

**ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ**  
Государственное образовательное учреждение высшего профессионального образования  
**«ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

---

**А. А. Мезенцев, В. М. Павлов, К. И. Байструков**

**ТЕХНИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ  
ЛАБОРАТОРНОГО КОМПЛЕКСА «ОРГАНИЗАЦИЯ  
ПУЛЬТОВ УПРАВЛЕНИЯ СОВРЕМЕННЫХ АСУ ТП»**

Учебное пособие

Издательство  
Томского политехнического университета  
Томск, 2010

УДК 681.58:004.353(075)

М442

**Мезенцев А. А.**

М442 Техническое и программное обеспечение лабораторного комплекса «Организация пультов управления современных АСУ ТП»: учебное пособие / А. А. Мезенцев, В. М. Павлов, К. И. Байструков. – Томск: Изд-во Томского политехнического университета, 2007. – 128 с.

Учебное пособие посвящено описанию технических характеристик и принципов организации аппаратных и программных составляющих пультов управления исследовательских установок и объектов промышленного назначения.

Учебное пособие состоит из 3 разделов. Первый раздел включает описание технического обеспечения современного программно-технического комплекса пультовой. Второй раздел включает описание технологии проектирования прикладного программного обеспечения пультов управления с использованием SCADA – систем в качестве инструментального обеспечения. Третий раздел включает в свой состав описание технологий САПР проектирования посредством современного перспективного программного комплекса Matlab, ориентированного на математическую обработку высокоинформативных потоков экспериментальных данных.

Пособие подготовлено на кафедре «Электроника и автоматика физических установок» ТПУ и предназначена для студентов очного обучения специальности 140306.

УДК 681.58:004.353(075)

Рекомендовано к печати Редакционно-издательским советом  
Томского политехнического университета

*Рецензенты*

Доктор технических наук, профессор, заведующий кафедрой  
«Информационно измерительной техники» ТУСУРа

*А. А. Светлаков*

Кандидат физико-математических наук,  
научный сотрудник Института сильноточной электроники СО РАН

*С. А. Попов*

© Томский политехнический университет, 2010

© Оформление. Издательство Томского политехнического университета, 2010

## СПИСОК СОКРАЩЕНИЙ

- ТОУ – технологический объект управления
- УТС – установки по управляемому термоядерному синтезу;
- ЯТЦ – ядерно-топливный цикл;
- ТОКАМАК [ТОроидальная КАмера и МАгнитная Катушка] – комплекс технических и программных средств, необходимых для получения управляемого термоядерного синтеза;
- УГК – удалённый графический контроллер;
- АСУ – автоматизированная система управления;
- ВСО – высокопроизводительная станция оператора АСУ;
- БДРИ – база данных результатов испытаний;
- MDC – Multiple Display Control – программное обеспечение управления видеостеной;
- GUI (ГПИ) Graphic User Interface – Графический Пользовательский Интерфейс;
- ГЭ – графический элемент;
- MCD [Matlab Compiler Driver] – программа управления процессом компиляции в Matlab;
- ПКП – панель коллективного пользования;
- T.M.D.S. – Transition Minimized Differential Signaling – дифференциальная передача сигналов с минимизацией перепадов уровней;
- JIT – Just-in-time compilation (также известна как dynamic translation) – компиляция «на лету» — это технология увеличения производительности программных систем, использующих байт-код, путём трансляции байт-кода в машинный код непосредственно во время работы программы;
- UDF – User Defined Function – библиотека пользовательских функций Simulink;
- FBD – Function Block Diagram – язык функциональных блоков. Визуально-ориентированный язык программирования, применяемый в SCADA системах для построения каналов управления;
- IL – Instruction List – язык инструкций. Скриптовый язык программирования, применяемый для разработки в SCADA системах. В том числе для разработки FBD блоков.
- TIC – Time Input Clock – оператор Matlab инициализирующий запуск системного счётчика;
- TOC – Time Output Clock – оператор Matlab инициализирующий останов системного счётчика и расчёт процессорного времени.

## СОДЕРЖАНИЕ

|  |    |
|--|----|
| ВВЕДЕНИЕ .....   | 6  |
| 1. ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СОВРЕМЕННОГО ПРОГРАММНО-ТЕХНИЧЕСКОГО КОМПЛЕКСА ПУЛЬТОВОЙ .....  | 7  |
| 1.1. Функции программно-технического комплекса пультовой.....  | 7  |
| 1.2. Состав программно - технического комплекса пультовой.....   | 7  |
| 1.3. Технические характеристики программно-технического комплекса пультовой .....  | 9  |
| 1.4. Описание модулей программно-технического комплекса пультовой .....  | 11 |
| 1.4.1. Панель коллективного пользования .....  | 11 |
| 1.4.2. Система управления графическими потоками .....  | 13 |
| 1.4.3. Пультовая секция .....  | 17 |
| 1.4.4. Сервер видеонаблюдения .....  | 18 |
| 1.5. Интерфейс DVI .....   | 21 |
| 1.5.1. Что такое DVI .....   | 21 |
| 1.5.2. Разновидности DVI .....   | 22 |
| 1.5.3. Ограничения при передаче DVI сигнала.....   | 23 |
| 2. Технологии проектирования прикладного программного обеспечения пультов управления с использованием SCADA-систем в качестве инструментального обеспечения..... | 24 |
| 2.1. SCADA-системы, назначение, структура, основные функции .....  | 24 |
| 2.2. Общие сведения о системе TRACE MODE, структура проекта .....  | 25 |
| 2.3. Каналы прохождения информации в системе TRACE MODE .....  | 28 |
| 2.3.1. Обработка данных в канале.....  | 30 |
| 2.3.2. Задание конфигурации канала. ....   | 31 |
| 2.4. Разработка графического интерфейса оператора.....   | 34 |
| 2.5. Особенности запуска монитора реального времени TRACE MODE .....   | 43 |
| 2.6. Создание распределенных систем управления .....   | 45 |
| 2.6.1. Организация сетевого взаимодействия между МРВ.....  | 46 |
| 2.6.2. Обмен с базами данных .....   | 48 |
| 2.6.3. Обмен между приложениями по протоколу DDE.....  | 50 |
| 2.6.4. Обмен между приложениями по протоколу NetDDE.....   | 54 |
| 2.6.5. Обмен между приложениями по протоколу OPC .....   | 58 |
| 2.7. Архивирование и документирование, система архивов TRACE MODE ...  | 60 |
| 2.7.1. Отчет тревог узла .....   | 61 |
| 2.7.2. Архивы SIAD .....   | 63 |
| 2.7.3. Индивидуальные (поканальные) архивы в памяти .....  | 63 |
| 2.8. Средства программирования микропроцессорных контроллеров в TRACE MODE 6 .....   | 65 |
| 3. Проектирование ПО с помощью программного комплекса Matlab .....   | 76 |
| 3.1. Технологии программирования GUI в Matlab.....   | 77 |
| 3.2. Алгоритмы программирования и верификации в среде Matlab .....   | 80 |
| 3.3. Архитектура М программы с использованием ГПИ.....   | 85 |
| 3.4. Проектирование ГПИ (GUI) в Matlab.....  | 89 |
| 3.5. Проектирование сложных (составных) графических элементов в среде Matlab.....  | 97 |

|         |  |     |
|---------|--|-----|
| 3.6.    | Комбинированный способ проектирования и редактирования GUI в Matlab.....       | 100 |
| 3.7.    | Использование компилятора в среде Matlab .....                                 | 101 |
| 3.8.    | Проблемы, возникающие при компилировании кода М .....                          | 107 |
| 3.9.    | Внешнее исполнение программ .....  | 107 |
| 3.9.1.  | Команды внешнего запуска приложений.....                                       | 107 |
| 3.9.2.  | Режим выполнения команд Windows .....  | 108 |
| 3.9.3.  | Режим выполнения команд Unix .....   | 108 |
| 3.9.4.  | Режим эмуляции команд DOS.....   | 109 |
| 3.10.   | Работа с таймером в Matlab .....   | 109 |
| 3.11.   | Тестирование возможностей Matlab по точности выполнения задач по таймеру ..... | 112 |
| 3.12.   | Визуальное, блочно-ориентированное проектирование в Simulink .....             | 114 |
| 3.12.1. | Интеграция пакета Simulink с системой Matlab .....                             | 114 |
| 3.12.2. | Технологии проектирования в среде Simulink .....                               | 116 |
| 3.12.3. | Пользовательские функции в Simulink .....                                      | 119 |
| 3.12.4. | S функция в виде М кода .....  | 120 |
| 3.12.5. | Состав S-функция в Simulink .....  | 124 |
|         | ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ.....  | 125 |
|         | Библиографический список.....  | 128 |

## ВВЕДЕНИЕ

Управление сложным распределённым технологическим объектом (ТОУ) возможно только при мощной централизации функций управления в пультовой. Прошлые десятилетия было возможным построить пульт управления на основе только технических средств и разработок, поскольку электронно-вычислительные машины (ЭВМ) были дороги и не перспективны в области решения задач управления. Органы управления пульта были построены на основе переключателей и кнопок, а вывод информации производится посредством ламп на мнемосхемах, которые располагались на мнемощитах. Мнемощит, в свою очередь, отражал статическую картину реальной технологической цепочки или конструкции.

Современная электроника позволяет создавать дешёвые и очень эффективные процессоры для ЭВМ, миниатюрные электронные схемы, позволяющие строить как распределённые вычислительные комплексы, так и локальные микропроцессорные системы управления. Так например на  $107 \text{ мм}^2$  уместается 820 млн. транзисторов в процессоре Pentium от Intel, а самый мощный суперкомпьютер Blue Gene/P от компании IBM преодолел рубеж производительности в 1 петафлопс (то есть 1 квадриллион операции с плавающей точкой в секунду). Колоссальную производительность обеспечивают 294912 четырехъядерных процессора IBM PowerPC 450 с тактовой частотой 850 МГц. Накопители на магнитных дисках достигли ёмкости в 1Тб, а скорость передачи данных по радиоканалам связи достигла 1 Гбит/с в мобильных сетях 4го поколения. Эффективные технологии производства тонкоплёночных транзисторов позволили строить активные LCD TFT матрицы с размером диагонали до 108"(270 см)<sup>1</sup>

С применением новейших технологий, становится возможным построить современный пульт управления с учётом требований современного производства. Таким образом, современная пультовая превращается в сложный программно-технический комплекс.

<sup>1</sup> Японская компания Sharp в 2007 году представила первую в мире матрицу TFT размером 108"

# **1. ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СОВРЕМЕННОГО ПРОГРАММНО-ТЕХНИЧЕСКОГО КОМПЛЕКСА ПУЛЬТОВОЙ**

## **1.1. Функции программно-технического комплекса пультовой**

Опишем функции программно-технического комплекса пульта управления для установок по управляемому термоядерному синтезу (УТС) и установок ядерно-топливного цикла (ЯТЦ), в качестве прототипа используется пульт ведущего физика и пульт общего управления для установки по УТС типа ТОКАМАК.

Пульт необходим для макетирования рабочего места оператора для установок по управляемому термоядерному синтезу типа ТОКАМАК или других ядерно-физических установок или физических установок атомной отрасли и должен выполнять следующие задачи:

1. Выполнение математического моделирования процессов физики плазмы посредством пакета матричных математических вычислений Matlab 6.5 и разработанных на кафедре ЭАФУ специальных программ;
2. Выполнение доступа к базе данных результатов испытаний (БДРИ) на основе DB PosgreeSQL Server для чтения/записи экспериментальных данных ТОКАМАКов;
3. Выполнение обработки экспериментальных;
4. Управление технологическим объектом в реальном масштабе времени посредством SCADA системы Trace Mode 6 и монитора реального времени с разработанной на кафедре ЭАФУ мнемосхемы технологического процесса и программы, написанной на языке FBD;
5. Вывод и конфигурирование больших объёмов графической информации на панель коллективного пользования (ПКП);
6. Подготовку специалистов по разработке приложения для АСУ ТП.

## **1.2. Состав программно - технического комплекса пультовой**

Программно-технический комплекс состоит из 4 основных блоков:

1. Панель коллективного пользования;
  - 1.1. 4 LCD панели Samsung TFT 40" 400PXn;
  - 1.2. Крепление специализированное настенное с функцией наклона – Ergotron;

2. Система управления графическими потоками, в составе которой находится удалённый графический контроллер (УГК);
  - 2.1. УГК ХТО-F1400F – удалённый графический контроллер;
  - 2.2. ХТОА-FP33LPAF – оптический PCI адаптер;
  - 2.3. САВ-ХТО-5F – многомодовый дуплексный fiber-optic кабель с Dual-LC разъёмами;
  - 2.4. Четыре кабеля соединительных SVGA card DVI-D (19М-19М), 5 м;
  - 2.5. Два кабеля соединительных SVGA card DVI-D (19М-19М), 1 м;
3. Пультовая секция со специализированной функциональной клавиатурой и высокопроизводительной станцией оператора пульта (ВСО);
  - 3.1. Рабочий стол оператора;
  - 3.2. Сидение мягкое, с подлокотниками, антистатическое;
  - 3.3. 2 монитора LCD TFT 19”;
  - 3.4. Высокопроизводительная станция оператора пульта;
  - 3.5. Специализированная функциональная клавиатура;
    - 3.5.1. Плата дискретного ввода вывода Advantech PCI 1750;
4. Сервер видеонаблюдения;
  - 4.1. Сетевой видео-аудиосервер «ОКО-Архив-Лайт»;
  - 4.2. Камера видеонаблюдения VZP-734 Super HAD CCD;
  - 4.3. Кабель соединительный RG59+2П с разъёмами Cr-50;
  - 4.4. Блок питания БП1-Ам.

В состав стенда входит следующее программное обеспечение:

1. SCADA система проектирования Trace Mode 6.05.1;
2. Монитор реального времени Trace Mode 6.05.1;
3. Среда матричных математических вычислений Matlab 6.5;
4. Windows XP Pro;
5. ПО видеостены MDC;
6. Система САПР проектирования (подготовки документации на всех стадиях разработки проекта) САПР Альфа-СА;
7. ПО визуализации и обработки данных АЦП DAQWorkADC;
8. ПО визуализации и обработки экспериментальных данных ТОКАМАКов DAQViewer;
9. Пакет программирования продукции National Instruments: Full Development System 778800-03; Base Package 778801-03; Starter Kit 778802-03;
10. ПО УГК Matrox Power Space 2.0;
11. ПО видеосервера «Тайфун».



### 1.3. Технические характеристики программно-технического комплекса пультовой

#### LCD панель Samsung Sync Master 400PXn



- диагональ – 40".
- разрешение, пикс. – 1366×768.
- габариты, мм – 971×582×118.
- вес, кг – 23.5.
- яркость, кд/м<sup>2</sup> – 500.
- контрастность – 1200:1.
- время отклика, мс – 8.
- углы обзора – 178°/178°.
- цветность, бит – 32.
- тип видеосигнала – аналоговый RGB, DVI dual-link (TMDS-код), компонентный, композитный, S-video.
- потребляемая мощность, Вт – 230.
- блок питания – встроенный 220 В.
- ОС – Windows XP Embedded.
- канал синхронизации и управления – RS-232C.
- порты в/в (USB 2.0, LAN), шт. – 3, 1.

#### Крепление настенное для LCD панели Ergotron TM



- монтаж – настенный.
- стандарт крепления – VESA 600×450.

#### Удалённый графический контроллер MATROX XTO-F1400



- длина канала связи, м – более 250.
- количество подключений, шт. – 4.
- видеоинтерфейс – DVI-I.
- разрешающая способность, пикс. – 1600×1200.
- частота смены кадров, Гц – 60.
- порты в/в (USB 2.0, аудио, duplex fiber optic), шт. – 6, 3, 1.
- габариты, мм – 300×290×142.

### **Fiber Optic адаптер ХТ0А-FP33LPAF**



- габариты, мм – 119×63.
- интерфейс локальной шины – PCI.
- интерфейс подключения периферийных устройств – fiber-optic (разъём dual LC).

### **DVI разветвитель ATEN VS-162 VIDEO SPLITTER**



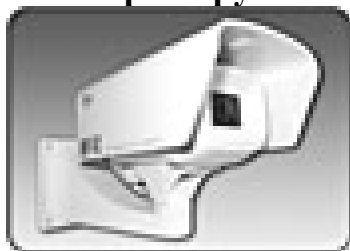
- напряжение питания: 220 В.
- тип разъёмов: DVI-I.
- пропускная способность, Гбит/с – 3.
- разрешающая способность, пикс. – 1600×1200.
- Частота смены кадров, Гц – 60.

### **Сетевой видео-аудиосервер «ОКО-Архив-Лайт»**



- количество видеовходов, шт. – 2 (BNC, 75 Ом).
- количество аудиовходов, шт. – 2.
- количество входов датчиков, шт. – 2.
- количество входов для подключения исполнительных устройств, шт. – 2.
- разрешение кодирования видео, пикс. – 758×576.
- скорость видеозаписи, кадр./с, – 25.
- алгоритм сжатия изображения – H.263.
- алгоритм сжатия фотографии – JPEG.
- потребляемая мощность, Вт – 8.
- информационные интерфейсы – RS-485, USB 2.0.
- питание, В – 220, внешний блок питания.

### Камера наружного видеонаблюдения VZP-734 Super HAD CCD



- фотоприёмник – ПЗС 1/4" .
- разрешение видео (PAL, NTSC), пикс. – 752×582, 768×492.
- минимальная освещённость на объекте при отношении сигнал/шум 10 люкс – 1.0(F2.0).
- напряжение источника питания, В – +8 ... +16.
- потребляемый ток, мА – не более 95.
- отношение сигнал/шум, – дБ – 44.
- габаритные размеры, мм – 125×95×235.

## 1.4. Описание модулей программно-технического комплекса пультовой

В соответствии с градацией, указанной в подпункте 1.2 рассмотрим более детально составные узлы программно-технического комплекса пультовой.

### 1.4.1. Панель коллективного пользования

Панель коллективного пользования представляет собой сборку из 4-х LCD TFT дисплеев с диагональю 40 " (Samsung 400PXn LS40BHPNS). Данные LCD-дисплеи представляют собой специализированный комплекс, ориентированный на построение видеостен. В состав каждого дисплея входит контроллер, работающий под управлением ОС Windows XP Embedded. Подобное решение, позволяет управлять выводом графических данных на экран с различной компоновкой дисплеев. В данном лабораторном комплексе используется режим 2×2 (рис. 1.1). Конфигурирование (настройка) дисплеев производится с помощью ПО MDC, которое поставляется вместе с LCD-дисплеями.

Контроллер дисплея позволяет работать LCD сборке в двух режимах: в режиме функциональных областей и в режиме видеостены. В первом случае, на каждом дисплее отображается отдельная картинка. Во втором случае на всех 4-х панелях отображается одна картинка, при этом сама LCD-сборка с точки зрения управления считается единым целым экраном.

Первый вариант управления используется при подготовке к эксперименту на физической установке, когда каждая функциональная об-

ласть (LCD-дисплей) отображает параметры конкретно технологической системы или всей установки в целом (по требованию оператора). Режим видеостены используется в период разряда, когда, например, требуется проследить динамику горения плазмы.

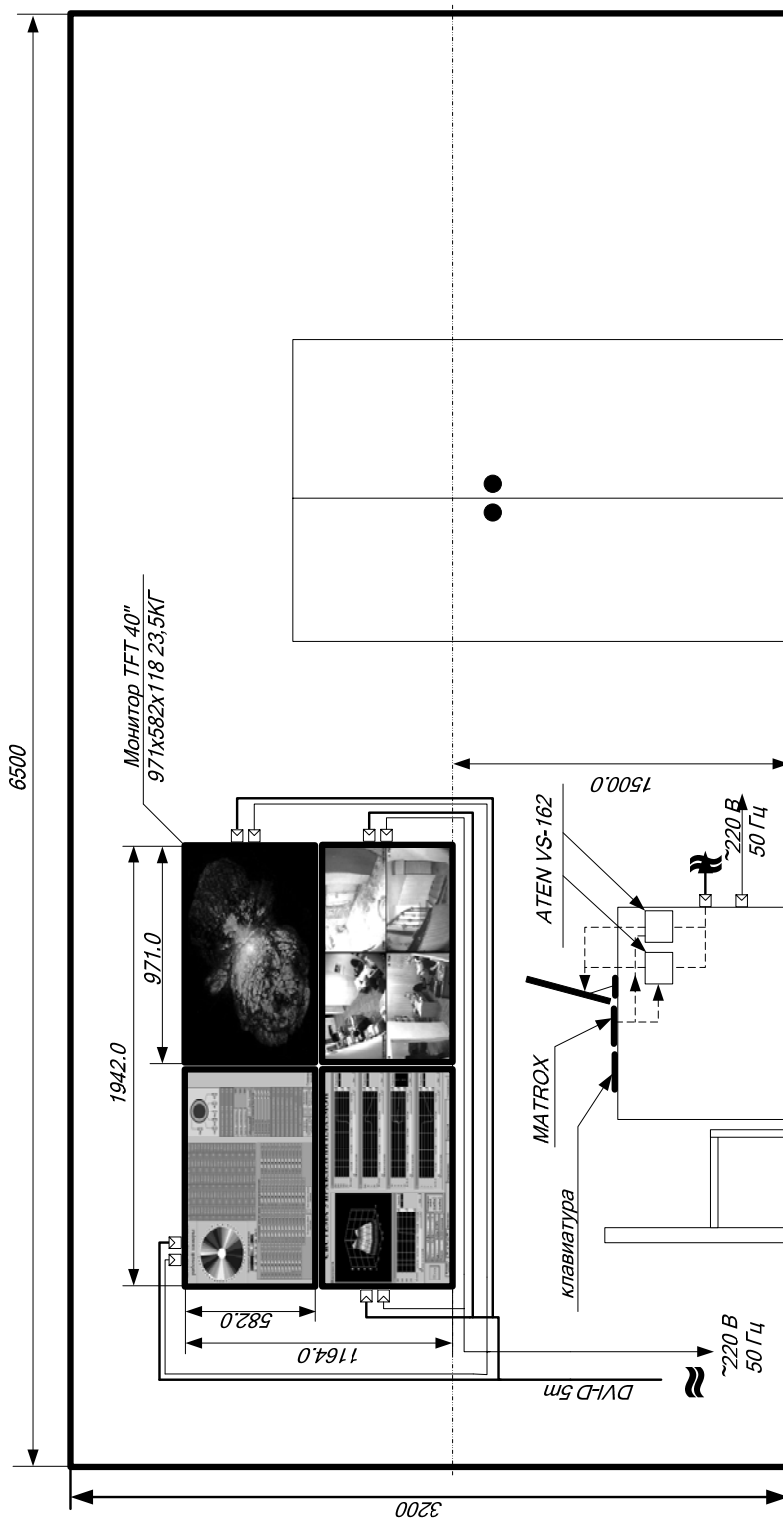


Рис. 1.1. Структура ПКП (лабораторный комплекс, вид сбоку)

Управление и конфигурирование LCD дисплеев производится по интерфейсу RS-232C, в соответствии со следующей архитектурой информационного канала (рис. 1.2).

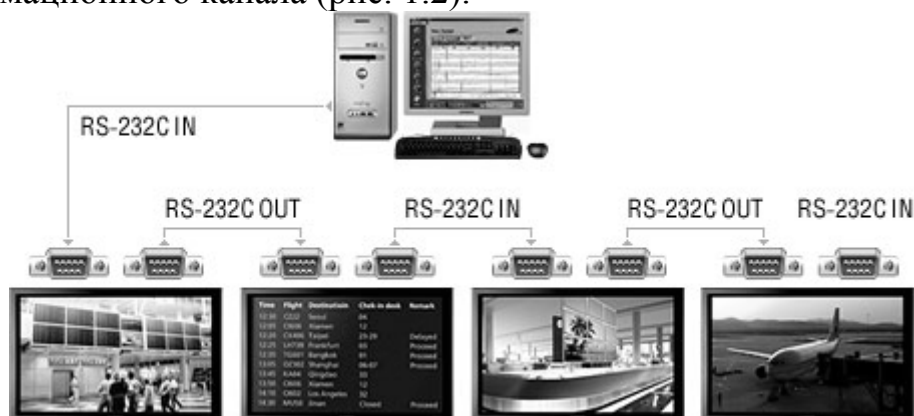


Рис. 1.2. Архитектура канала управления LCD дисплеями

Передача видеоданных производится по каналу DVI-D. Подключение LCD дисплеев к управляющей системе производится посредством разъемов DVI-D.

#### 1.4.2. Система управления графическими потоками

Система управления графическими потоками строится на основе удалённого графического контроллера (УГК).

Современная вычислительная техника способна выполнять широкий спектр задач. Но на какие бы ухищрения не шли инженеры, техника не может работать бесшумно, не выделять тепло, не излучать вредные для человека импульсы. Все эти факторы говорят о том, что комфортные условия для работы человека и машины различны. Логичным решением становится разнесение географически машинной и человеческой составляющей.

Подобную задачу можно решить, используя оборудование компании Matrox. На базе 4-х канального графического контроллера, был организован оптический канал удаленного управления выводом видеоданных. Особенностью решения является выделение пассивной составляющей системы управления, которая не требует охлаждения и абсолютно не шумит, за исключением движущихся частей. Рассмотрим структуру системы на рис. 1.3.

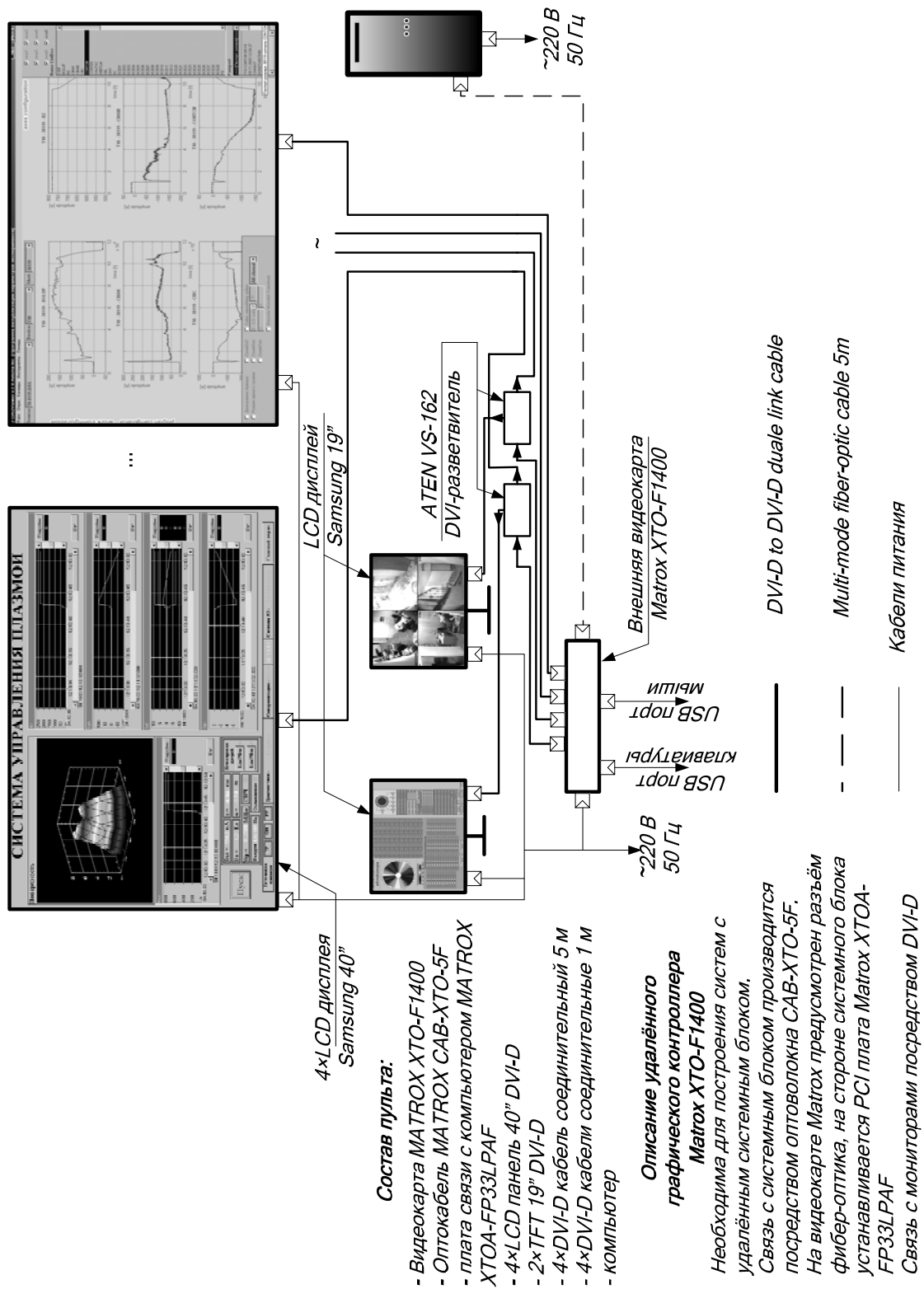


Рис. 1.3. Архитектура системы удалённого управления графическими потоками

В составе лабораторного комплекса используется 6 устройств вывода видеоданных:

- LCD-дисплеи видеостены (ПКП), шт. – 4;
- локальные LCD-дисплеи на столе оператора пульта, шт. – 2.

Видеоконтроллер Matrox XT-F1400 (УГК) позволяет управлять сразу четырьмя каналами видео по интерфейсу DVI, с пропускной способностью каждого канала 1600×1200 пикс., что при 10 битном T.M.D.S. кодировании и частоте развёртки картинки 60 Гц составляет  $1600 \times 1200 \times 10 \times 60 = 1152$  Мбит/с.

Поскольку устройств вывода изображения 6, а каналов передачи видеоданных Matrox всего 4, то необходимо использовать дополнительное оборудование. В случае лабораторного комплекса, локальные мониторы, которые расположены непосредственно на пультовой секции оператора пульта, выполняют роль дублирования некоторых экранов на ПКП. Таким образом, используем в двух каналах DVI, DVI-разветвители ATEN VS-162. Схема построения каналов передачи видеоданных показана на рис. 1.3.

Ранее упоминалось, что Matrox – это система удалённого управления видеопотоками. На этом основании в области пультовой секции оператора пульта располагается лишь пассивная составляющая графической системы. Высокопроизводительная станция АСУ, находится удалённо в благоприятном для работы электроники месте (например, в стойке, в аппаратном зале). Связь Matrox и ВСО производится по многомодовому оптическому волокну с dual LC вилками. В качестве приёмника сигнала в ВСО на шине PCI установлен оптический PCI адаптер ХТОО-FP33LPAF, который производит преобразование управляющих воздействий Matrox и передачу графических данных на ПКП. В силу вышесказанного, отметим, что вся система управления видеоданными строится на основе 4-х элементов:

1. Matrox XT-F1400 – УГК;
2. ХТОО-FP33LPAF – оптический PCI адаптер;
3. САВ-ХТОО-5F – многомодовый дуплексный fiber-optic кабель длиной 5 м (duplex, multi-mode fiber-optic cable with Dual-LC connectors);
4. SVGA card DVI-D (19M -19M) кабели соединительные для подключения к ПКП (4 шт. – 5 м, 2 шт. – 1 м).

Как было отмечено, УГК участвует не только в передаче видеоданных от ВСО к ПКП, но и в передаче управляющих воздействий от УГК к ВСО. Для этого на панели Matrox, используются разъёмы USB 2.0, для подключения периферийного оборудования (манипулятор мышь и

универсальная клавиатура). Лицевая панель УГК Matrox показан на рис. 1.4.



*Рис. 1.4. Вид УГК Matrox XTO-F1400*

Структура технического комплекса на основе УГК Matrox в сборе показана на рис. 1.5.



*Рис. 1.5. Вид системы удалённого графического управления в сборе*

Для управления выводом информации на ПКП и работы с четырьмя дисплеями используется специальное ПО Matrox Power Space 2.0, поставляемое вместе с УГК Matrox. При подключении к УГК Matrox нескольких мониторов, переход на каждый левостоящий монитор осуществляется переводом мыши в левую границу экрана текущего монитора. Для перевода курсора мыши в область правостоящего монитора необходимо переместить курсор мыши к правой границе текущего монитора. На каждом мониторе может запускаться отдельная задача, которая выполняется независимо от текущей задачи, либо несколько мониторов могут отображать выполнение текущей задачи (одно окно программы растянуто на несколько мониторов). На каждом рабочем столе каждого из подключенных к УГК мониторов, можно запустить новый рабочий стол. Переключение между рабочими столами может осуще-



ствляться сочетанием клавиш Ctrl+№\_экрана либо через интерфейс программы Matrox Power Space 2.0.

### 1.4.3. Пультовая секция

Пультовая секция оператора представляет собой стол с двумя LCD TFT мониторами (19"), функциональной клавиатурой наборной (поле 4x4 – 16 кнопок управления), клавиатуры универсальной и кресла оператора пульта. На рис. 1.6, показан сборочный чертёж пультовой секции.

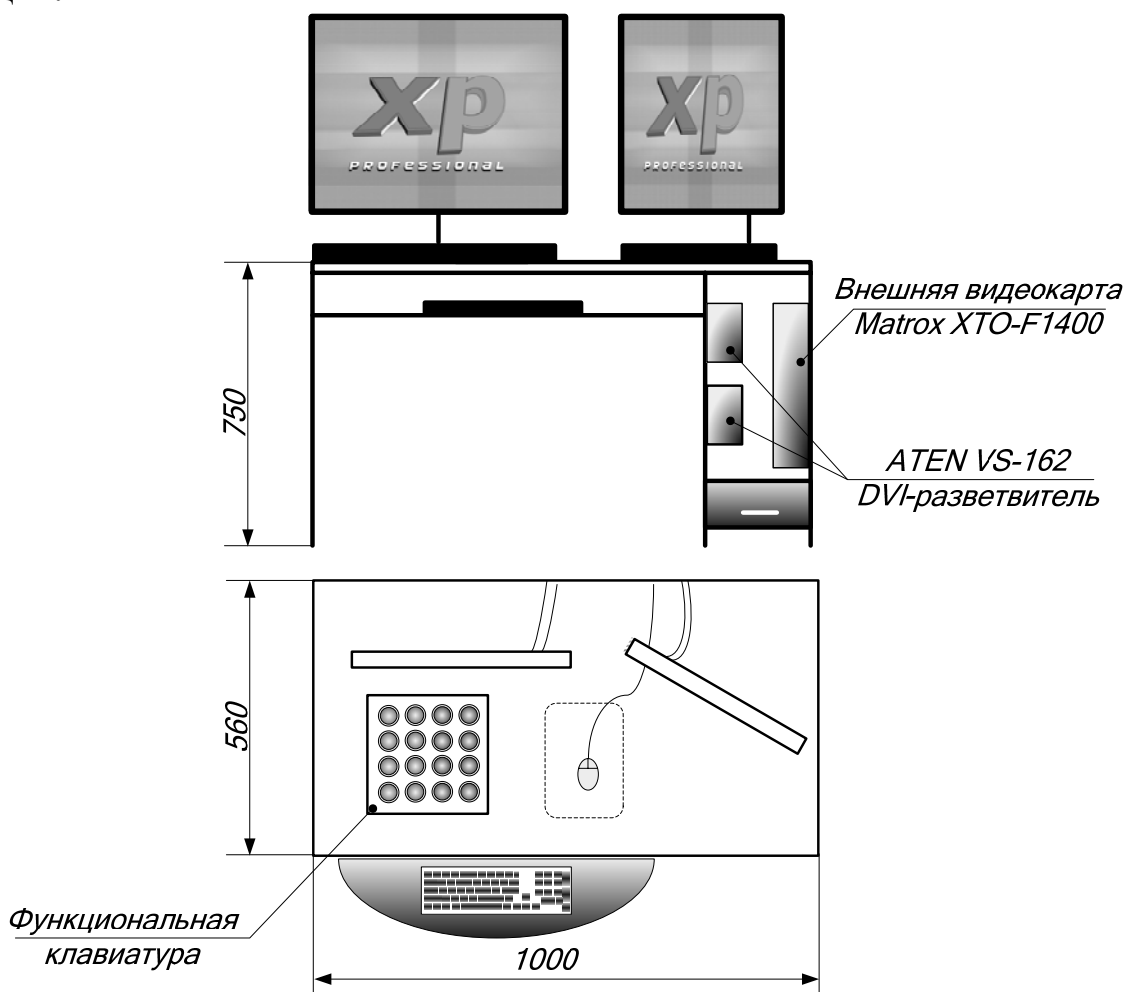


Рис. 1.6. Сборочный чертёж пультовой секции оператора пульта

В соответствии с рисунком 1.6, УГК и DVI разветвители крепятся на пультовой секции. ВСО находится вне пультовой секции и располагается в стойке щитовой. В рамках лабораторного комплекса, для подключения УГК к ВСО используется кабель длиной 5 м, следовательно расстояние, на которое можно удалить ВСО не более 5 м. На промышленных объектах это расстояние может быть увеличено до 250 м, а оп-

тический кабель прокладывается под фальшполом. Поскольку передать DVI сигнал далее 5...7 м по проводному кабелю невозможно (требуется реклоинг из-за потери фронта сигнала), то передача сигнала на расстояния более сотни метров производится по оптическому волокну (затухание сигнала ниже, чем по проводному каналу), в начале и в конце которого располагаются концевые оптические преобразователи.

На столе оператора пульта располагается функциональная клавиатура, с помощью которой выполняется управление технологическим оборудованием. Соединение функциональной клавиатуры с ВСО производится параллельно по шине проводов с платой дискретного ввода/вывода Advantech PCI 1730.

#### 1.4.4. Сервер видеонаблюдения

Для наблюдения за помещениями, в которых работает технологическое оборудование, а так же для наблюдения за работой самого технологического оборудования во время экспериментов или технологического цикла, используется система видеонаблюдения и архивирования.

В данном лабораторном комплексе используется система видео-аудио наблюдения «ОКО-Архив-Лайт», которая представляет из себя малоканальный сервер-хранилище с интерфейсными разъёмами для подключения видеокамер, датчиков движения, микрофонов и сигналов управления исполнительных механизмов. Структура видеосервера представлена на рис. 1.7.

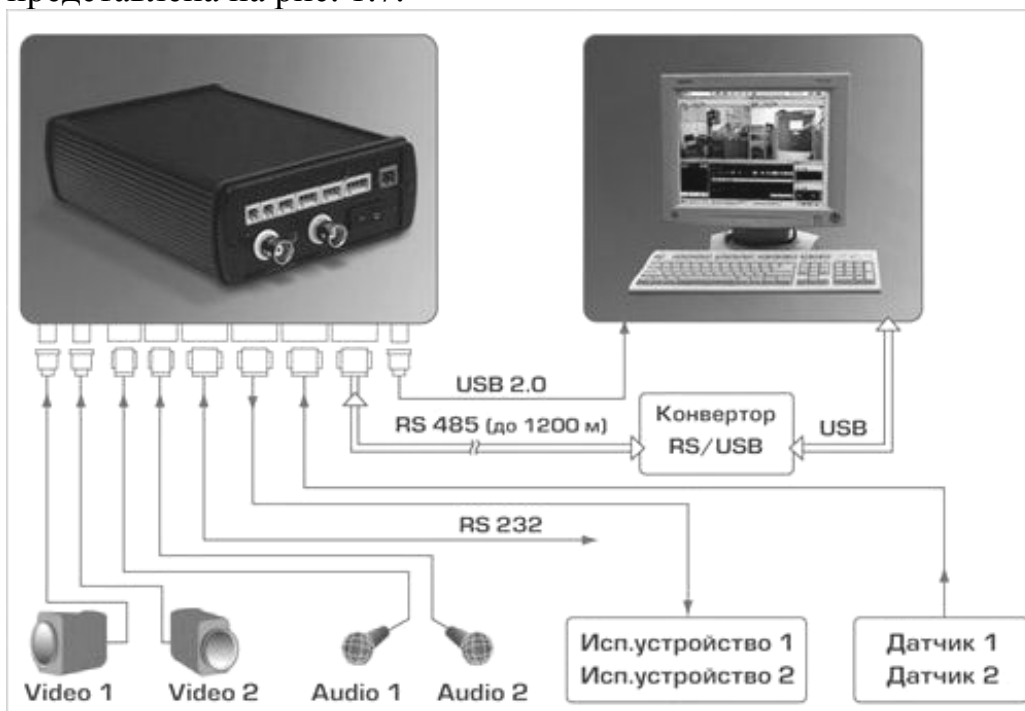


Рис. 1.7. Структура видео-аудио сервера «ОКО-Архив-Лайт»

В составе лабораторного комплекса используется малоканальный варианта сервера ОКО – «ОКО-Архив-Лайт» от производителя ЗАО «ЭВС». Сервер включает в свой состав накопитель на магнитных дисках (винчестер), расположенный в специальном лотке, позволяющем быстро монтировать и демонтировать архив данных. Не смотря на малые размеры, ОКО сервер обладает большими интерфейсными и функциональными возможностями (рис. 1.7):

- Подключение двух видеокамер.
- Два канала аудиозаписи.
- Два порта подключения исполнительных устройств.
- Два порта для подключения датчиков.

В лабораторном комплексе используется только один канал видео, остальные порты не задействованы.

Для того, чтобы просмотреть архивные видеоданные, ОКО архив содержит два типа каналов передачи данных:

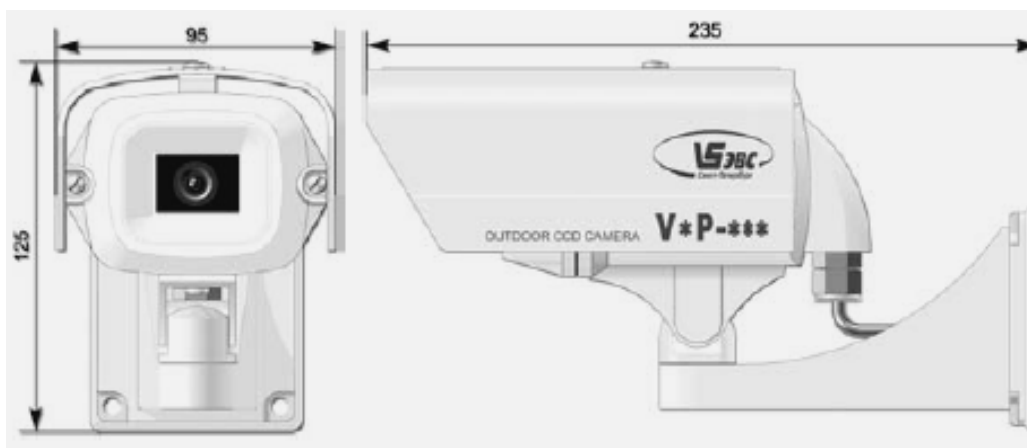
1. RS-485 (подразумевает передачу данных на большие расстояния, до 1200 м со скоростью до 1 Мбит/с).
2. USB 2.0 (подразумевает оперативный съём данных по интерфейсу USB с любого ПК или ноутбука).

Так же возможен вариант использования канала RS-485 для передачи сигнала, а подключение к ПК осуществлять по интерфейсу USB. Для этого в комплект поставки входит преобразователь интерфейса RS-485 – USB.

Для подключения камер на сервере используется разъём BNC.

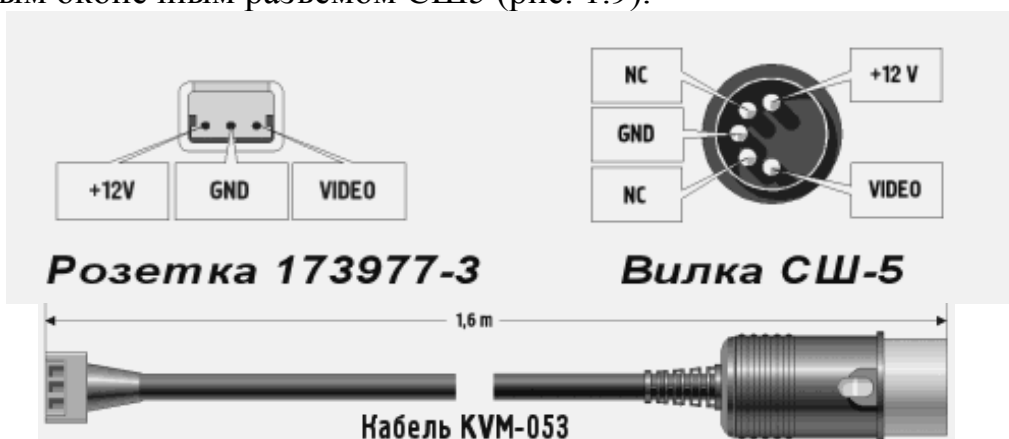
Для видеозахвата в лабораторном комплексе используется малогабаритная камера наружного наблюдения VZP-734 Super HAD CCD.

Наружная камера выполнена с использованием бескорпусной камеры серии V\*A, установленной в малогабаритный корпус из морозостойкого полимера со стеклянным иллюминатором. На иллюминатор установлен специальный радиатор, на который закреплена бескорпусная камера. Высокая экономичность камеры VZP-734 достигается тем, что стеклянный иллюминатор обогревается за счет естественных тепловых ресурсов схемы телевизионной камеры. Сзади на камеру установлена герметичная муфта, через которую проходит кабель камеры, заканчивающийся технологическим разъемом СШ-5. Для надежной герметизации корпуса в камере установлены 2 резиновые прокладки (под стеклянный иллюминатор, а также между передней и задней половинками корпуса). Вид камеры представлен на рис. 1.8.



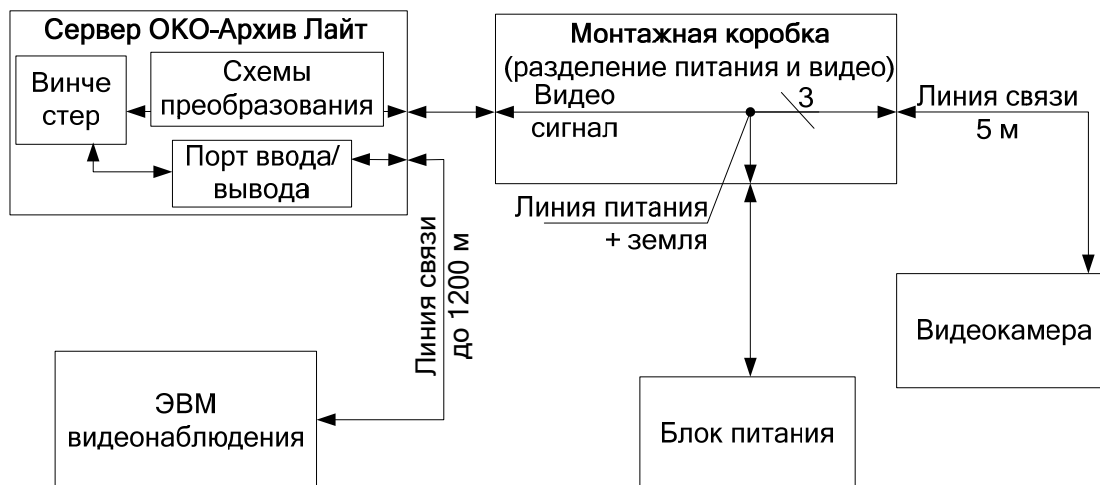
**Рис. 1.8.** Вид малогабаритной камеры наружного наблюдения VZP-734 Super HAD CCD

Для подключения камеры в комплект поставки входит кабель с выходным оконечным разъёмом СШ5 (рис. 1.9).



**Рис. 1.9.** Кабель соединительный для камеры VZP-734 Super HAD CCD

Однако напрямую подключение вилки СШ-5 кабеля (рис. 1.9) в BNC гнездо ОКО-Архива не возможно, поэтому используем следующую схему распайки отдельного кабеля соединительного RVM-053 (рис. 1.10).



**Рис. 1.10.** Структура видеосервера «ОКО-Архив-Лайт» и схема распайки канала передачи видеоданных

Следуя схеме, представленной на рис. 1.10, камера подключается к розетке 173977-3 (рис. 1.9), а в «ОКО-Архив-Лайт» включается вилка Ср-50 (в BNC разъём).

## 1.5. Интерфейс DVI

### 1.5.1. Что такое DVI

Этот интерфейс, полное имя которого – Digital Video Interface, можно назвать «цифровым RGB-интерфейсом». Здесь присутствует 4-е канала передачи данных: 3 из них соответствуют информации об основных цветах В (канал 0), G (канал 1) и R (канал 2), 4-ый же несет в себе сигнал тактовой частоты «Clock» (канал 3). Физически кабель DVI состоит из соответствующего количества витых пар. При этом достигается максимальная скорость потока данных, равная 1.65 Гбит/с, или 165 Мпикс. в секунду при 10-битном кодировании, что соответствует разрешению 1600×1200 пикс. (UXGA) при частоте обновления полей 60 Гц. При этом обеспечивается определенная гибкость, возможна передача сигналов меньшего разрешения при больших значениях кадровой развертки (например, 80 Гц).

Дальнейшее повышение пропускной способности так же возможно – это Dual Link DVI. Здесь все то же самое, но в двойном размере (кроме, естественно, канала 3, передающего тактовую частоту, который в дубликатах не нуждается). Dual Link DVI способен передавать сигналы QXGA (2048×1536 пикселей) при частоте смены полей 60 Гц, как и любые другие, рангом ниже.

Несмотря на явную избыточность Dual Link DVI в отношении современных дисплеев, содержащие его устройства уже производятся. Столь невероятные скоростные возможности DVI достигнуты за счет алгоритма кодирования сигналов, специально разработанного для этой цели. Называется данный алгоритм T.M.D.S – Transition Minimized Differential Signaling, или дифференциальная передача сигналов с минимизацией перепадов уровней, что можно было бы еще назвать «сверх-плотным архивированием данных без потерь». При этом дифференциальный, или балансный, способ передачи, когда по каждому проводнику витой пары проходит один и тот же прямой и инвертированный сигнал, обеспечивает эффективную защиту данных от синфазных помех.

На передающей стороне интерфейса DVI находится T.M.D.S-трансмисмиттер, в котором производится соответствующее преобразование оцифрованного RGB-сигнала и формирование последовательного потока данных в каждом из каналов. На приемной стороне, наоборот, происходит полное восстановление цифровых потоков по каналам R, G, B, а также сигнала Clock.

### **1.5.2. Разновидности DVI**

То, что говорилось выше, на самом деле относится к разновидности DVI-D, т.е. Digital. Однако, наряду с DVI-D существует еще и DVI-A, т.е. «Цифровой видео интерфейс аналоговый». Все дело в том, что для обеспечения более широкой совместимости в разъеме DVI, помимо трех рядов «цифровых» контактов, предусмотрены еще и аналоговые, на которые подается обычный аналоговый RGB-сигнал. Находится эта группа контактов противоположно основной группе контактов. В тех редких случаях, когда цифровые контакты не использованы, мы имеем дело с DVI-A. Когда же все подключено, то это – DVI-I (Integrated). Т.е. совмещенный. По этой же логике получается, что в разновидности DVI-D «вакантными» остаются аналоговые контакты. Современная аппаратура оборудована дисплеями и экранными меню, на которых отображается какой тип интерфейса используется в данный момент для передачи сигналов. Интерфейсы DVI-D и DVI-I, помимо описанных выше цифровых каналов, содержат еще два, предназначенных для обмена информацией между оснащенным видеопроцессором источником (например, PC с видеокартой) и дисплеем. Канал DDC (Display Data Channel) предназначен для передачи подробного «досье» дисплея процессору, который, ознакомившись с ним, выдает оптимальный для данного дисплея сигнал с нужным разрешением и экранными пропор-

циями. Такое досье, называемое EDID (Extended Display Identification Data, или подробные идентификационные данные дисплея), представляет собой блок данных со следующими разделами: название производителя, идентификационный номер модели, серийный номер, дата выпуска, размер экрана, поддерживаемые разрешения и собственное разрешение экрана. В случае, если монитор отказывается выдать информацию о себе (отсутствие DVI-совместимости), канал T.M.D.S блокируется.

При запуске DVI-совместимого источника активизируется процесс HPD (Hot Plug Detect, или опознание активного соединения).

### **1.5.3. Ограничения при передаче DVI сигнала**

У интерфейса DVI имеется лишь одно серьезное ограничение: длина кабеля не должна превышать пяти метров. При более длинных дистанциях не гарантируется стопроцентная достоверность передачи данных (лимит HDMI больше – 15 м). Происходит это из-за искажения фазы сигнала, увеличивающейся по мере потери крутизны фронтов импульсов. Для избежания этого эффекта необходимо выполнять восстановление сигнала – реклокинг (использование регенерированной несущей частоты сигнала с высокой стабильностью).

## **2. Технологии проектирования прикладного программного обеспечения пультов управления с использованием SCADA-систем в качестве инструментального обеспечения**

### **2.1. SCADA-системы, назначение, структура, основные функции**

В настоящее время SCADA-системы широко используются для проектирования программного обеспечения АСУ ТП. Под этим общим названием, как правило, объединяются две группы программных продуктов: компоненты, относящиеся к системе проектирования и программное обеспечение SCADA, функционирующее в составе АСУТП конкретного объекта.

Для начала остановимся на основных функциях, которые возлагаются на любую SCADA-систему. Об основном назначении SCADA-систем можно судить по русскому переводу этого понятия. Supervisory Control And Data Acquisition – система сбора данных и оперативного диспетчерского управления. Таким образом, в названии выделяются две основные функции, возлагаемые на SCADA-систему:

- сбор данных о контролируемом технологическом процессе;
- управление технологическим процессом, реализуемое ответственными лицами на основе собранных данных и правил (критериев), выполнение которых обеспечивает наибольшую эффективность и безопасность технологического процесса.

В более широкой интерпретации на SCADA-систему возлагается выполнение следующих основных функций:

- Прием измерительной информации о контролируемых технологических параметрах от контроллеров нижних уровней и датчиков.
- Сохранение принятой информации в архивах.
- Математическая обработка принятой информации.
- Графическое представление хода технологического процесса, а также измерительной и архивной информации в удобной форме.
- Прием команд оператора и передача их в адрес контроллеров нижних уровней и исполнительных механизмов.
- Регистрация событий, связанных с контролируемым технологическим процессом и действиями персонала,



ответственного за эксплуатацию и обслуживание системы.

- Оповещение эксплуатационного и обслуживающего персонала об обнаруженных аварийных событиях, связанных с контролируемым технологическим процессом и действиями персонала в аварийных ситуациях.
- Формирование сводок и других отчетных документов на основе архивной информации.
- Обмен информацией с автоматизированной системой управления предприятием в целом.
- Непосредственное автоматическое управление технологическим процессом в соответствии с заданными алгоритмами.

Если попытаться коротко охарактеризовать основные функции, то можно сказать, что SCADA-система собирает информацию о технологическом процессе, обеспечивает интерфейс с оператором, сохраняет историю процесса и осуществляет автоматическое управление процессом в том объеме, в котором это необходимо.

## **2.2. Общие сведения о системе TRACE MODE, структура проекта**

TRACE MODE 6 – это программный комплекс, предназначенный для разработки и запуска в реальном времени ПО распределенных автоматизированных систем управления технологическими процессами (АСУТП) и решения ряда задач управления предприятием (АСУП). Для решения задач АСУП в TRACE MODE 6 интегрирован пакет T-FACTORY.

Комплекс программ TRACE MODE 6 можно разделить на 3 части.

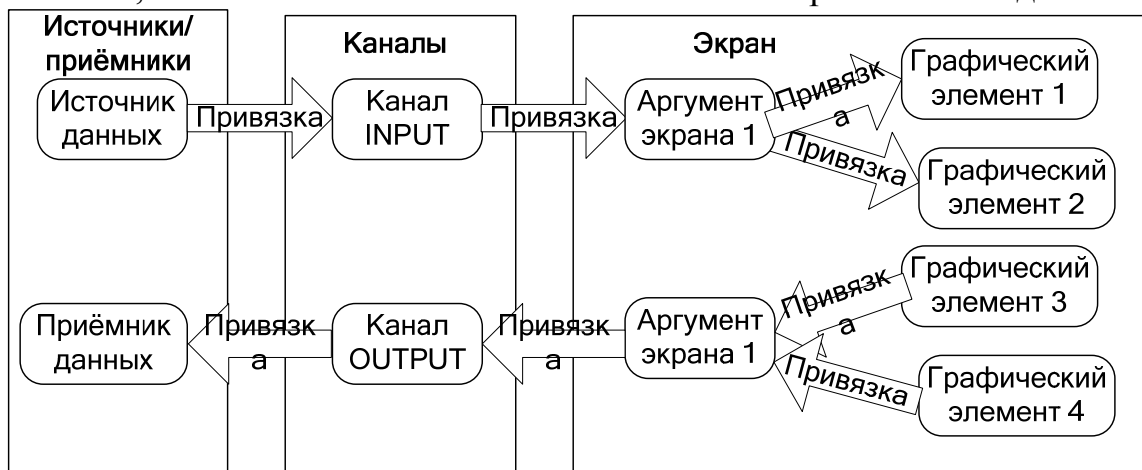
- **Интегрированная среда разработки проекта (ИС)** – единая программная оболочка, содержащая все необходимые средства для разработки проекта. Итогом разработки проекта в ИС является создание файлов, содержащих необходимую информацию об алгоритмах работы АСУ. Эти файлы затем размещаются на аппаратных средствах (компьютерах и контроллерах) и выполняются под управлением исполнительных модулей TRACE MODE.
- **Исполнительные модули (мониторы реального времени, МРВ)** – программные модули различного назначения, под управлением которых в реальном времени выполняются составные части проекта, размещаемые на отдельных компьютерах или в

контроллерах.

- **Драйверы обмена** – драйверы, используемые мониторами TRACE MODE для взаимодействия с устройствами, протоколы обмена с которыми не встроены в мониторы.

Прежде всего, следует обратить внимание на интегрированную среду разработки проекта TRACE MODE 6. Она объединяет функции всех редакторов: базы каналов (РБК), представления данных (РПД), шаблонов (РШ) и т.д. в одной программной оболочке. Работа в интегрированной среде разработки TRACE MODE 6 начинается с выбора стиля разработки проекта АСУТП. Разработка проекта TRACE MODE 6 может начинаться в одном из трех стилей: Простом, Стандартном, Комплексном. В процессе разработки всегда можно перейти к другому стилю, как только возникнет такая необходимость.

При выборе простого стиля (рис. 2.1) в дереве проекта автоматически создается один узел рабочей станции (RTM 1) и главный экран (Экран#1), после чего можно сразу приступить к разработке мнемосхемы технологического процесса. Новые экраны, каналы, источники данных, программы и прочие компоненты добавляются в том же дереве проекта с помощью несложных манипуляций мышью. Размещая на экране графические элементы (ГЭ), пользователь сразу получает возможность осуществить их динамизацию и привязку к аргументам экрана. Аргументы же, в свою очередь, привязываются к каналам, а каналы связываются с источниками и приемниками данных.



*Рис. 2.1. Схема информационных связей, простой стиль разработки*

Каналы это основные структуры данных TRACE MODE 6, в других SCADA подобные информационные единицы могут называться тэгами или точками. Канал может хранить информацию об одном технологическом параметре (сигнале ввода/вывода), либо результат

вычислений. Канал TRACE MODE 6, в зависимости от его типа, может содержать значение с плавающей точкой или целое значение. Источниками данных могут быть, например, PLC контроллеры, платы УСО или внутренние генераторы сигналов. Источник данных TRACE MODE 6 не имеет собственной информационной емкости, он всего лишь описывает, откуда должны поступать данные в канал. То есть источник данных не имеет смысла без привязки к каналу, а один канал в данный момент времени может быть связан только с одним источником данных. Зато один канал может быть связан с произвольным количеством других компонентов экранов, программ или шаблонов документов через их аргументы.

Простой стиль разработки проекта не сильно отличается от идеологии TRACE MODE 5, за исключением выделения источников/приемников данных в самостоятельную структуру. В полной мере новые возможности для построения человеко-машинного интерфейса открываются в более сложных, но в месте с тем и более гибких стилях Стандартный и Комплексный.

В стандартном стиле разработки (рис. 2.2) вводится еще два ключевых понятия, шаблон экрана и вызов экрана, которые в совокупности заменяют само понятие экрана в простом стиле разработки. Ядро TRACE MODE 6 рассматривает все компоненты, будь то экраны, программы на языке FBD или SQL-запросы, как «черные ящики», в которые данные передаются через входные аргументы, затем производятся некие действия внутри компонента, после чего из компонента через аргументы (на этот раз выходного типа) получают обработанные данные результат работы компонента. С точки зрения построения операторского интерфейса такая архитектура позволяет очень легко тиражировать однотипные фрагменты интерфейса пользователя, функции обработки измерительной информации и другие объекты.

В новой версии TRACE MODE 6, достаточно создать один шаблон экрана для некоторого агрегата и необходимое число его вызовов. После доработки шаблона, вызовы не нужно создавать заново, все индивидуальные привязки сохраняются. Если при доработке шаблона будет добавлен новый аргумент, то он автоматически появится во всех других вызовах экрана, и его нужно будет только привязать к каналам. Если доработка экрана не повлияет на аргументы экрана (то есть в нем будет дорисовано нечто статическое, либо связанное с параметрами, для которых динамические элементы, а значит и аргументы уже были созданы ранее), то никаких действий с вызовами экрана производить не потребуется, все 10 (или 100, 1000...)

экранов на рабочем месте оператора автоматически изменятся вместе с шаблоном.

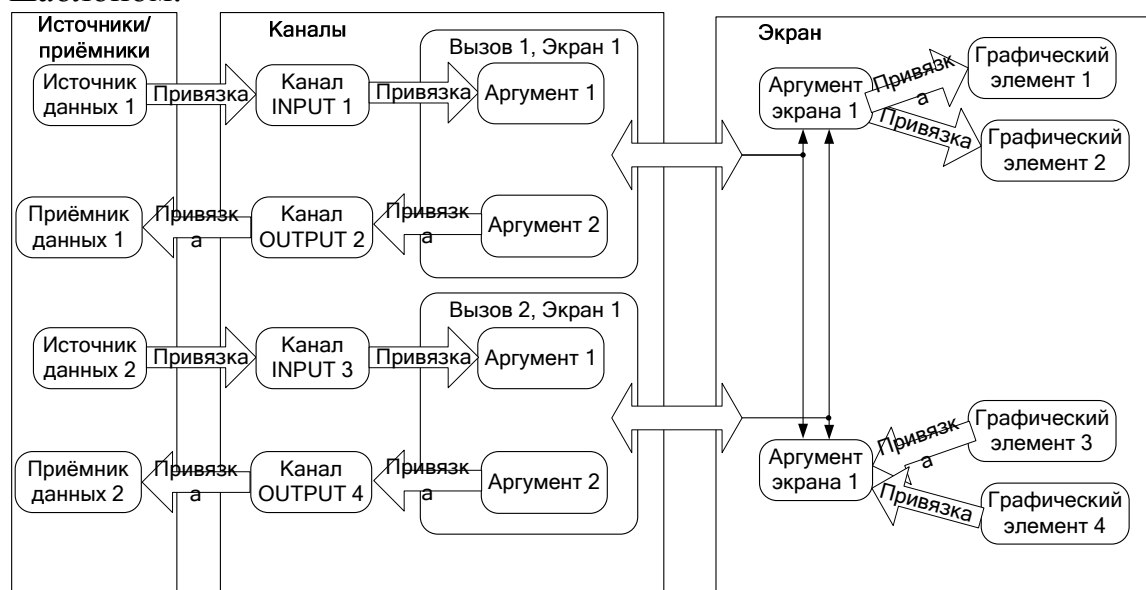


Рис. 2.2. Схема информационных связей, стандартный стиль разработки

### 2.3. Каналы прохождения информации в системе TRACE MODE

**Канал** – это базовое понятие системы. Данные с внешних устройств записываются в каналы. Данные из каналов посылаются на внешние устройства и выводятся в различных формах на экран монитора. В каналы оператор заносит управляющие данные. Значения из каналов записываются в архивы, оперативные отчеты и во все генерируемые документы. В каналах осуществляется преобразование данных. Меняя значения на системных каналах, можно управлять выводимой на экран информацией, звуковыми эффектами, архивами и т.д., то есть всей системой.

Совокупность всех каналов называется **база каналов**, она составляет математическую основу программного обеспечения каждого узла системы управления.

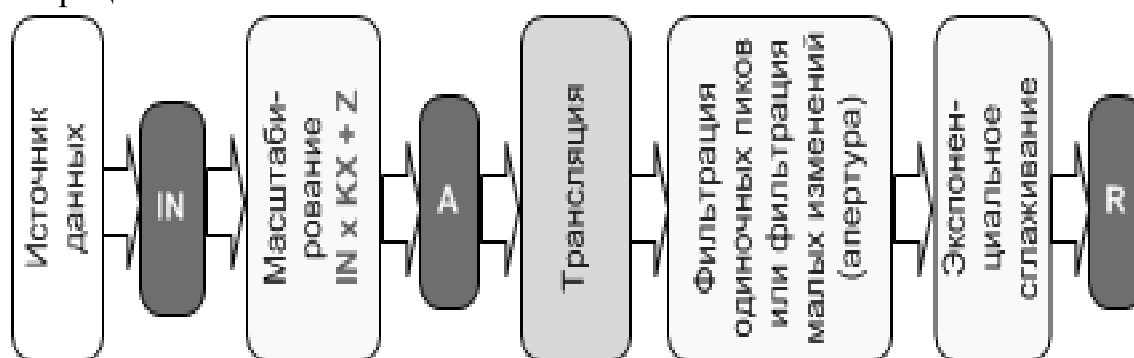
С точки зрения программиста канал представляет собой информационную структуру, включающую в свой состав совокупность переменных и констант, а также методов формирования и преобразования значений этих переменных. Данные поступают во входное значение канала и, пройдя заданную для канала обработку, формируют остальные значения.

В зависимости от направления движения информации, т.е. от внешних источников (данные с контроллеров, УСО или системные пе-

ременные) в канал или наоборот, каналы подразделяются на *входные (тип INPUT)* и *выходные (тип OUTPUT)*.

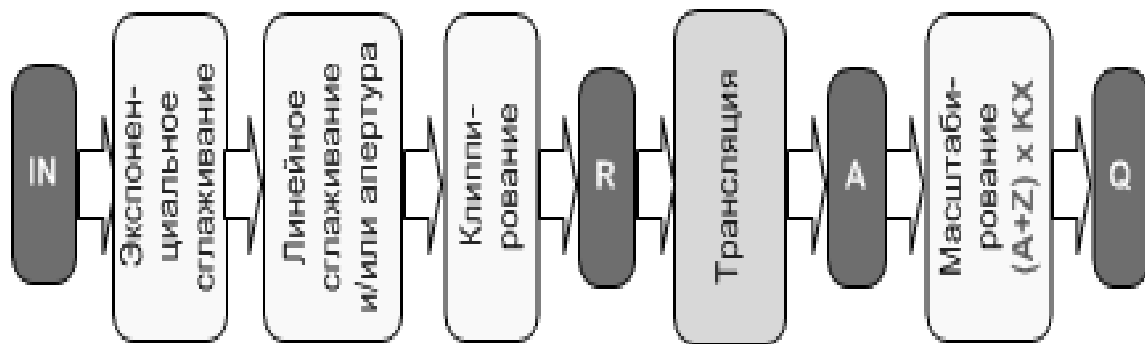
Переменные, константы и идентификаторы канала образуют список его атрибутов. Часть атрибутов задается в редакторах интегрированной среды разработки. Атрибуты могут быть изменены в реальном масштабе времени при работе монитора реального времени.

Каждый канал имеет четыре основных значения: In – входное, A – аппаратное, R – реальное и Q – выходное (рис. 2.3, 2.4). Значения канала также являются его атрибутами. Величина, поданная на вход канала, преобразуется с помощью его процедур. Результаты выполнения этих процедур формируют остальные значения канала. Значения канала могут быть представлены в одном из следующих форматов: *число с плавающей точкой*; *16-битовое целое число*. Первый формат предназначен для хранения и обработки аналоговых переменных, второй - для работы с дискретными переменными и для осуществления логических операций.



*Рис. 2.3. Схема обработки информации в канале типа INPUT*

**Входные каналы** (рис. 2.3). Входной канал запрашивает данные у внешнего источника (контроллер, другой МРВ и пр.) или получает значение какой-нибудь системной переменной (например, счетчик ошибок обмена, длина архива и пр.). Запрашиваемое каналом значение поступает на его вход. Для входных каналов не определено выходное значение. *Преобразование значений в них заканчивается формированием реального значения.*



*Рис. 2.4. Схема обработки информации в канале типа OUTPUT*

**Выходные каналы** (рис. 2.4). Выходной канал передает данные приемнику. Приемник может быть внешним (значение переменной в контроллере, в другом МРВ и пр.) или внутренним – одна из системных переменных, управляющая работой данного МРВ (номер проигрываемого звукового файла, номер экрана, выводимого на монитор, и пр.). И внешние и внутренние приемники данных всегда связываются с выходными значениями каналов.

**Аппаратные значения и входы/выходы.** Значения входных каналов, которые связываются с источниками, называются входными значениями. Путем масштабирования и смещения (предустановки для дискретных каналов) они преобразуются в аппаратные значения. В выходных каналах движение информации осуществляется в обратном направлении. Значения, которые связаны с приемниками данных, называются выходными. Они формируются из аппаратных значений. Аппаратные значения каналов имеют такое название, поскольку в них удобно получать величины унифицированных сигналов, с которыми работает аппаратура ввода/вывода (4-20 мА, 0-10 В и пр.).

**Реальные значения.** Эти значения предназначены для хранения значений контролируемых параметров или сигналов управления в реальных единицах (например, кг/час, °С, % и пр.). Для входных каналов реальные значения формируются из аппаратных. Если канал является выходным, то его реальное значение получается из входного после сглаживания и ограничения по шкале.

### 2.3.1. Обработка данных в канале

Для каждого канала определены две процедуры, реализующие обработку и формирование его значений. Эти процедуры называются трансляция и управление. Для входных аналоговых каналов их входное значение после масштабирования и сдвига формирует величину аппа-

ратного значения. Если канал предназначен для контроля дискретных сигналов, то аппаратное значение формируется логическим сложением входного значения и маски предустановки. Для выходных каналов аналогично осуществляется связь аппаратного и выходного значений. В этом случае меняется только направление движения информации.

Аппаратное и реальное значения канала связаны между собой процедурой трансляции. Кроме трансляции, для каждого канала можно включить еще одну процедуру. Эта процедура называется управление и позволяет осуществить еще один вызов программы пользователя, написанной на одном из встроенных языков. Для выходных каналов эта процедура формирует входное значение канала, а для входных каналов запуск данной процедуры позволяет осуществить дополнительные вычисления и сформировать значения различных атрибутов любых каналов в текущей базе. Таким образом, исходя из сказанного выше, можно выделить шесть основных составляющих канала. Это четыре значения: входное, выходное, аппаратное и реальное, а также две процедуры: трансляция и управление. Процедура трансляции связывает между собой аппаратное и реальное значения одного канала. Процедура управления позволяет вызывать произвольные программы для дополнительных расчетов или формирования сигналов управления.

### 2.3.2. Задание конфигурации канала

Кроме четырех основных переменных канала существует ряд атрибутов, полный перечень которых приведен на рис. 2.5–2.8.

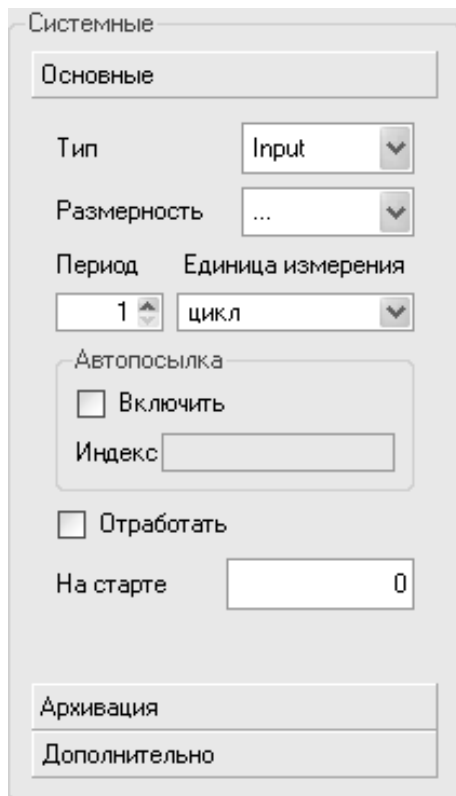


|             |  |           |                                  |  |
|-------------|--|-----------|----------------------------------|--|
| Имя         | <input type="text" value="channel1"/>                | Кодировка | <input type="text" value="TC1"/> | <input type="button" value="Справка"/> |
| Комментарий | <input type="text" value="channel to receive data"/> |           |                                  |  |

Рис. 2.5. Атрибуты каналов в TRACE MODE

В верхней части редактора канала любого класса содержатся поля для редактирования следующих атрибутов:

- имя канала (при создании канала задается по умолчанию и может быть изменено пользователем);
- кодировка класса канала, задаваемая по умолчанию в TRACE MODE. Кодировка может изменяться пользователем;
- комментарий, представляющий собой текстовую строку длиной до 40 символов (редактируется пользователем).



На вкладке «Основные» редактора канала могут быть заданы следующие атрибуты канала:

- **тип канала:** INPUT или OUTPUT. Числовые каналы типа INPUT предназначены для приема данных от источников, типа OUTPUT – для передачи данных приемникам. Монитор может автоматически устанавливать для канала тип, соответствующий привязанному источнику/приемнику.
- **размерность** реального значения канала (в физических величинах).
- **период пересчета** канала по времени.

*Рис. 2.6. Видеограма вкладки «Системные атрибуты / Основные»*

- **единица измерения** периода пересчета канала;
- **включить** автопосылку при установке этого флага монитор будет передавать в сеть реальное значение канала при каждом его изменении в виде широковещательного сообщения. На других узлах такое сообщение принимается каналами, которые связаны с данным;
- **индекс** автопосылки – индекс, по которому другие узлы идентифицируют широковещательное сообщение;
- **отработать** – установка этого флага является признаком необходимости пересчета и отработки канала при старте монитора реального времени;
- **на старте** – значение, указанное в этом поле, присваивается входному значению канала при старте монитора.

На вкладке «Архивация» редактора канала задаются следующие атрибуты канала:

- **СПАД** – признак архивирования атрибутов канала в базе данных реального времени SIAD.
- **Регистратор** – признак архивирования атрибутов канала в глобальный архив (регистратор).



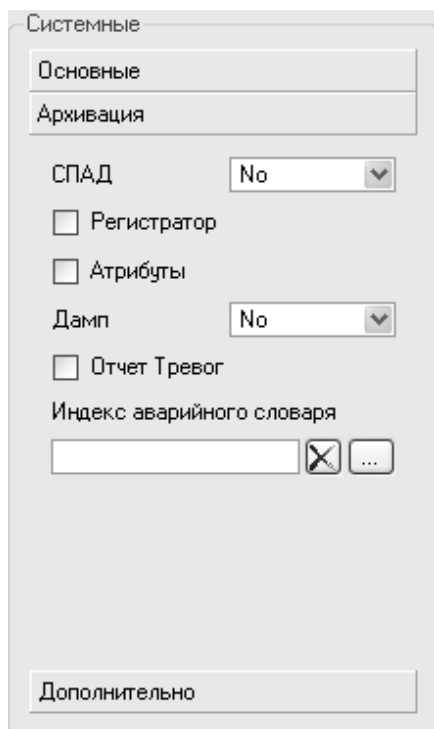


Рис. 2.7. Видеограма вкладки «Системные атрибуты / Архивация»

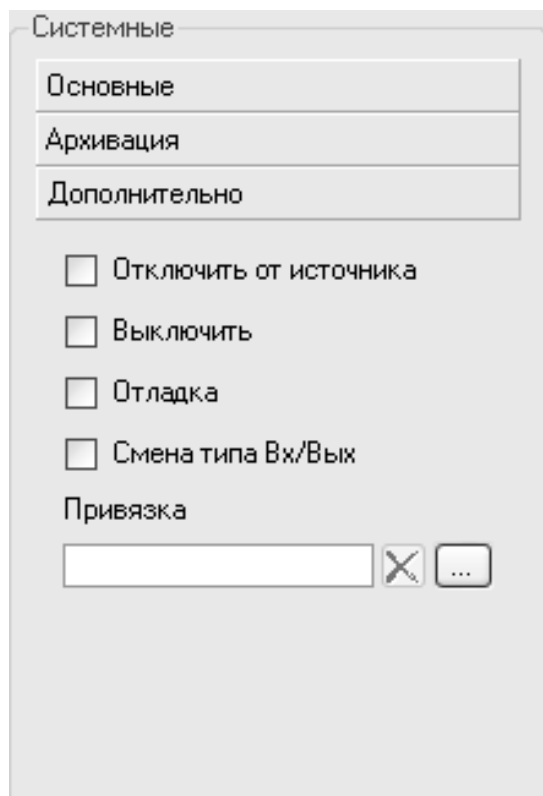


Рис. 2.8. Видеограма вкладки «Системные атрибуты/Дополнительно»


- **Атрибуты** – от этого флага сит набор атрибутов канала, архивируемых в указанный архив SIAD и/или регистратор.
- **Дамп** – признак использования дампа, выбирается из следующего списка: **No** – не использовать, **READ** – считывать значение канала из дампа при старте, **WRITE** – записывать информацию по данному каналу в дамп, **READ/WRITE** – выполнять обе операции.
- **Отчет тревог** – признак генерации сообщений для текстового архива (отчета тревог).
- **Индекс аварийного словаря** – словарь, сообщения из которого

будут использованы для событий данного канала (если словарь не задан, генерируются сообщения, заданные по умолчанию).

На вкладке «Дополнительно» редактора канала задаются следующие атрибуты:

- **Отключить от источника** – при установке этого флага канал отключается от источника/приемника;
- **Выключить** – установка этого флага означает остановку пересчета канала;

**Отладка** – если этот флаг установлен, в отладочный файл будет выводиться информация, определяемая каналом, связанным с системной переменной **@Debug** (группа СИСТЕМНЫЕ);

- **Смена типа Вх/Вых** – смена типа канала на одну обработку. Этот флаг используется для канала типа OUTPUT, связанного с приемником, – например, для канала, задающего уставку в контроллере. Если узел, на котором размещен такой канал, по каким-либо причинам остановлен, то при его дальнейшем запуске в канал вначале необходимо считать текущее значение уставки из контроллера (для этого тип канала должен быть INPUT), после чего переключить канал в рабочий режим;
- **Привязка** – задание источника данных для канала (рис. 2.9). Компонент для привязки выбирается в следующем диалоге (для вызова диалога нужно нажать кнопку ).

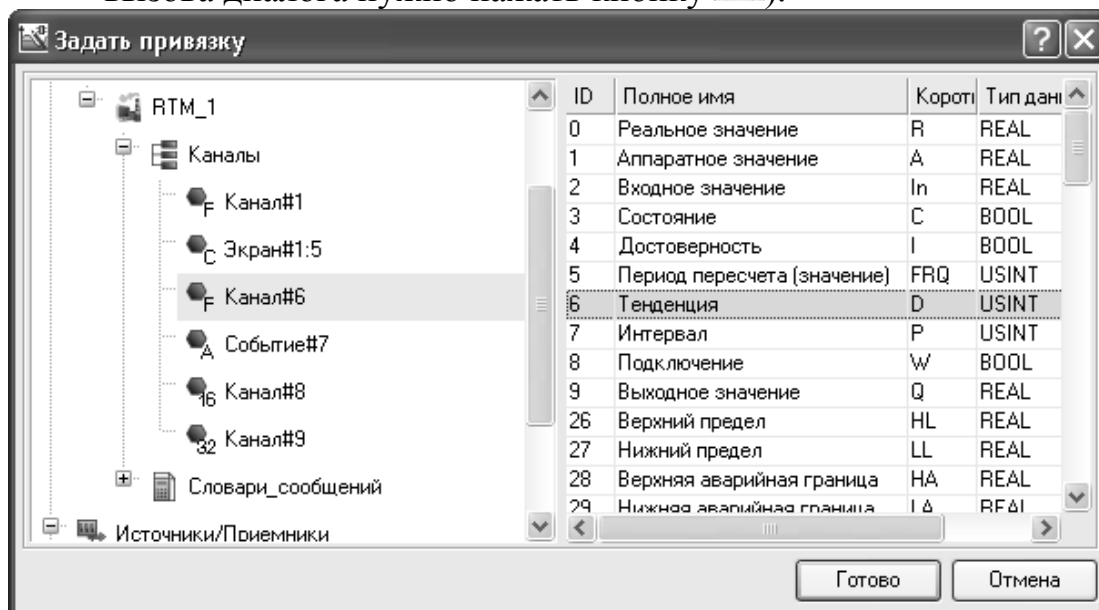


Рис. 2.9. Видеодиаграмма вкладки «Задать привязку»

## 2.4. Разработка графического интерфейса оператора

Одним из важнейших элементов, обеспечивающих взаимодействие человека с технической системой, является пульт оператора. Высокая эффективность системы «человек – техника» обеспечивается за счет следующих принципов:

1. минимизация времени выполнения отдельных действий и операций в трудовом процессе;
2. исключение грубых ошибок типа промахов;
3. минимизация вероятности ошибок, отрицательно сказывающиеся на ходе технологического процесса, качестве продукта или отрицательно влияющих на состояние техники или человека;

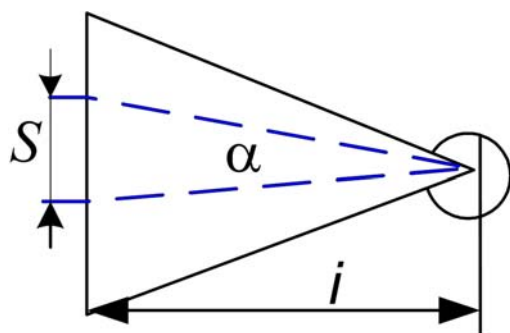
4. сохранение высокой работоспособности человека в течение длительного времени.

Эти задачи должны решаться, в том числе, и средствами отображения информации об управляемом объекте предоставляемые SCADA системами. Дадим общие рекомендации для построения информационной модели объекта или процесса управления.

Отбор и выдача информации о состоянии управляемого объекта должен осуществляться в такой форме, которая наиболее соответствует закономерностям восприятия и переработки ее человеком. Исходя из этого, применительно к задаче обнаружения сигналов наличие в поле зрения оператора множества информационных элементов существенно влияет на характер его деятельности. Время поиска кроме этого зависит от условий наблюдения: углового размера знаков расстояния считывания, яркости изображения, контраста и т.д.

При отображении объектов время зрительного поиска существенно зависит от объема значимого параметра или количества знаковых мест, которые он содержит. Время поиска возрастает с увеличением объема значимого параметра в связи с тем, что он превосходит размеры оперативного поля зрения, в результате чего увеличивается число зрительных фиксаций. Рекомендуемые ГОСТ линейные размеры мнемознаков (рис. 2.10) можно определить по следующей формуле:

$\operatorname{tg} \frac{\alpha}{2} = \frac{S}{2l}$ , где  $l$  – расстояние от глаз оператора до плоскости экрана,  $S$  –



линейный размер мнемознака,  $\alpha$  – угловой размер мнемознака. Рекомендуемые угловые размеры:

$\alpha \geq 20'$  - для простого мнемознака

$\alpha \geq 30'$  - для сложного мнемознака, состоящего из нескольких деталей

$\alpha \geq 6'$  - размер наименьшей детали сложного мнемознака.

*Рис. 2.10. Схема расчёта линейных размеров мнемознака*

Предельно допустимый угловой размер пульта оператора не более 90 градусов по вертикали и горизонтали.

Эффективность выполнения поисковых задач зависит также от структуры информационного поля. При хаотическом расположении большого числа элементов в информационном поле эффективность поиска мала и повышается за счет специальной организации информаци-

онного поля. Раньше и с большей точностью обнаруживаются знаки, находящиеся в левом верхнем квадранте, откуда обычно начинается маршрут движения глаз при визуальном сканировании.

Разнообразие элементов информационного поля увеличивает время поиска и частоту ошибок при операциях выделения полезной информации (состояний). Количество эталонов, с которыми сличают сканируемые элементы информационного поля, влияет на сложность задачи. Обычно поиск по нескольким эталонам сложная задача, время выполнения которой увеличивается с ростом числа эталонов.

Особое место в инженерной психологии занимает кодирование информации. Выбор оптимального кодирования определяются задачами, которые решает оператор. Для этого нужен тщательный анализ функционально значимых признаков кодируемого сообщения. Код должен быть максимально осмысленным и читаться подобно тексту. Информационные сигналы могут быть представлены: цифрами, буквами, условными знаками, геометрическими фигурами и их размерами, линиями, цветом, яркостью и другими методами оповещения оператора, включая звуковые.

**Кодирование формой** – универсальное средство кодирования информации, так как обеспечивает большой алфавит символов и их интерпретацию с объектами, хорошо известными человеку из практической деятельности. Простые фигуры – прямоугольники, треугольники, фигуры из прямых линий различаются лучше, чем многоугольники, круги или криволинейные фигуры. При кодировании формой особое значение приобретает вопрос о мере абстрактности значков. Обычно частичное воспроизведение признаков сигнала в его отображении дает высокую точность решения задачи декодирования. Вместе с тем следует соблюдать определенную меру «картинности». Мера наглядности определяется требованием хорошей различимости знака. Система кодирования должна быть совместимой с жизненным и процессуальным опытом оператора. Все болты на корпусе аппарата рисовать не стоит.

**Кодирование размером** – соотнесение длины, площади с количественной характеристикой реального объекта (объемом, весом, скоростью и т.д.)

**Кодирование пространственной ориентацией.** Осуществляется поворотом фигуры в поле зрения и ее перемещением (стрелки на шкалах).

**Буквенно-цифровое кодирование.** Основное внимание при буквенно-цифровом кодировании нужно уделять читаемости букв Q и O; C и G; 5 и 6; 3 и 5 и т.п., которые часто воспринимаются ошибочно.

**Цветовое кодирование.** Человек может точно идентифицировать не более 10-12 цветовых тонов. Более точно опознаются фиолетовый, голубой, зеленый и красные цвета. При совмещении цветовой статической и динамической информации рекомендуется использовать не только цвет, но и насыщенность. Статические элементы следует изображать в светлых тонах, динамические в насыщенных, сочных тонах. Необходимо учитывать, что при увеличении расстояния от экрана, цвет воспринимается измененным (желтый и голубовато-желтый кажутся белыми, синий и красный - черными).

**Кодирование яркостью.** Применяют, чтобы привлечь внимание оператора к определенному сигналу. Хорошее восприятие возможно не более чем четырех градаций яркости одного цвета, причем отношение яркости соседних уровней 1:5.

**Кодирование частотой мигания** используют редко, так как оно утомляет оператора. Красный мигающий цвет с частотой в 3-5 Гц используется для аварийных сообщений.

#### **Требования к информационной модели объекта.**

- **Тщательный выбор доминирующего критерия.** Представление информации оператору в сконцентрированном виде, в обобщенной форме, чтобы исключить время на выделение, поиск и суммирование информации.
- **Интегрирование сигналов в укрупненные единицы.** Рекомендуется в тех случаях, когда от оператора требуется
  - Одновременная оценка нескольких взаимосвязанных параметров различного качества.
  - Большой объем однородной информации
  - Сопоставление противоречивых или взаимосвязанных данных
  - Качественная оценка сложившейся ситуации.

В связи с тем, что в процессе работы у оператора может появиться необходимость в дополнительной детализирующей информации необходимо ее представление по вызову.

- **Учет последовательной организации внимания оператора.** Расположение элементов информационной модели должно соответствовать наиболее вероятной последовательности изменений сигнала или маршруту обзора оператором. Для привлечения внимания к элементам расположенным вне зоны доминирующего маршрута следует использовать усиление сигналов повышенной яркостью.
- **Максимальная разгрузка оперативной памяти.** Следует располагать всю информацию, необходимую для принятия решений на каждом шаге в границах оптимального поля зрения.

- **Обеспечение предвидения.** Своевременная подготовка оператора к выполнению нужной операции. Предыстория процесса предупреждающие сообщения о возможном ходе процесса и т.д. (предаварийные сигналы предупреждения).
- **Максимальное высвобождение внимания.** За счет автоматизации выполнения стандартных составляющих деятельности оператора. Например, стандартный код для идентичных сигналов (состояние “выключено” соответствует нижнее положение тумблера).
- **Минимизация нагрузки на долговременную память.** Использование кодов, максимально ассоциируемых с жизненным опытом (опасность/ запрет – красный цвет; температура - вертикальная шкала, верх – большее значение параметра; горячее - красный, желто-белый цвета; холодный - синий цвет)
- **Оптимизация дифференцированного восприятия сигналов.** При кодировании сигналов учитывать оперативные пороги восприятия.
- **Темп предъявления информации.** Учитывать то, что недопустима, как перегрузка, так и недогрузка оператора. Наиболее часто приходится бороться с перегрузкой оператора информацией. Это достигается:
  - фильтрацией информации по критерию;
  - сохранением текущей информации для последующего анализа;
  - упреждением вывода информации перед началом ее исполнения и т.д.

При проектировании пультов отображения информации необходимо учитывать быстродействие оператора как время решения задачи оператором от момента появления сигнала до выдачи управляющего воздействия в простых случаях. Это время  $T = a + b \cdot I$  зависит от времени на обработку единицы информации  $b$  и объема информации  $I$ , а также от времени выделения существенной информации из потока  $a$ .

Еще один из важных параметров пропускная способность оператора. Зависит от задач, которые он решает. Допустим, если в функцию оператора входит чтение с экрана символьной информации с равнороятным появлением символов, то пропускная способность  $C = n(\log_2 R)/T$ , где  $n$  - число правильно считанных символов,  $R$ -длина алфавита,  $T$ - время, затраченное на чтение. При чтении “про себя” пропускная способность равна 45; вслух - 30, корректорская работа – 18; сложение или умножение двух цифр – 12; счет предметов 3 бит/с.

Время появления цифр и букв на экране, при котором они все опознаются, составляют 0.06 и 0.09с., т.е. мгновенная пропускная способность при опознании таких символов равна 55бит/с.

На результаты деятельности оператора сильно влияет характер поступающего к нему информационного потока. Для определения напряженности работы оператора используют следующие коэффициенты.

**Коэффициент загруженности**  $\eta = \frac{\tau_0}{T_0}$ ,  $\tau_0$  - время в течение которого

оператор занят обработкой информации,  $T_0$  - общее время дежурства.

Коэффициент  $\eta$  не должен превышать значения 0,75. Второй коэффициент **Период занятости** – время непрерывной без пауз работы.  $t_n$  не должно превышать 15 минут.

## 2.5. Редактор представления данных TRACE MODE 6.

Графическое представление хода выполнения техпроцесса, а также управление техпроцессом с помощью графических средств являются одними из главных задач, решаемых TRACE MODE 6. Для разработки графического интерфейса оператора в интегрированную среду встроены **редактор представления данных (РПД)** (рис. 2.11.). Графический интерфейс оператора разрабатывается в РПД в виде набора **графических экранов**. Для создания шаблона экрана нужно выполнить команду **Экран**. Совокупность графических экранов узла образует его **графическую базу**. Совокупность графических баз всех узлов разрабатываемого проекта АСУТП образует **графическую часть** проекта. Графический экран/панель может содержать один или несколько графических **слоев**, каждый из которых, в свою очередь, может содержать один или несколько **подслоев**.

В слоях графического экрана/панели размещаются **графические элементы** (ниже соответственно ГЭ). Графические элементы имеют наборы настраиваемых **атрибутов**, **динамических свойств** и **функций управления**.

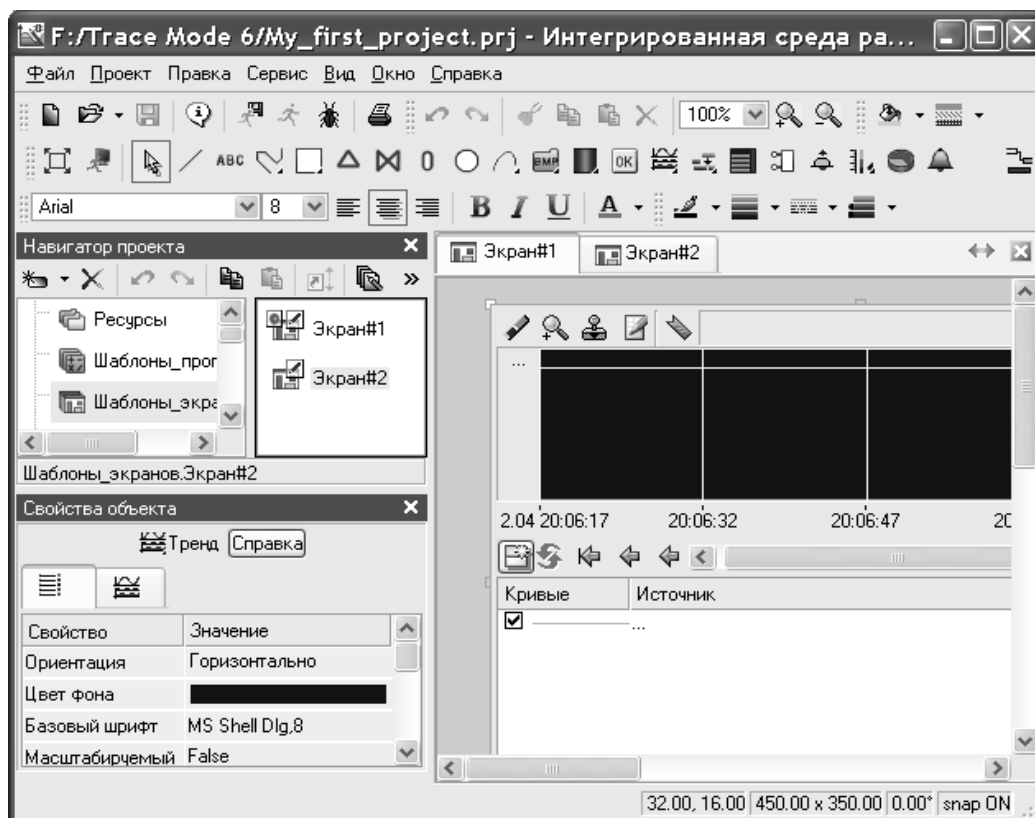






Рис. 2.11. Видеограмма рабочего окна редактора представления данных

Эти параметры определяют вид графических элементов и выполняемые ими функции отображения и управления при работе в реальном времени. РПД содержит большое количество встроенных графических элементов, позволяющих изобразить практически любой техпроцесс, вывести на дисплей всю необходимую информацию о ходе его выполнения, а также управлять техпроцессом. Состав главного меню и панелей инструментов РПД приведен ниже.




### ***Панель инструментов «Графические элементы»***

 С помощью инструментов этой панели выбираются графические элементы для размещения их в графических слоях экранов. С помощью кнопки  данной панели можно перейти в режим редактирования, с помощью кнопки  – в режим эмуляции. Кнопка  предназначена для переключения режима отображения графических экранов (обычный/полноэкранный).

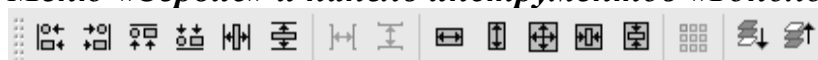
### ***Меню и панель инструментов «Правка»***

 Меню и панель инструментов



**Правка** содержат ряд типовых инструментов для редактирования графических экранов. Данные инструменты доступны также из контекстного меню ГЭ. В списке  (Масштаб) можно выбрать предустановленный масштаб или вручную задать произвольный. Для выбора предустановленного масштаба можно также использовать кнопки  /  или сочетания клавиш **CTRL+ПЛЮС/МИНУС** на цифровой клавиатуре. При выделении некоторой области **AxB** экрана с помощью мыши с удержанием клавиши **Z** экран масштабируется в  $\text{MIN}\{C/A, D/B\}$  раз, где **CxD** – размеры видимой области. Во всех случаях масштабирование производится относительно центра видимой области.

### ***Меню «Сервис» и панель инструментов «Топология экрана»***



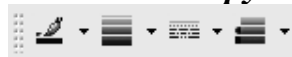
Данные панель инструментов и меню содержат команды для позиционирования и тиражирования ГЭ.

### ***Панель инструментов «Параметры текста»***







В режиме редактирования с помощью типовых инструментов данной панели задаются параметры текста в выделенном графическом элементе (выделенной группе ГЭ). Данные команды применимы только к такому тексту, который может быть введен /отредактирован с помощью клавиатуры.

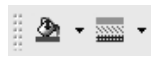
### ***Панель инструментов «Параметры линии»***





В режиме редактирования с помощью инструментов этой панели задаются параметры линии (линии контура) выделенного графического элемента (выделенной группы ГЭ):

-  – выбор **цвета линии**. По этой команде на экран выводится стандартный диалог выбора цвета;
-  – выбор **толщины линии**;
-  – выбор **стиля линии**;
-  – выбор **края линии** (плоский, квадратный, круглый).

### ***Панель инструментов «Параметры заливки»***



В режиме редактирования с помощью инструментов этой панели задаются параметры заливки выделенного графического элемента (выделенной группы ГЭ):  – выбор **цвета заливки**.  – выбор **стиля заливки**.

### ***Панель инструментов «Ресурсные библиотеки»***



Инструменты данной панели предназначены для операций с библиотеками строк, рисунков и других ресурсов, которые могут быть использованы при разработке графических экранов.

### ***Меню «Вид»***

Команды этого меню управляют видимостью редактора аргументов экрана, окна **Слои**, таблицы графических элементов и окна **Избранное**, а также панелей инструментов **Топология экрана** и **Параметры текста**. Меню **Вид** содержит также команду **Параметры экрана**.

В новой версии TRACE MODE уже не существует деления на статические и динамические элементы, принятого в пятой версии TRACE MODE. Любой текст может стать и динамическим текстом, и текстовым индикатором, и цветовым индикатором, и даже гистограммой. Необходимость в индикаторах, как отдельных элементах, полностью отпала. Расширены возможности динамизации для всех графических элементов:

- Любой элемент может быть кнопкой;
- Можно задавать независимые действия по нажатию и отжатию кнопки мыши;
- Любые графические элементы (за исключением таких, как тренд или стрелочный прибор) могут:
  - двигаться по произвольной траектории в реальном времени;
  - изменять свои геометрические размеры в реальном времени;
  - поворачиваться на произвольный угол в реальном времени.
- Любой 2D- графический элемент, обладающий площадью, может стать гистограммой (в том числе замкнутая кривая и динамический текст);
- Контур любого 2D- графического элемента может стать бегущей дорожкой.

Полностью переработаны тренды: Тренд сам определяет, откуда брать данные – из архива SIAD/SQL6 или временного буфера, то есть деления на архивный тренд и тренд реального времени больше нет, как нет и деления на дискретный и аналоговый тренды, все они успешно реализуются в рамках одного универсального тренда.

Доработаны кнопки:

- Текст кнопки и его цвет, а также цвет фона теперь может

меняться в зависимости от аргумента;

- Есть только один тип кнопки, для которой опционально можно задавать текст или картинку, «залипание» «твердой» кнопки - тоже опция;
- Кнопка элемент операционной системы, поэтому под Windows XP поддерживается подсветка при наведении курсора мыши.

## **2.5. Особенности запуска монитора реального времени TRACE MODE**

Система работает в реальном времени, если ее быстродействие адекватно скорости протекания физических процессов на объектах контроля или управления. То есть система управления должна собрать данные, произвести их обработку в соответствии с заданными алгоритмами и выдать управляющие воздействия за такой промежуток времени, который обеспечивает успешное решение поставленных перед системой задач. Для этой цели в составе TRACE MODE имеются специальные модули – Мониторы реального времени (MPB).

Все мониторы TRACE MODE циклически выполняют определенный набор операций. Структура задач, исполняемых MPB, определяется в базе каналов. MPB работают как интерпретаторы базы каналов. Каждый цикл системы включает несколько основных операций. Для решения задач, заданных в узле, монитор может создавать до 24 потоков. Приоритеты потоков заданы по умолчанию, однако их можно изменить на вкладке **Дополнительно** редактора узла. При редактировании приоритетов они выбираются из следующего списка.

-1 – Default;    4 – BELOW\_NORMAL;    12 – HIGHEST;  
0 – IDLE;        6 – NORMAL;                    14 – TIME\_CRITICAL;  
2 – LOWEST;    10 – ABOVE\_NORMAL;    77 – STOP

Ниже указаны задачи, выполняемые в потоках:

1 – основной поток, выполняемый монитором циклически. Один цикл включает следующие последовательно выполняемые этапы:

- последовательный анализ всех включенных каналов узла (по возрастанию порядкового номера), пересчет всех каналов (кроме каналов CALL) типа INPUT, которые должны пересчитываться в основном потоке;
- пересчет и обработка каналов класса CALL основного потока;
- пересчет каналов типа OUTPUT, которые должны пересчитываться в основном потоке, и анализ их выходного значения.

Если используются приоритеты потоков по умолчанию, и для цикла МРВ установлено время, недостаточное для выполнения всех его задач, система будет работоспособной (будет идти обмен по последовательному интерфейсу и сети, выполняться программы и т.п.), однако заданные временные характеристики пересчета/отработки каналов будут нарушаться (т.е. быстродействие системы снизится).

- 2 – прием данных по сети IP;
- 3 – передача данных по сети IP;
- 4 – действие – печать, вывод в файл и т.п.;
- 5 – выполнение задач в качестве сервера DDE, NetDDE;
- 6 – T-FACTORY (решение задач управления предприятием);
- 7 – обмен по протоколу MODBUS TCP/IP;
- 8 – ведение отчета тревог;
- 9 – копирование отчета тревог;
- 10 – дамп атрибутов канала для последующего безударного рестарта МРВ;
- 11 – обработка быстрых каналы;
- 12 – TCP/IP (TCP для удаленной отладки);
- 13 – вызов драйвера t12;
- 14 – выполнение функций MASTER при обмене по последовательному интерфейсу;
- 15 – выполнение функций SLAVE при обмене по последовательному интерфейсу;
- 16 – обмен с использованием модема;
- 17 – отображение графических экранов;
- 18 – ведение базы данных реального времени SIAD;
- 19 – генерация документов; выборка их архивов с помощью каналов класса CALL; выполнение программ, вызываемых каналами класса CALL;
- 20 – выполнение задач в качестве клиент DDE, NetDDE;
- 21 – обмен данными по сети сотовой связи GSM.

**Время цикла монитора.** При конфигурировании узла необходимо задать время, отводимое монитору на цикл (однократное выполнение задач основного потока). Управляя временем цикла и приоритетами потоков монитора, а также временными характеристиками пересчета отдельных каналов, можно оптимизировать быстродействие системы. Время цикла настраивается с помощью двух параметров, которые задаются в разделе **Пересчет** вкладки **Основные** редактора параметров узла. Параметр **Разрешение** задает разрешение таймера tick (в секун-

дах), параметр **Период** – период пересчета в единицах tick: Произведение этих параметров определяет время цикла монитора в секундах. Разрешение таймера (tick) может варьироваться в пределах, указанных в списке.

в MS Windows – не менее 0.01с;    в MS DOS – в диапазоне  
0.001 с – 0.055 с;

в MS WinCE – не менее 0.001с;    в MinOS7 и ROM-DOS –  
не менее 0.055с

По умолчанию разрешение таймера равно 0.055 с, период – 10. Реальное время цикла может быть немного больше установленного значения.

При конфигурировании МРВ может быть определен файл восстановления (дамп). В этот файл монитор записывает последние значения атрибутов каналов узла. Данные заносятся в дамп по каналам, для которых установлен соответствующий флаг, а также по каналам, созданным при работе в реальном времени. Дамп используется для восстановления значений каналов узла после рестарта монитора.

## **2.6.    Создание распределенных систем управления**

Распределенные системы в интегрированной среде TRACE MODE 6 разрабатываются методом конфигурирования информационных потоков. Основной принцип конфигурирования информационного потока в ИС заключается в независимом описании звеньев потока с дальнейшим заданием связей между звеньями.

До тех пор, пока речь идет о связи между компонентами одного узла, не возникает вопрос об аппаратно/программном интерфейсе, который должен быть задействован для обеспечения связи, – в этом случае достаточно выполнить конфигурирование свойств **связь/вызов** компонентов. Если взаимодействующие компоненты относятся к разным узлам, интерфейс связи, как правило, должен быть указан и сконфигурирован. Например, при задании связи двух каналов разных узлов по последовательному интерфейсу необходимо создать в узлах компоненты **СОМ-порт**, задать для них необходимые параметры и указать для канала-приемника используемый интерфейс связи.

Свойство **связь** может быть задано для компонента при копировании и вставке, в окне свойств, а в случае канала – и в его редакторе. Свойство **вызов** может быть задано для компонента при копировании и вставке и в окне свойств.

### 2.6.1. Организация сетевого взаимодействия между МРВ

Существуют три основные архитектуры сетевого обмена, которые используются в АСУ ТП: «Клиент/Сервер», «Master/Slave» (ведущий / ведомый) и обмен с помощью широковещательных пакетов.

Алгоритм Клиент/Сервер в основном используется в офисных приложениях, инициатором обмена является клиент. В АСУТП этот алгоритм может использоваться только на верхнем уровне, где не решаются задачи непосредственного управления объектом или процессом. Так как поток запросов клиентов не детерминирован, в сервере создается очередь запросов. Поэтому время получения ответа недетерминировано. В случае большого количества запросов сервер может не успевать обрабатывать запросы, а значительное время заниматься обслуживанием очереди.

В алгоритме «Master/Slave» одна ведущая станция в распределенной АСУ ТП управляет обменом. Процесс обмена описывается парами сетевых сообщений «Запрос данных → Ответ»; «Команда → Подтверждение». Запросы выдаются последовательно во времени, т.е. ведущий узел выдает запрос и ждет ответ, а затем выдает запрос другому подчиненному узлу. Алгоритм позволяет оценить точное время, затрачиваемое на обмен. Это время растет с ростом числа подчиненных узлов участвующих в обмене.

Широковещательные рассылки сообщений. Используются при необходимости одновременной рассылка информации многим потребителям. При таком способе обмена информацией возникает необходимость создания очередей со стороны клиента.

В TRACE MODE используются два алгоритма сетевого взаимодействия «Master/Slave» и широковещательный (в терминах TRACE MODE – «Автопосылка в сеть»).

При связи каналов в качестве **интерфейса взаимодействия** могут быть заданы:

- **CHNET** – по сети по протоколу IP;
- **CHMLINK** – по последовательному интерфейсу по протоколу M-LINK.

При взаимодействии с другим узлом во всех случаях (кроме приема сетевых автопосылок) канал с настроенным свойством **связь** посылает запрос (в случае интерфейса CHNET – на IP-адрес удаленного сетевого узла), содержащий указание удаленному каналу передать свое значение или изменить его, и принимает ответ, содержащий ин-

формацию о выполнении запроса. При приеме сетевой автопосылки ответ не генерируется.

Автопосылка – это способность монитора передавать в сеть реальные значения каналов в виде широковещательных сообщений (в сети TCP/IP широковещательные сообщения отправляются по специальному IP-адресу, все биты которого равны единице). Монитор генерирует такие сообщения по каналам любого типа (как INPUT, так и OUTPUT) с установленным флагом Автопосылка при каждом изменении их реального значения.

Вид взаимодействия каналов задает характеристика **вид связи**:

- **CHCOPY** – копирование/установка значения удаленного канала (запрос/ответ);
- **CHASEND** – прием сетевой автопосылки (подтверждение приема не генерируется). Каналы, принимающие автопосылки, должны иметь тип INPUT;
- **CHGROUP** – запрос/ответ по групповому номеру;
- **CHFAULT** – запрос/ответ по резервам;
- **CHCS** – специфический вид взаимодействия.

При конфигурировании связи каналов разных узлов их взаимодействие задается на вкладке **Информация** окна свойств канала (рис. 2.12) с помощью выбора из списка одной из следующих опций:

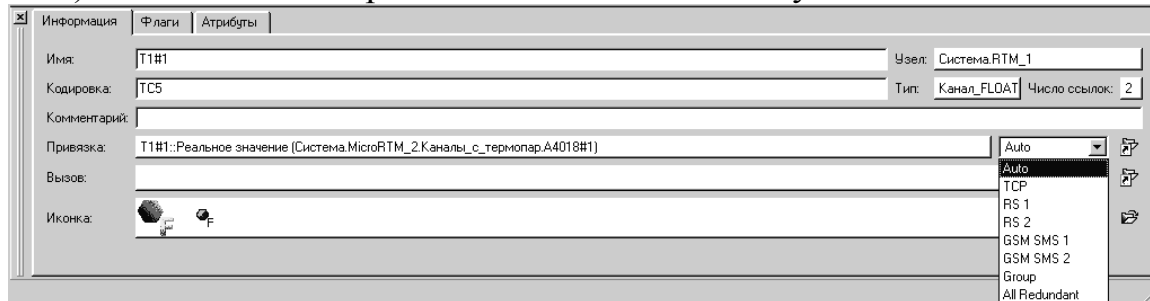


Рис. 2.12. Видеограма информационного табло каналов в TRACE MODE

- **Auto** – CHNET/CHMLINK. Если доступно взаимодействие по сети, будет установлен интерфейс CHNET вне зависимости от возможности взаимодействия по интерфейсу RS. При установке интерфейса CHMLINK используется первый обнаруженный COM-порт, который сконфигурирован корректно. Опция **АВТО** задается по умолчанию;
- **TCP** – по TCP;
- **RS 1** – CHMLINK, используется первый обнаруженный COM-порт, который сконфигурирован корректно;
- **RS 2** – CHMLINK, используется второй обнаруженный COM-порт, который сконфигурирован корректно;

- **GSM SMS 1** – в виде SMS-сообщения, используется первый обнаруженный модем;
- **GSM SMS 2** – в виде SMS-сообщения, используется второй обнаруженный модем;
- **Group** – CHGROUP.

Для управления обменом по сети используются каналы, связанные со следующими системными переменными TRACE MODE (группа СИСТЕМНЫЕ).

- **@Mode\_Control.**
- **@Status**

### 2.6.2. Обмен с базами данных

Для взаимодействия с базами данных (далее – БД) в TRACE MODE 6 встроена поддержка языка **SQL**. Основными функциями, выполняемыми TRACE MODE 6 при работе с БД, являются следующие:

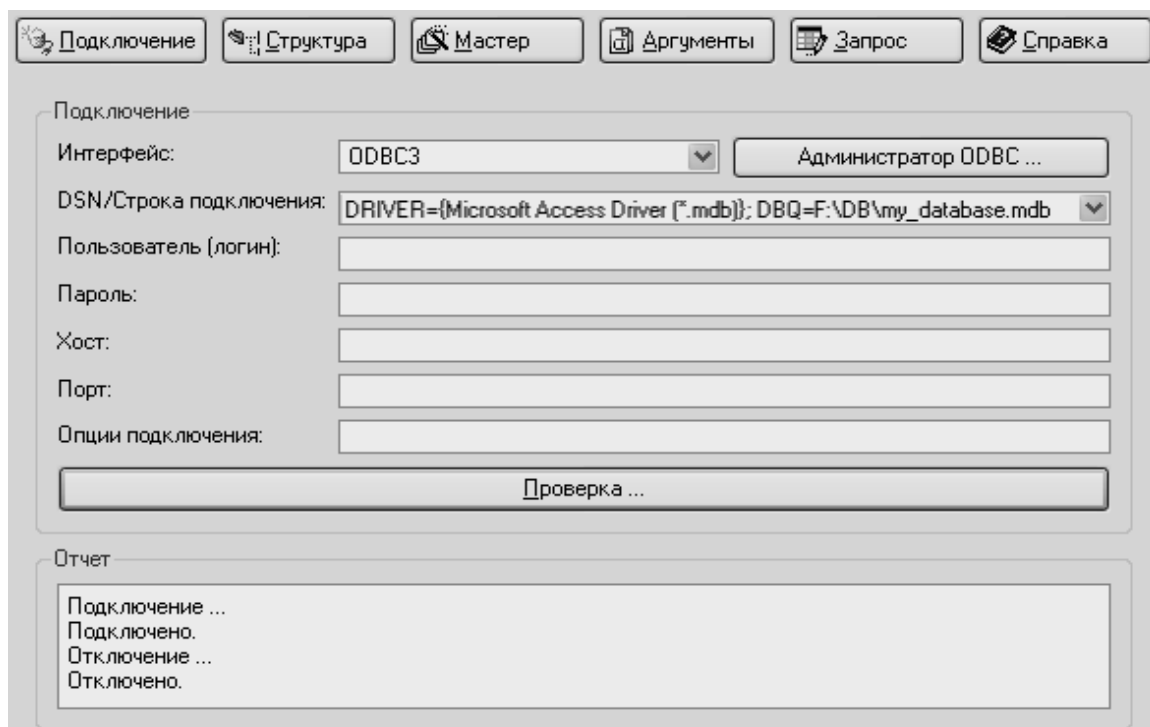
- подключение к БД по протоколу ODBC3;
- создание новых источников данных ODBC (при редактировании проекта);
- отображение информации о структуре БД (при редактировании проекта);
- создание и редактирование запросов к БД, в том числе при помощи мастера (при редактировании проекта);
- выполнение запросов к БД (при редактировании проекта и в реальном времени).

Для программирования связей с БД в интегрированную среду разработки проекта встроено **редактор связей с базами данных**. Этот редактор снабжен **мастером**, позволяющим пользователям конфигурировать основные операции по взаимодействию с базами данных с использованием языка SQL.

**Подключение к базе данных.** При открытии редактора связей с БД по умолчанию открывается окно **Подключение** (рис. 2.13). В этом окне конфигурируется подключение к базе данных. Для подключения к базе данных можно использовать два способа:

- зарегистрировать БД в качестве источника данных ODBC и подключиться к этому источнику;
- подключиться непосредственно к файлу БД.

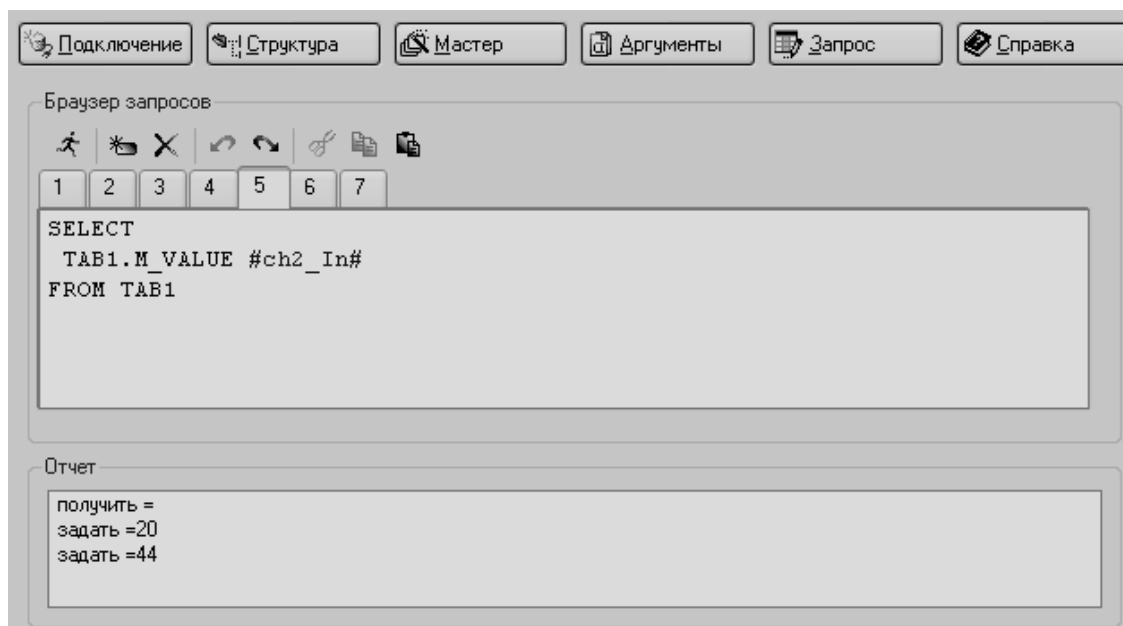




*Рис. 2.13. Видеограмма окна мастера подключения к БД*

Для подключения к источнику данных ODBC нужно указать его параметры в окне **Подключение** редактора связей с БД. В поле **Интерфейс** нужно выбрать протокол (ODBC3), в поле **DSN/Строка подключения** – имя зарегистрированного источника данных. В поле **Пароль** нужно ввести пароль, заданный для БД. Поля **Пользователь**, **Хост** и **Порт** надо оставить пустыми. Если подключение к БД выполнено корректно, в окне **Структура** редактора связей с БД отображается структура базы данных. В столбце **Тип** этого окна содержится информация о типе данных полей таблиц БД. В столбце **Обяз.поле** указывается, являются ли поля столбцов БД обязательными для заполнения (**нет** – могут принимать значение **NULL**, т.е. быть пустыми, **да** – поля обязательны для заполнения). В столбце **Сортировка** указывается порядок сортировки данных (**asc** – по возрастанию, **desc** – по убыванию).

**Создание SQL - запросов.** Запросы SQL могут быть созданы вручную в окне **Запросы** или с помощью мастера (рис. 2.14) (для его запуска нужно нажать кнопку **Мастер**). Запрос, построенный с помощью мастера, может быть в дальнейшем отредактирован в окне **Запросы**:



*Рис. 2.14. Видеозапись окна мастера создания SQL запросов в TRACE MODE*

Данное окно снабжено типовыми инструментами создания/удаления запросов и работы с буфером обмена.

Для каждого запроса – созданного вручную или с помощью мастера – в окне автоматически создается вкладка, на которой запрос может быть отредактирован. Каждому запросу автоматически присваивается **номер** (начиная с 1). Этот номер, отображаемый в заголовке вкладки, является идентификатором запроса и используется впоследствии для выполнения запроса в реальном времени.

**Выполнение SQL-запросов в реальном времени.** Для выполнения SQL-запросов к подключенной БД в реальном времени в узле должен быть создан канал класса CALL с типом вызова **SQL запрос**, настроенный на вызов шаблона связи с БД. При данном типе вызова атрибут **Глубина выборки** канала CALL не используется, тип канала не имеет. С помощью атрибута **Параметр** канала CALL можно задать номер возвращаемой строки из выборки, полученной из таблицы БД в результате запроса. Если **Параметр=0** (значение по умолчанию), возвращается последняя строка. Значение канала CALL задает **номер** выполняемого запроса. После отработки значение канала автоматически сбрасывается в 0.

### 2.6.3. Обмен между приложениями по протоколу DDE.

Суть DDE достаточно проста - определены несколько стандартных типов сообщений, при помощи которых можно передавать деск-

рипторы (описатели) объектов глобальной памяти, содержащих необходимые данные. Для адресации объектов в памяти используются: *Сервис* (Service) обычно имя приложения DDE сервера; наименование типа передаваемых данных называемое *темой* (topic), и группа данных, передаваемых по одному сообщению, называемая *элементом данных* (data item).

Элементами данных, как правило, являются текстовые строки, битовые образы, таблицы данных и др. Передавать в сообщениях такие данные (массивы) неудобно, поэтому в DDE вводится специальный способ кодирования данных. Для этого используется универсальный объект обмена между приложениями – атом (уникальный идентификатор для строки символов или указатель на другие данные). Windows поддерживает в памяти таблицу атомов, которая ставит в соответствие массиву данных целочисленное значение (адрес в таблице). Приложения могут добавлять или удалять строки из этой таблицы, а также извлекать строки по идентифицирующему номеру. Эта таблица является основной DDE механизма, т.е. между приложениями пересылаются адреса данных (атомы), а сами данные находятся в таблице (разделяемой памяти).

Мониторы TRACE MODE поддерживают обмен по DDE/NetDDE между собой и с приложениями WINDOWS, выступая одновременно в качестве сервера и клиента. Клиент инициирует обмен с сервером и задает один из следующих режимов обмена:

- **POKE** – изменение значения указанного параметра на сервере;
- **REQUEST** – запрос значения указанного параметра от сервера (обычный DDE-канал);
- **ADVISE** – режим, при котором сервер посылает клиенту значение указанного параметра при его изменении («горячий» DDE-канал).

### **Обмен «Приложение – MPB как DDE-сервер»**

При обмене с локальным MPB из приложения-DDE-клиента, возможно:

- чтение атрибута (0, R) каналов (в режиме REQUEST или ADVISE);
- чтение (REQUEST) и запись (POKE) любых других атрибутов каналов.

Если клиентом DDE является Excel, то DDE-обмен с MPB может быть сконфигурирован с помощью формул Excel или макросов VBA. Формула Excel может быть использована только для запроса значения

(в режиме ADVISE или REQUEST) – в этом случае она имеет следующий формат: =<server>|<topic>!<item> , где

**server** – имя сервера в формате RTM<k>, где **k** – индивидуальный номер узла;

**topic** – тема запроса (см. примеры ниже);

**item** – имя канала или уточненное имя атрибута.

Уточненное имя атрибута имеет следующий формат: <имя канала>.<номер атрибута>

*Пример: «Запрос значения атрибута R».* Для запроса реального значения канала **pila** в режиме ADVISE в ячейку таблицы Excel нужно записать следующую формулу:

=RTM2|GET!pila

В этом режиме значение в ячейке обновляется автоматически. В режиме ADVISE монитор посылает клиенту значение канала при каждом его пересчете.

Для запроса реального значения канала **pila** в режиме REQUEST в ячейку таблицы Excel нужно записать одну из следующих формул:

=RTM2|PUT!pila

=RTM2|PUT!pila.0

Во второй формуле тема запроса может быть произвольной. В этом режиме значение канала запрашивается и записывается в ячейку однократно при исполнении формулы.

## Обмен «MPB как DDE-клиент – Приложение»

Если монитор выступает в роли DDE-клиента, для конфигурирования обмена используются переменные DDE. Рассмотрим обмен на одном компьютере между монитором и Excel, выступающим в качестве DDE-сервера. Вкладка настройки обмена по DDE представлена на рис. 2.15.

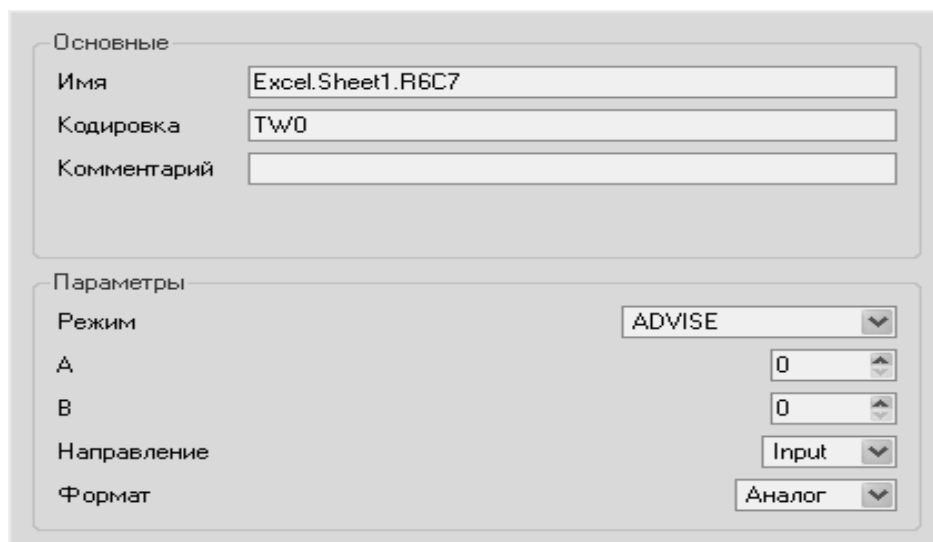
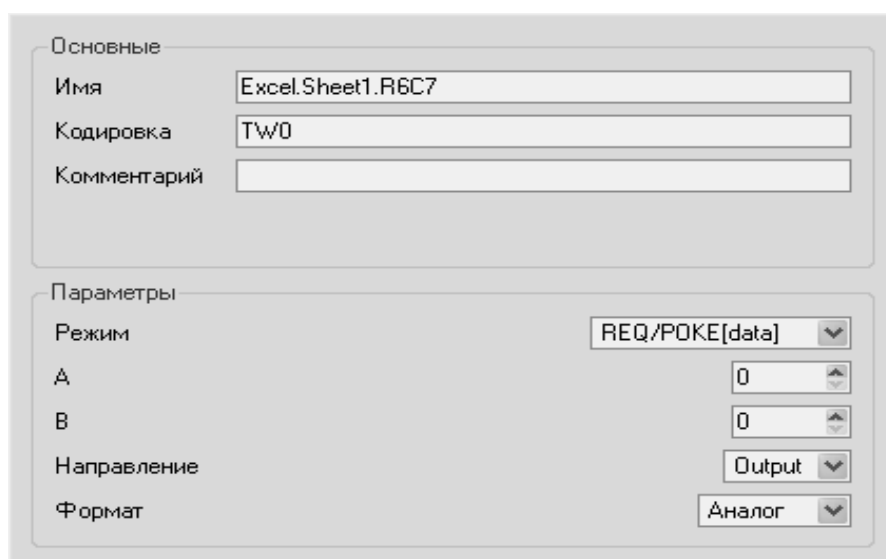


Рис. 2.15. Видеограмма окна настройки DDE обмена между MPB и Excel

**Пример: «Запрос значения в режиме REQUEST».** Для запроса в режиме REQUEST тип переменной DDE должен быть INPUT. Создадим в узле канал **ch2** и свяжем его с переменной DDE, параметры которой зададим как на рис. 2.15. Для запроса в режиме REQUEST параметру **Режим** переменной DDE можно также задать значение **REQ/POKE[data]** (рис. 2.16).

**Пример: «Запрос значения в режиме ADVISE».** Создадим в узле канал **ch1** и свяжем его с переменной DDE, параметры которой зададим как на рис. 2.15. Откроем Excel, создадим новую книгу и запустим узел – установится связь монитора с книгой Excel, активной в данный момент, в режиме ADVISE. В этом режиме DDE-сервер (Excel) будет посылать DDE-клиенту (монитору) значение (при каждом его изменении) ячейки **G6** (Row=6, Column=7) листа **Sheet1**, и это значение будет записано во вход канала **ch1**.



**Рис. 2.16.** Видеозапись окна настройки DDE обмена между MPB и Excel (режим *quest*)

Откроем Excel, создадим новую книгу и запустим узел – установится связь монитора с книгой Excel, активной в данный момент, в режиме REQUEST. В этом режиме монитор генерирует в соответствующем потоке запросы на чтение значения ячейки **G6** листа **Sheet1** и записывает это значение во вход канала **ch2**.

**Пример: «Задание значения в режиме POKE».** Для передачи в указанную ячейку таблицы Excel выходного значения канала (атрибут 9, **Q**), привязанная к этому каналу переменная DDE должна иметь тип OUTPUT. Создадим в узле канал **ch\_p** и свяжем его с переменной DDE, параметры которой зададим как на рис. 2.16. Для записи значения в Excel параметру **Режим** переменной DDE можно также задать значение **REQ/POKE[data]**. Откроем Excel, создадим новую книгу и запустим узел – установится связь монитора с книгой Excel, активной в данный момент, в режиме POKE. В этом режиме монитор генерирует в соответствующем потоке запрос на запись выходного значения (при каждом его изменении) канала **ch\_p** в ячейку **G6** листа **Sheet1**.

#### **2.6.4. Обмен между приложениями по протоколу NetDDE.**

Обмен по NetDDE требует предварительного запуска ряда служб ОС:

- NT DDE-сервер (clipsrv.exe);
- NETWORK DDE – обеспечение обмена по NetDDE (netdde.exe);
- NETWORK DDE DSMD (Network DDE Service Data Manager) – совместный доступ к данным DDE.

Если монитор сконфигурирован как NetDDE-клиент, при старте он запускает необходимые службы, если они не были запущены предварительно. Если необходимые службы запущены, монитор, в случае его запуска пользователем с правами администратора, автоматически создает разделяемый DDE-ресурс **RTM<k>\$** (**k** – индивидуальный номер узла, работающего под управлением MPB). Рассмотрим редактор свойств разделяемого DDE ресурса на рис. 2.17.

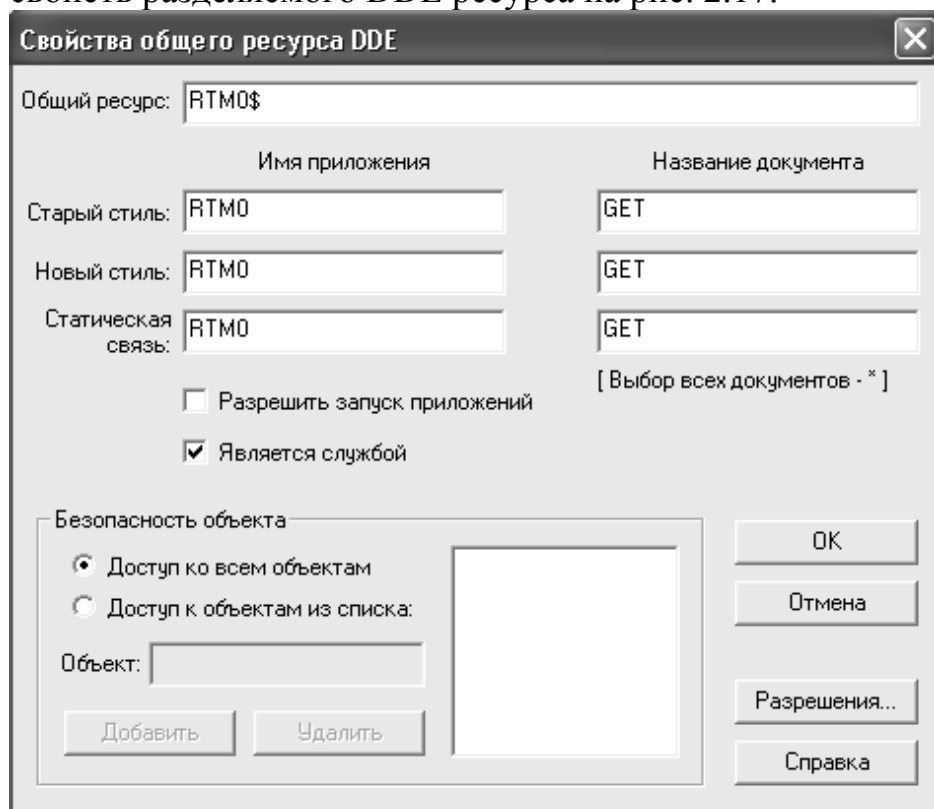


Рис. 2.17. Видеограмма окна настройки NetDDE обмена

При этом разрешения на доступ к ресурсу не конфигурируются и остаются заданными по умолчанию. Это означает, что все пользователи, которые обращаются к ресурсу, должны быть идентичными пользователю сервера (логин и пароль) и иметь права администратора.

Для участия различных пользователей в обмене по NetDDE необходимо до старта сервера и клиента выполнить следующие действия:

- пользователь с правами администратора должен «прописать» пользователей, участвующих в обмене, на сервере и клиенте NetDDE.
- пользователь с правами администратора должен с помощью DDE Share Manager вручную создать на сервере ресурс **RTM<k>\$**; и в диалоге **Разрешения** «прописать» участников

обмена для созданного ресурса, задав для них полные права:

### Обмен «Приложение – MPB как NetDDE-сервер»

При обмене с монитором – сервером NetDDE – из приложения возможен запрос значений атрибута (0, **R**) (в режимах ADVISE или REQUEST), а также запрос (REQUEST) и задание (POKE) значений других атрибутов каналов узла (как удаленного, так и локального).

Имя сервера задается в следующем формате: \\<name>\NDDE\$, где **name** – имя компьютера, на котором запущен MPB. Тема запроса задается как **RTM<k>\$**, где **k** – индивидуальный номер узла.

**Пример: «Запрос атрибута R».** Для запроса реального значения канала **ch1** узла **nodeA** в режиме ADVISE в ячейку таблицы Excel нужно записать следующую формулу (параметр **Update** в диалоге **Links** должен иметь значение **Automatic**):  
=`"\nodeA\NDDE$" | RTM3$.OLE!"ch1'`

### Обмен между MPB по NetDDE

Обмен по NetDDE между мониторами конфигурируется в узле, выступающем в роли клиента, – для этого используются переменные DDE. В приведенных примерах (рис. 2.18, 2.19) предполагается, что узел, выступающий в роли NetDDE-сервера, имеет индивидуальный номер 1 и запущен на компьютере с именем **nodeA**.

**Пример: «Запрос R».** Для запроса в режиме ADVISE значения атрибута **R** создадим в узле – клиенте NetDDE – канал **ch2** и свяжем его с переменной DDE, параметры которой зададим как на рис. 2.18. Запустим мониторы – между ними установится связь в режиме ADVISE. В этом режиме сервер (**nodeA**) будет посылать клиенту значение канала **ch1**, и это значение будет записываться во вход канала **ch2**.



*Рис. 2.18. Видеозапись окна настройки NetDDE обмена (режим ADVISE)*

*Рис. 2.19. Видеозапись окна настройки DDE обмена (режим REQUEST)*

Для запроса в режиме REQUEST значения атрибута **R** создадим в узле – клиенте NetDDE – канал **ch3** и свяжем его с переменной DDE, параметры которой зададим, как указано на рис. 2.19. Для запроса в режиме REQUEST параметру **Режим** переменной DDE можно также задать значение **REQ/POKE[data]**. Запустим мониторы – между ними установится связь в режиме REQUEST. В этом режиме клиент генерирует в соответствующем потоке запросы на чтение значения атрибута (0, **R**) канала **ch1** сервера и записывает это значение во вход канала **ch3**.

## 2.6.5. Обмен между приложениями по протоколу OPC

OPC (OLE for Process Control) – это стандарт взаимодействия между программными компонентами системы сбора данных и управления (SCADA), основанный на объектной модели COM/DCOM фирмы Microsoft. Через интерфейсы OPC одни приложения могут читать или записывать данные в другие приложения, обмениваться событиями, оповещать друг друга о нештатных ситуациях (тревогах), осуществлять доступ к данным, зарегистрированным в архивах (так намываемые «исторические» данные). Эти приложения могут располагаться как на одном компьютере, так и быть распределенными по сети, при этом независимо от фирмы-поставщика стандарт OPC, признанный и поддерживаемый всеми ведущими фирмами-производителями SCADA-систем и оборудования, обеспечит их совместное функционирование.

OPC – взаимодействие основано на клиент-серверной схеме. OPC-клиент (например, SCADA), вызывая определенные функции объекта OPC-сервера, подписывается на получение определенных данных, с определенной частотой. В свою очередь, OPC-сервер, опросив физическое устройство, вызывает известные функции клиента, уведомляя его о получении данных и вручая сами данные. Таким образом, при OPC-взаимодействии используются как прямые COM-вызовы (от клиента к серверу), так и обратные (от сервера к клиенту).

Рассмотрим использование MPB TRACE MODE в качестве OPC –сервера для обмена с другими SCADA – системами. Для доступа OPC-клиентов к каналам узла он должен исполняться под управлением специализированного монитора – OPC-сервера TRACE MODE 6.

Перед началом использования OPC-сервера TRACE MODE 6 его нужно зарегистрировать в операционной системе. Для этого используются командные файлы:

- **register\_opc\_proxy\_stub.cmd** (однократная процедура, после которой сервер можно регистрировать/разрегистравать). Для отмены нужно запустить командный файл **unregister\_opc\_proxy\_stub.cmd**;
- **register\_server.cmd** (для отмены регистрации нужно запустить файл **unregister\_server.cmd**);

Если OPC-сервер зарегистрирован, для его запуска вручную достаточно выполнить следующую команду: %OPC-сервер%/tm6opcDas.exe (рис. 2.20). При запросе клиента OPC-сервер запускается автоматически. Интерфейс оператора OPC-сервера TRACE MODE 6 включает консоль служебных сообщений сервера и графическую оболочку (рис. 2.21).

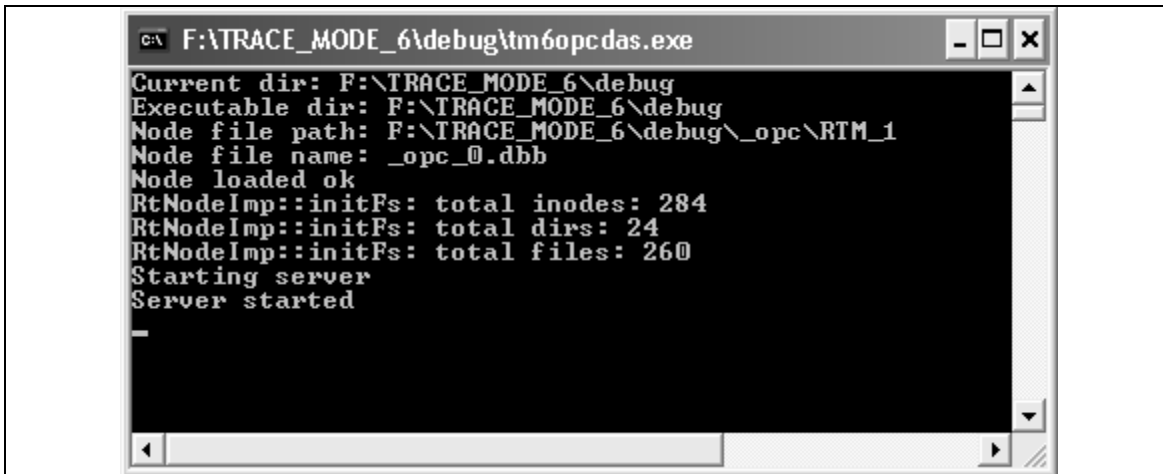


Рис. 2.20. Видеограмма консоли с результатами запуска OPC сервера

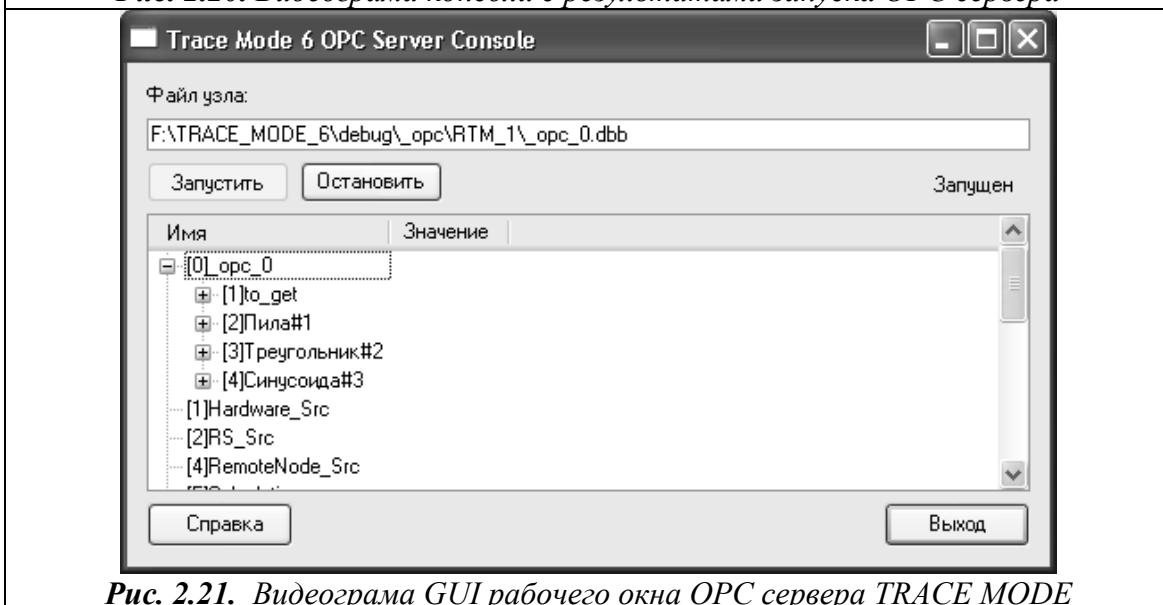


Рис. 2.21. Видеограмма GUI рабочего окна OPC сервера TRACE MODE

Оболочка OPC-сервера отображает каналы узла. В поле **Файл узла** отображается полный путь к файлу базы каналов (\*.dbb) узла. Кнопки **Запустить** и **Остановить** предназначены соответственно для запуска и останова сервера. Информация о состоянии сервера отображается в строке этих кнопок справа. При нажатии кнопки **Выход** сервер останавливается и выгружается.

Пусть узел, указанный при регистрации OPC-сервера, содержит каналы **Пила1**, **Треугольник1** и **Синусоида1**, связанные с соответствующими генераторами TRACE MODE.

Пусть сервер зарегистрирован на компьютере с именем \\Lab129. Запустим на другом компьютере OPC-клиент, выполним в оболочке клиента подключение к новому серверу AdAstra.Tm60pcServe. Создадим в клиенте переменные для чтения значений каналов **Пила1**, **Треугольник1** и **Синусоида1**, привязав их к реальным значениям каналов.

Текущие значения каналов отобразятся в оболочке клиента.

Для окончания сеанса выполним в клиенте команду **Disconnect** (из контекстного меню строки подключения) – OPC-сервер будет остановлен и выгружен.

## **2.7. Архивирование и документирование, система архивов TRACE MODE**

TRACE MODE поддерживает четыре типа архивов:

- Отчет тревог. Отчет тревог ведется в ASCII-формате. В этом архиве осуществляется фиксация событий.
- SIAD (база данных реального времени). В этот архив значения каналов записываются в бинарном формате. Условием новой записи является изменение значения канала.
- Глобальный регистратор – это архив, который ведет специализированный монитор. Значения архивируемых в регистраторе каналов посылаются ему по сети при их изменении.
- Индивидуальные (поканальные) архивы в оперативной памяти. Могут использоваться в бездисковых конфигурациях узлов, например, микропроцессорных контроллеров, работающих под управлением МикроМРВ.

Настройка параметров архивирования осуществляется в окне редактора проекта (рис. 2.22).

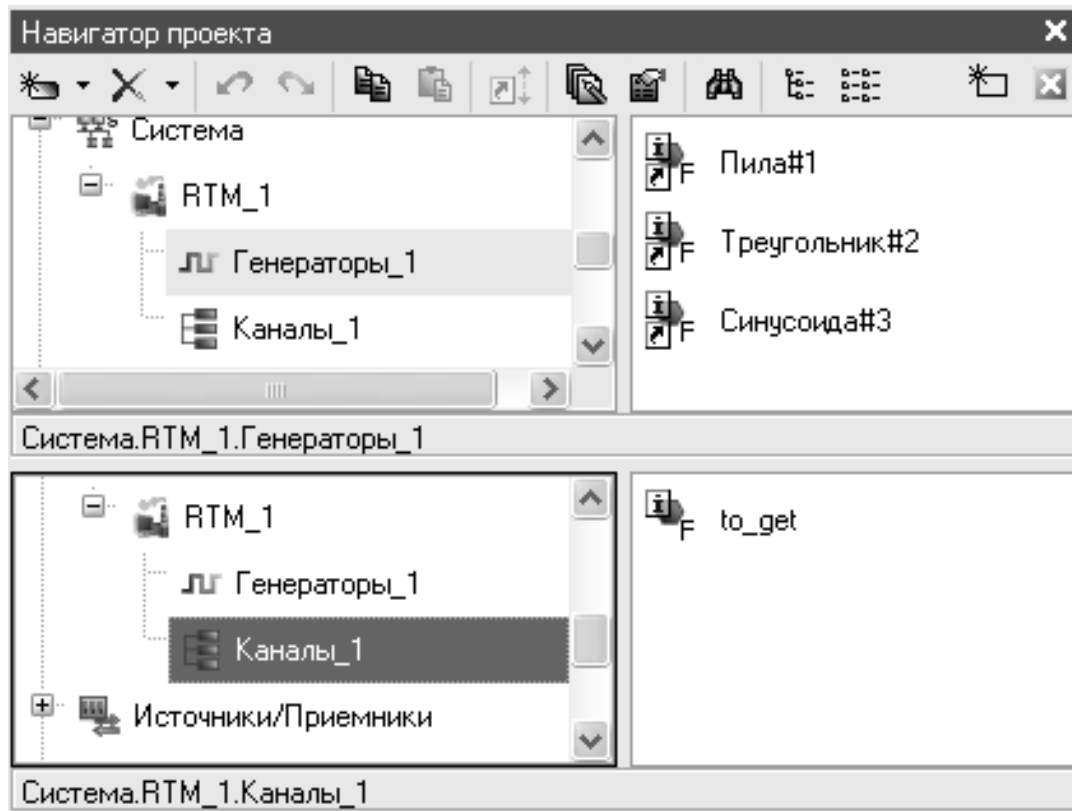


Рис. 2.22. Видеограмма окна навигатора проекта TRACE MODE

### 2.7.1. Отчет тревог узла

Отчет тревог (ОТ) – это текстовый файл (ASCII), в который заносятся сообщения, генерируемые в различных ситуациях при работе АСУ. Отчет тревог конфигурируется для узла (на вкладке **Отчет тревог/Дамп/Параметры** редактора узла). Если ОТ для узла не задан, монитор не генерирует сообщений. В отчет тревог могут быть записаны сообщения следующих видов:

- системные сообщения;
- сообщения по каналам;
- сообщения, генерируемые с помощью системной переменной **@Message** (группа СИСТЕМНЫЕ);
- интерактивные сообщения оператора.

Каждое сообщение о событии заносится в отчет тревог в виде отдельной строки. Сохранение сообщений в ОТ реализовано в виде отдельного потока с более низким приоритетом по сравнению с основным потоком МРВ. Для управления генерацией сообщений в реальном

времени используется переменная **@Logging** (группа СИСТЕМНЫЕ).

Монитор формирует две очереди сообщений – из первой сообщения записываются в файл ОТ и передаются на графическую консоль, из второй передаются по всем остальным направлениям (печать, GSM – сеть). По умолчанию размер очередей равен 64000 строк. Для индикации/изменения размеров очередей в реальном времени используется переменная **@q\_Queue\_Alarms** (группа ДИАГНОСТИКА). Если интенсивность генерации сообщений превышает скорость их выборки из очереди, очередь начинает расти. При достижении предельного размера очереди новые сообщения записываются поверх самых старых. Сообщение теряется, если по каким-либо причинам его не удалось вставить в очередь. Число потерянных таким образом сообщений для первой очереди индицирует переменная **@q\_Lost\_Alarms** (группа ДИАГНОСТИКА).

Максимальный размер файла ОТ ограничивается максимальным поддерживаемым размером файла в файловой системе. По достижении предельного размера файла, новые сообщения начинают записываться с начала ОТ(со второй строки). Предельно допустимое число строк в ОТ задается при его конфигурировании как один из параметров узла. Для диагностики ОТ предусмотрена переменная **@e\_Alarm\_Report** (группа ДИАГНОСТИКА), для создания резервной копии файла – **@Copy\_AR** (группа СИСТЕМНЫЕ).

**Формат строки ОТ.** Строка сообщения в отчете тревог содержит следующие предопределенные поля, разделенные пробелами:

**Date Time Category Name Coding Text UserID T\_ack N**, где

- **Date** и **Time** – дата и время события в форматах, заданных для ОТ в настройках узла. В зависимости от формата даты и времени, на них отводится соответствующее число знакомест в строке ОТ;
- **Category** – категория сообщения, (1 знакоместо **Категория**), задается символом: **<M> Сообщение**; **<W> Предупреждение**; **<E> Ошибка**; **<I> Информация**; **<A> Тревога**; **<R> Изменение атрибутов**; **<S> Пользовательское**; и т.д.);
- **Name** – для сообщения по каналу – имя канала; для системного сообщения, не связанного с каналом, – **<имя файла prj без расширения>\_<порядковый номер узла>**. При вводе пользовательского комментария в это поле записывается имя соответствующего канала класса **Пользователь**. На поле **Name** отводится 32 знакоместа;
- **Coding** – кодировка канала (21 знакоместо);
- **Text** – текст сообщения или пользовательский комментарий (48 знакомест);

- **UserID** – идентификатор пользователя, квитирававшего (подтверждение принятия) сообщение (5 знакомест). Сообщения могут квитираваться с помощью графических элементов;
- **T\_ack** – время квитиравания сообщения в формате **DD\_HH:MM:SS**;
- **N** – порядковый номер строки в ОТ в формате HEX (8 символов).

### 2.7.2. Архивы SIAD

Мониторы TRACE MODE поддерживают функцию записи значений атрибутов каналов в архивы SIAD. Эти архивы конфигурируются для узла при задании его параметров. Разрешение на архивирование и набор архивируемых атрибутов задаются для канала при его конфигурировании в редакторе или окне свойств. Если для канала задано архивирование, а флаг **Атрибуты** не установлен, сообщение в архив записывается при каждом изменении атрибута 0, **R** канала. Если для канала задано архивирование и установлен флаг **Атрибуты**, то, помимо записи сообщений об изменении атрибута 0, **R**, в архив записываются сообщения об изменении границ и атрибутов обработки канала.

Каждое сообщение в архиве содержит идентификаторы канала и атрибута, новое значение атрибута и время его изменения, а также некоторые другие параметры. Сохранение сообщений в архив реализовано в виде отдельного потока с более низким приоритетом по сравнению с основным потоком.

**Выборка и обработка данных SIAD.** Для выборки/обработки данных из архива в реальном времени используются каналы класса **CALL** с соответствующими типами вызова. Если подобный канал **CALL** имеет тип **OUTPUT**, он пересчитывается так же, как числовой канал **OUTPUT**. Если подобный канал **CALL** имеет тип **INPUT**, то он пересчитывается **R** раз со своим периодом (**R** – значение атрибута 0).

### 2.7.3. Индивидуальные (поканальные) архивы в памяти

С помощью каналов **CALL** могут быть созданы индивидуальные (поканальные) архивы в памяти. Данные таких архивов могут быть запрошены у удаленного узла по любому доступному интерфейсу. Для создания локального индивидуального архива используется канал **CALL** с типом вызова **LArc0** или **LArc1**. Канал, для которого создается архив, должен быть привязан к каналу **CALL**. Аргументы 0 и 1 канала **CALL** используются следующим образом:

- **arg0** – для служебных целей (в профайлере в этом аргументе отображается привязка канала CALL; тип данных **arg0** должен соответствовать типу данных архивируемого канала);
- **arg1** – значение этого аргумента (тип данных – любой целочисленный) задается как число секунд. Период запроса значения привязанного канала устанавливается монитором кратным периоду пересчета канала CALL и примерно соответствующим **arg1**. Если **arg1=0**, условия вычисления архивного значения (см. ниже) считаются заведомо выполненными.

Архивируемые данные записываются в последующие аргументы канала CALL. Эти аргументы используются парами (создаются вручную или автоматически): в аргумент с четным порядковым номером записывается архивируемое значение, в последующий аргумент с нечетным порядковым номером – время этого значения. При создании вручную для аргументов с четными порядковыми номерами должен быть задан соответствующий числовой тип данных, для аргументов с нечетными порядковыми номерами – **Date\_And\_Time**.

Модификации индивидуальных архивов определяются значением атрибута **Параметр** канала CALL:

- **Параметр** = 0 – архив реальных значений привязанного канала; этот архив представляет собой стек LIFO;
- **Параметр** = 1...60 – циклически перезаписываемый архив реальных значений привязанного канала, усредненных по интервалу 1...60 минут;
- **Параметр** = 61...120 – циклически перезаписываемый архив реальных значений привязанного канала, усредненных по интервалу 1...60 минут, с нарастающим итогом. Если не начат новый цикл записи в архив, архивное значение представляет собой сумму предыдущего архивного значения и среднего значения привязанного канала за последний интервал. При начале нового цикла записи в архив архивное значение представляет собой среднее значение привязанного канала за последний интервал.

Для запроса данных удаленных индивидуальных архивов используются каналы CALL с типами вызова **RemArc0** и **RemArc1**.



## 2.8. Средства программирования микропроцессорных контроллеров в TRACE MODE 6

Для программирования алгоритмов функционирования разрабатываемого проекта АСУ в TRACE MODE 6 включены языки **Техно ST**, **Техно SFC**, **Техно FBD**, **Техно LD** и **Техно IL**. Данные языки являются модификациями языков **ST** (Structured Text), **SFC** (Sequential Function Chart), **FBD** (Function Block Diagram), **LD** (Ladder Diagram) и **IL** (Instruction List) стандарта IEC61131-3. Вид интегрированной среды при редактировании программ показан на рис. 2.23.

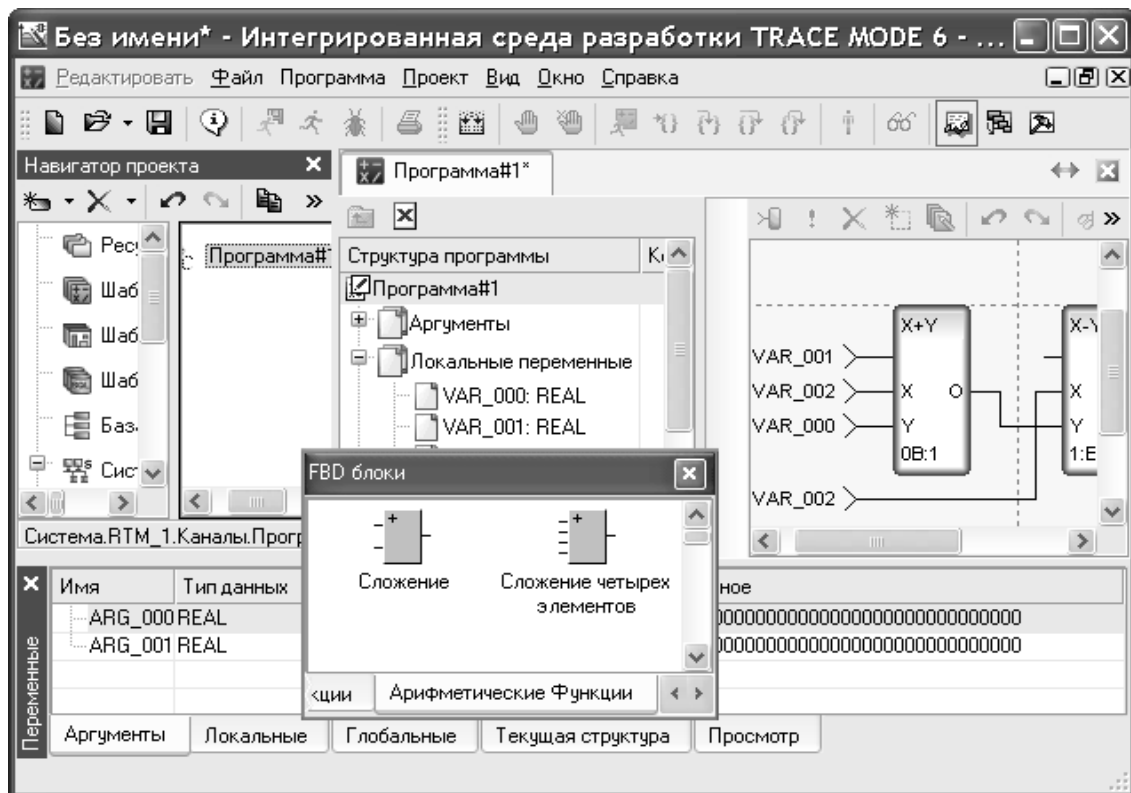


Рис. 2.23. Вид интегрированной среды разработки при редактировании программ

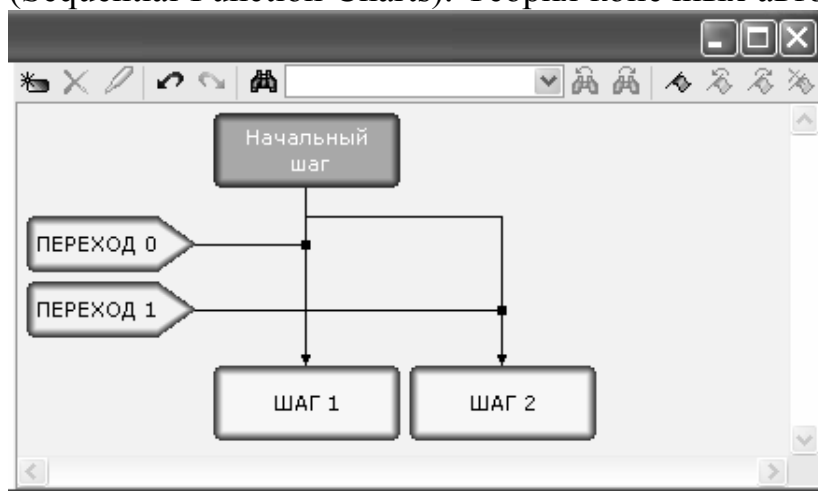
Основным языком программирования TRACE MODE 6 является **Техно ST**. Программы, разработанные на языках **Техно LD**, **Техно SFC** и **Техно FBD**, перед компиляцией транслируются в **Техно ST**. **IL** - программы перед компиляцией частично транслируются в **ST**, частично – в ассемблер.

Язык **ST** (Structured Text) относится к классу текстовых языков высокого уровня, его корни в таких известных языках программирования, как **ADA**, **PASCAL** и **C**. На основе этого языка можно создавать гибкие процедуры обработки данных; этот язык является основным для

программирования последовательных шагов и транзакций языка SFC, и, кроме этого, имеет «выходы» во все остальные языки, что делает его универсальным в применении разными категориями пользователей.

Язык инструкций **ИЛ** (Instruction List) в наборе стандартных языков - это унификация интерфейса языка программирования низкого уровня, не ориентированного на какую-либо микропроцессорную архитектуру. У языка ИЛ есть очень важное качество: на его основе можно создавать оптимальные по быстродействию программные единицы.

Язык последовательных функциональных схем (рис. 2.24) **SFC** (Sequential Function Charts).



Теория конечных автоматов, используемая для формализации состояний сложных процессов управления, опирается на различные графические модели описания состояний. Одной из наиболее известных является

*Рис. 2.24. Видеограма редактора SFC функциональных схем.*

ся модель, предложенная К. Петри, получившая название «сетей Петри» или диаграммы состояний. Она послужила теоретической основой языка SFC как наиболее важного из всего семейства стандартных языков.

Язык SFC позволяет формулировать логику программы на основе чередующихся процедурных шагов и транзакций (условных переходов), а также описывать последовательно-параллельные задачи в понятной и наглядной форме. Для SFC-шагов задаются выполняемые действия, для SFC-переходов – условия переходов между шагами, поэтому в дальнейшем SFC-переход иногда называется SFC-условием. Для перехода от одного шага к другому SFC-условие, действующее на этом переходе, должно быть истинным (т.е. возвращать TRUE или 1). Вид программы в SFC-редакторе показан на рис. 2. 24.

Строго говоря, SFC не является языком программирования – это средство проектирования прикладного ПО, которое всегда является комплексом большого числа программных единиц: программ, функциональных блоков, функций. Обеспечение параллельности выполнения программ, установление и контроль состояния порожденных про-

цессов, обеспечение синхронизации по приему и обработке данных, описание однозначно понимаемых и заказчиком, и исполнителем состояний автоматизируемого процесса - все это возможно при использовании SFC-языка программирования.

Основные достоинства SFC можно определить следующим образом:

- высокая выразительность. Язык SFC имеет те же возможности, что и диаграммы состояний, и является наиболее подходящим средством для описания динамических моделей;
- графическое представление. Благодаря графической мнемонике SFC является максимально легким в использовании и изучении. Вместе с тем он является наглядным средством представления логики на разных уровнях детализации;
- предварительное проектирование ПО. Использование языка SFC на ранних этапах проектирования прикладного ПО позволяет снять многочисленные непонимания между заказчиком, проектировщиком ПО и программистом.

Язык функциональных блоков **FBD** (Function Block Diagrams) позволяет создать программную единицу практически любой сложности на основе стандартных кирпичиков (арифметические, тригонометрические, логические блоки, ПИД регуляторы, блоки, описывающие некоторые законы управления, мультиплексоры и т.д.). Это языковое средство использует технологию инкапсуляции алгоритмов обработки данных и законов регулирования. Все программирование сводится к соединению готовых элементов в схему. В результате получается максимально наглядная и хорошо контролируемая программная единица.

FBD-программа представляет собой цепочку (диаграмму) последовательно выполняемых **функциональных блоков**. На рис. 2.25 показан вид окна программы, состоящей из двух блоков, в FBD-редакторе.

**Функциональный блок** – это графическое изображение вызова встроенной функции **Техно FBD** (FBD-блока) или функции (функции-блока), определенной пользователем. В верхней части FBD-блока выводится обозначение функции, выполняемой блоком (**X + Y** на рисунке). Именованные отрезки слева (**ARG\_000** и **ARG\_001**), обозначают входы блока (аргументы, переменные или константы функции). Отрезок без имени слева обозначает вход, управляющий выполнением блока (в дальнейшем – вход **RUN**). Блок выполняется, если **RUN=0** (значение по умолчанию). Отрезки, примыкающие к блоку справа, обозначают выходы блока (возвращаемые функцией значения).

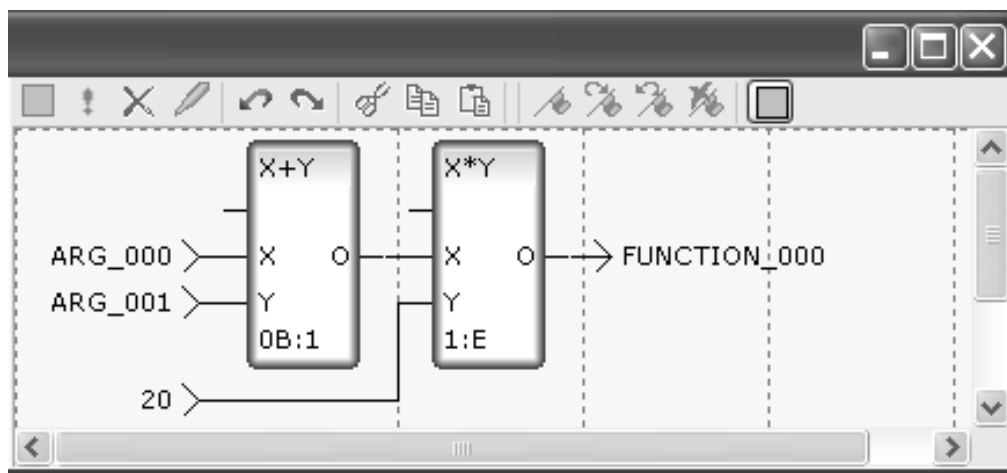


Рис. 2.25. Видеозапись рабочей области редактора FBD схем в TRACE MODE

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

- разместить необходимые функциональные блоки в рабочем поле FBD-редактора;
- соединить нужные входы и выходы блоков, образовав единую диаграмму;
- задать аргументы, переменные и константы программы;
- привязать входы/выходы FBD-диаграммы к аргументам, переменным и константам программы;
- скомпилировать программу.

Язык релейных диаграмм или релейной логики **LD** (Ladder Diagrams), применяется для описания логических выражений различного уровня сложности и использует в качестве базовых элементов программирования графические элементы «контакты» (contacts) и «катушки» (coils), связанные с входными и выходными каналами соответственно. Присутствие LD языка в стандарте определяется, скорее всего, данью традициям: огромное количество оборудования и алгоритмов было разработано для релейной техники. Сегодня, имея типовой набор цифрового ввода/вывода, можно создавать управляющие системы на отлаженной годами алгоритмической базе.

**LD**-программа представляет собой диаграмму последовательно выполняемых **функциональных блоков**. На рис. 2. 26 показан вид программы в LD-редакторе. **Функциональный блок** – это графическое изображение вызова встроенной функции **Техно LD** (LD-блока), функции (функции-блока), определенной пользователем, или FBD-блока.

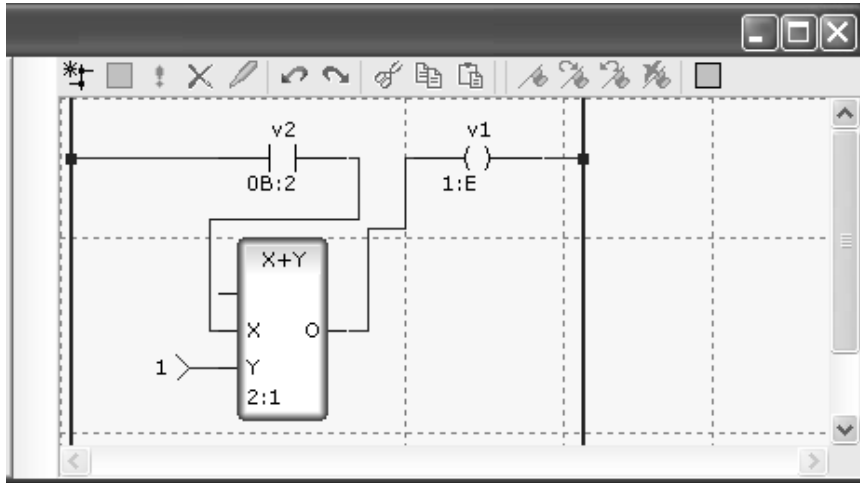



Рис. 2.26. Видеограма рабочей области редактора LD схем в TRACE MODE

Над LD-блоком выводится имя **связанной переменной** (v1, v2 на рисунке). Если связанная переменная не задана, над блоком отображаются три звездочки. В качестве изображения блока используется обозначение выполняемой этим блоком функции (| | и () на рисунках). Отрезок слева обозначает вход блока, отрезок справа – выход. Все LD-блоки имеют один вход (**in**) и один выход (**out**). Под блоком выводится его номер и, после двоеточия, номер следующего выполняемого блока (0B:2 на рисунках выше). Номера блоков задаются последовательно при их размещении в рабочем поле редактора; номера следующих выполняемых блоков определяются автоматически при размещении других блоков и соединении входов и выходов блоков (образовании диаграммы). На блоке, который выполняется первым в программе, после его номера отображается символ **B**; на блоке, который выполняется последним, – символ **E**. Используемые в программе FBD-блоки, а также функции и функции-блоки отображаются на LD-диаграмме в виде, аналогичном виду функциональных блоков в FBD-редакторе.

**Шины** изображаются на диаграмме в виде вертикальных линий. В **Техно LD** используются две **основные** шины (левая и правая) и **вспомогательные** шины. Между основными шинами размещаются все функциональные блоки LD-программы; на вспомогательные шины могут замыкаться выходы блоков, расположенных один над другим.

Независимо от языка на котором написана программа ее использование возможно только после ее успешной компиляции. Для компиляции программы нужно выполнить одно из следующих действий:

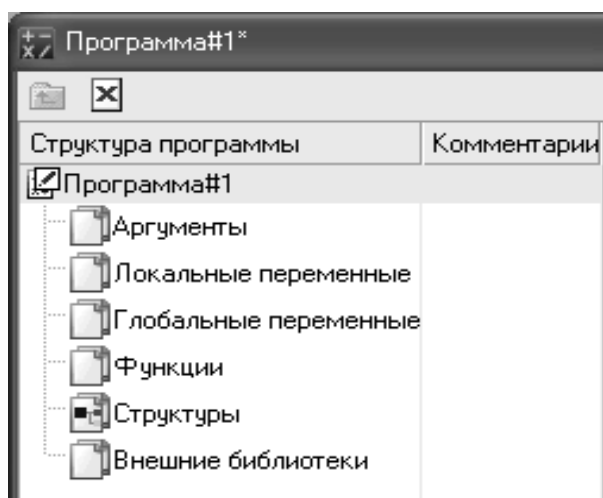
- выполнить команду **Компилировать** из меню **Программа** (или нажать клавишу **F7** или нажать ЛК на иконке  панели инструментов отладчика) – по этой команде создается только код для отладки программы в интегрированной среде разработки. Если

компилятор обнаруживает ошибки, он выводит соответствующие сообщения в окне, которое в этом случае открывается автоматически. Если компиляция прошла успешно, окно сообщений не открывается;

- выполнить экспорт проекта – по этой команде в папке узла, содержащего канал вызова программы, создается как отладочный, так и исполняемый код. При обнаружении ошибок в программе выводится сообщение о невозможности ее экспорта.

Для выполнения программы в реальном времени в узле должен быть создан канал класса CALL с типом вызова **Program**, настроенный на вызов шаблона программы. При данном типе вызова атрибуты **Параметр**, **Глубина выборки**, а также значение канала CALL не используются. Подобный канал CALL типа INPUT обрабатывается со своим периодом пересчета в соответствующем потоке. Подобный канал CALL типа OUTPUT обрабатывается при подаче 1 в его атрибут EXEC. В частности, для отработки такого канала из графики можно использовать функцию управления **Выполнить**.

**Вкладка «Структура программы».** В данном окне (рис. 2.27) в





виде дерева отображается структура программы. Чтобы открыть это окно, нужно дважды нажать ЛК на имени программы в навигаторе проекта.


Структурное дерево включает в себя все программные компоненты и используется для навигации по программе. При нажатии ЛК на любом элементе дерева автома-




*Рис. 2.27. Видеограмма окна просмотрщика структуры программы в TRACE MODE*

тически открывается соответствующий редактор. Созданный в табличном редакторе компонент (элемент) автоматически добавляется к структурному дереву. Окно структуры программы имеет панель следующих инструментов:

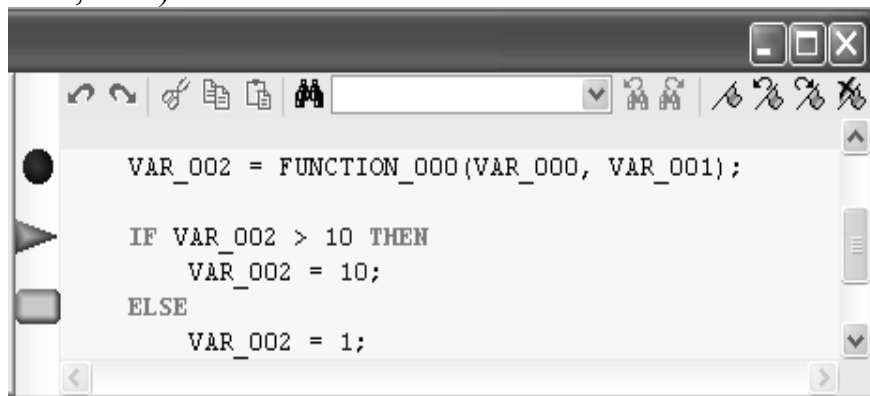
-  – переход на уровень определения программы, функции и т.п., к которой относится выделенный элемент;
-  – удаление тела выделенной программы, функции и т.п.

**Выбор языка программирования.** Язык программирования может быть независимо задан для основной программы, функции-блока, функции и шага SFC. Язык выбирается в диалоге автоматически появляющемся на экране при нажатии ЛК на имени вновь созданной программы или ее компонента (для которого язык может быть задан независимо) в окне структуры программы. После выбора языка программа (компонент) открывается в соответствующем редакторе.

Изменить язык можно только после удаления тела программы (компонента). Для этого нужно нажать ЛК на иконке  панели инструментов в окне структуры программы, после чего диалог выбора языка автоматически появляется на экране.

**Отладка программ.** Средства отладки (в том числе удаленной) включают в себя несколько режимов непрерывного и пошагового выполнения программы с возможностью установки точек останова. Для запуска требуемого режима отладки можно использовать команды меню **Программа** главного меню или аналогичные команды панели инструментов отладчика. Для вывода служебных сообщений отладчика и компилятора предусмотрены специальные окна. В листинге текстовых программ точка останова обозначается значком , закладка – значком . При пошаговой отладке текущий шаг обозначается значком .

В программах, заданных в графическом виде, закладки, текущий шаг и точки останова обозначаются соответствующим цветом (рис. 2.28, 2.29).



*Рис. 2. 28. Видеозапись окна отладчика программ в TRACE MODE*

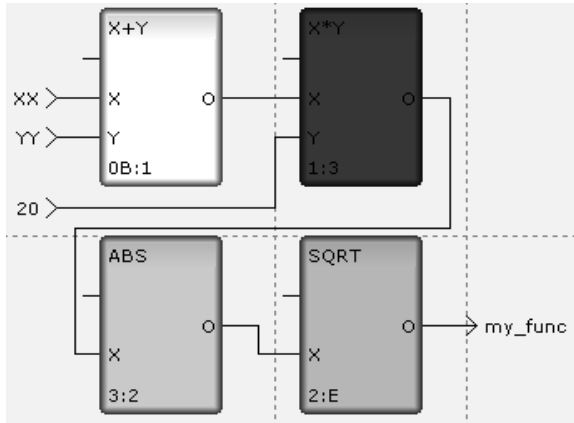


Рис. 2.29. Пример FBD программы

Меню **Программа** главного меню и панель инструментов отладчика содержат следующие команды для запуска требуемого режима отладки и настройки параметров редакторов программ:

- **Компиляция (F7)** – запустить компиляцию программы;
- **Установить/удалить точку останова (F9)** – установить/удалить точку останова программы;
- **Удалить все точки останова (Ctrl+Shift+F9)** – удалить все точки останова программы;
- **Удаленная отладка** – переключатель вида отладки: **локальная** (отжатое состояние) или **удаленная** (нажатое состояние);
- **Старт (F5)** – запустить выполнение программы в непрерывном режиме;
- **Выполнять до курсора (Ctrl+F10)** – запустить выполнение программы в непрерывном режиме до текущей позиции курсора;
- **Трассировка (F11)** – запустить пошаговое выполнение программы с пошаговым выполнением вызываемых функций;
- **Шаг (F10)** – запустить пошаговое выполнение программы с выполнением вызываемых функций в непрерывном режиме;
- **Выйти из функции (Shift+F11)** – выполнить текущую функцию/программу в непрерывном режиме (доступно в режиме отладки);
- **Стоп (Shift+F5)** – выйти из режима отладки;
- **Посмотреть значение переменной (Shift+F9)** – открыть диалог **Быстрый просмотр** (доступно в режиме отладки);
- **Переменные** – открыть/закрыть окно переменных;
- **Стек** – открыть/закрыть окно стека вызовов функций;
- **Сообщения** – открыть/закрыть окно сообщений компилятора и отладчика.



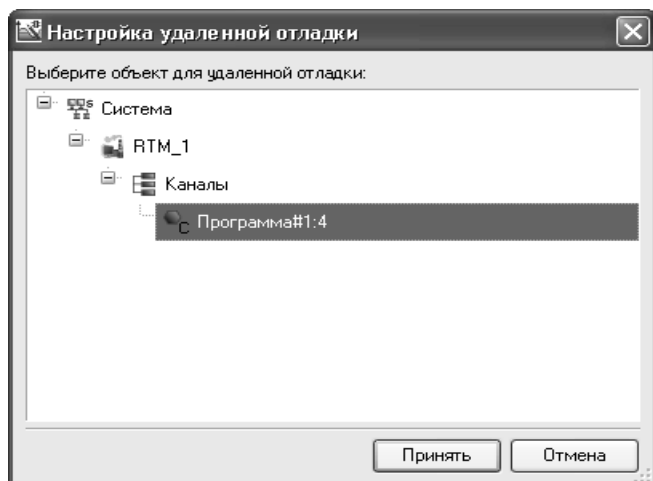


Рис. 2.30. Видеограмма окна настройки удалённой отладки программ в TRACE MODE

**Удаленная отладка.** При удаленной отладке ИС запрашивает значения аргументов и глобальных переменных программы у узла (рис. 2.30), выполняемого под управлением монитора на удаленном компьютере или в контроллере. Это позволяет отлаживать программу на реальных значениях

обрабатываемых сигналов. При переходе к данному виду отладки нужно выбрать программу в следующем диалоге:

**Окно просмотра переменных.** Данное окно включает в себя в виде вкладок 5 окон просмотра текущих значений переменных (рис. 2.31). В окне **Локальные** отображаются переменные текущего программного компонента (в том числе переменные объектов, определенных в текущем компоненте, – на рисунке таким объектом является **var\_002**).

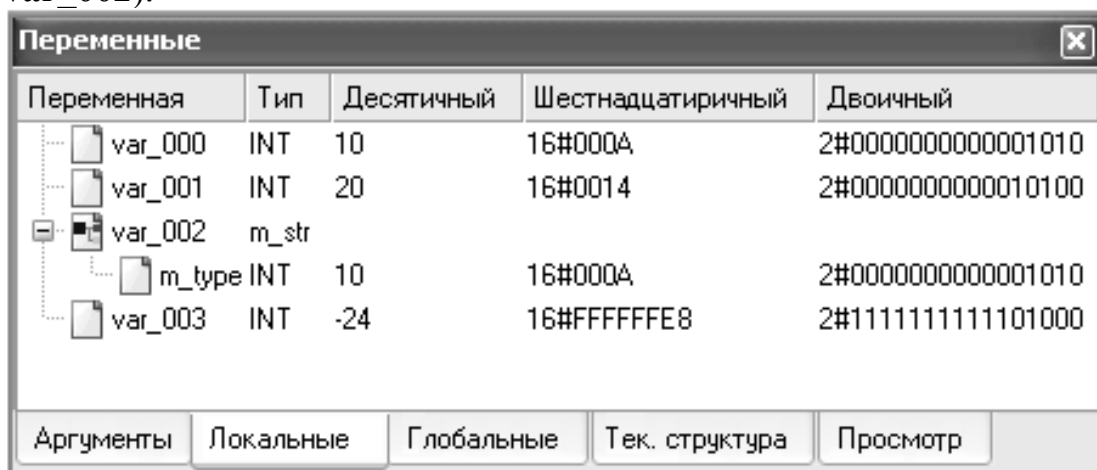


Рис. 2.31. Видеограмма окна редактора переменных в TRACE MODE

В окнах **Аргументы**, **Глобальные** и **Текущая структура** отображаются соответственно аргументы, глобальные переменные и переменные объекта. При пошаговой отладке программы в окнах **Локальные** и **Глобальные** можно вручную задать значения переменных (для перехода к заданию значения нужно дважды нажать ЛК в поле значения или выполнить команду **Изменить значение** из контекстного ме-

ню). Для восстановления значений аргументов по умолчанию контекстное меню окна **Аргументы** содержит команды **Восстановить значение по умолчанию** и **Восстановить все значения по умолчанию**.

В окне **Просмотр** отображаются переменные и выражения, заданные пользователем. По умолчанию оно пустое. Список переменных и выражений для просмотра задается с помощью диалога **Быстрый просмотр**. Корректировать список можно в самом окне с помощью команд контекстного меню, которое выводится на экран при нажатии ПК в области окна. Меню содержит следующие команды:

**Вставить** – добавить строку в список;

**Удалить** – удалить выделенную строку из списка;

**Переименовать** – редактировать выделенную строку списка.

Текущие значения переменных отображаются на диаграммах FBD и LD (рис. 2.32, 2.33).

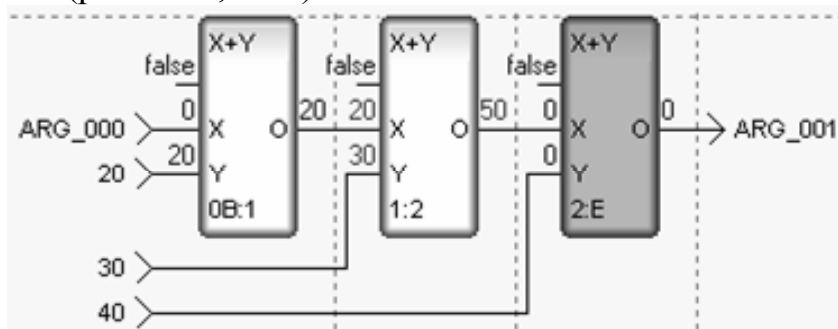


Рис. 2.32. Диаграмма FBD программы с входными и выходными параметрами

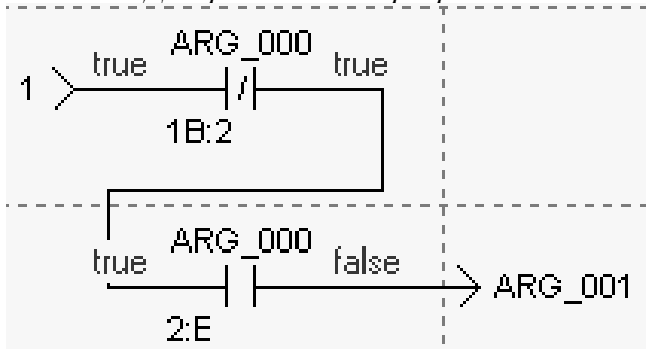
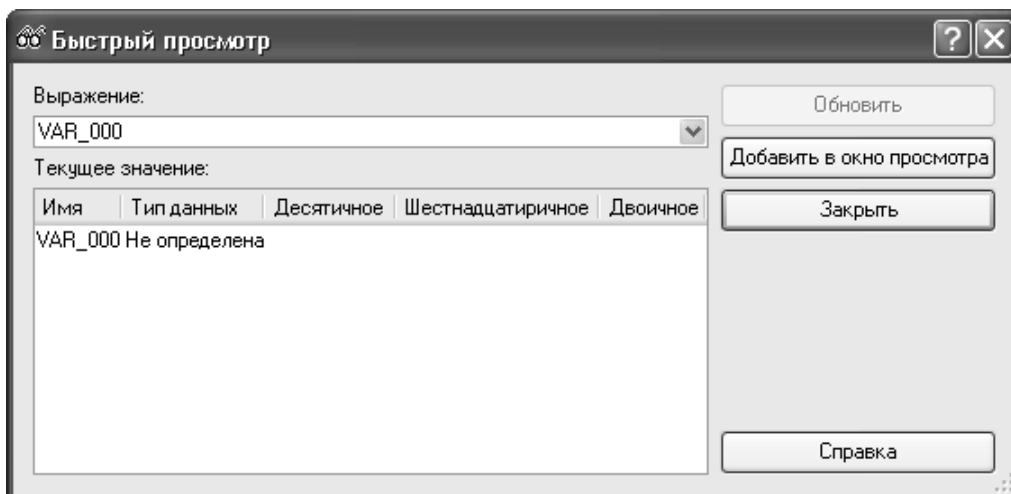


Рис. 2.33. Диаграмма LD программы с входными и выходными параметрами

**Диалог «Быстрый просмотр».** Данный диалог используется для оценки значения переменных и выражений в процессе отладки, а также для задания списка переменных и выражений для отображения в окне-вкладке **Просмотр** окна просмотра переменных. Для входа в диалог нужно в режиме отладки нажать кнопку  инструментальной панели или выполнить команду **Посмотреть значение переменной** меню **Программа** (при отладке в цикле диалог недоступен). В поле **Выражение** задается переменная или выражение, для оценки переменной или выражения нужно нажать кнопку **Обновить**, для добавления пе-

ременной или выражения в окно **Просмотр** – кнопку **Добавить** (рис. 2.34).



*Рис. 2.34. Видеозапись окна быстрого просмотрщика значений переменных в TRACE MODE*

### 3. Проектирование ПО с помощью программного комплекса Matlab

Matlab – лаборатория матричных математических вычислений. Matlab начал развиваться ещё в 60 годах, когда потребовался высокоуровневый язык для эффективных матричных вычислений. В данное время Matlab – это один из представителей серьёзных программных комплексов, ориентированный не только на поддержку математического ядра с численными вычислениями, но и, ориентированный на символьные вычисления, за счёт поддержки ядра Maple. Matlab поддерживает виртуальное проектирование за счёт интеграции с языком виртуальных блоков Simulink, осуществляет поддержку основополагающих языков программирования за счёт встроенных возможностей работы с компилятором (Matlab Compiler Driver), таких как Java, C, C++, C#, FORTRAN. Matlab активно работает с внешними программами за счёт собственных функций интерпретатора, он выступает интерпретатором команд Windows, Unix, DOS, языка Perl. Так же поддерживает специализированные ОС ориентированные интерфейсные модули-функции MEX. Matlab активно использует поддержку наиболее популярных драйверов для работы с технологическим оборудованием и проектирование виртуальных приборов для работы электронных, программных щитов. Язык М, который входит в состав Matlab является универсальным высокоуровневым языком, поддерживающим несколько архитектур программирования: скрипт, функциональное программирование, ООП. Подобные возможности сокращают срок программирования необходимого программного модуля для обработки экспериментальных данных до нескольких недель, а понятный и простой по синтаксису язык М не требует особых знаний в программировании.

Поскольку возможности Matlab так широки, а перспективы развития язык М и пакета Matlab очевидны, в этой главе мы рассмотрим вопросы, касающиеся разработки программных модулей для АСУ ТП, применительно к обработке экспериментальных данных.

**NOTE:** *Стиль и состав изложенного в этой главе текста, предполагает, что читающий имеет базовые знания в области работы с языком М, средой Matlab и Simulink!*

### 3.1. Технологии программировании GUI в Matlab

Вполне очевидно, что программирование приложений общего целевого пользования невозможно без графического пользовательского интерфейса. Оператор не умеет программировать на языке программирования, а программа не может управляться без использования команд на языке программирования, которые знает программист.

Анализ среды проектирования Matlab показал, что в Matlab возможны две технологии программирования ГПИ (рис. 3.1).

|                        | Разработка графического интерфейса пользователя путём непосредственного программирования на целевом языке программирования                      | Использование визуальной среды разработки графического интерфейса пользователя   |
|------------------------|---|--|
| Назначение             | Программирование необходимых алгоритмов работы с данными и графических интерфейсов представления данных с использованием классов и объектов     | Программирование графических объектов при помощи специальной среды разработки с инспектором свойств объектов   |
| Особенности            | Использование специальных команд языка программирования для управления объектами и выполнения специализированных функций                        | Возможность проектирования графического интерфейса пользователя без использования специальных знаний о языке программирования                            |
| Положительные качества | Эффективный код программы, полный контроль над работой алгоритмов, возможность быстрой отладки программ, высокая скорость разработки программ   | Возможность быстрого позиционирования графических объектов в рабочей области   |
| Недостатки             | Необходимость знания широкого спектра специальных функций, наличие длительного процесса позиционирования разнотипных объектов в рабочей области | Отсутствие контроля за действиями конструктора объектов, невозможность оптимизации конечного кода программы, увеличенное время выполнения конечного кода |

Рис. 3.1. Технологии программирования GUI в Matlab

- Непосредственное программирование на целевом языке с использованием синтаксиса команд;
- Использование специального конструктора графических интерфейсов (**GUIDE**).

**Достоинства технологии непосредственного программирования:**

- Низкая избыточность кода;
- Полный контроль над проектированием кода;
- Малое время выполнения кода;
- Быстрая отладка и переработка кода.

**Недостатки технологии непосредственного программирования:**

- Длительное время позиционирования ГЭ в графической области;
- Необходимость специальных знаний объектов и классов объектов Matlab, а так же синтаксиса команд языка М;

**Достоинства технологии проектирования на основе GUIDE:**

- Универсальный межплатформенный код;
- Простота позиционирования ГЭ в графической области;
- Нет необходимости в специальных знаниях в области программирования ГПИ в Matlab;

**Недостатки технологии проектирования на основе GUIDE:**

- Избыточный код;
- Длительное время выполнения кода;
- Длительное время отладки и переработки кода;
- Отсутствие контроля над выполнением кода.

В качестве примера проектирования, используя эти технологии, рассмотрим текст программ написанных с использованием технологии непосредственного программирования и программирования с использованием GUIDE.

Результат проектирования на основе технологии непосредственного программирования (рис. 3.2, 3.3).

```
1 function programm(varargin)
2 % Пример программы, для выполнения функции сложения 2х чисел
3 % AntoXa Inc. 02.02.2007 10:51:44
4 %*****
5 - figure('na', 'Программа сложения двух чисел', 'num', 'off', 'menu', 'none', 'color', [.7529 .7529 .7529]);
6 - uicontrol('sty', 'te', 'str', 'Первое число', 'pos', [100 300 80 15]);
7 - uicontrol('sty', 'ed', 'tool', 'Введите первое число для сложения', 'pos', [100 280 80 20], 'tag', '1', 'call', @
8 - uicontrol('sty', 'te', 'pos', [190 280 20 20], 'str', '+');
9 - uicontrol('sty', 'te', 'str', 'Второе число', 'pos', [220 300 80 15]);
10 - uicontrol('sty', 'ed', 'tool', 'Введите второе число для сложения', 'pos', [220 280 80 20], 'tag', '2', 'call', @
11 - uicontrol('sty', 'te', 'pos', [310 280 20 20], 'str', '=');
12 - uicontrol('sty', 'te', 'tool', 'Результат', 'pos', [340 280 80 20], 'tag', '3');
13
14 function edit(varargin)
15 % Расчётное ядро
16 % AntoXa Inc. 02.02.2007 11:01:40
17 %*****
18 - s1 = str2num(get(findobj(0, 'tag', '1'), 'str'));
19 - s2 = str2num(get(findobj(0, 'tag', '2'), 'str'));
20 - if (isempty(s1) == 0 & isempty(s2) == 0)
21 -     set(findobj(0, 'tag', '3'), 'str', num2str(s1+s2));
22 - end;
```

**Рис. 3.2.** Текст программы *programm.m*

Результат представлен 22 строками текста.

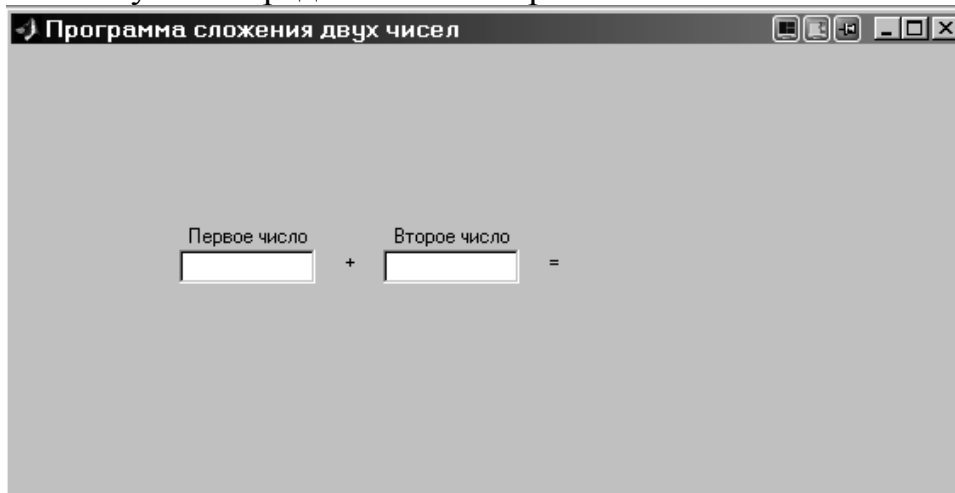


Рис. 3.3. Результат выполнения программы *programm.t*

Результат программирования на основе торого метода представлен на рис. 3.4, 3.5.

```
426 -         set(gui_hFigure,'HandleVisibility', 'on');
427 -     end
428 -     gui_Handles = guidata(gui_hFigure);
429 - else
430 -     gui_Handles = [];
431 - end
432
433 - if nargin
434 -     [varargout{1:nargout}] = feval(gui_State.gui_OutputFcn, gui_hFigure, [], gui_Handles);
435 - else
436 -     feval(gui_State.gui_OutputFcn, gui_hFigure, [], gui_Handles);
437 - end
438
439 - if ishandle(gui_hFigure)
440 -     set(gui_hFigure,'HandleVisibility', gui_HandleVisibility);
441 - end
442 - end
443
444 - function gui_hFigure = local_openfig(name, singleton)
445 - if nargin('openfig') == 3
446 -     gui_hFigure = openfig(name, singleton, 'auto');
447 - else
448 -     % OPENFIG did not accept 3rd input argument until R13,
449 -     % toggle default figure visible to prevent the figure
450 -     % from showing up too soon.
451 -     gui_OldDefaultVisible = get(0,'defaultFigureVisible');
452 -     set(0,'defaultFigureVisible','off');
453 -     gui_hFigure = openfig(name, singleton);
454 -     set(0,'defaultFigureVisible',gui_OldDefaultVisible);
455 - end
```

Рис. 3.4. Часть текста программы *programm1.m*

Результат представлен 455 строками текста.



*Рис. 3.5. Результат выполнения программы `program1.m`*

Исходя из полученного результата видно, что количество строк текста второго кода на порядок больше, а исходя из временных показателей программа, написанная с использованием второй технологии требует большего времени для выполнения.

На основании проведённого анализа сделаем вывод, что построение программного кода методом непосредственного программирования – более эффективно.

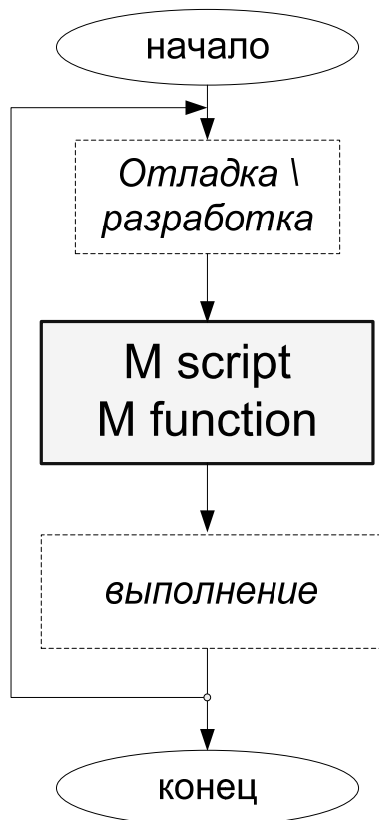
### **3.2. Алгоритмы программирования и верификации в среде Matlab**

Любой процесс программирования в Matlab сводится к созданию кода написанного на языке М – М-функции.

Простейший алгоритм программирования кода на языке М в среде Matlab представлен на рис. 3.6.

Алгоритм, показанный на рис. 3.6, является основным на начальной стадии программирования и состоит из этапов разработки, выполнения и отладки М кода. Однако, для поддержки расширенной структур М кода, поддержки МEX функций и исполняемых файлов, необходимо применить расширенный алгоритм программирования.



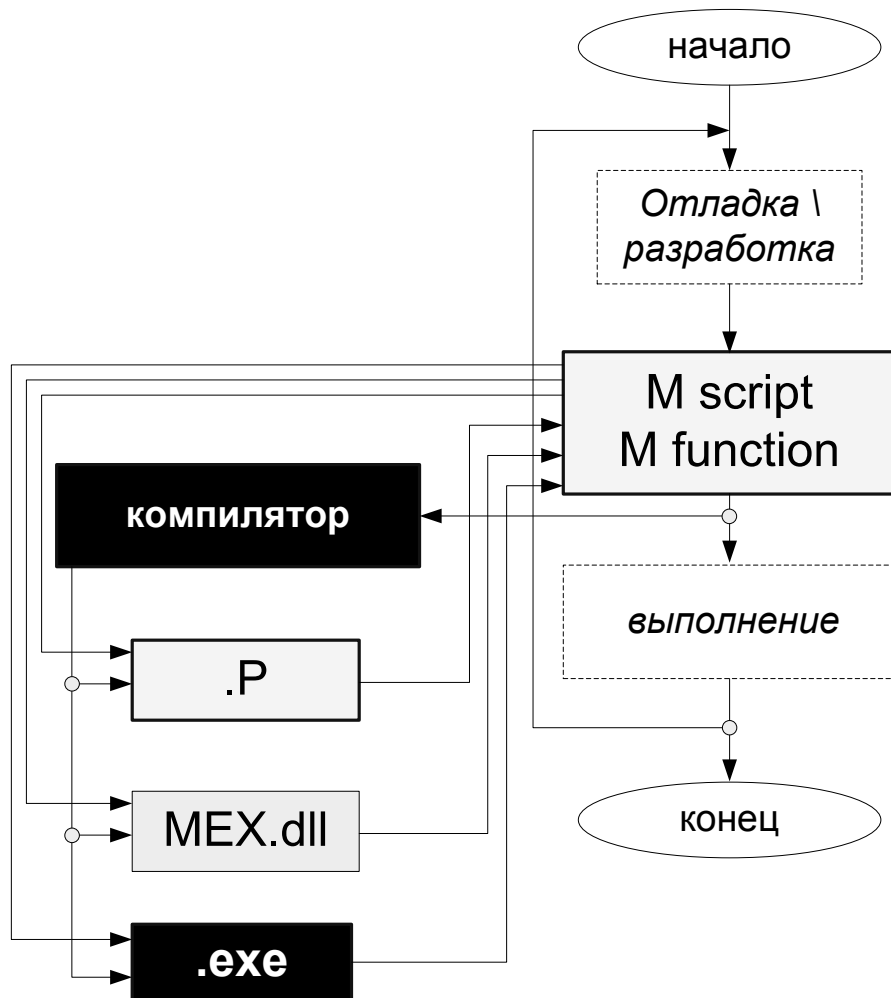


*Рис. 3.6. Простейший алгоритм программирования M-функции*

Процесс разработки (рис. 3.6) представлен этапами отладки в текстовом редакторе (встроенном или внешнем), в конце которого имеется готовый M код программы. Процесс отладки сопровождается периодическим выполнением. Алгоритм повторяется до тех пор, пока результат выполнения программы не будет удовлетворять требованиям программиста.

Расширенный алгоритм программирования M-кода представлен на рис. 3.7. Он состоит из простейшего алгоритма, в конце которого полученный M код с использованием компилятора транслируется или компилируется в один из выходных кодов. Выходные модули кодов, в дальнейшем используются как вместо M кода, так и в качестве дополнительных функций в текстах других M-функций.

Рассмотрим основные особенности P, MEX и EXE-кодов.



**Рис.3.7.** Алгоритм программирования и структура расширенного кода М. (Р, МЕХ – свободно-распространяемые модули функционального расширения)

### **Использование Р-кода**

Особенности кода Р:

- Код Р – псевдо код, созданный на основе байт кода;
- Поддерживается в Matlab на уровне М-функции;
- Не позволяет редактировать и открывать код в среде Matlab;
- Формат кода: ASCII.
- Позволяют осуществить взаимодействие с М-функцией через оперативную память без использования дополнительных средств взаимодействия.

Использование кода Р в Matlab позволяет:

- Повысить скорость выполнения программ в Matlab;
- Скрыть участки программы, являющейся интеллектуальной собственностью разработчика.

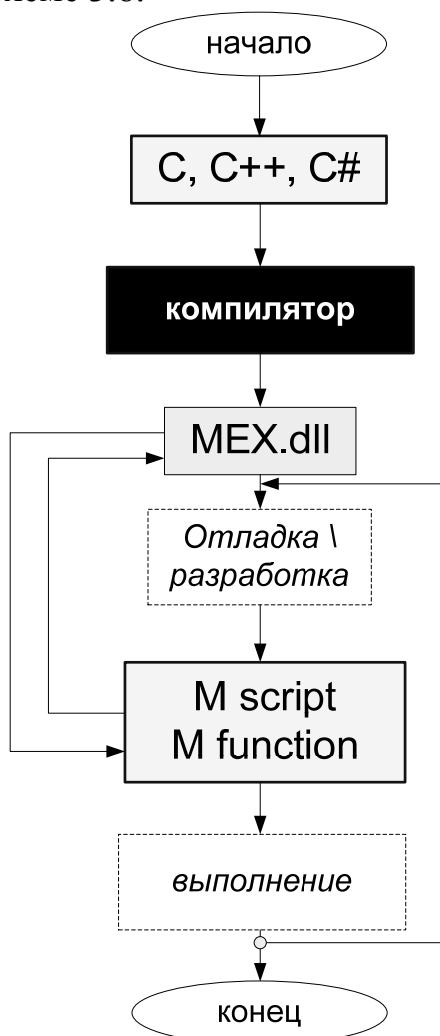
Этот код мы будем использовать при создании коммерческой версии программы, для того, чтобы закрыть часть кода программы.

### **Использование MEX функций**

Особенности MEX функций:

- Поддерживаются в Matlab на уровне М функций;
- Позволяют использовать сторонние коды (С, С++, С#, Fortran) в составе М-кода;
- Позволяют осуществить взаимодействие с М-функцией через оперативную память без использования дополнительных средств взаимодействия.

Рассмотрим алгоритм создания и использования MEX-функций на схеме 3.8.



**Рис. 3.8.** Алгоритм интеграции С-код в М-код с последующим использованием

Процесс создания MEX-кода начинается с того, что разработчик, имея М код, который в последствии транслируется Matlab в С-код, ли-

бо имея уже готовые С коды, компилирует МЕХ функцию. Полученную МЕХ-функцию программист использует в процессе работы с М-кодами или отдельно от Matlab.

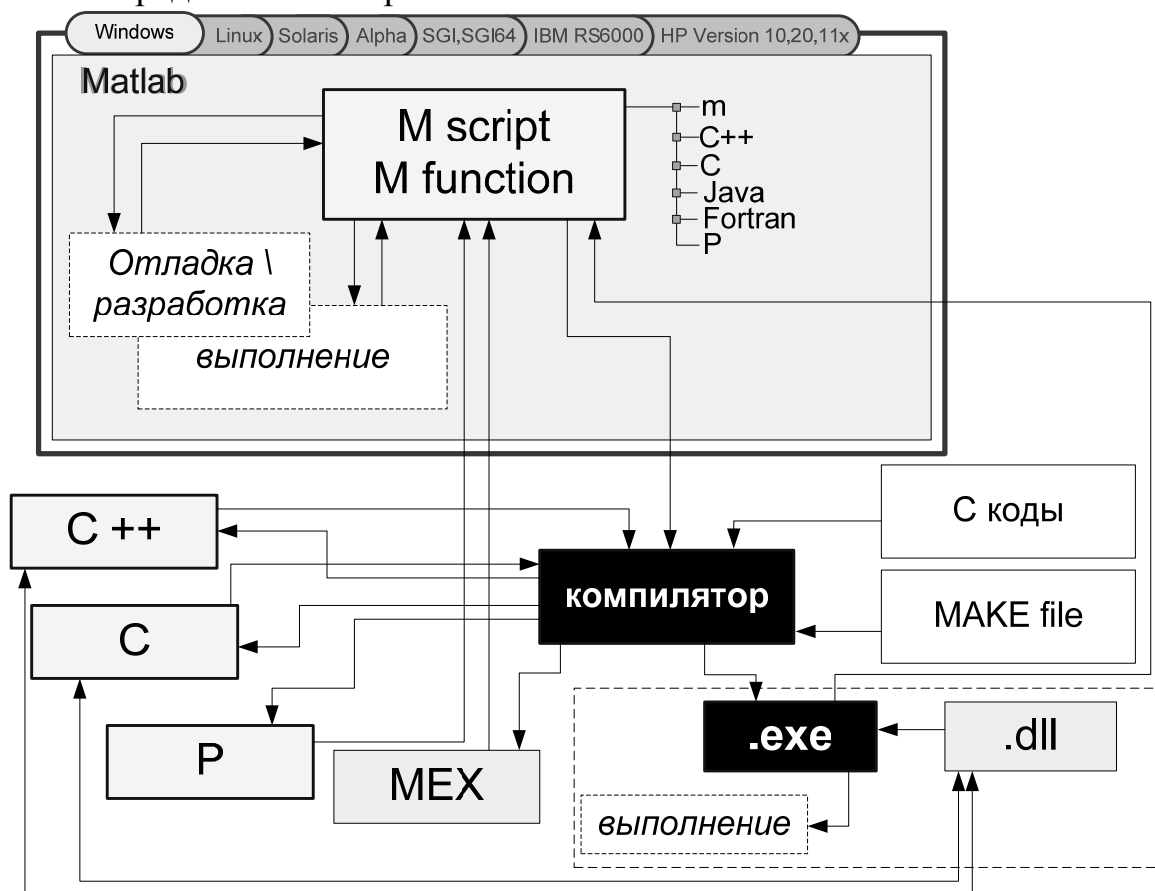
Подобный алгоритм мы будем использовать для создания собственных МЕХ-функций.

### **Использование свободно исполняемого кода EXE**

Особенности свободно исполняемого кода EXE:

- Код EXE позволяет выполнять необходимые функции без использования среды Matlab;
- Код EXE имеет малое время выполнения задач.

Общая схема процессов программирования и верификации ПО в Matlab представлена на рис. 3.9.



**Рис. 3.9.** Сводная схема алгоритмов программирования в среде Matlab.

Схема 3.9 предполагает создание в среде Matlab М-кода с использованием С, С++, Java и других поддерживаемых Matlab языков. Затем, текст программы может быть откомпилирован или транслирован с использованием одного из поддерживаемых компиляторов. При этом в

процессе трансляции участвуют исходные С коды интерпретатора Matlab а так же MAKE файлы определяющие процесс работы компилятора. Свободно исполняемым от Matlab кодом является только EXE, который использует стандартные библиотеки \*.dll Matlab для исполнения операционной средой. Компилятор так же используется для создания кодов Р, МЕХ или С, которые используются при выполнении М кодов как дополнительные функции. При компилировании или транслировании, Matlab включает их в состав конечного кода.

### 3.3. Архитектура М программы с использованием ГПИ

Известно, что в Matlab все функции написаны на М-языке и расположены в директориях Matlab в виде отдельных М-файлов(функций). Обращение в отдельно взятой функции, приводит к выполнению М-файла с соответствующим названием. На самом деле, поскольку язык М является функциональным, то в одном и том же М-файле может содержаться и несколько М-функций. Таким образом, мы получаем архитектуру программы представленную на рис. 3.10.

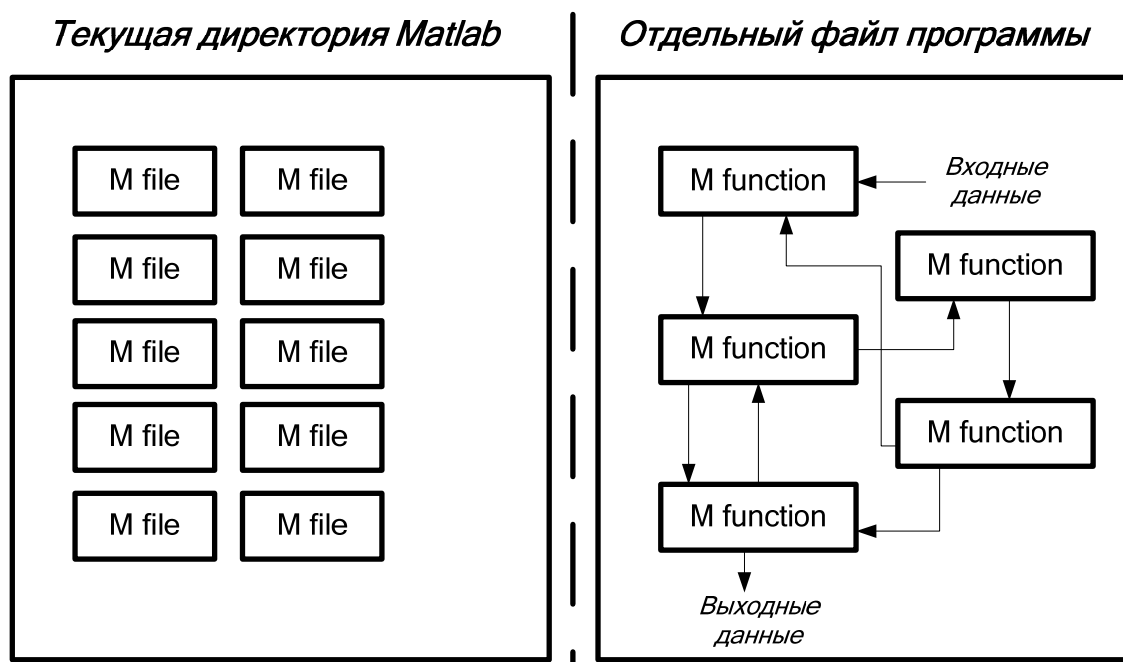


Рис. 3.10. Архитектура М-программы: совокупность М-файлов (слева), один М-файл программы (справа)

При этом вызов сторонних функций непосредственно из текста функции будет идентичной, как для первого, так и для второго варианта архитектуры программы, а вызов функции из графического объекта будет отличаться.

Поставим задачей проверить два типа вызова М-кодов. Непосредственно из текста `m` кода и из объекта `uicontrol(callback)`.

Рассмотрим примеры программ двух типов.

Текст программы, реализующий первый вариант представляет собой два М-файла, первый производит обращение ко второму файлу, а второй файл производит вычисление по заранее известной формуле и возвращает результат в первый файл. Каждый файл является отдельной функцией.

```
function fun1
% первый М-файл
% выполним обращение ко второму модулю
out = fun2(4);
% распечатаем результат
helpdlg(int2str(out), 'А вот и результат');
```

```
function out = fun2(in)
%второй М-файл
out = sqrt(in*2 +1);
```

Очевидно, что если выполнить первый модуль в Matlab, то мы получим ответ.

Теперь откомпилируем функцию `fun1` и выполним в Windows.

Команда компилятору: **`mcc -m -B sgl fun1`**.

Программа работает.

Программа-компилятор автоматически настроила адресацию и при компилировании первого модуля учла во внимание адресацию на другой модуль.

Теперь выполним второй тип вызова функций. В этом случае в одном М-файле запишем тексты обеих функций. В случае не использования графических элементов мы получим работоспособный код, который не будем рассматривать. Рассмотрим код с использованием ГЭ, в поле `callback` которого используем обращение к функции расчётной.

```
function fun1
% М-файл fun1 и функция fun1
% выполним обращение ко второму модулю
figure;
uicontrol( ...
'Style', 'pushbutton', ...
'String', 'Start', ...
'Callback', 'out = fun2(4); helpdlg(int2str(out), "А вот и результат");');

function out = fun2(in)
%М-файл fun1 и функция fun2
out = sqrt(in*2 +1);
```

Выполним первую функцию в среде Matlab, очевидно, что она не работает.

Теперь скомпилируем функцию в exe-код с помощью команды:

```
mcc -m -B sgl fun1
```

Очевидно, что программа определила связь на несуществующую функцию. Поскольку при нажатии на кнопку программы пытается найти в текущей директории файл с названием *fun2.m* которого нет.

Теперь воспользуемся следующим методом, способным решить проблему.

Пример программы построен с использованием функций @ и глобальных переменных:

```
function fun1  
global d  
d = 4;  
% выполним обращение ко второму модулю  
figure;  
uicontrol( ...  
  'Style', 'pushbutton', ...  
  'String', 'Start', ...  
  'Callback', @fun2);  
  
function varargout = fun2(varargin)  
global d;  
out = sqrt(d*2 + 1);  
helpdlg(int2str(out), 'А вот и результат');
```

При запуске в среде Matlab программа работает. Попробуем откомпилировать её и выполнить как независимое приложение.

```
mcc -m -B sgl fun1
```

Как видим, программа работает. В этом примере, мы интегрировали, второй модуль в первый и в параметре callback использовали косвенную адресацию (вызов функции) на функцию @fun2. Дело в том, что при использовании параметра callback действие программы равносильно тому, что вы вводите команду в строке команд Matlab. При этом невозможно обратиться к выполняемой в данный момент программе, и вам приходится делать ещё один модуль, например fun2. Но обращение программы *fun1* к *fun2* через объект *uicontrol* – *callback* при компилировании не учитывается, так как отследить действия программиста в этой области не возможно. При использовании косвенного обращения к функции вы как бы пользуетесь динамическими объектами и в то же время находитесь в рамках *m* кода. Однако, при косвенном обращении к функции невозможно использовать входные и выходные переменные, поэтому будем применять глобальные переменные *global*,

которые позволяют использовать переменную в тех М-файлах, и функциях, в которых она объявлена.

Подобное решение значительно упрощает и расширяет функции работы с интерфейсом. Использование подобного метода позволит более эффективно и наглядно построить программный код.

Тем же самым методом будем пользоваться при работе с внешними кодами программ и модулей, например с МEX.

Как уже было сказано ранее, использование динамических *.dll* библиотек, оправдано для работы среды Matlab в купе с Windows. Это неоспоримый факт, который существует со времён создания первых языков программирования и самих ОС.

Как мы только что видели, для того, чтобы обратиться к внешнему файлу из Matlab, можно пользоваться любыми способами доступа. Но при работе с исполняемым модулем *exe* необходимо применить некоторую хитрость – косвенную адресацию. Соответственно, для того, чтобы в среде Matlab оказались обработанные в *.dll* данные необходимо выполнить адресацию из *m* кода, а если вы используете объект *ui-control* с параметром *callback*, то этот *m* код должен вызываться через значок *@*. Попробуем написать пример *m* функции, реализующей эти операции.

Для начала создадим *program\_kernel\_fun.dll* в которой будет зашита некоторая математическая функция.

Текст программы приведён ниже:

```
function out = program_kernel_fun(in)
% яро программы
%+++++
out = in + 2*in^2;
```

Команда компилятору: *mcc -x program\_kernel\_fun.mt*.

А теперь создадим функцию управления которая выполнена в одном М-файле.

```
function initial_fun
% функция, которая обращается к dll
%+++++
global d;
figure;
uicontrol('Style', 'pushbutton', ... % кнопка управления
'String', 'RUN ME', ...
'Callback', @calldll);

%-----
% дополнительные функции
%-----
```



```

function varargout = calldll(varargin)
% функция обращающаяся к dll
%+++++
global d;
d = program_kernel_fun(3); %функция преобразования
helpdlg(int2str(d), 'А вот и ответ'); % вывод ответа

```

Команда компилятору: **mcc -m -B sgl initial\_fun.m**

Теперь можно запустить модуль **initial\_fun.exe**. Результат очевиден, программа работает.

Такой метод позволяет использовать библиотеки компонентов и свободно исполняемые модули, написанные на интерпретируемом языке М с использованием ГЭ, и затем откомпилированные.

### 3.4. Проектирование ГПИ (GUI) в Matlab

Поскольку язык М, который входит в состав пакета Matlab, является ООП, то логичным выводом напрашивается использование его достоинств для быстрой разработки GUI. Концепции ООП нам уже известны и они позволяют работать с графическими объектами, как с автономными модулями. Программирование, в этом случае, сводится лишь к конфигурированию параметров каждого графического объекта в области окна программы. Давайте рассмотрим на примере среды Matlab 6.5 R13 способы программирования GUI.

Структура графических классов и объектов в Matlab представлена на рис. 3.11. Описание графических объектов представлено в табл. 3.1.

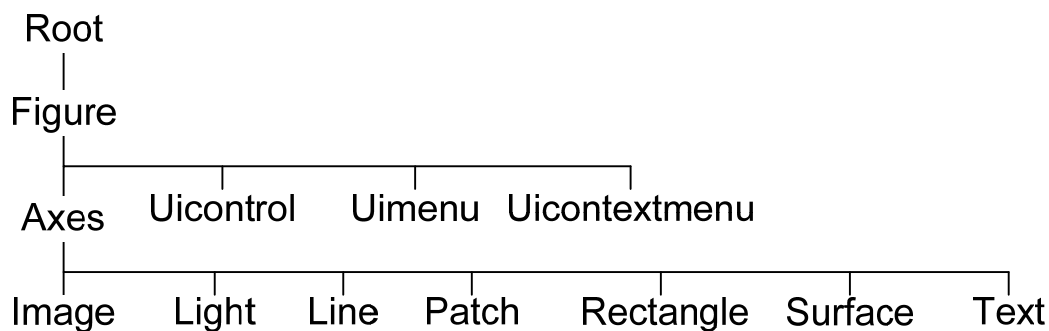


Рис. 3.11. Структура графических классов в Matlab

Табл. 3.1. Описание графических классов Matlab

| Объект | Описание  |
|--------|---|
| Root   | Верхушка иерархии, представляющая экран монитора                  |
| Figure | Окно программы, представляющее графический интерфейс пользователя |

|               |   |
|---------------|---|
| Axes          | Область системы координат для построения графиков и поверхностей  |
| Uicontrol     | Класс представляет набор динамических графических объектов управления (кнопки, флаги, слайдеры, редакторы текста) |
| Uimenu        | Меню окна программы   |
| Uicontextmenu | Контекстные меню в области окна программы, вызываемые правым кликом на графическом объекте                        |
| Image         | Двумерные растровые картинки  |
| Light         | Источник света изменяющий цветовую карту объекта на который он направлен  |
| Line          | Линия   |
| Path          | Конфигурируемый полигон   |
| Rectangle     | Двумерная изменяемая форма  |
| Surface       | Трёхмерные матричные поверхности  |
| Text          | Конфигурируемый статический текст с возможностью редактирования   |

Для дальнейшего построения GUI рассмотрим простейшие типы графических объектов в виде кодов программы на языке М:

### Объект «Кнопка без фиксации»

```
function x(varargin)
uicontrol('style', 'pushbutton', ...
'string', 'run me', ...
'horizontalalignment', 'left', ...
'fontsize', 14, ...
'units', 'normal', ...
'position', [.1 .1 .4 .1], ...
'backgroundcolor', [0 0 0], ...
'foregroundcolor', [1 0 0], ...
'tag', '1', ...
'callback', @fig)

function fig(varargin)
button = questdlg('2+2=4?', 'Continue Operation', 'Yes', 'No', 'Yes');
if strcmp(button, 'Yes')
set(findobj(0, 'tag', '1'), 'str', 'Правильно');
elseif strcmp(button, 'No')
set(findobj(0, 'tag', '1'), 'str', 'Не верно');
end
```

### Объект «Кнопка с фиксацией»

```
function x(varargin)
uicontrol('style', 'togglebutton', ...
'string', 'run me', ...
'horizontalalignment', 'left', ...
'fontsize', 14, ...
'units', 'normal', ...
'position', [.1 .1 .4 .1], ...
'backgroundcolor', [.8 .8 .8], ...
'foregroundcolor', [1 0 0], ...
'tag', '1', ...
'callback', @fig)
```

```
function fig(varargin)
get(findobj(0, 'tag', '1'), 'value');
if ans == 1
set(findobj(0, 'tag', '1'), 'str', 'once again');
else
set(findobj(0, 'tag', '1'), 'str', 'run me');
end
```

### Объект «Раскрывающийся список»

```
function x(varargin)
uicontrol('style', 'popupmenu', ...
'string', '< 1 >|< 2 >|< 3 >|< 4 >|< 5 >|< 6 >|< 7 >', ...
'horizontalalignment', 'left', ...
'fontsize', 10, ...
'fontweight', 'bold', ...
'units', 'normal', ...
'position', [.1 .1 .1 .1], ...
'backgroundcolor', [.9 .5 .1], ...
'foregroundcolor', [0 0 0])
```

### Объект «Пролистываемый список»

```
function x(varargin)
uicontrol('style', 'listbox', ...
'string', '< 1 >|< 2 >|< 3 >|< 4 >|< 5 >|< 6 >|< 7 >', ...
'horizontalalignment', 'left', ...
'fontsize', 10, ...
'fontweight', 'bold', ...
'units', 'normal', ...
'position', [.1 .2 .15 .1], ...
'backgroundcolor', [.9 .5 .1], ...
'foregroundcolor', [0 0 0])
```

### Объект «Редактируемый текст»

```
function x(varargin)
uicontrol('style', 'edit', ...
'string', 'enter the текст', ...
'units', 'normal', ...
'position', [.1 .1 .2 .15]);
```

### Объект «Статический текст»

```
function x(varargin)
uicontrol('style', 'text', ...
'string', 'не enter the текст', ...
'units', 'normal', ...
'position', [.1 .1 .2 .15]);
```

### Объект «Статический текст» с возможностью редактирования

```
function x(varargin)
text(.1,.1,'run me', 'buttondownfcn', 'set(gca, "editing", "on"));
set(gca, 'visible', 'off');
```

### Объект «Слайдер»

```
function x(varargin)
uicontrol('style', 'slider', ...
'tooltipstring', 'подвигай бегунок', ...
'min', 0, ...
'max', 100, ...
'sliderstep', [.2 .1], ...
'units', 'normal', ...
'position', [.1 .1 .5 .05]);
```

### Объект «Окно программы»

```
function x(varargin)
figure;
```

### Объект «Система координат»

```
function x(varargin)
axes('units', 'pixels', ...
'position', [100 100 400 300], ...
'xcolor', [1 0 0], ...
'ycolor', [0 1 0], ...
'XGrid', 'on', ...
'YGrid', 'on');
```

### Объект «Меню»

```
function x(varargin)
f = uimenu('Label', 'Workspace');
uimenu(f, 'Label', 'New Figure', 'Callback', 'figure');
```

```

uimenu(f,'Label','Save','Callback','save');
uimenu(f,'Label','Quit','Callback','exit',...
'Separator','on','Accelerator','Q');

```

## Объект «Флаг»

```

function x(varargin)
uicontrol('style','checkbox');

```

## Объект «Контекстное меню»

```

function x(varargin)
cmenu = uicontextmenu;
hline = plot(1:10,'UIContextMenu',cmenu);
cb1 = ['set(hline,'LineStyle','--')];
cb2 = ['set(hline,'LineStyle',':')];
cb3 = ['set(hline,'LineStyle','-')];
item1 = uimenu(cmenu,'Label','dashed','Callback',cb1);
item2 = uimenu(cmenu,'Label','dotted','Callback',cb2);
item3 = uimenu(cmenu,'Label','solid','Callback',cb3);

```

Визуально представим графические классы на рис. 3.12.

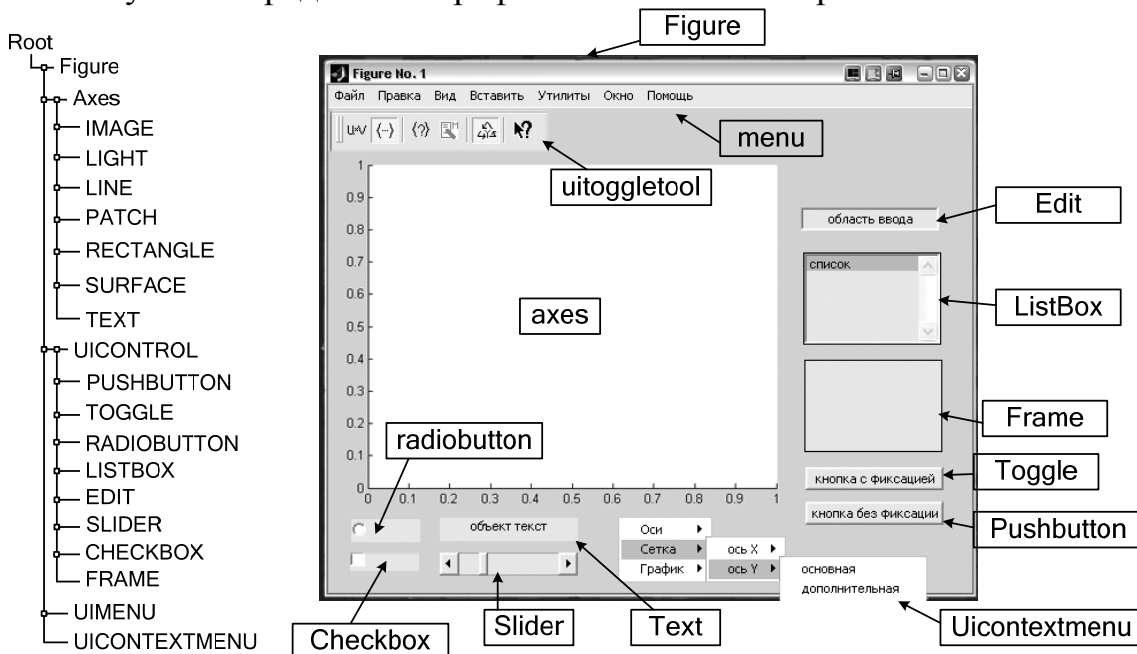


Рис. 3.12. Графические классы Matlab 6.5

Приведём пример использования классов, указанных в табл. 3.1, для подготовки программного модуля конфигуратора параметров для настройки технологической системы.

Создадим новый файл `x.m`, в котором инициализируем функцию.

```

function x(varargin)
% Программа конфигуратор параметров настройки
% технологического оборудования

```

%+++++

Выполним эскизный проект графического интерфейса пользователя, например, так как показано на рис. 3.13.

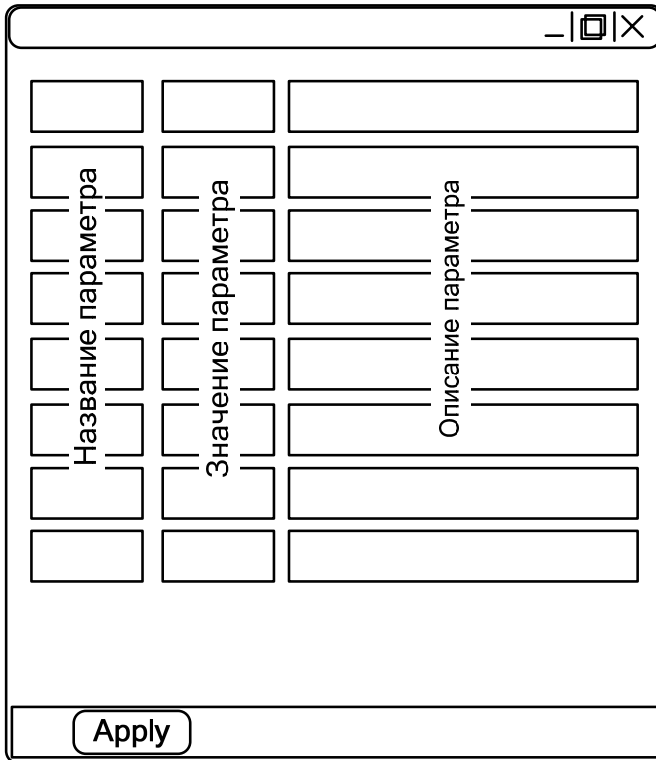


Рис. 3.13. Эскиз графического интерфейса программы конфигурирования параметров технологического оборудования

Инициализируем переменную типа *cell* для хранения параметров создаваемой программы. Переменная типа *cell* очень полезна тем, что может содержать в своём составе переменные различных типов, а так же переменная типа *cell* поддерживается компилятором.

```
config = [{'A='}, {1}, {'Параметр А'}, {'1'};  
{'Б='}, {2}, {'Параметр Б'}, {'2'};  
{'В='}, {3}, {'Параметр В'}, {'3'};  
{'Г='}, {4}, {'Параметр Г'}, {'4'};  
{'Д='}, {5}, {'Параметр Д'}, {'5'};  
{'Е='}, {6}, {'Параметр Е'}, {'6'};  
{'Ж='}, {7}, {'Параметр Ж'}, {'7'}];
```

Назначение переменной *config*, хранить значения первого, второго и третьего столбца графических элементов интерфейса.(рис. 3.13).

Затем, выполним построение окна программы с помощью кода:

```
figure;  
set(1, 'menubar', 'none', ...
```

```
'name', 'Это определённо программа :)', ...  
'numbertitle', 'off', ...  
'windowstyl', 'modal', ...  
'resize', 'off', ...  
'color', [.5 .75 .85], ...  
'uni', 'pix', ...  
'pos', [500 400 230 200]);
```

Далее выделим область кнопки «Apply» (см. рис. 3.13).

```
uicontrol('sty', 'frame', ...  
'uni', 'norm', ...  
'pos', [0 0 1 .1], ...  
'back', [0.77 0.92 0.83], ...  
'fore', [1 1 0]);
```

Далее, построим первый столбец графических элементов. Поскольку все элементы одинаковы по конфигурированию, за исключением параметра string и tag, то воспользуемся для их построения циклом.

```
% Первый столбец графических объектов  
left = .05;  
top = 1;  
hight = .1;  
width = .1;  
spacing = .01;  
  
for i = 1:7  
    uicontrol('uni', 'norm', ...  
            'sty', 'text', ...  
            'str', config{i,1}, ...  
            'fontweight', 'bold', ...  
            'fontsize', 10, ...  
            'back', [.5 .75 .85], ...  
            'pos', [left top-(hight+spacing)*i width hight]);  
end;
```

При выполнении этого действия, значение параметра position пришлось модифицировать.

Подобным способом построим второй и третий столбец графических элементов.

```
% Второй столбец графических объектов  
left = .15;  
  
for i = 1:7  
    uicontrol('uni', 'norm', ...
```

```
'sty', 'edit', ...
'str', config{i,2}, ...
'fontweight', 'bold', ...
'fontsize', 10, ...
'back', [.5 .75 .85], ...
'tag', config{i, 4}, ...
'pos', [left top-(hight+spacing)*i width hight]);
end;
```

```
% Третий столбец графических объектов
left = .3;
width = .5;
```

```
for i = 1:7
uicontrol('uni', 'norm', ...
'sty', 'text', ...
'str', config{i,3}, ...
'fontweight', 'bold', ...
'fontsize', 10, ...
'back', [.5 .75 .85], ...
'pos', [left top-(hight+spacing)*i width hight]);
end;
```

```
% Кнопка расчёта параметров
uicontrol('uni', 'norm', ...
'str', 'apply', ...
'fore', [1 1 0], ...
'back', [.9 .5 .1], ...
'fontweight', 'bold', ...
'pos', [.1 .01, .2 .08], ...
'call', @calc);
```

Последней строкой текста является кнопка получения результата «Apply». Для выполнения действия при нажатии на кнопку укажем ссылку на функцию *calc*, которая тоже находится в файле *x.m*. Текст функции *calc* представлен ниже.

```
function calc(varargin)
% Расчётное ядро программы, которое собирает редактированные данные
%+++++
for i = 1:7
g(i) = str2num(get(findobj(0, 'tag', num2str(i)), 'str'));
end;
% Запишем форматированные данные в файл результата exp.txt
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f\n',g);
fclose(fid)
```



После того как параметры в ячейках второго столбца графических элементов были собраны (по аналогии, в цикле), следует сформировать необходимый формат данных и записать их в файл. Пусть это будет текстовый файл с заранее известным именем `exp.txt`.

После того как код программы полностью написан, можно выполнить компиляцию файла программы, для того, чтобы не использовать Matlab при запуске новой программы.

Для этого воспользуемся командой `mcc -m -B sgl x`. Что соответствует компированию кода М в свободно исполняемый файл `x.exe`.

Более подробно об использовании компилятора в Matlab будет рассказано ниже.

### **3.5. Проектирование сложных (составных) графических элементов в среде Matlab**

При проектировании интерфейса пользователя может не хватать возможностей простейших классов ГПИ Matlab для реализации требуемого графического элемента управления, например, такого как всплывающее меню. Использование ГЭ всплывающее меню очень эффективно, поскольку позволяет сэкономить графическое пространство при построении, например, среду обработки данных, когда видимая область монитора недостаточна для чёткого визуального анализа экспериментальных данных.

Анализируя графический объект всплывающее меню, отметим, что принцип его действия тоже основан на группировке графических элементов по определённому признаку, однако визуализация сгруппированных объектов производится при условии, что указатель графического объекта курсор находится в области активной площадки данного меню или его пиктограммы. Проанализируем алгоритм работы на основании рис. 3.14.

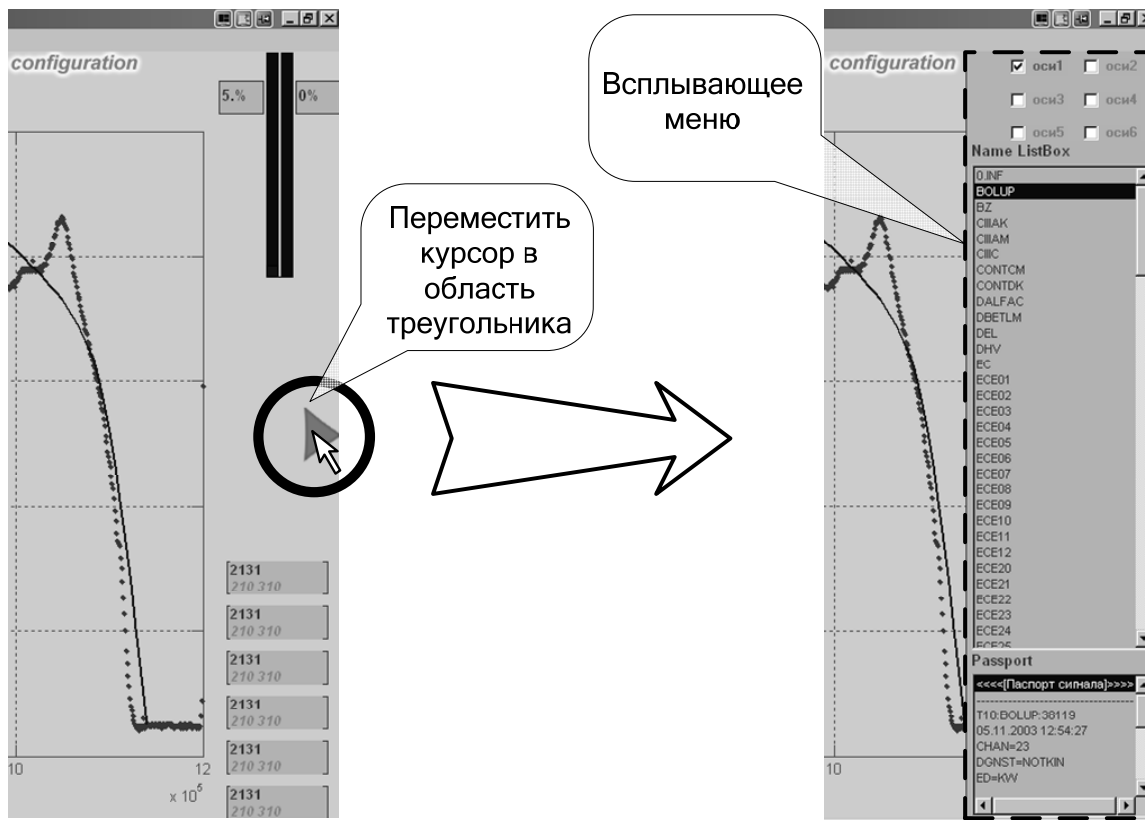


Рис. 3.14. Пример видеозаписи графической формы «Всплывающее меню»

На рис. 3.14 видно, что при наведении курсора мыши на пиктограмму всплывающего меню (зелёный треугольник) в графической области основного окна появляется дополнительное меню, которое содержит новую дополнительную информацию.

### ***Технология группирования графических элементов по признаку***

В основе объектно-ориентированного программирования лежат классы и объекты. Каждый объект в объектно-ориентированном программировании (ООП) имеет свой тип (или класс). Класс представляет собой тип данных, имеющих в своём составе свойства (параметры объекта) и методы, которые можно выполнять над этим объектом или он может выполнять.

Например, в GUI Matlab существует класс объектов *icontrol*, предназначенный для построения объектов управления (кнопки, флажки, списки). Класс *icontrol*, содержит в своём составе объект *pushbutton*, *toggle*, *listbox*, *edit*.

В соответствии с принципами ООП, у каждого объекта класса *icontrol*, существуют свойства и методы влияния на эти свойства.

В частности, для выполнения ГЭ всплывающее меню требуются свойства *tag* и *userdata*. Данные свойства зарезервированы под индивидуальную метку объекта и пользовательское поле произвольных данных соответственно.

На основании поставленного требования, сгруппируем необходимые объекты по свойству *userdata*, задав каждому объекту одинаковую метку в поле *userdata*. А индивидуальная метка *tag*, будет необходима для поиска и управления конкретным объектом.

Для управления видимостью объекта в области фигуры, в которой он задан, воспользуемся свойством объектов, *visible*.

Команды управления свойствами объектов в среде Matlab заданы как: *set* и *get*, соответственно: установить и получить свойство объекта. На основании представленных возможностей Matlab приведу структурную схему технологии управления сгруппированными объектами при реализации сложного(составного) объекта «всплывающее меню» (рис. 3.15).

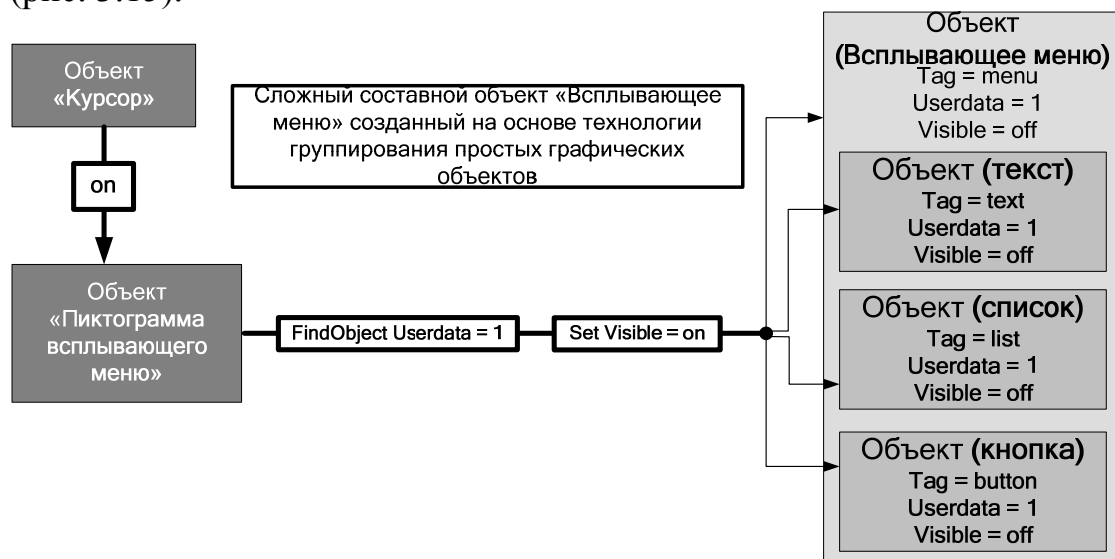


Рис. 3.15. Структурная схема технологии управления сгруппированными объектами и технология работы

Алгоритм управления всплывающим меню следующий:

1. Пользователь позиционирует объект курсор в области пиктограммы объекта всплывающего меню;
2. объект пиктограмма всплывающего меню посылает последовательно команды: найти объекты со свойством *Userdata = 1* (*FindObject Userdata = 1*) и установить свойство видимости объекта в положение включен (*Set Visible = on*);
3. на основании команды объекта пиктограммы всплывающего меню, объект всплывающего меню вместе с составными частями

(объект текст, список, кнопка), активизируется. Для этого свойство *Visible* переводится в положение *on*.

Следует отметить, что для того, чтобы графические объекты при позиционировании их в графической области не перекрывались друг другом, их необходимо инициализировать в порядке видимости для пользователя. Так для построения ГЭ всплывающее меню (см. рис. 3.15), сначала инициализируется ГЭ *frame*, затем ГЭ *checkbox* и *listbox*.

### **3.6. Комбинированный способ проектирования и редактирования GUI в Matlab**

Задача комбинированного способа проектирования, заключается в возможности редактирования кода с ГПИ, написанного на основе технологии непосредственного программирования, при помощи специальных графических редакторов (GUIDE, INSPECT).

Подобные алгоритмы требуются при реализации программы, ГПИ которой требуется редактировать пользователю совершенно не знакомому с программированием. Рассмотрим возможность построения этого процесса при помощи встроенных Matlab редакторов GUIDE и Inspect.

GUIDE – программный компонент, написанный на языке Java с элементами М языка.

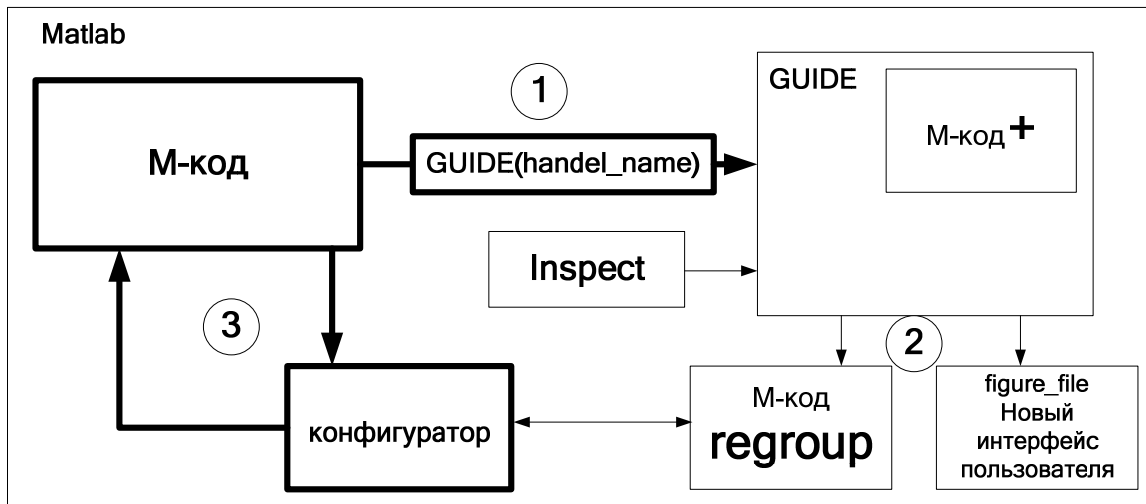
Данный продукт предназначен для визуального проектирования и редактирования GUI, в среде Matlab. Компонент поставляется в составе стандартного дистрибутива Matlab. Запуск компонента производится командой GUIDE в строке команд Matlab или из главного меню Matlab.

Компонент INSPECT – программный компонент, написанный на языке Java с элементами М языка.

Данный продукт предназначен для визуального редактирования свойств графических объектов Matlab и входит в стандартный комплект поставки дистрибутива Matlab. Запуск компонента производится посредством команды inspect из окна команд Matlab или соответствующей пиктограммой из окна GUIDE.

На основании предложенных компонентов рассмотрим специализированную технологию редактирования рабочих кодов и графического интерфейса на основе редактора GUIDE и inspect.

Подробно рассмотрим технологию на основе структурной схемы 3.16.



**Рис. 3.16.** Структурная схема технологии проектирования графических интерфейсов комбинированным способом

На основании данной схемы проанализируем предложенный алгоритм.

Алгоритм редактирования рабочего кода будем проводить в 3 этапа:

1. по команде из ПО вызывается редактор GUIDE, в котором производится редактирование кода графического модуля программы (*М-код +*);
2. после того, как редактирование завершено, пользователь экспортирует переработанный проект (*М-код regroup*) в директорию проекта посредством команды главного меню **file\export**;
3. из окна программы (ГПИ), пользователь запускает программу конфигуратора обновлённого М кода, которая заменяет текущий М-код ПО, новым М-кодом с обновлённым интерфейсом.

В последнем случае, необходимо просто скопировать программный код с обновлённым интерфейсом в директорию с программой и сохранить под именем исходного программного кода. Задание параметров ГЭ нового интерфейса программы производится инспектором свойств *Inspect*.

### 3.7. Использование компилятора в среде Matlab

В этом разделе рассмотрим вопросы применимости конечного кода компилятора для АСУ ТП, а также вопросы действительной необходимости использования компилятора.

Мы знаем, что выходными кодами компиляторов в Matlab могут быть:

- C-code;
- C++-code;
- P-code;
- Exe-code;
- MEX-code;
- S-code.

Вполне очевидно, что необходимость получения С-кода из написанного М-кода очень важна, так как код С используется повсеместно. Поскольку одним из языков основателей Matlab является С и все основополагающие функции Matlab написаны на С, то компилятору не составит труда собрать все С-коды, которые участвуют в выполнении программы написанной на языке М. Далее вы можете использовать полученные С или С++ коды в собственных нуждах.

Код Р является псевдо кодом. Мы знаем, что разработка технологии JIT компанией SUN Microsystems последовала после потребности сократить время опознавания кода при компилировании. Для транслирования в машинный предоставляется код в удобочитаемом виде, в байт-коде. Псевдо код Р – нечто приближенное к байт коду, однако им не являющееся. Кроме того, что Р код, по теории, должен выполняться интерпретатором Matlab быстрее, чем код М, код Р скрыт для понимания его текста человеком. Таким образом код Р представлен как закодированный, быстро выполняемый код, который не может выполняться без интерпретатора.

При разработке собственного программного модуля иногда важно получить свободно исполняемый код (stand-alone). Этот код должен выполняться на любом совместимом ПК без использования среды разработки. При этом код не поддерживает перепрограммирование, однако он должен выполняться быстрее, чем интерпретируемый. На практике же сколько либо значительных задержек в выполнении интерпретируемого кода, в отличии от исполнения компилируемого, не отмечено.

MEX-коды в Matlab позиционируются как коды расширений функций для Matlab. Они выполнены в виде внешних автономных кодов \*.dll и интерпретируются Matlab как собственные функции. Использовать эти модули, применительно к готовым программным комплексам, разработанным для АСУ ТП, можно в качестве функциональных библиотек, поставляемых отдельно к уже готовому программному

обеспечению, выполненному в виде интерпретируемого кода или stand-alone.

Что такое MEX? MEX – это целевой код, который получается при транслировании входного для Matlab Compiler (mcc) кода в целевой код. Дословно это звучит так:

**«Код, воспроизводимый Matlab Compiler, независимо от конечного целевого типа называется MEX, выполняемым или библиотекой».**

Использование MEX-функций позволяет решить следующие важные прикладные задачи: создавать в Matlab приложения, выполняемые в дальнейшем в операционной системе без Matlab; обращаться из Matlab к функциям существующих DLL-библиотек и/или исполняемых программ; скрывать от пользователя внутреннюю структуру разработанной программы; повышать скорость выполнения соответствующих M-программ путем разработки MEX-функций, например на языке C, и выполнения замены существующих M-функций.

В пакет Matlab, как вам известно, так же входит Simulink. Simulink – среда визуального проектирования, ориентированная на блочное моделирование систем и комплексов. Большие возможности по расширению функций Simulink позволяют выполнять множество задач по проектированию как электронных, так и технических схем.

В стандартный набор библиотек Simulink кроме всех прочих входят User-Defined Function, которые позволяют добавлять в состав разрабатываемой модели функции написанные на языке M в виде собственных S-functions Simulink или функций написанных в соответствии с синтаксисом языка M. Кроме всего прочего, S-функции, которые могут быть включены в состав модели могут быть как скомпилированными в S файл, так и написанными на языке M в соответствии с архитектурой MEX. Использование блочно-ориентированной технологии визуального проектирования очень важно с точки зрения САПР проектирования для АСУ ТП, так как оно позволяет не вдаваясь в тонкости более низкоуровневых языков программирования описать необходимый алгоритм функционирования технологического оборудования. Примерами такого проектирования могут стать языки FBD и PL, однако следует понимать, что построение сложных математических алгоритмов на этих языках не возможно, в отличие от языка M.

В качестве результата обсуждений приведём пример программы для унификации доступа к функциям компилятора Matlab.

Эскизный проект графического интерфейса программы приведён на рис. 3.17.

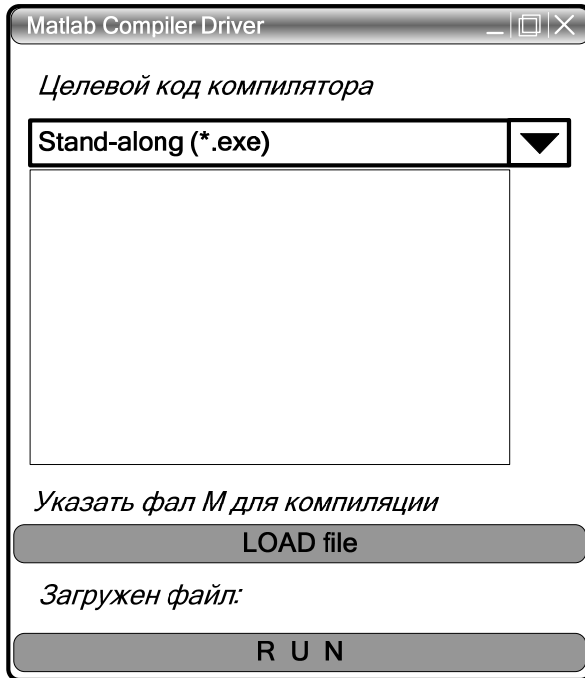


Рис. 3.17. Эскиз видеогамы графического пользовательского интерфейса программы MCD

Запишем в окне редактора код программы инициализации глобальной переменной:

```
function mcd(varargin)
% Программа Matlab Compiler Driver.
% В функции программы входит транслирование кода M в:
% * C
% * C++
%
% Компилирование в:
% * exe code
% * MEX code
% * S code
%=====
% Объявление переменной
global config
```

Затем опишем графический интерфейс пользователя программы.

```
% окно программы и основные кнопки
figure;

% окно программы
set(1, 'windowstyle', 'modal', 'numbertitle', 'off', 'name', 'M C D', ...
'uni', 'pix', 'resize', 'off', 'pos', [350 350 200 190], 'color', [1 1 1]);
```



```

% текст над кнопкой
uicontrol('sty', 'text', 'str', 'Целевой код компилятора', 'back', [1 1 1], ...
'fore', [.6 .6 .6], 'uni', 'norm', 'pos', [0.0005 0.900 0.999 0.1], 'fontweight', 'bold');

% меню режима компиляции
uicontrol('style', 'popupmenu', ...
'string', 'c-code (*.c)|c++ code (*.cpp)|mex-function (*.dll)|simulink s-function|p-
code (*.p)|stand alone c-code (*.exe)', ...
'units', 'norm', 'position', [.001 .800 .999 .1], 'back', [1 1 1], 'fontweight', 'bold', ...
'tool', 'Выберите тип целевого кода', 'callback', @editmenu);

% текст над кнопкой
uicontrol('sty', 'text', 'str', 'Указать файл M для компиляции', 'back', [1 1 1], ...
'fore', [.6 .6 .6], 'uni', 'norm', 'pos', [0.0005 0.300 0.999 0.1], 'fontweight', 'bold');

% кнопка загрузки
uicontrol('str', 'LOAD file', 'uni', 'norm', 'fontweight', 'bold', 'tool', 'Выберите файл
для компиляции', ...
'pos', [.001 .200 .999 .1], 'back', [.04 .54 .84], 'fore', [0.5 1 0.5], 'call', @op);

% текст над кнопкой
uicontrol('sty', 'text', 'str', 'Загружен файл: ', 'back', [1 1 1], 'horizontalalignment',
'left', ...
'fore', [.6 .6 .6], 'uni', 'norm', 'pos', [0.0005 0.100 0.999 0.1], 'fontweight', 'bold',
'tag', 's');

% кнопка компиляции
uicontrol('style', 'push', 'str', 'R U N', 'units', 'pixels', 'fontweight', 'bold', ...
'tag', '1', 'enable', 'off', 'back', [.04 .54 .84], 'fore', [0.5 1 0.5], 'tool', 'Запуск процесс
са компиличирования', ...
'uni', 'norm', 'pos', [.001 .001 .999 .1], 'call', @run);

```

Запишем функцию загрузки файла для компиляции.

```

function op(varargin)
% Функция выполняемая при загрузке файла для компиляции

global config;

% выбор файла при нажатии кнопки загрузки файла
[config.file, config.path] = uigetfile({'*.m', 'MATLAB M-Files (*.m)'},
'Load file')

% проверка выполнения действия загрузки файла
if config.path ~= 0
set(findobj(0, 'tag', '1'), 'en', 'on');

% вывод имени и пути файла в окне программы

```

```

set(findobj(0, 'tag', 's'), 'str', ['Загружен файл: ', config.file], 'tool',
[config.path, config.file]);
else
% set(findobj(0, 'tag', '1'), 'en', 'off');
end

```

Запишем функцию, отвечающую за выбор типа компиляции, т.е. функцию, выполняемую при инициализации объекта список.

```

function editmenu(varargin)
% функция производит селекцию выбора типа компиляции
global config;
switch get(gcf, 'value')
case 1
config.compil.x='-m';
case 2
config.compil.x='-p';
case 3
config.compil.x='-x';
case 4
config.compil.x='-S';
case 5
config.compil.x='-B pcode';
case 6
config.compil.x='-B sgl'
end

```

Запишем функцию, отвечающую за формирование команды компилятору, после нажатия кнопки RUN.

```

function run(varargin)
% функция конструктор команды компилятору
global config
% Команда компилятору
eval(['mcc ', config.compil.x, ' ', config.path, config.file]);
% Открыть директорию, в которой расположен результат
компиляции
eval(['!explorer ', config.path]);

```

Текст программы, должен находиться в одном М файле. После программирования М файла, программу можно компилировать для внешнего исполнения.

### **3.8. Проблемы, возникающие при компиляции кода М**

Использование компилятора в среде Matlab не обязательно, но полезно. Однако следует упомянуть некоторые особенности использования компилятора в среде Matlab 6.5.

Если перед вами стоит задача выполнить компиляцию конечного кода, то старайтесь не использовать переменные типа *structure*, потому что при компиляции вы получите ошибку. На аналог переменной *structure*, *cell* компилятор не ругается. Проблема использования компилятором переменной типа *structure* пока не исследована.

### **3.9. Внешнее исполнение программ**

При проектировании собственных приложений иногда требуется выполнять запросы к внешним модулям. Например, запускать на выполнение программы и утилиты либо исполнять готовые скрипты на других операционных системах (ОС). Поэтому в этой главе мы будем рассматривать вопросы, связанные с внешним исполнением программ из Matlab.

#### **3.9.1. Команды внешнего запуска приложений**

В интерпретаторе Matlab заложены команды для различных ОС. Исполнение команд сторонних ОС может осуществляться непосредственно в тексте программы написанной на М языке, а результат выводиться в переменные Matlab. Подобные возможности значительно расширяют функциональность языка и способствуют его внедрению и развитию разработчиками.

В качестве поддержки в Matlab выбраны следующие типы ОС:

- DOS;
- Windows;
- Unix.

А так же языки программирования:

- Perl;
- Java.

### 3.9.2. Режим выполнения команд Windows

Для работы с командами операционной системы в Matlab можно пользоваться специальной командой `system`. Эта команда применяется для интеллектуального исполнения внешних команд из Matlab. Формат команды следующий: `system('command_name')`. Для примера попробуем выполнить команду `ipconfig` в командной строке Matlab. Как результат – ошибка! Это правильно, потому что Matlab не знает такой команды. Теперь выполним команду при помощи внешнего интерфейса – команды `system`. `system('ipconfig')`. На самом деле, исполнить команду операционной системы можно при помощи знака «!». В этом случае формат команды будет выглядеть следующим образом: `!ipconfig`. Очевидно, что результат идентичен, в чём же разница? Важной особенностью использования последнего метода в том, что с его помощью можно исполнить команду операционной системы и вывести результат в среде, команду которой вы выполняете, в нашем случае Windows. Для реализации этого случая необходимо указать символ «&» в конце команды в следующем формате: `!ipconfig&`.

В чём же преимущество использования команды `system`? Вполне очевидно, что при использовании обеих команд визуально, программист не заметит разницы, однако для компьютера это неочевидно. Для программы или среды разработки (в случае Matlab) выполнение программы очевидно тогда, когда результат выводится во внутреннюю переменную среды. В этом случае, возможно использовать только команду `system`, поскольку лишь она позволяет программе видеть результат. Для реализации этого случая исполним в командной строке Matlab следующий скрипт: `[index, result] = system('ipconfig');`. Следует заметить, что в области просмотра переменных Matlab, появились две переменные: первая является индикатором состояния выполнения команды, вторая содержит результат выполнения команды. Переменная результата – `result` имеет тип `char` и, следовательно, вы можете обрабатывать результат исполнения команды `ipconfig` непосредственно в Matlab.

### 3.9.3. Режим выполнения команд Unix

Поскольку в данный момент, мы работаем в ОС Windows, протестировать режим исполнения команд Unix мы не сможем. В этом можно убедиться, исполнив следующий скрипт:

```
[s,w] = unix('why')
s = 1
w =
```

*why: Command not found.*

Здесь переменная *s*, говорит о том, что команда не была выполнена по причине ошибки, в переменной *w* получен текст ошибки.

### 3.9.4. Режим эмуляции команд DOS

Для того, чтобы эмулировать команды DOS, как и в предыдущих случаях нужно использовать специализированную команду DOS. Формат команды следующий: *dos('edit')*. Исполнив данный скрипт, вы сможете видеть, что открылся, привычный для dos редактор. Вы можете его использовать для написания *m* файла. Однако для режима эмуляции DOS предусмотрена ещё одна хитрость. В Matlab возможно открыть сеанс DOS непосредственно в командной строке Matlab. Для этой цели существует команда *dos command*. Вы можете пользоваться командами DOS, а затем, по окончании, командой *exit* завершить сеанс.

## 3.10. Работа с таймером в Matlab

При проектировании систем управления, очень важно запускать и останавливать отдельно взятые программы или функции в определённое технологией процесса или регламентом работ время. Для этого используется таймер. В случае с Matlab, это объект таймер. Как и ГЭ объект таймера имеет свойства и поддерживает конфигурирование своих свойств командами *set* и *get*.

Рассмотрим конструкцию объекта таймера. Синтаксис записи таймера следующий:  $T = \text{timer}(\text{PropertyName1}, \text{PropertyValue1}, \text{PropertyName2}, \text{PropertyValue2}, \dots)$ .

Набор свойств объекта таймера представлен в табл. 3.2.

Табл. 3.2. Некоторые свойства объекта таймер

| <b>Название свойства</b> | <b>Описание свойства</b>                               | <b>Принимаемые значения</b> |
|--------------------------|--|-----------------------------|
| Name                     | Название таймера                                       | string                      |
| Period                   | Задержка в секундах между выполнением TimerFcn         | double > 0.001 s            |
| Running                  | Индикатор текущего состояния таймера                   | 'off', 'on'                 |
| StartDelay               | Задержка в секундах между стартом таймера и первым вы- | double >= 0                 |

|               |  |  |
|---------------|--|--|
|               | полнением функции TimerFcn   |  |
| StartFcn      | Функция, которую выполняет таймер сразу после старта   | String, Function handle, cell  |
| StopFcn       | Функция, которую выполняет таймер при остановке  | String, Function handle, cell  |
| ErrorFcn      | Функция, которую выполняет таймер при возникновении ошибки. Данная функция выполняется до StopFcn. | String, Function handle, cell  |
| Tag           | Метка пользователя   | string   |
| TimerFcn      | Функция, выполняемая таймером  | String, Function handle, cell  |
| ExecutionMode | Режим распределения событий таймером   | Принимаемые значения:<br>'singleShot'<br>'fixedSpacing'<br>'fixedDelay'<br>'fixedRate' |

Т.о., для того, чтобы получить объект таймера необходимо задать конструкцию таймера с указанием свойств запуска и выполнения таймера.

Например: `t = timer('TimerFcn', @r, 'StartDelay', .1, 'stopfcn', @f2, 'tag', 'timer');`

Где t – переменная, содержащая handle таймера.

При этом таймер после запуска выполнит задержку 0.1 сек, затем выполнит функцию @r, затем функцию @f2. Найти таймер можно по свойству tag, в котором прописана метка 'timer'.

Запуск таймера осуществляем функцией *start(t)*. Досрочный останов таймера осуществляется функцией *stop(t)*. Чтобы найти все объекты таймера, которые были созданы ранее, необходимо использовать команду *timerfind*. Удалить конструкцию таймера можно при помощи команды *delete*. Установка свойств таймера производится командой *set*, взять свойство таймера, например, running, можно при помощи команды *get*.

Из таблицы 3.2 очевидно использование объекта таймера в нескольких режимах (см. параметр Execution Mode). Режим singleShot означает выполнение таймера единожды, остальные режимы, - многократно.

Для визуального представления всех режимов рассмотрим временные диаграммы (рис. 3.18, 3.19).

## Single Shot mode

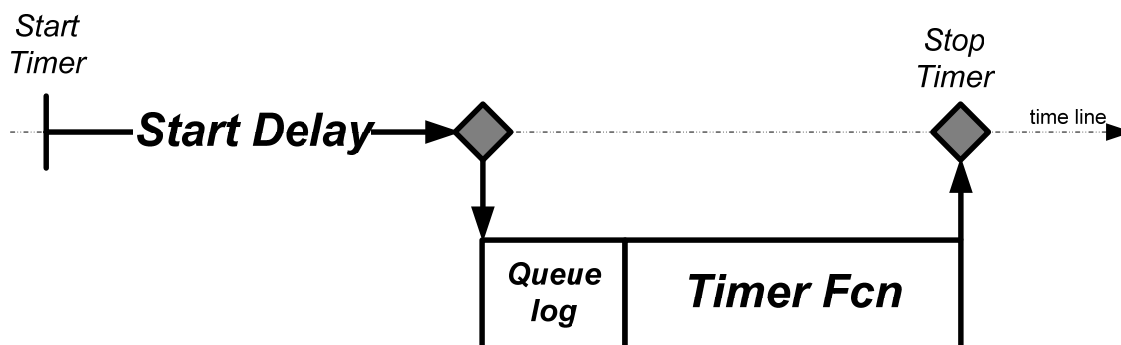


Рис. 3.18. Режим выполнения таймера singleShot (временная диаграмма)

В этом режиме объект таймера выполняется один раз, по истечении задержки start delay, выполняется функция timer fcn, затем таймер останавливается, но не удаляется, а находится в оперативной памяти ПК. В этом состоянии объект таймера можно запустить ещё раз командой start(timer).

Возможен вариант многократного выполнения таймера, для этого предусмотрены 3 режима работы таймера. Рассмотрим их на временной диаграмме 3.19.

## Multiple times

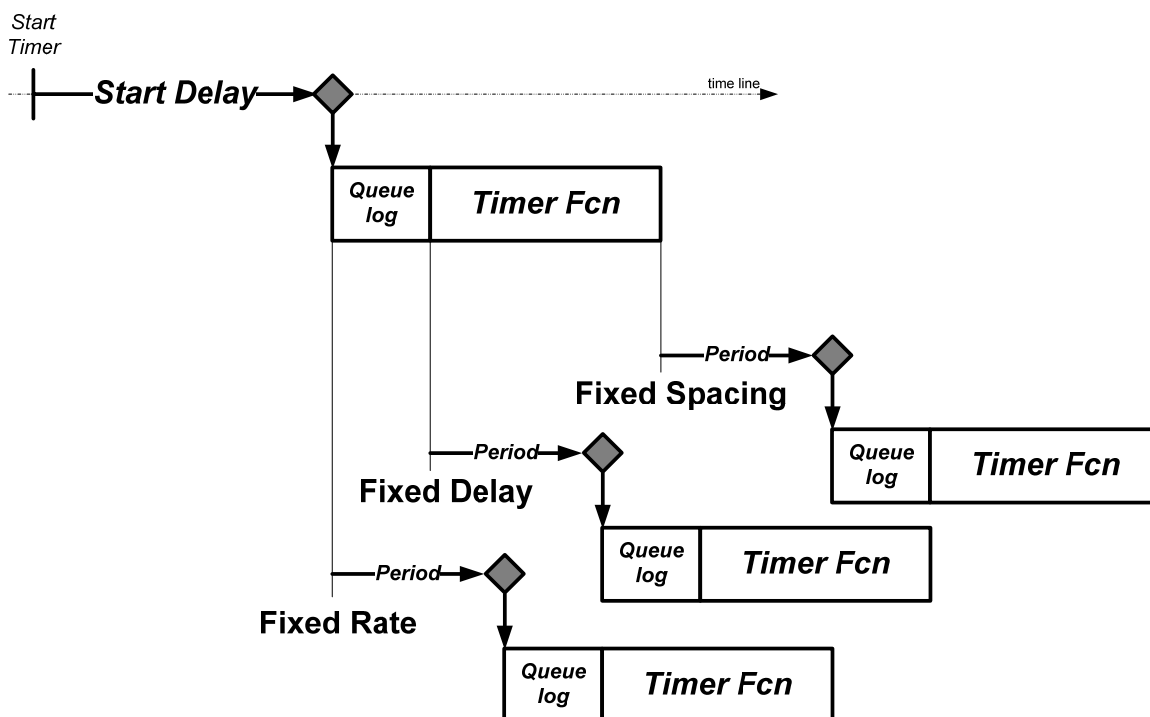


Рис. 3.19. Режимы многократного выполнения объекта таймера (временная диаграмма)

В этом случае возможны три исхода: с точной привязкой к временным интервалам (новый отсчёт начинается сразу после окончания предыдущего), с фиксированной задержкой (после чего функция таймера выполняется параллельно счёту нового периода) и с точным отслеживанием интервалов с учётом выполнения функции таймера.

Определение временных координат может быть произведено за счёт системного таймера посредством команд *tic* [time input clock] и *toc* [time output clock]. Представим общую конструкцию кода для определения времени выполнения команд.

```
% скрипт расчёта времени выполнения программы  
tic;  
% тело программы (ядро)  
v = toc;  
s = [int2str(floor(v/3600)), ':',int2str(floor(v/60)), ':', int2str(floor(mod(v,60)))];
```

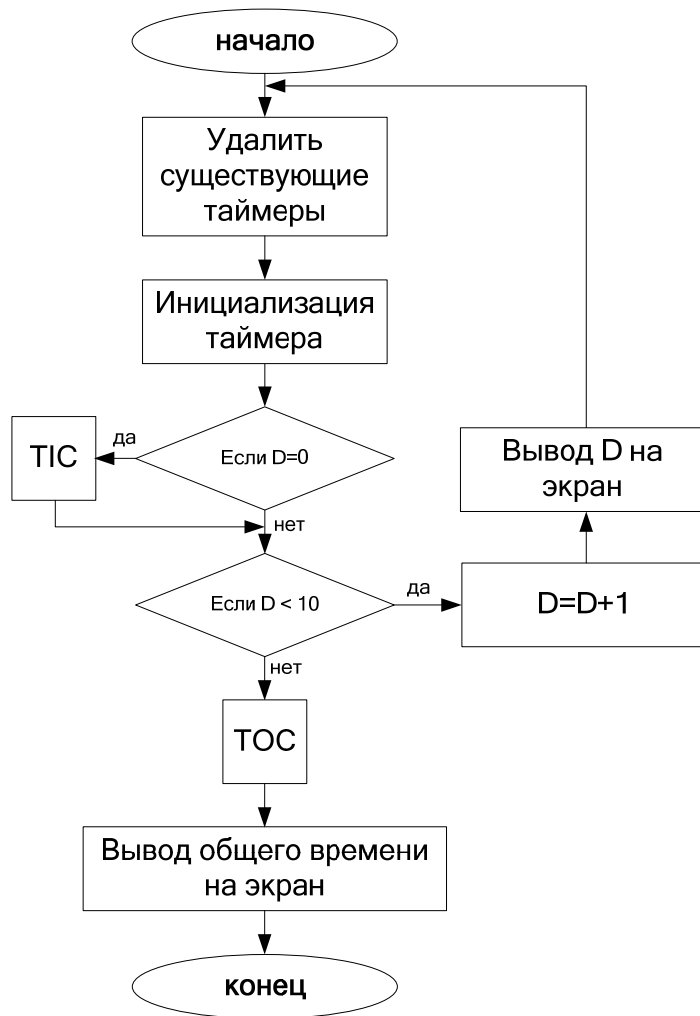
В переменной *s* содержится строка-отчёт в формате `char`.

### **3.11. Тестирование возможностей Matlab по точности выполнения задач по таймеру**

Для реализации алгоритмов управления требуется, чтобы существовала возможность фонового выполнения задач. В случае Matlab этот процесс можно реализовать следующим образом: создать скрипт или функцию, которую необходимо выполнить; создать объект таймера и включить в функцию выполнения по таймеру созданную ранее функцию или скрипт.

Тестирование, точности синхронизации объекта *timer*, будем осуществлять системным таймером, в ОС Windows XP, в среде Matlab по схеме 3.20.





*Рис. 3.20. Алгоритм тестирования точности синхронизации объекта timer Matlab системным таймером ОС Windows*

В качестве нагрузки для программы использовались:

- операция создания нового объекта *timer* на каждом цикле эксперимента;
- операция удаления старого объекта *timer* на каждом цикле эксперимента;
- вывод количества опытов на экран.

В результате тестирования возможностей Matlab по синхронизации с системным таймером в ОС Windows XP, при выполнении простейших операций, было установлено, что точность привязки к временным интервалам 100 мс по заднему фронту составила не более 18 мс (6 мс – 18 мс) на цикл. Опыт состоял из 10 циклов и проводился 10

раз. Разница результатов между опытами  $\pm 6$  мс. Режим тайминга – «singleShot».

Очевидно, что при использовании режима работы таймера fixed Rate, результат будет много более точным, однако в первом случае условия жестче. Следует так же отметить, что при в режиме тестирования мы использовали задачи создания нового таймера а так же удаление старого, что отнимало большое количество времени, а в режиме использования заранее подготовленных объектов таймеров для каждой задачи эти операции будут производиться только один раз. Так же при тестировании использовалась операция вывода на экран, которая является очень тяжёлой для быстрого выполнения.

### **3.12. Визуальное, блочно-ориентированное проектирование в Simulink**

Как программное средство, Simulink – типичный представитель визуально-ориентированных языков программирования. Используя графические технологии проектирования, на всём этапе проектирования, пользователь практически не имеет дело с обычным программированием в командной строке. Стоит отметить, что подобные методы программирования хороши, только на начальных этапах проектирования. При проектировании автоматизированных систем, наиболее эффективно использовать методы непосредственного программирования.

Simulink – это отдельный пакет, который работает под управлением среды разработки Matlab. Однако, как и Matlab, Simulink поддерживает интеграцию с другими системами, в том числе и с Matlab. Поддержка Simulink дополнительных пакетов расширения, даёт практически безграничные возможности проектирования. Создание собственных блоков Simulink, реализующих специализированные функции, так же не представляет труда.

#### **3.12.1. Интеграция пакета Simulink с системой Matlab**

Углубляться в подробное описание операций по работе с Simulink мы не будем, поскольку этот документ предназначен для людей, уже имеющих базовые знания в области работы с Simulink. Однако кратко опишем интерфейс среды.

После инсталляции пакета Simulink (отдельно от пакета Matlab или нет), Simulink автоматически интегрируется в пакет Matlab. Это

проявляется появлением соответствующего значка в панели инструментов Matlab, рядом со знаком « ? ».

Интерфейс среды Simulink показан на рисунке 3.21.

Видеограма, представленная на рисунке 3.21, изображена в составе трёх основных окон Simulink:

- Область библиотеки графических блоков проектирования (слева);
- Рабочая область проектирования (справа, верх);
- Графическая область представления результатов (справа, низ).

Каждая библиотека, в свою очередь, содержит в своём составе специализированные компоненты:

- — — Библиотеки компонентов Simulink;
- • • • • Блоки выбранной библиотеки;
- — — Описание блока. (рис. 3.22)

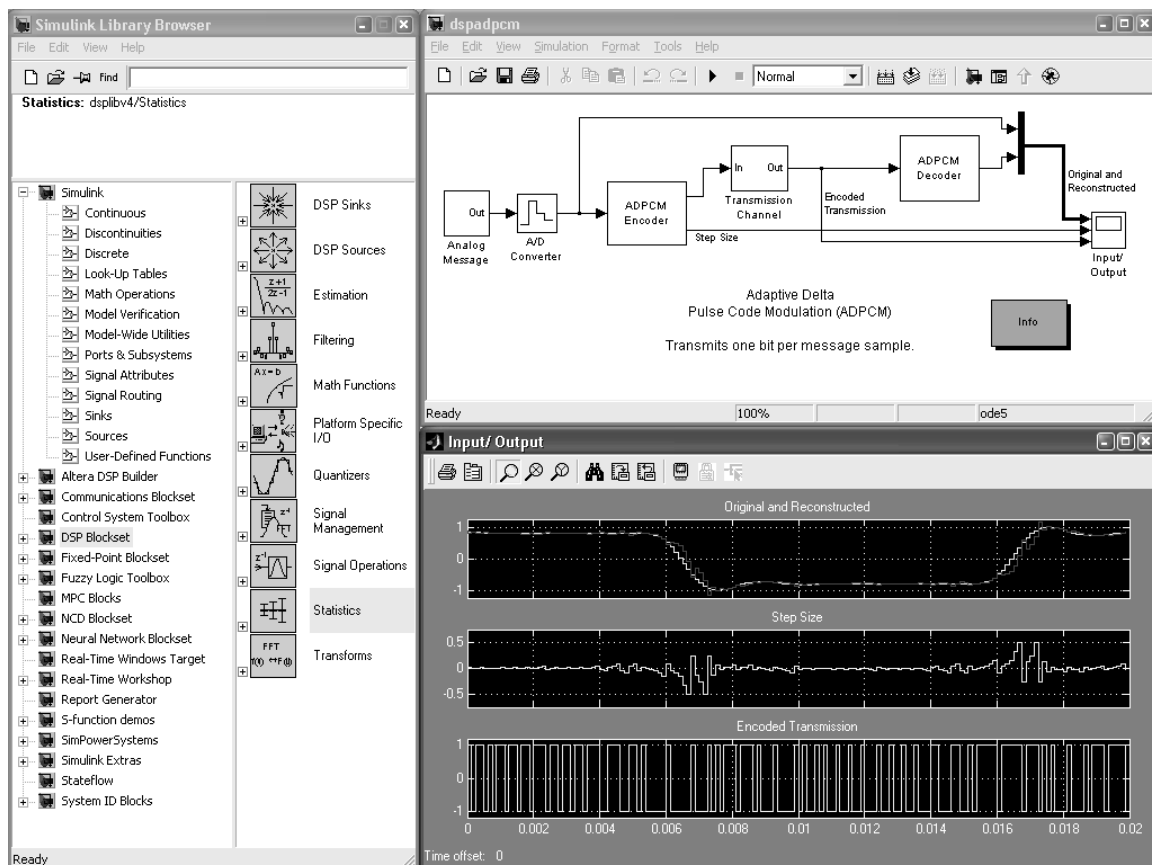


Рис. 3.21. Видеограма графического пользовательского интерфейса визуального проектирования пакета Simulink

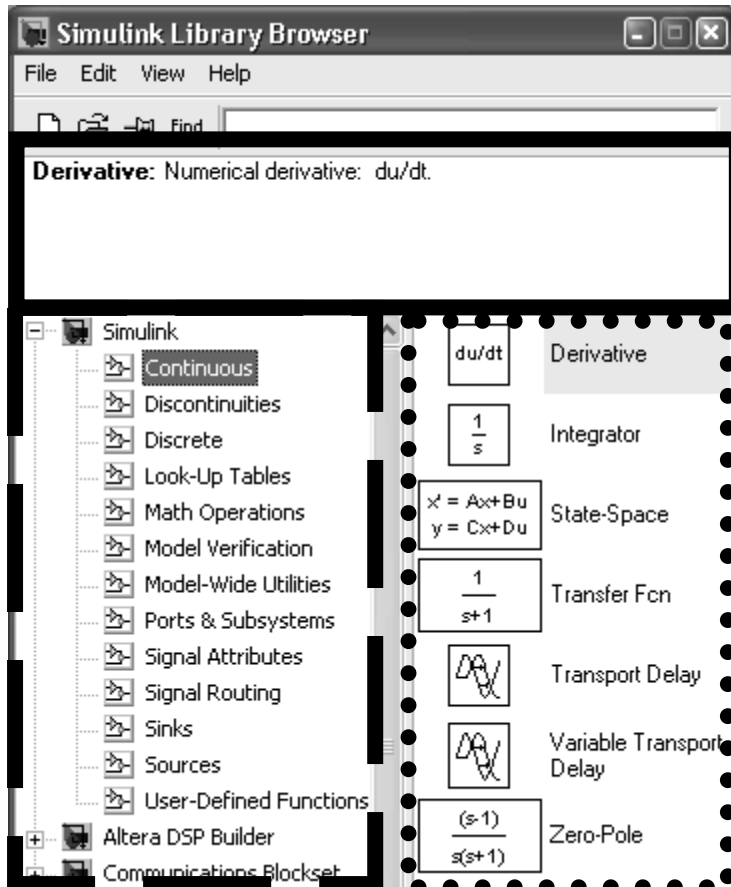


Рис. 3.22. Видеозапись просмотра библиотек компонентов Simulink

Используя блоки библиотек S-functions вы будете строить свои модели.

### 3.12.2. Технологии проектирования в среде Simulink

Ранее, мы уже упоминали, что Matlab и Simulink – расширяемые системы. Это значит, что вы можете использовать собственные функции при проектировании сложных алгоритмов. Однако большинство из необходимых блоков уже предусмотрено в библиотеках Simulink.

В большинстве случаев, проектирование сводится к визуальному. Это значит, что вы должны из одной из библиотек, перетащить необходимый компонент в рабочую область. Затем сгруппировать все компоненты и соединить их необходимым образом в рабочей области (рис. 3.23).

#### ШАГ ПЕРВЫЙ:

- Откройте Simulink;
- Откройте обозреватель компонентов;
- Откройте новое окно рабочего проекта.

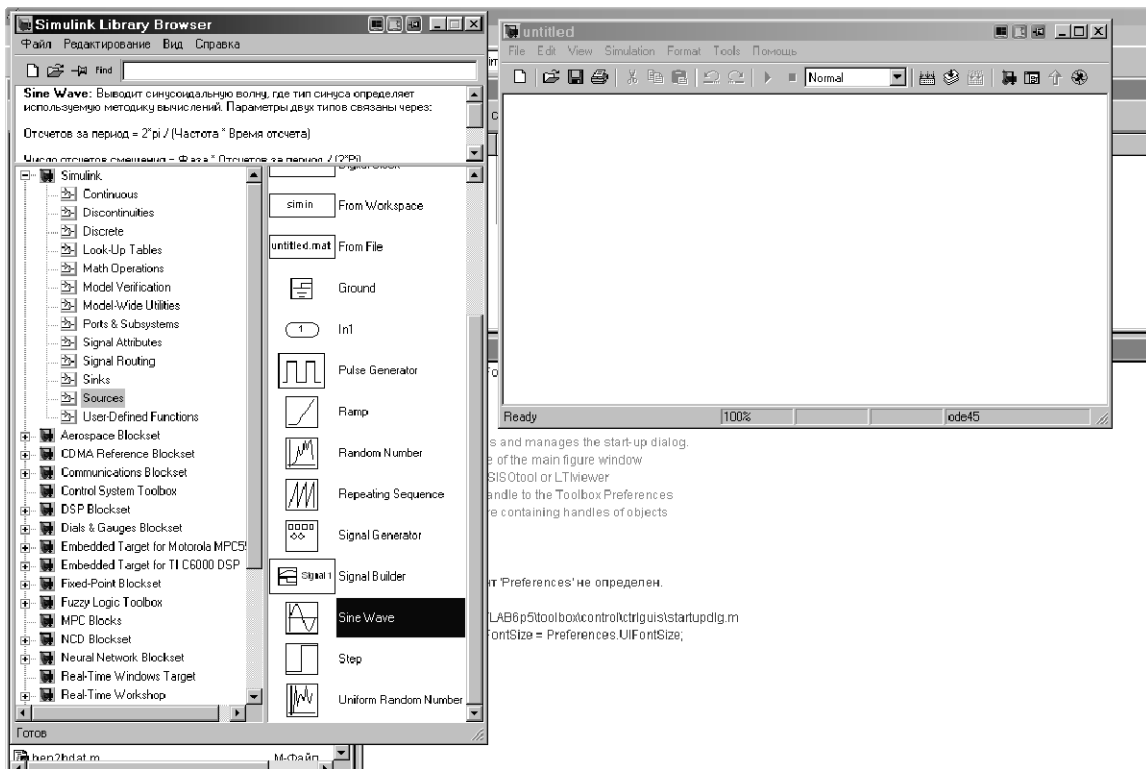


Рис. 3.23. Требуемый для визуального проектирования состав окон Simulink

Затем, необходимо создать саму модель, путём переноса в рабочую область необходимых компонентов.

### ШАГ ВТОРОЙ:

- перетащите из библиотек компоненты необходимые компоненты;
- соедините входы и выходы компонентов линиями (рис. 3.24).

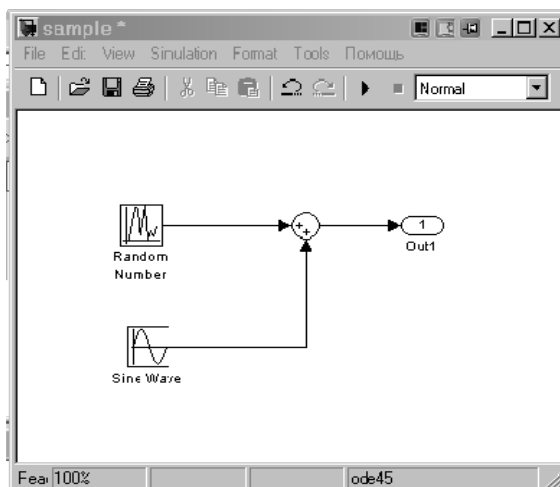


Рис. 3.24. Готовый проект модели

### ШАГ ТРЕТИЙ:

Запустите модель на выполнение.

Подобная конструкция алгоритма работы с Simulink хороша лишь на первых этапах проектирования, когда производится отладка модели. Для того, чтобы получить управляющие сигналы от полученной модели, необходимо уметь запускать модель программно и с необходимыми входными параметрами.

Для того, чтобы получить результаты выполнения модели Simulink в Matlab, например, в рабочую область, необходимо использовать специализированные блоки Simulink. Например, to Workspace. Для запуска модели программно, необходимо использовать команду `sim`. Так, например, для модели описанной на рисунке 3.20, необходимо выполнить команду `[T,X,Y]=SIM('SAMPLE')`. После выполнения команды в командной строке Matlab, в рабочей директории Matlab должны появиться переменные времени `t` и `y` – амплитуды сигнала. Заметьте, что рабочие области Simulink при работе с моделью мы не использовали! При выполнении команды в теле функции результаты выполнения команды `sum` будут возвращены в область оперативной памяти функции.

Возникает вопрос, как быть, если требуется внести небольшие изменения в саму модель или построить процесс проектирования модели автоматически?

Язык программирования Matlab позволяет выполнить добавление или удаление блоков Simulink в области проектирования. Например, команда `add_block('src', 'dest', 'parameter1', value1, ...)`, производит добавление блока в рабочую область Simulink. Так же существуют команды управления блоками:

- `delete_block('___')` – удалить блок из рабочей области Simulink;
- `set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)` – установить параметры для отдельного блока в рабочей области Simulink;
- `get_param(handle, 'parameter')` – взять свойства отдельного блока в рабочей области Simulink;
- `find_system(sys, 'c1', cv1, 'c2', cv2, ... 'p1', v1, 'p2', v2, ...)` – найти систему, блок, линию в рабочей области Simulink;
- `h = add_line('sys', 'oport', 'iport', 'autorouting', 'on')` – добавить линию в рабочую область Simulink;
- `delete_line('sys', 'oport', 'iport')` – удалить линию из рабочей области Simulink;
- `simplot` - построить моделируемые данные в специализированном окне.

Совокупность этих команд может построить процесс проектирования модели в среде Simulink программно, без участия оператора.

### 3.12.3. Пользовательские функции в Simulink

При проектировании в Simulink у опытного инженера возникнет вопрос, а как же сделать функции, которых нет в библиотеках? На самом деле в Simulink в группе Simulink существует библиотека User-Defined Functions (UDF), т.е. функции определяемые пользователем. Что это значит?

Библиотека UDF в версии Matlab 6.5 Simulink 5.0 состоит из следующих блоков:

- Fcn – основной блок использующий переменную  $u$  в качестве аргумента. Он позволяет строить собственные численные формулы в виде:  $\sin(u)/u+5$ ;
- Matlab Fcn – блок использующий в собственном синтаксисе команд, команды Matlab и функции Matlab. Например:  $\text{drim}(u(1), u(2))$  – в процессе выполнения модели задействован файл  $\text{drim.m}$  у которого есть две входные переменные типа `double array` и мы используем первые их значения.
- S-Function – этот блок используется для добавления в состав модели Simulink новых пользовательских S функций Simulink. Создание этих функций из C или FORTRAN кода производится при помощи компилятора, как мы уже говорили. Однако для M существует следующая особенность, вы можете использовать в этом блоке M функцию, однако структура кода должна быть требуемого формата. (см. пункт 3.12.4).
- S-функция Builder – для начинающих пользователей в составе библиотеки UDF существует S-блок подготовки S-функций с помощью специального редактора, использование которого позволит вам за 7 шагов построить необходимую функцию.

Описанные блоки являются универсальными, позволяют быстро добавить в вашу Simulink модель необходимые функции, используя весь математический аппарат Matlab. Графически блоки в рабочей области Simulink представлены на рис. 3.25.

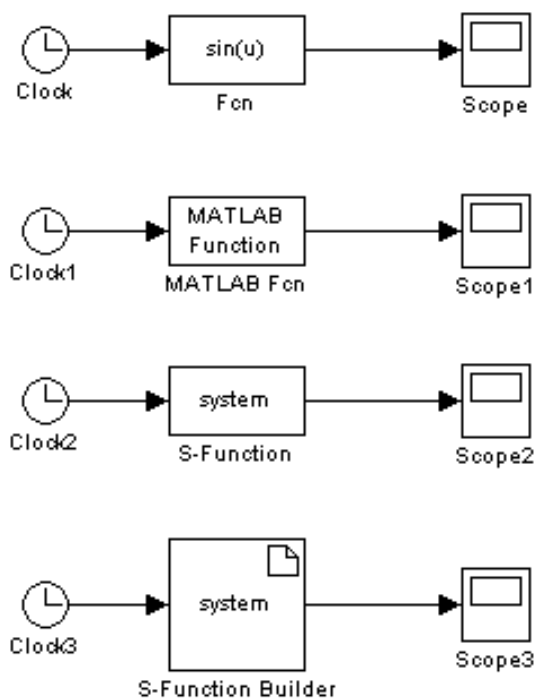


Рис. 3.25. Блоки User-Defined-Function Simulink

### 3.12.4. S функция в виде M кода

Пакеты S-функций – это специальные функции, которые применяются средой Simulink для работы. Однако среда Simulink содержит в своём составе специализированные блоки в библиотеке *Simulink/user defined function*. Мы говорили, что S-function в Matlab, могут быть написаны на M, FORTRAN, C, C++, Ada кодах. В качестве S-function можно использовать так же MEX-функцию. Компилирование MEX функции может быть произведено с помощью любого поддерживаемого Matlab компилятора командой *mcc -S funname*.

Однако для создания S-функции из M-функции можно обойтись без компилятора. Поясню на примере.

Для того, чтобы Simulink понимал написанный вами M код, необходимо, чтобы структура кода имела специализированный формат S-функции.

Формат записи S-функции должен удовлетворять следующей схеме:

**[sys,x0,str,ts] = sfun(t,x,u,flag,varargin)**

Тут:

**sys** – настраиваемый возвращаемый аргумент (для *flag=3*, *sys* – выходной параметр S-функции);



**x0** – значение начального состояния системы (*игнорируется при flag = 0*);

**str** – переменная зарезервирована для использования фигурой.

**ts** – вектор, состоящий из 2-х значений [a b]. Данный параметр означает, что необходимо запускать модель каждые *a* секунд, начиная с *b* секунды после начала моделирования. В качестве *default* параметра можно использовать [-1 0] (*запускать в каждый момент моделирования, начиная с самого начала*).

**t** – текущее время;

**x** – вектор состояния;

**u** – входной вектор значений;

**flag** – ключ, который используется для многофункциональности S – блока;

**varargin** – совокупность входных данных модели типа cell.

Рассмотрим структуру параметра flag на основе табл 3.3, поскольку именно она определяет пути использования S функции.

Табл. 3.3. Структура параметра S-функции flag.

| flag | Подпрограммы S-функции | описание  |
|------|------------------------|---|
| 0    | mdlInitializeSizes     | Определяет основные характеристики блоков S функций                         |
| 1    | mdlDerivatives         | Производит расчёт переменных  |
| 2    | mdlUpdate              | Производит отработку тайминга   |
| 3    | mdlOutputs             | Подготовка выходных переменных  |
| 4    | mdlGetTimeOfNextVarHit | Производит расчёт следующего шага. Используется только с mdlInitializeSizes |
| 9    | mdlTerminate           | Выполнение необходимых завершающих моделирование действий                   |

Рассмотрим пример S-функции, выполненной в виде М-кода. Для описания примера выберем готовую S-функцию из директорий Matlab – sfun\_varargm.m

Первый блок кода производит инициализацию самой функции и определяет исходы переменных в зависимости от параметра flag, о котором говорилось ранее. При отсутствии параметра flag или его не правильном формате исход otherwise приводит к ошибке инициализации функции в целом.

```
function [sys,x0,str,ts] = sfun_varargm(t,x,u,flag,varargin)
switch flag,
case 0, % Initialization
```

```

[sys,x0,str,ts]=mdlInitializeSizes(varargin{:});
case 2, % Update
sys=mdlUpdate(t,x,u,varargin{:});
case 3, % Outputs
sys=mdlOutputs(t,x,u);
case 9, % Terminate
sys=[];
otherwise
error(['Unhandled flag = ',num2str(flag)]);
end

```

Далее описываются встроенные функции: инициализации (определяющая количество входных переменных), формирующая выходную переменную, и т.д.

```

function [sys,x0,str,ts]=mdlInitializeSizes(varargin)
sizes = simsizes;
nParams = nargin;
if nParams <= 0
error(['At least one parameter must be supplied']);
end
sizes.NumContStates = 0;
sizes.NumDiscStates = nParams;
sizes.NumOutputs = nParams;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = zeros(1,nParams);
str = [];
ts = [1 0];

```

```

function sys=mdlUpdate(t,x,u,varargin)
for i=1:nargin-3
sys(i) = x(i) + varargin{i};
end

```

```

function sys=mdlOutputs(t,x,u)
for i=1:length(x)
sys(i) = x(i);
end
%endfunction mdlOutputs
%[eof] sfun_varargm.m

```

В общем случае S-функция в М-коде должна выглядеть именно так. Добавление кода производится в блок S-функции в рабочем окне Simulink (рис. 3.26).

Далее приведу видеогаммы полученной модели и параметры её настройки посредством блоков Simulink.

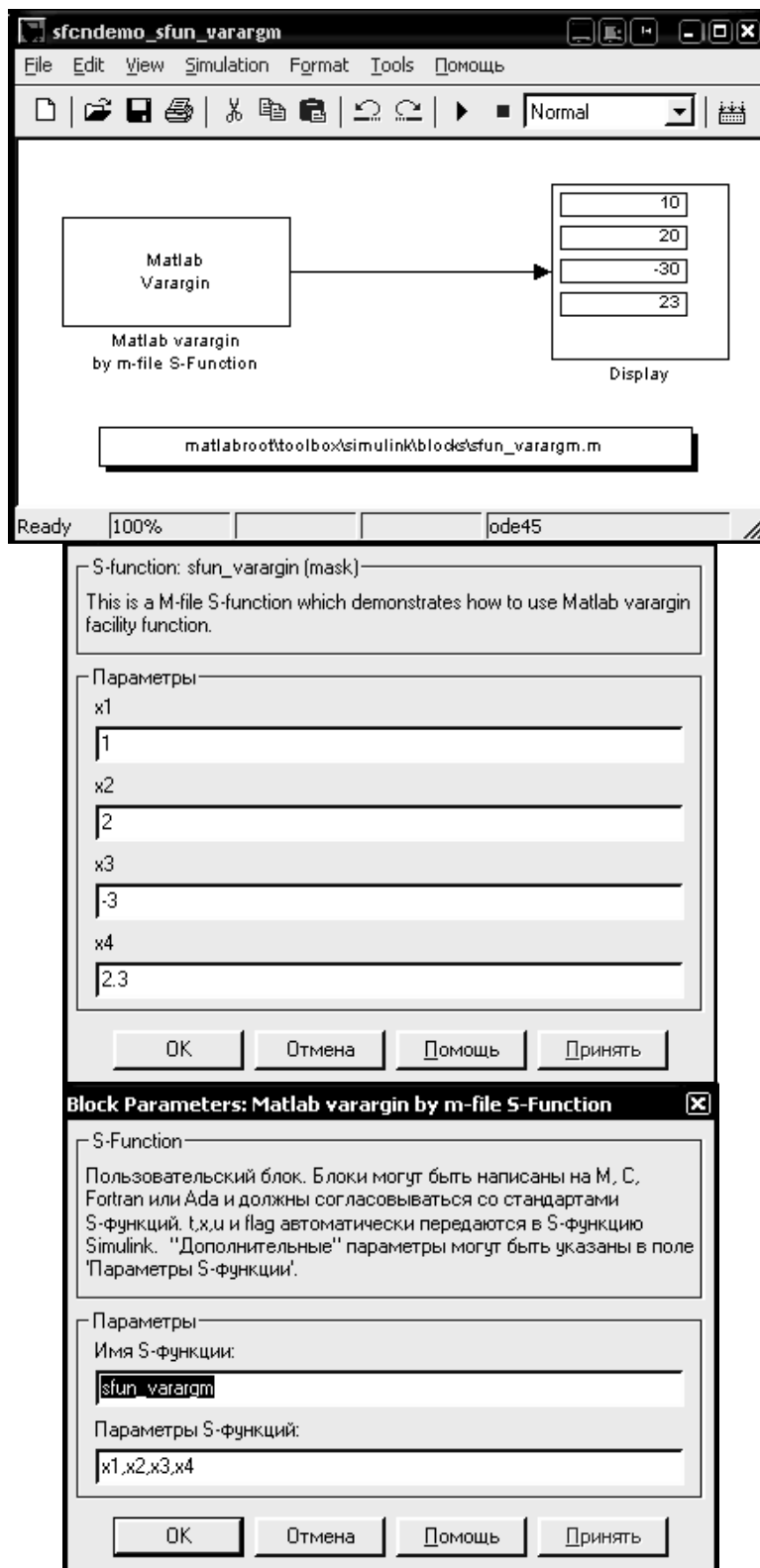


Рис. 3.26. Видеогамма модели, блок ввода параметров S-функция

### 3.12.5. Состав S-функции в Simulink

Состав S-функций в Simulink можно наблюдать на рис. 3.27. Подробное описание доступно по команде *sfundemos* в Matlab.

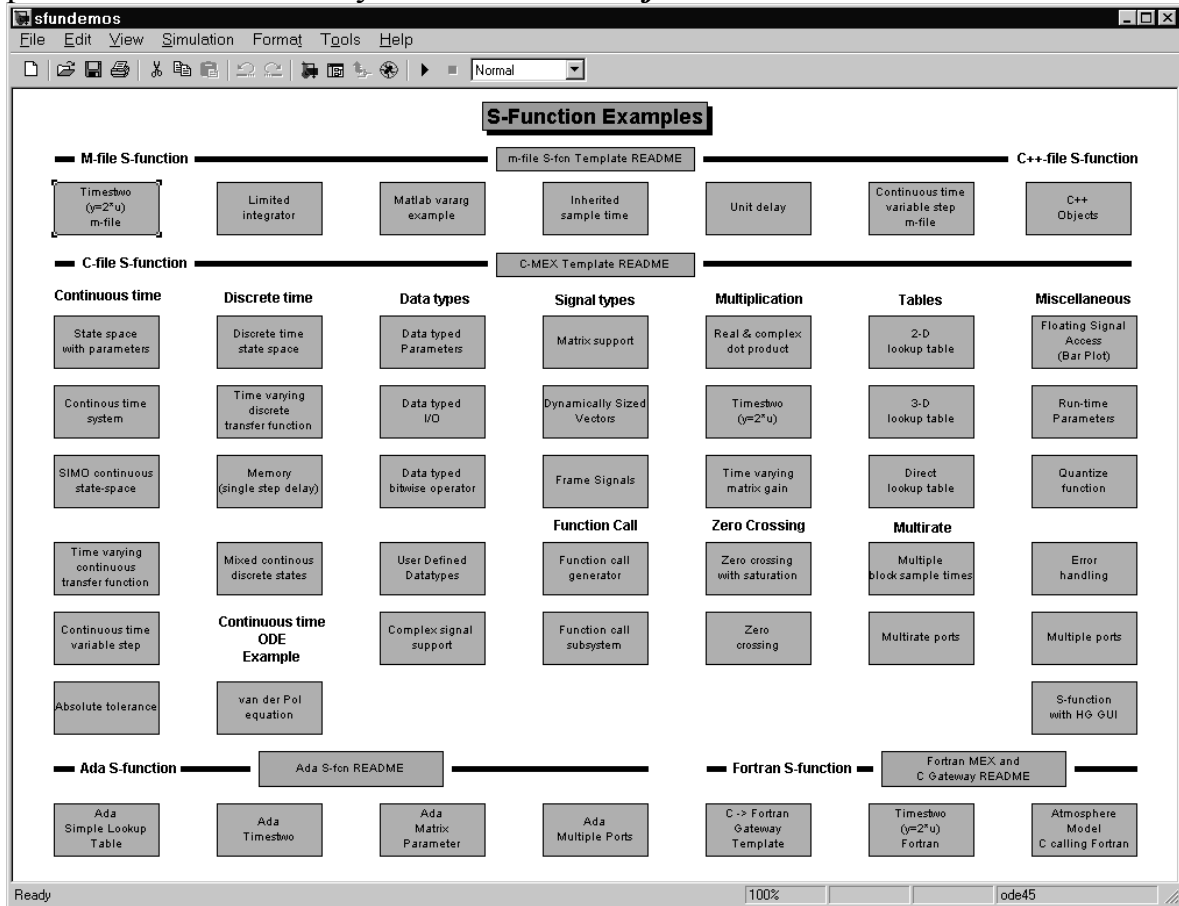


Рис. 3.27. Состав S-функций в Simulink

## ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

### Вопросы к разделу № 1:

- 1) Назначение УГК;
- 2) Какие основные правила необходимо соблюдать при подключении мониторов к УГК;
- 3) Какие типы мониторов можно подключать в УГК;
- 4) Типы DVI интерфейсов;
- 5) Какого типа DVI интерфейс используется в составе лабораторного комплекса;
- 6) Какова максимальная длина проводного кабеля для передачи DVI сигнала;
- 7) Рассчитать объём графической информации, передаваемой по одному DVI каналу на LCD панель;
- 8) Какова максимальная пропускная способность DVI канала передачи медиаданных;
- 9) Какой тип кодирования применяется при передаче медиасигнала по DVI интерфейсу;
- 10) Возможно ли построение видеостены в составе действующих комплектующих лабораторного комплекса без участия УГК Matrox;
- 11) Каким образом к 4х каналному УГК подключается 6 LCD мониторов;
- 12) Какие типы мониторов используются к лабораторном комплексе;
- 13) Технические характеристики LCD монитора Samsung 400PX;
- 14) Обоснуйте выбор типа мониторов в лабораторном комплексе;
- 15) Технологии отображения информации, применяемые в промышленности;
- 16) Структура видеоархива ОКО;
- 17) Технические характеристики видеосервера ОКО Архив Лайт;
- 18) Какого типа сигналы архивирует ОКО Архив Лайт;
- 19) Структура канала передачи данных ОКО Архив Лайт ←→ПК оператора;
- 20) Типы клавиатур применяемых в промышленности;
- 21) Какого типа клавиатура используется в составе лабораторного комплекса;
- 22) Каким образом сигналы от функциональной клавиатуры регистрируются в ВСО;
- 23) Почему в составе пульта оператора используется ПО «Тайфун»;

- 24) Почему в составе пульта оператора используется SCADA система;
- 25) Какая SCADA система используется в составе пульта оператора;
- 26) В каких режимах работает Видеостена. Приведите их характеристики;

**Вопросы к разделу № 2:**

- 27) Приведите общие сведения о назначении, структуре и основных функциях SCADA - систем.
- 28) Общие сведения о программном комплексе TRACE MODE. Поясните понятие «канал прохождения информации». Типы каналов.
- 29) Назовите атрибуты каналов ввода – вывода. Основные переменные (значения) на каналах и процедуры их обработки.
- 30) Какого типа архивы поддерживаются в TRACE MODE? Как выполнить настройку МРВ для архивирования информации на каналах ввода-вывода?
- 31) Организация работы системы в реальном времени. Приоритеты выполнения задач. Временные характеристики системы и ее настройка.
- 32) Назовите типовые режимы сетевого обмена данными между узлами распределенной АСУ ТП. Приведите их характеристики.
- 33) Как организуется работа МРВ TRACE MODE при работе в составе сетевых комплексов. Обмен по сети. Настройка сетевого обмена.
- 34) Охарактеризуйте механизмы межзадачного обмена данными с приложениями WINDOWS, поддерживаемые в TRACE MODE.
- 35) Рассмотрите механизм динамического обмена данными (DDE) между МРВ и электронной таблицей Excel.
- 36) Поясните, как организовать информационный обмен МРВ с базами данных предприятия через механизм ODBC.
- 37) Поясните, как организовать информационный обмен МРВ с другими SCADA – системами с использованием механизма OPC.
- 38) Приведите общие требования при реализации интерфейса оператора автоматизированной системы.
- 39) Какие графические возможности заложены в TRACE MODE для создания интерфейса пульта оператора.
- 40) Приведите характеристики языков программирования, соответствующих стандарту МЭК 1131.

- 41) Охарактеризуйте встроенные средства разработки и отладки программ в TRACE MODE

**Вопросы к разделу № 3:**

- 42) Что такое Matlab и почему он применяется в САПР проектировании;
- 43) Что такое язык М: тип языка, возможности, область применения;
- 44) Преимущества и недостатки технологии непосредственного программирования;
- 45) Преимущества и недостатки технологии визуально-ориентированного программирования;
- 46) Алгоритм построения MEX кодов;
- 47) Назначение MEX кодов Matlab;
- 48) Алгоритм построения EXE кодов;
- 49) Назначение EXE кодов;
- 50) Алгоритм проектирования Р кодов;
- 51) Назначение Р кодов;
- 52) Что такое JIT;
- 53) Какие типы компиляторов используются в Matlab;
- 54) Что такое msc;
- 55) Что такое Lcc;
- 56) Описать общую конструкцию объекта таймера в Matlab;
- 57) Перечислите основные свойства объекта таймер в Matlab.
- 58) Режимы работы объекта таймер в Matlab;
- 59) Что такое Simulink;
- 60) Опишите структуру S function;
- 61) Перечислите блоки пользовательских функций в Simulink;
- 62) Перечислить команды управления моделированием и программирования пользовательских блоков в Simulink.

## Библиографический список

1. «DVI». Электронный журнал «Hi-Fi News.ru». <http://www.hifinews.ru>. Дата обращения 14.04.2007.
2. «Цифровое подключение ЖК мониторов». Журнал: «Tom's hardware guide». <http://www.thg.ru/graphic/20041203/index.html>. Дата обращения 14.04.2007.
3. ГОСТ 22269-76. Рабочее место оператора. Взаимное расположение элементов рабочего места.
4. Средства визуализации для управления // Intelligent enterprise. – 2006. – №6.
5. TRACE MODE 6. Руководство пользователя. В 2-х томах. – 2006.
6. Глухов Ф.В. Новые технологии разработки операторского интерфейса в SCADA TRACE MODE 6 // XI международная конференция Управление производством в системе TRACE MODE, М: AdAstra Research Group, Ltd, 2005.
7. Б.А. Душков, Б.Ф. Ломов Основы инженерной психологии. – М.: Высшая школа, 1977. – 335 с.
8. А. Локотков Что должна уметь система SCADA //Современные технологии автоматизации, №3, 1998.
9. А.Н. ЛЮБАШИН. Стандартная технология программирования контроллеров // Промышленные АСУ и контроллеры, №5, 2000.

### Список рекомендуемой литературы

1. Дьяконов В. П., Matlab 6.5 SP1/7 + Simulink 5/6. Основы применения. Серия «Библиотека профессионала». – М.: СОЛОН-Пресс, 2005. – 800 с.: ил.
2. В.П.Дьяконова, В.В. Круглова «Математические пакеты расширения MATLAB. Специальный справочник.» – С. Пб.: «Питер», 2001. – 480с.
3. Lurie В. J. Classical feedback control width Matlab / Boris J. Lurie, Enright. p. cm. – (Control Engineering; 6)
4. А. Ф. Дащенко, В. Х. Кирилов, Л. В. Коломиец, В. Ф. Оробей, Matlab в инженерных и научных расчётах. – Одесса: «Астропринт», 2003. – 214с.



Мезенцев Антон Алексеевич  
Павлов Вадим Михайлович  
Байструков Константин Иванович

**ТЕХНИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ  
ЛАБОРАТОРНОГО КОМПЛЕКСА «ОРГАНИЗАЦИЯ  
ПУЛЬТОВ УПРАВЛЕНИЯ СОВРЕМЕННЫХ АСУ ТП»**

Научный редактор


Редактор

Подписано к печати . Формат 60x84/16. Бумага «Классика».  
Печать RISO. Усл.печ.л. 6,86. Уч.-изд.л. 6,21.  
Заказ . Тираж 100 экз.



Томский политехнический университет  
Система менеджмента качества  
Томского политехнического университета  
сертифицирована  
NATIONAL QUALITY ASSURANCE по стандарту  
ISO 9001:2000



ИЗДАТЕЛЬСТВО  ТПУ. 634050, г. Томск, пр. Ленина, 30.