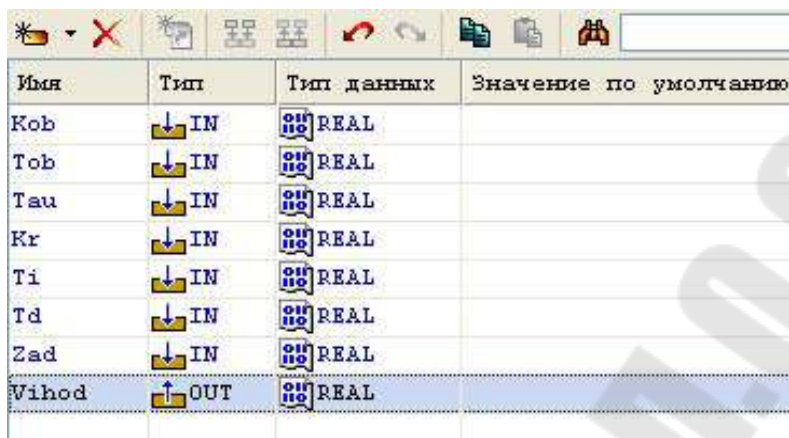


Рисунок 2.20 – Редактирование аргументов Программы

Выделением ЛК в дереве шаблона «**Программа#1**» строки «**Аргументы**» вызвать табличный редактор аргументов кнопкой  и создать в редакторе аргументов следующие аргументы программы

Kob, Tob, Tau, Kr, Ti, Td, Zad,. При этом аргументы Kob, Tob, Tau, Kr, Ti, Td, Zad должны быть типа IN, а Vihod – OUT (рисунок 2.21).



Имя	Тип	Тип данных	Значение по умолчанию
Kob	IN	REAL	
Tob	IN	REAL	
Tau	IN	REAL	
Kr	IN	REAL	
Ti	IN	REAL	
Td	IN	REAL	
Zad	IN	REAL	
Vihod	OUT	REAL	

Рисунок 2.21 – Аргументы программы

Выделить в дереве шаблона строку **Программа#1** и в открывшемся диалоге «**Выбор языка**» выбрать язык **FBD** (рисунок 2.22).

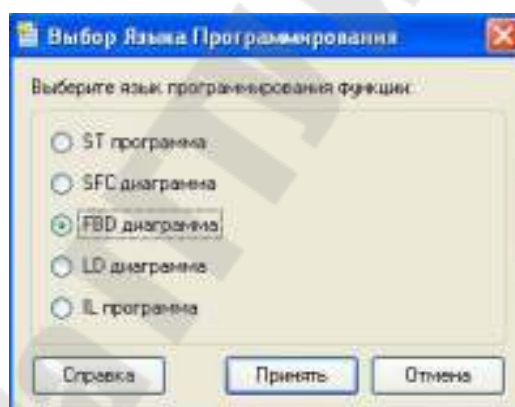


Рисунок 2.22 – Выбор языка программирования


По нажатию кнопки «**Принять**» в открывшемся окне редактора программ с объявленными переменными создать программу в соответствии с заданием (см. раздел «**Контрольное задание**»). Для выбора палитры FBD блоков необходимо ЛК мыши нажать на кнопку  после чего появится окно выбора FBD блоков (рисунок 2.23). При разработке программы верхние входы FBD блоков не используются т.к. они предназначены для изменения порядка пересчета блоков, а информационными входами, являются входы начиная со второго.



Рисунок 2.23 – Палитра FBD блоков

Для создания системы управления выберем следующие блоки: из раздела «**Арифметические Функции**» FBD блок **вычитание (X–Y)**; из раздела «**Регулирование**» FBD блок **модель объекта (OBJ)** и звено **PID (PID)**. После размещения всех блоков программа будет выглядеть следующим образом рисунок 2.24.

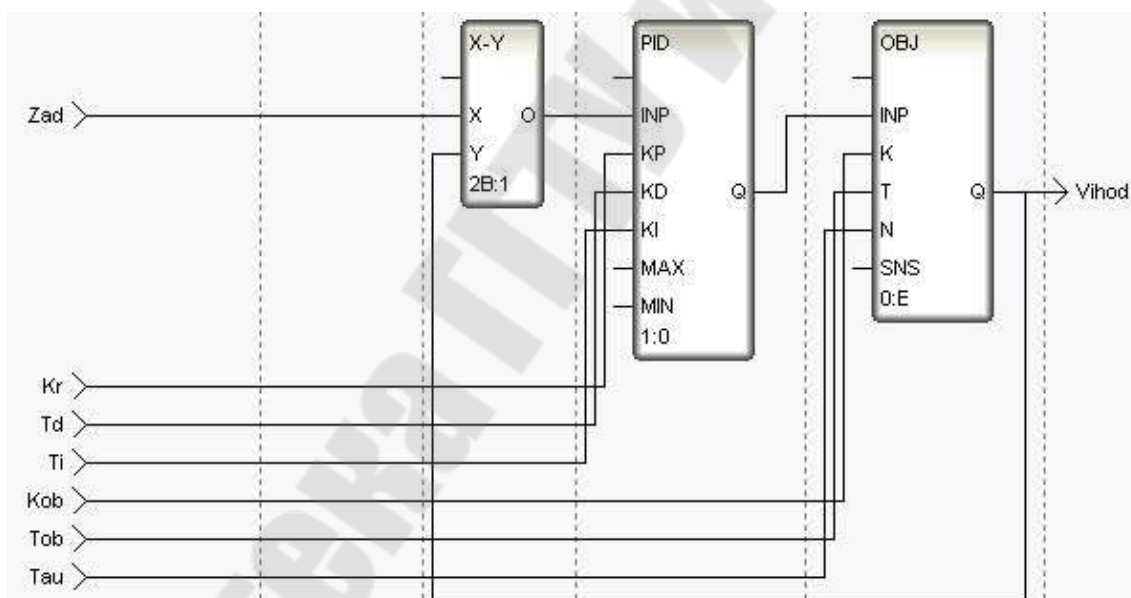




Рисунок 2.24 – Программа реализации системы управления

С помощью иконки  на инструментальной панели редактора или «горячей клавиши» F7, скомпилировать программу и убедиться в успешной компиляции в окне «**Выход**» (Output), вызываемого из инструментальной панели с помощью иконки  (рисунок 2.25).



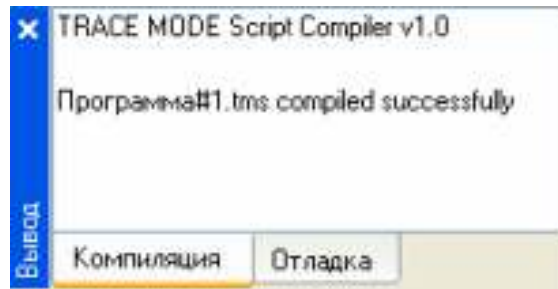


Рисунок 2.25 – Результат успешной компиляции программы

## 2.5 Привязка аргументов программы

Выполним привязку аргументов программы к атрибутам каналов. Вызвать свойства компонента **Программа#1** через контекстное меню. Выбрать вкладку «**Аргументы**».

Двойным щелчком в поле «**Привязка**» аргумента программы **У** вызвать окно настройки связи, выбрать в левом окне канал класса «**Вызов**» **Экран#1**, а в правом окне выбрать вкладку «**Аргументы**» и указать в ней аргумент **Vihod** и кнопкой «**Привязка**» подтвердить связь (рисунок 2.26).

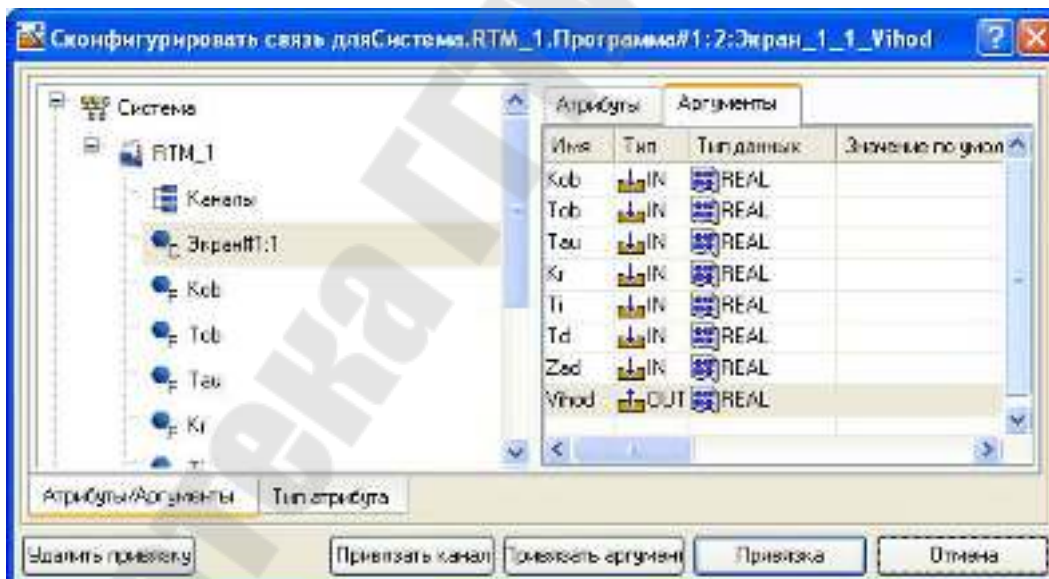


Рисунок 2.26 – Связь выходных аргументов

Аналогично в поле «**Привязка**» привязать аргументы программы к атрибутам каналов – аргументы Kob, Tob, Tau, Kr, Ti, Td, Zad к реальному значению каналов Kob, Tob, Tau, Kr, Ti, Td, Zad (рисунок 2.27).

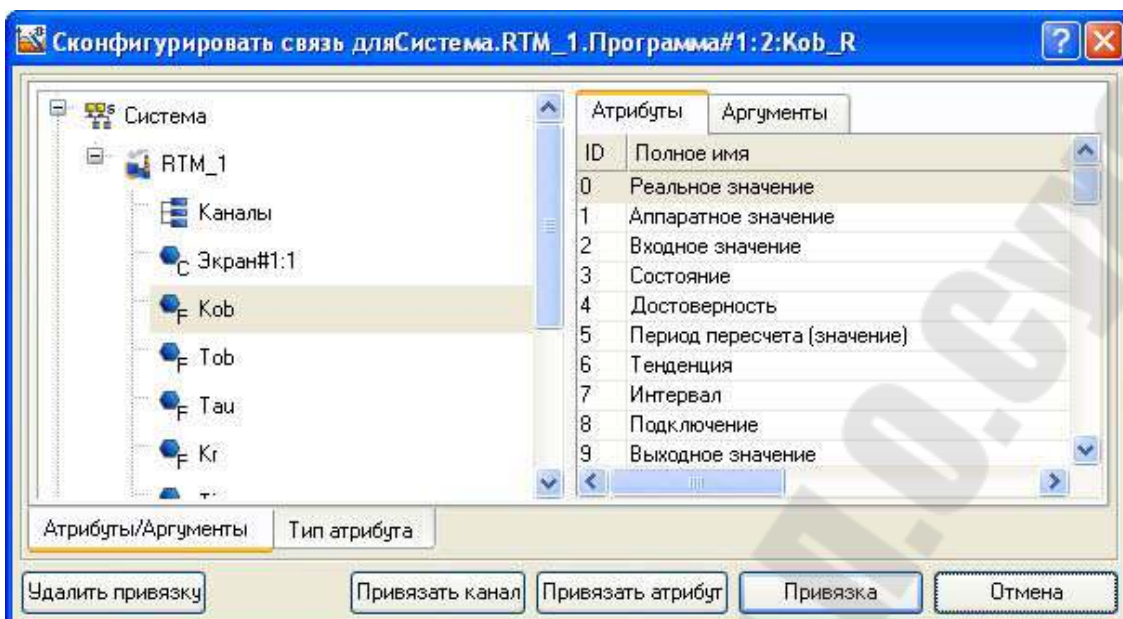


Рисунок 2.27 – Связь аргументов программы с аргументами экрана




В результате, получим следующие привязки – рисунок 2.28

Имя	Тип	Тип данных	Значение по умолчанию	Привязка
Kob_R	↓ IN	REAL		• Kob:Реальное значение (Система.RTM_1)
Tob_R	↓ IN	REAL		• Tob:Реальное значение (Система.RTM_1)
Tau_R	↓ IN	REAL		• Tau:Реальное значение (Система.RTM_1)
Kr_R	↓ IN	REAL		• Kr:Реальное значение (Система.RTM_1)
Ti_R	↓ IN	REAL		• Ti:Реальное значение (Система.RTM_1)
Td_R	↓ IN	REAL		• Td:Реальное значение (Система.RTM_1)
Zad_R	↓ IN	REAL		• Zad:Реальное значение (Система.RTM_1)
Экран_1_1_Vihod	↑ OUT	REAL		• cЭкран#1:1:Vihod (Система.RTM_1)

Рисунок 2.28 – Окончательная настройка связи

После закрыть окно свойств компонента **Программа#1**.

## 2.6 Запуск проекта

Сохранить проект с помощью кнопки . На инструментальной панели выберем иконку  и подготовим проект для запуска в реальном времени. ЛК выделим в слое «Система» узел **RTM\_1** (рисунок 2.29), а после, нажав ЛК иконку  на инструментальной панели, запустим профайлер. Запуск/останов профайлера осуществляется с по-

мощью иконки  на его инструментальной панели или клавишной комбинации **Ctrl+R**.

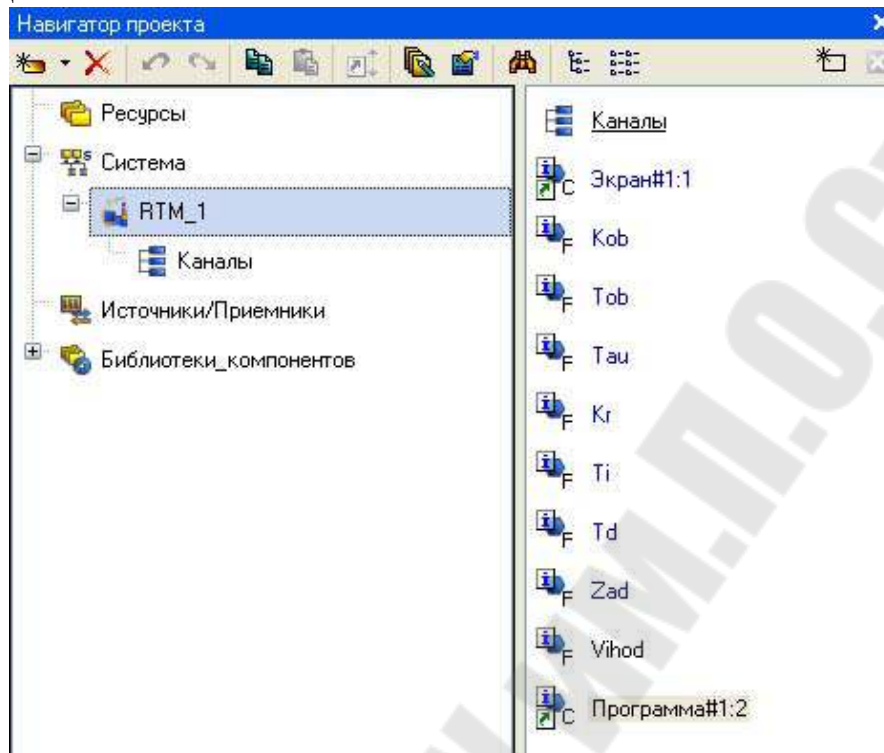


Рисунок 2.29 – Навигатор проекта

Для составления отчета необходимо в меню **Файл** выполнить команду «**Документировать проект**» полученный \*.html файл необходимо внести в отчет.

### 3 Контрольные задания

Варианты заданий выбираются по порядковому номеру в списке группы.

Вариант задания выбирается исходя из номера буквы в алфавите по инициалам студента. Номер первой буквы **Фамилии** в алфавите – **K** коэффициент усиления объекта, **Имени** – **T** постоянная времени объекта, **Отчества** – **τ** время запаздывания.

$$W(s) = \frac{K \cdot e^{-\tau s}}{T \cdot s + 1}$$

Расчет настроек ПИД регулятора производится по формулам:

$$Kr = \frac{1.2}{K_T}, Tи = \frac{2\tau}{Kr}, Tд = 0.4\tau \cdot Kr.$$

В Trace Mode вместо времени интегрирования вводится коэффициент равный  $1/T_{и}$ .

Таблица 1.1 – Данные для расчета передаточной функции

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	

#### 4 Содержимое отчета

В отчете описать основные элементы проекта, аргументы элементов операторского интерфейса, программы, созданные каналы и привязки. Привести скриншоты основного окна операторского интерфейса в режиме симуляции и **FBD** программы.

Подобрать настройки ПИД регулятора самостоятельно и сравнить переходные процессы с расчетными настройками. Выполнить несколько различных настроек (не менее 3). В отчете привести тренды.

#### 5 Контрольные вопросы

1. Языки МЭК 61131-3. Общие характеристики;
2. Принципы построения программ на языках МЭК 61131-3;
3. Язык FBD. Блок задания модели объекта OBJ;
4. Язык FBD. Блок ПИД регулятора.



## Список литературы

1. Trace Mode 6. Руководство пользователя. Интегрированная SCADA/HMI-SOFTLOGIC-MES-EAM-HRM-система для разработки АСУ ТП, АСКУЭ и систем управления производством. 8-е изд. – М.: «AdAstra Research Group», Ltd, 2006. – 619 с. ил.
2. Руководство пользователя TRACE MODE 6 БЫСТРЫЙ СТАРТ, AdAstra Research Group, Ltd. Москва. 2008. – 168с.
3. Руководство пользователя TRACE MODE 6 Том 1, AdAstra Research Group, Ltd. Москва. 2008. – 518 с.
4. Руководство пользователя TRACE MODE 6 Том 2, AdAstra Research Group, Ltd. Москва. 2008.– 534 с.

**Литвинов Дмитрий Александрович**  
**Ковалев Алексей Викторович**

**ПРОГРАММИРОВАНИЕ ПРОМЫШЛЕННЫХ  
МИКРОКОНТРОЛЛЕРОВ НА ЯЗЫКАХ  
СТАНДАРТА IEC 61131-3**

**Практикум**  
**по дисциплине «Локальные информационные системы»**  
**для студентов специальности 1-36 04 02**  
**«Промышленная электроника»**  
**специализации 1-36 04 02 02**  
**«Техника и средства электронной связи»**  
**дневной формы обучения**

**Часть 2**

Подписано к размещению в электронную библиотеку  
ГГТУ им. П. О. Сухого в качестве электронного  
учебно-методического документа 05.04.16.

Рег. № 45Е.

<http://www.gstu.by>