Владимир Давыдов

SCADA-СИСТЕМЫ В УПРАВЛЕНИИ. Введение (SCADA-система GeniDAQ)

Учебное пособие

Санкт-Перербург
2010

Содержание

Предисловие

Учебное пособие ориентировано на многоуровневую образовательно-профессиональную подготовку бакалавров и магистров по направлению 220200 "Управление в технических системах" и предназначено для подготовки магистров. Согласно "Перечня направлений ВПО РФ для ГОС третьего поколения" направление 220200 входит в группу направлений 220000 "Автоматизация и управление" вместе с направлениями подготовки 220100 "Системный анализ и управление" и 220300 "Автоматизация технологических процессов и производств".

Учебное пособие посвящено бурно развивающимся в последнее время SCADA-системам (Supervisory Control and Data Acquisition, системам диспетчерского управления и сбора данных). В нем рассмотрены вопросы, связанные с использованием SCADA-систем в управлении и разработкой SCADA-приложений.

В учебном пособии дается обзор SCADA-систем, рассматривается промышленный стандарт OPC, являющийся средством интеграции компонентов в промышленной автоматизации. Далее, в качестве базовой, рассматривается SCADA-система GeniDAQ (фирма Advantech, США), ее инструментальный язык программирования VBA и средства привязки языка VBA к SCADA-системе. Отличительной особенностью SCADA-системы Advantech GeniDAQ является простота ее освоения при наличии всех характерных свойств и особенностей присущих SCADA-системам.

Прилагаемый к учебному пособию материал содержит установочные файлы для инсталляции SCADA-системы Advantech GeniDAQ, демонстрационные проекты и техническую документацию. Это делает учебное пособие самодостаточным. Для удобства пользователей учебное пособие содержит упражнения и варианты заданий на курсовое проектирование, выполнение которых способствует более успешному освоению изучаемого материала, а демонстрационные примеры по основным изучаемым разделам являются удобным средством получения справочного материала. Это позволяет использовать пособие и для самостоятельного изучения материала.

Для облегчения и ускорения освоения изучаемого материала учебное пособие поддерживает следующие методические приемы.

- Изучение материала начинается с использования очень простой и удобной для начального изучения SCADA-системы Advantech GeniDAQ.
- Теоретические и практические занятия чередуются по каждой теме и занимают соответственно 40% и 60% учебного времени.
- Теоретические занятия проводятся с использованием мультимедийного проектора, а практические занятия проводятся индивидуально в компьютерном классе с локальной вычислительной сетью.
- Часть учебного материала осваивается в процессе самостоятельной работы.
- При изучении материала используются специально разработанные демонстрационные примеры и каждый обучающийся снабжается электронной документацией.

- Для удобства пользователей учебное пособие содержит упражнения и варианты заданий на курсовое проектирование, выполнение которых способствует более успешному освоению изучаемого материала, а сквозные демонстрационные примеры по основным изучаемым разделам являются удобным средством получения справочного материала. Это позволяет использовать пособие и для самостоятельного изучения материала.

Соглашения, принятые в учебном пособии

Исходные тексты программ, приводимые в учебном пособии, для удобства читателей печатаются с использованием моноширинного шрифта `Courier New`, а служебные слова языка в тексте учебного пособия и в исходных текстах программ выделяются **полужирным шрифтом**. Названия окон, полей окон, меню, команд, клавиш, кнопок и названия демонстрационных примеров (приложений) в тексте пособия также выделяются **полужирным шрифтом**. Выбор команды меню показывается с помощью символа (|), например, **File | Run**. Активизация акселератора (коминации клавиш) показывается с помощью символа (+), например, **Ctrl+S**.

Курсивом в тексте выделяются определяющие вхождения новых понятий, а также отдельные слова или выражения, на которые следует обратить внимание.

Имена файлов и их расширения пишутся без кавычек и без выделения.

Кроме шрифтовых выделений, используется три типа специальных абзацев: советы, замечания и примечания.

> ***Совет***
> Внимательно изучите все описанные далее характеристики и особенности SCADA-систем — это важно.

> ***Замечание***
> Обратите внимание на то, что а частном случае окно редактора задач может не использоваться. Именно такая ситуация имела место в приложении **Занятие 4**.

> ***Примечание***
> Атрибуты пользователя следует получить у преподавателя. Пользователь должен обладать правами администратора.

Ваши отзывы об учебном пособии, конструктивные замечания и критику направляйте по адресу: **davydov@aivt.ftk.spbstu.ru**.

Благодарности

Я благодарю моих студентов Санкт-Петербургского государственного политехнического университета за вопросы и замечания, которые способствовали

улучшению изложения материала. Особо выражаю благодарность за конкретную помощь Осипову Игорю (улучшение интерфейса OPC-серверов и оформления их исходного кода), Денисову Александру (лаконичное описание языка VBA и его тестирование).

Часть 1. SCADA-система Advantech GeniDAQ

Глава 1. Обзор SCADA-систем

Глава 2. OPC — промышленный стандарт и средство интеграции компонентов в промышленной автоматизации

Глава 3. Краткий обзор SCADA-системы GeniDAQ

Глава 4. Учебник по SCADA-системе GeniDAQ

Глава 5. OPC-технология информационной интеграции SCADA-системы и промышленного контроллера

Глава 1. Обзор SCADA-систем

За последние 10-15 лет за рубежом и в нашей стране резко возрос интерес к проблемам построения высоконадежных и высокоэффективных систем диспетчерского управления и сбора данных, получивших название *SCADA-систем* (SCADA — Supervisory Control And Data Acquisition, диспетчерское управление и сбор данных). Что же понимают под SCADA-системой и какими свойствами и особенностями она обладает?

1.1. Введение

Современное производство не может обойтись без *автоматизации*, которая позволяет повысить качество, снизить процент брака, увеличить выход продукции, снизить себестоимость продукции, сэкономить ресурсы и продлить срок службы технологического оборудования. В настоящее время автоматизированные системы управления промышленными предприятиями и технологическими процессами (*АСУ ТП*) имеют иерархическую структуру, включающую следующие неотъемлемые уровни [1]:

- *нижний уровень* — технологические устройства (датчики, контроллеры, исполнительные механизмы и т.д.);
- *средний уровень* — *SCADA*-системы;
- *верхний уровень* — приложения, управляющие ресурсами предприятия.

> ***Совет***
> Внимательно изучите все описанные далее характеристики и особенности SCADA-систем — это важно.

В рамках данного учебного курса мы будем интересоваться SCADA-системами. Диспетчерское управление и сбор данных является основным и перспективным методом автоматизированного управления сложными динамическими системами в жизненно важных и критичных областях. На основе SCADA-систем строятся крупные автоматизированные системы управления в промышленности, энергетике, на транспорте, в космической, военной областях и в различных государственных структурах. SCADA-система реализует процесс сбора информации в реальном времени с удалённых объектов для обработки, анализа и управления такими объектами. Прообразом современных SCADA-систем на ранних стадиях развития автоматизированных систем управления являлись системы телеметрии и сигнализации. SCADA-система — это совокупность устройств управления и мониторинга, а также средств их взаимодействия с технологическим объектом. Под термином SCADA понимают набор программных и аппаратных средств для реализации операторских рабочих мест, а поскольку программные средства являются здесь определяющими, то зачастую о SCADA-системах говорят лишь как о программном обеспечении.

Как отмечалось ранее, диспетчерское управление и сбор данных является основным и в настоящее время остается наиболее перспективным методом автоматизированного управления сложными динамическими системами (процессами) в жизненно важных и критичных с точки зрения безопасности и надежности областях [2]. За последние 10-15 лет за рубежом и в нашей стране резко возрос интерес к проблемам построения высоконадежных и высокоэффективных систем диспетчерского управления и сбора данных. С одной стороны, это связано со значительным прогрессом в области вычислительной техники, программного обеспечения и телекоммуникаций, что увеличивает возможности и расширяет сферу применения автоматизированных систем. С другой стороны, развитие информационных технологий, повышение степени автоматизации и перераспределение функций между человеком и аппаратурой обострило проблему взаимодействия человека-оператора с системой управления. Расследование и анализ большинства аварий и происшествий в авиации, наземном и водном транспорте, промышленности и энергетике, часть из которых привела к катастрофическим последствиям, показало следующее. Если в 60-х годах ошибка человека являлась первоначальной причиной лишь 20% инцидентов (80%, соответственно, за технологическими неисправностями и отказами), то в 90-х годах доля "человеческого фактора" возросла до 80%, причем, в связи с постоянным совершенствованием технологий и повышением надежности электронного оборудования и машин, доля эта может еще возрасти.

Основной причиной такой тенденции является старый, традиционный подход к построению сложных автоматизированных систем управления, который часто применяется и в настоящее время. Это ориентация, в первую очередь, на применение новейших технических (технологических) достижений, стремление повысить степень автоматизации и функциональные возможности системы и, в то же время, недооценка необходимости построения эффективного *человеко-машинного интерфейса*, ориентированного на пользователя (оператора). Не случайно, что именно на последние 15 лет, т. е. на период появления мощных, компактных и недорогих вычислительных средств, пришелся пик исследований в США по проблемам человеческого фактора в системах управления, в том числе по оптимизации архитектуры и HMI-интерфейса систем диспетчерского управления и сбора данных.

Изучение материалов по проблемам построения эффективных и надежных систем диспетчерского управления показало необходимость применения нового подхода при разработке таких систем (human-centered design), ориентированного в первую очередь на человека-оператора (диспетчера), вместо традиционного и повсеместно применявшегося подхода "hardware-centered", при котором при построении системы основное внимание уделялось выбору и разработке технических средств и программного обеспечения. Применение нового подхода привело к созданию SCADA-систем. Использование их в реальных космических и авиационных разработках и сравнительные испытания систем в Национальном управлении по аэронавтике и исследованию космического пространства США подтвердили их эффективность, позволив увеличить производительность операторов, на порядок

уменьшить процедурные ошибки и свести к нулю критические (некорректируемые) ошибки операторов.

1.2. Функциональная структура SCADA-системы

Существует два типа управления удаленными объектами в SCADA-системах: автоматическое и инициируемое оператором системы [2]. В SCADA-системах можно выделить четыре основных функциональных компонента — человек-оператор, компьютер взаимодействия с человеком (рабочая станция), компьютер взаимодействия с задачей (объектом) и задача (объект управления). Человеку-оператору в SCADA-системе присущи пять основных функций, которые можно рассматривать как набор вложенных циклов, в которых оператор:

- планирует, какие следующие действия необходимо выполнить;
- обучает (программирует) компьютерную систему на последующие действия;
- отслеживает результаты полуавтоматической работы системы;
- вмешивается в процесс в случае критических событий, когда автоматика не может справиться, либо, при необходимости, подстраивает параметры процесса;
- обучается в процессе работы (получает опыт).

Данное представление SCADA-системы явилось основой для разработки современных методологий построения эффективных диспетчерских систем.

1.3. Особенности SCADA как процесса управления

Процессу управления в современных диспетчерских системах присущи следующие особенности [2]:

- процесс SCADA применяется в системах, в которых обязательно наличие человека (оператора, диспетчера);
- процесс SCADA был разработан для систем, в которых любое неправильное воздействие может привести к отказу (потере) объекта управления или даже катастрофическим последствиям;
- оператор несет, как правило, общую ответственность за управление системой, которая, при нормальных условиях, только изредка требует подстройки параметров для достижения оптимальности;
- активное участие оператора в процессе управления происходит нечасто и в непредсказуемые моменты времени, обычно в случае наступления критических событий (отказы, нештатные ситуации и т.п.);
- действия оператора в критических ситуациях могут быть жестко ограничены по времени (несколькими минутами или даже секундами).

1.4. Основные требования к SCADA-системам

К SCADA-системам предъявляются следующие основные требования.

- Надёжность системы (технологическая и функциональная). Никакой единичный отказ или единичная ошибка не должны вызывать выдачу ложного выходного воздействия.
- Безопасность управления.
- Точность обработки и представления данных.
- Простота расширения системы.
- Интуитивность, простота использования.

Требования безопасности и надежности управления в SCADA-системах сводятся к следующему. Никакой единичный отказ оборудования не должен вызывать выдачу ложного выходного воздействия (команды) на объект управления. Никакая единичная ошибка оператора не должна вызывать выдачу ложного выходного воздействия (команды) на объект управления. Все операции по управлению должны быть интуитивно-понятными и удобными для оператора (диспетчера).

В силу требований, предъявляемых к SCADA-системам, спектр их функциональных возможностей определен и реализован практически во всех SCADA-пакетах. Перечислим основные возможности и средства, присущие всем SCADA-системам и различающиеся только техническими особенностями реализации:

- автоматизированная разработка, дающая возможность создания программного обеспечения (ПО) системы автоматизации без реального программирования, т.е. без написания программного кода в классическом смысле;
- непосредственное автоматическое управление технологическим процессом в соответствии с заданными алгоритмами;
- средства приёма первичной информации о контролируемых технологических параметрах от контроллеров нижних уровней и датчиков;
- средства управления и регистрации сигналов в аварийных ситуациях;
- средства приёма команд оператора и передача их исполнительным механизмам и контроллерам нижних уровней;
- средства регистрации событий, связанных с контролируемым технологическим процессом и действиями персонала, ответственного за эксплуатацию и обслуживание системы;
- средства хранения информации с возможностью ее постобработки;
- средства вторичной обработки принятой информации;
- средства визуализации информации о технологическом процессе и архивной информации в удобной для восприятия форме;
- средства формирования сводок и других отчётных документов на основе архивной информации;

- средства обмена информацией с автоматизированной системой управления предприятием;
- возможность работы прикладной системы с наборами параметров, рассматриваемыми как "единое целое".

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 объёме, в котором это необходимо и присуще всем SCADA-системам.

1.5. Функциональные возможности SCADA-систем

В основе большинства SCADA-пакетов лежит использование нескольких программных компонентов (базы данных реального времени, ввода-вывода, предыстории, аварийных ситуаций) и программных администраторов (доступа, управления, сообщений). Следует отметить, что в целом технологии проектирования систем автоматизации на основе SCADA-систем очень похожи:

- Разработка архитектуры системы автоматизации в целом. На этом этапе определяется функциональное назначение каждого узла системы автоматизации.
- Решение вопросов, связанных с возможной поддержкой распределенной архитектуры, необходимостью введения узлов с "горячим резервированием" и т. п.
- Создание прикладной системы управления для каждого узла. На этом этапе специалист в области автоматизируемых процессов наполняет узлы архитектуры алгоритмами, совокупность которых позволяет решать задачи автоматизации.
- Приведение в соответствие параметров прикладной системы с информацией, которой обмениваются устройства нижнего уровня (например, программируемые логические контроллеры) с внешним миром (датчики температуры, давления и др.).
- Отладка созданной прикладной программы в режиме эмуляции и в реальном режиме.

Перечисленные выше возможности SCADA-систем в значительной мере определяют стоимость и сроки создания ПО, а также сроки ее окупаемости.

1.6. Программные платформы SCADA-систем

Программно-аппаратные платформы, на которых реализованы SCADA-системы, являются важной характеристикой для оценки их функциональности. Анализ таких платформ необходим, поскольку от него зависит ответ на вопросы распространения SCADA-системы на имеющиеся вычислительные средства, а также оценка стоимости эксплуатации системы. В различных SCADA-системах этот вопрос решен по разному

[3, 4]. Так, SCADA-система FactoryLink (United States DATA Co.) имеет весьма широкий список поддерживаемых программно-аппаратных платформ: DOS, OS/2, Unix, VMS, AIX, Windows NT/2K/XP. В то же самое время, в таких SCADA-системах, как RealFlex (BJ Software Systems), Sitex (Jade Software) основу программной платформы принципиально составляет единственная операционная система реального времени QNX. Подавляющее большинство SCADA-систем реализовано на MS Windows платформах. Именно такие системы предлагают наиболее полные и легко наращиваемые MMI-средства (Man Machine Interface). Учитывая продолжающееся усиление позиций фирмы Microsoft на рынке операционных систем (ОС) следует отметить, что даже разработчики многоплатформенных SCADA-систем, такие как United States DATA Co, приоритетным считают дальнейшее развитие своих SCADA-систем на платформе Windows (табл. 1.1).

Таблица 1.1. *SCADA-системы на российском и западном рынках*

SCASA-системы	**Фирма-изготовитель**	**Страна**	**2K/XP поддержка**	**OPC**
InTouch	Wonderware	США	✓	✓
GeniDAQ	Advantech	США	✓	✓
Advantech FX	Advantech	США	✓	✓
Genesis32	Iconics	США	✓	✓
Trace Mode	AdAstra	Россия	✓	✓
Citect	Citect Pty. Ltd	Австралия	✓	✓
Factory Link	US DATA	США	✓	✓
LabView	National Instruments	США	✓	✓
RSView	Rockwell Software	США	✓	✓
RealFlex	BJ Software Systems	США	-	-
Sitex	Jade Software	Англия	-	-
iFIX	Intelution	США	✓	✓
MasterSCADA	inSAT	Россия	✓	✓
PI System	OSI Software	США	✓	✓
Контур	Объединение Юг	Украина	✓	✓

Некоторые фирмы, до сих пор поддерживавшие SCADA-системы на базе ОС реального времени (РВ), начали менять ориентацию, выбирая системы на платформе Windows. Фирма BJ Software Systems объявила о том, что SCADA-пакет RealFlex дальше развиваться не будет и перевод его на платформу Photon не планируется. Все более очевидным становится применение ОС реального времени, в основном, во встраиваемых системах, где они действительно хороши. Таким образом, основным полем, где сегодня разворачиваются главные события глобального рынка SCADA-систем, стали ОС MS Windows NT/2K/XP. Быстрое развитие OPC-технологий (см. далее подразд. 1.10), низкие цены аппаратного обеспечения,

распространённость ОС Windows на офисных рынках с её солидными техническими характеристиками — главные причины того, что абсолютное большинство производителей SCADA-пакетов мигрировали в сторону этой операционной системы.

1.7. Средства сетевой поддержки SCADA-систем

Одной из основных черт современного мира систем автоматизации является их высокая степень интеграции. В любой из них могут быть задействованы объекты управления; исполнительные механизмы; аппаратура, регистрирующая и обрабатывающая информацию; рабочие места операторов; серверы баз данных и т. д.

Очевидно, что для эффективного функционирования в этой разнородной аппаратной среде SCADA-система должна обеспечивать высокий уровень сетевого сервиса. Желательно, чтобы она поддерживала работу в стандартных сетевых средах (Arcnet, Ethernet и т .д.) с использованием стандартных протоколов (Netbios, TCP/IP и др.), а также обеспечивала поддержку наиболее популярных сетевых стандартов из класса промышленных интерфейсов (Profibus, CANbus, Lon, Modbus и т.д.). Этим требованиям в той или иной степени удовлетворяют практически все рассматриваемые SCADA-системы, с тем только различием, что набор поддерживаемых сетевых интерфейсов, конечно же, в различных SCADA-системах разный. Сетевые средства SCADA-системы GeniDAQ используют протокол TCP/IP, который обеспечивает возможность работы с данными технологического процесса в реальном времени с любого узла сети, а также дистанционное управление процессом. Сетевые средства SCADA-систем Citect, Trace Mode поддерживаются такими сетевыми протоколами, как NetBEUI, IPX/SPX, TCP/IP.

1.8. Встроенные языки программирования SCADA-систем

Встроенные языки программирования — мощное средство SCADA-систем, облегчающее процесс реализации сложных алгоритмов обработки и анализа данных. Кроме того, встроенные языки программирования являются мощным и универсальным средством адаптации систем к требованиям прикладной задачи.

Первые версии SCADA-систем либо не имели подобных языков, либо эти языки реализовывали небогатый набор функций. В современных версиях SCADA-систем функциональные возможности языков становятся существенно богаче. Явно выделяются два подхода.

- Ориентация встроенных языков программирования на технологов. Функции в таких языках являются высокоуровневыми, не требующими профессиональных навыков программирования при их использовании. Количество таких функций в базовых поставках относительно невелико, хотя существуют свободно распространяемые библиотеки дополнительных функций.

- Ориентация на системных интеграторов. В этом случае в качестве языков чаще всего используются Visual Basic-подобные языки.

В каждом языке допускается расширение набора функций. В языках, ориентированных на технологов, это расширение достигается с помощью дополнительных инструментальных средств (Toolkits). Разработка дополнительных функций выполняется обычно программистами-профессионалами. Разработка новых функций при втором подходе выполняется разработчиками приложений (как и в традиционных языках программирования). Полнота использования возможностей встроенных языков (особенно при втором подходе) требует соответствующего уровня квалификации разработчика, если, конечно, в этом есть необходимость. Требования задачи могут быть не столь высокими, чтобы применять всю "мощь" встроенного языка. Во всех языках функции разделяются на группы, часть из которых присутствует практически во всех языках: математические функции, функции работы со строками, SQL-обмен, DDE-обмен и т. д. Каждая из функций во встроенном языке выполняется в синхронном или асинхронном режиме. В синхронном режиме выполнение следующей функции не начинается до тех пор, пока не завершилось исполнение предыдущей. При запуске асинхронной функции управление переходит к следующей функции, не дожидаясь завершения исполнения предыдущей функции. В связи с этим возникает несколько вопросов. С каким приоритетом исполняется каждый из фрагментов, допускается ли рекурсия при обработке событий и если да, то каков уровень вложенности? В SCADA-системах уровень вложенности пока не стандартизован, но оговаривается особо в рамках каждой из них.

SCADA-системы VisiDAQ (Advantech, США) и GeniDAQ включают в себя встроенную среду программирования на языке сценариев [5], совместимом с языком программирования высокого уровня Visual Basic for Applications (VBA). Язык Visual Basic (VB) является одним из наиболее популярных языков программирования. Это позволяет разрабатывать управляющие стратегии практически любого уровня сложности. Редактор сценариев SCADA-систем VisiDAQ и GeniDAQ предназначен для редактирования основного сценария и Basic-сценариев внутри задач. Основной сценарий полностью контролирует процесс выполнения задачи, включая ее запуск и/или останов. Кроме управления задачами, SCADA-система VisiDAQ предоставляет разнообразные команды для ввода-вывода данных, включая операции открытия, чтения, записи, закрытия последовательного порта; операции доступа к информации в центре обработки данных и т. п.

В SCADA-системе Citect [6, 7] встроен гибкий язык программирования Cicode, созданный специально для мониторинга и управления приложениями. Это структурированный язык, похожий на Visual Basic и C. Применение языка Cicode предоставляет пользователю доступ к данным проекта в режиме реального времени, а также ко всем переменным, сигналам тревог, трендам, отчетам и т. д. Язык Cicode поддерживает многозадачность и удаленный вызов процедур, позволяет создавать программы любой степени сложности.

В SCADA-системе InTouch используются скрипты. Скрипты в InTouch — это программные фрагменты, активизируемые по событиям (по нажатию клавиши, кнопки, открытию окна, изменению значения переменной и т. д.).

1.9. Базы данных

Многие SCADA-системы, в частности, Genesis, InTouch используют ANSI SQL синтаксис, который является независимым от типа базы данных. Таким образом, приложения виртуально изолированы, что позволяет менять базу данных без серьезного изменения самой прикладной задачи, создавать независимые программы для анализа информации, использовать уже наработанное программное обеспечение, ориентированное на обработку данных. Так, IndustrialSQL Server компании Wonderware использует подобный подход, впервые превращая реляционную технологию в разумное решение для систем промышленной автоматизации. IndustrialSQL Server — опора пакета промышленной автоматизации Wonderware FactorySuite2000. Несмотря на то, что IndustrialSQL Server поставляется компанией Wonderware как самостоятельный продукт, он, в то же время, является одним из главных компонентов пакета FactorySuite2000, являясь, можно сказать, его "сердцем". Будучи интегрированным со SCADA-системой InTouch, IndustrialSQL Server способен накапливать при помощи серверов ввода/вывода информацию практически от любых измерительных приборов и устройств сбора данных. IndustrialSQL Server представляет собой расширение Microsoft SQL Server. Объединение серверов IndustrialSQL Server и Microsoft SQL Server незаметно для пользователя. Можно сказать, что IndustrialSQL Server превращает Microsoft SQL Server в сервер реляционной базы данных реального времени. При этом клиенты могут напрямую обращаться к IndustrialSQL Server при помощи тех же утилит, что используются сервером Microsoft SQL Server. Выбор Microsoft SQL Server в качестве основы для IndustrialSQL Server объясняется несколькими причинами. Во-первых, в мире существует более 200 миллионов пользователей Microsoft SQL Server. Во-вторых, Microsoft SQL Server является самой продаваемой базой данных для ОС Windows. В-третьих, SQL поддерживается всеми крупными производителями серверов баз данных, большинством средств разработки и языков программирования.

Родственный Citect продукт, называемый Plant2SQL, позволяет предоставлять технологическую информацию, являющуюся прерогативой SCADA-систем. Plant2SQL поддерживает простой доступ к данным технологического процесса, как из приложений, так и со стороны пользователей. Пользователям теперь доступны самые последние данные технологического процесса, что позволяет им принимать решения во всеоружии, полностью владея информацией о процессе производства.

1.10. Организация взаимодействия с устройствами нижнего уровня

Современные SCADA-системы не ограничивают выбор аппаратуры нижнего уровня (контроллеров), так как предоставляют большой набор драйверов или серверов ввода/вывода и имеют хорошо развитые средства создания собственных программных модулей или драйверов новых устройств нижнего уровня. Для подсоединения драйверов ввода/вывода к SCADA-системе в настоящее время используются следующие механизмы:

- ставший стандартом динамический обмен данными (DDE);
- собственные протоколы, связанные со SCADA-системами, реально обеспечивающие самый скоростной обмен данными;
- новый OPC-протокол, который, с одной стороны, является стандартным и поддерживается большинством SCADA-систем (см. приведенную ранее табл. 1.1), а с другой стороны, лишен недостатков протоколов DDE.

Изначально протокол DDE применялся в первых человеко-машинных интерфейсах в качестве механизма разделения данных между прикладными системами и устройствами типа ПЛК (программируемых логических контроллеров). Для преодоления недостатков DDE, прежде всего для повышения надежности и скорости обмена, разработчики предложили свои собственные решения (протоколы), такие как AdvancedDDE или FastDDE — протоколы, связанные с пакетированием информации при обмене с ПЛК и сетевыми контроллерами. Но такие частные решения приводят к ряду проблем:

- для каждой SCADA-системы пишется свой драйвер для поставляемого на рынок оборудования;
- в общем случае, два пакета не могут иметь доступ к одному драйверу в одно и то же время, поскольку каждый из них поддерживает обмен именно со своим драйвером.

Взамен этих протоколов появился новый стандарт OPC (OLE for Process Control), ориентированный на рынок промышленной автоматизации, основанный на COM-технологии фирмы Microsoft. Основная цель OPC-стандарта заключается в определении механизма доступа к данным любого устройства из приложений. OPC позволяет производителям оборудования поставлять программные компоненты, которые стандартным способом обеспечат клиентов данными с ПЛК. При широком распространении OPC-стандарта появляются следующие преимущества:

- OPC позволяет определять на уровне объектов различные системы управления и контроля, работающие в распределенной гетерогенной среде;
- OPC устраняет необходимость использования различного нестандартного оборудования и соответствующих коммуникационных программных драйверов;
- у потребителя появляется больший выбор периферийного оборудования при разработке приложений.

С использованием OPC-решений, интеграция в гетерогенных системах становится достаточно простой. Применительно к SCADA-системам OPC-серверы, расположенные на всех компьютерах системы управления производственного предприятия, стандартным способом могут поставлять данные в программу визуализации, базы данных и т. п., уничтожая, в некотором смысле, само понятие неоднородной системы.

Для обмена данными с контроллерами в SCADA-системе Genie — предшественнике SCADA-системы GeniDAQ, могут использоваться встраиваемые драйверы, DDE-обмен, но в нем не поддерживается OPC-спецификация. SCADA-система GeniDAQ имеет ряд ключевых отличий, обеспечивающих решение более широкого круга задач на новом уровне, в том числе с поддержкой OPC-спецификации.

В SCADA-системе InTouch поддерживается также пакетированный DDE-обмен — FastDDE. Применение последнего заметно повышает эффективность и производительность обмена данными благодаря уменьшению общего количества DDE-пакетов, которыми клиент и сервер обмениваются между собой. Но принципиальные недостатки, связанные с надежностью и зависимостью от количества загруженных в текущий момент приложений Windows, остались. Необходимость в появлении более совершенного технологичного протокола созрела. С целью расширения возможностей стандартного протокола DDE на локальную сеть компания Wonderware предложила протокол NetDDE. Он позволяет приложениям, запущенным на объединенных в локальную сеть компьютерах, вести DDE-обмен. Для реализации функций OPC-клиента Wonderware предлагает OPCLink-сервер, преобразующий OPC в SuitLink-протокол. Параметры производительности протокола SuiteLink превосходят параметры DCOM. OPCLink-сервер обеспечивает прием информации с OPC-сервера и передачу ее по протоколу SuiteLink в SCADA-систему InTouch и наоборот. Именно OPCLink-сервер рекомендуется устанавливать на одном узле с OPC-сервером, чтобы для сетевых передач использовался SuiteLink-протокол, а не DCOM.

Для обмена данными с контроллерами в SCADA-системе Citect могут использоваться следующие способы: встраиваемые драйверы, DDE-обмен, OPC-протоколы. Первый путь предполагает создание динамических библиотек, выполняющих функцию драйверов. В составе SCADA-системы Citect поставляется более чем 120 драйверов ввода/вывода. Связь через DDE-сервер использует стандартный коммуникационный протокол Windows. SCADA-система Citect поддерживает связь с любым DDE-сервером. Система Citect может функционировать в качестве и OPC-сервера и OPC-клиента.

1.11. Открытость SCADA-систем

Система является открытой, если для нее определены и описаны используемые форматы данных и процедурный интерфейс, что позволяет подключить к ней "внешние", независимо разработанные компоненты.

Разработка собственных программных модулей. Перед фирмами-разработчиками систем автоматизации часто встает вопрос о создании собственных (не

предусмотренных в рамках SCADA-систем) программных модулей и включение их в создаваемую систему автоматизации. Поэтому вопрос об открытости системы является важной характеристикой SCADA-систем. Фактически открытость системы означает доступность спецификаций системных (в смысле SCADA) вызовов, реализующих тот или иной системный сервис. Это может быть и доступ к графическим функциям, функциям работы с базами данных и т. д.

Драйверы ввода-вывода. Современные SCADA-системы не ограничивают выбор аппаратуры нижнего уровня, так как предоставляют большой набор драйверов или серверов ввода-вывода и имеют хорошо развитые средства создания собственных программных модулей или драйверов новых устройств нижнего уровня. Сами драйверы разрабатываются с использованием стандартных языков программирования. Вопрос, однако, в том, достаточно ли только спецификаций доступа к ядру системы, поставляемых фирмой-разработчиком в штатном комплекте (система Trace Mode), или для создания драйверов необходимы специальные пакеты (системы FactoryLink, InTouch), или же, вообще, разработку драйвера нужно заказывать у фирмы-разработчика.

Разработки третьих фирм. Объекты ActiveX — это объекты, в основе которых лежит Microsoft COM. Технология COM определяет общую схему взаимодействия компонентов программного обеспечения в среде Windows и предоставляет стандартную инфраструктуру, позволяющую объектам обмениваться данными и функциями между прикладными программами. Большинство SCADA-систем являются контейнерами, которые уведомляются ActiveX о происшедших событиях. Этот факт очень важно оценивать при выборе SCADA-пакета, поскольку это расширяет область применения системы непрофессиональными программистами.

Подавляющее большинство SCADA-пакетов, несмотря на их реализацию на стандартной платформе Windows, являются закрытыми, если речь идет о структуре данных, с которыми они работают. Конечно большинство из них (в том числе GeniDAQ) имеют стандартные интерфейсы типа DDE, OLE, ODBC и др., сохраняя при этом специализированные, нестандартные форматы файлов для архивов, графиков и сигнализаторов. Для того чтобы получить доступ к данным в этих системах, необходимо воспользоваться специализированными пакетами разработчика, а часто это вообще невозможно. SCADA-система Citect не делает секрета из формата своих данных — все параметры и данные приложения хранятся в формате dBase, понимаемом любым пакетом. Кроме этого, данные могут сохраняться и в более современных SQL базах данных, используя стандартные интерфейсы Citect. Эта открытость будет по достоинству оценена теми, кто не имел возможности, скажем, отобразить на экране данные из архива в том виде, котором требуется оператору, а не так, как умеет SCADA.

1.12. О реальном времени

Одним из существенных недостатков SCADA-систем на платформах Windows 95/98/Me по сравнению с таковыми же системами на платформах ОС реального времени (ОС РВ) является отсутствие поддержки реального времени.

Ситуация резко изменилась с появлением Windows NT/2000/XP. Выход в свет этой ОС стимулировал разработку новых подходов в поддержке жесткого реального времени. Прежде всего, сама по себе Windows NT весьма успешно теснит ОС РВ. Тем не менее, Windows NT имеет ряд ограничений. Такие ее особенности, как предпочтение аппаратного прерывания над программным, выполнение в подпрограмме обработки аппаратных прерываний лишь необходимых действий с выполнением последующей обработки через очередь отложенных процедур, отсутствие приоритетной обработки процессов в очереди отложенных процедур, не позволяют отнести Windows NT к категории классических ОС реального времени. Ряд фирм (LP Elektronik, Imagination Systems, RadSys, Spectron Microsystems, Ventur Com) предприняли более радикальные попытки превратить Windows NT в ОС жесткого реального времени. Рассмотрим некоторые ключевые особенности реализации такой идеи на подсистеме реального времени RTX (Real Time Extension), предложенной фирмой Ventur Com. Именно эта реализация получает в настоящее время наиболее широкое распространение. Фирмы-разработчики SCADA-систем незамедлительно начали предлагать применение новых решений. Так, набор прикладных интерфейсов программирования RTX 4.1 (Ventur Com) в SCADA-системе FIX позволяет:

- осуществлять полный контроль над задачами реального времени;
- использовать фиксированную систему из 128 приоритетов для контроля RTX-задач;
- применять стандартные средства обмена данными между задачами;
- обращаться к стандартным функциям из Win32API.

Появление подобных решений наряду с собственными характеристиками Windows NT наносит сильный "удар" по SCADA--системам на базе ОС РВ, поскольку отнимает у них очень важный "козырь" — преимущества жесткого реального времени, и, для некоторых приложений, теснит применение ОС РВ во встраиваемых системах. Сегодня для весьма широкого спектра промышленных приложений уже есть реальная возможность использовать Windows NT как единую операционную систему, со всеми вытекающими отсюда последствиями, работающую даже в бездисковых конфигурациях в сетевых контроллерах ввода/вывода, скажем на платформе CompactPCI или VME, наряду с использованием на рабочем месте оператора-технолога.

Не секрет, что во многих SCADA-пакетах обновление данных организовано на основе опроса параметров в рамках одной основной задачи и соответствующего обновления экрана по заданным промежуткам времени. Этот алгоритм очень напоминает корпоративную многозадачность, реализованную в Windows 3.X. При этом последствия для работы всего приложения практически идентичны — если по каким либо причинам работа любой функции основной задачи нарушается, то нарушается работа всего пакета. Так, например, задержка в получении данных, необходимых для обновления результатов на одном(!) из графиков часто приводит в таких системах к полной остановке работы всего приложения до тех пор, пока запрос не будет обработан. Хотя здесь нечему удивляться. Многие SCADA-пакеты появились в эпоху Windows 3.X и их ядро не изменялось с тех далеких пор.

Ядро Citect было изначально задумано и реализовано с учетом принципов построения систем реального времени. Анимация объектов на экране, обмен данными, функции, созданные пользователем и встроенные в систему обрабатываются своими независимыми потоками с четким распределением приоритетов. При этом многопоточность реализована по схеме с вытеснением, когда задача с большим приоритетом вытесняет задачу с меньшим, а внутри класса задач с одним приоритетом вытеснение производится по круговой схеме (round robin) путем выделения квантов процессорного времени. Для искушенных пользователей предусмотрена возможность непосредственного использования объектов синхронизации(!) работы потоков и управления их созданием и завершением.

Что думает оператор про *SCADA*-систему, когда он нажимает на кнопку включения мотора (любого ответственного механизма), но ничего не происходит? А если сигнал тревоги (alarm), сгенерированный на объекте, поступает к оператору не сразу, а через 5-10 секунд? Адекватная работа с сигналами из реальной аппаратуры, особенно когда они составляют замкнутую цепь управления — задача номер 1 для любого SCADA-пакета. К сожалению, большинство SCADA-пакетов очень плохо масштабируются при изменении нагрузки, т. е. при изменении количества сигналов и параметров (окон, элементов управления, количества выполняемых функций и т. д.). Если мы будем сравнивать скорость отклика системы при минимальном числе входных параметров, то получим весьма схожие результаты, но принципиально различные при увеличении нагрузки. Основной причиной этого является отсутствие в большинстве SCADA-пакетов ядра реального времени, так как это сделано в Citect. Именно поэтому теоретическим и практическим пределом для обычных SCADA-пакетов является 10000-30000 параметров (tags), в то время как Citect может работать с числом переменных до 500000. И эта величина не является теоретическим пределом, скорее практическим. Конечно, такие большие системы реализуются достаточно редко, но даже и в диапазоне 5000-10000 параметров Citect имеет бесспорные и ощутимые преимущества. SCADA-система Citect работает с разрешением по времени, равным одной миллисекунде. Это относится как к работе с данными, так и к обновлению графиков реального времени (trends) и других объектов.

1.13. Средства визуализации

Средства визуализации — одно из базовых свойств SCADA-систем. В каждой из них существует графический объектно-ориентированный редактор с определенным набором анимационных функций. Используемая векторная графика дает возможность осуществлять широкий круг операций над выбранным объектом. Объекты могут быть простыми (линии, прямоугольники, текстовые объекты и т. д.) и сложные. Возможности агрегирования сложных объектов в разных SCADA-системах различны. Все SCADA-системы включают библиотеки стандартных графических символов, библиотеки сложных графических объектов, обладают целым рядом других стандартных возможностей. Но, тем не менее, каждая SCADA-система по-своему уникальна и, несмотря на поддержание стандартных функций, обладает присущими только ей особенностями. При рассмотрении графических возможностей

SCADA-систем InTouch и Citect предлагается обратить внимание не только на возможности инструментов по созданию графических объектов, но и на другие предоставляемые пользователю услуги, облегчающие и ускоряющие процесс разработки приложений.

1.14. Состояние тревоги и события

Состояние тревоги, в дальнейшем Alarm — это некоторое сообщение, предупреждающее оператора о возникновении определенной ситуации, которая может привести к серьезным последствиям, и потому требующее его внимания, а часто и вмешательства. А принял ли оператор сообщение об Alarm? Чтобы снять эти сомнения, в системах управления принято различать неподтвержденные и подтвержденные Alarm. Alarm называется подтвержденным после того, как оператор отреагировал на сообщение об Alarm. До этого Alarm оставался в состоянии неподтвержденного. Наряду с Alarm в SCADA-системах существует понятие события. События представляют собой обычные статусные сообщения системы и не требуют реакции оператора. Обычно событие генерируется при возникновении в системе определенных условий (типа регистрации оператора в системе). От эффективности подсистемы Alarm зависит скорость идентификации неисправности, возникшей в системе, или идентификации технологического параметра, вышедшего за установленные регламентом границы. Быстродействие и надежность этой подсистемы могут существенно сократить время простоя технологического оборудования. Например, если оператор не получит вовремя информацию о том, что двигатель насоса перегрелся, это может привести в лучшем случае к выходу насоса из строя, а то и к крупной аварии. Причины, вызывающие состояние Alarm, могут быть самыми разными. Неисправность может возникнуть в самой SCADA-системе, в контроллерах, каналах связи, в технологическом оборудовании. Может выйти из строя датчик или нарушатся его метрологические характеристики. Параметры технологического процесса могут выйти за границы, установленные регламентом и т. д.

1.15. Отображение и архивирование данных

Графическое представление значений технологических параметров во времени способствует лучшему пониманию динамики технологического процесса предприятия. Поэтому подсистема создания трендов (графиков) и хранения информации о параметрах с целью ее дальнейшего анализа и использования для управления является неотъемлемой частью любой SCADA-системы. Тренды реального времени отображают динамические изменения параметра в текущем времени. При появлении нового значения параметра в окне тренда происходит прокрутка графика справа налево. Таким образом, текущее значение параметра выводится всегда в правой части окна. Тренды становятся архивными после того, как данные будут записаны на диск, и можно будет использовать режим прокрутки предыдущих значений назад с целью посмотреть прошлые значения. Отображаемые

данные тренда в таком режиме будут неподвижны, и будут отображаться только за определенный период.

1.16. Стоимость системы

Стоимость освоения системы. Процедура освоения SCADA-систем достаточно проста с точки зрения программиста и не требует длительного времени, поэтому эти затраты относительно невелики. Основной составляющей стоимости является оплата труда программистов, осуществляющих эту работу.

Стоимость сопровождения или "стоимость владения". Эта составляющая обычно наиболее "скрыта от глаз покупателя" и зависит от многих факторов. Например:

- стоимость "риска" покупки, который определяется такими параметрами как рыночная надёжность фирмы-дистрибутора инструментального пакета (трудно говорить о надёжности фирмы, если её, скажем, штат 1-5 человек), рыночная стабильность фирмы-изготовителя продукта;
- стоимость коммуникаций с фирмой-поставщиком;
- "время реакции" поставщика на проблемы покупателя;
- наличие реального прикладного опыта и хорошего знания поставляемого продукта специалистами фирмы-поставщика, наличие в принципе у поставщика специалистов по продукту;
- степень открытости, адаптируемости и модернизируемости продукта.

Эти и многие другие факторы, влияющие на "стоимость владения" необходимо учитывать при выборе системы. Можно подчеркнуть, что концентрация разработчиков SCADA-систем на поле Windows NT способствует снижению "стоимости владения" пользователем этими продуктами.

Стоимость разработки прикладных систем. Стоимость, связанная с трудозатратами на разработку прикладных программ при использовании SCADA-систем, существенно уменьшается по сравнению с использованием традиционного программирования. В качестве примера можно привести работы по созданию системы информационного обслуживания в CERN, которая содержала несколько десятков узлов. Эти работы вместе с отладкой заняли около трех месяцев и выполнялись с помощью системы FactoryLink. Но следует учесть, что вышесказанное относилось к стоимости разработки системы. Необходимо отметить, что стоимость изготовления системы составляет обычно 30-50% от стоимости разработки системы.

Срок окупаемости SCADA–системы. Чтобы оценить время окупаемости SCADA-системы необходимо учесть множество факторов, включая количество проектов, реализуемых на основе этой системы, стоимость этих проектов и т. д. Ориентировочно, если речь идет о бизнесе системного интегратора, реализация 2-3 проектов при приобретении системы разработки SCADA окупает ее. Одним из важных свойств SCADA всегда была открытость. Сейчас открытость дополняется новыми средствами передачи данных между процессами — OLE, OPC, встраиваемыми программными объектами (ActiveX).

1.17. SCADA-продукты на российском рынке

В [8] приведены характеристики SCADA-систем, представленных на российском рынке. Так как эти характеристики получены в результате анкетирования компаний — поставщиков SCADA-продуктов, то они не претендуют на исчерпывающий перечень SCADA-систем. Тем не менее, эти характеристики являются весьма информативными для выбора SCADA-системы.

Анкета, использованная при получении указанных сведений, составлена с учетом мнения ряда компаний о том, какие характеристики SCADA-систем важны при их оценке и выборе. Анкета включает около 100 критериев, распределенных по *четырем категориям* и *основным группам* внутри каждой категории. Категории перечислены в соответствии с их *приоритетностью* (с точки зрения выбора SCADA-системы) и соответствуют основным этапам жизненного цикла SCADA-систем.

1.17.1. Базовые свойства

Базовые свойства SCADA-системы определяют идеологическую и функциональную основу АСУ ТП.

- *Открытость*: интерфейс прикладных программ API (Application Program Interface) к базе данных реального времени; стандартный интерфейс ODBC к реляционным БД и архивам; управление программируемыми логическими контроллерами PLC через DDE, DLL, OLE и OPC; обмен с приложениями через API, DLL, DDE, OLE, COM; API-доступ к архивам и др.
- *Поддержка контроллеров*: количество поддерживаемых контроллеров; наличие промышленных шин Profibus, Canbus, Foundation fieldbus, Modbus и др.; связь с контроллерами напрямую, по собственному протоколу, по промышленной шине или протоколу третьей фирмы; доступ к контроллерам — опрос, получение меток времени, инициативная передача, подписка; наличие средств разработки драйверов; среднее время разработки нового драйвера и др.
- *Архивация и тренды*: вывод в тренды в РВ; запоминание информации во временном хранилище — буфере, файле — для последующей перезаписи в архив; непосредственный вывод в архивные тренды; возможность просмотра трендов "через лупу"; максимальное число кривых в тренде; максимальное число трендов в окне; выбор и/или отслеживание параметров с использованием предустановки или онлайновый; одновременный вывод трендов в РВ и архивных трендов; возможность вывода значения параметра в точке, указанной курсором; использование для работы с трендами модулей или объектов ActiveX и др.
- *Алармы*: количество уровней приоритетов; возможность группирования алармов; онлайновая фильтрация; маскирование алармов по нижним/верхним границам изменения параметров; наличие алармов по скорости изменения параметров или по отклонению параметров от уставки; автоматическое выполнение определенных действий при возникновении оговоренных алармов; генерация E-mail сообщений и др.

- *Возможности HMI*: наличие объектно-ориентированного графического редактора; библиотека стандартных графических символов; техника "drug-and-drop"; библиотека сложных графических объектов; стандартные возможности оконного редактирования — масштабирование, изменение размеров, прокрутка, манипуляции с "пером" и т. п.; возможность работы с трендами; возможность навигации по страницам изображения; анимация, мультимедиа и пр.
- *WEB-технологии:* Java, ActiveX; мониторинг; управление и пр.
- *Система отчетов:* встроенный генератор отчетов; метод "вырезать/вставить"; печать, архивация отчетов и др.
- *Масштабируемость:* по базе данных РВ; по количеству узлов в сети.
- *Опции/автоматизация:* управление технологическим процессом на основе статистической обработки полученной информации; сохранение конкретной конфигурации системы в файле с последующей ее перезагрузкой; планирование заданий; автообработка событий при изменении параметра и др.
- *Поддержка ОС:* Windows 95, Windows NT, AIX, UNIX, VMS, QNX, DOS.
- *Защита доступа:* через пароль; собственная система защиты; уровни доступа — чтение, запись, приоритеты, классы, авторизация.
- *Резервирование:* узла, сервера ввода/вывода, трендов, алармов, контроль резервирования.
- *Стоимость*.
- *Рыночная политика:* ценовая и лицензионная гибкость.

1.17.2. Возможности среды разработки

Возможности среды разработки SCADA-системы определяется общими средствами расширения инструментальных возможностей *SCADA*-пакетов.

- Разработка в on-line.
- Тестирование/отладка в on-line.
- Возможность симуляции.
- Разработка распределенных приложений.
- Язык программирования/сценариев.
- Графические языки.
- Возможность одновременной разработки нескольких проектов.
- Многопользовательская разработка.
- Наличие пакетов разработчика (toolkit), позволяющих создавать драйверы ввода-вывода, графические объекты и т. п.
- Пакет комплексной автоматизации. Наличие пакета комплексной автоматизации означает вхождение SCADA-системы в интегрированную программную систему, предназначенную для автоматизации управления в единой информационной

среде на основных производственных уровнях – от уровня непосредственного управления (уровень контроллеров) до уровня автоматизированной системы управления предприятием (АСУП).

1.17.3. Возможности конфигурирования системы

- *Может ли любой узел быть* сервером ввода-вывода, сервером трендов, сервером алармов.
- *Может ли любой узел быть* клиентом, взаимодействующим с любым сервером.
- *Может ли любой узел быть* только мониторинговой подсистемой.
- *Конфигурируемость по числу точек ввода-вывода.*
- *Максимальное число точек ввода-вывода.*

1.17.4. Характеристики сопровождения/эксплуатации

Характеристики сопровождения/эксплуатации SCADA-систем определяют эксплутационные возможности и услуги пользователям при сопровождении систем распространителями.

- Локализация и мультиязыковая поддержка.
- Рыночный стаж.
- Простота использования.
- Время обучения технического персонала.
- Наличие документации на русском языке.
- Наличие продуктов других фирм, поддерживающих данный SCADA-пакет.
- Количество инсталляций.
- Наличие обучающих курсов.
- Время изучения продукта.

Перечень SCADA-систем на российском рынке, фирм-производителей и характеристик SCADA-систем по состоянию на 1999 год имеется в [8]. Еще раз обращаем внимание на то, что эти данные не являются исчерпывающими, так как получены путем анкетирования фирм-производителей. При этом анкетировались не все фирмы (не все они были известны при анкетировании) и не все фирмы откликнулись на просьбу заполнить анкеты. К тому же, результаты анкетирования к настоящему времени несколько устарели. Тем не менее, эти данные очень важны и существенны — их изучение принесет проектировщику большую пользу.

1.17. Заключение

По функциональным возможностям все рассмотренные SCADA-системы в целом сравнимы. Технология программирования близка к интуитивному восприятию автоматизируемого процесса. Мощное объектно-ориентированное

программирование, используемое в большинстве этих пакетов, делает эти продукты легкими в освоении и доступным для широкого круга пользователей. Все системы можно считать в той или иной степени открытыми, обеспечивающими возможность дополнения функциями собственной разработки, имеющими открытый протокол для разработки собственных драйверов, развитую сетевую поддержку, возможность включения ActiveX-объектов и доступ к стандартным базам данных. Важной особенностью всех SCADA-систем является поддержка OPC-спецификации. Это делает их привлекательными. Построение прикладной системы на основе любой из рассмотренных SCADA-систем резко сокращает набор необходимых знаний в области классического программирования, позволяя концентрировать усилия по освоению знаний в прикладной области. У разработчиков SCADA-систем на платформе Windows NT/2K/XP появилась возможность использовать расширение реального времени (RTX). Следует отметить тенденции включения SCADA-систем в системы комплексной автоматизации предприятия. Это обеспечивает точную, своевременную информацию на каждом уровне производства. Применение в SCADA-системах новых технологий, разработка инструментальных средств комплексной автоматизации предприятия свидетельствуют о стремлении и возможности фирм-разработчиков постоянно совершенствовать свои продукты, что является немаловажным фактором при выборе инструментального средства, даже если не все его технологические решения в ближайшее время будут использованы Вами. Так как общее поле деятельности ведущих компаний-производителей описываемых инструментальных систем сегодня концентрируется в области MS Windows, а общие технические возможности систем достаточно близки, то главный упор делается на качество технической поддержки, на качество обучения пользователей, на концентрацию и качество дополнительных комплексных услуг по освоению и внедрению конечной системы управления. Другими словами, упор делается на сокращение издержек системных интеграторов и конечных пользователей на инжиниринг и менеджмент своих проектов, на уменьшение стоимости сопровождения конечной системы. Именно эти показатели сегодня, в основном, влияют на рейтинг и рыночный успех той или иной SCADA-системы. Пожалуй, эти показатели даже более важны, чем абсолютные стоимостные характеристики SCADA-систем.

На основе сравнительного анализа SCADA-систем можно выделить следующую представительную группу SCADA-систем:

- Intouch (одна из наиболее мощных SCADA-систем);
- Citect (одна из наиболее мощных SCADA-систем);
- TRACE MODE (SCADA-система среднего класса отечественной разработки и производства);
- GeniDAQ (простая и недорогая SCADA-система).

SCADA-система Intouch (компания Wonderware, США) насчитывает более 100000 инсталляций во всем мире, причем 3000 из них выполнены компанией KLINKMANN (компания обеспечивает техническую поддержку и документацию на русском языке). В KLINKMANN разработано более 100 драйверов промышленного оборудования,

включая систему для беспроводной связи и управления на основе стандарта GSM, для применения с программным обеспечением Wonderware.

SCADA-система Vijeo Citect (компания Citect Pty. Ltd, Австралия) занимает более 6% мирового рынка SCADA-систем и более 60% австралийского рынка. Эта система в наибольшей степени отвечает свойству открытости в возможности подключения практически к любым контроллерам. Количество подключаемых точек ввода-вывода больше, чем в SCADA-системе Intouch, но Vijeo Citect проще в освоении. На российском рынке обеспечивается техническая поддержка Vijeo Citect, имеется система центров обучения и документация на русском языке. Демонстрационная версия Vijeo Citect имеет ограничения, несущественные для использования в учебном процессе (шесть часов непрерывной работы, при работе с внешними каналами ввода-вывода — 10 минут).

SCADA-система TRACE MODE (компания AdAstra Research Group, Ltd, Москва) представляет собой 32-разрядную SCADA и SoftLogic-систему для разработки АСУ ТП мирового класса. С более чем 3500 инсталляциями TRACE MODE является одной из наиболее покупаемых в России для разработки крупных диспетчерских комплексов и программирования PC-контроллеров. Система поддерживает более 300 марок контроллеров и УСО. Благодаря новым технологиям сквозного программирования операторских станций и контроллеров, а также автоматического построения проекта, мощь TRACE MODE теперь сочетается с простотой разработки. TRACE MODE имеет самую современную архитектуру, основанную на DCOM. Поддерживаются технологии OPC (сервер/клиент), ActiveX, Internet/Intranet, SQL/ODBC, DDE.

SCADA-система GeniDAQ является самой простой и недорогой SCADA-системой, наиболее простой в изучении и освоении. Ее демонстрационная версия имеет ограничения, несущественные для использования в учебном процессе (два часа непрерывной работы, ограниченное число каналов ввода-вывода и др.). С другой стороны, *GeniDAQ* является вполне представительной *SCADA*-системой.

Из сказанного следует, что для применения в учебном процессе перспективным является изучение SCADA-систем GeniDAQ и Vijeo Citect, причем именно в такой последовательности.

Глава 2. OPC — промышленный стандарт и средство интеграции компонентов в промышленной автоматизации

Ранее указывалось, что информационная автоматизированная система управления промышленным предприятием имеет открытую архитектуру (рис. 2.1), включающую следующие неотъемлемые уровни [1]:

- Нижний уровень, включающий полевые шины и отдельные контроллеры (Field Management), представляющие собой "интеллектуальные" технологические устройства: датчики, контролеры, механизмы и т. д. Поток информации от нижнего уровня должен быть предоставлен пользователю и всем приложениям верхнего уровня, использующим их, посредством цифровых коммуникационных протоколов связи. При этом в системе не должно возникать проблем несовместимости.
- Уровень АСУ ТП (Process Management) — уровень работы SCADA-систем для сбора данных и диспетчерского управления технологическими процессами на производстве. Этот уровень обеспечивает вторичную обработку данных, которые получены с нижнего уровня, сохранение данных и их доступность приложениям и пользователям верхнего уровня.
- Уровень автоматизированной системы управления предприятия (АСУП) — уровень приложений управления ресурсами предприятия (Business Management). Информация с уровня АСУ ТП должна доступной для уровня АСУП, т. е. доступ к данной информации со стороны прикладных программ также не должен вызывать проблемы несовместимости.

Для обеспечения совместимости между уровнями и создания эффективной интегрированной системы управления предприятием системный интегратор или разработчик АСУ ТП должен извлекать данные технологического процесса в реальном времени с самого нижнего уровня и обеспечивать "прозрачный" путь получаемым данным к самым верхним уровням. Чтобы получить систему, отвечающую всем потребностям заказчика, системному интегратору или разработчику необходимо использовать инструментальные средства управления различных уровней — SCADA-пакеты, базы данных, электронные таблицы. Ключ к этому — открытая и эффективная коммуникационная архитектура взаимодействия между приложениями, которую предлагает стандарт OPC (OLE for Process Control).

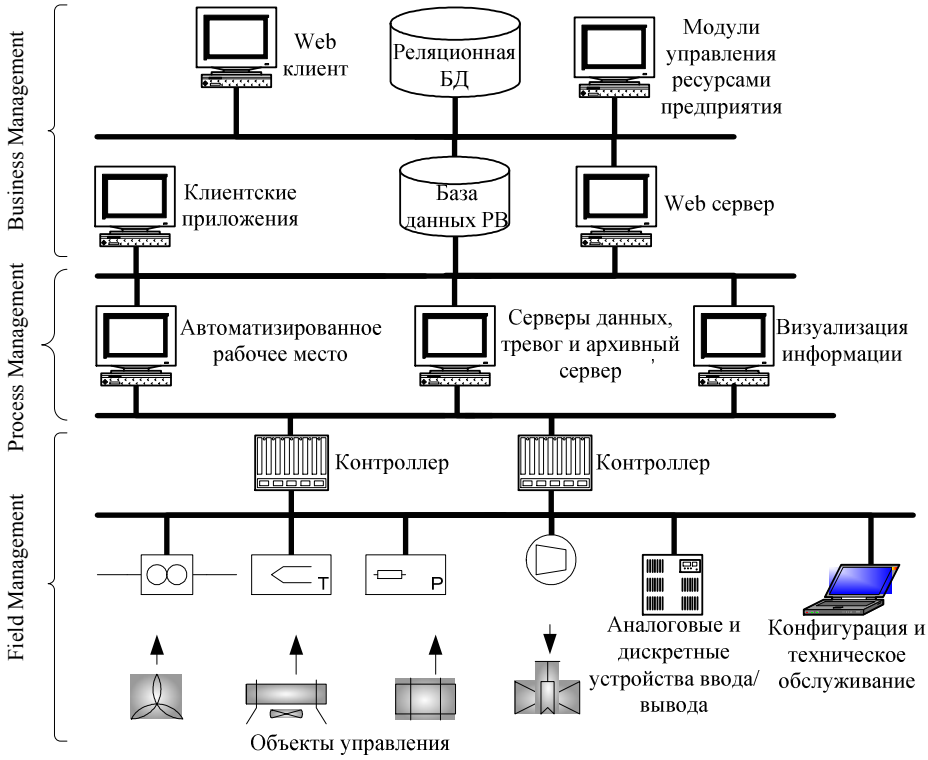

Рис. 2.1. Архитектура информационной автоматизированной системы управления промышленным предприятием

2.1. Что такое OPC?

Попытаемся определить, что такое OPC. *OPC* — технология связывания и внедрения объектов для систем промышленной автоматизации, предназначенная для обеспечения универсального механизма обмена данными между датчиками, исполнительными механизмами, контроллерами, устройствами связи с объектом и системами представления технологической информации, оперативного диспетчерского управления, а также системами управления базами данных.

На сегодняшний день технология OPC в определенной степени реализована, но продолжает развиваться. Консорциум OPC Foundation, объединяющий более 300 производителей средств промышленной автоматизации, разрабатывает все аспекты, связанные с взаимодействием между компонентами программного обеспечения, между программным обеспечением и между системами типа SCADA и технологическим оборудованием. OPC Foundation привлекает к разработке спецификаций обмена ведущих производителей оборудования и систем автоматизации, которые стараются максимально учесть свой опыт и предоставить абсолютно всё необходимое тому, кто будет использовать OPC. Существуют много

спецификаций OPC — Data Access (доступ к данным реального времени), Alarms & Events (обработка тревог и событий), Historical Data Access (доступ к архивным данным) и т. д. Поэтому можно определить OPC как стандарт взаимодействия между программными компонентами сбора данных и управления, 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, осуществлять доступ к данным, зарегистрированным в архивах. Эти приложения могут располагаться как на одном компьютере, так и быть распределенными в сети. При этом, независимо от фирмы-поставщика, стандарт OPC, признанный и поддерживаемый всеми ведущими фирмами-производителями SCADA-систем и оборудования, обеспечит их совместное функционирование.

Популярный класс OPC-приложений представляют собой OPC-серверы конкретных аппаратных устройств, обеспечивающие предоставление информации о состоянии параметров многочисленных устройств технологического процесса, полученной OPC-приложением, OPC-клиентам на локальном компьютере или в компьютерной сети. OPC-сервер создает свою абстракцию устройств (это загружаемый модуль, предоставляющий интерфейс нижнего уровня с оборудованием, на котором исполняется OPC-сервер). OPC-сервер скрывает аппаратно-зависимые детали, такие как интерфейс ввода/вывода, контролеры прерываний и механизм коммуникации в многопроцессорных системах — любые функции, зависящие от конкретной архитектуры и машин, позволяя любому OPC-клиенту записывать и считывать данные с устройства. Устройство, для которого есть OPC-сервер, может использоваться вместе с любой современной SCADA-системой.

Современные SCADA-системы не ограничивают выбор аппаратных средств нижнего уровня, так как SCADA-пакеты предоставляют большой набор драйверов или серверов ввода/вывода (в т. ч. OPC-серверы) и имеют хорошо развитые средства создания собственных программных модулей или драйверов для новых и нестандартных устройств нижнего уровня. Производители аппаратных средств нижнего уровня, используя спецификации OPC, имеют возможность разрабатывать свой OPC-сервер для обеспечения доступа к данным реального времени и передачи данных приложениям — клиентам различных производителей программного обеспечения для промышленной автоматизации (в т. ч. SCADA-системам).

2.2. DCOM и OPC-приложения

Технология OPC основана на модели распределенных компонентных объектов *DCOM* — Distributed COM. Модель представляет собой сетевое расширение COM или распределённую COM (Component Object Model). Технология COM — Моделей Компонентных Объектов — создана первоначально Microsoft для совместного использования различных офисных приложений в Windows, например, электронных таблиц Excel, редактора Word и т. п.

COM поддерживает модель "клиент-сервер" [9]. Объекты, называемые серверами, предоставляют некие функции в распоряжение объектов, называемых клиентами.

Серверы всегда являются COM-объектами, т. е. объектами, которые подчиняются спецификации COM. С другой стороны, клиенты могут быть COM-объектами или не быть таковыми. Это значит, что некоторые объекты могут быть простыми объектами C++, приложениями Visual Basic и т. п. Каждый COM-объект существует внутри конкретного сервера. Этот сервер содержит программный код реализации операций, а также данные активного COM-объекта. Один сервер может обеспечивать несколько объектов и даже несколько COM-классов. Используются три типа серверов (рис. 2.2):

- внутризадачный сервер (in-process) — объекты находятся в динамически подключаемой библиотеке и, следовательно, выполняются в том же процессе, что и клиент;
- локальный сервер (local) — объекты находятся в отдельном процессе, выполняются в том же компьютере, что и клиент;
- удаленный сервер (remote) — объекты находятся в DLL или в отдельном процессе, который расположен на удаленном от клиента компьютере.

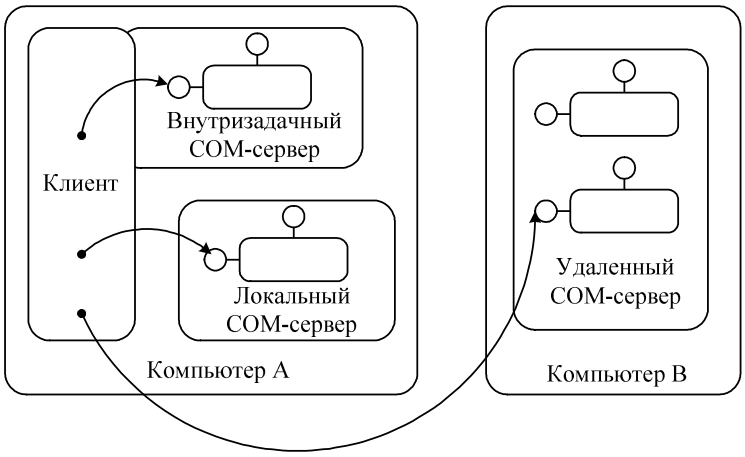

Рис. 2.2. Разновидности OPC-серверов

Клиенты и локальные серверы находятся в различных адресных пространствах. OPC берет на себя заботу о передаче данных между ними. В случае внутризадачных серверов, родительский процесс загружает DDL-файл, содержащий COM-сервер. Это означает, что внутризадачный сервер находится в том же адресном пространстве, что и вызвавший его процесс. В случае DCOM вызов любой функции объекта перехватывается специальным агентом-посредником, называемым Proxy Manager или Stub Manager. Proxy Manager получает все запросы на транспортировку и определяет, стандартные ли данные запрашивает или предлагает клиент. Если это так, Proxy Manager передает запрос подходящему Proxy-объекту. Подходящие Proxy и Stub — это объекты, предназначенные для транспортировки данного интерфейса и/или типа данных. Аналогичный процесс происходит на стороне Stub Manager. Встроенные Proxy и Stub называются транспортировщиками интерфейса.

Популярным вариантом OPC-клиента является SCADA-система. С помощью OPC-серверов SCADA-система может обращаться к любому аппаратному устройству (рис. 2.3) путем вызова определенных функций OPC-сервера, подписываться на получение определенных данных какого-либо канала или устройства с требуемой частотой. В свою очередь, OPC-сервер опрашивает аппаратное устройство, вызывает известные функции клиента, подтверждает клиенту получение данных и посылает требуемые клиенту данные.

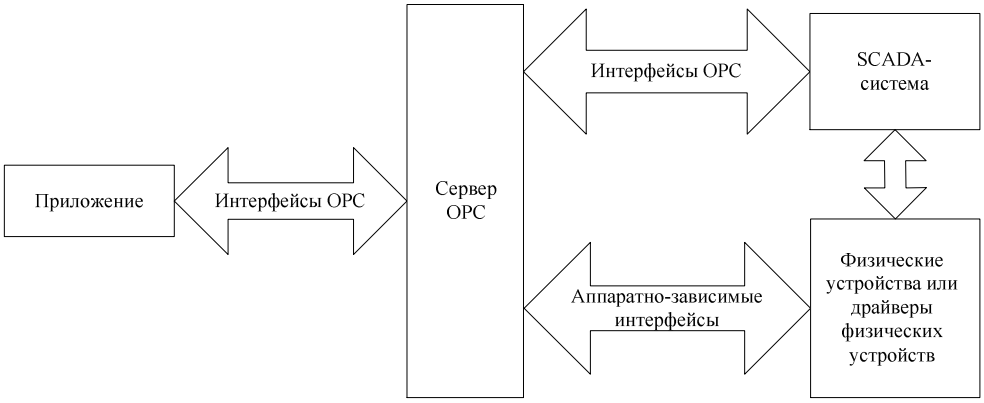

Рис. 2.3

Обмен данными между OPC-клиентом и OPC-сервером может быть реализован в трех режимах — синхронного чтения/записи, асинхронного чтения/записи и режиме подписки (только чтение).

При синхронном чтении/записи клиент посылает серверу запрос со списком интересующих его переменных и ждёт, когда сервер его выполнит.

При асинхронном чтении/записи клиент посылает серверу запрос, а сам продолжает работать. Когда сервер выполнил запрос, клиент получает уведомление. Этот режим обладает большей гибкостью и рекомендуется в большинстве случаев, т. к. он минимизирует затраты процессорного времени и сетевых ресурсов, особенно при передаче большого количества данных.

В случае *подписки* клиент передаёт серверу список интересующих его переменных, а сервер присылает клиенту информацию об изменившихся переменных из этого списка. При этом сервер будет передавать клиенту информацию только тогда, когда данные изменились, причём эти данные передаются единым блоком. Эти меры позволяют существенно ускорить обмен данными, особенно если речь идёт о взаимодействии через сеть.

Кроме механизмов обмена данными, при реализации OPC-серверов необходимо учитывать еще один важный аспект — источник получаемых данных. В зависимости от реализации OPC-сервера, клиенты могут свободно получать данные от разных источников — непосредственно от устройства или из кэша OPC-сервера.

2.3. Почему OPC?

Ранее уже упоминалось, что основная цель OPC-стандарта заключается в определении *механизма доступа к данным любого устройства из различных приложений*. OPC позволяет производителям оборудования стандартизовать свои продукты, обеспечивает подключение оборудования различных производителей, в т. ч. подключение к SCADA-системам.

Идеальной выглядела бы следующая картина. Все в мире признают OPC своим стандартом. При этом все поставщики оборудования снабжают свои продукты OPC-серверами. Все поставщики программ для систем управления делают свои продукты OPC-клиентами — а все производители операционных систем поддерживают в своих ОС технологии COM/DCOM, а также предоставляют сервисный инструментарий. И при этом все это делают на высоком профессиональном уровне и очень грамотно рассказывают сборщикам систем, как это всё собирать и конфигурировать. Тогда пользователи будут иметь широкую возможность выбора оптимальных для своей системы компонентов, а не создавать систему управления "с нуля" и не выбрасывать свой имеющий контроллер. Таким образом, по образному выражению Теркеля [10], *OPC — это свобода выбора.*

Преимущество применения стандарта *OPC* с точки зрения интеграции достаточно прозрачно. С применением OPC-стандарта появляется возможность *интегрировать в единую систему неоднородные узлы*.

OPC-стандарт можно рассматривать как “универсальный переходник”. Если мы заменяем какой-нибудь компонент клиентского приложения, то нет необходимости корректировать серверное приложение, поскольку COM обеспечивает эффективное управление изменением программы. Если мы хотим добавить в систему новое оборудование, то достаточно его включить в OPC-сервер. При этом клиентские программы не требуют изменений, в т. ч. для *SCADA*-систем.

В настоящее время, консорциум OPC Foundation набирает силу в разработке открытых промышленных стандартов на основе OPC-стандарта на базе PC и операционных систем Microsoft. Как указывалось ранее, в состав OPC Foundation входят более 300 членов, среди которых практически все мировые ведущие производители технологического оборудования, систем автоматизированного управления и программного обеспечения. Членами организации являются, например, фирмы Iconics Inc. (США), Wonderware (США), Citect (Австралия), AdAstra (Россия), Siemens (Германия), Rockwell Software (США), Intellution (США), Indusoft Russia (Россия), Fastwel Inc. (Россия), ABB Automation (США), Fieldbus Foundation (США), Toshiba Corp. (Япония), Hitachi (Япония), National Instruments (США) и др. Организация пытается охватить все аспекты, связанные с взаимодействием с технологическим оборудованием. Ведущие производители стараются максимально учесть свой опыт и предоставить абсолютно всё необходимое тому, кто будет использовать OPC. Этот факт показывает *большой авторитет OPC-технологии*. Это перспективная технология для использования в развитии автоматизированных систем управления.

2.4. Заключение

OPC-технология содержит стандарты, обеспечивающие взаимодействие программных средств промышленной автоматизации. В технологию заложены богатые возможности, которые дают руководителям предприятия возможность интеграции разнородных систем и обеспечивают разработчикам свободу выбора с применением OPC-спецификаций, позволяют не задумываться по поводу поддержки аппаратуры завтрашнего дня. Консорциум OPC Foundation создал интерфейс, который объединяет все компоненты систем автоматизации и визуализации ведущих производителей. OPC станет технологией, которую не сможет проигнорировать ни один производитель средств промышленной автоматизации.

Глава 3. Краткий обзор SCADA-системы GeniDAQ

Рассмотрим основные отличительные особенности SCADA-системы GeniDAQ и ее место на рынке SCADA-систем.

3.1. Почему мы рассматриваем GeniDAQ?

Производитель SCADA-системы GeniDAQ — американское отделение фирмы Advantech, которая является известным производителем компьютеров и электроники для промышленной автоматизации. Отличительной чертой SCADA-системы GeniDAQ является то, что Вы можете получить полноценную SCADA-систему (Runtime-версию) для Windows всего за несколько сотен долларов. Даже "доморощенные" пакеты с неясным будущим за такую цену уже давно никто не предлагает. А почувствовать себя честным пользователем лицензионно чистой копии добротно сделанного продукта фирмы с мировым именем почти даром — просто приятно. Секрет низкой цены в этом случае раскрывается просто — SCADA-система GeniDAQ предназначена для программной поддержки аппаратуры фирмы Advantech и, в первую очередь, содержит драйверы именно для нее. Вместе с тем, никто не запрещает использовать SCADA-систему GeniDAQ и с оборудованием других изготовителей. Последнее объясняется тем, что SCADA-пакет GeniDAQ полнофункционально поддерживает OPC-спецификацию. С этой точки зрения, SCADA-систему GeniDAQ можно рассматривать в качестве OPC-клиента. Большинство ведущих производителей оборудования в мире поставляет на рынок свои аппаратные средства вместе с соответствующими OPC-серверами. С помощью OPC-технологии можно легко создавать приложения, которые работают не только с устройствами фирмы Advantech, но и с аппаратурой любых других фирм.

Другой важной отличительной чертой SCADA-пакета GeniDAQ является прекрасно продуманный интерфейс пользователя. Намеренно сократив число "степеней свободы" в инструментальной части пакета и написав прекрасную справку (Help, помощь), авторы пакета создали уникальный по простоте освоения программный продукт. В худшем случае, уже через несколько часов знакомства с пакетом, Вы сможете написать что-нибудь содержательное и работающее. Если же это окажется не так, то Вам стоит задуматься о смене профессиональной ориентации. Прекрасно выполненная демоверсия пакета одновременно служит и хорошим учебным пособием — разобравшись в работе десятка примеров несложных программ, можно спокойно принять решение о приобретении полной версии [11].

3.2. Системная архитектура GeniDAQ

SCADA-пакет GeniDAQ имеет открытую архитектуру[5, 12], что обеспечивает наибольшую гибкость и эффективность (рис. 3.1).

Центр обработки данных является основным информационным хранилищем в GeniDAQ и представляет собой набор библиотек динамической компоновки (DLL), которые предназначены для размещения и хранения всех данных, связанных с работой приложения под управлением исполнительной среды GeniDAQ. Результаты выполнения всех функциональных блоков и данные, вводимые пользователем с помощью элементов управления в экранных формах, передаются в центр обработки данных.

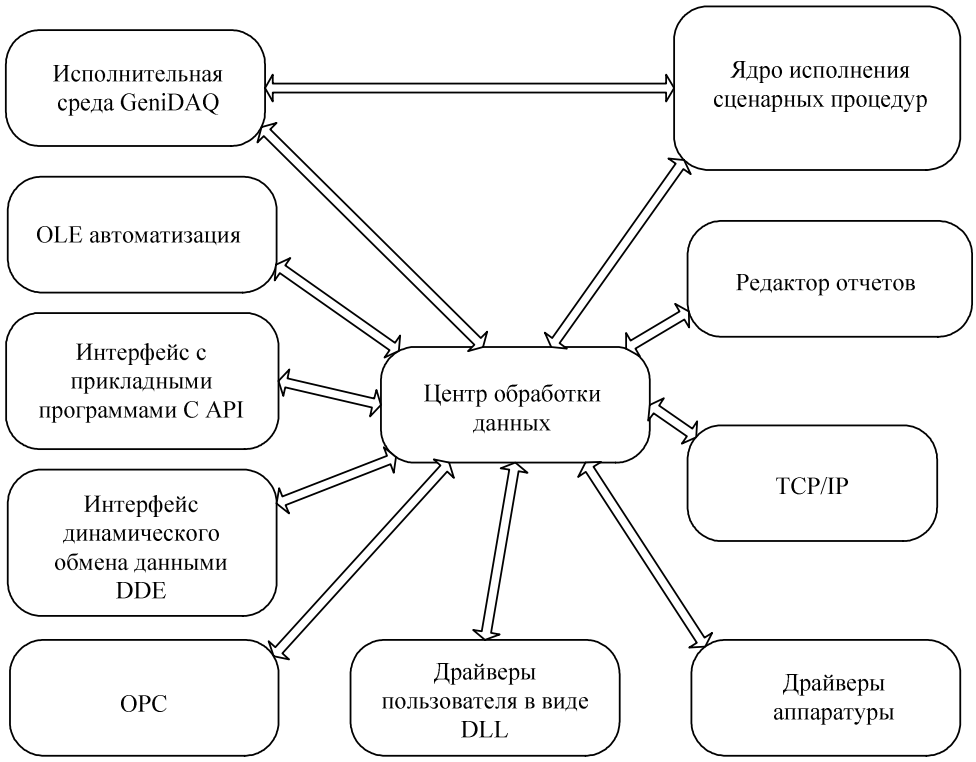

Рис. 3.1. Системная архитектура SCADA-пакета GeniDAQ

Центр обработки данных располагается в физической (не виртуальной) оперативной памяти для обеспечения наиболее быстрого сохранения и извлечения данных. Ключом для поиска информационного объекта в центре обработки данных является имя тега. Для увеличения скорости доступа к информации в центре обработки данных могут быть использованы эффективные алгоритмы, реализация которых имеется в библиотеке классов Microsoft Foundation Classes (MFC). Каждый объект в центре обработки данных имеет имя тега, идентификатор функционального блока, период обновления информации, счетчик доступа и значение, связанное с объектом. Другие приложения Windows могут использовать имя тега для получения идентификатора функционального блока из центра обработки данных. Идентификатор функционального блока, наряду с именем тега, также может

применяться в качестве ключа для поиска объекта в центре обработки данных. Центр обработки данных поддерживает ряд способов взаимодействия пакета с другими приложениями Windows:

- интерфейс прикладного программирования C API;
- интерфейс динамического обмена данными DDE;
- интерфейс связывания и внедрения (встраивания) объектов OLE Automation;
- OPC-интерфейс для подключения к OPC-серверам;
- интерфейс TCP/IP для организации обмена данными в сети.

Интерфейс прикладного программирования C API представляет собой наиболее эффективное средство взаимодействия между SCADA-пакетом GeniDAQ и другими приложениями, поскольку с его помощью осуществляется взаимодействие между программными компонентами GeniDAQ.

Интерфейс связывания и внедрения объектов OLE Automation предназначен для организации взаимодействия между GeniDAQ и другими приложениями Windows, поддерживающими механизм OLE. Интерфейс OLE Automation является основой *интерфейса OPC*, который можно рассматривать как дальнейшее развитие OLE Automation. OPC-интерфейс используется для обмена данными между GeniDAQ и OPC-серверами для управления и сбора данных в системах промышленной автоматизации.

С помощью *интерфейса TCP/IP* данные о технологическом процессе можно наблюдать и регистрировать в режиме реального времени в любой точке локальной сети.

Исполнительная среда GeniDAQ обеспечивает возможность переключения нескольких задач в процессе исполнения, а также позволяет осуществлять однократный вызов задач. Этим объясняется способность GeniDAQ одновременно исполнять несколько задач. Поскольку один вызов задачи является наименьшей единицей исполнения, пользователи получают возможность разрабатывать приложение таким образом, чтобы логически и функционально связанные блоки группировались в одной и той же задаче. Использование основного сценария (подробнее см. далее) для управления исполнением задач позволяет вызывать те или иные задачи только при выполнении определенных условий. В результате появляется возможность разработки сложных систем путем объединения простых задач. Простое приложение, содержащее одну задачу, может исполняться без использования сценарных процедур. Имеется возможность определения приоритетов выполнения задач для оптимизации работы системы.

Ядро подсистемы исполнения сценарных процедур пакета GeniDAQ представляет собой набор библиотек динамической компоновки, с помощью которых выполняется предварительная компиляция сценарных процедур на этапе разработки, и исполнение сценариев в процессе выполнения приложения. Бейсик-сценарий обеспечивает возможность не только манипуляции задачами, входящими в стратегию (приложение), но и взаимодействия с Windows-приложениями и другими приложениями посредством механизмов DDE, OLE, OPC и ODBC (SQL). Синтаксис

Бейсик-сценария совместим с Microsoft VBA (Visual Basic for Application, используемым в Excel, Word, Access и т. п.) и Microsoft Visual Basic. Бейсик-сценарий и VBA имеют ряд несовместимых функций, но более 95% функций и процедур абсолютно идентичны. При использовании в Бейсик-сценарии базовых функций имеется возможность компиляции и исполнения программ, написанных на Visual Basic, без каких-либо изменений. Номера ошибок и сообщения об ошибках в Бейсик-сценарии также совместимы с имеющимися в Visual Basic. В состав Бейсик-сценария входит редактор, позволяющий разрабатывать собственные средства взаимодействия с оператором. Средство программирования сценарных процедур делает пакет GeniDAQ одним из наиболее удобных и современных инструментов для разработки программного обеспечения верхнего уровня систем сбора данных и оперативного диспетчерского управления (SCADA).

Драйверы в виде библиотек динамической компоновки. В пакете GeniDAQ, как и в более ранних версиях пакета, используются загружаемые драйверы в виде библиотек динамической компоновки. Для обмена информацией с исполнительной средой GeniDAQ в драйверах используется интерфейс прикладного программирования C API. *Драйверы устройств* могут непосредственно обмениваться данными с центром обработки данных посредством вызовов функций языка C. Таким же способом может осуществляться информационный обмен между GeniDAQ и другими приложениями Windows.

Редактор отчетов является отдельным приложением, которое полностью независимо от исполнительной среды. Редактор отчетов обеспечивает пользователю возможность формирования отчетов и вывода их на экран монитора или печатающее устройство.

Глава 4. Учебник по SCADA-системе GeniDAQ

Данная глава является основной главой первой части учебного пособия. В ней с использованием демонстрационных примеров приводятся общие правила выполнения основных операций редактора задач, редактора форм отображения и редактора сценариев SCADA-пакета GeniDAQ [13].

> ***Совет***
> На прилагаемом DVD-диске содержатся файлы для инсталляции демонстрационной версии SCADA-системы GeniDAQ и указания по ее инсталляции. Настоятельно советуем вам установить SCADA-систему и пользоваться ею при изучении материала — это важно.

Для создания приложений (стратегий) и окон отображения с помощью редактора задач и редактора форм отображения следует использовать мышь, что позволит значительно повысить степень интуитивного восприятия предоставляемого GeniDAQ графического интерфейса. Далее будет использоваться следующая терминология, относящаяся к способам работы с указательными устройствами в среде операционной системы Windows.

Указание объекта заключается в помещении подвижного (не текстового) курсора мыши в область отображения объекта (пиктограммы, кнопки, поля, имени файла и т. п.) на экране монитора путем перемещения мыши.

Щелчок и двойной щелчок — помещение подвижного курсора мыши в область отображения объекта на экране монитора путем перемещения мыши с последующим нажатием и быстрым отпусканием ее левой клавиши (щелчок) или с последовательным двойным нажатием и отпусканием левой клавиши мыши (двойной щелчок). Указанные действия обычно ассоциируются с выбором объекта и с выполнением операции, связанной с данным объектом. Выбор блока в окне редактора задач или элемента отображения в окне редактора форм отображения приведет к выделению выбранного объекта вдоль периметра рамкой. Объект остается выбранным (выделенным) до выполнения следующей операции.

Перенос курсора — перемещение подвижного курсора мыши при одновременном нажатии и удержании ее левой клавиши с последующим отпусканием. При выполнении операции переноса с одновременным выбором объекта происходит его перемещение до отпускания левой клавиши мыши. Операция переноса курсора может быть использована также для выделения нескольких объектов в окне редактора задач или редактора форм отображения.

Использование справочной системы. В процессе работы с GeniDAQ имеется возможность получения справочной информации путем выполнения команды **Help | Help Topics** построителя стратегии. Кроме того, в каждом окне диалога имеется кнопка **Help**, позволяющая вызвать относящуюся к текущему диалогу

справочную информацию. Контекстная справка, если она предусмотрена, может быть вызвана путем выбора кнопки на панели инструментов с изображением стрелки и вопросительного знака.

Учебные стратегии GeniDAQ. Для быстрого освоения приемов работы со SCADA-системой GeniDAQ в комплект поставки включены девять учебных приложений (стратегий), которые после инсталляции SCADA-системы помещаются в папку \Program Files\Advantech\GeniDAQ\Strategy. В учебных стратегиях используется программный эмулятор сигналов **Advantech DEMO I/O=1H**, который позволяет создавать и запускать на исполнение демонстрационные и обучающие приложения без применения каких-либо дополнительных аппаратных средств. В этой же папке имеется и еще ряд стратегий, изучение которых представляет интерес. Далее, с целью обучения, приводится описание всех учебных приложений и, дополнительно, разработанных автором некоторых других стратегий (приложений). Такой подход обеспечивает быстрое формирование у пользователя начальных навыков работы со SCADA-системой.

4.1. Занятие 1 "Демонстрация базовых приемов работы — одна задача реального времени с отображением результатов ее работы в одном экранном окне"

Цель занятия состоит в демонстрации базовых приемов работы со SCADA-системой GeniDAQ. Далее показывается простой способ использования средств, предоставляемых GeniDAQ, для сбора в реальном масштабе времени измерительной информации и отображения ее в экранном окне на графике зависимости от времени и на цифровом индикаторе. Для этого используются программный эмулятор сигналов **Advantech I/O**, блок аналогового ввода для сбора измерительной информации и два индикатора — график зависимости от времени и цифровой индикатор.

4.1.1. Используемый инструментарий

*Используемый инструментарий***.** Для построения пользовательского *приложения*, выполняющего требуемые функции, используется построитель приложений GeniDAQ Builder и программный эмулятор **Advantech DEMO I/O**. Процесс построения приложения состоит из двух этапов:

- собственно проектирования приложения (в SCADA-системе GeniDAQ этот этап называют построением *стратегии*);
- разработка визуального отображения работы созданного приложения для оперативного персонала (в SCADA-системе GeniDAQ этот этап называют построением *форм отображения*).

На первом этапе построитель приложений использует *редактор задач*, имеющий собственное окно и панель инструментов (рис. 4.1).

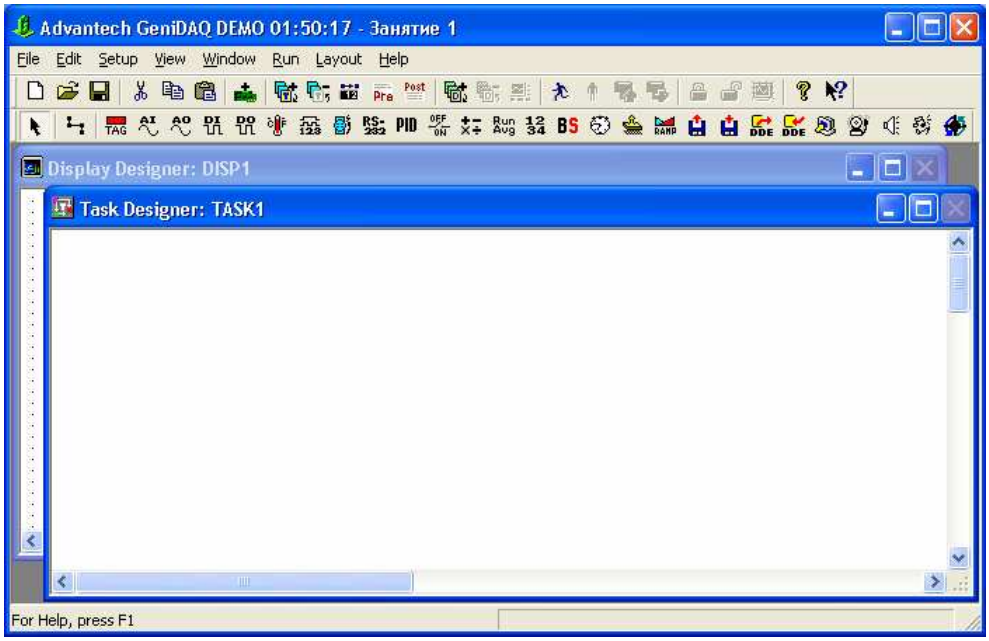

Рис. 4.1. Главное окно построителя приложений с окном **Task Designer: TASK1** и панелью инструментов редактора задач

На втором этапе построитель приложений использует *редактор форм отображения*, который также использует собственное окно и панель инструментов (рис. 4.2).

Редактор задач использует визуальную (информационно-поточную) модель программирования, которая значительно удобнее, чем традиционная линейная архитектура текстовых языков программирования. При проектировании приложения для сбора данных и управления пользователь (разработчик) создает блок-схему стратегии. При этом он не должен уделять особого внимания различным логическим и синтаксическим соглашениям, принятым в стандартных языках программирования. При проектировании приложения разработчик просто выбирает объекты (пиктограммы функциональных блоков на панели инструментов редактора задач) и соединяет их проводниками для передачи данных от одного блока к другому. Каждый функциональный блок, представлен пиктограммой на панели инструментов редактора задач и предназначен для выполнения соответствующей встроенной функции обработки данных, поступающих от аппаратуры или вводимых пользователем. Ряд блоков позволяет организовывать взаимодействие непосредственно с низкоуровневыми драйверами аппаратуры.

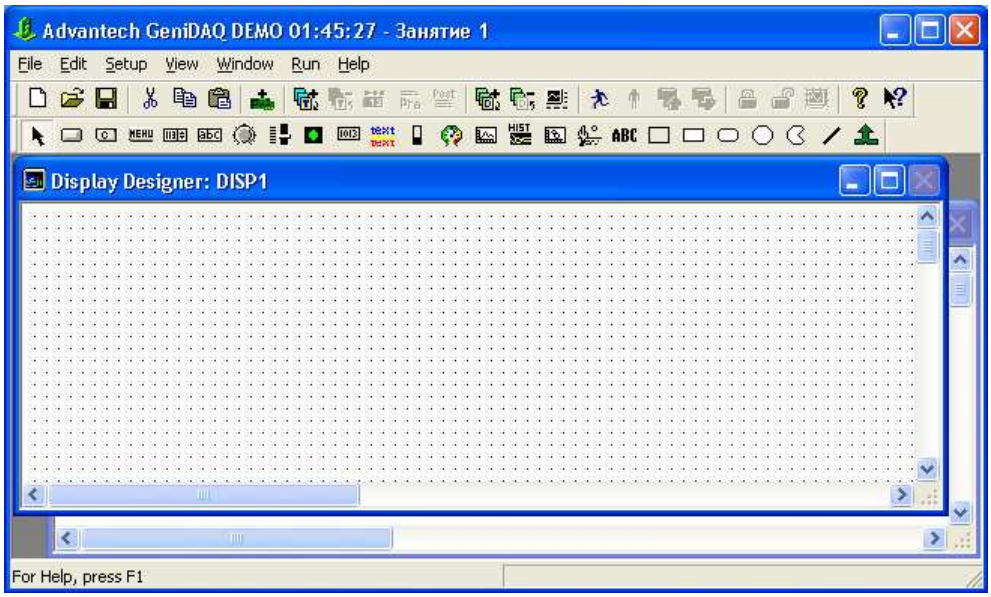

Рис. 4.2. Главное окно построителя приложений с окном **Display Designer: DISP1** и панелью инструментов редактора форм отображения

Ранее уже упоминалось, что редактор задач состоит из двух основных компонент: панели инструментов и окна. Для создания стратегии в клиентскую область окна редактора задач с помощью операции "перенести и оставить" (Drag and Drop) помещаются требуемые функциональные блоки. При этом размещение функционального блока выполняется путем щелчка левой кнопкой мыши по его пиктограмме на панели инструментов, последующем перемещении курсора мыши в требуемую позицию окна редактора задач и повторного щелчка левой кнопкой мыши. После размещения в окне всех необходимых функциональных блоков и их конфигурирования устанавливаются связи между ними в соответствии с требованиями реализуемого алгоритма сбора данных и управления. Для этой цели используется проводник, представленный соответствующей пиктограммой на панели инструментов редактора задач.

В рамках данного занятия используется один функциональный блок редактора задач — *блок аналогового ввода*. Данный функциональный блок предназначен для приема информации от устройств, имеющих подсистему ввода аналоговых сигналов, и передачи указанных сигналов другим функциональным блокам и элементам отображения. Двойной щелчок левой клавишей мыши на пиктограмме блока аналогового ввода в окне редактора задач приводит к появлению диалогового окна настройки параметров блока. Пример диалогового окна приведен на рис. 4.3. Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров блока аналогового ввода будут также рассмотрены в последующих примерах.

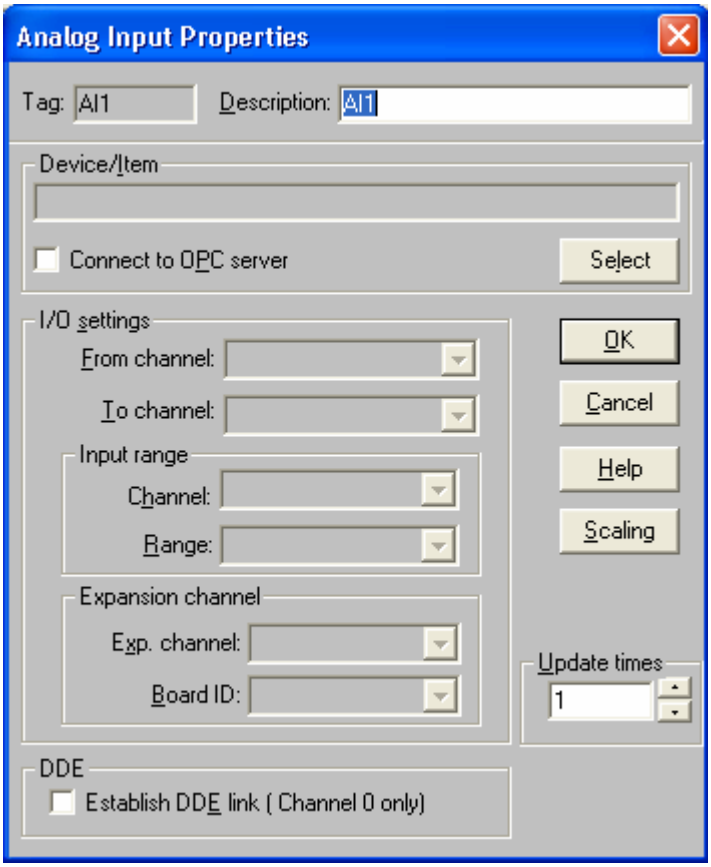

Рис. 4.3. Вид диалогового окна настройки блока аналогового ввода

Редактор форм отображения предназначен для создания графических мнемосхем автоматизированных рабочих мест оператора, обеспечивающих динамическое представление информации о контролируемом технологическом процессе в удобной для восприятия форме. Кроме того, редактор обеспечивает возможность использования растровых изображений, создаваемых разработчиком, в качестве фоновых рисунков. Оператор может не только следить за ходом контролируемого процесса, но и осуществлять управление его параметрами в процессе исполнения стратегии, спроектированной с помощью редактора задач.

По завершении или во время разработки стратегии может быть принято решение о том, какая информация и в каком виде должна быть представлена оператору в окне редактора форм отображения в процессе исполнения стратегии. Перед началом создания мнемосхемы рекомендуется уделить серьезное внимание предварительной проработке ее внешнего вида — перечню элементов отображения/управления (динамических элементов), виду фоновых рисунков (статическая часть мнемосхемы),

а также тому, какие именно данные должны быть отображены с помощью динамических элементов.

Редактор форм отображения, как и редактор задач, состоит из двух основных компонент: панели инструментов с пиктограммами динамических элементов (элементов отображения/управления) и окна. Для создания мнемосхемы в клиентскую область окна редактора форм отображения с помощью операции "перенести и оставить" (Drag and Drop) помещаются требуемые динамические элементы. После размещения в окне всех необходимых динамических элементов, их конфигурирования и создания фоновых рисунков создание мнемосхемы закончено и ею можно пользоваться.

В рамках данного занятия используется два динамических элемента редактора форм отображения — цифровой индикатор (Numeric String) и временной график (Realtime Trend Graph).

Цифровой индикатор предназначен для отображения значения параметра на выходе присоединенного функционального блока стратегии, а также для вывода на экран текстовых строк, поступающих с выхода блока процедуры пользователя или блока Бейсик-сценария в процессе исполнения стратегии. Цифровой индикатор может быть помещен в окно формы отображения и связан с выходной переменной функционального блока задачи, входящей в стратегию. Имеется возможность установки требуемых размеров индикатора. Цифровой индикатор может использоваться для отображения вещественных (с плавающей точкой) и целочисленных переменных, а также текстовых строк. Кроме того, имеется возможность установки требуемого количества цифр после десятичной точки при отображении вещественных значений, выбора типа шрифта, его цвета и размера, а также метода выравнивания отображаемого значения или строки. Двойной щелчок левой клавишей мыши на пиктограмме цифрового индикатора в окне редактора форм отображения приводит к появлению диалоговой окна настройки параметров индикатора. Пример диалогового окна приведен на рис. 4.4. Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров цифрового индикатора будут также рассмотрены в примерах, следующих далее.

Временной график предназначен для отображения на мнемосхеме (в окне формы отображения) временной зависимости одной или нескольких переменных стратегии. Количество зависимостей величин на выходах функциональных блоков от времени, отображаемых с помощью динамического элемента, может быть неограниченным, однако из практических соображений не рекомендуется строить более восьми временных зависимостей на одном графике. Имеется возможность выбора цвета графика и диапазонов по осям абсцисс (время) и ординат (параметр) Двойной щелчок левой клавишей мыши на пиктограмме временного графика в окне редактора форм отображения приводит к появлению диалоговой окна настройки параметров графика. Пример диалогового окна приведен на рис. 4.5. Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров временного графика будут рассмотрены в примерах, следующих далее.

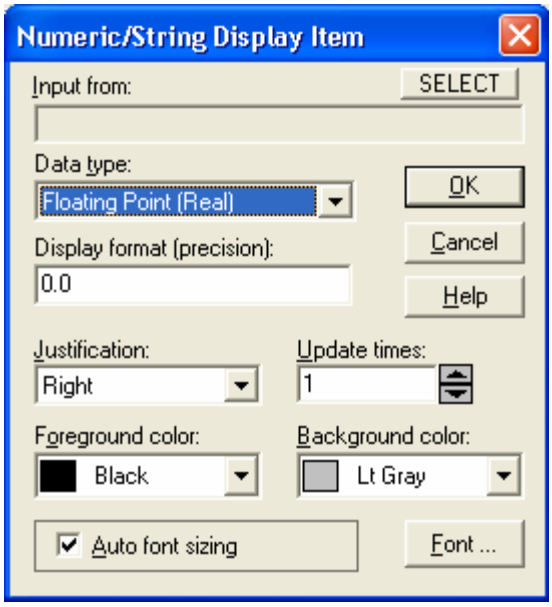

Рис. 4.4. Вид диалогового окна для задания параметров цифрового индикатора

Программный эмулятор Advantech DEMO I/O представляет собой программный эмулятор платы ввода/вывода, который позволяет проверить большинство функций без использования каких-либо аппаратных средств ввода-вывода. Эмулятор имеет в своем составе четыре канала аналогового ввода и два канала цифрового ввода — вывода. Канал 0 аналогового ввода моделирует синусоидальный сигнал ± 5 вольт, канал 1 — сигнал прямоугольной формы ± 5 вольт, канал 2 — пилообразный сигнал ± 5 вольт и канал 3 — случайный сигнал. Каждый используемый эмулятор должен иметь собственный уникальный базовый адрес. Устанавливайте базовый адрес для каждого эмулятора ввода/вывода в соответствующем диалоговом окне.

4.1.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. *Первое действие — запустите построитель приложений и создайте новое приложение*. После включения компьютера на экране компьютера появляется запрос на ввод атрибутов пользователя. Запустите построитель приложений двойным щелчком левой кнопкой мыши на ярлыке **GeniDAQ Builder 4.25**, расположенном на рабочем столе. В результате на экран будет выведено главное окно построителя приложений (рис. 4.6). Для запуска построителя задач в демонстрационном режиме последовательно нажмите кнопки **Exit** и **OK**. Демонстрационный режим отличается непрерывным двухчасовым интервалом работы и меньшим количеством входных и выходных каналов. В нашем случае

это несущественно. Нажмите кнопку **New** на панели инструментов (ее легко найти, так как все кнопки снабжены всплывающими подсказками). После этого в главном окне построителя приложений появится диалоговое окно настройки, которое может быть настроено так, как это показано на рис. 4.7. При работе в учебной лаборатории в поле **Location** качестве каталога следует использовать каталог, к которому имеется полный доступ. После нажатия кнопки **OK** и подтверждения необходимости создания каталога **Занятие 1**, который до этого не существовал, в главном окне построителя задач появятся пустые окна редактора задач **TASK1** и редактора форм отображения **DISP1**. Произведите щелчок мышью в области окна редактора задач и окно с заголовком **Task Designer: TASK1** будет выдвинуто на передний план (см. приведенный ранее рис. 4.1). В окне построителя задач появится еще одна панель инструментов, содержащая пиктограммы встроенных функциональных блоков редактора задач.

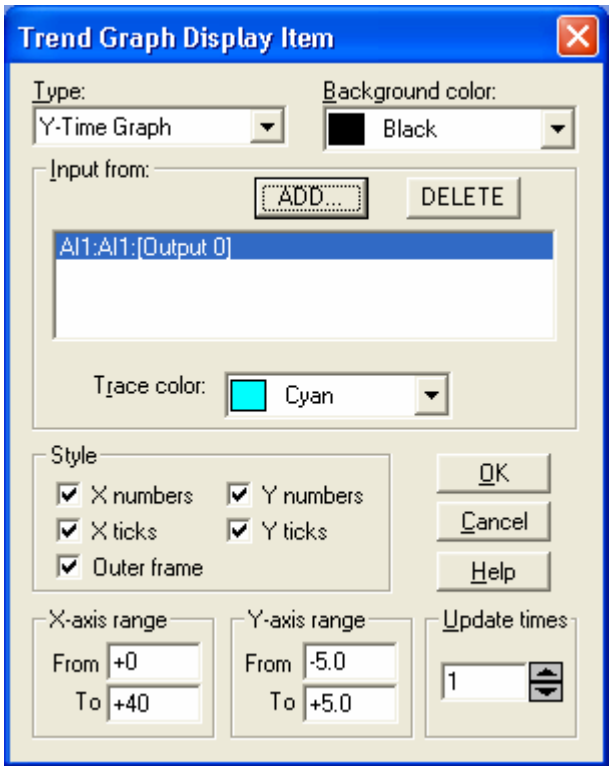

Рис. 4.5. Пример диалогового окна настройки графика времени

Примечание

Атрибуты пользователя следует получить у преподавателя. Пользователь должен обладать правами администратора.

2. *Второе действие — задайте временные параметры запуска приложения*. Для этого нажмите кнопку **Task Properties** на панели инструментов или выполните команду **Setup | Task Properties**, в появившемся окне задайте требуемое значение периода (рис. 4.8) и нажмите кнопку **OK**. В нашем примере задан период запуска приложения 100 миллисекунд.
3. *Третье действие — разместите в окне редактора задач блок аналогового ввода и настройте его*. На панели инструментов выберите кнопку с мнемоникой **AI**, поместите на нее курсор мыши, щелкните левой кнопкой мыши, переместите мышь в окно редактора задач и еще раз щелкните левой кнопкой мыши. При этом в окне редактора задач появится блок аналогового ввода с позиционным обозначением **AI1**. Данную операцию принято называть "перенести и оставить" (Drag and Drop) — рис. 4.9. Произведите двойной щелчок левой кнопкой мыши над блоком аналогового ввода и появится диалоговое окно его настройки. Для выбора источника аналогового сигнала, если для блока аналогового ввода не указано связанное с ним устройство (поле **Device/Item** пусто), нажмите кнопку **Select**, в появившемся окне диалоговом окне выберите программный эмулятор сигнала (рис. 4.10) и нажмите кнопку **OK**. В данном случае **1H** является 16-ричным адресом экземпляра эмулятора. В результате диалоговое окно настройки блока аналогового ввода приобретает вид, показанный на рис. 4.11. Нулевые значения каналов в полях **From channel** и **To channel** означают, что в эмуляторе будет использован синусоидальный сигнал с амплитудой 5 вольт и нулевым смещением. Значение в поле **Update times** является делителем, который позволяет вызывать блок аналогового ввода и сканировать соответствующие ему каналы устройства аналогового ввода реже, чем вызывается вся задача, в которую входит данный функциональный блок. Например, пусть задача вызывается (сканируется) один раз в 100 миллисекунд. Для того чтобы блок аналогового ввода, входящий в данную задачу, вызывался один раз в 200 миллисекунд, следует установить в поле **Update times** значение 2. В этом случае значение на выходе блока аналогового ввода будет обновляться через каждые два вызова задачи, содержащей блок. Делитель может изменяться в диапазоне от 1 до 32767. В нашем примере задача и блок аналогового ввода выполняются синхронно, с одинаковым периодом 100 миллисекунд. Для сохранения значений параметров блока аналогового ввода нажмите кнопку **OK**.

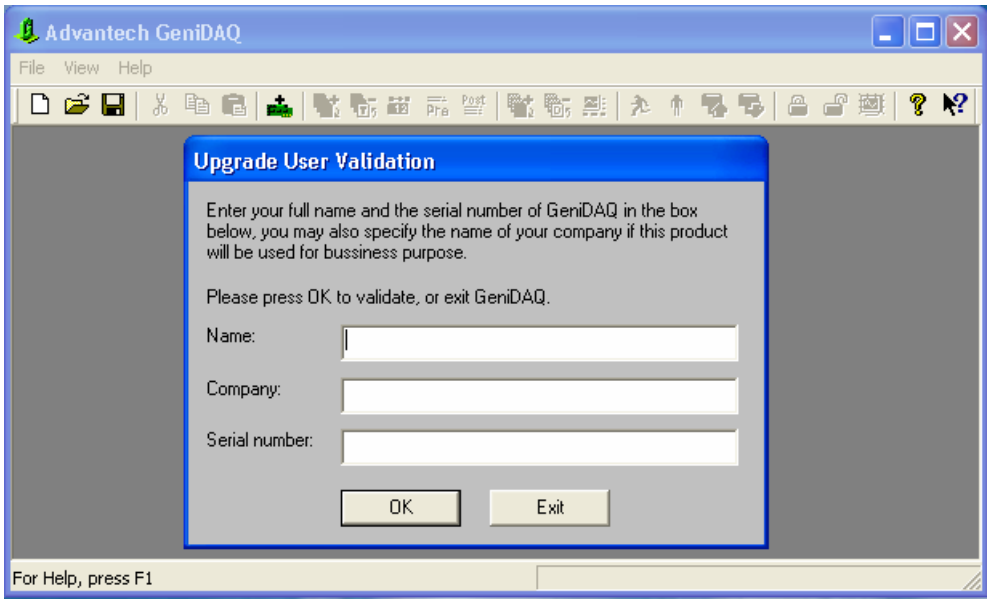

Рис. 4.6. Главное окно построителя приложений

4. *Четвертое действие — перейдите в окно редактора форм отображения и сконфигурируйте его*. Чтобы сделать активным окно **DISP1** редактора форм отображения щелкните по этому окну левой кнопкой мыши. При этом в верхней части главного окна появится панель инструментов редактора форм отображения (см. приведенный ранее рис. 4.2). На панели инструментов выберите кнопку **RealTime Trend Graph** (временной график) и с помощью мыши перетащите ее и положите во внутреннюю область окна редактора форм отображения (рис. 4.12). Для настройки свойств временного графика щелкните дважды левой кнопкой мыши над пиктограммой временного графика. В результате появится диалоговое окно настройки, показанное ранее на рис. 4.5. Для привязки графика к источнику информации в этом окне нажмите кнопку **ADD** и в появившемся окне выполните необходимую настройку (рис. 4.13). Затем с помощью кнопок **OK** закройте это и предыдущее окна. Аналогичным образом перетащите в окно редактора форм отображения и настройте цифровой индикатор. С этой целью на панели инструментов выберите кнопку **Numeric String** (цифровой индикатор) и с помощью мыши перетащите ее и положите во внутреннюю область окна редактора форм отображения (рис. 4.14). Для настройки свойств цифрового индикатора щелкните дважды левой кнопкой мыши над пиктограммой цифрового индикатора. В результате появится диалоговое окно настройки, показанное ранее на рис. 4.4. Для привязки графика к источнику информации в этом окне нажмите кнопку **SELECT** и в появившемся окне выполните необходимую настройку (рис. 4.15). Затем с помощью кнопок **OK** закройте это и предыдущее окна.

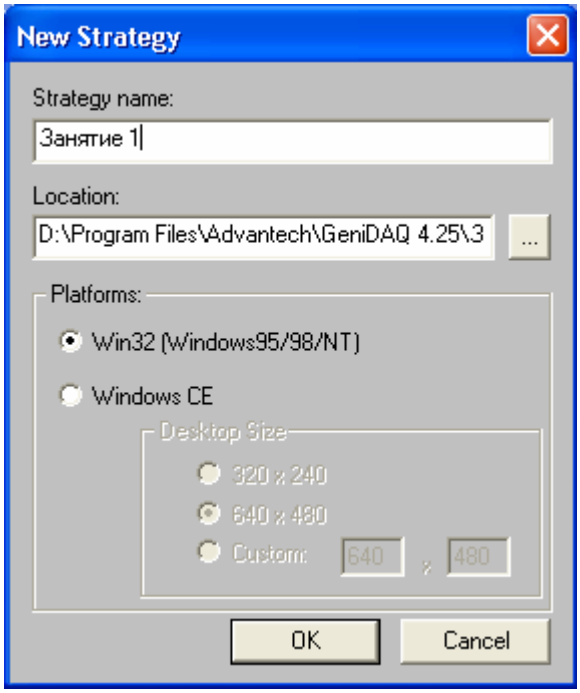

Рис. 4.7. Окно настройки приложения

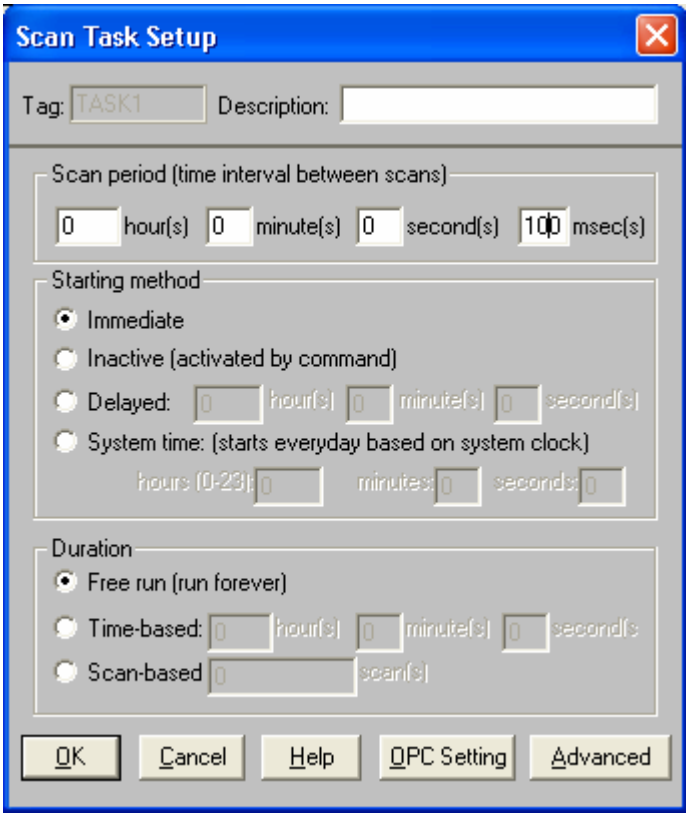

Рис. 4.8. Задание временных параметров запуска приложения

5. *Пятое действие — сохраните созданный проект (стратегию).* Для этого выполните команду **File | Save As**, в появившемся окне диалога укажите необходимую информацию (рис. 4.17) и нажмите кнопку **Сохранить**.
6. *Шестое действие — запуск и останов проекта*. После сохранения созданного проекта имеется возможность немедленного запуска созданного проекта с помощью кнопки **Start** на панели инструментов (рис. 4.17). Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
7. *Седьмое действие — завершение работы GeniDAQ*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

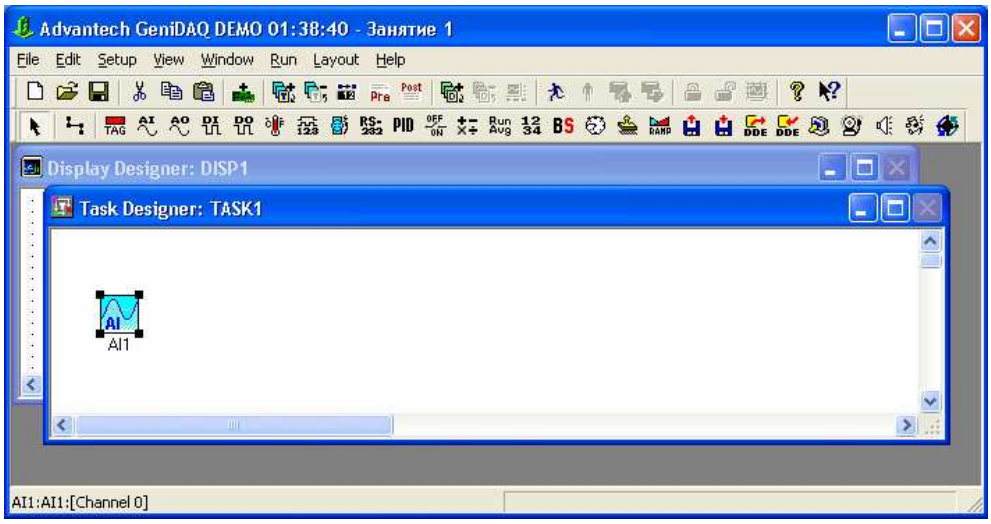

Рис. 4.9. Добавление блока аналогового ввода

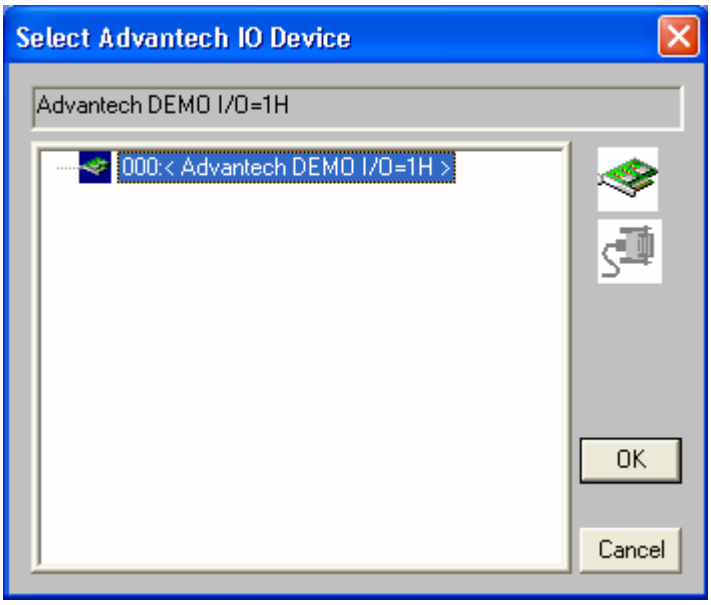

Рис. 4.10. Добавление программного эмулятора ввода/вывода

4.1.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

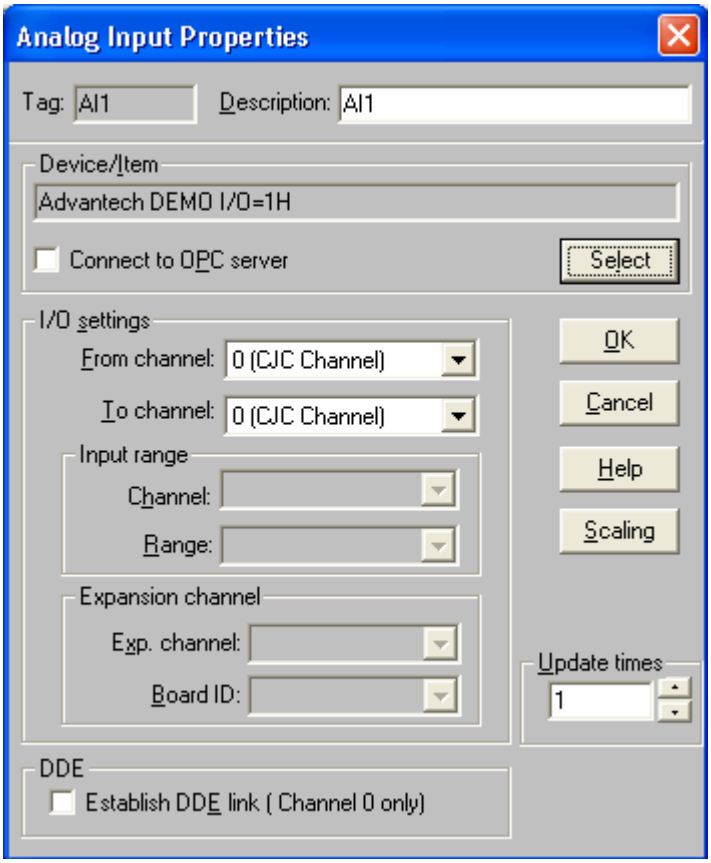

Рис. 4.11. Параметры блока аналогового ввода

Упражнение 4.1. Повтор создания и тестирования приложения "Занятие 1"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 1**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Упражнение 4.2

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 0.2 секунды. С помощью эмулятора и четырех блоков аналогового ввода, помещенных в окно редактора задач, смоделируйте соответственно гармонический, прямоугольный, пилообразный и случайный сигналы. В окне редактора форм отображения покажите указанные процессы на общем графике времени и с помощью цифровых индикаторов. Снабдите элементы отображения поясняющими надписями.

4.2. Занятие 2 "Переключение окон отображения"

Цель занятия состоит в изучении правил разработки приложения с несколькими окнами отображения и способа переключения окон отображения в процессе выполнения приложения. Для этого используется программный эмулятор сигналов **Advantech I/O**, блок аналогового ввода для сбора измерительной информации с канала 1 программного эмулятора и два индикатора для отображения получаемой информации в разных окнах отображения — в виде временного графика и цифрового индикатора. Новым является то, что для переключения окон отображения используется кнопка меню.

4.2.1. Используемый инструментарий

*Используемый инструментарий***.** Новым на данном занятии является использование для построения стратегии *командной кнопки (кнопки меню)* редактора форм отображения. Данный элемент управления предназначен для создания командных кнопок в окнах отображения, позволяющих управлять процессом исполнения стратегии. С помощью кнопки меню можно реализовать одну из трех функций **Function**: *действие* **Action**, *переключение окна отображения* **Display switching** или *управление задачей* **Task control**.

Функция **Action** позволяет выбрать одну из следующих операций при нажатии на кнопку меню: **NONE** — функция выключена; **START** — запуск стратегии на исполнение; **STOP** — завершение исполнения стратегии; **PAUSE** — приостанов исполнения стратегии; **RESUME** — возобновление исполнения после приостанова; **CLOSE** — завершение исполнения стратегии и завершение сеанса исполнительной среды; **LOCK** — блокировка пунктов меню и других органов управления Windows; **UNLOCK** — разблокировка и др. При выборе второй функции нажатие на кнопку во время исполнения стратегии приведет к выдвижению на передний план окна отображения, идентификатор которого выбран в поле **Display Switch** диалоговой панели настройки параметров элемента управления. Использование именно функции **Display switching** и демонстрируется на данном занятии. Двойной щелчок левой клавишей мыши на пиктограмме кнопки меню в окне редактора форм отображения приведет к появлению диалоговой окна настройки параметров кнопки. Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров кнопки меню будут рассмотрены в нижеследующих примерах.

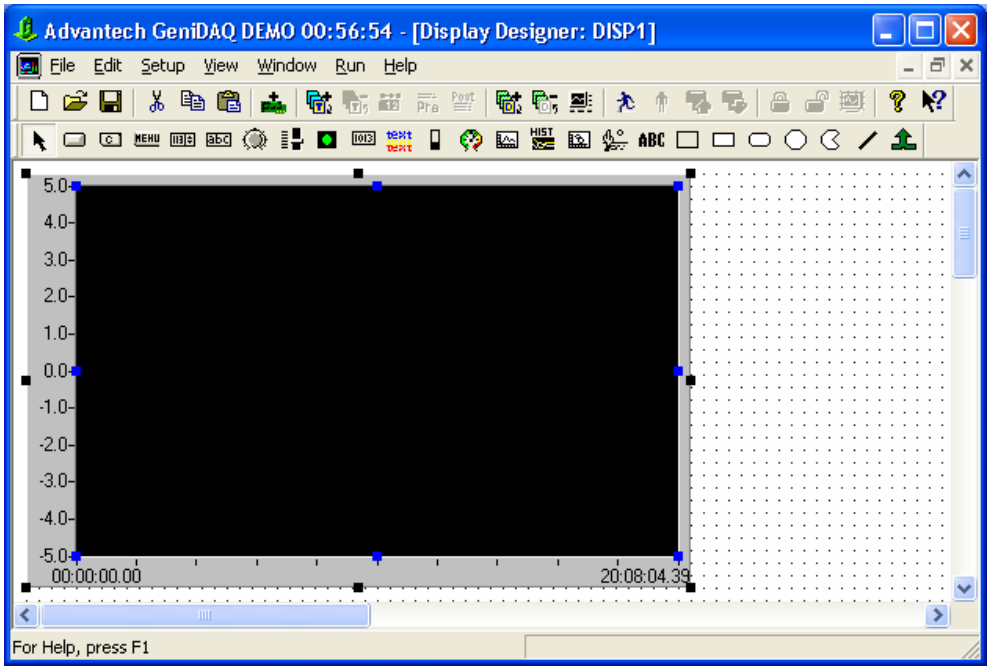

Рис. 4.12. Элемент отображения временного графика

4.2.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.18.
2. Выполните *второе действие* из занятия 1. В нашем примере также задан период запуска приложения 100 миллисекунд.
3. Выполните *третье действие* из занятия 1. Сконфигурируйте блок аналогового ввода в соответствии с рис. 4.19. Единичные значения номеров каналов **From channel** и ***To channel*** означают, что в эмуляторе будет использован меандр (прямоугольный сигнал) со значениями ± 5 вольт. Для сохранения значений параметров блока аналогового ввода нажмите кнопку ***OK***.
4. Выполните *четвертое действие* из занятия 1 в части графика времени. Сконфигурируйте график времени в соответствии с рис. 4.20, а затем с помощью кнопок **OK** закройте окна.
5. *Пятое действие*. Выполните команду **File | Add/Delete | Add Display**. В главном окне построителя приложений появится еще одно окно отображения с заголовком **Display Disigner: DISP2**. Поместите в это окно цифровой индикатор аналогично

тому, как это указано в действии 4 из занятия 1 и сконфигурируйте цифровой индикатор в соответствии с рис. 4.21.

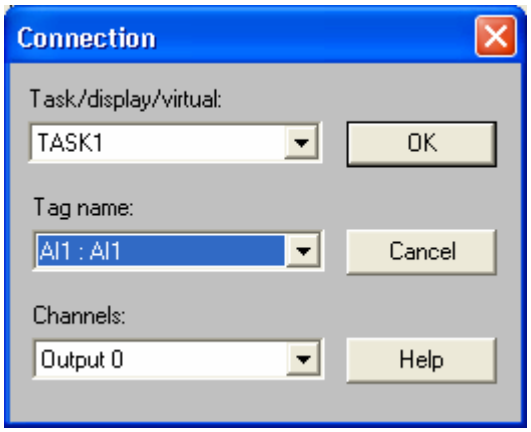

Рис. 4.13. Задание отображаемого параметра для графика времени

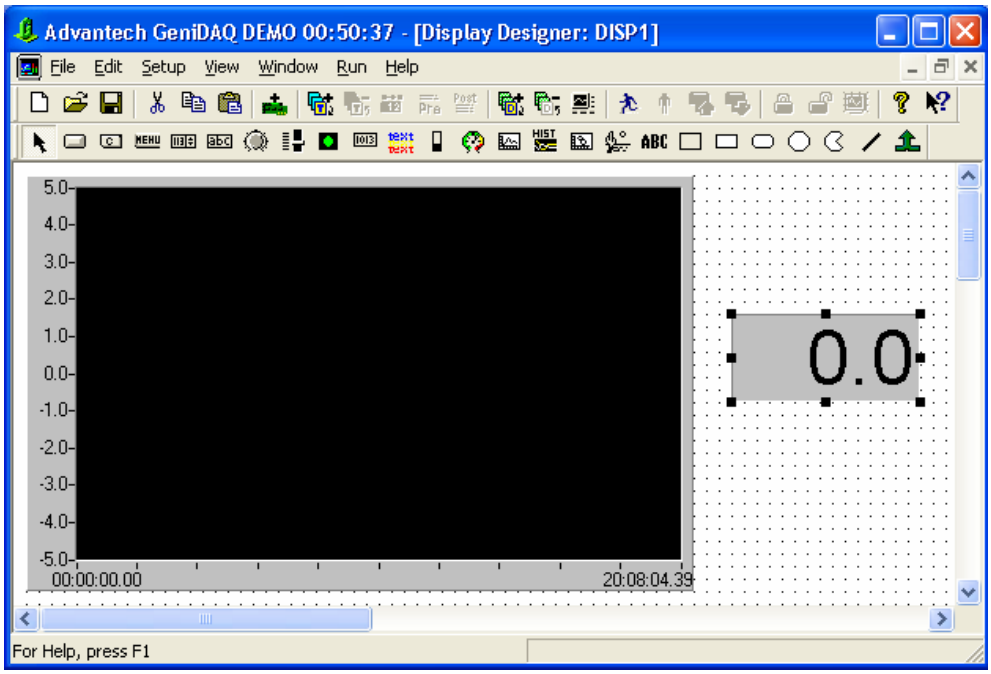

Рис. 4.14. Цифровой индикатор

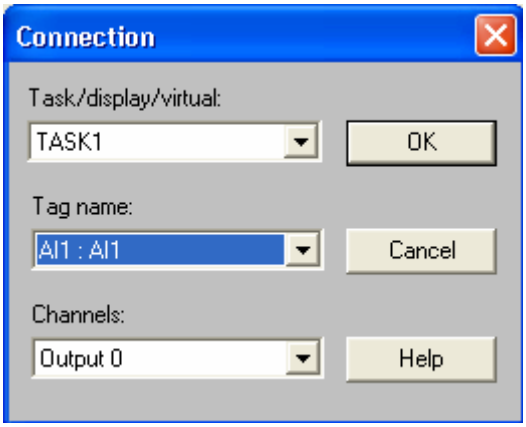

Рис. 4.15. Источник информации для цифрового индикатора

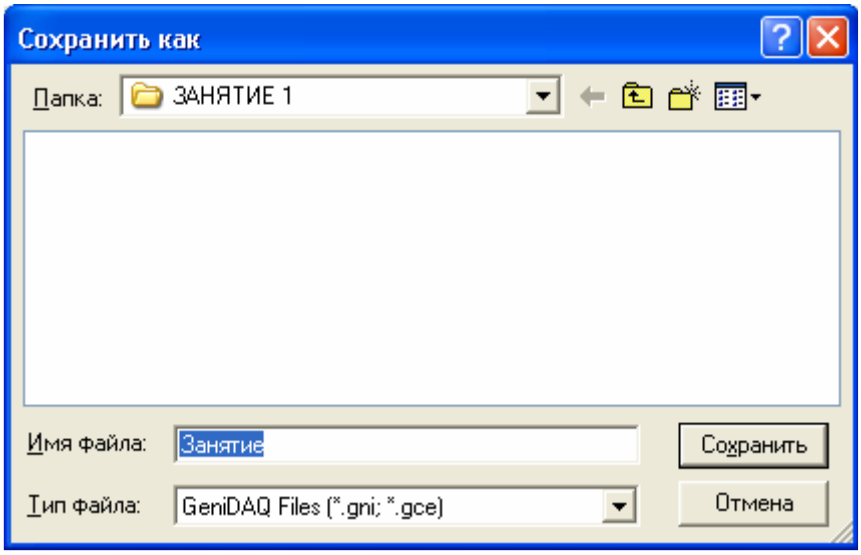

Рис. 4.16. Сохранение созданного программного проекта (стратегии)

6. *Шестое действие*. Перетащите в окно отображения **DISP2** кнопку меню **Menu Button** (рис. 4.22). Сконфигурируйте ее в соответствии с рис. 4.23.
7. *Седьмое действие*. Активизируйте окно отображения **DISP1** и перетащите в него кнопку меню **Menu Button** (рис. 4.24). Сконфигурируйте ее в соответствии с рис. 4.25.
8. *Восьмое действие*. Сохраните созданный проект (стратегию) аналогично пятому действию из занятия 1.

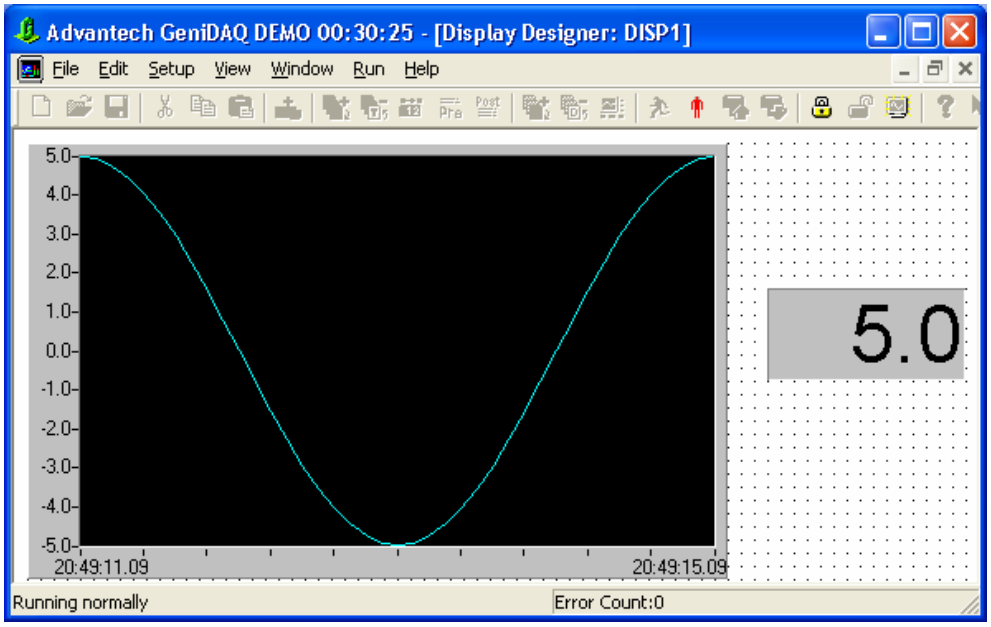

Рис. 4.17. Демонстрация работы проекта

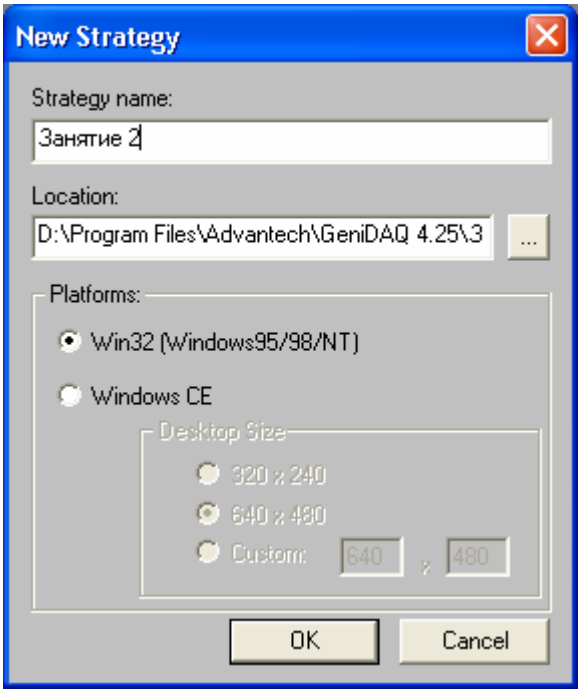

Рис. 4.18. Конфигурирование параметров создаваемого приложения

9. *Девятое действие*. После сохранения файлов созданного проекта имеется возможность немедленного запуска созданного проекта с помощью кнопки **Start** на панели инструментов (рис. 4.26). Для перехода в окно отображения **DISP2** в процессе работы приложения достаточно нажать кнопку меню **В DISP2**. Результат такого действия показан на рис. 4.27. Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
10. *Десятое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

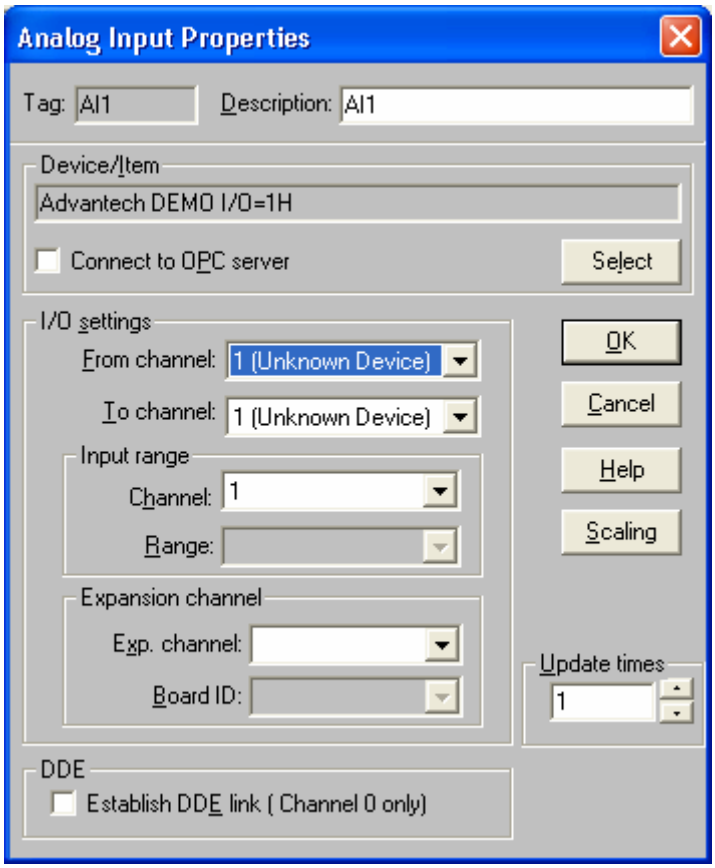

Рис. 4.19. Конфигурирование блока аналогового ввода

4.2.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

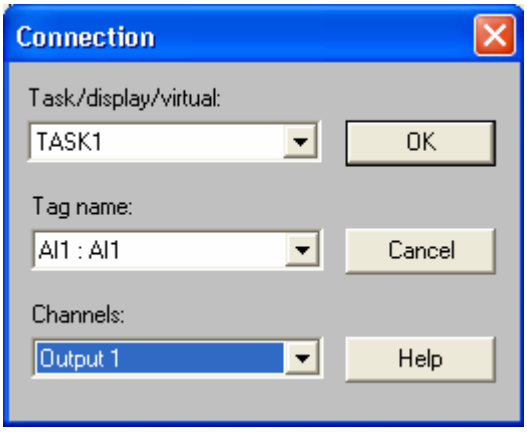

Рис. 4.20. Конфигурирование графика времени

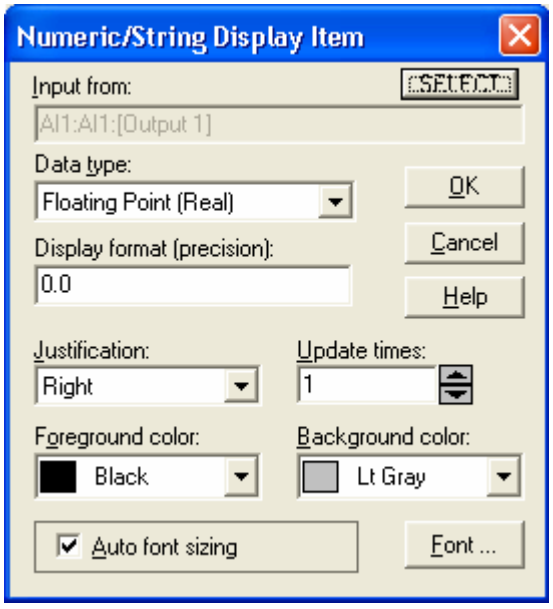

Рис. 4.21. Конфигурирование цифрового индикатора

Упражнение 4.3. Повтор создания и тестирования приложения "Занятие 2"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 2**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

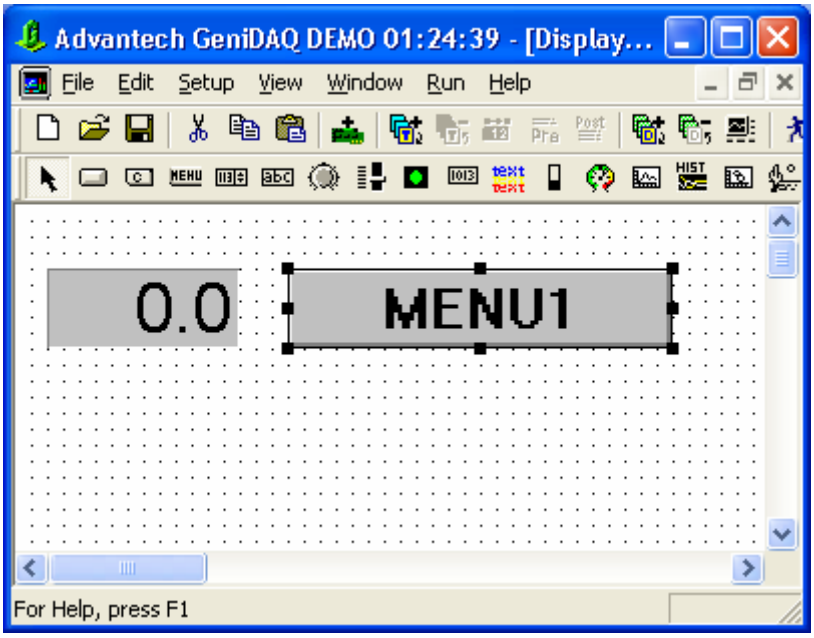

Рис. 4.22. Создание кнопки меню в окне **DISP2**

Упражнение 4.4

Разработайте стратегию, обладающую следующими возможностями. Создайте три окна редактора задач с периодами запуска 0.5, 1.0 и 1.5 секунды соответственно. С помощью эмулятора и блоков аналогового ввода, помещенных в окна редактора задач, смоделируйте соответственно гармонический, прямоугольный и пилообразный сигналы. Создайте четыре окна отображения. В первом и третьем окнах на графиках времени отобразите выходы блоков аналогового ввода в виде синусоидального и пилообразного сигналов. Во втором окне отобразите с помощью цифрового индикатора выход блока аналогового сигнала в виде меандра. Четвертое (главное) окно отображения должно содержать общий график времени и цифровые индикаторы для отображения выходов блоков аналоговых входов. После запуска стратегии должно отображаться главное окно. Предусмотреть переключение между окнами отображения по кольцу и из каждого окна отображения в главное окно отображения.

Упражнение 4.5

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 0.5 секунды. С помощью эмулятора и четырех блоков аналогового ввода, помещенных в окно редактора задач, смоделируйте соответственно гармонический, прямоугольный, пилообразный и случайный сигналы. Создайте пять окон отображения. В первом окне редактора форм отображения покажите указанные процессы на общем графике времени, а в остальных четырех отобразите их с

помощью цифровых индикаторов. После запуска стратегии должно отображаться окно с временным графиком. Предусмотреть переключение между окнами отображения по кольцу вправо (**DISP1-DISP2-DISP3-DISP4-DISP5-DISP1**-…). Снабдите элементы отображения поясняющими надписями.

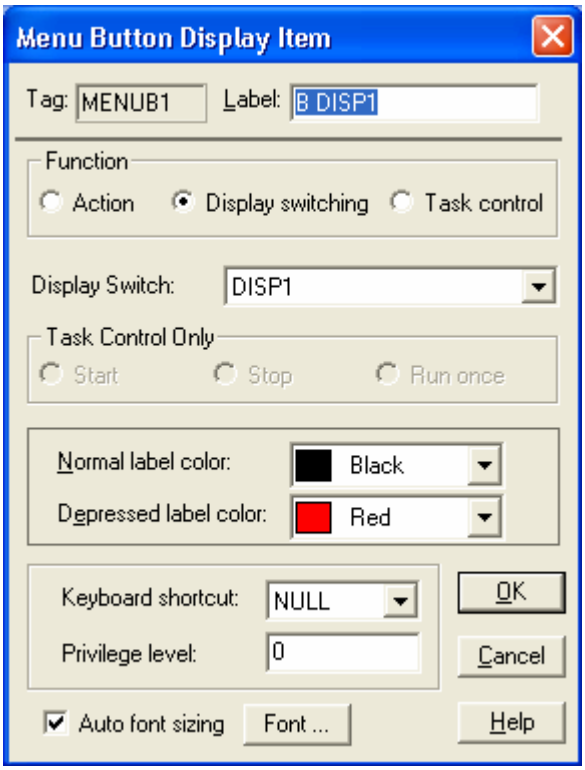

Рис. 4.23. Конфигурирование кнопки меню в окне **DISP2**

Упражнение 4.6

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 0.1 секунды. С помощью эмулятора и четырех блоков аналогового ввода, помещенных в окно редактора задач, смоделируйте соответственно гармонический, прямоугольный, пилообразный и случайный сигналы. Создайте пять окон отображения. В первом окне редактора форм отображения покажите указанные процессы на общем графике времени, а в остальных четырех отобразите их с помощью цифровых индикаторов. После запуска стратегии должно отображаться окно с временным графиком. Предусмотреть переключение между окнами отображения по кольцу влево (**DISP1-DISP5-DISP4-DISP3-DISP2-DISP1**-…). Снабдите элементы отображения поясняющими надписями.

Упражнение 4.7

Разработайте стратегию, обладающую следующими возможностями. Создайте два окна редактора задач с периодами запуска 1.0 и 1.5 секунды соответственно. С помощью эмулятора и блоков аналогового ввода, помещенных в окна редактора задач, смоделируйте соответственно пилообразный и случайный сигналы. Создайте два окна отображения. В первом окне отобразите пилообразный сигнал с помощью временного графика и цифрового индикатора, а во втором окне аналогично отобразите случайный сигнал. После запуска стратегии должно отображаться окно **DISP1**. Предусмотрите переключение между окнами отображения. Снабдите элементы отображения поясняющими надписями.

4.3. Занятие 3 "Просмотр и изменение порядка выполнения функциональных блоков редактора задач"

Цель занятия состоит в изучении средств просмотра и изменения порядка выполнения функциональных блоков редактора задач. Используются программный эмулятор сигналов **Advantech I/O**, два блока аналогового ввода и два блока вычисления с одним оператором, в одном из которых выполняется сложение выходных сигналов блоков аналогового ввода, а в другом – вычитание. Результаты вычислений представляются в окне отображения с помощью цифровых индикаторов, временного графика и поясняющих строк. Изменяется порядок выполнения функциональных блоков стратегии и оценивается его влияние на результат.

4.3.1. Используемый инструментарий

Используемый инструментарий. Новым на данном занятии является использование для построения стратегии блоков вычисления с одним оператором и средств просмотра и изменения порядка выполнения функциональных блоков стратегии**.**

*Блок вычисления с одним оператором***.** Данный блок предназначен для выполнения одной математической операции, такой как сложение, вычитание, умножение, деление и т. д. По крайней мере, один функциональный блок стратегии должен быть присоединен к входу блока вычисления с единственным оператором. Значение на выходе присоединенного блока будет являться первым операндом в производимой математической операции. Вторым оператором может быть константа, заданная в соответствующем поле диалоговой панели настройки параметров блока, либо значение на выходе другого присоединенного функционального блока стратегии (рис. 4.28). Результат на выходе блока вычисления с единственным оператором может быть представлен в виде целого (**Integer**) либо вещественного (**Floating Point**) числа. Операторы и функции блока вычисления с единственным оператором приведены в табл. 4.1. Операторы, после которых стоит символ "*", требуют использования в качестве аргументов (операндов) значения целого (**Integer**) типа. Некоторые из перечисленных выше операторов требуют только один аргумент

(операнд). Логические операции (and, or, xor) требуют использования двух аргументов целого типа. Операторы abs, not, inv, sqrt, log, ln, exp, jct требуют использования одного аргумента (целого или с плавающей точкой в зависимости от типа операции).

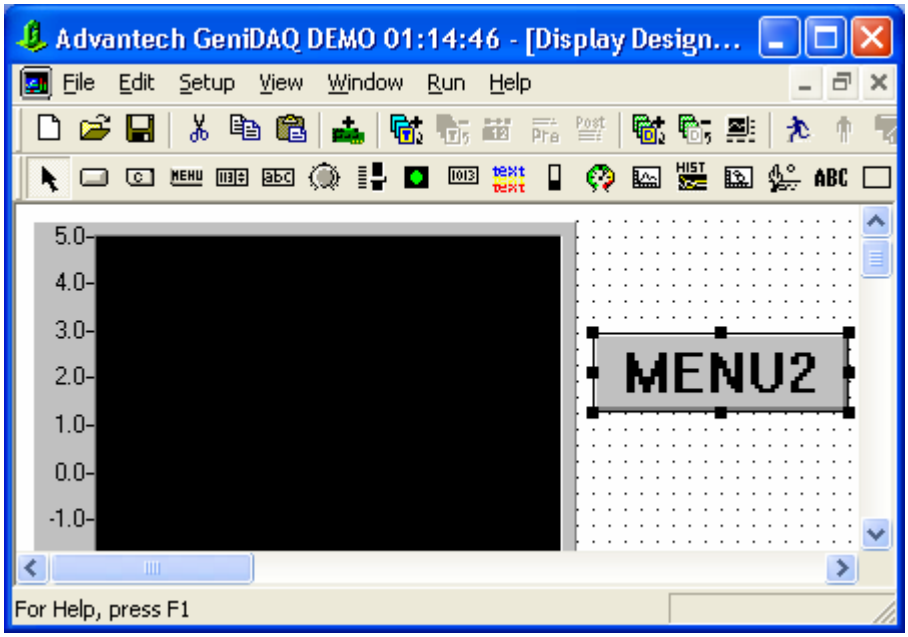

Рис. 4.24. Создание кнопки меню в окне **DISP1**

При передаче значений операндов блоку вычисления с одним оператором необходимо принимать ряд мер предосторожности. Операциям деления и вычисления остатка от целочисленного деления в качестве второго аргумента не должно передаваться нулевое значение. Кроме того, некоторым операторам (sqrt, ln, log) в качестве аргумента должно передаваться только положительное значение. Если не следовать указанным правилам, то при исполнении стратегии будут возникать неустранимые ошибки.

Представленный в таблице оператор jct предназначен для выполнения специальной функции, которая заключается в передаче на выход блока значения, поступающего на его вход. Данная функция наиболее удобна в случае, если требуется установить связь между элементом управления *кнопка* (Binary Button) редактора форм отображения и несколькими функциональными блоками стратегии. Элемент управления *кнопка* может быть связан только с одним функциональным блоком стратегии. Оператор jct, вход которого связан с элементом управления *кнопка*, может быть использован в качестве "точки ветвления" для установления связи с любым количеством функциональных блоков стратегии. Блок вычисления с одним оператором имеет два входа — **Операнд 1** и **Операнд 2** — и один выход, по которому выводится результат выполнения математической операции, выбранной в

поле **Operator** диалогового окна настройки параметров блока. Справочную информацию о параметрах настройки диалогового окна, показанного на рис. 4.28, можно получить также с помощью кнопки **Help**. Конкретные настройки параметров блока вычислений с единственным параметром рассмотрены в примере, следующем далее.

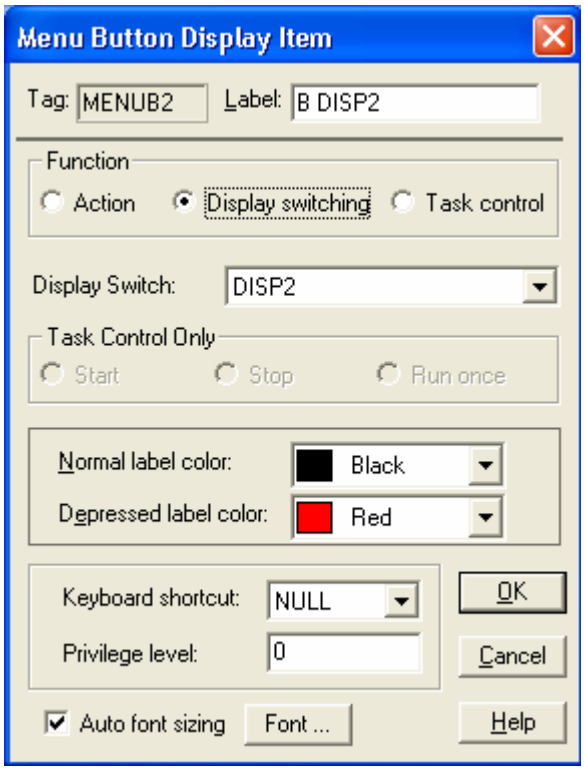

Рис. 4.25. Конфигурирование кнопки меню в окне **DISP1**

Для просмотра и изменения порядка выполнения функциональных блоков стратегии предназначены команды **View | Order Layout**, **Lauout | Complete Reorder** и **Layout | Exchange Order**. Их использование для просмотра и изменения порядка выполнения функциональных блоков также рассмотрено в примере, следующем далее.

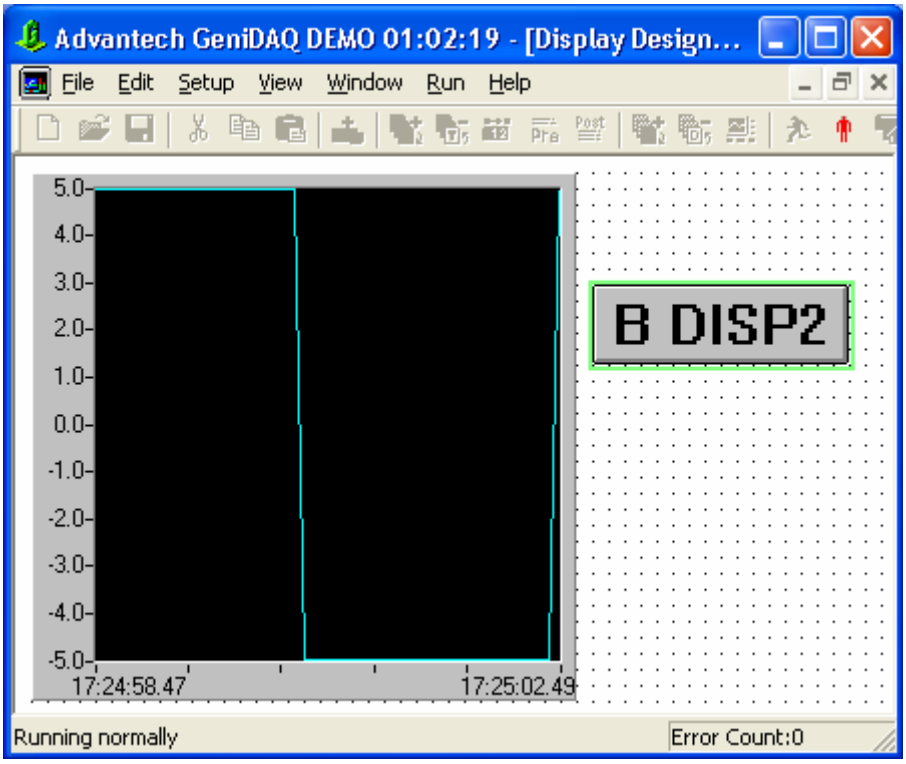

Рис. 4.26. Запуск проекта

4.3.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.29.
2. Выполните *второе действие* из занятия 1. В нашем примере также задан период запуска приложения 100 миллисекунд.
3. *Третье действие*. В окно редактора задач добавьте два блока аналогового ввода с позиционными обозначениями **AI1**, **AI2** соответственно и блок сложения **Single Calculation** (рис. 4.30). Настройте блоки аналогового ввода в соответствии с рис. 4.31 и 4.32.
4. *Четвертое действие*. С помощью проводника установите связь между блоком аналогового ввода **AI1** и блоком сложения. Для этого поместите курсор на пиктограмму проводника на панели инструментов редактора задач и произведите щелчок левой кнопкой мыши. Курсор мыши при этом примет вид катушки с нитками. Поместите курсор в область отображения блока **AI1** и произведите

щелчок левой кнопкой мыши. В появившемся окне диалога (рис. 4.33) выберите канал **Cannel0** и нажмите кнопку **OK**. Переместите курсор по горизонтали в позицию, расположенную над блоком суммирования, щелкните левой кнопкой мыши, затем поместите курсор в область отображения блока суммирования и еще раз щелкните левой кнопкой мыши. В появившемся окне диалога (рис. 4.34) выберите **Operand1** и нажмите кнопку **OK**. Аналогичным образом установите связь между блоком аналогового ввода **AI2** и блоком суммирования (выберите соответственно **Cannel0** *и* **Operand2**). В результате желаемые связи будут установлены (рис. 4.35), а блок сложения настроен нужным образом. Чтобы посмотреть настройку блока сложения, достаточно дважды щелкнуть левой кнопкой мыши над его изображением в окне редактора задач (рис. 4.36).

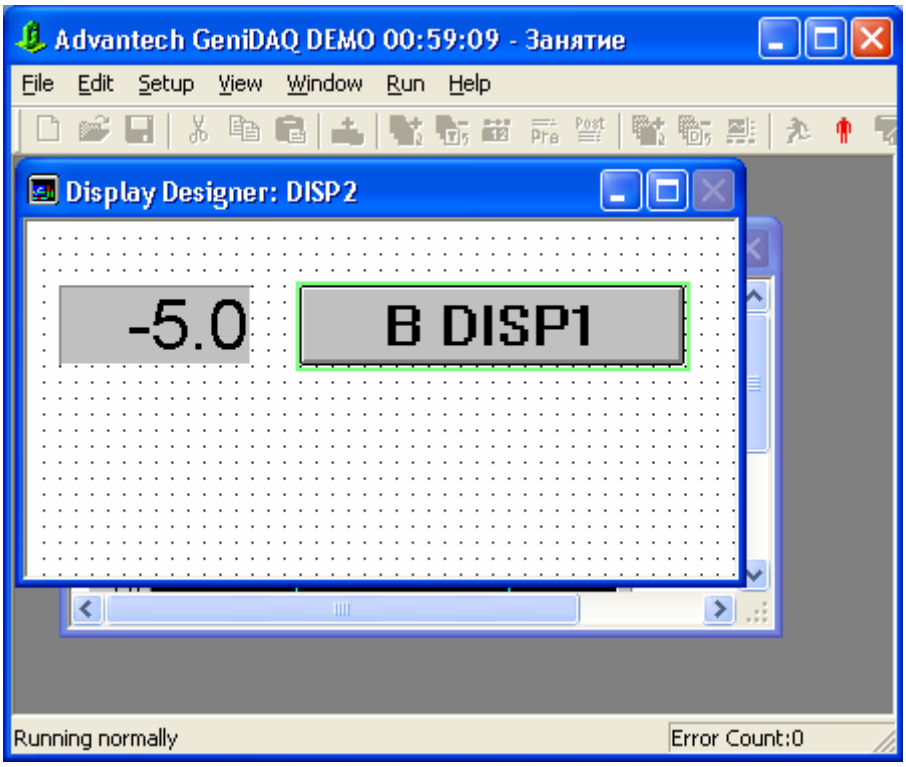

Рис. 4.27. Работа проекта после переключения окна отображения

5. *Пятое действие.* Аналогичным образом установите в окне редактора задач блок вычитания и установите связи между блоками аналогового ввода и блоком вычитания. Результирующий вид окна редактора задач показан на рис. 4.37, а настройка блока вычитания — на рис. 4.38.
6. *Шестое действие.* Сконфигурируйте график времени в соответствии с рис. 4.39 аналогично указанному в четвертом действии из занятия 1.

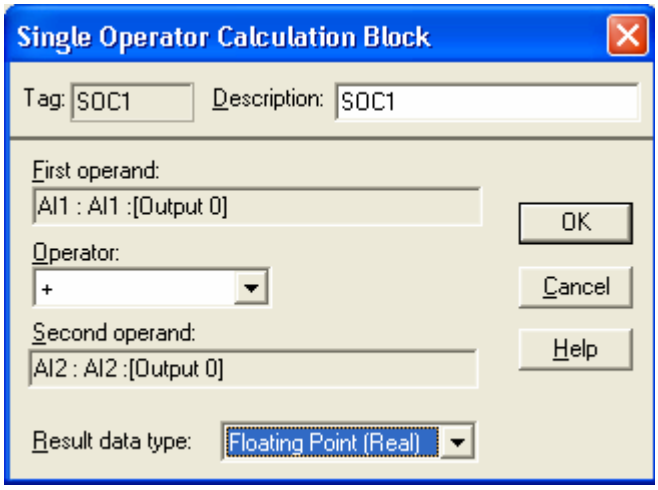

Рис. 4.28. Диалоговое окно настройки блока вычислений с одним оператором

Таблица 4.1. *Операторы и функции вычисления с единственным операторм*

Оператор	**Результат на выходе блока**
nop	Всегда 0
+	*Операнд1* + *Операнд2*
-	*Операнд1* - *Операнд2*
x	*Операнд1* * *Операнд2*
/	*Операнд1* / *Операнд2*
pow	*Операнд1* в степени *Операнд2*
mod*	Остаток целочисленного деления *Операнда1* на *Операнд2*
and*	Логическое И операндов
or*	Логическое ИЛИ операндов
xor*	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ операндов
max	Максимальное значение из двух операндов
min	Минимальное значение из двух операндов
>=	1, если *Операнд1* >= *Операнд2*; 0 — в противном случае

Оператор	Результат на выходе блока
<=	1, если *Операнд1* <= *Операнд2*; 0 — в противном случае
>	1, если *Операнд1* > *Операнд2*; 0 — в противном случае
<	1, если *Операнд1* < *Операнд2*; 0 — в противном случае
equ	1, если *Операнд1* равен *Операнду2*; 0 — в противном случае
neq	1, если *Операнд1* не равен *Операнд2*; 0 — в противном случае
abs	Абсолютная величина (модуль) *Операнда1*
not*	Логическое НЕ *Операнда1*
inv	Инверсия *Операнда1*
sqrt	Квадратный корень из *Операнда1*
log	Десятичный логарифм *Операнда1*
ln	Натуральный логарифм *Операнда1*
exp	*exp(Операнд1)*
jct	Оператор объединения (описание приведено ниже)

7. *Седьмое действие.* Сохраните созданный проект (стратегию) аналогично пятому действию из занятия 1.
8. *Восьмое действие.* После сохранения файлов созданного проекта имеется возможность немедленного запуска созданного проекта с помощью кнопки **Start** на панели инструментов (рис. 4.40). Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
9. *Девятое действие.* Посмотрите порядок выполнения функциональных блоков стратегии. Для этого включите команду **View | Order Layout** и на пиктограммах блоков аналогового ввода **AI1** и **AI2** будут показаны номера, определяющие порядок исполнения блоков в стратегии (рис. 4.41).

Для изменения порядка выполнения функциональных блоков стратегии выключите команду **View | Order Layout** и включите команду **Layout | Complete Reorder**. В результате этого в окне редактора задач появятся номера, определяющие порядок выполнения всех блоков стратегии, но в этом режиме можно изменить порядок выполнения только функциональных блоков. Для этого щелкните левой кнопкой мыши вначале над изображением блока **AI2**, а потом — над изображением блока **AI1** (рис. 4.42). Чтобы изменить порядок выполнения всех блоков стратегии выключите команду **Layout | Complete Reorder** и включите команду **Layout | Exchage Reorder**. Установите выполнение блоков в следующей последовательности: **AI2**, **AI1**, **SOC2** и **SOC1**. Для этого последовательно

щелкните левой кнопкой мыши над изображением блоков **SOC1** и **SOC2** (рис. 4.43). Выключите команду **Layout | Exchage Reorder**.

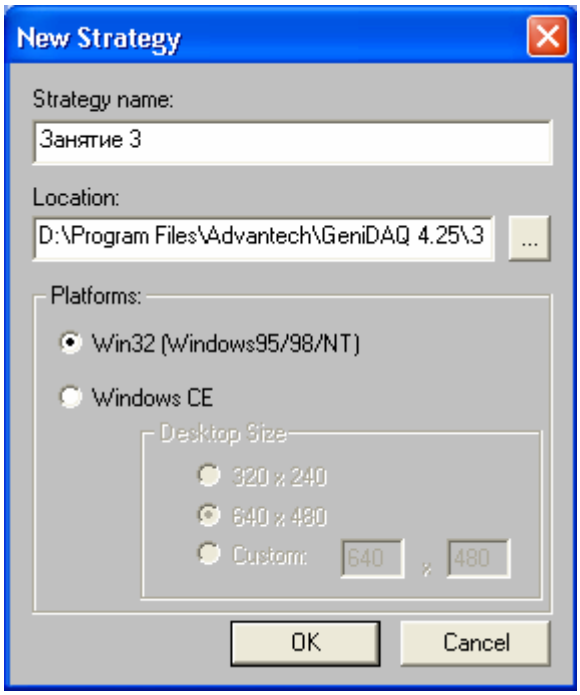

Рис. 4.29. Конфигурирование параметров стратегии

Запустите модифицированный проект после его сохранения с помощью кнопки **Start** на панели инструментов (рис. 4.44). Остановите работу проекта с помощью кнопки **Stop** на панели инструментов.

10. *Десятое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

4.3.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.8. Повтор создания и тестирования приложения "Занятие 3"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 3**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

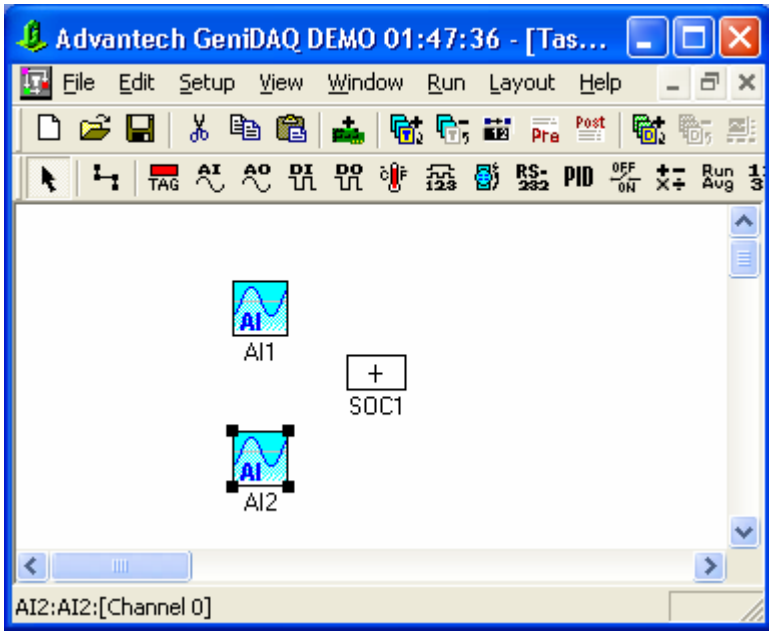

Рис. 4.30. Окно редактора задач с блоками аналогового ввода и блоком сложения

Упражнение 4.9

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 0.2 секунды. С помощью эмулятора и двух блоков аналогового ввода, помещенных в окно редактора задач, смоделируйте соответственно гармонический и случайный сигналы. С помощью блоков вычисления с одним оператором ослабьте случайный сигнал в десять раз и сложите его с гармоническим сигналом. В окне редактора форм отображения покажите указанные процессы на общем графике времени и с помощью цифровых индикаторов. Снабдить элементы отображения поясняющими надписями. Измените порядок выполнения функциональных блоков в окне редактора задач и пронаблюдайте вызванные этим изменения.

4.4. Занятие 4 "Использование инструментов рисования редактора форм отображения"

Цель занятия состоит в изучении правил применения инструментов рисования для создания пользовательских элементов отображения. Используются инструменты рисования и кнопка с двумя состояниями, имеющиеся на панели инструментов редактора задач. С помощью инструментов рисования создается некоторый графический объект. С помощью кнопки с двумя состояниями созданный объект изменяет свой цвет (элемент анимации).

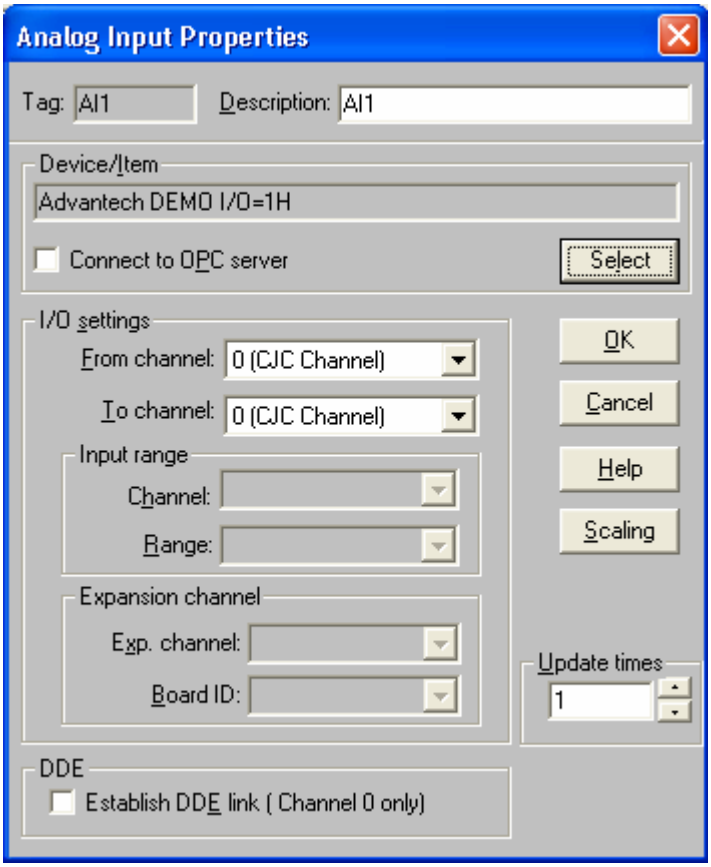

Рис. 4.31. Конфигурация блока аналогового **AI1**

4.4.1. Используемый инструментарий

Новым на данном занятии является использование в окне отображения *кнопки с двумя состояниями* и *инструментов рисования (эллипс, прямоугольник)* для создания графического объекта и его анимации**.**

Кнопка с двумя состояниями (*Binary Button*) может быть с помощью функционального блока *тег* (*Tag*) связана и использована для управления логическим состоянием любой двоичной или целочисленной переменной стратегии. Она позволяет передавать свое состояние из формы отображения в задачу. При нажатии кнопки с двумя состояниями производится передача логической 1 (или логического 0) связанному с ней блоку *тег*. Активизировать кнопку с двумя состояниями (нажать на кнопку) можно следующими способами:

- поместить курсор мыши на изображение кнопки в окне отображения и произвести щелчок левой клавишей мыши;

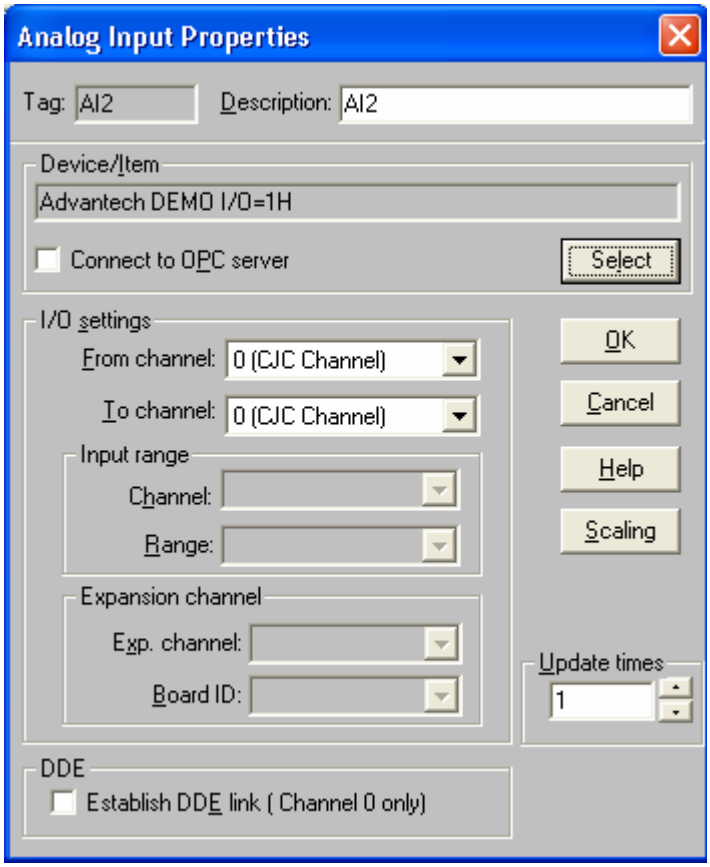

Рис. 4.32. Конфигурация блока аналогового **AI2**

- с помощью клавиши **Tab** на клавиатуре перевести фокус ввода на кнопку и нажать клавишу **Enter**;
- нажать клавишу, выбранную в комбинированном списке **Keyboard mapping** диалогового окна настройки параметров кнопки (рис. 4.45).

Размер и тип шрифта, используемого для отображения надписи на кнопке, могут быть установлены путем нажатия кнопки **Font** диалогового окна и выбора требуемого шрифта и его параметров из перечня шрифтов, зарегистрированных в ОС Windows. Поле *мег* (**Tag**) содержит идентификатор кнопки, который используется при установлении связи между кнопкой и другими элементами отображения либо функциональными блоками *мег* стратегии.

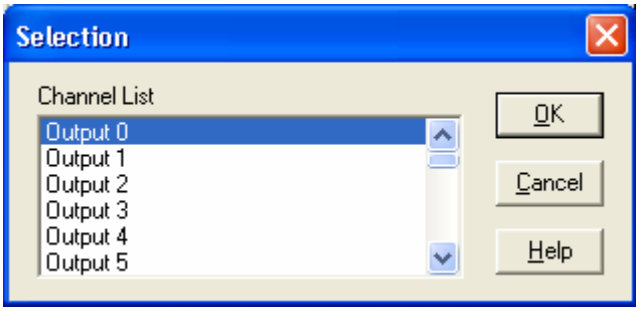

Рис. 4.33

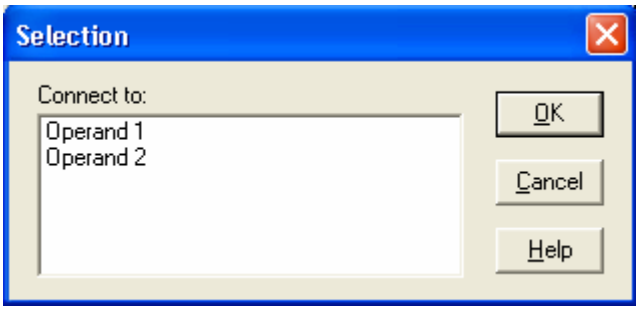

Рис. 4.34

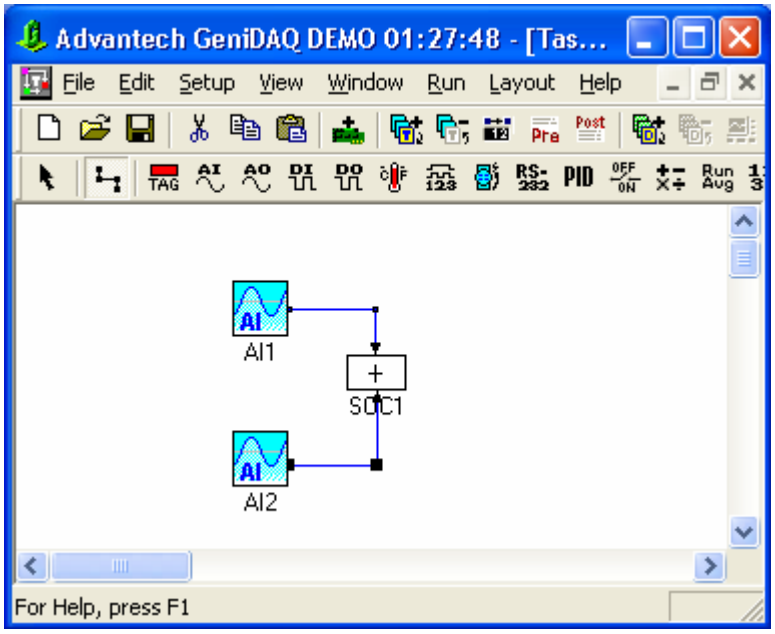

Рис. 4.35. Связь блоков аналогового ввода с блоком суммирования

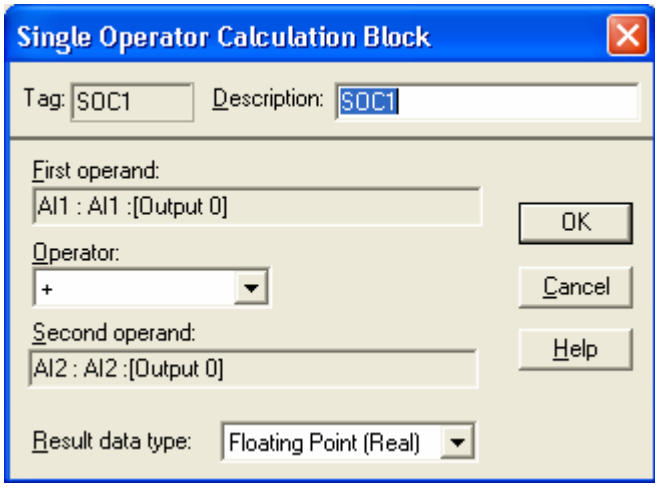

Рис. 4.36. Результирующая настройка блока суммирования

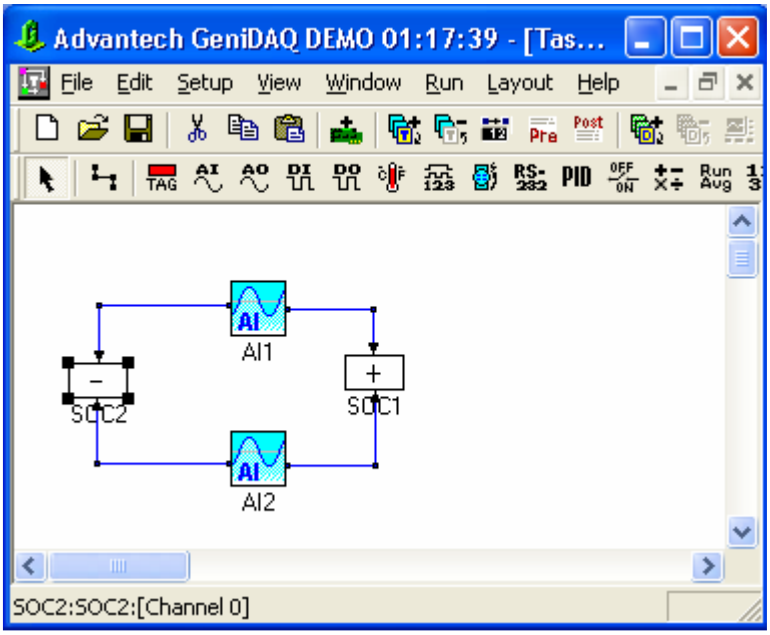

Рис. 4.37. Окно редактора задач с блоками аналогового ввода и блоками сложения и вычитания

Обратите особое внимание на то, что *связь между кнопкой и любыми функциональными блоками в задачах должна осуществляться через функциональный блок тег редактора задач*. Поле *надпись на кнопке* (**Label**) предназначено для ввода надписи, которая будет отображаться на кнопке в процессе исполнения стратегии.

Максимальное количество символов, из которых состоит надпись, составляет 30. Поле *режим функционирования* (**Operating style**) представлено комбинированным списком и используется для выбора режима функционирования кнопки.

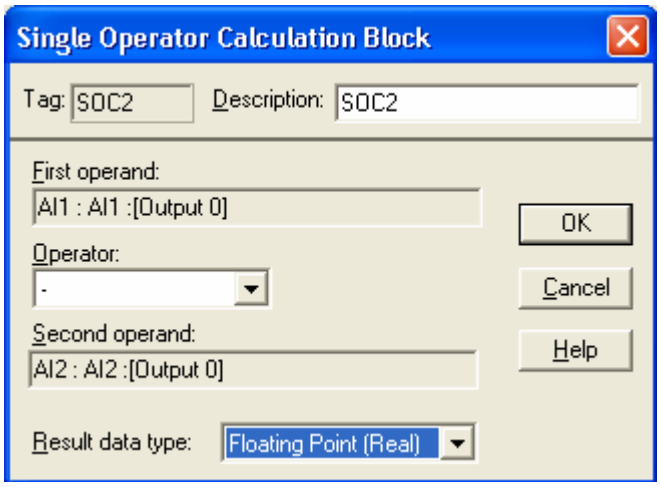

Рис. 4.38. Результирующая настройка блока вычитания

При выборе значения **On-Off** элемент управления будет выполнять функцию кнопки с фиксацией состояния после нажатия и отпускания с удержанием выходного значения для каждого состояния. При выборе значения **Monentary** элемент управления будет выполнять функцию кнопки без фиксации состояния (нажатие с мгновенным возвратом в отжатое состояние). Соответствующее значение на выходе удерживается только в течение времени, пока кнопка удерживается в нажатом состоянии. При выборе значения **Radio Button** элемент управления будет выполнять функцию кнопки, состояние которой зависит от состояния других связанных с ней кнопок. Данная кнопка носит название "радиокнопка". Поле *управление от клавиатуры* (**Keyboard mapping**), представленное комбинированным списком, позволяет выбрать клавишу на клавиатуре, с помощью которой возможно управлять состоянием кнопки при передаче ей фокуса ввода. Возможно использование функциональных клавиш от **F2** до **F9** или от **A** до **Z**. Поле *уровень привилегий* (**Privilege level**) предназначено для защиты функций управления положением коммутационных аппаратов, связанных с данной кнопкой. Уровень привилегий может принимать значения от 0 до 255, причем большему значению соответствует более высокий уровень привилегий. Таким образом, если для кнопки установлен уровень привилегий, равный 100, то нажать на данную кнопку смогут только пользователи с правами доступа от 100 и выше. Поля *цвет надписи на кнопке* (**Normal label color, Depressed label color**) позволяют установить цвета надписи на кнопке в нажатом и отпущенном состояниях. Реализована поддержка до 16 цветов. Переключатель *выходное значение* (**Output value**) предназначен для установки значений, передаваемых кнопкой другим элементам отображения и функциональным блокам *тег* стратегии при нажатии и отпускании.

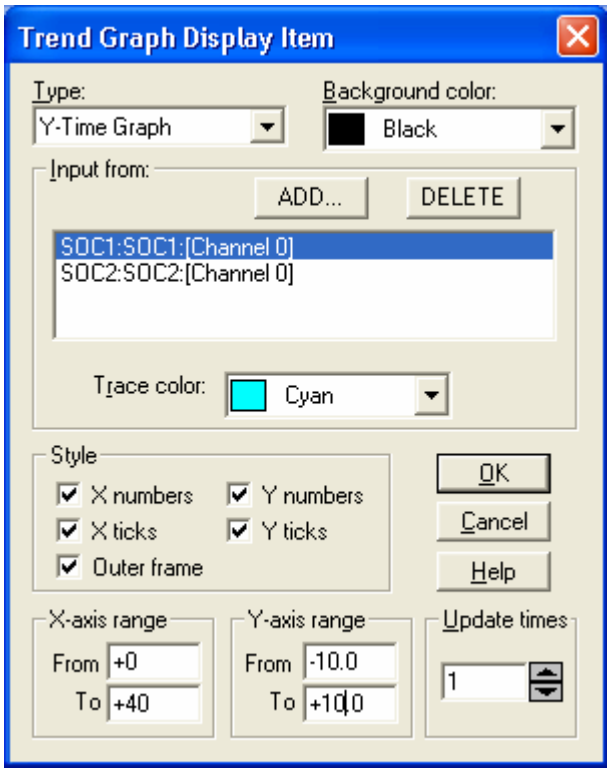

Рис. 4.39. Конфигурация графика времени

Имеется возможность передачи **0 в отжатом состоянии, 1 в нажатом состоянии** или **1 в отжатом состоянии, 0 в нажатом состоянии**. Независимый переключатель *звуковой сигнал при нажатии* (**Beep when pressed**) позволяет формировать звуковой сигнал при любом изменении состояния кнопки. Переключатель *автоподбор размера шрифта* (**Auto font sizing**) в отмеченном состоянии активизирует функцию автоматического выбора размера надписи при изменении размеров кнопки в процессе разработки стратегии.

Инструмент рисования "прямоугольник" предназначен для создания в окне экранной формы графического объекта прямоугольной формы, цвет которого определяется логическим состоянием на выходе присоединенного функционального блока стратегии (рис. 4.46).

Поле **Input from** содержит идентификатор выхода функционального блока, связанного с элементом отображения. Поле **Pen color** позволяет выбрать цвет рамки графического объекта. Возможен выбор одного из 16 цветов. Поле **Pen size** позволяет установить толщину рамки графического объекта.

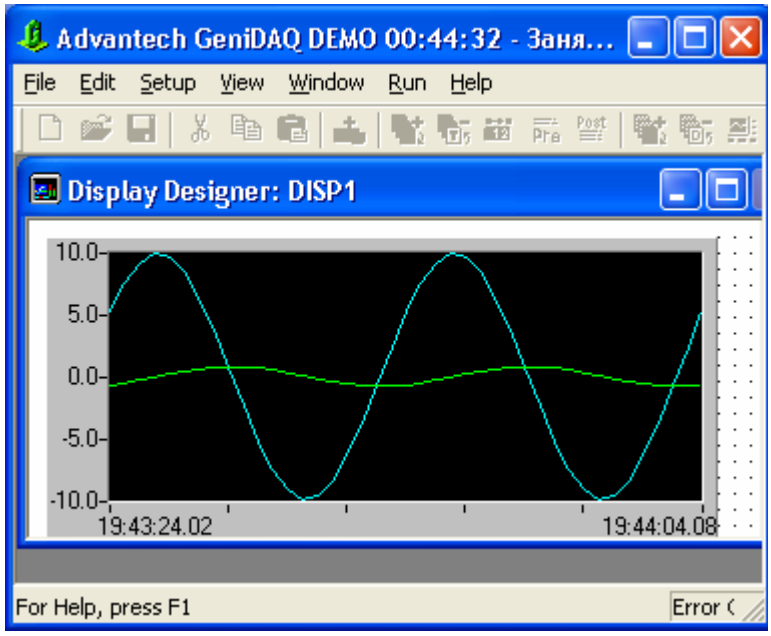

Рис. 4.40. Работа приложения сразу же после его запуска

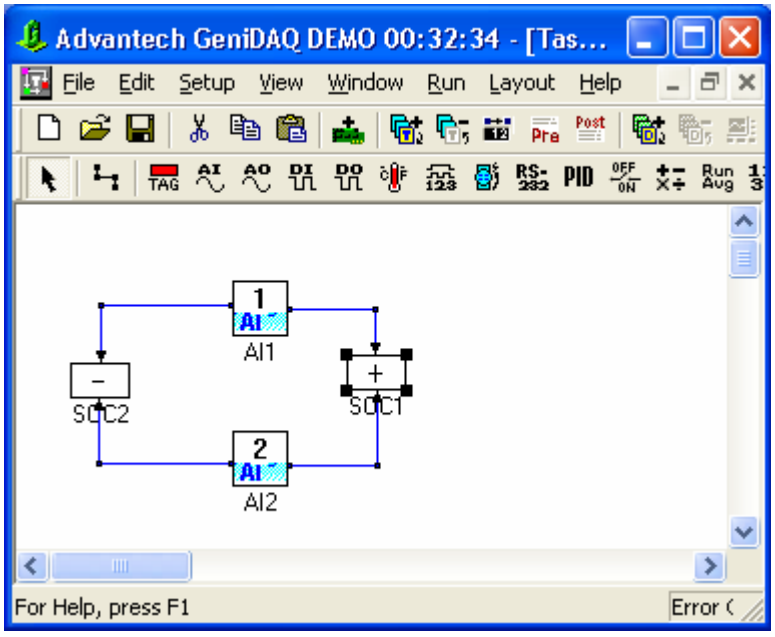

Рис. 4.41. Просмотр порядка выполнения функциональных блоков стратегии

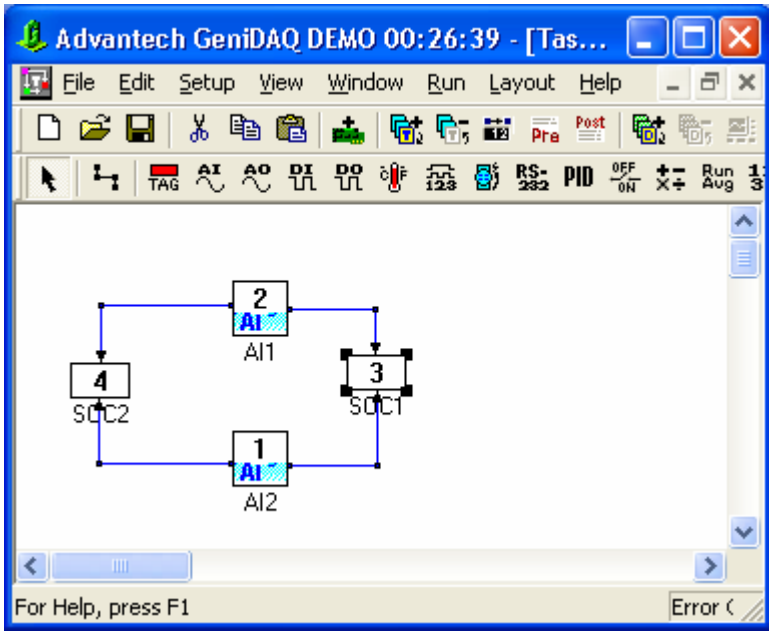

Рис. 4.42. Вид окна редактора задач после изменения порядка выполнения функциональных блоков

Толщина рамки задается в пикселях. Поле **Color when normal** позволяет выбрать цвет внутренней области графического объекта при поступлении на его вход логической единицы. Поле **Color when activated** позволяет выбрать цвет внутренней области графического объекта при поступлении на его вход логического нуля.

Инструмент рисования "эллипс". предназначен для создания в окне экранной формы графического объекта, имеющего форму эллипса, цвет которого определяется логическим состоянием на выходе присоединенного функционального блока стратегии (рис. 4.47).

Поле **Input from** содержит идентификатор выхода функционального блока, связанного с элементом отображения. Поле **Pen color** позволяет выбрать цвет рамки графического объекта. Возможен выбор одного из 16 цветов. Поле **Pen size** позволяет установить толщину рамки графического объекта. Толщина рамки задается в пикселях. Поле **Color when normal** позволяет выбрать цвет внутренней области графического объекта при поступлении на его вход логической единицы. Поле **Color when activated** позволяет выбрать цвет внутренней области графического объекта при поступлении на его вход логического нуля.

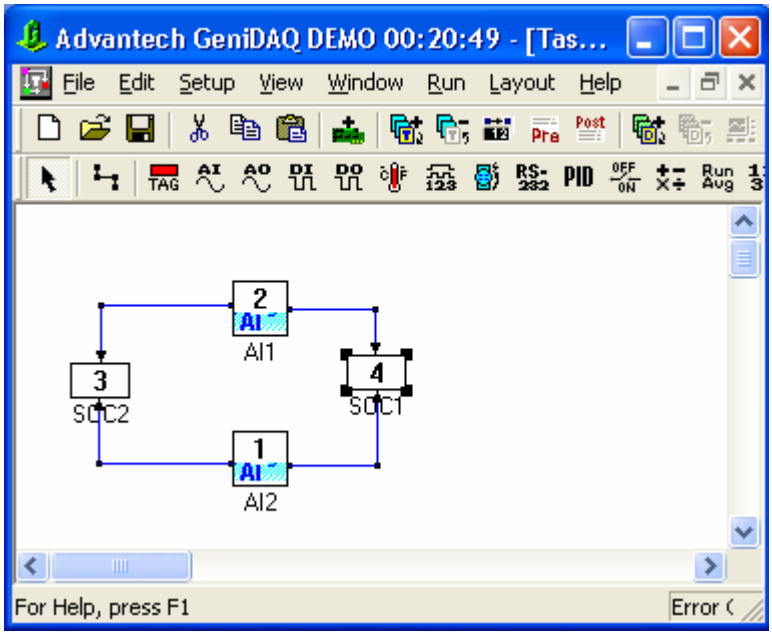

Рис. 4.43. Вид окна редактора задач после изменения порядка выполнения всех блоков стратегии

4.4.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.48.
2. *Второе действие*. Добавьте в окно редактора форм отображения кнопку с двумя состояниями **Binary Button**. Выполните над ней двойной щелчок левой кнопкой мыши и в появившемся диалоговом окне задайте параметры кнопки в соответствии с приведенным ранее рис. 4.45. Для сохранения параметров нажмите кнопку **OK**.
3. *Третье действие*. Произведите щелчок левой кнопкой мыши над пиктограммой инструмента рисования *эллипс* (**Oval Draw**) на панели инструментов редактора форм отображения. Переместите курсор в окно отображения, нажмите левую кнопку мыши и, не отпуская ее, нарисуйте круг. С помощью команд **Edit | Copy** и **Edit | Paste** скопируйте созданный круг (он появится в левом вернем углу окна отображения), уменьшите его размер. Для этого поместите курсор в угол рамки черного цвета и переместите курсор в направлении середины рамки. Поместите круг меньшего размера внутрь круга большего размера.

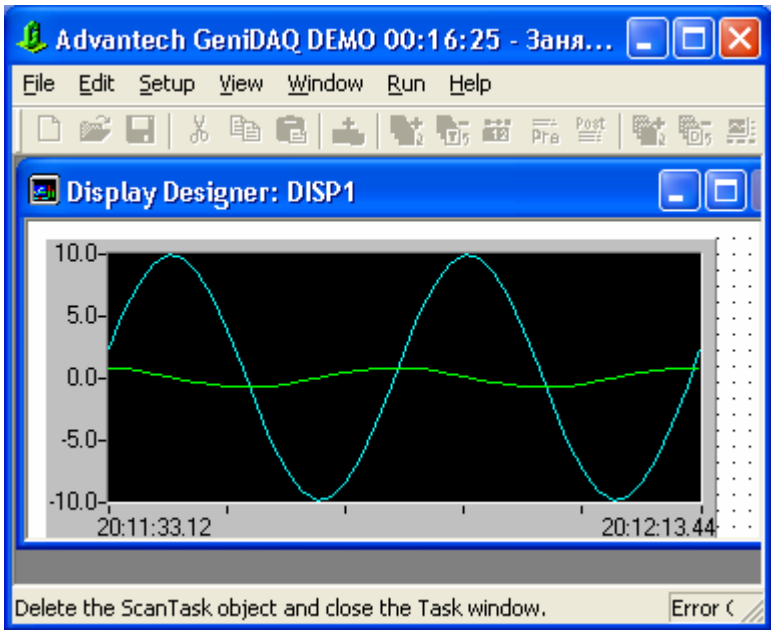

Рис. 4.44

Аналогичным образом создайте в окне отображения два графических примитива *прямоугольник* (**Rectangle Draw**). Расположите созданные графические примитивы таким образом, чтобы они образовали графическое изображение насоса. При необходимости используйте команды **Edit | Bring to Front** и **Edit | Send to Back**. Объедините графические примитивы один графический объект. Для этого выполните операцию переноса курсора для одновременного выделения всех примитивов, образующих графическое изображение насоса, и выполните команду **Edit | Make Object**. Окно отображения приобретет вид, показанный на рис. 4.49. Выполните двойной щелчок левой кнопкой мыши над изображением созданного графического объекта, в появившемся окне диалога нажмите кнопку **Select**, установите связь графического объекта с кнопкой с двумя состояниями в соответствии с рис. 4.50 и нажмите кнопку **OK**. Окно диалога приобретает вид, показанный на рис. 4.51. Нажатием кнопки **OK** закройте окно диалога.

4. *Четвертое действие*. Сохраните созданный проект (стратегию) аналогично тому, как это делалось на занятии 1. Запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.52). Понажимайте на кнопку с двумя состояниями — цвет графического объекта будет меняться при каждом нажатии на кнопку. Остановите работу проекта с помощью кнопки **Stop** на панели инструментов.
5. *Пятое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

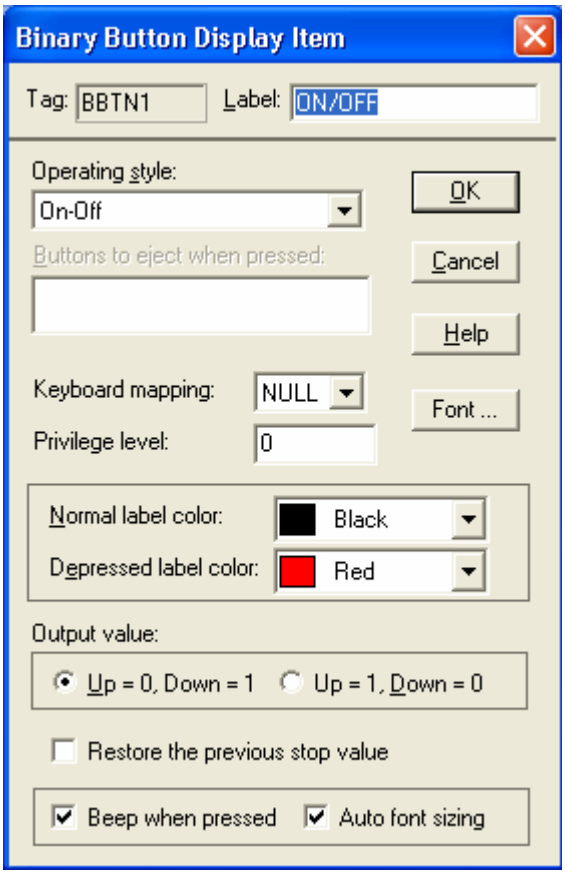

Рис. 4.45. Диалоговое окно конфигурирования кнопки с двумя состояниями

Замечание

Обратите внимание на то, что а частном случае окно редактора задач может не использоваться. Именно такая ситуация имела место в приложении **Занятие 4**.

4.4.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.10. Повтор создания и тестирования приложения "Занятие 4"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 4**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

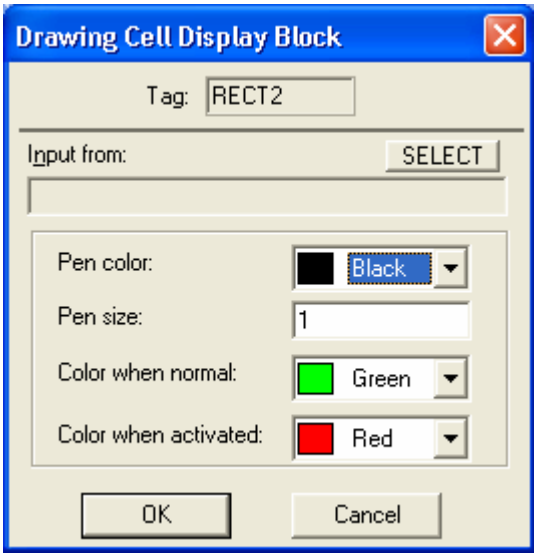

Рис. 4.46. Окно конфигурирования графического примитива **Прямоугольник**

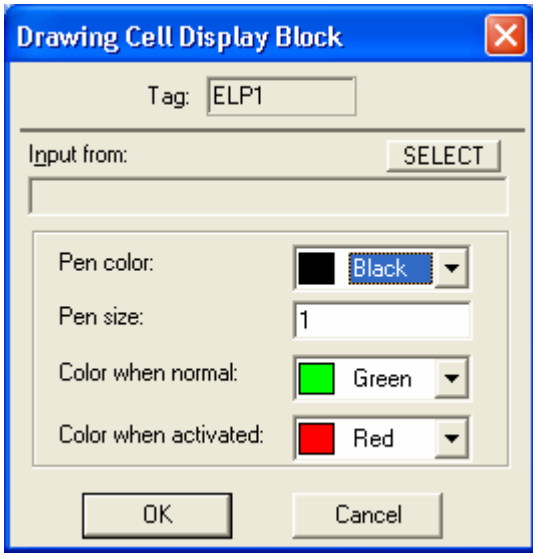

Рис. 4.47. Окно конфигурирования графического примитива **Эллипс**

4.5. Занятие 5 "Использование тега для связи между задачей и формой отображения"

Цель занятия состоит в изучении правил использования функционального блока *тег* (**Tag**) редактора задач для получения данных, вводимых пользователем с помощью какого либо элемента управления, используемого в окне отображения. Используются функциональные блок *тег* (**Tag**), блок *архива тревог* (**Alarm Log**) редактора задач и инструменты *инкрементный регулятор* (**Numeric Control**), *индикатор* (**Indicator**) и *текстовые строки* (**Text String**) редактора форм отображения.

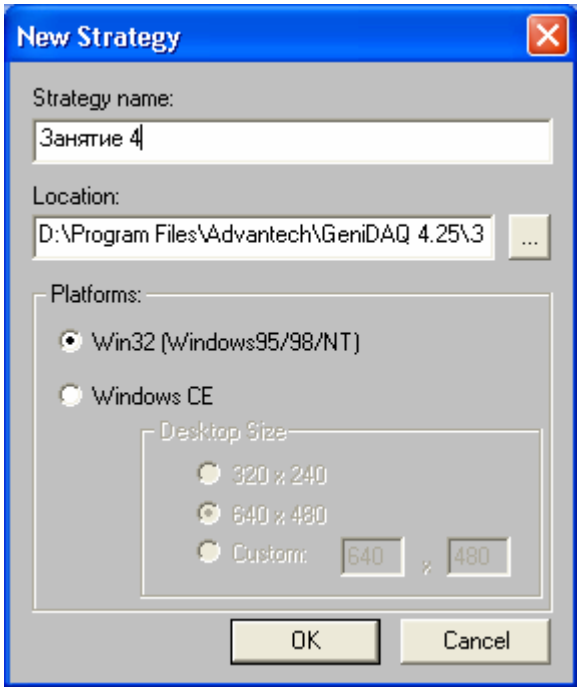

Рис. 4.48. Конфигурирование параметров стратегии

4.5.1. Используемый инструментарий

Для построения пользовательского *приложения*, выполняющего требуемые функции, используются перечисленные ранее функциональные блоки редактора задач и инструменты редактора форм отображения. Дадим их краткую характеристику.

Функциональный блок тег (Tag) редактора задач предназначен для установления связи между элементами управления редактора форм отображения с одной стороны и функциональными блоками редактора задач или виртуальными тегами с другой стороны (рис. 4.53). На данном занятии тег используется для связи управляющего элемента из окна отображения с функциональным блоком редактора задач.

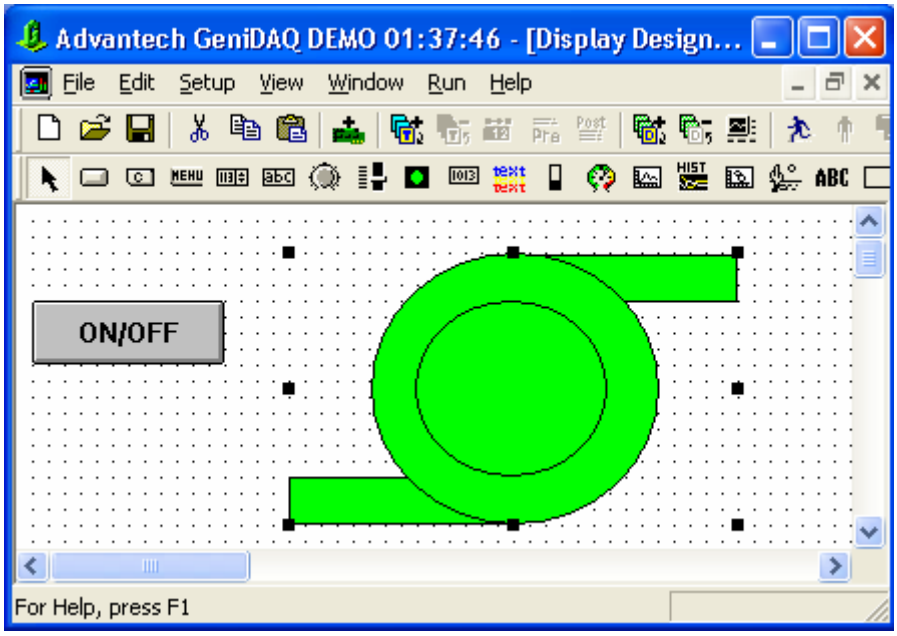

Рис. 4.49. Вид окна отображения после создания графического объекта (насоса)

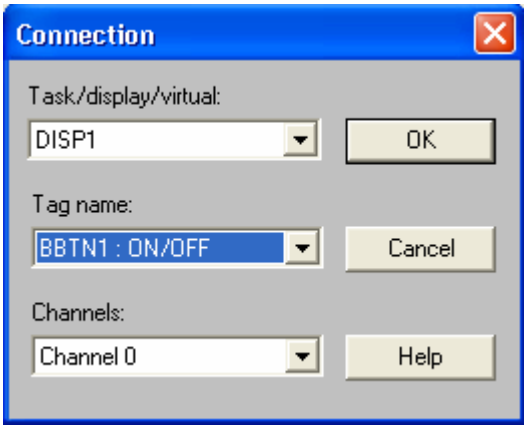

Рис. 4.50. Установление связи графического объекта с кнопкой

Блок *тег* имеет непосредственный доступ к значениям элемента управления редактора форм отображения или виртуального тега центра обработки данных, указанного в полях группы **Attaching to**. На рис. 4.53 таким элементом является управляющий элемент **NCTL1** окна отображения. Получаемые значения *тег* передает функциональному блоку окна редактора задач, присоединенному к его выходу.

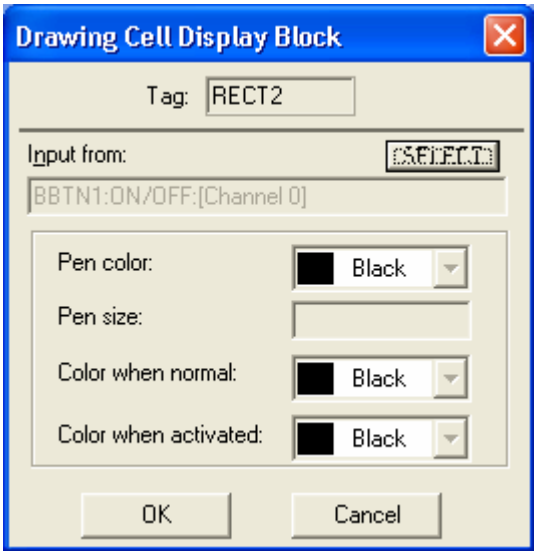

Рис. 4.51. Конфигурирование параметров графического объекта

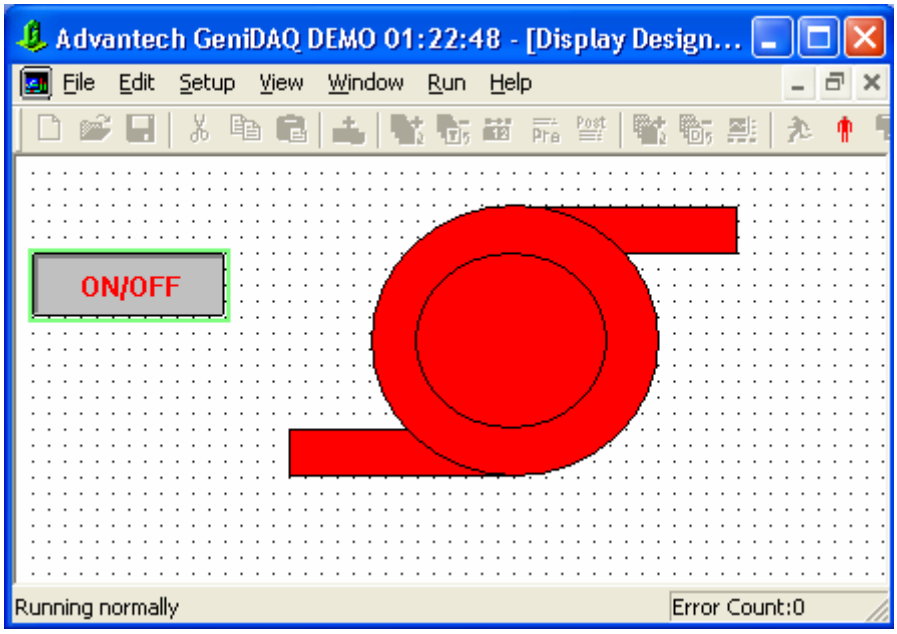

Рис. 4.52. Работа приложения

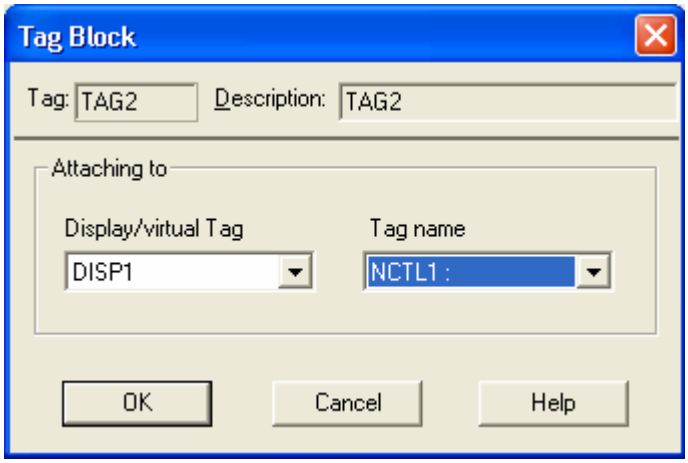

Рис. 4.53. Конфигурирование функционального блока *тег*

Функциональный блок архива тревог (Alarm Log) редактора задач предназначен для сохранения в архиве информации о зафиксированных аварийных событиях, связанных с сигналом, поступающим на вход блока архива тревог. Блок имеет вход и выход. Тревоги можно фиксировать в файле архива событий (..\GeniDAQ 4.25\genieDAQ.elf). Для включения такого режима необходимо выполнить команду **Setup | Runtime Preference...** и в появившемся окне диалога установить флаг **Enable event log**. Сообщения об аварийных событиях могут в процессе исполнения стратегии отображаться в окне *журнала событий* (окно делается видимым при выполнении команды **View | Event Log**), когда значение на входе блока попадает в следующие диапазоны:

1. Выше верхнего предельного значения.
2. Между максимальным и верхним предельным значениями.
3. Между минимальным и нижним предельным значениями.
4. Ниже нижнего предельного значения.

На выходе блока присутствует целое число, соответствующее событию, зафиксированному блоком. Это позволяет представлять информацию об аварийных событиях в графической форме путем установления связи между блоком архива тревог и элементом отображения *индикатор*. Зависимость значений на выходе блока от зафиксированного блоком события приведена в табл. 4.4.

***Таблица 4.2.** Зависимость значения на выходе блока тревог от зафиксированного им события*

Аварийное событие (значение на входе блока)	**Значение на выходе блока**
Выше верхнего предельного значения	4

Аварийное событие (значение на входе блока)	Значение на выходе блока
Между максимальным и верхним предельным значениями	2
Между максимальным и минимальным значениями	0
Между минимальным и нижним предельным значениями	1
Ниже нижнего предельного значения	3

Диалоговое окно конфигурирования блока архива тревог показано на рис. 4.54. При исполнении стратегии имеется возможность подтверждения восприятия оператором событий, зафиксированных блоком, путем двойного щелчка левой клавишей мыши в окне *журнала событий*. До подтверждения информация об аварийном событии отображается в окне *журнала событий* в виде строки красного цвета. Функциональный блок *архива тревог* имеет один вход, на который может поступать сигнал от другого функционального блока стратегии. Значение на входе проверяется блоком на вхождение в пределы, заданные в группе параметров *значения параметров тревоги* (**Alarm Settings**) диалогового окна настройки параметров блока в процессе разработки стратегии. На выходе блока присутствует целое число, соответствующее событию, зафиксированному блоком (см. табл. 4.2).

Инкрементный регулятор (Numeric Control) редактора форм отображения может быть помещен в окно отображения и связан с входной переменной функционального блока задачи, входящей в стратегию, и/или элементами отображения. Имеется возможность установки требуемых размеров регулятора. Инкрементный регулятор предназначен для ввода оператором числовых значений с помощью клавиатуры или мыши и передачи введенных значений связанному функциональному блоку стратегии, что позволяет реализовывать функции оперативного диспетчерского управления. Диалоговое окно конфигурирования инкрементного регулятора показано на рис. 4.55. Вводимые и отображаемые значения могут быть целого или вещественного типа (групповое окно **Data type**). Формат вводимого значения (количество цифр после десятичной точки — **Digits of precision**) может быть установлен только для вещественных чисел. Тип и размер используемого шрифта выбираются путем нажатия кнопки **Font...** Поле **Privilege level** предназначено для защиты функций управления, связанных с инкрементным регулятором. Уровень привилегий может принимать значения от 0 до 255, причем большему значению соответствует более высокий уровень привилегий. Таким образом, если для регулятора установлен уровень привилегий, равный 100, то изменить значение на его выходе смогут пользователи с правами доступа от 100 и выше. Поле **Initial value** определяет значение на выходе инкрементного регулятора при запуске стратегии на исполнение. Поле **Step value** определяет величину уменьшения/увеличения значения на выходе регулятора при однократном щелчке левой клавишей мыши на кнопках уменьшения/увеличения, расположенных справа от области отображения значения на выходе инкрементного регулятора. Поля **High limit/Low limit** предназначены для установки диапазона изменения значения на выходе инкрементного регулятора.

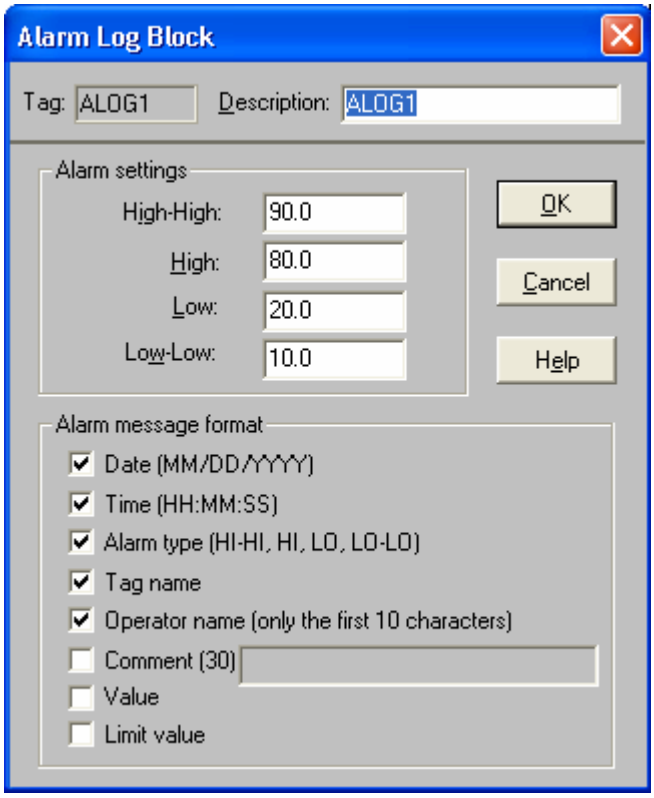

Рис. 4.54. Конфигурирование функционального блока *архива тревог*

Индикатор (Indicator) редактора форм отображения представляет собой единичный индикатор, предназначенный для отображения логического состояния связанного с ним дискретного выхода функционального блока стратегии. Индикатор переводится в состояние *включено* при появлении логической единицы (ненулевое целое значение) на выходе связанного с ним функционального блока и в состояние *выключен* при появлении логического нуля (нулевое целое значение) на выходе связанного с ним функционального блока. Размеры, форма и цвет индикатора в разных состояниях могут быть установлены с помощью диалогового окна настройки параметров индикатора (рис. 4.56).

Поле **Input from** устанавливает связь индикатора с одной из переменных задачи, входящей в стратегию. Перед началом настройки параметров индикатора необходимо установить связь с функциональным блоком, логическое состояние на выходе которого предполагается отображать. Для этого в окне настройки индикатора следует нажать кнопку **SELECT** и в появившемся окне выбрать идентификатор задачи, идентификатор функционального блока и номер присоединяемого выхода функционального блока.

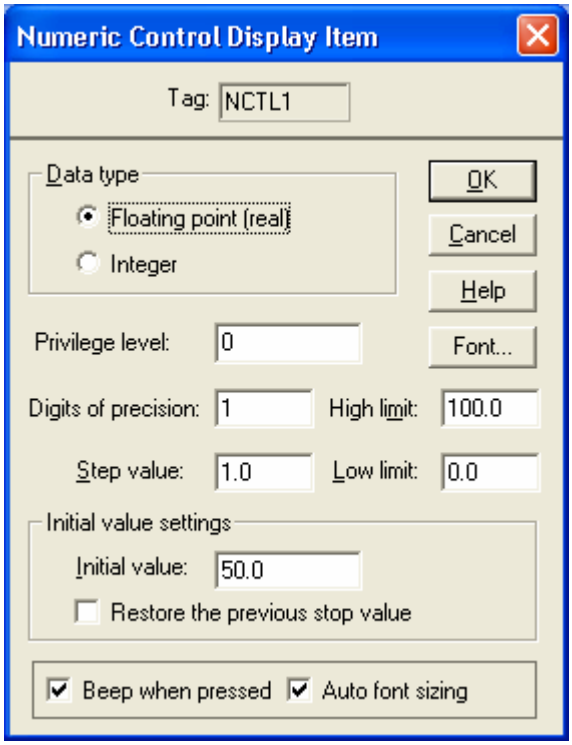

Рис. 4.55. Конфигурирование *инкрементного регулятора*

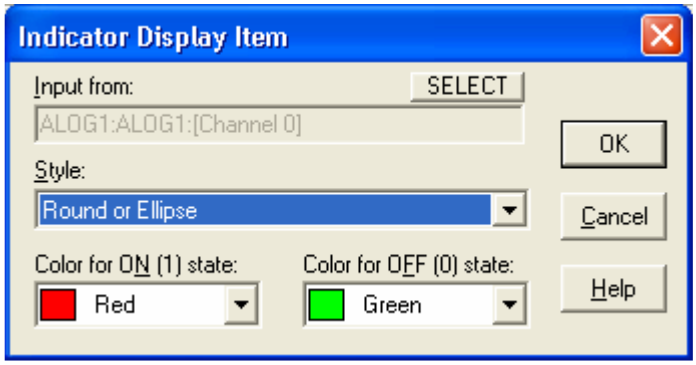

Рис. 4.56. Конфигурирование *индикатора*

При запуске стратегии на исполнение логическое состояние на выходе присоединенного функционального блока будет отображаться соответствующим цветом индикатора. Поле **Style** предназначено для выбора формы индикатора. Индикатор может иметь прямоугольную (квадратную) форму или форму эллипса (круга). Поля **Color for ON (1) state/Color for OFF (0) state** предназначены для

выбора цвета индикатора во включенном и выключенном состояниях. Реализована поддержка до 16 цветов.

Элемент отображения *Текстовая строка (Text string)* редактора форм отображения не имеет средств связи с функциональными блоками и другими элементами отображения/управления стратегии и предназначен для вывода на экран монитора статической символьной строки, которая определяется на этапе разработки стратегии. Имеется возможность выбора типа и размера шрифта, которым будет выводиться текстовая строка (рис. 4.57).

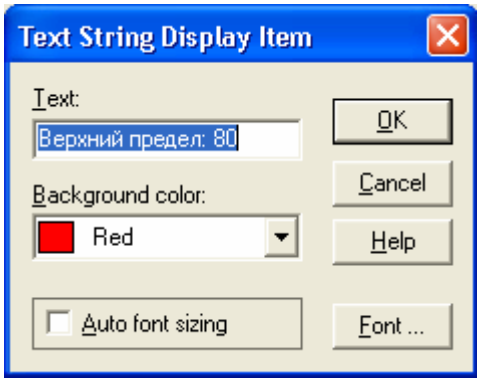

Рис. 4.57. Конфигурирование *текстовой строки*

Поле **Text** предназначено для ввода строки символов, которая будет выводиться во внутренней области элемента отображения во время исполнения стратегии. Поле **Background color** позволяет выбрать цвет внутренней области элемента отображения. Если требуется установить цвет и размер шрифта, следует снять отметку с флажка **Auto font sizing** и нажать кнопку **Font...** диалогового окна, после чего выбрать требуемый вид, размер и цвет в появившемся диалоговом окне настройки параметров шрифта.

4.5.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.57.
2. Выполните *второе действие* из занятия 1. В нашем примере также задан период запуска приложения 100 миллисекунд.
3. *Третье действие*. Добавьте в окно отображения инкрементный регулятор и настройте его в соответствии с рис. 4.55. Добавьте в окно отображения индикатор и настройте его в соответствии с рис. 4.56. Добавьте в это же окно три текстовых строки и настройте их в соответствии с рис. 4.57, 4.59 и 4.60. В результате окно отображения приобретает вид, показанный на рис. 4.61.

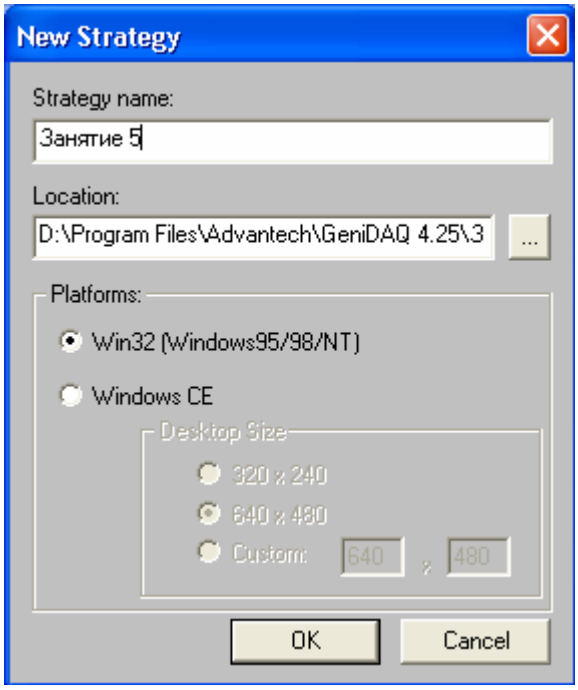

Рис. 4.58. Конфигурирование приложения

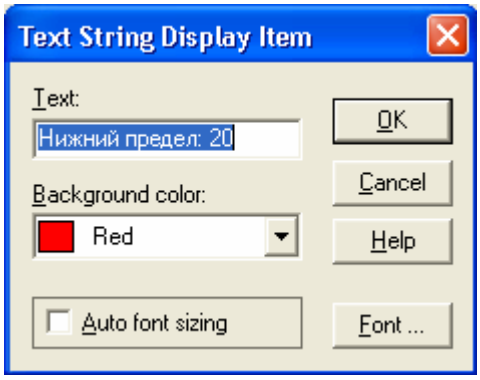

Рис. 4.59. Конфигурирование *текстовой строки*

4. *Четвертое действие*. Выдвиньте на передний план окно редактора задач и разместите в нем функциональные блоки *тег* и блок *архива тревог*. Произведите двойной щелчок левой кнопкой мыши над блоком *тег* и появится диалоговое окно его настройки (см. выше рис. 4.53). Сконфигурируйте блок *архива тревог* в соответствии с рис. 4.54. Соедините выход *тега* со входом блока архива *тревог*. Окно редактора задач приобретет вид, показанный на рис. 4.62.

Рис. 4.60. Конфигурирование *текстовой строки*

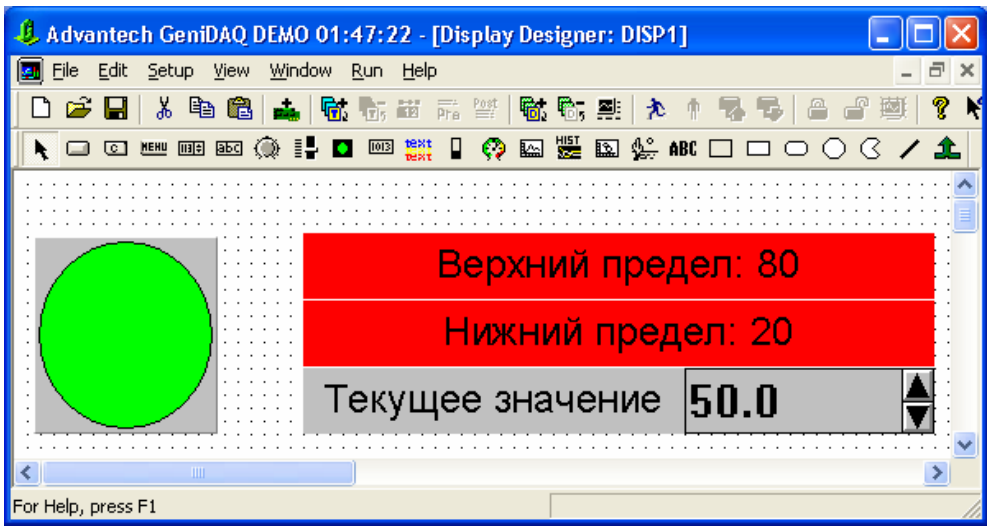

Рис. 4.61. Вид окна отображения

5. *Пятое действие*. Сохраните созданный проект (стратегию) аналогично пятому действию из занятия 1. Выполните команду **Setup | Runtime Preference...** и в появившемся окне диалога установите флаг **Enable event log**. Запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.63). Для отображения журнала событий выполните команду **View | Event Log** (рис. 4.64). Используя инкрементный регулятор задайте текущие значения 81 и 19 и обратите внимание на изменение состояния индикатора и связанного с этим появление звуковых сигналов. Подтвердите аварийные сообщения двойным щелчком левой клавиши мыши на строках журнала событий, отображаемых красным цветом. Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.

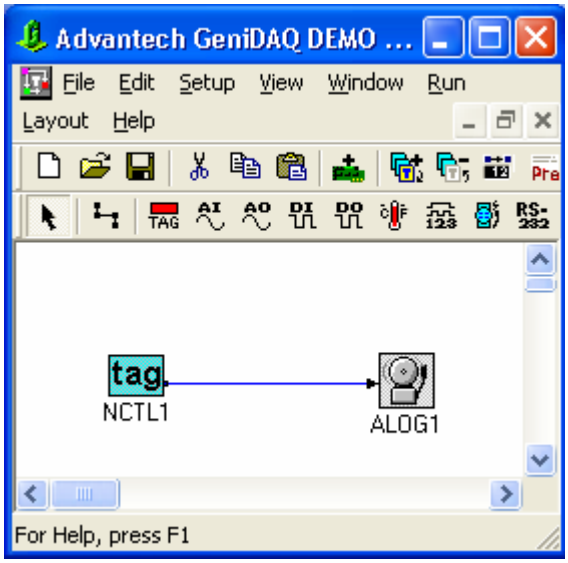

Рис. 4.62. Вид окна редактора задач

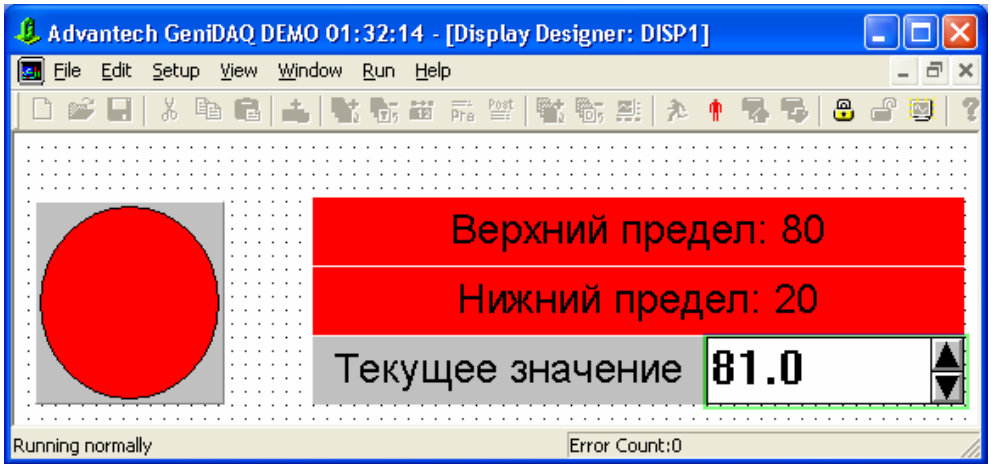

Рис. 4.63. Демонстрация работы проекта в окне отображения

6. *Шестое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

Рис. 4.64. Вид журнала событий

4.5.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.11. Повтор создания и тестирования приложения "Занятие 5"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 5**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Упражнение 4.12

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 0.2 секунды. С помощью эмулятора и блока аналогового ввода, помещенного в окно редактора задач, смоделируйте гармонический сигнал. Поместите в окно редактора задач функциональный блок тревог и подайте на него выходной сигнал блока аналогового ввода. В окне отображения с помощью индикатора отобразите состояние гармонического сигнала (для диапазона High-Low зеленым цветом, а в остальных случаях — красным). Подключите журнал событий и подтвердите получение аварийных сигналов. Снабдить элементы отображения поясняющими надписями.

Упражнение 4.13

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 1 секунда. Сконфигурируйте окно отображения в соответствии с рис. 4.65. В окне редактора задач используйте блок архива тревог и сконструируйте окно таким образом, чтобы в процессе работы обеспечить следующую последовательность активизации индикаторов: 3-2-1-2-3-4-5-4-3 и т. д.

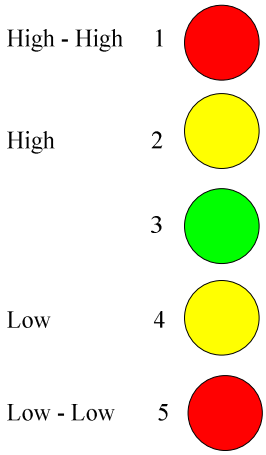

Рис. 4.65. Вид окна отображения (цвета индикаторов: 1, 5 — красный, 2, 4 — желтый, 3 — зеленый, альтернативный цвет — белый)

Упражнение 4.14

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 1 секунда. Сконфигурируйте окно отображения в соответствии с рис. 4.65. В окне редактора задач используйте блок архива тревог и сконструируйте окно таким образом, чтобы в процессе работы обеспечить следующую последовательность активизации индикаторов: 3-4-5-4-3-2-1-2-3 и т. д.

Упражнение 4.15

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 1 секунда. Сконфигурируйте окно отображения в соответствии с рис. 4.66. В окне редактора задач используйте блок архива тревог и сконструируйте окно таким образом, чтобы в процессе работы обеспечить следующую последовательность активизации индикаторов: 2-1-2-3-2-1 и т. д.

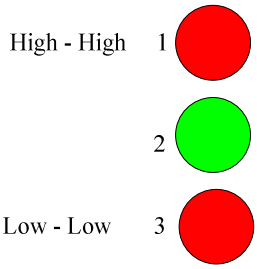

Рис. 4.66. Вид окна отображения (цвета индикаторов: 1, 3 — красный, 2 — зеленый, альтернативный цвет — белый)

Упражнение 4.16

Разработайте стратегию, обладающую следующими возможностями. Задайте период запуска задачи 1 секунда. Сконфигурируйте окно отображения в соответствии с рис. 4.66. В окне редактора задач используйте блок архива тревог и сконструируйте окно таким образом, чтобы в процессе работы обеспечить следующую последовательность активизации индикаторов: 2-3-2-1-2-3 и т. д.

Замечание

Рассматриваемые далее учебные примеры используют программы, написанные на языке Visual Basic for Applications (VBA). Поэтому следующий раздел главы посвящен рассмотрению языка VBA.

4.6. Использование языка VBA в SCADA-системах

Язык *VBA* широко применяется в SCADA-системах в качестве инструментального средства программирования. Далее последовательно приводится краткое описание основных средств VBA, не зависящих от среды его использования [14, 15], и рассматриваются средства привязки языка VBA к SCADA-системе GeniDAQ (фирма Advantech, США).

4.6.1. Средства языка VBA, инвариантные к среде использования (Microsoft Visual Basic 6.3)

Язык VBA, в отличие от языка Visual Basic (VB), не является языком объектно-ориентированного программирования в строгом смысле этого слова, но в нем широко используются элементы объектно-ориентированного подхода и связанные с ним понятия. VBA — подмножество языка VB, которое включает почти все средства создания приложений, структуры данных, управляющие структуры и возможность создания пользовательских типов данных.

Язык VBA является языком визуального и событийно управляемого программирования. Он позволяет создавать нестандартные диалоговые окна с набором элементов управления, процедуры, обрабатывающие события, возникающие при тех или иных действиях системы и пользователя. Проект (программа) на языке VBA, в отличие от программ на других языках, — результат побочной деятельности по созданию некоторого объекта (документа). Более того, проект VBA нельзя создать независимо от какого-либо объекта, даже если никакие свойства этого объекта не используются.

Язык VBA имеет графическую интегрированную среду разработки и реализует концепцию визуального программирования, управляемого событиями. Одной из важных областей применения языка VBA является его использование для программирования алгоритмов управления и других целей в SCADA-системах. Применительно к этому и ведется дальнейшее изложение материала.

4.6.1.1. Типы данных

В языке VBA имеются следующие типы данных: **Byte** (байт), **Boolean** (логический), **Integer** (целое), **Long** (длинное целое), **Single** (с плавающей точкой обычной точности), **Double** (с плавающей точкой двойной точности), **Currency** (денежный), **Decimal** (масштабируемое целое), **Date** (даты и время), **Object** (объект), **String** (строка), **Variant** (тип используется по умолчанию) — тип данных, определяемый пользователем, а также специальные типы объектов (табл. 4.3).

Таблица 4.3. *Типы данных языка VBA*

Тип	Размер в байтах	Диапазон значений
Boolean	2	**True** или **False**
Byte	1	От 0 до 255 (беззнаковое)
Date	8	От 0:00:00 1 января 0100 г. до 23:59:59 31 декабря 9999 г.
Currency	8	Характеризует денежную величину и принимает значения от 922 337 203 685 477.580 8 до -922 337 203 685 477.580 7
Decimal	12	От 0 до +/-79 228 162 514 264 337 593 543 950 335 без десятичной точки. От 0 до +/-7.922 816 251 426 433 759 354 395 033 5 с 28 знаками справа от десятичной точки; наименьшее ненулевое число +/-0.0000000000000000000000000001 (+/-1E-28)
Double (double precision floating point)	8	От -1.797 693 134 862 315 70E+308 до 4.940 656 458 412 465 44E-324 для отрицательных значений. От 4.940 656 458 412 465 44E-324 до 1.797 693 134 862 315 70E+308 для положительных значений
Integer	2	От -32 768 до 32 767
Long (long integer)	4	От -2 147 483 648 до 2 147 483 647
Object	4	Переменная объектного типа, содержащая ссылку на объект (объект может быть целого типа)
Single (single precision floating point)	4	От -3.402 823 5E+38 до -1.401 298E-45 для отрицательных значений. От 1.401 298E-45 до 3.402 823 5E+38 для положительных значений
String (variable length)	Реализация зависит от платформы	От 0 примерно до 2 млрд. двухбайтовых юникод-символов

Тип	Размер в байтах	Диапазон значений
Variant	Для переменной любого типа	Для переменной любого типа

Переменные и константы типа **Boolean** хранятся как 16-битные числа, но могут принимать только значения **True** или **False**. Используйте эти ключевые слова, чтобы присвоить одно из двух состояний переменной типа **Boolean**. Значение по умолчанию для переменных типа **Boolean** — **False**. Когда числовые типы данных приводятся к типу **Boolean**, 0 становится **False**, а все остальные значения — **True**. И наоборот, когда значение с типом **Boolean** приводится к числовому типу, **False** становится 0, а **True** — 1.

4.6.1.2. Переменные

В языке VBA *переменная* (variable) используется для временного хранения данного в оперативной памяти. Переменная должна быть *определена* прежде, чем ее можно использовать. Определение переменной производится при помощи операторов **Dim**, **Private**, **Static** или **Public**, которые также определяют и *область видимости* (область действия) переменной. Например, следующий оператор объявляет целую переменную (тип **Integer**):

```
Dim N As Integer
```

Если тип при определении переменной опущен, то по умолчанию переменная получает тип Variant. Например, следующий оператор определяет переменную x типа **Variant**:

```
Dim x
```

Тип переменной или константы можно также определить непосредственно при ее первом вхождении в текст программы с помощью *специальных символов в конце идентификаторов*: % — для **Integer**, & — для **Long**, ! — для **Single**, # — для **Double**, @ — для **Currency**, $ — для **String**. Например, идентификатор Primer$ является идентификатором переменной типа **String**. Однако подобная практика не соответствует хорошему стилю программирования. Приведем еще несколько примеров:

```
IntegerVar% = 123           ' Переменная типа Integer
DoubleVar# = 123.45         ' Переменная типа Double
Const Integerconst% = 123   ' Константа типа Integer
```

> ***Замечание***
> Если специальный символ в конце идентификатора отсутствует и переменная не определена явным образом, то такой идентификатор соответствует переменной с типом **Variant**.

4.6.1.3. Задание времени жизни и области видимости переменных

Время жизни и область видимости определяют соответственно место использования переменной в приложении, а также время ее существования после создания переменной (время жизни).

Область видимости переменной определяет часть кода, которая "знает" о существовании данной переменной. Так, в частности, при определении переменной в процедуре или функции получить или изменить ее значение можно только из кода этой процедуры или функции. Существуют три типа области видимости переменной.

- Переменные уровня процедуры или функции распознаются только в процедуре или функции, в которой они описаны. Они определяются в теле процедуры или функции при помощи *операторов* ***Dim, Private*** *или* ***Static*** (по умолчанию **Private**). Такие переменные называются *локальными*. При выходе из процедуры такие переменные становятся недоступными. При выходе из процедуры или функции, в которой при помощи операторов **Dim** или **Private** была определена локальная переменная, такая переменная "уничтожается", т. е. освобождается память, занятая такой переменной. Сказанное не относится к локальным переменным, определенным с помощью служебного слова **Static** — их время жизни максимально.
- Переменные уровня *контейнера* используются только в контейнере (форме, модуле, классе), в котором они описаны, но не в других контейнерах данного проекта. Контейнерные переменные определяются обязательно при помощи служебного слова **Private** в области описания контейнера, т. е. перед определением процедур и функций.
- Переменные уровня модуля, описанные при помощи *оператора* ***Public***, являются доступными внутри данного проекта, а также других проектов, которые ссылаются на данный проект. Такие переменные называются *глобальными (открытыми)*. Время жизни открытой переменной совпадает со временем работы программы. *Глобальные переменные могут использоваться только в модулях*.

Еще раз отметим, что закрытая (**Private**) переменная сохраняет свое значение, только, пока активна процедура, функция или контейнер, в которых эта переменная определена. При их завершении значение переменной теряется и при повторном запуске переменную надо заново разместить в памяти и инициализировать. Переменные, описанные с использованием оператора **Static** или **Public**, сохраняют свое значение в течение всего времени работы программы.

> ***Примечание***
> Синтаксис определения переменных с помощью операторов **Static, Private** и **Public** аналогичен синтаксису определения переменных с помощью оператора **Dim** (вместо служебного слова **Dim** можно использовать одно из служебных слов **Static, Private** или **Public**).

4.6.1.4. Опция явного определения переменных

Для обязательного определения всех переменных в начале модуля, в так называемой области модуля **General Declarations**, надо поместить *директиву* ***Option Explicit***.

Если данная опция включена, то нужно явно определять все переменные, если же выключена, то всем необъявленным переменным присваивается тип по умолчанию. Пример:

```
Option explicit
Dim MyVar       ' Определение переменной
MyInt = 10      ' Ошибка: используется неопределенная переменная
MyVar = 10      ' Ошибка отсутствует
```

4.6.1.5. Соглашения о записи имен

Язык VBA является регистрово-независимым языком, т. е. не имеет значение, какими буквами написаны *служебные* слова — прописными или строчными. Более того, встроенный редактор языка VBA автоматическая заменяет регистр, как в служебных словах, так и в определенных явно именах переменных и функций. При присвоении имен подпрограммам, константам, переменным и аргументам необходимо учитывать следующие правила.

- В языке VBA не различаются строчные и прописные буквы, поэтому, например, идентификатор MyIdentificator и myidentificator являются эквивалентными и указывают на одну область памяти. По умолчанию в языке VBA все одинаковые идентификаторы автоматически приводятся к единообразному виду, соответственно его первому вхождению в текст программного кода.
- Имена должны начинаться с буквы.
- Идентификатор (имя) не может содержать пробел, точку, восклицательный знак или специальные символы @, &, $, # (однако, в конце идентификаторов переменных специальные символы использовать можно).
- Имена не должны содержать более 255 символов.
- Не рекомендуется использовать имена, совпадающие с названиями стандартных синтаксических конструкций — служебными словами, функциями, объектами, методами и т. д. Совпадение названий может привести к замещению соответствующего стандартного элемента языка. При возникновении такой ситуации для вызова встроенных функций, операторов или методов необходимо явно указывать связанную с ними библиотеку типов. Например, если была объявлена переменная с именем Left, то функция Left должна вызываться как VBA.Left.
- Имена, определенные в одной области видимости, должны быть уникальны. Это означает, например, что в одной процедуре нельзя объявить две переменные с одинаковым именем. Однако имя переменной, определенной как глобальная, вполне может совпадать с именем переменной, описанной локально. В этом случае соответствующая глобальная переменная замещается локальной переменной. Локальные переменные, описанные в разных подпрограммах, независимы, поэтому их имена могут совпадать.

Кроме этого, при создании идентификатора рекомендуется использовать *формальные префиксы*, указывающие на тип переменной (табл. 4.4). Использование префиксов создает единообразие в обозначении переменных.

Таблица 4.4. *Префиксы переменных, указывающие на их тип*

Тип данных	Префикс
Boolean	Bln
Byte	Byt
Currency	Cur
Date	Dtm
Double	Dbl
Integer	Int
Long	Lng
Object	Obj
Single	Sng
String	Str
Пользовательский тип данных	Udt
Variant	Vnt

4.6.1.6. Определение переменной

Определение переменной до ее использования в тексте программы в языке VBA не является обязательным. Если при определении переменной не был явно указан тип переменной или она вообще не была описана, то ей автоматически присваивается тип **Variant**:

```
VarVal = "10"                 ' VarVal имеет тип String
                              '   и значение "10"
Dim VarValue As Variant
VarValue = "10"               ' VarValue имеет тип String
                              '   и значение "10"
VarValue = VarValue + 10      ' VarValue имеет тип Integer
                              '   и значение 20
VarValue = VarValue & " штук" ' VarValue имеет тип String
                              '   и значение "20 штук"
```

4.6.1.7. Служебное слово Null

Переменные типа **Variant** могут иметь *особое значение* ***Null***, которое означает, что переменные отсутствуют, неизвестны или неприменимы. Например, по умолчанию данные в полях таблицы базы данных имеют тип **Variant**. Поэтому, если оставить поле пустым, ему будет присвоено значение **Null**. Функция IsNull проверяет, является ли указанное значение **Null**:

```
Dim VntVar, BlnVar As Boolean
VntVar = 3
BlnVar = IsNull( VntVar )   ' VntVar примет значение False
```

4.6.1.8. Комментарии

Комментарии в языке VBA пишутся за символом ' вплоть до конца строки. Можно также использовать для этих целей служебное слово **Rem** в начале строки или после оператора. Пример использования комментариев:

```
Dim Inta As Integer
' ***************************
' * Inta – целая переменная *
' ***************************   *****
Dim Strb As String: Rem Комментарий д. б. после двоеточия
Rem – это тоже строка комментария
```

4.6.1.9. Перенос строки кода

В соответствии с синтаксисом языка VBA оператор должен располагаться в одной строке. Однако это неудобное ограничение можно обойти. Для этого достаточно использовать комбинацию символов " _" (пробел + символ подчеркивания) в конце строки и последующая строка будет восприниматься как продолжение предыдущей:

```
MsgBox "Имя состоит только" & vbCr _
       & " из букв латинского алфавита"
' MsgBox выводит окно с заданным сообщением
' vbCr осуществляет переход на новую строку
' & является операцией сложения строк
```

4.6.1.10. Строки и строковые операции

Строка представляет собой последовательность символов, которая должна быть окружена кавычками:

```
Dim StrS As String
StrS = "Это строка"
```

В языке VBA имеется две строковые операции: *присваивание* и *конкатенация*. Конкатенация обозначается символом & или символом +:

```
Dim StrS As String
StrS = "Visual Basic " & "for " + "Applications"
```

Служебное слово ***Empty*** возвращает ссылку на пустую строку. Того же эффекта можно добиться, поставив рядом две кавычки (""). Например:

```
If Empty = "" Then MsgBox "Равносильны"
```

4.6.1.11. Даты

Тип ***Date*** подразумевает как время, так и дату. Отображаются даты из интервала от 1 января 100 года до 31 декабря 9999 года, а время из интервала от 0:00:00 до 23:59:59. Значения дат могут быть представлены в любом распознаваемом формате и должны быть окаймлены знаками “#”. Например, следующие два оператора присваивают переменным DtmD1 и DtmD2 одно и то же значение — 31 января 2003 года:

```
Dim DtmD1 As Date, DtmD2 As Date
DtmD1 = #1/31/2003#
DtmD2 = #31 Jan 2003#
```

4.6.1.12. Статические массивы

Массив, как и скалярную переменную, надо определять с помощью операторов **Dim**, **Static**, **Private** и **Public**, которые также задают область видимости и время действия массива. В массиве допускается описание до 60 размерностей. При определении размерности надо указывать верхнюю и нижнюю границы. Если нижний индекс не задан явно, нижняя граница массива определяется директивой **Option Base**. Если данная директива отсутствует, нижняя граница массива равняется нулю:

```
Dim IntA( 11 ) As Integer
```

`IntA( 0 )` — первый элемент массива, `IntA( 11 )` — последний, 0 — базовый индекс, всего 12 элементов.

Пример определения матрицы из 2 строк и 3 столбцов:

```
Dim SngB( 1, 2 ) As Single
```

Пример изменения нижней границы массива — массив из 11 элементов:

```
Option Base 1
Dim IntA( 11 ) As Integer
```

При определении массива можно использовать служебное слово **To**:

```
Dim IntA( 1 To 12 ) As Integer
```

Элементы массива можно инициализировать либо последовательностью операторов

```
Dim SngB( 1, 1 ) As Single
' Два оператора в одной строке
SngB( 0, 0 ) = 2 : SngB( 0, 1 ) = 4
SngB( 1, 0 ) = 1 : SngB( 1, 1 )= 6
```

либо оператором цикла

```
Dim IntM( 1 To 9, 1 To 9 ) As Integer, Inti As Integer, _
    Intj As Integer
' Циклы For-Next, переменные цикла Inti и Intj
' Шаг по умолчанию равен 1
For Inti = 1 To 9
    For Intj = 1 To 9
        IntM( Inti, Intj ) = Inti * Intj
    Next
Next
```

Удобным способом определения одномерных массивов является функция Array, преобразующая список элементов, разделенных запятыми, в вектор из этих значений и присваивающая их переменной типа **Variant**. Допустима инициализация как одномерного массива, так и многомерного, за счет применения вложенных конструкций с функциями Array.

Пример инициализации одномерного массива с использованием функции Array:

```
Dim VntNum As Variant, DblS As Double
VntNum = Array( 10, 20 )
DblS = VntNum( 0 ) + VntNum( 1 )
MsgBox DblS
```

Пример инициализации двумерного массива с использованием вложенных конструкций с функциями Array:

```
Dim VntCityCountry As Variant
VntCityCountry = Array( _
    Array( "Санкт-Петербург", "Россия" ), _
    Array( "Кейптаун", "ЮАР" ) )
' Отобразится "Санкт-Петербург"
MsgBox VntCityCountry( 0 ) ( 0 )
MsgBox VntCityCountry( 0 ) ( 1 ) ' Отобразится "Россия"
```

4.6.1.13. Динамические массивы

Иногда в процессе выполнения программы требуется изменять размер массива. В этом случае первоначально массив определяют как динамический. Для этого при определении массива не надо указывать размерность:

```
Dim SngR( ) As Single
```

Затем в программе следует вычислить необходимый размер массива и изменить размер динамического массива с помощью *оператора* ***ReDim***.

В следующем примере сначала объявляется динамический массив, а затем устанавливаются границы его индекса:

```
Dim DblR( ) As Double
ReDim DblR( 1 To 10 )
```

Допустимо повторное использование оператора **ReDim** для изменения числа элементов и размерностей массива. В следующем примере создается массив с результатами бросания монеты. Монета бросается до тех пор, пока три раза не выпадет орел. Размерность динамического массива после каждого броска корректируется с сохранением в нем ранее записанных результатов бросания монеты за счет использования *служебного слова* ***Preserve***.

```
Dim VntAttempt( ), Inti As Integer, IntScore As Integer, _
    IntCoin As Integer
Inti = 0: IntScore = 0
' Цикл Do, выполняется, пока не выполнится условие Until
Do
    Inti = Inti + 1
    IntCoin = Int( 2 * Rnd( ) )
    ' Функция Int( ) возвращает целую часть числа
    ' Функция Rnd( ) возвращает случайное число из
    '   интервала от 0 до 1
    ' 0 - решка, 1 - орел
    If IntCoin = 1 Then IntScore = IntScore + 1
    ReDim Preserve VntAttempt( Inti )
    VntAttempt( Inti ) = Inti
Loop Until IntScore = 3
```

4.6.1.14. Определение границ индексов массива

Функции LBound, UBound возвращают минимальное и максимальное допустимые значения указанного индекса массива. Например, в следующем коде отобразится 100 и 5:

```
Dim VntA( 1 To 100, 0 To 5 )
MsgBox UBound( VntA, 1 ) & vbCr & UBound( VntA, 2 )
' vbCr – оператор новой линии
```

Следующие операторы позволяют перебирать элементы массива без указания его размерности:

```
Dim VntD As Variant, Inti As Integer
VntD = Array( "Пн", "Вт", "Ср", "Чт", "Пт" )
For Inti = LBound( VntD ) To UBound( VntD )
    MsgBox VntD( Inti )
Next
```

4.6.1.15. Константы

Константы в отличие от переменных, *не могут изменять свои значения*. Константы могут записываться либо в виде *литералов*, либо определяться как *именованные константы*. Именованные константы определяются при помощи служебного слова **Const**, перед которым, не обязательно, может находиться определяющее область видимости служебное слово **Public** или **Private** (при его отсутствии предполагается по умолчанию **Private**). Как и переменные, именованные константы делятся на три группы — локальные, контейнерные и глобальные. Если именованная константа определяется внутри процедуры или функции (**Private**), то она является локальной и доступна только внутри процедуры или функции. Контейнерные именованные константы доступны только внутри контейнера (формы, модуля или класса), в котором они определены. Глобальные именованные константы могут определяться только в модуле с использованием служебного слова **Public.** Они доступны во всей программе. Приведем примеры констант-литералов и именованных констант:

```
' Именованные константы для типа Byte
' *********************************
Public Const BytB1 As Byte = 255
Private Const BytB2 As Byte = 0     ' или эквивалентно
Const BytB2 As Byte = 0
Const BytB2 As Byte = 121
' Литералы и именованные константы для типа Currency
' ************************************************
1000000.0000@                       ' Литерал
Public Const CurC1 As Currency = 1000000.0000@
Private Const CurC2 As Currency = 1000000.0000@
' или эквивалентно
Const CurC3 As Currency = 1000000.0000@
' Литералы и именованные константы для типа Date
' **********************************************
#January 1, 1993#                   ' Литерал
Public Const DtmD1 = #January 1, 1993#
```

```
Private Const DtmD2 = #January 1, 1993#
' или эквивалентно
Const DtmD3 = #January 1, 1993#
' Литералы и именованные константы для типа Double
' ************************************************
1.5                                  ' Литерал
Public Const DblD1 As Double = 1.5
Private Const DblD2 As Double = 1.5
' или эквивалентно
Const DblD3 As Double = 1.5
' Литералы и именованные константы для типа Integer
' ************************************************
1                                    ' Литерал
Public Const IntI1 As Integer = 2
Private Const IntI2 As Integer = 3
' или эквивалентно
Const IntI3 As Integer = 3
' Литералы и именованные константы для типа Long
' *********************************************
1&                                   ' Литерал
Public Const LngL1 As Long = 2&
Private Const LngL2 As Long = 3&
' или эквивалентно
Const LngL3 As Long = 3&
' Литералы и именованные константы для типа Single
' ************************************************
17.4!                                ' Литерал
Public Const SngS1 As Single = 17.4!
Private Const SngS2 As Single = 17.4!
' или эквивалентно
Const SngS2 As Single = 17.4!
' Литералы и именованные константы для типа String
' ************************************************
"Hello"                              ' Литерал
Public Const StrS1 As String = "Hello"
Private Const StrS2 As String = "Hello"
' или эквивалентно
Const StrS2 As String = "Hello"
```

4.6.1.16. Перечисляемый тип

Перечисляемый тип предоставляет удобный способ работы с целочисленными константами и позволяет ассоциировать их значения с именами. Этот тип определяется при помощи *служебного слова* ***Enum***, перед которым может идти модификатор доступа **Public** или **Private**. В следующем далее примере задается перечисляемый тип для идентификации стороны монеты:

```
Enum Coin
    Head = 1
    Tail = -1
End Enum
' Подпрограмма Attempt( )
```

```
Sub Attempt( )
    Dim r As Integer
    ' Оператор Randomize используется для указания выборки
    '   псевдослучайных чисел
    Randomize
    r = 2 * Int( 2 * Rnd( ) ) - 1
    ' Оператор выбора
    Select Case r
        Case Head
            MsgBox "Орел!"
        Case Tail
            MsgBox "Решка"
    End Select
End Sub
```

Если перечисляемым константам целого типа (в данном случае Head и Tail) явно не заданы значения, то по умолчанию они полагаются равными 0, 1, …

4.6.1.17. Тип данных, определяемых пользователем (структурный тип)

Наряду с массивами, представляющими нумерованный набор элементов одинакового типа, существует еще один способ создания *структурного типа* — определенный пользователем тип или, в привычной терминологии для программистов — *запись (структура)*. Запись — это совокупность нескольких элементов, каждый из которых может иметь свой тип. Элемент записи называется полем. Запись является частным случаем класса, в котором не определены методы. Данный тип определяется при помощи *служебного слова* ***Type***, перед которым может идти модификатор доступа **Public** или **Private**. Оператор **Type** используется *только на уровне модуля*. Появлению в модуле класса оператора **Type** должно предшествовать служебное слово **Private**. В следующем далее примере оператор **Type** используется для определения типа данных MyType, инкапсулирующего в себе информацию о персонажах сказок. У этой записи имеются три поля. Первое поле имеет тип **Integer** и содержит идентификационный номер героя, второе поле имеет тип **String** и специфицирует его имя. Третье поле имеет тип **Variant** и может содержать любой тип данных, и даже динамический массив.

```
' Объявление структурного типа
Private Type MyType
    ID As Integer
    Name As String
    Info As Variant
End Type
' Использование структурного типа
Sub TestMyType( )
    ' Массив структур
    Dim a( 1 ) As MyType
    ' Динамический массив строк для поля Variant
    Dim txt( ) As String
' Пересоздание и инициализация массива строк
    ReDim txt( 1 )
```

```
    txt( 0 ) = " из"
    txt( 1 ) = " страны чудес"
' Инициализация первого элемента массива структур
    a( 0 ).ID = 2
    a( 0 ).Name = " Алиса"
    a( 0 ).Info = txt
    ReDim txt( 2 )
    txt( 0 ) = " Пух"
    txt( 1 ) = " - наш любимый "
    txt( 2 ) = "персонаж"
    a( 1 ).ID = 1
    a( 1 ).Name = " Винни"
    a( 1 ).Info = txt
    ' Вывод в окно Immediate
    Debug.Print "    a0    "
    Debug.Print a( 0 ).ID & a( 0 ).Name _
          & a( 0 ).Info( 0 ) & a( 0 ).Info( 1 )
    Debug.Print "    a1    "
    Debug.Print a( 1 ).ID & a( 1 ).Name _
          & a( 1 ).Info( 0 ) & a( 1 ).Info( 1 ) _
          & a( 1 ).Info( 2 )
End Sub
```

4.6.1.18. Приоритеты операций

Выражения в языке VBA состоят из операндов (константы, переменные, указатели функций и подвыражения) и операций. При вычислении значения выражения порядок выполнения операций определяется их приоритетами. Приоритет операций позволяет установить последовательность выполнения вычислений. В языке VBA существует несколько правил, строго регламентирующих порядок вычисления значения выражения.

- Операции, расположенные внутри круглых скобок, всегда выполняются раньше, чем операции вне скобок.
- При проведении вычислений в смешанных выражениях, содержащих операции разных типов, сначала выполняются арифметические операции, затем операции сравнения и логические операции.
- Арифметические операции выполняются в следующем порядке по убыванию приоритетов: возведение в степень ^, изменение знака (унарный минус -), равнозначные по приоритету умножение и деление * и /, деление по модулю **Mod**, равнозначные по приоритету сложение и вычитание (бинарные операции плюс + и минус -), сложение строк &. Операции одинакового приоритета выполняются в порядке их расположения в выражении слева направо.
- Операции сравнения имеют равный приоритет и выполняются в порядке их расположения в выражении слева направо.
- Логические операции выполняются в следующем порядке по убыванию их приоритетов: **Not**, **And**, **Or**, **Xor**, **Eqv**, **Imp**.

4.6.1.19. Математические операции

В языке VBA поддерживается стандартный набор *математических операций*. Следующий далее код демонстрирует их применение:

```
Dim x As Double, y As Double, z As Double
x = 1 : y = 3   ' Инициализация переменных в одной строке
z = x + y       ' Сложение
z = x - y       ' Вычитание
z = x * y       ' Умножение
z = x / y       ' Деление
' Целочисленное деление (результат — целое)
z = x \ y
z = x Mod y     ' Остаток от деления по модулю
z = x ^ y       ' Возведение в степень
```

4.6.1.20. Операции отношения

В языке VBA используется стандартный набор *операций отношения*:

```
Dim x As Double, y As Double, b As Boolean
x = 1 : y = 3
b = ( x < y )   ' Меньше
b = ( x <= y )  ' Меньше или равно
b = ( x > y )   ' Больше
b = ( x >= y )  ' Больше или равно
b = ( x <> y )  ' Не равно
b = ( x = y )   ' Равно
```

4.6.1.21. Логические операции

Логические операции языка VBA перечислены в табл. 4.5.

Таблица 4.5. *Логические операции языка VBA*

Операции	Описание
`exp1` **And** `exp2`	Логическое умножение (результат **True** при `exp1=exp2=`**True**)
`exp1` **Or** `exp2`	Логическое сложение (результат **True** при `exp1` и/или `exp2` **True**)
`exp1` **Xor** `exp2`	Исключающее ИЛИ (результат **True** тогда и только тогда, когда только один операнд имеет значение **True**)
Not `exp`	Логическое отрицание (результат при `exp=`**True** и наоборот)
`exp1` **Imp** `exp2`	Логическая импликация (принимает значение **False** лишь в случае, когда exp1=**True**, а exp2=**False**)
`exp1` **Equ** `exp2`	Логическая эквивалентность (результат **True**, когда операнды равны друг другу)
`exp1` **Like** `exp2`	Операция сравнения объектов типа String (результат **True**, если строки равны друг другу)

Действие операции сравнения **Like** зависит от директивы **Option Compare**, которая располагается в области описания модуля. По умолчанию для каждого модуля

считается установленной инструкция **Option Compare Binary**, при которой различаются строчные и прописные буквы, т. е. следующий оператор вернет **False**:

```
Debug.Print "AA" Like "aa"
```

Если же в области описания модуля указан оператор **Option Compare Text**, то при сравнении строчные и прописные буквы не различаются, и тот же самый оператор на этот раз вернет **True**.

В следующем далее примере в поле ввода диалогового окна пользователь вводит свое имя латинскими буквами. При нажатии кнопки **OK** программа анализирует информацию и отображает соответствующее сообщение:

- если пользователь забыл ввести имя, то его об этом информируют;
- если во введенном имени присутствуют знаки, отличные от букв латинского алфавита, пользователя об этом информируют;
- если имя состоит только из букв латинского алфавита, то с пользователем здороваются.

```
Option Compare Text
Sub DemoCompare( )
    Dim StrName As String, IntLng As Integer, _
        IntI As Integer
    StrName = InputBox( "Введите имя" )
    ' InputBox( ) выводит окно с полем ввода. Возвращает
    '   введенное выражение
    StrName = Trim( StrName )
    ' Trim( ) урезает лидирующие и последние пробелы в
    '   строке
    IntLng = Len( StrName )
    ' Len( ) возвращает длину строки
    If IntLng = 0 Then
        MsgBox "Забыли ввести имя"
        Exit Sub
    Else
        For IntI = 1 To IntLng
            If Not Mid( StrName, IntI, 1 ) _
                        Like "[A-Z]" Then
            ' Функция Mid( ) возвращает подстроку строки,
            '   содержащую, начиная со специфицированной
            '   позиции, указанное число символов
            ' "[A-Z]" - строка, содержащая все символы,
            '   находящиеся между A и Z
                MsgBox "Имя должно состоять только" _
                       & vbCr & _
                       "из букв латинского алфавита"
                Exit Sub
            End If
        Next IntI
    End If
    MsgBox "Привет, " & StrName
End Sub
```

4.6.1.22. Операции присваивания

Оператор присваивания присваивает значение выражения переменной, константе или свойству объекта (свойство — элемент ООП, позволяющий получать и устанавливать значения параметров текущего состояния объекта). Оператор присваивания всегда включает знак равенства =. Например, в результате, действия следующей пары операторов

```
x = 2
x = x + 2
```

переменной x будет присвоено значение 4.

Для присваивания переменной ссылки на объект в операторе присваивания применяется служебное слово **Set**. В общем случае оператор **Set** имеет следующий синтаксис:

```
Set varname = {[ New ] expression | Nothing}
```

Здесь служебное слово **New** используется при создании нового экземпляра класса, а служебное слово **Nothing** позволяет освободить все системные ресурсы и ресурсы памяти, выделенные для объекта, на который имелась ссылка (проще говоря, объект удаляется из памяти). Пример:

```
' Определение объекта типа Drives
Dim objFSO As FileSystemObject
Dim objDrives As Drives
Set objFSO = New FileSystemObject
Set objDrives = objFSO.Drives
```

4.6.1.23. Оператор With

*Оператор **With*** избавляет программиста от утомительной обязанности использовать большое количество повторений имени одного и того же объекта при работе с его свойствами и методами. Кроме того, он структурирует код, делая его более прозрачным:

```
With frmFirst
    .Font.Italic = True
    .PictureTiling = True
End With
```

Это эквивалентно следующей записи:

```
frmFirst.Font.Italic = True
frmFirst.PictureTiling = True
```

4.6.1.24. Операторы управления

В языке VBA имеются несколько операторов управления ходом выполнения программы. Функционально они делятся на две группы:

- операторы перехода и выбора (**Goto**, **If** и **Select**);
- операторы повтора (**For Next**, **For Each**, **Do-Loop** и **While**).

*Оператор условного перехода **If*** задает выполнение определенных групп операторов в зависимости от значения выражения. Например, если скидка (скажем, 5%)

применяется только к суммам больше 1000, то в языке VBA это можно записать следующим образом:

```
If Сумма > 1000 Then Скидка = 0.05 Else Скидка = 0
```

что эквивалентно

```
Скидка = 0
If Сумма > 1000 Then Скидка = 0.05
```

Рекомендуется использование блочной формы синтаксиса, которая часто упрощает восприятие оператора условного перехода. Приводимый ранее пример со скидкой можно записать в следующей эквивалентной блочной структуре:

```
If Сумма > 1000 Then
    Скидка = 0.05
Else
    Скидка = 0
End If
```

или

```
Скидка = 0
If Сумма > 1000 Then
    Скидка = 0.05
End If
```

Дерево условий может оказаться гораздо более сложным, чем просто проверка одного условия. В этом случае используется *оператор* ***If Then ElseIf***, который позволяет проверять множественные условия. Следующий пример демонстрирует то, как производится порядок проверки условий. В нем, в зависимости от величины вводимого числа, отображается сообщение о принадлежности числа либо интервалу [0, 1], либо интервалу (1, 2], либо о не принадлежности числа этим двум интервалам.

```
Sub DemoElseIf( )
    x = InputBox( "Введите число" )
    ' Преобразование строки в число
    x = Val( x )
    If 0 <= x And x <= 1 Then
        MsgBox "Число из интервала [0, 1]"
    ElseIf 1 < x And x <= 2 Then
        MsgBox "Число из интервала (1, 2]"
    Else
        MsgBox "Число либо отрицательное, либо больше 2"
    End If
End Sub
```

Не применяйте оператор условного перехода, если это возможно:

```
Sub TrueOrFalseSlower( )
    Dim BlnIsYes As Boolean
    Dim IntI As Integer
    IntI = 3
    If IntI = 5 Then
        BlnIsYes = True
    Else
```

```
        BlnIsYes = False
    End If
    MsgBox BlnIsYes
End Sub
```

В этом случае предпочтительнее использовать приводимый далее код, который не только изящнее, но и работает быстрее:

```
Sub TrueOrFalseFaster( )
    Dim BlnIsYes As Boolean
    Dim IntI As Integer
    IntI = 3
    BlnIsYes = ( IntI = 5 )
    MsgBox BlnIsYes
End Sub
```

Оператор выбора ***Select Case*** выполняет одну из нескольких групп операторов в зависимости от значения выражения. Оператор выбора очень эффективен, когда надо проверить одну переменную или выражение, принимающие ограниченное количество значений (типы **Byte**, **Integer**, **Long**). В следующем примере, как и ранее, в зависимости от величины введенного числа, отображается сообщение, указывающее на величину числа или диапазон, которому оно принадлежит.

```
Sub DemoSelect( )
    Dim IntX As Integer
    IntX = InputBox( "Введите целое число" )
    Select Case IntX
        Case 1
            MsgBox "Число равно 1"
        Case 2, 3
            MsgBox "Число равно 2 или 3"
        Case 4 To 6
            MsgBox "Число от 4 до 6"
        Case Is >= 7
            MsgBox "Число не менее 7"
        Case Else
            MsgBox " Число менее 1"
    End Select
End Sub
```

Оператор ***For Next*** повторяет выполнение группы операторов указанное число раз, а именно, пока переменная цикла изменяется от начального значения до конечного с указанным шагом. Если шаг не указан, то он полагается равным 1. Альтернативный способ выхода из цикла предоставляет оператор **Exit For**. В следующем примере при помощи оператора цикла находится сумма элементов массива.

```
Sub DemoFor1( )
    Dim VntA As Variant
    VntA = Array( 1, 4, 12, 23, 34, 3, 23 )
    VntS = 0
    For Vnti = LBound( VntA ) To UBound( VntA )
        VntS = VntS + VntA( Vnti )
    Next
```

```
    MsgBox VntS
End Sub
```

В следующем коде находится сумма всех четных целых из интервала от 1 до 100.

```
Sub DemoFor2( )
    Dim IntI As Integer, IntS As Integer
    IntS = 0
    For IntI = 2 To 100 Step 2
        IntS = IntS + IntI
    Next
    MsgBox IntS
End Sub
```

Оператор ***For Each*** повторяет выполнение группы операторов для каждого элемента массива или семейства. Альтернативный способ выхода из цикла предоставляет оператор **Exit For**. В следующем далее коде оператор **For Each** используется для суммирования элементов массива:

```
Sub DemoForEach( )
    Dim VntA As Variant, DblS As Double
    VntA = Array( 1, 4, 12, 23, 34, 3, 23 )
    DblS = 0
    For Each b In VntA
        DblS = DblS + b
    Next
    MsgBox DblS
End Sub
```

Оператор ***While*** выполняет последовательность операторов, пока заданное условие возвращает значение **True**. Оператор повтора **While** в отличие от оператора **For** выполняется не заданное число раз, а пока выполняется условие. В следующем примере бросается игральная кость до тех пор, пока не выпадет шесть очков. При выпадении шести очков игра заканчивается, и отображается сообщение с указанием, на каком броске она закончилась.

```
Sub DemoWhile( )
    Dim IntAttempt As Integer, IntScore As Integer
    Randomize
    IntScore = Int( 6 * Rnd( ) ) + 1
    IntAttempt = 1
    While IntScore < 6
        IntAttempt = IntAttempt + 1
        IntScore = Int( 6 * Rnd( ) ) + 1
    Wend   ' Служебное слово окончания цикла While
    MsgBox "Победили на попытке: " & IntAttempt
End Sub
```

Оператор ***Do*** повторяет выполнение набора операторов, пока условие имеет значение **True** (случай **While**) или пока оно не примет значение **True** (случай **Until**):

```
Do [{While | Until} condition]
   [statements]
   [Exit Do]
   [statements]
```

```
Loop
```
или
```
Do
   [statements]
   [Exit Do]
   [statements]
Loop [{While | Until} condition]
```

В любом месте управляющей структуры **Do** может быть размешено любое число операторов **Exit Do**, обеспечивающих альтернативные возможности выхода из цикла **Do**. Примером использования оператора цикла **Do Until** может быть следующий далее код, который обеспечивает повторение цикла до тех пор, пока в поле ввода диалогового окна не будет введен пароль (в данном случае Admin):

```
Sub DemoPassword( )
    Dim StrPs As String
    Do
        StrPs = InputBox( "Введите пароль" )
    Loop Until StrPs = "Admin"
End Sub
```
или эквивалентно
```
Sub DemoPassword( )
    Dim StrPs As String
    Do While StrPs <> "Admin"
        StrPs = InputBox( "Введите пароль" )
    Loop
End Sub
```

Оператор безусловного перехода ***Goto*** задает переход на указанную строку внутри подпрограммы или функции. Для использования оператора безусловного перехода надо какой-то строке присвоить метку. Метка должна начинаться с буквы и заканчиваться двоеточием. В качестве примера использования оператора безусловного перехода рассмотрим далее игру, в которой игроку даны десять попыток броска игральной кости. В случае если при какой-то из этих попыток выпадает шесть очков, игрок выигрывает, и игра заканчивается.

```
Sub DemoGoTo( )
    Dim IntI As Integer, IntScore As Integer
    Randomize
    For IntI = 1 To 10
        IntScore = Int( 6 * Rnd( ) ) + 1
        If IntScore = 6 Then GoTo lblMessage
    Next
    GoTo lblEnd
    lblMessage: MsgBox "Выиграли при броске " & IntI
    lblEnd:
End Sub
```

4.6.1.25. Основные программы и функции языка

Язык VBA содержит большое количество *стандартных подпрограмм и функций*, которые существенно расширяют возможности языка. Далее рассматриваются основные подпрограммы и функции, сгруппированные по их назначению.

4.6.1.25.1. Математические функции

В языке VBA имеется представительная группа *математических функций* (табл. 4.6).

Таблица 4.6. *Математические функции языка VBA*

Функция	Описание
`Abs`	Модуль (абсолютная величина) аргумента функции
`Atn`	Арктангенс аргумента функции, выраженного в радианах
`Cos`	Косинус аргумента функции, выраженного в радианах
`Exp`	Экспонента, т. е. результат возведения основания натурального логарифма в степень, указанную аргументом функции
`Log`	Натуральный логарифм аргумента функции
`Rnd`	Функция Rnd(number) возвращает очередное псевдослучайное число из интервала [0, 1]. Если значение параметра number меньше нуля, то возвращается число, зависящее от number. Если значение параметра number больше нуля или этот параметр опущен, то возвращает следующее случайное число в последовательности
`Sgn`	Знак аргумента функции (1 — аргумент положительный, 0 — аргумент нулевой, -1 — аргумент отрицательный)
`Sin`	Синус аргумента функции, выраженного в радианах
`Sqr`	Квадратный корень из аргумента функции
`Tan`	Тангенс аргумента функции, выраженного в радианах
`Fix, Int`	Функции отбрасывают дробную часть числа и возвращают целое значение. Различие между ними состоит в том, что для отрицательного значения аргумента функция Int возвращает отрицательное ближайшее целое число, меньшее либо равное аргументу, а Fix — ближайшее отрицательное целое число, большее либо равное аргументу

Функция `Rnd` генерирует очередное число из последовательности псевдослучайных чисел. У таких последовательностей, если совпадают первые члены, то совпадают и все последующие. Перед вызовом функции `Rnd` следует применять *оператор* ***Randomize***, который определяет первый член этой последовательности. Если этот оператор используется без параметра, то первый член последовательности привязан к текущему системному времени, что делает число, возвращаемое функцией `Rnd`, "более" случайным. В языке VBA нет функции, возвращающей число π, но для его нахождения можно использовать функцию `Atn`:

```
Dim Pi As Double
Pi = 4.0 * Atn( 1 )
```

Конечно, можно задать π и явным образом, указав достаточное число значащих цифр, но этот подход менее удобен.

4.6.1.25.2. Функции проверки типов

Функция проверки типа проверяет, имеет ли ее аргумент заданный тип (табл. 4.7).

Таблица 4.7. *Функции проверки типов языка VBA*

Функция	Проверяемый тип
`IsArray`	Возвращает **True**, если аргумент функции является массивом
`IsDate`	Возвращает **True**, если аргумент функции является датой
`IsEmpty`	Возвращает **True**, если аргумент функции был явно объявлен
`IsError`	Возвращает **True**, если аргумент функции является кодом ошибки
`IsNull`	Возвращает **True**, если аргумент функции имеет значение **Null**
`IsNumeric`	Возвращает **True**, если аргумент функции является значением переменной
`IsObject`	Возвращает **True**, если аргумент функции является объектом

4.6.1.25.3. Функции преобразования типов

Преобразование строки в число. Функция `Val` возвращает подходящее число, заданное в аргументе функции в виде строки. При наличии другого десятичного разделителя целой и дробной частей (например, запятой) будет преобразована только часть строки-аргумента, предшествующая недопустимому разделителю. Приведем далее примеры преобразования строк в числа:

```
Sub DemoStrToNum( )
    Dim StrS As String
    StrS = "23432"
    Debug.Print Val( StrS )          ' Отобразится 23432
    StrS = "123-45-45"
    Debug.Print Val( StrS )          ' Отобразится 123
    StrS = #12/15/1999#
    Debug.Print Val( StrS )          ' Отобразится 15,12
    StrS = "198,005"
    Debug.Print Val( StrS )          ' Отобразится 198
    StrS = "198.005"
    Debug.Print Val( StrS )          ' Отобразится 198,005
End Sub
```

Преобразование числа в строку. Функция `CStr` возвращает значение типа **String**, являющееся строковым представлением числового аргумента функции. Например, следующий далее код проверяет, является ли 0 одной из цифр сгенерированного целого случайного числа из интервала от 1 до 1000. Для этого число преобразуется в строку, а затем с помощью функции `InStr` проверяется, является ли указанная строка подстрокой данной.

```
Sub DemoNumToStr( )
    Dim IntI As Integer
    IntI = 1000 * Rnd( ) + 1
```

```
    If InStr( CStr( IntI ), "0" ) > 0 Then
        MsgBox "0 встретился"
    Else
        MsgBox "0 не встретился"
    End If
End Sub
```

Кроме функций `Val` и `CStr`, в языке VBA имеется целый ряд *функций преобразования типов* (табл. 4.8).

Таблица 4.8. *Функции преобразования типов языка VBA*

Функция	Тип, в который преобразуется аргумент функции (возвращаемое значение)
`CBool`	**`Boolean`**
`CByte`	**`Byte`**
`CCur`	**`Currency`**
`CDate`	**`Date`**
`CDbl`	**`Double`**
`CDec`	**`Decimal`**
`CInt`	**`Integer`**
`CLng`	**`Long`**
`CSng`	**`Single`**
`CVar`	**`Variant`**
`CStr`	**`String`**

4.6.1.25.4. Форматирование числового значения функцией Format

Чтобы представить числовое значение, заданное выражением, как дату, время, денежное значение или в специальном формате, следует использовать *функцию* `Format`, которая возвращает значение типа **`Variant`** (**`String`**), содержащее выражение, отформатированное в соответствии с описанием формата:

```
Format( Expression[, NameUserFmt[, FirstDayOfWeek
        [, FirstWeekOfYear]]] ),
```

где `Expression` — любое допустимое выражение; `NameUserFmt` — любое допустимое именованное или определяемое пользователем выражение формата; `FirstDayOfWeek` — константа, определяющая первый день недели и `FirstWeekOfYear` — константа, определяющая первую неделю года. Примером именованного формата является `Fixed` — формат для вывода вещественного числа с двумя значащими цифрами после десятичной точки. *Именованные числовые форматы* и *форматы даты и времени* приведены далее в табл. 4.9 и 4.10).

Таблица 4.9. *Именованные числовые форматы*

Формат	Описание
General Number	Число без разделителя тысяч
Currency	Использует установки страны из панели управления. Отображает две цифры справа от десятичной точки
Fixed	Отображает, по крайней мере, одну цифру слева и две справа от десятичной точки
Standard	Отображает, по крайней мере, одну цифру слева и две справа от десятичной точки и выводит разделитель тысяч
Percent	Отображает число в виде процентов и выводит две цифры справа от десятичной точки (числовое значение умножается на 100)
Scientific	Использует формат с плавающей десятичной точкой (мантисса-порядок). В мантиссе выводится одна значащая цифра слева от десятичной точки и две цифры справа. Порядок начинается с буквы "E", за которой следует знак порядка и порядок.
Yes/No	Отображает No (Нет), если число равно 0 и Yes (Да) в противном случае
True/False	Отображает False (Ложь), если число равно 0 и True (Истина) в противном случае
On/Off	Отображает Off (Выкл), если число равно 0 и On (Вкл) в противном случае

Таблица 4.10. *Именованные форматы даты и времени*

Формат	Описание
General Date	Выводит дату или время. Если нет дробной части, то выводит только дату
Long Date	Выводит дату в соответствии с полным форматом ОС Windows для даты
Medium Date	Выводит дату в соответствии с обычным форматом ОС Windows для даты
Short Date	Выводит дату в соответствии с сокращенным форматом ОС Windows для даты
Long Time	Выводит часы, минуты и секунды
Medium Time	Выводит часы и минуты в 12-часовом формате
Short Time	Выводит часы и минуты в 24-часовом формате

Далее приводится пример использования именованных форматов в функции Format:

```
Sub DemoUsgFormatFunction( )
    Dim x As Double
    x = 4654646.544564
    MsgBox Format( x, "General Number" ), , _
                   "General Number"
    ' Аргументы , , "General Number" задают заголовок окна
    '   сообщения
```

```
    ' Будет выведено: 4654646,544564
    MsgBox Format( x, "Currency" ), , "Currency"
    ' Будет выведено: 4 654 646,54р.
    MsgBox Format( x, "Fixed" ), , "Fixed"
    ' Будет выведено: 4654646,54
    MsgBox Format( x, "Standard" ), , "Standard"
    ' Будет выведено: 4 654 646,54
    MsgBox Format( x, "Percent" ), , "Percent"
    ' Будет выведено: 465464654,46%
    MsgBox Format( x, "Scientific" ), , "Scientific"
    ' Будет выведено: 4,65E+06
    MsgBox Format( x, "Yes/No" ), , "Yes/No"
    ' Будет выведено: Да
    MsgBox Format( x, "True/False" ), , "True/False"
    ' Будет выведено: Истина
    MsgBox Format( x, "On/Off" ), , "On/Off"
    ' Будет выведено: Вкл
    MsgBox Format( Now, "General Date" ), , "General Date"
    ' Служебное слово Now используется для получения
    '   текущей даты и времени
    ' Будет выведено: 05.01.2007 11:10:51
    MsgBox Format( Now, "Long Date" ), , "Long Date"
    ' Будет выведено: 5 Январь 2007 г.
    MsgBox Format( Now, "Medium Date" ), , "Medium Date"
    ' Будет выведено: 05-янв-07
    MsgBox Format( Now, "Short Date" ), , "Short Date"
    ' Будет выведено: 05.01.2007
    MsgBox Format( Now, "Long Time" ), , "Long Time"
    ' Будет выведено: 11:12:26
    MsgBox Format( Now, "Medium Time" ), , "Medium Time"
    ' Будет выведено: 11:12
    MsgBox Format( Now, "Short Time" ), , "Short Time"
    ' Будет выведено: 11:13
End Sub
```

Наряду с именованными числовыми форматами и форматами даты и времени, в функции `Format` можно применять *пользовательские форматы*, позволяющие настроить вид отображаемых значений по желанию пользователя. При построении пользовательского формата применяются специальные символы, приведенные далее в табл. 4.11.

Таблица 4.11. *Символы, используемые при построении пользовательского формата*

Символ	Описание
`0`	Резервирует позицию цифрового разряда. Отображает цифру или нуль. Если у числа, представленного параметром, есть какая-нибудь цифра в той позиции разряда, в которой в строке формата находится нуль, то функция отображает эту цифру параметра, а если нет — в этой позиции отображается нуль

Символ	Описание
#	Резервирует позицию цифрового разряда. Отображает цифру или ничего. Если у числа, представленного параметром, есть какая-нибудь цифра в той позиции разряда, в которой в строке формата находится символ #, то функция отображает эту цифру параметра, а если нет — в этой позиции не отображается ничего. Действие этого символа аналогично действию символа нуль, за исключением того, что лидирующие и завершающие нули не отображаются
.	Резервирует позицию десятичного разделителя. Указание точки в строке формата определяет, сколько разрядов необходимо отображать слева и справа от десятичной точки
%	Задает процентное отображение числа
,	Задает разделитель разряда сотен от тысяч
:	Задает разделитель часов, минут и секунд в категории форматов времени
/	Задает разделитель дня, месяца и года в категории форматов даты
E+, E-, e+, e-	Задает разделитель мантиссы и порядка в экспоненциальном формате
d, m, y	Резервирует позицию при выводе дня, месяца, года в категории форматов даты
h, m, s	Резервирует позицию при выводе часа, минуты, секунды в категории форматов времени

Далее приведен пример использования пользовательских форматов:

```
Sub DemoUsgFormatFunction( )
    MsgBox Format( 1.2^2, "##.###" )
    ' Будет выведено: 1,44
    MsgBox Format( 1.2^2, "##.000" )
    ' Будет выведено: 1,440
    MsgBox Format( Sin( 1 )*Exp( 5 ), "#.##e+##" )
    ' Будет выведено: 1,25e+2
    MsgBox Format( Now, "hh:mm:ss" )
    ' Будет выведено: 13:17:12
    MsgBox Format( Now, "dd/mm/yyyy" )
    ' Будет выведено: 05.01.2007
End Sub
```

4.6.1.25.5. Форматирование числовых, процентных, денежных значений, значений даты и времени специализированными функциями

Наряду с универсальной функцией `Format`, для форматирования выводимых значений можно использовать специализированные функции `FormatNumber` (форматирование числовых значений), `FormatPercent` (форматирование процентных значений), `FormatCurrency` (форматирование денежных значений), и `FormatDateTime` (форматирование даты и времени).

Для *форматирования числовых значений* можно использовать специализированную функцию `FormatNumber`, имеющую следующий формат:

```
FormatNumber( Expression[, NumDigitsAfterDecimal
```

```
[, IncludeLeadingDigit[,
UseParensForNegativeNumbers
[, GroupDigits]]]] ),
```

где `Expression` — обязательный параметр, указывающий форматируемое числовое выражение; `NumDigitsAfterDecimal` — необязательный параметр, задающий число знаков, отображаемых после десятичной точки (по умолчанию в дробной части выводятся две цифры); `IncludeLeadingDigit` — необязательный параметр, указывающий, надо ли отображать целую нулевую часть (допустимыми значениями являются константы: `vbTrue` — отображать, `vbFalse` — не отображать и `vbUseDefault` — отображать); `UseParensForNegativeNumbers` —необязательный параметр, определяющий, надо ли отрицательные числа отображать в скобках (допустимыми значениями являются константы `vbTrue` — отображать, `vbFalse` — не отображать и `vbUseDefault` — не отображать); `GroupDigits` — необязательный параметр, определяющий, надо ли группировать цифры (допустимыми значениями являются константы `vbTrue` — группировать, `vbFalse` — не группировать и `vbUseDefault` — группировать).

Далее приведен пример использования функции `FormatNumber`:

```
Sub DemoUsgFormatNumberFunction( )
    MsgBox FormatNumber( Sin( 4 ) )
    ' Будет выведено: -0,76
    MsgBox FormatNumber( Sin( 4 ), 3 )
    ' Будет выведено: -0,757
    MsgBox FormatNumber( Sin( 4 ), 3, vbTrue )
    ' Будет выведено: -0,757
    MsgBox FormatNumber( Sin( 4 ), 3, vbFalse )
    ' Будет выведено: -,757
    MsgBox FormatNumber( Sin( 4 ), 3, vbFalse, vbTrue )
    ' Будет выведено: (,757)
    MsgBox FormatNumber( Sin( 4 ), 3, vbFalse, vbFalse )
    ' Будет выведено: -,757
    MsgBox FormatNumber( 14547, 3, vbFalse, vbFalse, _
                         vbTrue )
    ' Будет выведено: 14 547,000
End Sub
```

Для *форматирования процентных значений* можно использовать специализированную функцию `FormatPercent`, имеющую такой же синтаксис, как и функция `FormatNumber`. Например, следующая далее функция

```
Sub DemoUsgFormatPercentFunction( )
    x = 0.2342
    MsgBox FormatPercent( x, 2 )
End Sub
```

выведет значение `23.42%`, т. е. число, записанное в формате процентов, у которого после десятичной точки отображаются две цифры.

Для *форматирования денежных значений* можно использовать специализированную функцию `FormatCurrency`, имеющую такой же синтаксис, как и функция `FormatNumber`. Например, следующая далее функция

```
Sub DemoUsgFormatPercentFunction( )
    MsgBox FormatCurrency( 12312.3453, 2 )
End Sub
```

выведет значение 12 312,35р.

Для *форматирования даты и времени* можно использовать специализированную функцию FormatDateTime, имеющую следующий синтаксис:

```
FormatDateTime( Date[, NamedFormat] ),
```

где Date — обязательный параметр, задающий форматируемую дату; NamedFormat — необязательный параметр, указывающий тип форматирования (допустимыми значениями являются константы vbGeneralDate, vbLongDate, vbShortDate, vbLongTime, vbShortTime).

Далее приведен пример использования функции FormatDateTime:

```
Sub DemoUsgFormatDateTimeFunction( )
    MsgBox FormatDateTime( Now, vbGeneralDate )
    ' Будет выведено: 05.01.2007 15:33:35
    MsgBox FormatDateTime( Now, vbLongDate )
    ' Будет выведено: 5 Январь 2007 г.
    MsgBox FormatDateTime( Now, vbShortDate )
    ' Будет выведено: 05.01.2007
    MsgBox FormatDateTime( Now, vbLongTime )
    ' Будет выведено: 15:34:21
    MsgBox FormatDateTime( Now, vbShortTime )
    ' Будет выведено: 15:34
End Sub
```

4.6.1.26. Процедуры и функции

Процедура является самостоятельной частью кода, которая имеет имя и может иметь параметры, выполнять последовательность операторов и изменять значения своих параметров. Процедура имеет следующий синтаксис:

```
[Private|Public|Friend] [Static] Sub name[( arglist )]
    [statements]
    [Exit Sub]
    [statements]
End Sub
```

Здесь **Public** — служебное слово, указывающее на то, что процедура является *открытой* и доступна для всех других процедур во всех модулях, кроме модулей, содержащих оператор **Options Private**; **Private** — служебное слово, указывающее на то, что процедура является *закрытой*, и областью ее видимости является текущий модуль; **Friend** — служебное слово, используемое только в модуле класса и указывающее на то, что процедура является *дружественной*, и областью ее видимости является проект, содержащий модуль класса; **Static** — служебное слово, указывающее на то, что локальные переменные процедуры сохраняются в промежутках времени между вызовами этой процедуры; name — имя процедуры, удовлетворяющее стандартным правилам именования переменных; arglist — список параметров, значения которых передаются в процедуру или возвращаются из процедуры при ее вызове, разделителем в списке параметров является запятая;

`statements` — любая группа операторов, выполняемых в процедуре; **Exit Sub** — оператор, приводящий к немедленному выходу из процедуры и **End Sub** — оператор, отмечающий конец процедуры.

Процедуру можно вызвать из другой процедуры или функции двумя способами.

- С помощью указателя:

```
name arglist,
```

где `name` — имя вызываемой процедуры, а `arglist` — список аргументов, передаваемых процедуре (он должен соответствовать по количеству и типу списку параметров, заданному в процедуре при ее определении, аргументы в списке разделяются запятыми).

- С помощью оператора **Call**:

```
Call name( arglist )
```

Обратите внимание, что в этом случае список фактических параметров заключается в скобки. При первом способе скобки не использовались.

Используя именованные параметры, можно вводить аргументы в любом порядке и опускать необязательные. При этом после имени аргумента ставятся двоеточие и знак равенства, после которого помещается значение аргумента. Приводимый далее код показывает основные способы передачи параметров на примере вызова процедуры `CircleLenght`:

```
Private Sub CircleLength( DblR As Double )
    Dim DblPi As Double
    DblPi = Atn( 1# ) * 4#
    MsgBox 2# * DblR * DblPi
End Sub
Private Sub ShowResults( )
    CircleLength DblR:=1# ' Первый способ вызова процедуры
    Dim DblRs As Double
    DblRs = 2#
    CircleLength DblRs    ' Второй способ вызова процедуры
    ' Третий способ вызова процедуры
    Call CircleLength( 3# )
End Sub
```

Кроме процедуры, в языке VBA имеется *функция* **Function**. Синтаксис функции содержит те же элементы, что и процедура. Оператор **Exit Function** приводит к немедленному выходу из функции. Подобно процедуре, функция является самостоятельной процедурой, которая может получать значения аргументов, выполнять последовательность операторов и изменять значения своих аргументов. Однако в отличие от процедуры, когда требуется использовать возвращаемое функцией значение, функция **Function** может применяться в правой части выражения, как и любая другая стандартная функция, например, `Cos`. Функция **Function** вызывается в выражении по своему имени, за которым следует список параметров, заключенный в скобки. Для возврата значения из функции в ее теле следует присвоить значение имени функции. Любое число таких операторов присваивания может находиться в любом месте функции.

В следующем далее коде приводится пример функции DblSum, которая находит сумму двух своих аргументов:

```
Sub DemoFun( )
    MsgBox DblSum( 1#, 3# )
End Sub
Function DblSum( DblX As Double, DblY As Double ) _
                 As Double
    DblSum = DblX + DblY
End Function
```

Передача параметров процедуры или функции по ссылке или значению. При вызове процедуры или функции в нее передаются некоторые аргументы. Если внутри процедуры или функции этим аргументам нужно присвоить какие-то значения, которые нужно сохранять после выхода из процедуры или функции, то в качестве параметров в процедуру или функцию следует передавать физические адреса переменных (использовать передачу параметров по ссылке). Благодаря этому внутри процедуры или функции может быть модифицировано их содержание. Для явного указания передачи параметров в процедуру или функцию по ссылке используется служебное слово **ByRef**. Другим способом передачи параметров в процедуру или функцию является передача их по значению. При этом способе передачи параметра в процедуру попадает не сама переменная, а ее значение (копия аргумента из вызова). Передача параметра по значению задается служебным словом **ByVal** и применяется для передачи исходных данных. В следующем далее примере показано отличие передачи параметра по ссылке от передачи параметра по значению:

```
Sub DemoByValByRef( ByVal IntA As Integer, IntB As _
                    Integer, ByRef IntC As Integer )
    ' IntA передается по значению
    ' по умолчанию IntB передается по ссылке
    ' IntC передается по ссылке
    IntA = IntA + 1
    IntB = IntB + IntA
    IntC = IntC + IntA
End Sub
Sub Test( )
    Dim IntA As Integer
    Dim IntB As Integer
    Dim IntC As Integer
    IntA = 1: IntB = 10: IntC = 100
    DemoByValByRef IntA, IntB, IntC
    MsgBox IntA
    ' Отобразится 1
    MsgBox IntB
    ' Отобразится 12
    MsgBox IntC
    ' Отобразится 102
End Sub
```

4.6.1.27. Рекурсивные процедуры и функции

В языке VBA возможно создание рекурсивных процедур и функций, т. е. процедур и функций, вызывающих сами себя. Классическим примером рекурсивной функции является функция, возвращающая очередной член последовательности чисел Фибоначчи. Два первых члена ряда Фибоначчи равны 1, а каждый его последующий член представляет собой сумму двух предыдущих. Таким образом, n-е число Фибоначчи $F_i(n)$ определяется следующим соотношением:

$F_i(n) = F_i(n-1) + F_i(n-2)$, где $F_i(1) = F_i(2) = 1$.

В следующем далее примере показано использование рекурсивной функции для вычисления очередного числа Фибоначчи:

```
Sub Test( )
    MsgBox LngFi( 4& )
End Sub
Function LngFi( LngN As Long ) As Long
    If LngN=1& Or LngN=2& Then
        LngFi = 1&
    Else
        LngFi = LngFi( LngN-1& ) + LngFi( LngN-2& )
    End If
End Function
```

4.6.1.28. Встроенные (стандартные) диалоговые окна

В проектах на языке VBA часто встречаются две разновидности стандартных окон диалога — окна сообщений и окна ввода. Они встроены в язык и, если их возможностей достаточно, то можно обойтись без проектирования пользовательских окон диалога. Окно сообщений (`MsgBox`) выводит простейшие сообщения для пользователя, а окно ввода (`InputBox`) обеспечивает ввод информации.

Функция InputBox выводит на экран диалоговое окно, содержащее сообщение, поле ввода и кнопки **OK** и **Cancel.** Она устанавливает режим ожидания ввода текста пользователем, а затем, при нажатии на кнопку **OK,** возвращает значение типа `String`, содержащее текст, введенный в поле ввода. При нажатии кнопки **Cancel** возвращается пустая строка (**Empty**). Функция имеет следующий формат:

```
InputBox( Prompt[, Title][, Default][, Xpos][, Ypos]
          [, Helpfile, Context] )
```

где используются следующие параметры.

- `Prompt` — обязательное строковое выражение, отображаемое как сообщение в диалоговом окне. Строковое выражение Prompt может содержать несколько строк. Для разделения строк допускается использование символа возврата каретки (`Chr(13)`), символа перевода строки (`Chr(10)`) или комбинации этих символов (`Chr(13)&Chr(10)`).
- `Title` — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения.

- `Default` — строковое выражение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другую строку. Если этот параметр опущен, то поле ввода изображается пустым.
- `Xpos` — числовое выражение, задающее расстояние в пикселях по горизонтали между левой границей диалогового окна и левым краем экрана. Если этот параметр опушен, то диалоговое окно выравнивается по горизонтали по центру экрана.
- `Ypos` — числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экрана. Если этот параметр опущен, то диалоговое окно помещается по вертикали примерно на одну треть высоты экрана.
- `Helpfile` — необязательное строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, то необходимо указать также параметр `Context`.
- `Context` — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр `Helpfile`.

В качестве примера приведем далее код, выполнение которого приведет к появлению на экране диалогового окна ввода, представленного на рис. 4.67:

```
Sub DemoInputBox( )
    Dim StrName As String
    StrName = InputBox( "Введите ваше имя", _
                        "Пример окна ввода" )
    MsgBox "Здравствуйте " & StrName & "!"
End Sub
```

При вводе данных при помощи функции `InputBox` разумно в программе предусмотреть как *проверку вводимых данных*, так и сам факт ввода таковых или отказа от ввода путем нажатия кнопки **Cancel.** Следующий далее пример демонстрирует, как это можно сделать при вводе дат. В примере предлагается ввести будущую дату по отношению к текущей. При помощи функции `IsDate` сначала проверяется, является ли введенное данное датой, и если таковым является, оно преобразуется функцией `CDate` к типу **Date** и сравнивается с текущей.

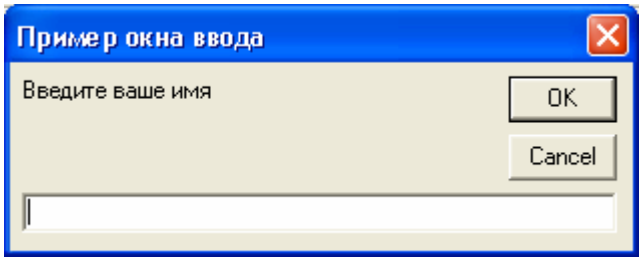

Рис. 4.67. Вид диалогового окна ввода

```
Sub DemoInputWithChecking( )
```

```
    Dim VntDate As Variant
    VntDate = InputBox( "Введите будущую дату:", _
                        "Пример", Date+1 )
    ' Третий параметр InputBox( ) используется для того,
    '   чтобы в окне ввода был текст по умолчанию,
    '   которым в данном случае будет являться следующий
    '   календарный день
    If Not IsDate( VntDate ) Then
        Select Case VntDate
            Case ""
                MsgBox _
                "Нажали кнопку Cancel или ничего не ввели"
            Case Else
                MsgBox "Некорректная дата"
        End Select
    Else
        Select Case CDate( VntDate )
            ' Is является значением CDate(VntDate)
            Case Is <= Date
                MsgBox "Должна быть будущая дата"
            Case Is > Date
                MsgBox VntDate & " - введенная дата"
        End Select
    End If
End Sub
```

Конечно, если вы уверены в корректности формата вводимых данных, то проверку на тип данных осуществлять не надо, но, тем не менее, позаботиться о проверке того, была ли нажата кнопка **Cancel** или ничего не введено, стоит, например, так, как это делается в следующем далее коде:

```
Sub DemoInputWithChecking1( )
    Dim VntX As Variant
    VntX = InputBox( "Введите значение:", "Пример" )
    If VntX <> "" Then
        MsgBox VntX & " – введенное значение"
    Else
        MsgBox "Нажали Cancel или ничего не ввели"
    End If
End Sub
```

Функция MsgBox выводит на экран окно диалога, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа **Integer**, указывающее, какая кнопка была нажата. Функция имеет указанный далее синтаксис

```
MsgBox( Prompt[, Buttons][, Title][, Helpfile, Context] )
```

и содержит следующие параметры.

- `Prompt` — обязательное строковое выражение, отображаемое как сообщение в окне диалога.
- `Buttons` — числовое выражение, представляющее сумму значений (именованных констант), которые указывают число и тип отображаемых кнопок,

тип используемого значка, основную кнопку и модальность окна сообщения. Значение этого параметра по умолчанию равно 0 (окно диалога содержит только одну кнопку **ОК**). Значения констант, определяющих число, тип кнопок и тип используемого значка приведены в табл. 4.12, 4.13; значения, определяющие основную кнопку окна диалога, приведены в табл. 4.14.

- `Title` — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения.
- `Helpfile` — необязательное строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, то необходимо указать также параметр `Context`.
- `Context` — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр `Helpfile`.

Таблица 4.12. *Именованные константы, определяющие кнопки окна диалога (могут комбинироваться в параметре* Buttons*)*

Именованная константа	Значение	Отображаемые кнопки
`vbOKOnly, ebOKOnly`	0	ОК
`vbOKCancel, ebOKCancel`	1	ОК Отмена
`vbAbortRetryIgnore, ebAbortRetryIgnore`	2	Прервать Повтор Пропустить
`vbYesNoCancel, ebYesNoCancel`	3	Да Нет Отмена
`vbYesNo, ebYesNo`	4	Да Нет
`vbRetryCancel, ebRetryCancel`	5	Повтор Отмена

Таблица 4.13. *Именованные константы, определяющие информационные значки окна диалога (могут комбинироваться в параметре* Buttons*)*

Именованная константа	Значение	Отображаемые значки
`vbCritical, ebCritical`	16	
`vbQuestion, ebQuestion`	32	
`vbExclamation, ebExclamation`	48	

vbInformation, ebInformation	64	

Таблица 4.14. *Именованные константы, определяющие основную кнопку окна диалога (могут комбинироваться в параметре* Buttons*)*

Именованная константа	Значение	Номер основной кнопки
vbDefaultButton1, ebDefaultButton1	0	1
vbDefaultButton2, ebDefaultButton2	256	2
vbDefaultButton3, ebDefaultButton3	512	3
vbDefaultButton4, ebDefaultButton4	768	4

При написании программ с откликом в зависимости от того, какая кнопка окна диалога нажата, вместо возвращаемых значений удобнее использовать именованные константы, перечисленные в табл. 4.15. Их использование делает код программы более прозрачным для чтения и, к тому же, именованные константы легче запомнить.

Таблица 4.15. *Именованные константы, идентифицирующие нажатую кнопку*

Именованная константа	Значение	Нажатая кнопка
vbOK, ebOK	1	**OK**
vbCancel, ebCancel	2	**Отмена** (**Cancel**)
vbAbort, ebAbort	3	**Прервать** (**Abort**)
vbRetry, ebRetry	4	**Повторить** (**Retry**)
vbIgnore, ebIgnore	5	**Пропустить** (**Ignore**)
vbYes, ebYes	6	**Да** (**Yes**)
vbNo, ebNo	7	**Нет** (**No**)

Проиллюстрируем использование окон сообщений. В приводимом далее примере вычисляется квадрат введенного значения 2, и найденное значение затем последовательно отображается в пяти окнах сообщения. Первое окно является простым окном с сообщением (рис. 4.68), второе — с сообщением, выводимым в две строки (рис. 4.69), третье — с сообщением и информационным значком (рис. 4.70), четвертое — с сообщением, информационным значком и двумя кнопками **Да** и **Нет**, причем кнопка **Да** установлена как кнопка, выполняемая по умолчанию (рис. 4.71), а пятое — с сообщением, информационным значком и пользовательским заголовком (рис. 4.72):

```
Sub DemoMsgBox( )
    Dim VntX As Variant
    Dim DblY As String
    VntX = InputBox( "Введите x" )
    DblY = VntX^2
    MsgBox "VntX="+VntX
    MsgBox "VntX="+VntX+Chr(13)+Chr(10)+"DblY=VntX^2="+DblY
    MsgBox "DblY="&DblY, ebInformation
```

```
    MsgBox "DblY="&DblY, _
           ebInformation+ebDefaultButton1+ebYesNo
    MsgBox "DblY="&DblY, ebInformation, _
           "Нахождение квадрата"
End Sub
```

Рис. 4.68. Простое окно сообщения

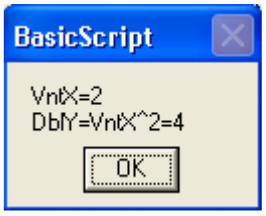

Рис. 4.69. Окно с сообщением, выводимым в две строки

Рис. 4.70. Окно с сообщением и информационным значком

Рис. 4.71. Окно с сообщением, информационным значком и кнопками **Да** (по умолчанию) и **Нет**

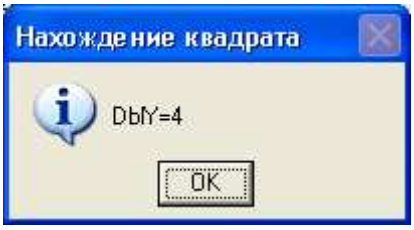

Рис. 4.72. Окно с сообщением, информационным значком и пользовательским заголовком

Определение кнопки, выбранной в окне сообщения. Функция `MsgBox` удобна для вывода той или иной информации. Однако если надо узнать, какой выбор сделал пользователь при нажатии отображаемых в диалоговом окне кнопок, то процедуру `MsgBox` надо использовать не как процедуру, а как функцию. В этом случае значение, возвращаемое функцией `MsgBox`, надо присваивать какой-то переменной, а ее параметры заключать в скобки.

Рассмотрим далее пример использования функции `MsgBox` с анализом нажатой кнопки. В приводимом примере на экране отображается диалоговое окно с двумя кнопками **Да** (выбрана по умолчанию), **Нет** и вопросительным знаком. При нажатии кнопки **Да** (нажатии клавиши **Enter**) выдается сообщение с текущей датой, а при нажатии кнопки **Нет** — пустое окно:

```
Sub DemoYesNo( )
    Dim IntAnswer As Integer
    IntAnswer = MsgBox( "Показать текущую дату?", _
        ebYesNo + ebQuestion + ebDefaultButton1 )
    Select Case IntAnswer
        Case ebYes
            MsgBox Now      ' Текущая дата и время
        Case ebNo
            MsgBox "", ebExclamation
        End Select
End Sub
```

4.6.1.29. Стандартные функции и операторы для работы с файлами

В языке VBA для работы с файлами можно использовать ряд стандартных функций и операторов, которые можно разделить на *следующие группы*, выполняющие:

- получение информации о файле EOF, FileDateTime, FileLen, FreeFile, Loc, LOF;
- управление файлами Dir, Kill, Name, FileCopy;
- задание или получение атрибутов файла FileAttr, GetAttr, SetAttr;
- открытие файла Open;
- получение или установку в файле позиции чтения-записи Seek;
- чтение данных из текстового файла Input, Input #, Line Input #;
- запись данных в текстовый файл Print #, Write #;

- закрытие файла Close.

4.6.1.29.1. Получение информации о файле

Функция EOF возвращает значение **True** при достижении конца открытого файла, **False** — в противном случае и имеет следующий синтаксис:

```
EOF( filenumber )
```

Здесь `filenumber` — номер файла, который был использован в операторе `Open` при открытии файла.

Функция FileDateTime возвращает значение **Variant**(**Date**), представляющее дату и время создания или последней модификации файла, и имеет следующий синтаксис:

```
FileDateTime( pathname )
```

Здесь `pathname` — строка, специфицирующая имя файла (может содержать полный путь, включая диск и папки). Приведем далее пример использования этой функции:

```
Sub DemoFileDateTime( )
    Dim VntMyStamp
    ' Файл t.txt модифицирован 7 января 2007 в 14:13:31
    VntMyStamp = FileDateTime( "t.txt " )
    ' Возвращает 7.01.2007 14:13:31
End Sub
```

Функция FileLen возвращает размер с типом **Long** указанного файла в байтах (до открытия файла) и имеет следующий синтаксис:

```
FileLen( pathname )
```

Здесь `pathname` — строка, специфицирующая имя файла (может содержать полный путь, включая диск и папки).

Функция LOF возвращает размер с типом **Long** открытого файла в байтах и имеет следующий синтаксис:

```
LOF( filenumber )
```

Здесь `filenumber` — номер файла, который был использован в операторе Open при открытии файла.

Функция FreeFile возвращает значение **Integer**, представляющее свободный последний номер открытого файла, и имеет следующий синтаксис:

```
FreeFile[( Rangenumber )]
```

Здесь `Rangenumber` — необязательный аргумент, определяющий диапазон для возвращаемого номера файла (при аргументе 0 диапазон от 0 до 255, соответствует умолчанию; при аргументе 1 диапазон от 256 до 511).

Функция Loc возвращает значение **Long**, представляющее текущую позицию чтения-записи в открытом файле, и имеет следующий синтаксис:

```
Loc( filenumber )
```

Здесь `filenumber` — номер файла, который был использован в операторе Open при открытии файла.

4.6.1.29.2. Управление файлами

Функция Dir возвращает значение **String**, представляющее имя файла, каталога или диска, соответствующее указанной маске или атрибуту файла, и имеет следующий синтаксис:

```
Dir[(pathname[, attributes])]
```

Здесь `pathname` — последовательность символов, обозначающая путь или имя файла (она может содержать маски и обозначения дисков); `attributes` — задает атрибуты искомых файлов (скрытый, системный или метка тома). Если последний аргумент не указан, то функция возвращает файлы без атрибутов, соответствующие маске `pathname`. Если опущены все аргументы, то функция возвращает очередное для текущего каталога имя подкаталога, служебное имя "." или ".." или имя файла. В качестве аргумента `attributes` могут использоваться комбинации атрибутов, приведенных в табл. 4.16.

***Таблица 4.16.** Атрибуты файлов*

Именованная константа	Значение	Описание
`vbNormal`	0	Обычный
`vbHidden`	2	Скрытый
`vbSystem`	4	Системный
`vbVolume`	8	Метка тома
`vbDirectory`	16	Каталог

Приведем далее пример использования этой функции:

```
Sub DemoDir( )
    Dim MyFile, MyPath, MyName
    ' При наличии возвращает "WIN.INI"
    MyFile = Dir( "C:\WINDOWS\WIN.INI" )
    ' Возвращает первый из файлов с расширением .ini из
    '   указанного каталога
    MyFile = Dir( "C:\WINDOWS\*.INI" )
    ' Возвращает первый *.TXT файл из текущего каталога
    MyFile = Dir( "*.TXT" )
    ' Отображает каталоги диска C:\
    MyPath = "c:\"
    MyName = Dir( MyPath, vbDirectory )   ' Первый каталог
    Do While MyName <> ""       ' Найдено непустое имя
        If MyName <> "." And MyName <> ".." Then
            ' Найденное имя не является служебными именем
            '   "." или "..". О GetAttr( ) см. далее в
            '   подразд. 1.29.3
            If ( GetAttr( MyPath & MyName ) And _
                 vbDirectory ) = vbDirectory Then
                Debug.Print MyName
            End If
        End If
        MyName = Dir
```

```
    Loop
End Sub
```

Оператор Kill удаляет указанный аргументом файл и имеет следующий синтаксис:

```
Kill pathname
```

Здесь `pathname` — строка, специфицирующая имя удаляемого файла (может содержать полный путь, включая диск и папки, и содержать маски * и ?). Приведем далее пример использования этого оператора:

```
Sub DemoKill( )
    Kill "TestFile" ' Удаление файла в текущем каталоге
    ' Удаление всех *.TXT файлов в текущем каталоге
    Kill "*.TXT"
End Sub
```

Оператор Name переименовывает файл или каталог и имеет следующий синтаксис:

```
Name oldpathname As newpathname
```

Здесь `pathname` — строка, специфицирующая имя переименуемого файла (может содержать полный путь, включая диск и папки); `newpathname` — строка, специфицирующая имя нового файла (может содержать полный путь, включая диск и папки). Приведем далее пример использования этого оператора:

```
Sub DemoName( )
    Dim OldName, NewName
    OldName = "OLDFILE": NewName = "NEWFILE"
    Name OldName As NewName    ' Переименование файла
    OldName = "C:\TMP\OLDFILE"
    NewName = "C:\TMP1\NEWFILE"
    ' Перемещение и переименование файла
    Name OldName As NewName
End Sub
```

Оператор FileCopy копирует файл и имеет следующий синтаксис:

```
FileCopy source, destination
```

Здесь `source` — строка, специфицирующая имя копируемого файла (может содержать полный путь, включая диск и папки); `destination` — строка, специфицирующая новое имя файла (может содержать полный путь, включая диск и папки). Приведем далее пример использования этого оператора:

```
Sub DemoFileCopy( )
    Dim SourceFile, DestinationFile
    SourceFile = "NEWFILE"
    DestinationFile = "COPY"
    FileCopy SourceFile, DestinationFile
End Sub
```

4.6.1.29.3. Задание или получение атрибутов файла

Функция FileAttr возвращает значение **Long**, представляющее режим, в котором находится открытый файл (табл. 4.17), и имеет следующий синтаксис:

```
FileAttr( filenumber, returntype )
```

Здесь `filenumber` — номер файла, который был использован при открытии файла оператором Open; `returntype` — имеет значение 2 для 32-разрядных систем.

Таблица 4.17. *Значения, возвращаемые функцией FileAttr*

Возвращаемое значение	Режим файла
1	Input
2	Output
4	Random
8	Append
32	Binary

Приведем далее пример использования этой функции:

```
Sub DemoFileAttr( )
    Dim FileNum, Mode
    FileNum = 1
    Open "t.txt" For Append As FileNum
    Mode = FileAttr( FileNum, 1 )    ' Возвращает 8
    Close FileNum
End Sub
```

Функция GetAttr возвращает значение **Integer**, представляющее атрибуты файла, каталога или диска (табл. 4.18). Если заданный файл имеет несколько атрибутов, то возвращается сумма соответствующих отдельным атрибутам значений. Функция имеет следующий синтаксис:

```
GetAttr( pathname )
```

Здесь `pathname` — специфицирует файл, каталог или диск.

Таблица 4.18. *Атрибуты файлов*

Именованная константа	Значение	Описание
`vbNormal`	0	Обычный
`vbReadOnly`	1	Только для чтения
`vbHidden`	2	Скрытый
`vbSystem`	4	Системный
`vbVolume`	8	Метка тома
`vbDirectory`	16	Каталог
`vbArchive`	32	Архивный

Оператор SetAttr устанавливает атрибуты файла и имеет следующий синтаксис:

```
SetAttr pathname, attributes
```

Здесь `pathname` — строка, специфицирующая имя файла (может содержать полный путь, включая диск и папки); `attributes` — специфицирует атрибуты файла (табл. 4.18, если заданный файл имеет несколько атрибутов, то задается сумма соответствующих отдельным атрибутам значений).

4.6.1.29.4. Открытие файла

Оператор Open выполняет открытие файла и имеет следующий синтаксис:

```
Open pathname For mode [Access access] [lock] As
```

```
[#]filenumber [Len=reclength]
```

где:

- `pathname` – путь к файлу и его полное имя.
- `mode` — режим доступа к файлу (Append: запись в конец; Output: запись в начало; Binary: двоичный доступ; Input: чтение; Random: чтение-запись, по умолчанию).
- `access` — тип доступа к файлу, необязательный параметр (Read: чтение, Write: запись, Read Write: чтение-запись).
- `lock` — необязательный параметр, определяющий совместный доступ к файлу (Shared: совместный доступ, Lock Read: блокировка чтения, Lock Write: блокировка записи, Lock Read Write: блокировка чтения и записи).
- `filenumber` — номер файла от 1 до 511.
- `reclength` — необязательный параметр, определяет размер буфера (не более 32767).

Пример использования оператора **Open** приведен ранее в описании функции FileAttr.

4.6.1.29.5. Получение или установка в файле позиции чтения-записи

Функция Seek позволяет получить или установить в файле позицию чтения-записи и может использоваться в двух синтаксических формах.

- Получение из файла позиции чтения-записи:

```
Seek( filenumber )
```

Здесь `filenumber` — номер файла, открытого оператором **Open**. Функция возвращает значение **Long** в диапазоне от 1 до 2^{31}-1, специфицирующее текущую позицию чтения-записи.

- Установка в файле позиции чтения-записи:

```
Seek [#]filenumber, position
```

Здесь `filenumber` — номер файла, открытого оператором **Open**; `position` — значение **Long** в диапазоне от 1 до 2^{31}-1, специфицирующее следующую текущую позицию чтения-записи.

4.6.1.29.6. Чтение данных из текстового файла

Особенность *текстовых файлов* заключается в способе доступа к данным, которые в них содержатся. Текстовый файл является файлом последовательного доступа, в котором содержится последовательность ASCII-символов. Текстовый файл может быть форматированным, то есть содержать пробелы, табуляторы, символы перехода на новую строку и другие специальные знаки. Чтение данных из текстового файла осуществляется с помощью функции Input, а также операторов **Input #** и **Line Input #**.

Функция Input является низкоуровневой и позволяет последовательно прочитать заданное в ней количество символов. Функция имеет следующий синтаксис:

```
Input( number, [#]filenumber )
```

Здесь number — целочисленный аргумент, определяющий число читаемых символов, начиная с текущего (вначале текущим является первый символ файла); filenumber — номер файла, открытого с помощью оператора **Open**.

Оператор Input # позволяет читать форматированные текстовые файлы, где в качестве разделителя используются запятые. Оператор имеет следующий синтаксис:

```
Input #filenumber, varlist
```

Здесь filenumber — номер файла, открытого с помощью оператора **Open**; varlist — список переменных, разделенных запятыми, которым последовательно присваиваются читаемые данные.

Оператор Line Input # выполняет посимвольное чтение очередной строки текстового файла. Символы окончания строки (CHR(13) или CHR(13)+ CHR(10)) в прочитанную строку не включаются. Оператор имеет следующий синтаксис:

```
Line Input #filenumber, varname
```

Здесь filenumber — номер файла, открытого с помощью оператора **Open**; varname — имя переменной с типом **Variant** или **String**, в которую помещается прочитанная строка. При выполнении этого оператора проверка достижения конца файла не производится. Поэтому для проверки достижения конца файла следует использовать функцию EOF.

В приведенном далее иллюстрирующем примере выполняется построчное считывание данных текстового файла с их выводом в окно **Immediate**, а затем читаются и выводятся в это же окно еще пять символов этого же файла, начиная с третьего от начала:

```
Sub Demo( )
    Dim StrLine
    Open "text.txt" For Input As #1
    On Error GoTo ErLabel:    ' Обработка ошибки
    While EOF( 1 ) = False
        Line Input #1, StrLine
        Debug.Print StrLine
    Wend
    Seek #1, 3
    Debug.Print Input( 5, #1 )
    Close #1
    Exit Sub
ErLabel:
    MsgBox "Ошибка!"
End Sub
```

4.6.1.29.7. Запись данных в текстовый файл

Запись в текстовый файл выполняется с помощью операторов **Print #** и **Write #**. Запись всегда производится в файл, открытый для записи (режим Output) или для добавления (режим Append). В первом случае данные, содержащиеся в файле, уничтожаются, а во втором — запись происходит в конец файла.

При использовании *оператора Write #* вывод происходит в одну строку, выводимые данные разделяются запятыми, а в конце строки вставляются символы перевода каретки и новой строки. Оператор имеет следующий синтаксис:

```
Write #filenumber, [outputlist]
```

Здесь `filenumber` — номер файла, открытого с помощью оператора **Open**; `outputlist` — имя переменной или список переменных, разделенных запятыми (разделителями могут быть также пробелы и точки с запятой). Приведем далее пример использования этого оператора:

```
Sub Demo( )
    ' Открытие файла для записи
    Open "TESTFILE" For Output As #1
    Write #1, "Hello World", 234 ' Разделитель - запятая
    Write #1,                    ' Запись пустой строки
    Dim MyBool, MyDate, MyNull, MyError
    MyBool = False: MyDate = #2/12/1969#: MyNull = Null
    MyError = CVErr( 32767 )
    Write #1, MyBool; " is a Boolean value"
    Write #1, MyDate; " is a date"
    Write #1, MyNull; " is a null value"
    Write #1, MyError; " is an error value"
    Close #1                     ' Закрытие Файла
End Sub
```

В результате содержимое файла TESTFILE получает следующий вид:

```
"Hello World",234

#FALSE#," is a Boolean value"
#1969-02-12#," is a date"
#NULL#," is a null value"
#ERROR 32767#," is an error value"
```

Оператор Print # является более гибким оператором записи в файл и имеет следующий синтаксис:

```
Print #filenumber, [{Spc(n)|Tab[(n)]}] [expression]
    [charpos]
```

Здесь `filenumber` — номер файла, открытого с помощью оператора **Open**; `Spc(n)|Tab[(n)` — опция, позволяющая указать тип (пробел или табулятор) и число (n) разделителей между значениями переменных; `expression` — список выражений, разделяемых запятыми, значения которых выводятся в файл; `charpos` — тип разделителя между выводимыми значениями (по умолчанию — точка с запятой). Если аргумент charpos опущен, то происходит переход к следующей строке. Приведем далее пример использования этого оператора:

```
Sub Demo( )
    Open "TESTFILE" For Output As #1
    Print #1, "This is a test"
    Print #1,
    Print #1, "Zone 1"; Tab; "Zone 2"
    Print #1, "Hello"; " "; "World"
```

```
    Print #1, Spc( 5 ); "5 leading spaces "
    Print #1, Tab( 10 ); "Hello"
    Dim MyBool, MyDate, MyNull, MyError
    MyBool = False: MyDate = #2/12/1969#: MyNull = Null
    MyError = CVErr( 32767 )
    Print #1, MyBool; " is a Boolean value"
    Print #1, MyDate; " is a date"
    Print #1, MyNull; " is a null value"
    Print #1, MyError; " is an error value"
    Close #1
End Sub
```

В результате содержимое файла TESTFILE получает следующий вид:

```
This is a test

Zone 1       Zone 2
Hello World
    5 leading spaces
        Hello
False is a Boolean value
12.02.1969  is a date
Null is a null value
Error 32767 is an error value
```

4.6.1.29.8. Закрытие файла

После окончания работы с открытым файлом его необходимо корректно закрыть. Автоматическое закрытие всех открытых файлов происходит при завершении работы приложения. Однако, при этом возможно возникновение различных неконтролируемых сбоев, что может привести к искажению файлов. Кроме того, незакрытый файл увеличивает требования к ресурсам.

Оператор Close предназначен для закрытия ранее открытого файла и имеет следующий синтаксис:

```
Close [filenumberlist]
```

Здесь `filenumberlist` — номер или список разделенных запятыми номеров файлов, которые надо закрыть (каждому номеру необязательно может предшествовать символ #). Если этот аргумент опущен, то будут закрыты все открытые ранее файлы.

> ***Примечание***
>
> Наряду с перечисленными ранее стандартными функциями и операторами для работы с файлами можно использовать объекты и коллекции объектов языка VBA, такие, как File Object, Files Collection, FileSystemObjtct Object, Folder Object, Folders Collection и TextStreamObject. Для получения справочной информации о них достаточно в интегрированной среде разработки (ИСР) VBA-программ в окне редактирования кода поместить курсор на любое служебное слово и нажать клавишу **F1**. В появившемся окне **Visual Basic Reference** следует выбрать вкладку **Содержание**, открыть раздел **Objects** и посмотреть темы File Object, Files Collection, FileSystemObjtct Object, Folder Object, Folders Collection и TextStreamObject. Дополнительную информацию о перечисленных ранее объектах и их коллекциях можно найти в [15, глава 6] и [14].

4.6.2. Использование языка VBA в SCADA-системе GeniDAQ

VBA-подобный язык, используемый в SCADA-системе GeniDAQ, представляет собой мощное средство программирования, совместимое с языком Microsoft Visual Basic для приложений и лицензированное у компании Summit Software Inc. Он является важным компонентом SCADA-системы GeniDAQ, обеспечивающим возможность разработки специализированных фрагментов приложения на языке программирования высокого уровня. Поскольку языки Visual Basic и их диалекты являются одними из самых популярных, простых для изучения и использования языков программирования, то наличие диалекта языка VBA в SCADA-системе GeniDAQ позволяет значительно упростить и повысить эффективность программирования.

Синтаксис используемой в SCADA-системе GeniDAQ версии языка VBA практически совместим с синтаксисом языков Microsoft VBA (используется, например, в приложениях Excel, Word, Access и т. п.) и Microsoft Visual Basic. Хотя версия языка VBA SCADA-системы GeniDAQ и имеет ряд несовместимых функций, но более 95% функций и процедур абсолютно идентичны. Поэтому в SCADA-системе GeniDAQ можно использовать программы, написанные на языке Visual Basic практически без изменений. Номера ошибок и сообщения об ошибках в VBA-программах SCADA-системы GeniDAQ также совместимы с имеющимися в языке Visual Basic. Специализированные, несовместимые функции и процедуры, введенные в язык VBA для привязки к SCADA-системе, обеспечивает не только привязку, но и возможность манипуляции задачами, входящими в приложение, взаимодействия с операционными системами Windows и другими приложениями посредством технологий DDE, OLE, ODBC (SQL) и COM-DCOM (OPC-серверы). Сказанное позволяет заключить, что при разработке VBA-программ для SCADA-системы GeniDAQ можно воспользоваться многочисленной литературой по языку VBA и приведенным ранее описанием этого языка.

> ***Замечание***
> В SCADA-системе GeniDAQ используется более ранняя версия языка VBA, чем описанная ранее и в современной литературе. Поэтому для уточнения имеющихся немногочисленных отличий целесообразно использовать встроенную в ИСР VBA-программ справочную систему, которую можно активизировать командой **Help | Help Topics** или путем нажатия клавиши **F1**.

Далее применительно к SCADA-системе GeniDAQ последовательно рассматриваются ИСР VBA-программы, встроенная в ИСР справочная система языка VBA, структура программного проекта приложения и средства привязки VBA-программы к использующему ее SCADA-приложению.

4.6.2.1. Интегрированная среда разработки VBA-программы

Вид ИСР VBA-программы показан на рис. 4.73. *Интегрированная среда разработки VBA-программы* представляет собой текстовый редактор с рядом удобных возможностей для редактирования и отладки кода VBA-программы. Исходный текст

будет компилироваться в псевдокод сразу после редактирования, так что не будет необходимости компилировать его вновь при запуске программы. Таким образом, ИСР представляет собой мощное средство разработки VBA-процедур обработки данных, специфичных для задачи приложения. Окно ИСР является модальным, что требует его закрытия перед продолжением работы.

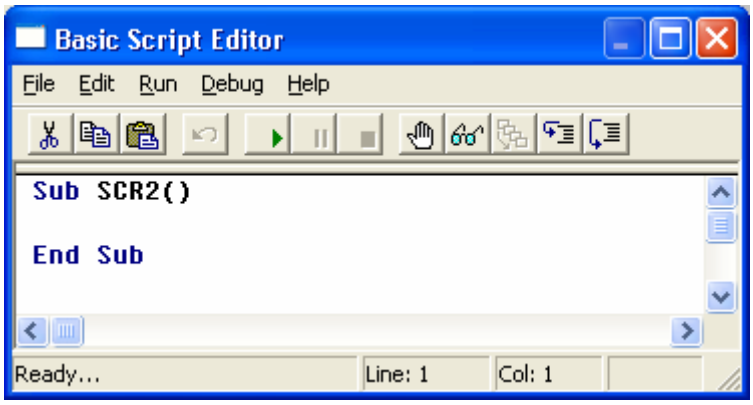

Рис. 4.73. Вид ИСР VBA-программ

В окне ИСР имеется *панель инструментов*, содержащая следующие кнопки:

(вырезать, cut) — удалить выделенный фрагмент текста и поместить его в универсальный буфер обмена Clipboard, действие этой кнопки эквивалентно выполнению команды **Edit | Cut** или активизации акселератора **Ctrl+X**;

(скопировать, copy) — поместить копию выделенного фрагмента текста в универсальный буфер обмена Clipboard, действие этой кнопки эквивалентно выполнению команды **Edit | Copy** или активизации акселератора **Ctrl+C**;

(вставить, paste) — поместить содержимое универсального буфера обмена Clipboard в текст, начиная от текущей позиции курсора, действие этой кнопки эквивалентно выполнению команды **Edit | Paste** или активизации акселератора **Ctrl+V**;

(отменить, undo) — отменить результат последней выполненной операции редактирования, действие этой кнопки эквивалентно выполнению команды **Edit | Undo** или активизации акселератора **Ctrl+Z**;

(исполнить, start) — начать исполнение программы, действие этой кнопки эквивалентно выполнению команды **Run | Start** или нажатию клавиши **F5**;

(прервать, break) — прервать исполнение программы и поместить указатель команд на следующую строку, подлежащую исполнению;

(остановить, end) — остановить исполнение программы, действие этой кнопки эквивалентно выполнению команды **Run | End**;

 (точка останова, toggle breakpoint) — добавить или удалить точку останова в текст программы, действие этой кнопки эквивалентно выполнению команды **Debug | Toggle Breakpoint** или нажатию клавиши **F9**;

 (контрольное значение, add watch) — вывести диалоговую панель **Add Watch** для просмотра значений переменных, действие этой кнопки эквивалентно выполнению команды **Debug | Add Watch...** или активизации акселератора **Shift+F9**;

 (вызовы, calls) — вывести на экран список процедур, вызванных текущей исполняющейся процедурой (кнопка доступна только после прерывания исполнения программы с помощью команды **прервать***);*

 (следующая инструкция, single step) — выполнить инструкцию в следующей строке процедуры и прервать исполнение (если в строке имеется вызов другой процедуры, то исполнение будет продолжено со следующей инструкции вызванной процедуры), действие этой кнопки эквивалентно выполнению команды **Debug | Single Step** или нажатию клавиши **F8**;

 (следующая процедура, procedure step) — выполнить инструкцию в следующей строке процедуры и прервать исполнение (если в строке имеется вызов другой процедуры, то данная процедура будет выполнена целиком), действие этой кнопки эквивалентно выполнению команды **Debug | Procedure Step** или активизации акселератора **Shift+F8**.

При помещении курсора в область отображения кнопки панели инструментов и удержании его в неподвижном состоянии в течение около 1 секунды в область ниже курсора будет выведено краткое описание данной кнопки (всплывающая подсказка). В нижней строке окна — строке статуса — при этом будет выдана более подробная справка о действии кнопки.

Для поиска синтаксических ошибок в тексте процедуры можно использовать команду **Run | Syntax Check**. На экран монитора будет выведено сообщение, указывающее на наличие или отсутствие ошибок в тексте процедуры. Сообщение об ошибке относится к первой строке, в которой была обнаружена ошибка, и содержит краткое пояснение причины, по которой синтаксис в указанной строке признан неверным. При нажатии на кнопку **OK** в диалоговом окне сообщения об ошибке или нажатии клавиши **Enter** строка процедуры, в которой обнаружена синтаксическая ошибка, будет выделена в окне редактирования. При попытке запуска на исполнение или отладки процедуры, для которой не была проведена проверка синтаксиса, указанная проверка будет выполнена автоматически перед началом исполнения.

При завершении работы ИСР (команда **File | Close**, акселератор **Ctrl+W**) результат выполнения операции зависит от того, были ли внесены какие-либо изменения в исходный текст процедуры, а также от наличия синтаксических ошибок в исходном тексте. Если в процессе работы в исходный текст процедуры были внесены какие-либо изменения, на экран монитора будет выведена диалоговое окно с запросом сохранения выполненных изменений. При этом, если исходный текст сценария содержит синтаксические ошибки и выбрана (нажата) кнопка **Да** диалогового окна, на экран будет выведено окно, содержащее сообщение о наличии ошибок и запрос на подтверждение намерения пользователя продолжить сохранение программы.

Нажатие кнопки **Да** данного диалогового окна приведет к сохранению исходного текста сценария вместе со всеми имеющимися ошибками. Если в исходный текст сценария не было внесено никаких изменений, произойдет мгновенное завершение работы ИСР независимо от наличия ошибок в исходном тексте процедуры, допущенных во время предыдущего сеанса работы.

В окне ИСР с помощью команды **Help | Help Topics** можно получить справочную информацию о языке, используемом для создания VBA-программы.

4.6.2.2. Структура VBA-программы (программный проект приложения)

В общем случае, *VBA-программа (программный проект приложения)* содержит следующие компоненты.

- *Главная процедура* (главный сценарий, Main Script) программного проекта приложения, будучи включенной в программный проект, осуществляет управление исполнением всего приложения один раз при его запуске. Вид ИСР главной процедуры показан на рис. 4.74. Для включения в программный проект приложения главной процедуры достаточно выполнить в главном окне SCADA-системы команду **File | Add/Delete | Add Main Script** (рис. 4.75). Исключение из программного проекта приложения главной процедуры выполняется командой **File | Add/Delete | Delete Main Script** (см. рис. 4.75). Главная процедура может быть применена для выполнения таких операций, как запуск, останов задач приложения и т. п. Главная процедура является необязательным элементом программного проекта приложения.

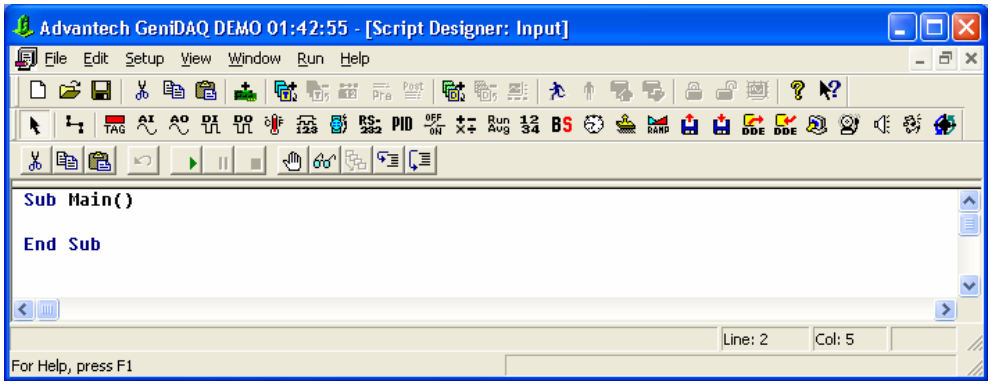

Рис. 4.74. Вид ИСР главной процедуры

- *Предварительная процедура (Pre-Task Script) и пост-процедура (Post-Task Script) задачи*. Каждая сканируемая задача приложения имеет предварительную процедуру и пост-процедуру. Предварительная процедура выполняется один раз перед первым запуском задачи, а пост-процедура — один раз после последнего завершения задачи. Для их редактирования достаточно выполнить команду **Setup | Pre task Script** или команду **Setup | Post task Script** (рис. 4.76 и 4.77). По

умолчанию эти процедуры не выполняют никаких действий (см. рис. 4.76 и 4.77). Указанные два типа процедур при определенных условиях используются для инициализации или сброса значений, связанных с объектами, которые входят в приложения.

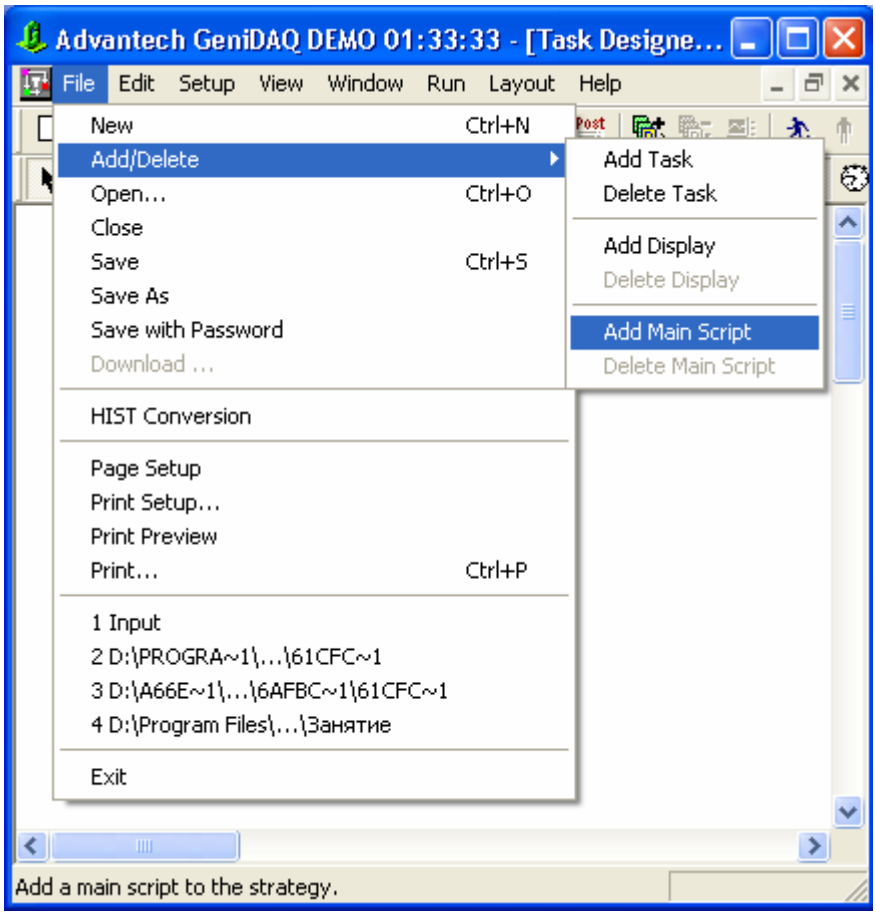

Рис. 4.75. Включение главной процедуры в программный проект приложения

- *Бейсик-процедура* (Бейсик-процедура, Basic Script) автоматически включается в программный проект при включении в задачу приложения функционального блока **BASIC Script**. Таким образом, число Бейсик-процедур программного проекта совпадает с общим числом функциональных блоков **BASIC Script** в приложении. Использование Бейсик-процедур обеспечивает максимальную гибкость при программировании специализированных вычислительных процедур и процедур сбора данных и управления. Вид ИСР Бейсик-процедуры показан ранее на рис. 4.73.

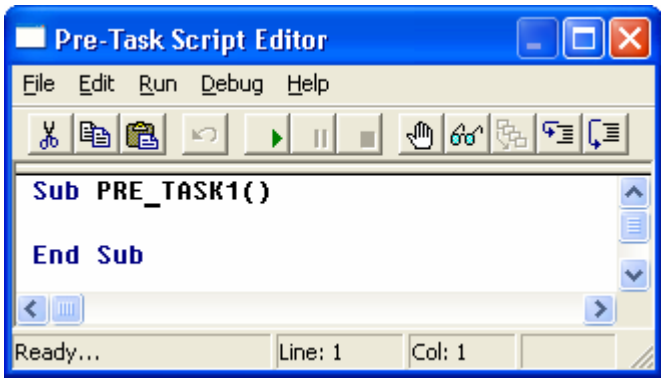

Рис. 4.76. ИСР предварительной процедуры задачи программного проекта приложения

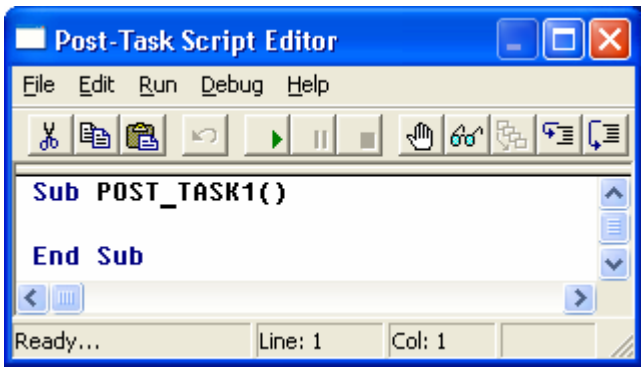

Рис. 4.77. ИСР пост-процедуры задачи программного проекта приложения

4.6.2.3. Справочная система ИСР VBA-программ

Как было указано ранее, *справочную информацию* о языке, используемом для создания VBA-программы, можно получить в ИСР с помощью команды **Help | Help Topics** (рис. 4.78).

При активизации гиперссылки **Command Reference** получаем алфавитно-упорядоченный список гиперссылок на справки об элементах языка BasicScript Language (рис. 4.79).

Аналогичным образом, при активизации гиперссылок **Directives ... Objects** получаем соответствующие алфавитно-упорядоченные списки гиперссылок на справки об элементах языка BasicScript Language по функциональным группам. И, наконец, при активизации гиперссылки **BasicScript Help Contents** получаем окно справки, представленное на рис. 4.80. Если в этом окне активизировать гиперссылку **Dialog Editor**, можно получить алфавитно-упорядоченный список гиперссылок на справки о редакторе диалога (рис. 4.81).

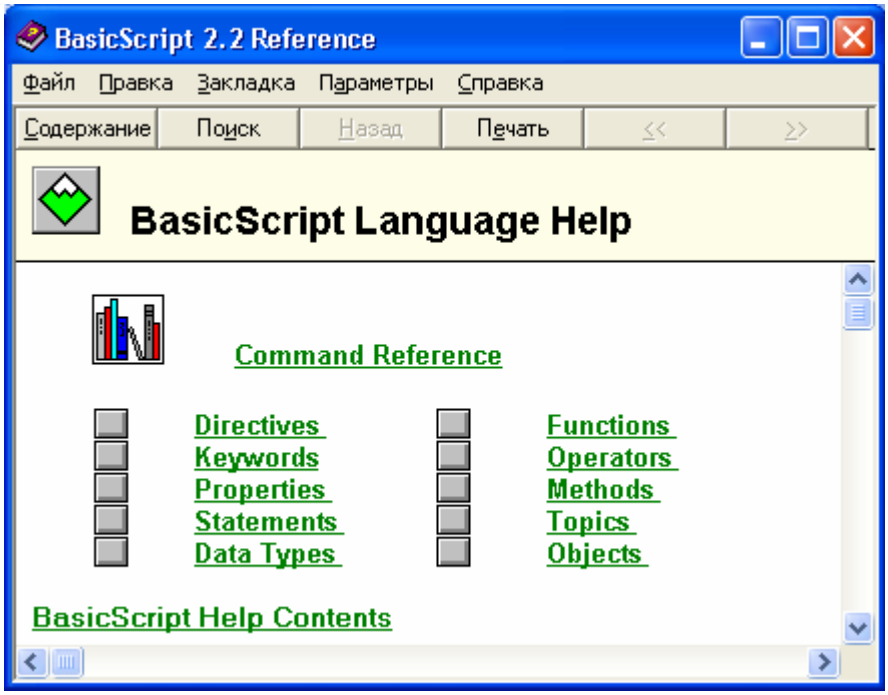

Рис. 4.78. Окно справки ИСР VBA-программ

4.6.2.4. Привязка VBA-программы к использующему ее приложению

Для реализации управления задачами и данными в процессе работы приложения в язык VBA включен ряд специализированных средств. К их числу относятся средства управления задачами, работы с тегами, управления окнами экранных форм и работы с Бейсик-процедурами.

4.6.2.4.1. Управление задачами

Средства управления задачами предназначены для управления сканируемыми задачами, входящими в приложение. Указанные средства могут применяться в главной процедуре, а также в предварительной процедуре и пост-процедуре задачи (задач). К средствам управления задачами относятся следующие.

Тип ScanTask — тип объекта *сканируемая задача*. Этот тип используется для определения переменной, которая позволяет получить доступ к объекту, возвращаемому функцией `GetScanTask( )`:

```
' Пример 4.1
' В основном сценарии стратегии выполняется запуск задачи
'   TASK1
Sub Main( )
    Dim MyTask As ScanTask
```

```
    Set MyTask = GetScanTask( "TASK1" )
    MyTask.Start
End Sub
```

См. также `GetScanTask` (функция), `Start` (метод).

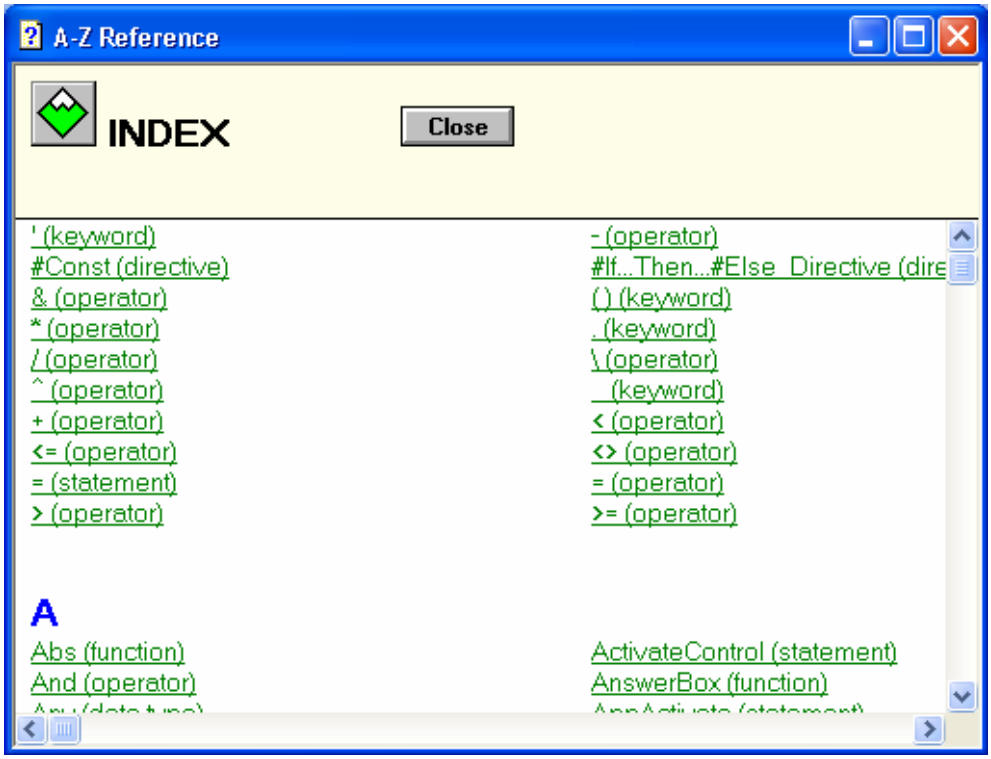

Рис. 4.79. Алфавитно-упорядоченный список гиперссылок на справки об элементах языка BasicScript Language

Функция GetScanTask(имя_задачи) — при успешном завершении возвращает объект типа **ScanTask**. Если задача с заданным именем отсутствует или имя указано неверно, возвращается значение **Null**. Объект типа **ScanTask** является именем переменной типа **ScanTask**. Имя_задачи является символьной строкой, идентифицирующей задачу, которая входит в приложение. Имя задачи должно быть представлено прописными буквами и заключено в кавычки, например `"TASK1"`.

Приведенный ранее пример 4.1 иллюстрирует также использование этой функции. См. также ScanTask (тип объекта), Start (метод).

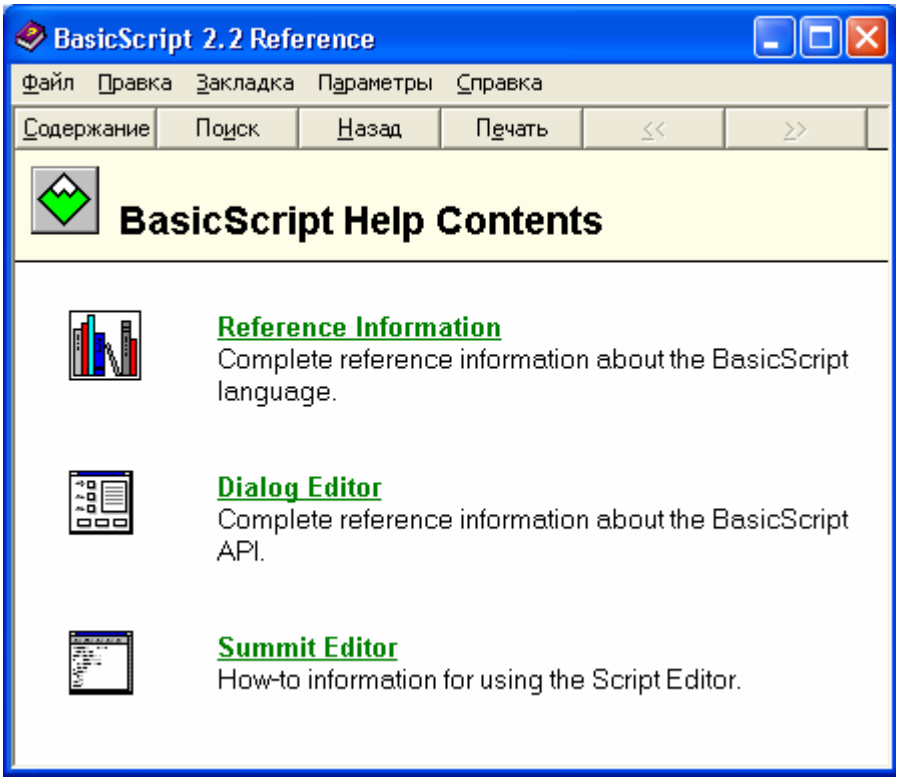

Рис. 4.80. Окно справки BasicScript Help Contents

Метод Start, выполняемый для объекта типа **ScanTask** (задача), предназначен для включения задачи, для которой вызывается указанный метод, в список задач, вызываемых исполнительной средой в процессе выполнения стратегии. Метод может быть применен к объекту типа **ScanTask**. Указанными объектами являются задачи, исполняющиеся в рамках приложения. При вызове указанного метода производится немедленный запуск заданной задачи независимо от режима запуска, установленного в диалоговом окне **Scan Task Setup** редактора задач, появляющемся при выполнении команды **Setup | Task Properties... (Ctrl+T)**. Возврат управления из данного метода происходит сразу после активизации задачи, для которой он вызывается.

Приведенный ранее пример 4.1 иллюстрирует использование и этого метода. См. также **ScanTask** (тип объекта), GetScanTask (функция).

Метод Stop предназначен для исключения задачи, для которой вызывается указанный метод, из списка задач, вызываемых исполнительной средой в процессе выполнения приложения. Метод может быть применен к объекту типа **ScanTask**. Указанными объектами являются задачи, исполняющиеся в рамках приложения.

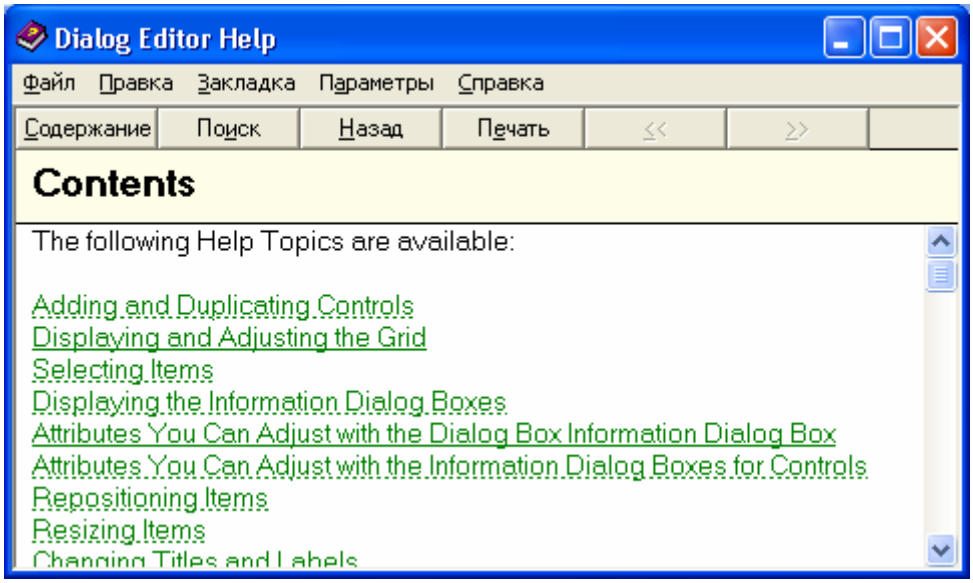

Рис. 4.81. Окно справки для создания диалогового окна

```
'Пример 4.2
'В данном примере останавливается задача TASK1.
'  Предполагается, что эта задача уже запущена
Sub Main( )
    Dim MyTask As ScanTask
    Dim IntFileLength As Integer
    Set MyTask = GetScanTask( "TASK1" )
    ' Открываем файл для ввода и присваиваем ему номер 1
    Open "test.dat" For Input As #1
    ' Определяем размер файла
    IntFileLength = Lof( 1 )
    ' Если файл слишком большой, то останавливаем задачу
    If IntFileLength > 32000 Then
        MyTask.Stop
    End If
End Sub
```

См. также **ScanTask** (тип объекта), GetScanTask (функция).

Метод SingleScan, выполняемый для объекта типа **ScanTask** (задача), предназначен для однократного исполнения задачи, для которой вызывается указанный метод. При вызове указанного метода производится немедленное однократное исполнение заданной задачи независимо от периода запуска, установленного в диалоговом окне **Scan Task Setup** редактора задач, появляющемся при выполнении команды **Setup — Task Properties...** (**Ctrl+T**). Возврат управления из данного метода происходит только по завершении однократного исполнения задачи, для которой он вызывается.

```
'Пример 4.3
'В данном примере 100 раз вызывается задача TASK1
```

```
Sub Main( )
    Dim IntI As Integer
    Dim MyTask As ScanTask
    Set MyTask = GetScanTask( "TASK1" )
    For IntI = 1 To 100
        MyTask.SingleScan
    Next IntI
End Sub
```

См. также **ScanTask** (тип), GetScanTask (функция).

Метод GetStatus, выполняемый для объекта типа **ScanTask** (задача), предназначен для получения текущего состояния задачи, для которой вызывается указанный метод. Результатом выполнения данного метода является возвращаемое целочисленное значение, отражающее текущее состояние проверяемой задачи. Значение 0 означает, что проверяемая задача не запущена и не остановлена. Значение 2 означает, что проверяемая задача находится в состоянии ожидания очередного вызова в соответствии с периодом сканирования, установленным в диалоговом окне **Scan Task Setup** редактора задач, появляющемся при выполнении команды **Setup | Task Properties...** (**Ctrl+T**), или выполнение задачи приостановлено на заданный интервал времени. Значение 3 означает, что проверяемая задача находится в состоянии исполнения во время очередного вызова в соответствии с периодом сканирования, установленным в диалоговом окне **Setup | Task Properties...** (**Ctrl+T**). Рекомендуется использовать данный метод до запуска или для остановки исполнения задачи.

```
' В примере 4.4 выполняется проверка состояния задачи TASK1
Sub Main( )
    Dim IntMyStatus As Integer
    Dim MyTask As ScanTask
    Set MyTask = GetScanTask( "TASK1" )
    MyStatus = MyTask.GetStatus
    If IntMyStatus = 0 Then
        MyTask.Start
        MyTask.Stop
    End If
End Sub
```

См. также **ScanTask** (тип), GetScanTask (функция), Start (метод), Stop (метод).

4.6.2.4.2. Работа с тегами

Средства работы с тегами предназначены для получения доступа к функциональным блокам и другим тегам, расположенным в центре обработки данных SCADA-системы. Они могут применяться в любой компоненте программного проекта (в основной процедуре, в предварительной и пост-процедурах задач, а также в Бейсик-процедурах). Понятие *тег* в определенном контексте может ассоциироваться как с идентификатором объекта (имя тега), так и с самим объектом определенного типа (класса). Примером *тега* является идентификатор функционального блока, входящего в задачу, однако имя тега записывается следующим образом. Для блока аналогового ввода, поле **Tag** диалогового окна

настройки параметров которого содержит идентификатор AI1, а поле **Description** — АЦП1, имя тега представляется как AI1:АЦП1.

Тип Tag используется для определения переменной, которая позволяет получить доступ к объекту, возвращаемому функцией GetTag( ).

```
' В примере 4.5 считывается значение на канале 0 блока
'   аналогового ввода
Sub Main( )
    Dim MyTag As Tag
    Dim SngVoltage As Single
    Set MyTag = GetTag( "TASK2", "AI1" )
    SngVoltage = MyTag.Value
End Sub
```

См. также GetTag (функция), Value (свойство).

Функция GetTag при успешном выполнении возвращает объект типа **Tag** и имеет два аргумента. Первый аргумент является строкой и задает имя задачи, формы отображения или виртуальной задачи ("VIRTASK"), а второй также является строкой, содержащей имя функционального блока, элемента управления или виртуального тега (должно быть представлено прописными буквами).

Приведенный ранее пример 4.5 иллюстрирует использование и этой функции. См. также **Tag** (тип объекта), Value (свойство).

Свойство Value предназначено для получения доступа к значению, связанному с объектом типа **Tag**. Объект типа **Tag** в этом случае должен указывать на функциональный блок, имеющий один или несколько входов.

Приведенный ранее пример 4.5 иллюстрирует использование и этого свойства. См. также GetTag (функция), **Tag** (тип объекта).

Свойство Array(номер_канала) предназначено для получения доступа к значению, связанному с заданным каналом объекта типа **Tag**. Объект типа **Tag** в этом случае должен указывать на функциональный блок, имеющий один или несколько входов. Номер канала может принимать значения от 0 до 15.

```
' В примере 4.6 считывается значение на 16-ом канале блока
'   аналогового ввода
Sub Main( )
    Dim MyTag As Tag
    Dim SngVoltage As Single
    Set MyTag = GetTag( "TASK2", "AI1" )
    SngVoltage = MyTag.Array( 15 )
End Sub
```

См. также GetTag (функция), **Tag** (тип объекта).

4.6.2.4.3. Управление окнами экранных форм

Функция Display(номер_окна) используется в основной, предварительной и пост-процедурах для переключения между несколькими окнами форм отображения, входящими в приложение. В качестве параметра оператору передается номер окна экранной формы. Например, для выдвижения на передний план окна экранной формы DISP1 следует использовать вызов функции `Display( 1 )`.

```
' Пример 4.7 основного сценария стратегии
Sub Main( )
    Dim IntMyStatus As Integer
    Dim MyTask As ScanTask
    Set MyTask = GetScanTask( "TASK1" )
    IntMyStatus = MyTask.GetStatus
    If IntMyStatus = 3 Then
        ' Отображение окна DISP1 на переднем плане
        Display( 1 )
    End If
End Sub
```

4.6.2.4.4. Работа с Бейсик-процедурами

Средства работы с Бейсик-процедурами предназначены для управления выходными каналами функциональных блоков Бейсик-сценарий. Указанные средства могут применяться *только в Бейсик-процедурах*. Далее приводится описание операторов, выполняющих управление выходными каналами функциональных блоков Бейсик-сценарий.

Оператор Outputf используется для передачи одного вещественного значения из блока Бейсик-сценария другому функциональному блоку приложения и имеет два формата:

```
Outputf значение
Outputf канал, значение
```

Здесь `канал` — целое число в диапазоне от 0 до 7, представляющее номер канала, по которому будет осуществляться передача вещественного значения, а `значение` — имя переменной типа **Single** или значение, передаваемое другому функциональному блоку приложения. Если явно не указан номер канала, по которому выводится значение с плавающей точкой, предполагается канал с номером 0.

```
' В примере 4.8 выполняется вывод вещественного значения по
'   каналу 0
    Dim MyTag As Tag
    Dim SngVoltage As Single
    Set MyTag = GetTag( "TASK2", "AI1" )
    SngVoltage = MyTag.Value
    If SngVoltage > 0.0 Then
        Outputf SngVoltage
    Else
        Outputf -SngVoltage
    End If
```

См. также Outputi (оператор), Outputl (оператор), Outputs (оператор)

Оператор Outputi используется для передачи одного целочисленного значения из блока Бейсик-сценария другому функциональному блоку приложения и имеет два формата:

```
Outputi значение
Outputi канал, значение
```

Здесь `канал` — целое число в диапазоне от 0 до 7, представляющее номер канала, по которому будет осуществляться передача целочисленного значения, а `значение` —

имя переменной типа **Integer** или целочисленное значение, передаваемое другому функциональному блоку приложения. Если явно не указан номер канала, по которому выводится значение с плавающей точкой, предполагается канал с номером 0.

```
' В примере 4.9 выполняется вывод целочисленного значения по
'   каналу 0
    Dim MyTag As Tag
    Dim IntRamp As Integer
    Set MyTag = GetTag( "TASK2", "RMP1" )
    IntRamp = MyTag.Value
    If IntRamp > 3 Then
        Outputi IntRamp
    Else
        Outputi -1
    End If
```

См. также Outputf (оператор), Outputl (оператор), Outputs (оператор)

Оператор Outputl используется для передачи одного целочисленного значения двойной точности из блока Бейсик-сценария другому функциональному блоку приложения и имеет два формата:

```
Outputl значение
Outputl канал, значение
```

Здесь `канал` — целое число в диапазоне от 0 до 7, представляющее номер канала, по которому будет осуществляться передача целочисленного значения двойной точности, а `значение` — имя переменной типа **Long** или целочисленное значение двойной точности, передаваемое другому функциональному блоку приложения. Если явно не указан номер канала, по которому выводится значение с плавающей точкой, предполагается канал с номером 0.

```
' В примере 4.10 выполняется вывод значения целого
'   типа двойной точности по двум каналам
    Dim LngCount As Long
    ' …
    If LngCount >= 0 Then
        Outputl 0, LngCount
    Else
        Outputl 1, LngCount
    End If
```

См. также Outputf (оператор), Outputi (оператор), Outputs (оператор).

Оператор Outputs используется для передачи одной строки из блока Бейсик-сценария другому функциональному блоку приложения и имеет два формата:

```
Outputl строка$
Outputl канал, строка$
```

Здесь `канал` — целое число в диапазоне от 0 до 7, представляющее номер канала, по которому будет осуществляться передача строки, а `значение` — имя переменной типа **String** или строковое значение, передаваемое другому функциональному блоку приложения. Если явно не указан номер канала, по которому выводится значение с плавающей точкой, предполагается канал с номером 0.

```
' В примере 10 выполняется вывод строки
    Dim StrS1 As String
    Dim StrS2 As String
    StrS1 = "Hello,"
    StrS2 = "World!"
    Outputs 3, StrS1 + StrS2
```

См. также Outputf (оператор), Outputi (оператор), Outputl (оператор).

4.6.2.5. Использование в VBA-программе диалоговых окон. Управляющие элементы

Проиллюстрируем создание и использование окна диалога на примере диалогового окна с кнопками **OK**, **Cancel**, управляющими элементами отмечаемая кнопка (**Check box**), элемент группирования (**Group box**), статический текст (**Text**), элемент редактирования (**Text box**)и раскрывающийся список (**Drop list box**).

Для создания демонстрационного примера выполните следующие действия.

1. *Первое действие*. Запустите построитель приложений двойным щелчком левой кнопкой мыши на ярлыке GeniDAQ Builder 4.25, расположенном на рабочем столе. В результате на экран будет выведено главное окно построителя приложений (см. рис. 4.6). Для запуска построителя задач в демонстрационном режиме нажмите кнопки **Exit** и **OK**. Демонстрационный режим отличается непрерывным двухчасовым интервалом работы и меньшим количеством входных и выходных каналов. В нашем случае это несущественно. Нажмите кнопку **New** на панели инструментов (ее легко найти, так как все кнопки снабжены всплывающими подсказками). После этого в главном окне построителя приложений появится диалоговое окно настройки, которое может быть настроено так, как это показано на рис. 4.82. После нажатия кнопки **OK** и подтверждения необходимости создания каталога DemoDialog, который до этого не существовал, в главном окне построителя задач появятся пустые окна редактора задач (**TASK1**) и редактора форм отображения (**DISP1**). Произведем щелчок мышью в области окна редактора задач и окно с заголовком **Task Designer: TASK1** будет выдвинуто на передний план. В окне построителя задач появится еще одна панель инструментов, содержащая пиктограммы встроенных функциональных блоков редактора задач.

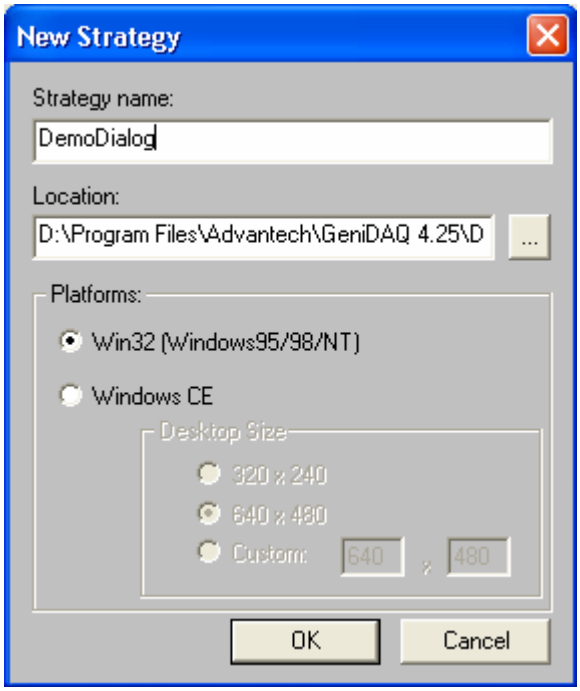

Рис. 4.82. Конфигурирование приложения **DemoDialog**

2. *Второе действие*. Задайте временные параметры запуска задачи. Для этого нажмите кнопку **Task Properties** на панели инструментов или выполните команду **Setup | Task Properties**, в появившемся окне задайте требуемое значение периода (см. рис. 4.8) и нажмите кнопку **OK**. В нашем примере задан период запуска приложения 100 миллисекунд.
3. *Третье действие*. На панели инструментов выберите кнопку с мнемоникой **BS** (BASIC Script), поместите на нее курсор мыши, щелкните левой кнопкой мыши, переместите мышь в окно редактора задач и еще раз щелкните левой кнопкой мыши. При этом в окне редактора задач появится функциональный блок Бейсик-сценария с позиционным обозначением **SCR1** (рис. 4.83).
4. *Четвертое действие*. Теперь сделайте активным окно **DISP1** редактора форм отображения, для чего щелкните по этому окну левой кнопкой мыши. При этом в верхней части главного окна появится панель инструментов редактора форм отображения. На панели инструментов выберите кнопку **Binary Button** (нажимаемая кнопка) и с помощью мыши перетащите ее и положите во внутреннюю область окна редактора форм отображения (рис. 4.84). Для настройки свойств кнопки щелкните дважды левой кнопкой мыши над пиктограммой кнопки. В результате появится диалоговое окно настройки. Выполните настройку кнопки в соответствии с рис. 4.85, а затем с помощью

кнопки **OK** закройте окно (в ответ на предупреждение еще раз нажмите кнопку **OK**). С помощью этой кнопки будем управлять окном диалога.

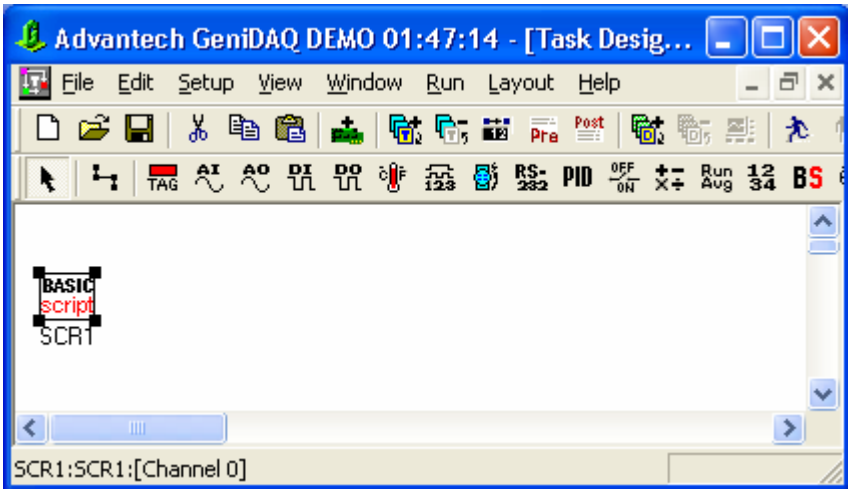

Рис. 4.83. Вид окна редактора задач приложения DemoDialog

5. *Пятое действие*. Теперь сделайте вновь активным окно **TASK1** редактора задач, для чего щелкните по этому окну левой кнопкой мыши. При этом в верхней части главного окна появится панель инструментов редактора задач. На панели инструментов выберите кнопку **Tag** (тег) и с помощью мыши перетащите ее и положите во внутреннюю область окна редактора форм отображения (рис. 4.86). Для настройки свойств тега щелкните дважды левой кнопкой мыши над пиктограммой тега. В результате появится диалоговое окно настройки. Выполните настройку тега в соответствии с рис. 4.87, а затем с помощью кнопки **OK** закройте окно. Созданный тег будет использован для передачи в Бейсик-процедуру информации о состоянии кнопки в окне отображения.
6. *Шестое действие*. Двойным щелчком левой кнопки мыши по функциональному блоку Бейсик-сценария активируйте ИСР Бейсик-процедуры и выполните команду **Edit | Insert New Dialog…**. В результате появится ИСР диалогового окна, представленная на рис. 4.88. Разместите в шаблоне окна диалога управляющие элементы группирования, статического текста, редактирования, отмечаемую кнопку и раскрывающийся список в соответствии с рис. 4.89. Настройте диалоговое окно и добавленные управляющие элементы в соответствии с рис. 4.90-4.95. Командой **File | Save As…** сохраните созданный диалог в файле не магнитном диске, а командой **File | Exit And Return** включите шаблон диалога в Бейсик-процедуру.
7. *Седьмое действие*. Выполните программную поддержку созданного диалога. С этой целью в Бейсик-процедуру введите следующий текст:

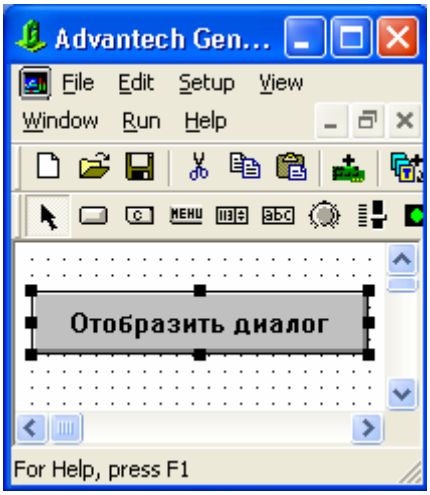

Рис. 4.84. Окно редактора форм отображения приложения **DemoDialog**

```
' Создание и обработка диалога
' Массив для раскрывающегося списка
Public DropListBox1$( 2 )
' Шаблон диалога (создается автоматически)
Begin Dialog UserDialog ,,264,113,"TestDialog"
    OKButton 220,8,40,14
    CancelButton 220,28,40,14
    GroupBox 4,4,208,104,"Блок группирования",.GroupBox1
    TextBox 8,16,44,12,.TextBox1
    Text 60,17,108,8,"Элемент редактирования",.Text1
    CheckBox 8,36,160,8,"Отмечаемая кнопка",.CheckBox1
    DropListBox 8,50,68,58,DropListBox1$,.DropListBox1
    Text 84,53,124,8,"Раскрывающийся список",.Text2
End Dialog
' Инициализация, создание и обработка диалога
Sub SCR1()
    ' Тег кнопки, управляющей диалогом
    Dim button As Tag
    ' Получение значения состояния кнопки
    Set button = GetTag( "DISP1", "BBTN1" )
    If button.value = 1 Then
        ' Кнопка нажата - инициализация, создание и обработка
        '   диалога
        ' Инициализация
        ' Данные для раскрывающегося списка
        DropListBox1$( 0 ) = "Один"
        DropListBox1$( 1 ) = "Два"
        DropListBox1$( 2 ) = "Три"
        Dim dlg As UserDialog           ' Объект диалога
        ' Индекс элемента раскрывающегося списка
        dlg.DropListBox1 = 2
```

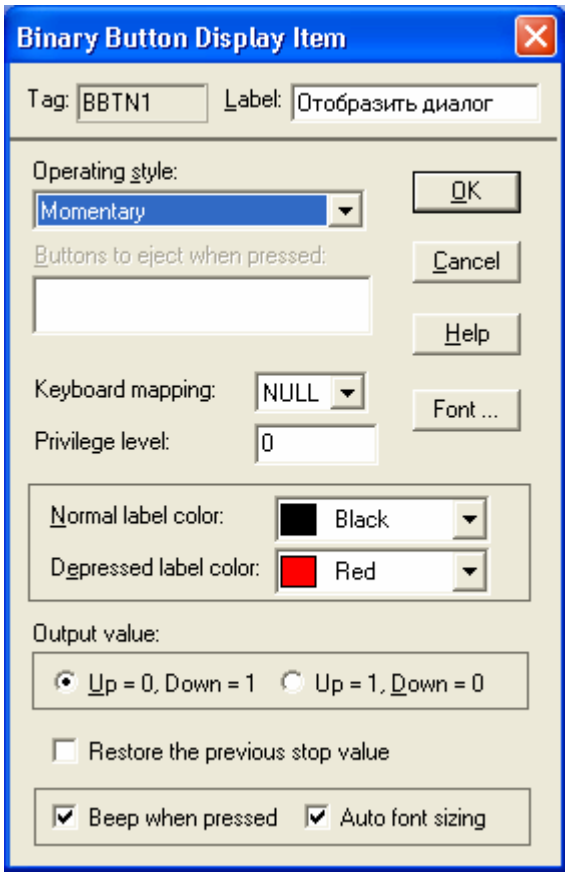

Рис. 4.85. Конфигурирование кнопки

```
' Текст для элемента редактирования
dlg.TextBox1 = "Пример"
' Отмечаемая кнопка выбрана
dlg.CheckBox1 = 1
' Для возвращаемого элемента диалога (-1 при нажатии кнопки
'   OK)
Dim IntRC As Integer
' Создание диалога и работа с ним
IntRC = Dialog( dlg )
' Обработка диалога
If IntRC = -1 Then
    ' Нажата кнопка OK
    ' Состояние раскрывающегося списка
    MsgBox "Раскрывающийся список: " & _
        DropListBox1$( dlg.DropListBox1 )
    ' Состояние элемента редактирования
    MsgBox "Элемент редактирования: " & dlg.TextBox1
```

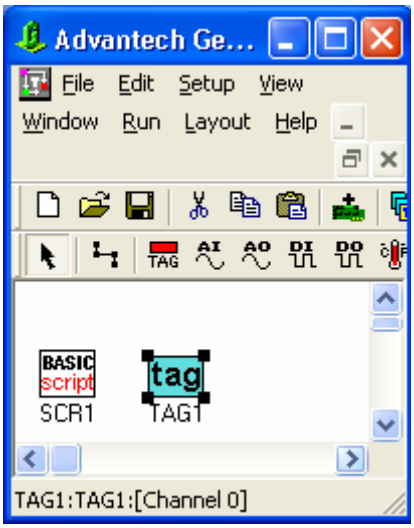

Рис. 4.86. Окно редактора задач приложения DemoDialog

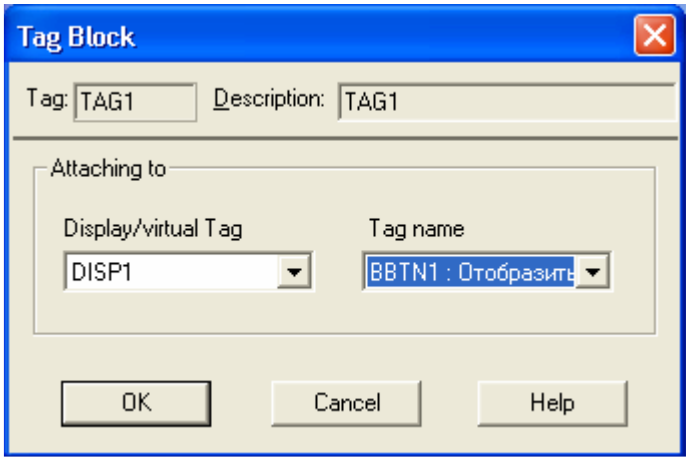

Рис. 4.87. Конфигурирование тега

```
            ' Состояние отмечаемой кнопки
            MsgBox "Отмечаемая кнопка: " & dlg.CheckBox1
        End If
    End If
End Sub
```

Командой **File | Close** закройте ИСР программы и сохраните Бейсик-процедуру.

8. *Восьмое действие*. Протестируйте созданное приложение. С этой целью кнопкой **Start** активизируйте приложение и в окне отображения нажмите кнопку **Отобразить диалог**. В результате появится диалоговое окно, представленное на

рис. 4.96. В этом окне изменим текст в элементе редактирования, состояние отмечаемой кнопки, выберем другой элемент в раскрывающемся списке и для считывания информации из окна диалога нажмем кнопку **OK**. В результате появится последовательность сообщений, отображающих полученную информацию (рис. 4.97 — 4.99).

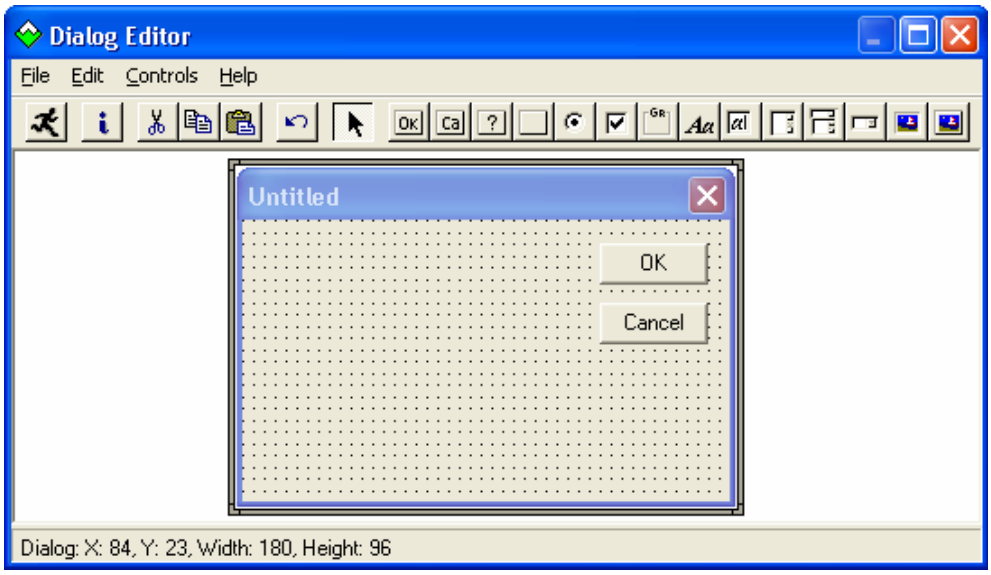

Рис. 4.88. ИСР диалогового окна

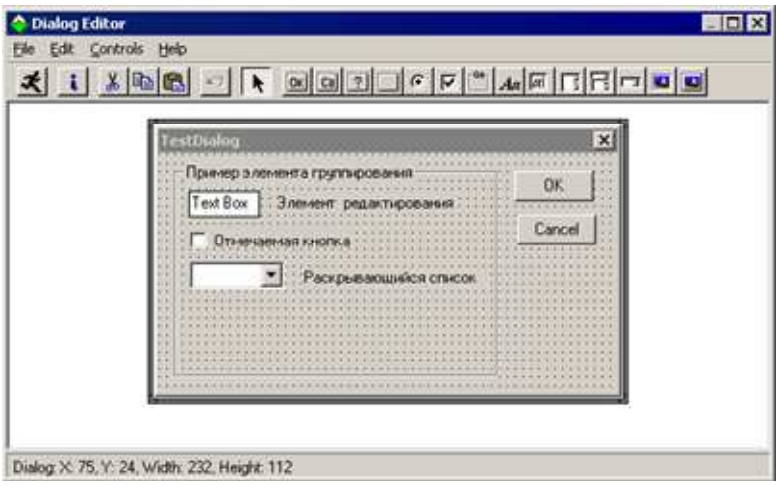

Рис. 4.89. Вид окна диалога

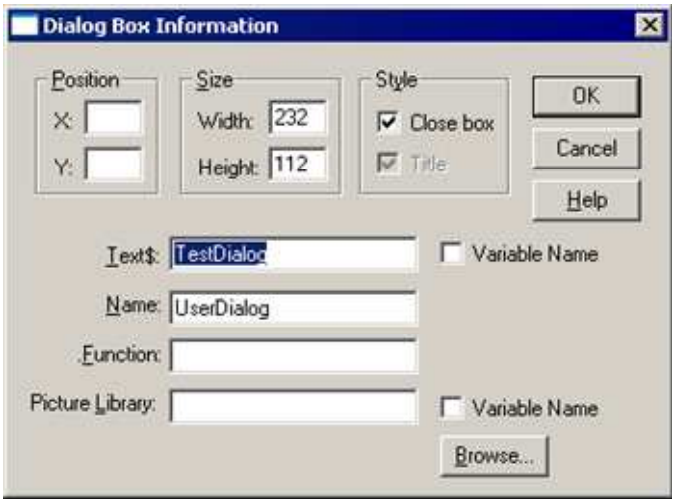

Рис. 4.90. Конфигурирование окна диалога

Рис. 4.91. Конфигурирование элемента группирования

9. *Девятое действие*. Завершите работу приложения. С этой целью кнопкой **Stop** остановите работу приложения и командой **File | Exit** завершите работу SCADA-системы.

4.6.3. Упражнения

Упражнение 4.17. Повтор создания и тестирования приложения DemoDialog

Повторите создание и тестирование рассмотренного ранее приложения **DemoDialog**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Рис. 4.92. Конфигурирование элемента редактирования

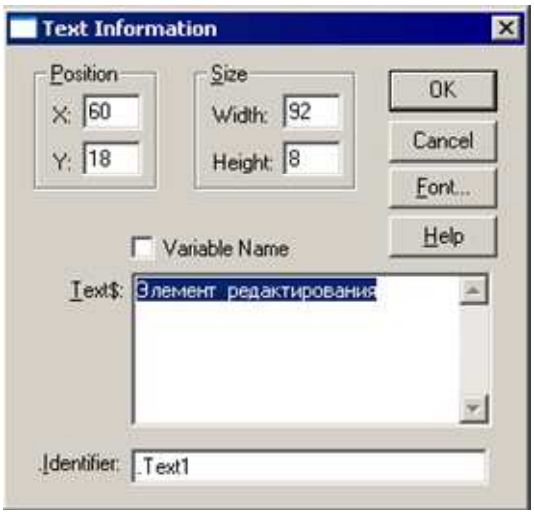

Рис. 4.93. Конфигурирование статического текста

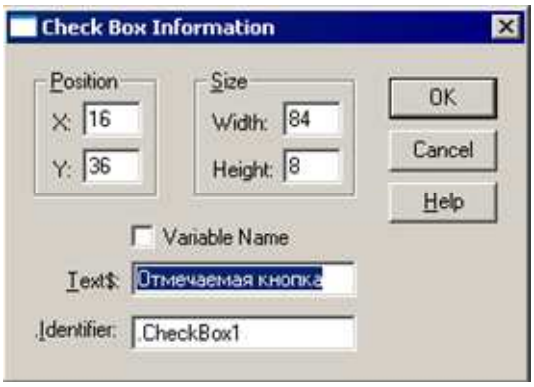

Рис. 4.94. Конфигурирование отмечаемой кнопки

Рис. 4.95. Конфигурирование раскрывающегося списка

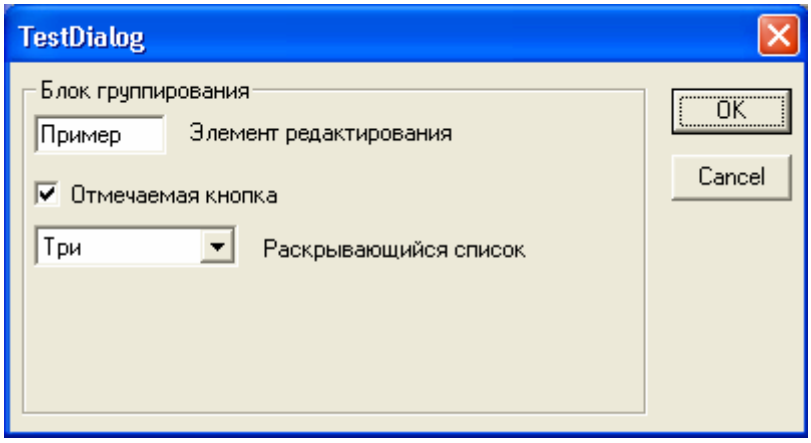

Рис. 4.96. Вид окна диалога

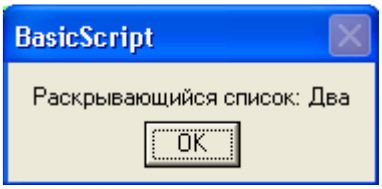

Рис. 4.97. Информация, полученная из раскрывающегося списка

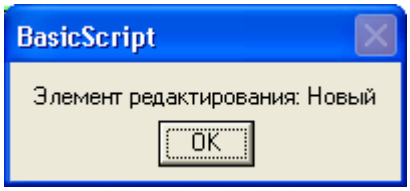

Рис. 4.98. Информация, полученная из элемента редактирования

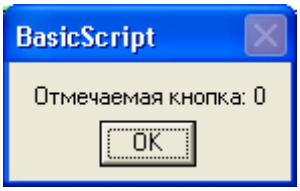

Рис. 4.99. Информация, полученная из отмечаемой кнопки

4.7. Занятие 6 "Использование функционального блока Бейсик-сценария"

Цель занятия состоит в формировании у пользователя представления о правилах использования функционального блока *Бейсик-сценария* (*BASIC Script*) редактора задач. Используются функциональные блок *тег* (**Tag**), блок *аналогового ввода* (**Analog Input**), блок *Бейсик-сценария* (**BASIC Script**) редактора задач и инструменты *инкрементный регулятор* (**Numeric Control**), индикатор (**Indicator**) и *цифровой индикатор* (**Numeric String**) редактора форм отображения. Блок *Бейсик-сценария* используется на занятии следующим образом. С его помощью получается значение аналогового сигнала канала 0 программного эмулятора сигналов **Advantech DEMO I/O**. Полученное значение сравнивается с величиной, вводимой пользователем с помощью *инкрементного регулятора*. Если значение сигнала на канале 0 превышает вводимое пользователем, то цвет *индикатора* меняется с зеленого на красный. В ином случае — цвет изменяется на зеленый.

4.7.1. Используемый инструментарий

Новым инструментом на данном занятии является *функциональный блок Бейсик-сценария (BASIC Script)* редактора задач. Блок разработан для обеспечения максимальной гибкости программирования в GeniDAQ, позволяющей реализовывать эффективные вычисления, логические операции, условные переходы, ветвления и циклы. Как указывалось ранее, в блоке используется язык программирования, синтаксис и функции которого совместимы с принятыми в Microsoft Visual Basic и Microsoft Visual Basic for Applications (VBA). Однако язык программирования Бейсик-сценариев является интерпретатором, что вызвано необходимостью достижения компромисса между простотой в применении и быстродействием. Блок Бейсик-сценария предназначен для реализации небольших алгоритмов обработки данных. Разработка и использование больших и сложных программ не рекомендуется. Выходы блока (их всего 8) блока могут соединяться с неограниченным количеством других функциональных блоков стратегии. Имеется возможность создания программ, выполняющих обработку значений на входах блока Бейсик-сценария и вывод вычисленных значений на его выходы, а также позволяющих пропускать циклы стратегии, при которых не происходит вывод значений переменных другим функциональным блокам. В последнем случае, все блоки стратегии, присоединенные к выходам блока с подобным алгоритмом работы, исключаются из обработки в пропускаемых циклах. После двойного щелчка левой

кнопкой мыши на пиктограмме блока Бейсик-сценария появляется окно редактора Бейсик-сценариев (рис. 4.77), подробно описанного в предыдущем разделе. С помощью данного редактора имеется возможность отладки, трассировки и редактирования программного кода сценариев. Таким образом, редактор сценариев представляет мощное средство разработки процедур обработки данных, специфичных для задачи пользователя. Окно редактора сценариев является модальным, что требует его закрытия перед продолжением работы с другими редакторами GeniDAQ. Блок Бейсик-сценария имеет доступ к данным, расположенным в центре обработки данных GeniDAQ. Таким образом, входные связи, присоединяемые к блоку, не несут никакой смысловой нагрузки и игнорируются.

4.7.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.101.
2. Выполните *второе действие* из занятия 1. В нашем примере также задан период запуска приложения 100 миллисекунд.
3. *Третье действие*. Добавьте в окно отображения **DISP1** цифровой индикатор, индикатор, инкрементный регулятор и две текстовых строки (рис. 4.102).
4. *Четвертое действие*. Выдвиньте на передний план окно редактора задач **TASK1** и разместите в нем блок аналогового входа **AI1**, блок тег и блок Бейсик-сценария. Выполните настройку блока аналогового ввода на прием измерительной информации от канала 0 программного эмулятора **Advantech DEMO I/O**. С помощью проводника установите связь между блоком аналогового ввода **AI1** и блоком Бэйсик-сценария **SCR1**. Окно редактора задач приобретет вид, показанный на рис. 4.103. Перед соединением функциональных блоков тег и **SCR1** настройте тег в соответствии с рис. 4.104 и соедините блоки с помощью проводника. При этом окно редактора задач приобретет окончательный вид, показанный на рис. 4.105. Сконфигурируйте блок Бейсик-сценария в соответствии с рис. 4.100.
5. *Пятое действие*. Перейдите в окно редактора форм отображения и настройте цифровой индикатор, индикатор и инкрементный регулятор в соответствии с рис. 4.106-4.108.

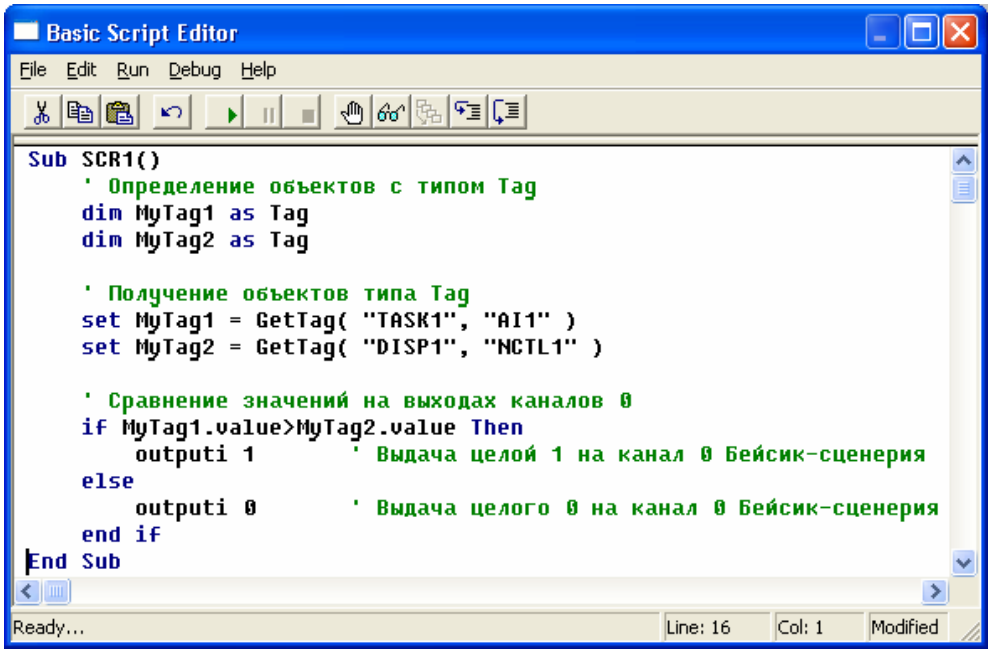

Рис. 4.100. Конфигурирование функционального блока **Бейсик-сценария**

6. *Шестое действие.* Сохраните созданный проект (стратегию). Для этого выполните команду **File | Save As**, в появившемся диалоговом окне укажите необходимую информацию и нажмите кнопку **Сохранить**. Это нужно выполнить для обоих файлов проекта с расширениями **.gni** и **.HLD**. После сохранения файлов созданного проекта запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.109). В процессе работы приложения обратите внимание на состояние индикатора и проанализируйте, почему индикатор ведет себя таким образом. Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
7. *Седьмое действие. Девятое действие.* Завершите работу приложения. Для этого выполните команду **File | Exit**.

4.7.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.18. Повтор создания и тестирования приложения "Занятие 6"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 6**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

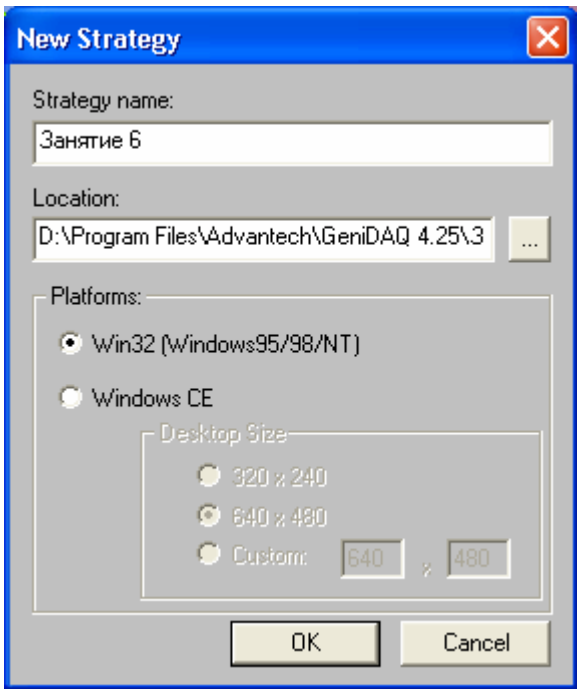

Рис. 4.101. Диалоговое окно конфигурирования приложения

Упражнение 4.19

Разработайте стратегию, обладающую следующими возможностями. В окно редактора задач поместите функциональный блок аналогового ввода, использующий эмулятор пилообразного сигнала, и функциональный блок Бейсик-сценария. В окно редактора форм отображения поместите два цифровых индикатора. Настройте первый из цифровых индикатора таким образом, чтобы он отображал сигнал на выходе блока аналогового ввода. Запрограммируйте блок Бейсик-сценария так, чтобы настройка обеспечила отображение на втором индикаторе максимального значения на выходе блока аналогового ввода.

Упражнение 4.20

Разработайте стратегию, обладающую следующими возможностями. В окно редактора задач поместите функциональный блок аналогового ввода, использующий эмулятор гармонического сигнала, и функциональный блок Бейсик-сценария.

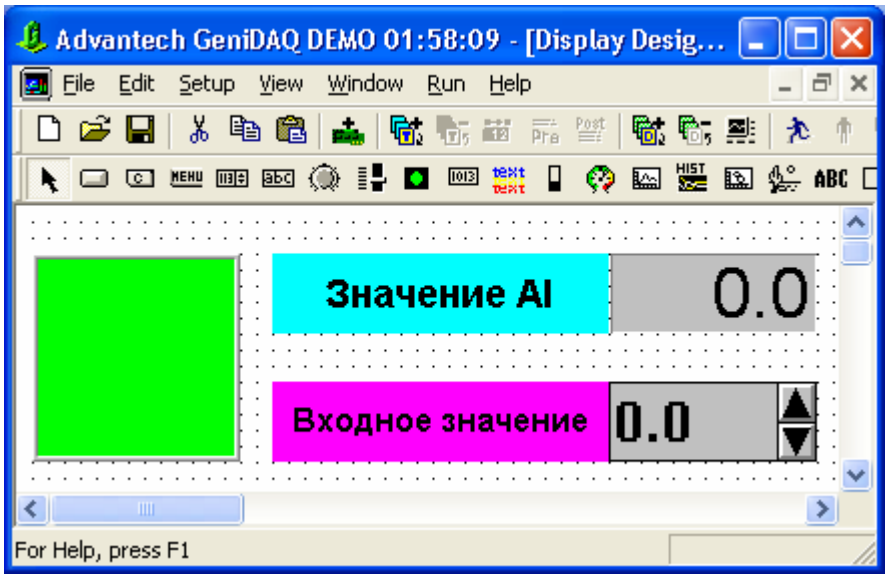

Рис. 4.102. Вид окна редактора форм отображения после добавления управляющих элементов

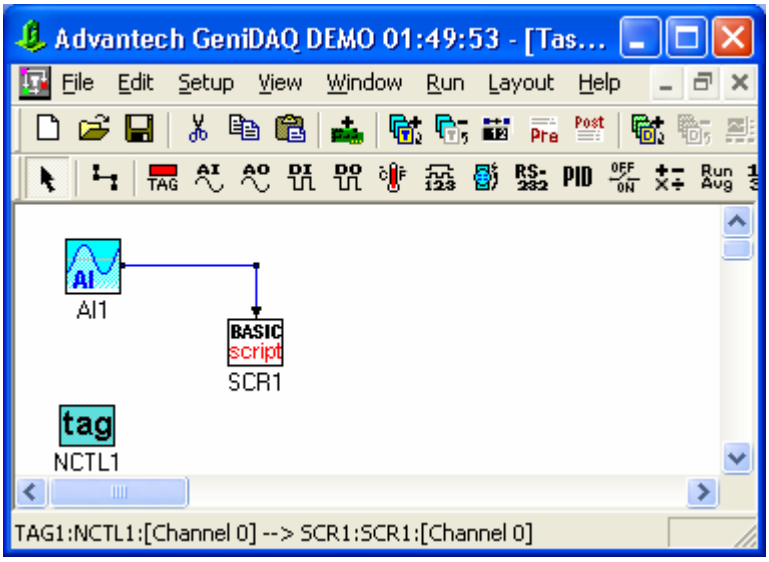

Рис. 4.103. Вид окна редактора задач

В окно редактора форм отображения поместите два цифровых индикатора. Настройте первый из цифровых индикатора таким образом, чтобы он отображал сигнал на выходе блока аналогового ввода. Запрограммируйте блок Бейсик-сценария

так, чтобы настройка обеспечила отображение на втором индикаторе минимального значения на выходе блока аналогового ввода.

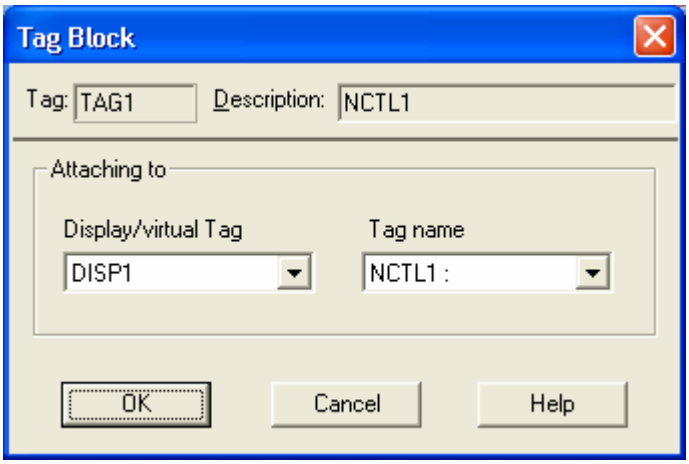

Рис. 4.104. Конфигурирование блока *тег*

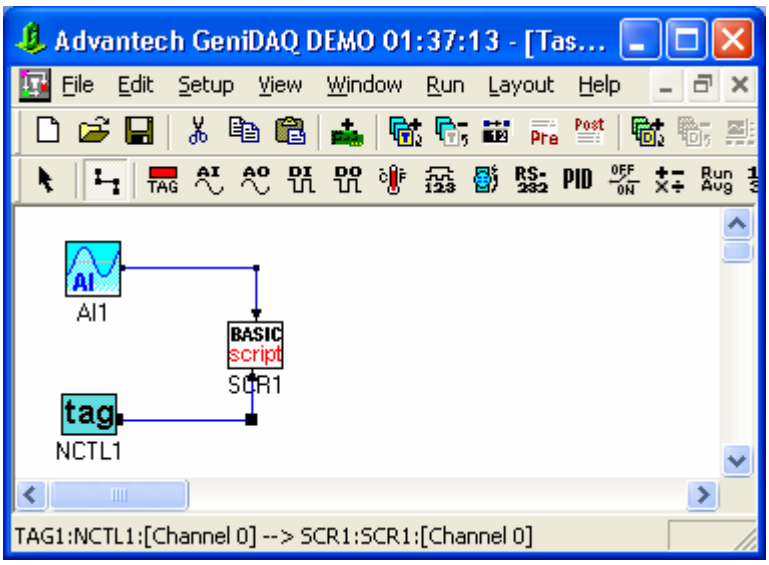

Рис. 4.105. Окончательный вид окна редактора задач

Упражнение 4.21

Разработайте стратегию, обладающую следующими возможностями. В окно редактора задач поместите функциональный блок аналогового ввода, использующий эмулятор случайного сигнала, и функциональный блок Бейсик-сценария. В окно редактора форм отображения поместите два цифровых индикатора. Настройте

первый из цифровых индикатора таким образом, чтобы он отображал сигнал на выходе блока аналогового ввода. Запрограммируйте блок Бейсик-сценария так, чтобы настройка обеспечила отображение на втором индикаторе максимального значения на выходе блока аналогового ввода.

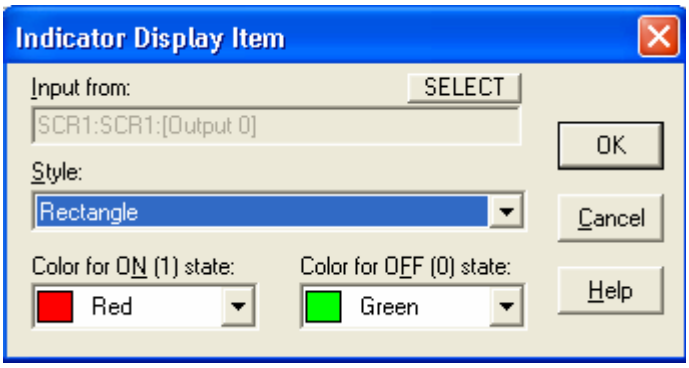

Рис. 4.106. Конфигурирование индикатора

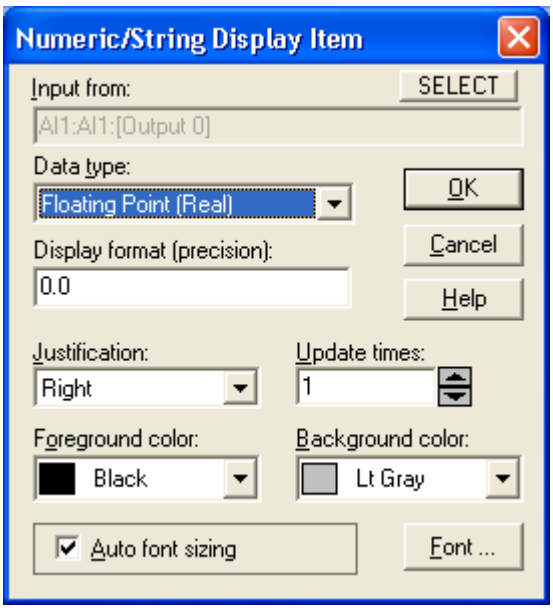

Рис. 4.107. Конфигурирование цифрового индикатора

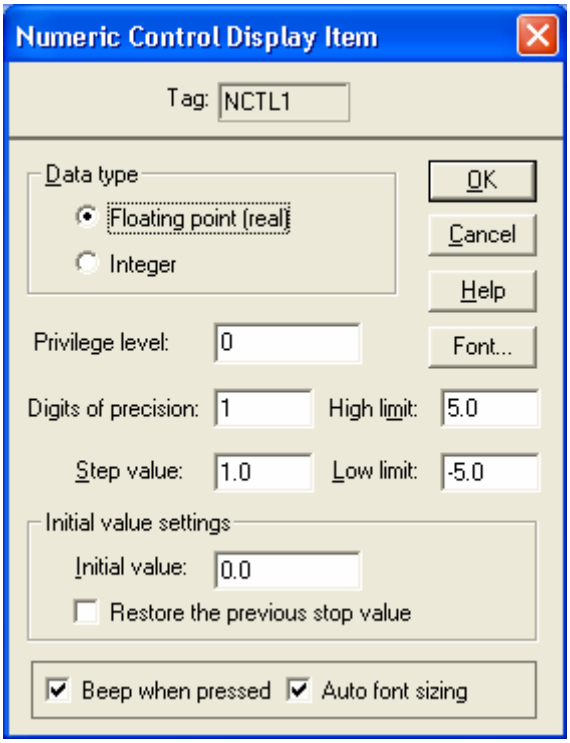

Рис. 4.108. Конфигурирование инкрементного регулятора

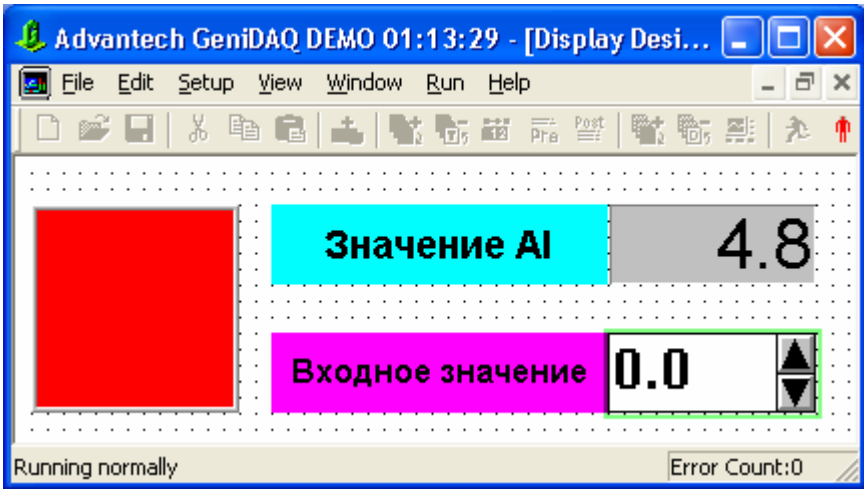

Рис. 4.109. Демонстрация работы проекта в окне отображения

4.8. Занятие 7 "Совместное использование функциональных блоков Бейсик-сценария и виртуального тега"

Цель занятия состоит в изучении правил совместного применения функциональных блоков Бейсик-сценария (**BASIC Script**) и *виртуального тега* (**Virtual Tag**). Используются функциональный блок тег (**Tag**), который связывается с виртуальным тегом, блок аналогового ввода (**Analog Input**), блок **Бейсик-сценария** редактора задач и инструменты индикатор (**Indicator**), два цифровых индикатора (**Numeric String**) и две текстовых строки (**Text String**) редактора форм отображения. Блок Бейсик-сценария используется на занятии следующим образом. С его помощью получается значение аналогового сигнала канала 0 программного эмулятора сигналов **Advantech DEMO I/O**. Полученное значение сравнивается с величиной, хранимой в виртуальном теге. Если значение сигнала на канале 0 превышает хранимое в виртуальном теге, то это значение запоминается в виртуальном теге и на выходе блока Бейсик-сценарий формируется единичный сигнал. Иначе — значение, хранимое в виртуальном теге, не изменяется, а на выходе блока Бейсик-сценарий формируется нулевой сигнал. Цвет индикатора — красный, пока значение сигнала на канале 0 превышает хранимое в виртуальном теге. В противоположном случае — цвет изменяется на зеленый.

4.8.1. Используемый инструментарий

Новым инструментом на данном занятии является *функциональный блок виртуальный тег* (*Virtual Tag*). Виртуальный тег, созданный в редакторе задач, сохраняется в центре обработки данных так же, как и остальные встроенные функциональные блоки. Виртуальные теги доступны всем задачам, что обеспечивает возможность их использования для обмена данными между несколькими задачами.

Для создания виртуального тега выполните команду **Setup | Add/Delete Virtual Tags** и на экран будет выведено диалоговое окно, содержащее таблицу виртуальных тегов (рис. 4.110). Для создания виртуального тега и добавления его в таблицу следует нажать кнопку **Add Tag**, в появившемся окне диалога (рис. 4.111) ввести его имя, тип, начальное значение и нажать кнопку **OK**. При этом виртуальный тег с введенным именем будет создан и добавлен в таблицу виртуальных тегов (см. приведенный ранее рис. 4.110). Для изменения параметров виртуального тега надо выбрать его в списке тегов (см. приведенный ранее рис. 4.110), нажать кнопку **Update**, в появившемся окне диалоговом окне (см. приведенный ранее рис. 4.111) задать требуемые параметры и нажать кнопку **OK**. Для удаления виртуального тега следует выбрать его имя в таблице виртуальных тегов и нажать кнопку **Delete**. После модификации таблицы виртуальных тегов нажатием кнопки **Close** можно закрыть диалоговое окно, показанное на рис. 4.110.

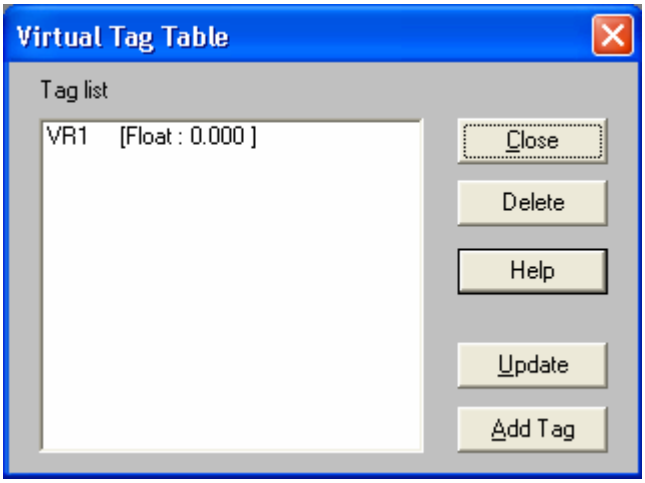

Рис. 4.110. Диалоговое окно для модификации таблицы виртуальных тегов

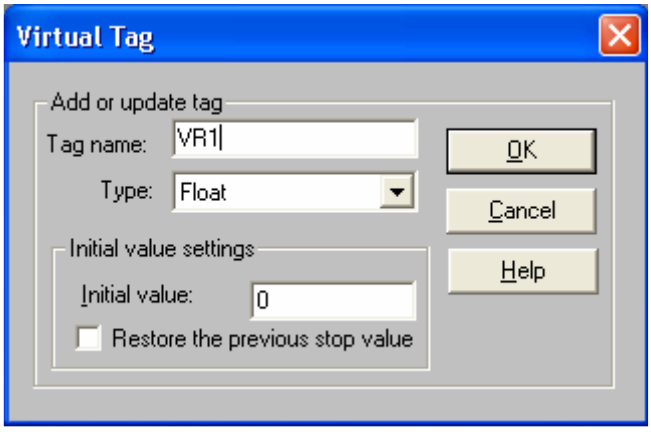

Рис. 4.111. Задание параметров добавляемого виртуального тега

После создания, виртуальные теги не помещаются в набор инструментов редактора задач. Доступ к виртуальным тегам в редакторе задач может быть получен через блок тег. Для этого следует в окне конфигурирования тега (рис. 4.112) в списке **Display/virtual Tag** выбрать значение **VIRTASK**, а в списке **Tag name** выбрать имя виртуального тега (например, **VR1**), к которому требуется получить доступ. После нажатия кнопки **OK**, блок тег может быть соединен с другими функциональными блоками стратегии, имеющими входы и выходы. Таким образом, виртуальный тег может быть как источником, так и приемником данных. Например, ему могут передаваться значения от блока аналогового ввода или он может передавать данные блоку дискретного вывода.

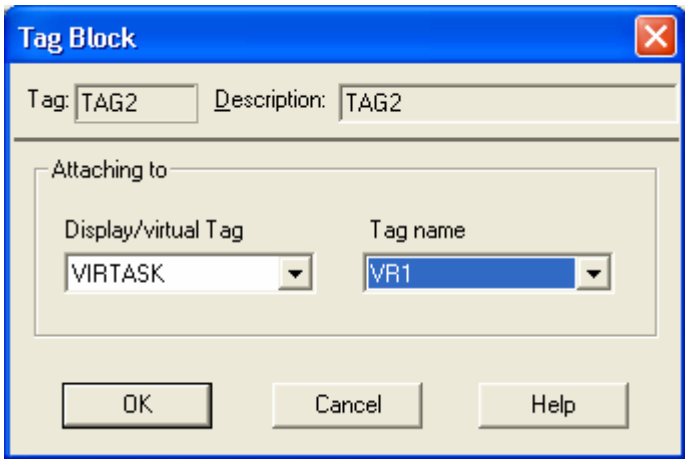

Рис. 4.112. Диалоговое окно для настройки доступа к виртуальному тегу

Значение, связанное с виртуальным тегом, может быть отображено на экране монитора путем установления связи с соответствующим элементом отображения (рис. 4.113).

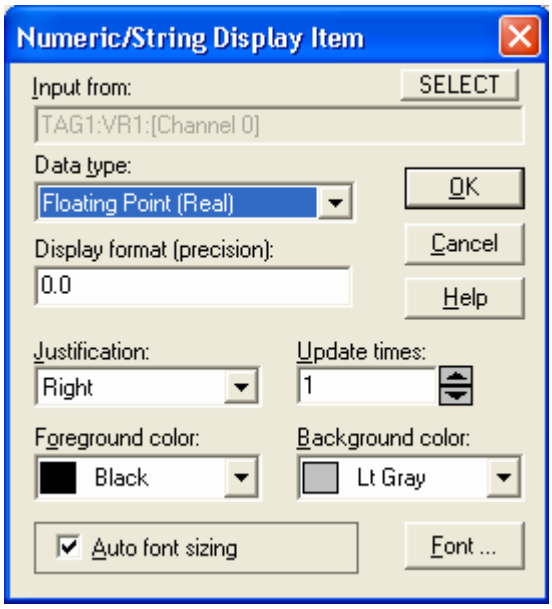

Рис. 4.113. Связь виртуального тега с элементом отображения редактора форм отображения

В Бейсик-сценарии виртуальный тег является тем же объектом, что и другие функциональные блоки стратегии. Имеется возможность получить и установить значение, связанное с виртуальным тегом, путем использования функций `GetTag( )`

и `SetTag( )`. Сказанное иллюстрирует рис. 4.114. Отличие между виртуальным тегом и другими функциональными блоками редактора задач состоит в том, что после установки новых значений предыдущие значения, связанные с блоками редактора задач, не могут быть вновь считаны задачами стратегии из центра обработки данных. Таким образом, использование виртуальных тегов дает возможность передачи данных из Бейсик-сценариев и других приложений задачам и экранным формам текущей стратегии.

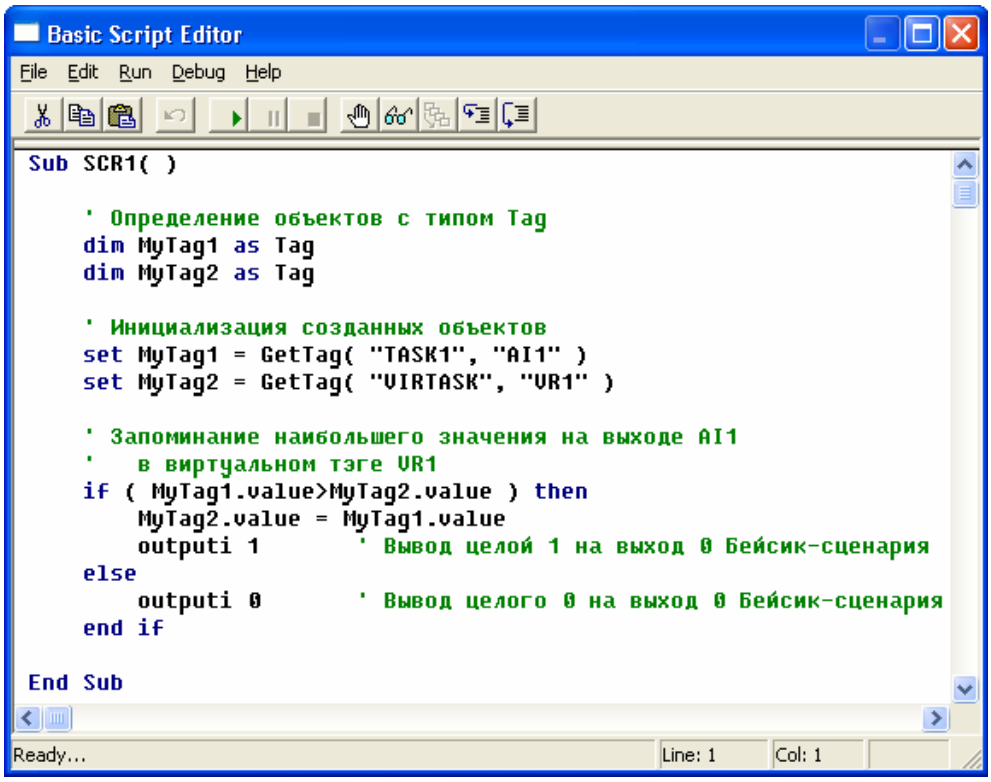

Рис. 4.114. Использование виртуального тега в редакторе сценариев

4.8.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.115.
2. Выполните *второе действие* из занятия 1. В нашем примере задан период запуска приложения 1 секунда.

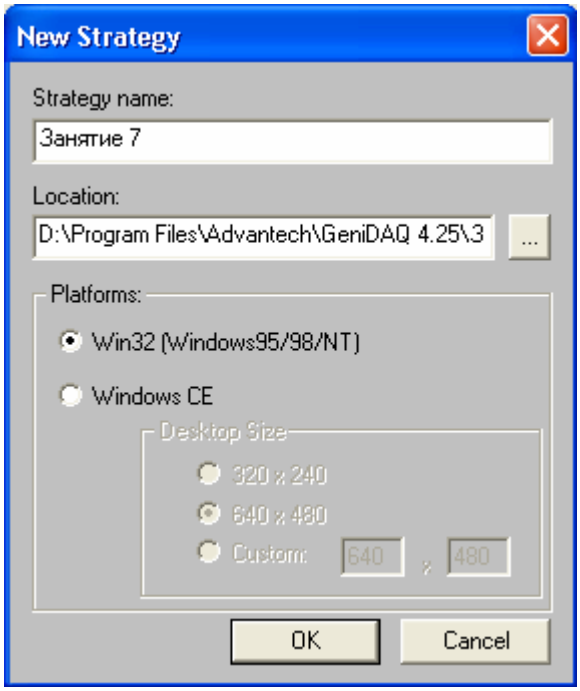

Рис. 4.115. Диалоговое окно конфигурирования приложения

3. *Третье действие*. Сделайте активным окно редактора задач и создайте объект виртуальный тег. Для этого выполните команду **Setup | Add/Delete Virtual Tags** и в появившемся диалоговом окне (см. рис. 4.110) нажмите кнопку **Add Tag**. Появится следующее диалоговое окно и с его помощью сконфигурируйте виртуальный тег в соответствии с рис. 4.111.
4. *Четвертое действие*. В окно редактора задач поместите функциональные блоки аналогового ввода, Бейсик-сценария и тег. Произведите двойной щелчок левой кнопкой мыши, поместив курсор на изображение блока аналогового ввода, и выполните его настройку на ввод информации от канала 0 программного эмулятора **Advantech DEMO I/O** в соответствии с рис. 4.116. Установите связь между функциональным блоком тег и созданным объектом виртуальный тег в соответствии с описанной ранее схемой и рис. 4.112. Произведите двойной щелчок левой кнопкой мыши, поместив курсор на пиктограмму блока Бейсик-сценарий и введите исходный текст программы в соответствии с рис. 4.114. Хотя это и не является обязательным, но, для наглядности, соедините блоки аналогового ввода и тег с функциональным блоком Бейсик-сценария. После этого окно редактора задач приобретет вид, показанный на рис. 4.117.
5. *Пятое действие*. Активизируйте окно редактора форм отображения. Добавьте в это окно элементы отображения индикатор, два цифровых индикатора и две текстовых строки (рис. 4.118). Сконфигурируйте индикатор в соответствии с

рис. 4.119. Сконфигурируйте *цифровые индикаторы* в соответствии с рис. 4.113, рис. 4.120 и текстовые строки в соответствии с рис. 4.118.

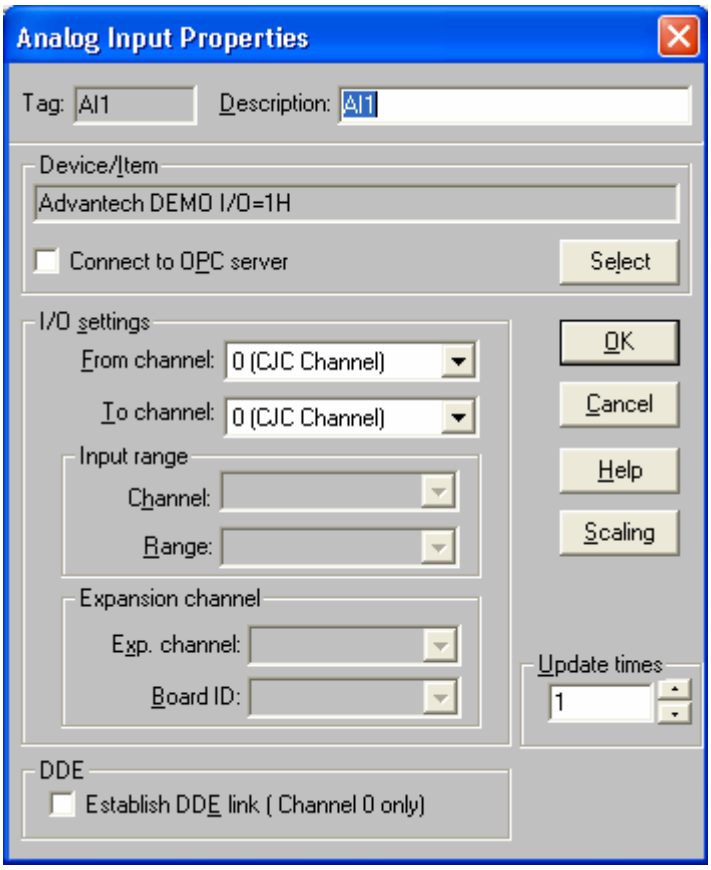

Рис. 4.116. Конфигурирование блока аналогового ввода

6. *Шестое действие*. Сохраните созданный проект (стратегию). Для этого выполните команду **File | Save As**, в появившемся окне диалога укажите необходимую информацию и нажмите кнопку **Сохранить**. Это нужно выполнить для обоих файлов проекта с расширениями `.gni` и `.HLD`.
7. *Седьмое действие*. После сохранения файлов созданного проекта запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.121). В процессе работы приложения обратите внимание на состояние индикатора. Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
8. *Восьмое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

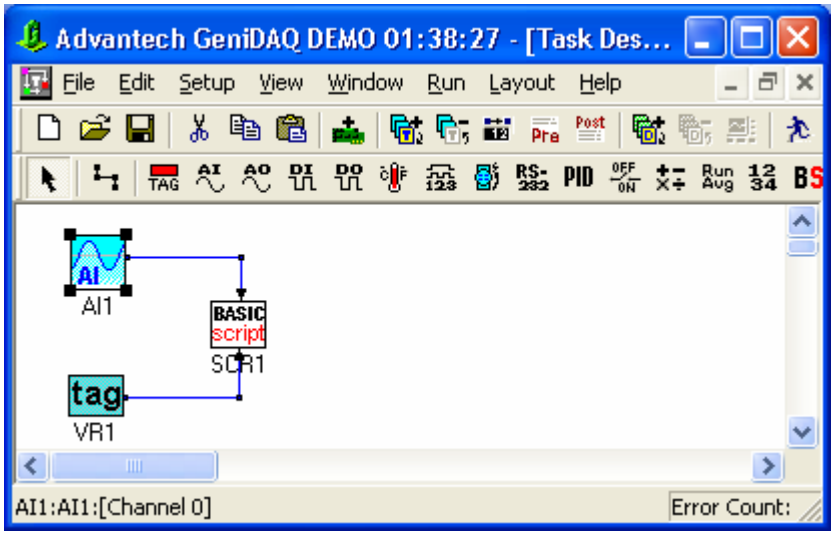

Рис. 4.117. Окно редактора задач

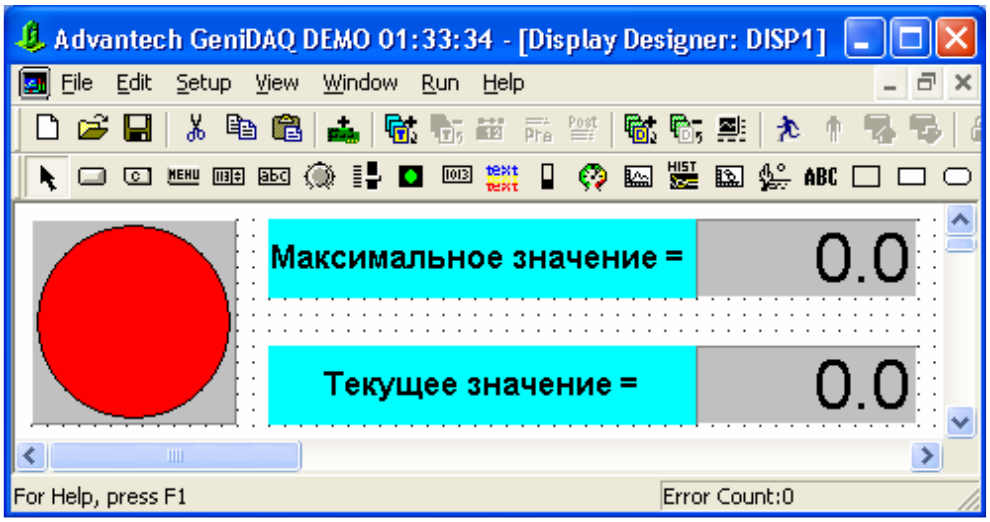

Рис. 4.118. Окно редактора форм отображения

4.8.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

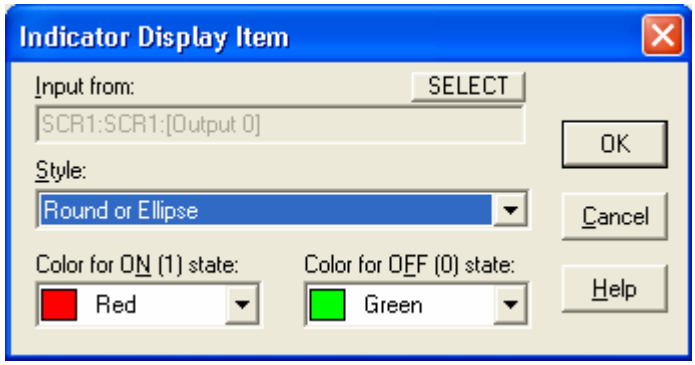

Рис. 4.119. Конфигурирование индикатора

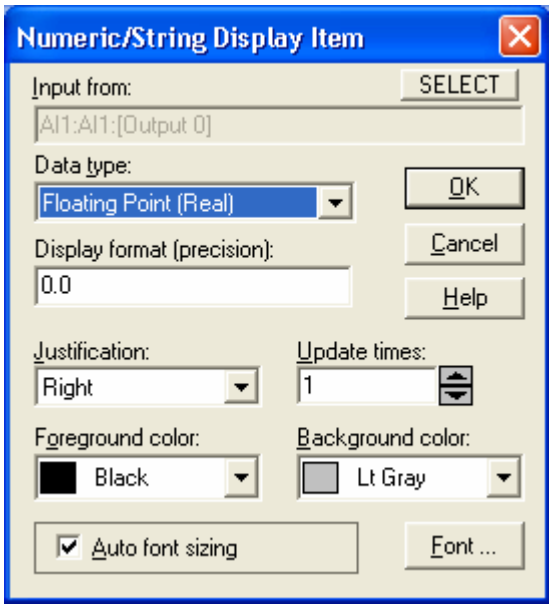

Рис. 4.120. Конфигурирование цифрового индикатора

Упражнение 4.22. Повтор создания и тестирования приложения "Занятие 7"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 7**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Упражнение 4.23

Разработайте стратегию, обладающую следующими возможностями (в качестве аналога используйте стратегию из занятия 7).

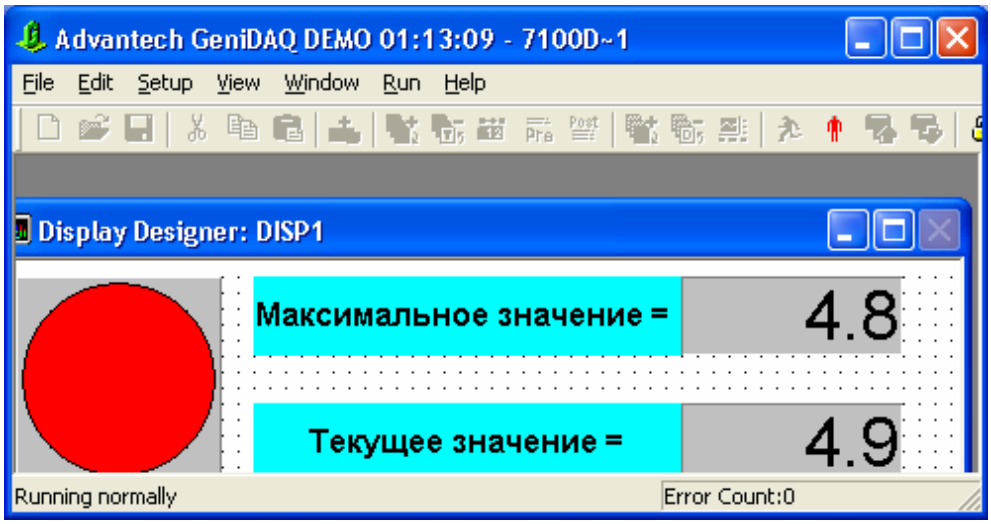

Рис. 4.121. Демонстрация работы проекта в окне отображения

В окно редактора задач поместите функциональный блок аналогового ввода, использующий эмулятор пилообразного сигнала, функциональный блок Бейсик-сценария и два функциональных блока тег, каждый из которых связывается с соответствующим виртуальным тегом. В окно редактора форм отображения поместите три цифровых индикатора. Настройте первый из цифровых индикаторов таким образом, чтобы он отображал сигнал на выходе блока аналогового ввода. Запрограммируйте блок Бейсик-сценария и настройте оставшиеся цифровые индикаторы так, чтобы они работали по следующему алгоритму. Второй цифровой индикатор должен отображать максимальное значение сигнала на выходе блока аналогового ввода, а третий — минимальное значение сигнала на выходе блока аналогового ввода. Все три индикатора в окне отображения снабдить поясняющим текстом.

Упражнение 4.24

Разработайте стратегию, обладающую следующими возможностями (в качестве аналога используйте стратегию из занятия 7). Используйте два окна редактора задач. Придумайте стратегию, в которой происходит передача информации между задачами с использованием виртуальных тегов. Сконфигурируйте окно отображения для подтверждения указанной передачи.

4.9. Занятие 8 "Программирование основного сценария"

Цель занятия состоит в изучении базовых правил разработки *основного сценария* и способов его применения для управления задачами. С этой целью в рамках стратегии

создается задача и основной сценарий для управления этой задачей. Используются блок аналогового ввода (**Analog Input**) редактора задач, инструменты график времени (**Realtime Trend Graph**), цифровой индикатор (**Numeric String**) и текстовые строки (**Text String**) редактора форм отображения, а также *основной сценарий* стратегии.

4.9.1. Используемый инструментарий

Новым инструментом на данном занятии является *основной сценарий*. Особенности его использования описаны в разд. 4.6 "Использование языка VBA в SCADA-системах".

4.9.2. Проектирование приложения

Проектирование приложения. Для реализации поставленного задания выполните следующие действия.

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.122.
2. *Второе действие*. Сделайте активным окно редактора задач и поместите в него функциональный блок аналогового ввода. Произведите двойной щелчок левой кнопкой мыши, поместив курсор на изображение блока *аналогового ввода* в окне редактора задач, и выполните его настройку на ввод информации от канала 0 программного эмулятора **Advantech DEMO I/O** в соответствии с приведенным ранее рис. 4.116.
3. *Третье действие*. Активизируйте окно редактора форм отображения. Добавьте в это окно элементы отображения график времени, цифровой индикатор и текстовую строку (рис. 4.123). Сконфигурируйте *график времени* и *цифровой индикатор* в соответствии с рис. 4.124 и 4.125.
4. *Четвертое действие*. Задайте временные параметры запуска задачи **TASK1** таким образом, чтобы ее запуск осуществлялся под управлением основного сценария. Для этого выполните команду **Setup | Task Properties...** и настройте параметры запуска задачи в соответствии с рис. 4.126. При этом будет задан пассивный запуск задачи по команде из основного сценария.
5. *Пятое действие*. Добавьте в стратегию основной сценарий. Для этого выполните команду **File | Add/Delete | Add Main Script** и в появившемся окне редактора сценариев введите текст, показанный на рис. 4.127.

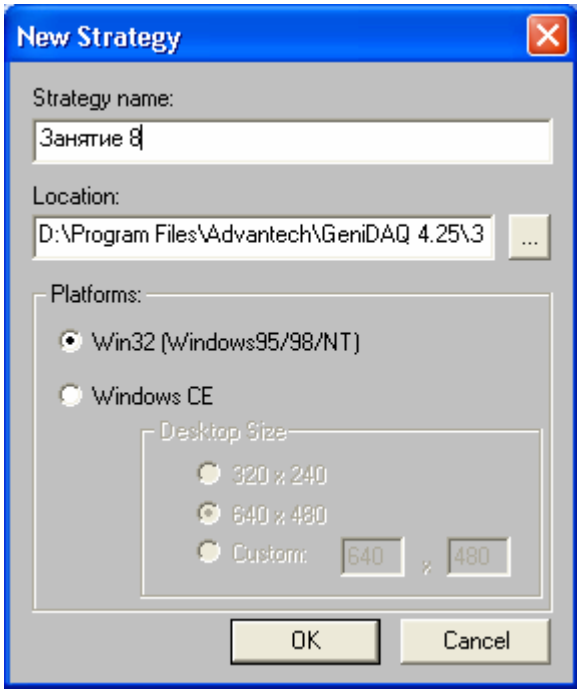

Рис. 4.122. Диалоговое окно конфигурирования приложения

6. *Пятое действие*. Сохраните созданный проект (стратегию). Для этого выполните команду **File | Save As**, в появившемся окне диалога укажите необходимую информацию и нажмите кнопку **Сохранить**. Это нужно выполнить для обоих файлов проекта с расширениями `.gni` и `.HLD`.
7. *Седьмое действие*. После сохранения файлов созданного проекта запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.128). Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
8. *Восьмое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

4.9.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

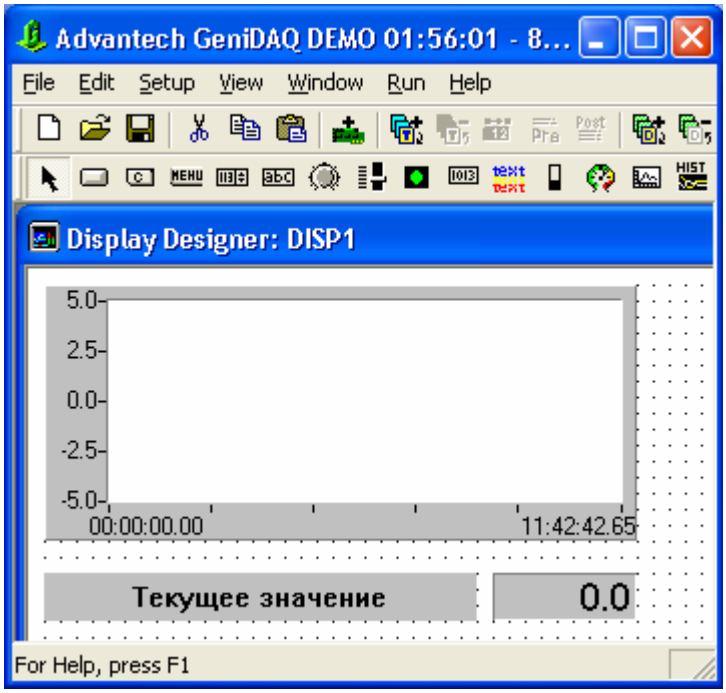

Рис. 4.123. Окно отображения

Упражнение 4.25. Повтор создания и тестирования приложения "Занятие 8"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 8**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

4.10. Занятие 9 "Управление несколькими задачами"

Цель занятия состоит в изучении способа *управления несколькими задачами* в рамках одной стратегии с помощью настройки параметров выполнения каждой задачи. С этой целью создаются две отдельные задачи, каждая из которых содержит блок аналогового ввода. Создается окно отображения, содержащее два элемента отображения временной график, на которые выводятся сигналы с выходов блоков аналогового ввода. Параметры каждой задачи настраиваются таким образом, чтобы они имели разные режимы выполнения.

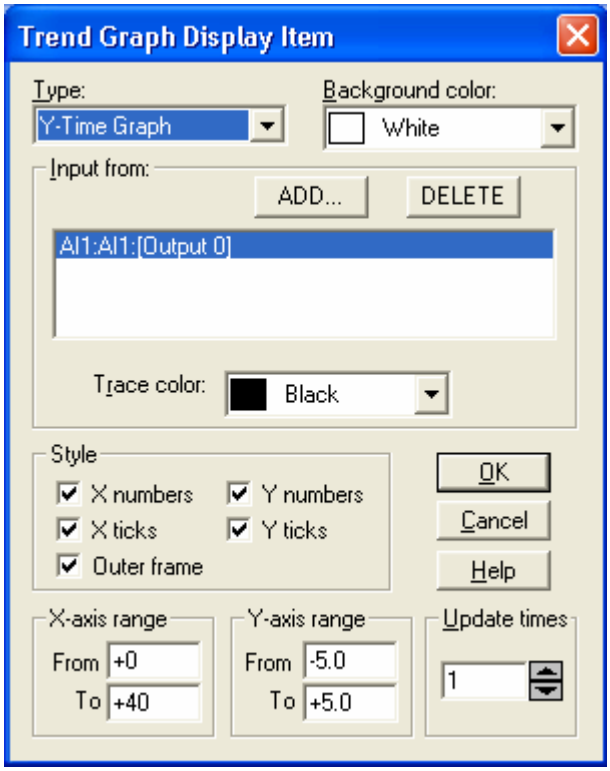

Рис. 4.124. Конфигурирование графика времени

4.10.1. Используемый инструментарий

Новым инструментом на данном занятии является использование диалогового окна **Scan Task Setup**, с помощью которого задаются параметры запуска задачи (рис. 4.129) — интервал запуска, способ запуска и продолжительность выполнения.

4.10.2. Проектирование приложения

1. Выполните *первое действие* из занятия 1. Создаваемое приложение настройте в соответствии с рис. 4.130.
2. *Второе действие*. Сделайте активным окно **TASK1** редактора задач и поместите в него функциональный блок аналогового ввода. Произведите двойной щелчок левой кнопкой мыши, поместив курсор на изображение блока аналогового ввода в окне редактора задач, и выполните его настройку на ввод информации от канала 0 программного эмулятора **Advantech DEMO I/O** в соответствии с приведенным ранее рис. 4.116. Добавьте в стратегию вторую задачу **TASK2**, для чего выполните команду **File | Add/Delete | Add Task**. Аналогичным образом поместите в него функциональный блок аналогового ввода. Произведите двойной

щелчок левой кнопкой мыши, поместив курсор на изображение блока аналогового ввода в окне редактора задач, и выполните его настройку на ввод информации от канала 1 программного эмулятора **Advantech DEMO I/O** в соответствии с рис. 4.131.

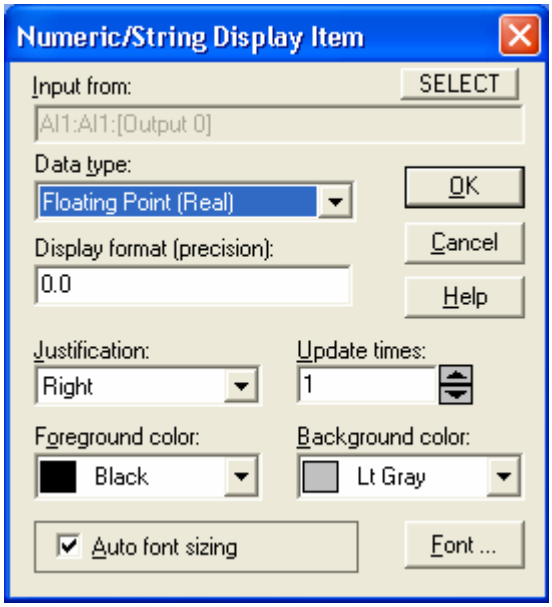

Рис. 4.125. Конфигурирование цифрового индикатора

3. *Третье действие*. Активизируйте окно редактора форм отображения. Добавьте в это окно два элемента отображения график времени и две текстовые строки (рис. 4.132). Сконфигурируйте графики времени в соответствии с рис. 4.133 и 4.134. При этом на одном из них будет отображаться информация с выхода блока аналогового ввода задачи **TASK1**, а на втором — задачи **TASK2**.
4. *Четвертое действие*. Задайте временные параметры запуска задачи **TASK1**. Сделайте активным окно задачи **TASK1**, для чего поместите курсор во внутреннюю область ее окна и произведите щелчок левой кнопкой мыши. Выполните команду **Setup | Task Properties... (Ctrl+T)** задайте параметры задачи в соответствии с рис. 4.135. Такая настройка обеспечивает запуск задачи в 16 часов 28 минут и ее выполнение в течение 15 секунд. Аналогичным образом задайте временные параметры запуска задачи **TASK2**. Сделайте активным окно задачи **TASK2**, для чего поместите курсор во внутреннюю область ее окна и произведите щелчок левой кнопкой мыши. Выполните команду **Setup | Task Properties... (Ctrl+T)**, задайте параметры задачи в соответствии с рис. 4.136. Такая настройка обеспечивает запуск задачи **TASK2** по истечении 15 секунд от момента запуска всей стратегии с ее остановкой после 10 циклов выполнения.

5. *Пятое действие*. Сохраните созданный проект (стратегию) аналогично тому, как это делалось на занятии 1. Для этого выполните команду **File | Save As**, в появившемся окне диалога укажите необходимую информацию и нажмите кнопку **Сохранить**. Это нужно выполнить для обоих файлов проекта с расширениями `.gni` и `.HLD`.

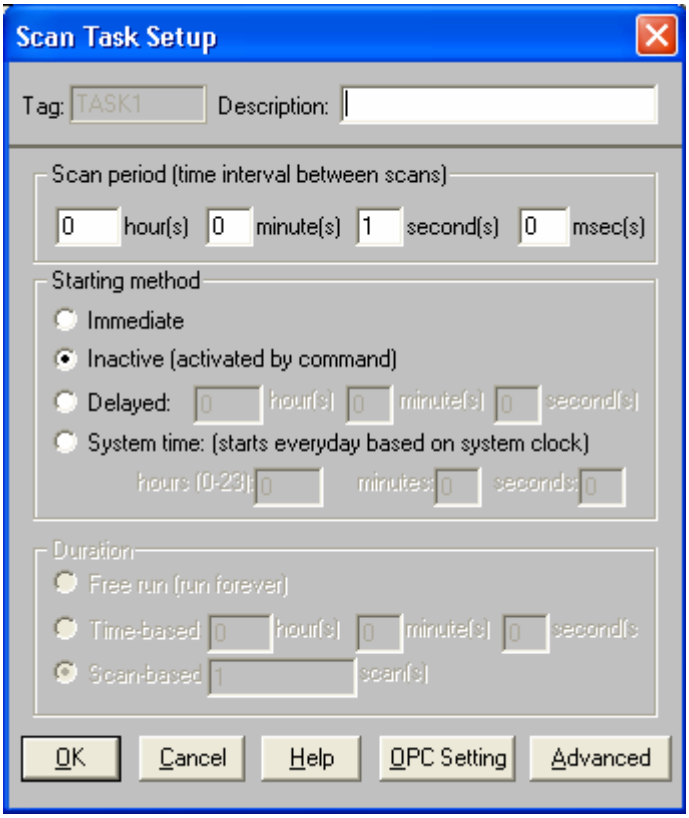

Рис. 4.126. Задание временных параметров запуска задачи

6. *Шестое действие*. После сохранения файлов созданного проекта запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.137). Обратите внимание на особенности запуска и останова задач. Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов.
7. *Седьмое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

4.10.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

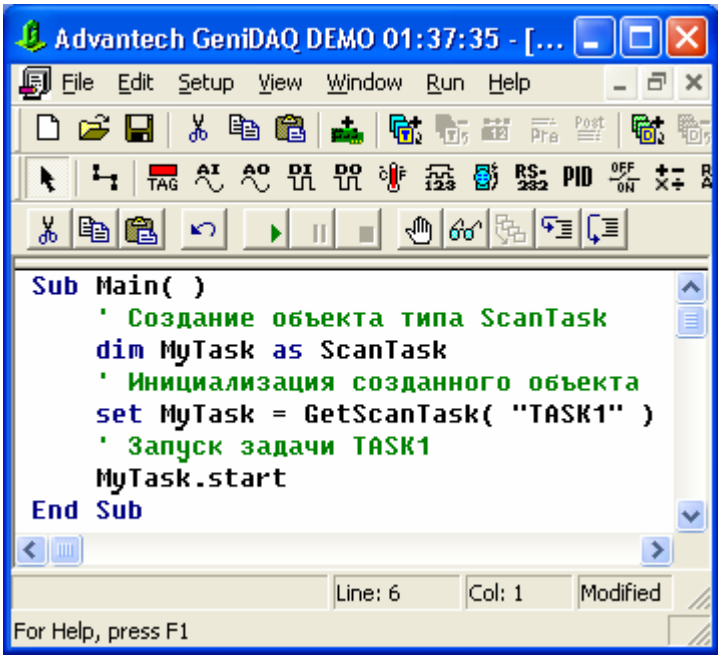

Рис. 4.127. Основной сценарий стратегии

Упражнение 4.26. Повтор создания и тестирования приложения "Занятие 9"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 9**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Упражнение 4.27

Разработайте стратегию, обладающую следующими возможностями. Создайте две задачи. В окно редактора задач каждой из задач поместите функциональный блок аналогового ввода, использующий эмулятор гармонического сигнала и прямоугольного сигнала. Настройте параметры запуска задач таким образом, чтобы период их выполнения составлял 1 секунду, обеспечьте запуск третьей задачи с задержкой 10 секунд и ее выполнение 15 раз и пассивный запуск второй задачи с помощью главного сценария. В окно редактора форм отображения поместите два

графика и настройте их таким образом, чтобы они отображали сигналы на выходах соответствующих блоков аналогового ввода.

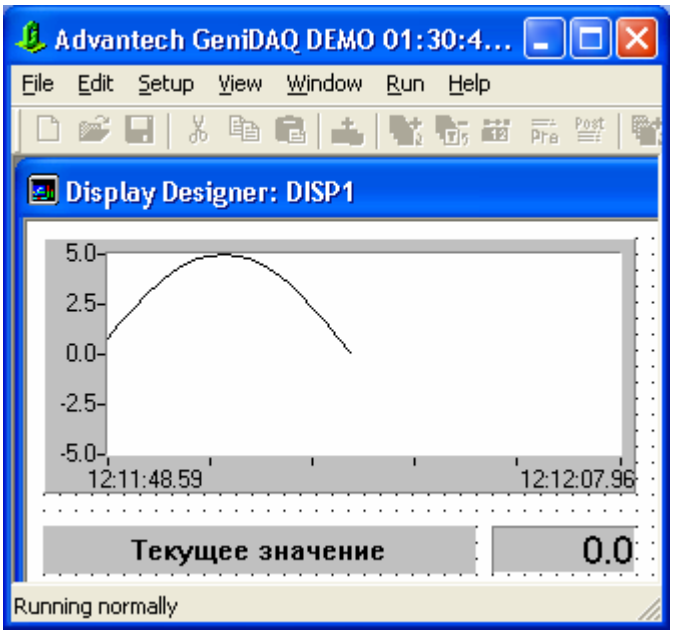

Рис. 4.128. Демонстрация работы проекта в окне отображения

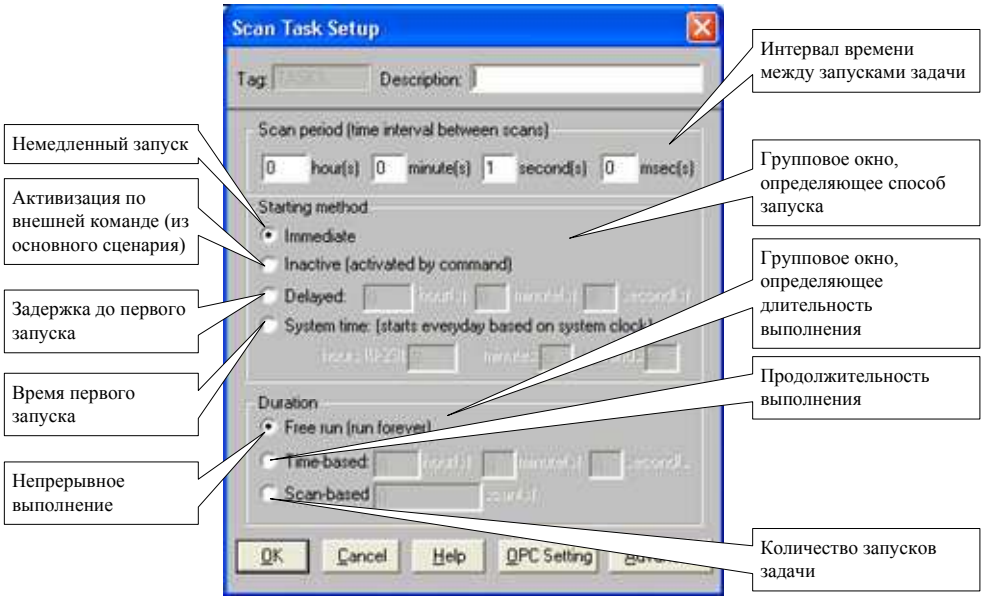

Рис. 4.129. Диалоговое окно задания параметров запуска задачи

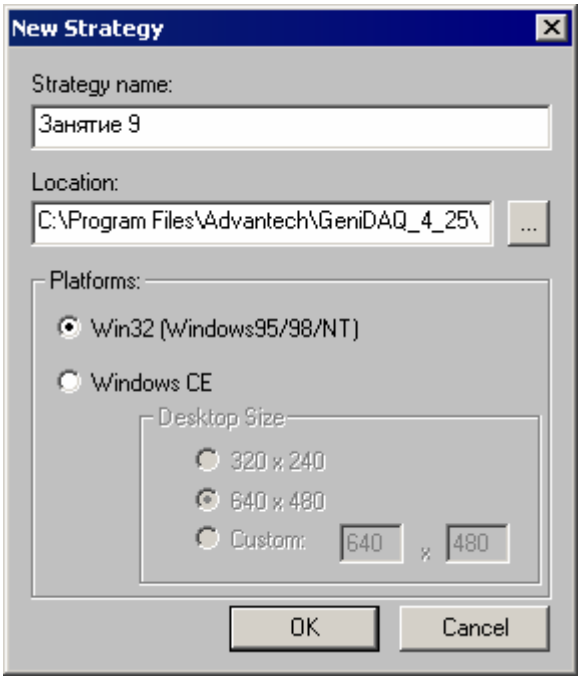

Рис. 4.130. Диалоговое окно конфигурирования приложения

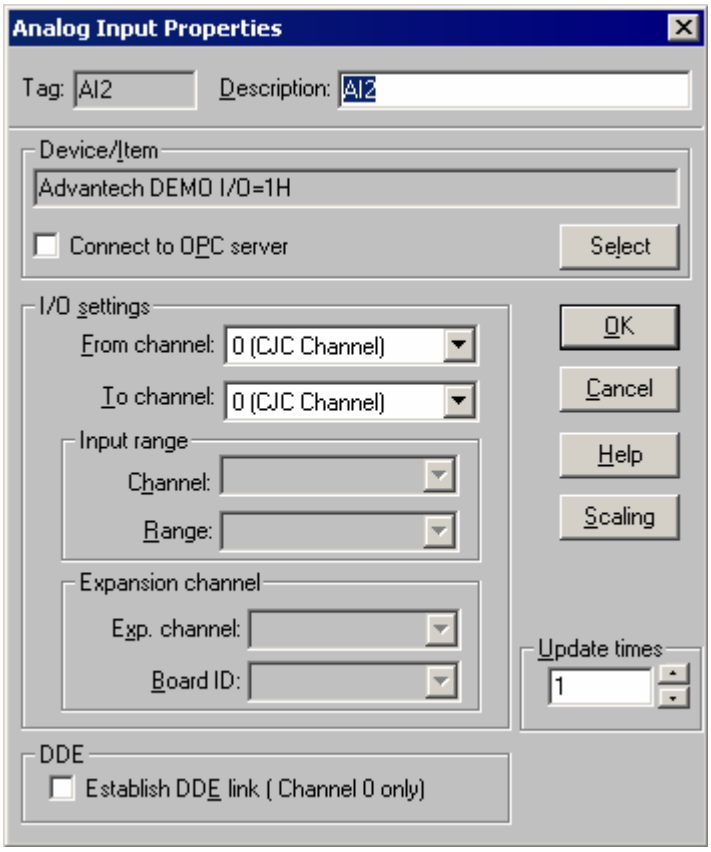

Рис. 4.131. Конфигурирование блока аналогового ввода задачи **TASK2**

Упражнение 4.28

Разработайте стратегию, обладающую следующими возможностями. Создайте две задачи. В окно редактора задач каждой из задач поместите функциональный блок аналогового ввода, использующий эмулятор гармонического сигнала и прямоугольного сигнала. Настройте параметры запуска задач таким образом, чтобы период их выполнения составлял 1 секунду, обеспечьте запуск третьей задачи с задержкой 10 секунд и ее выполнение 15 раз и запуск второй задачи в заданное время и ее выполнение в течение 30 секунд. В окно редактора форм отображения поместите два графика и настройте их таким образом, чтобы они отображали сигналы на выходах соответствующих блоков аналогового ввода.

4.11. Занятие 10 "DDE-обмен с использованием блока Бейсик-сценария редактора задач"

Цель занятия состоит в изучении способа *динамического обмена данными* между созданной стратегией (источник) и внешним приложением (электронная таблица, приемник). Динамический обмен данными (DDE-обмен) реализуется с помощью функционального блока Бейсик-сценария редактора задач, предварительного и пост сценариев задачи. С этой целью в рамках стратегии создается задача, содержащая блок аналогового ввода, виртуальный тег и блок Бейсик-сценария. В папку сценария помещается файл электронной таблицы, в который созданное приложение в процессе работы выводит результаты измерений, выполненных блоком аналогового ввода. Описания функционального блока Бейсик-сценария редактора задач, предварительного и пост сценария задачи приведены в занятии 6 и разд. 4.6. Виртуальный тег описан в занятии 7.

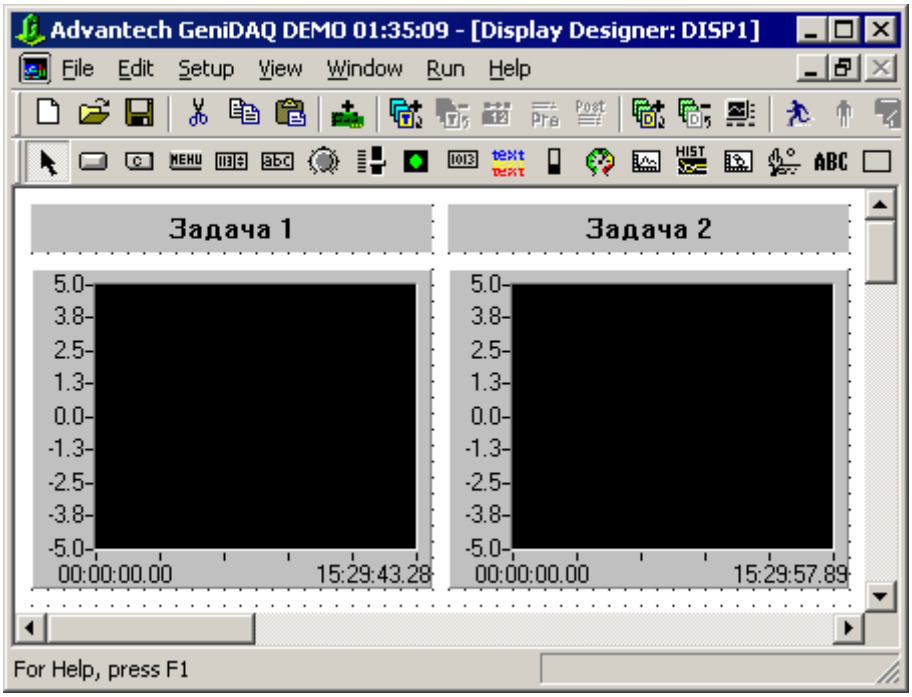

Рис. 4.132. Конфигурация окна отображения

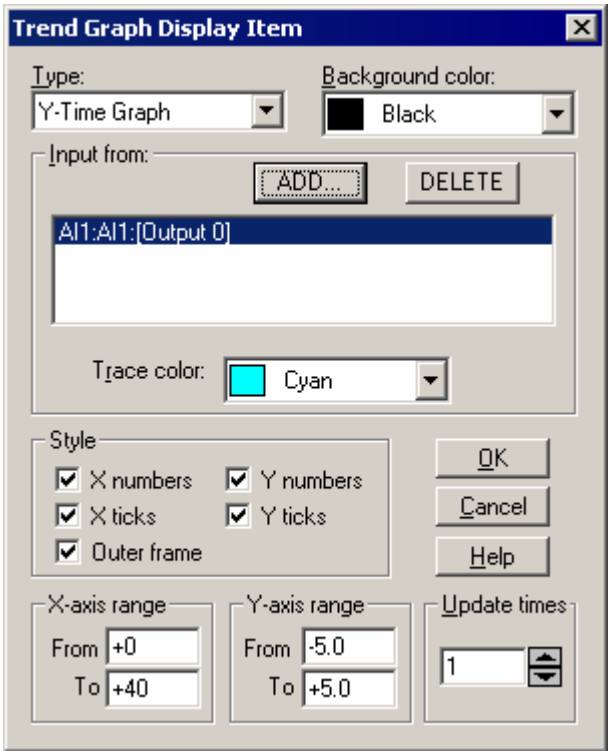

Рис. 4.133. Конфигурация графика для блока аналогового ввода задачи **TASK1**

4.11.1. Проектирование приложения

1. Выполните *первое действие* из занятия 1. Создаваемое приложение назовите **Занятие 10**.
2. *Второе действие*. Сделайте активным окно **TASK1** редактора задач и поместите в него функциональные блоки аналогового ввода, тега и Бейсик-сценария. Выполните настройку блока аналогового ввода в соответствии с приведенныи ранее рис. 4.116, добавьте виртуальный тег (команда **Setup | Add/Delete Virtual Tags | Add Tag**) и настройте его в соответствии с рис. 4.138, настройте тег и блок Бейсик-сценария в соответствии с рис. 4.139 и 4.140. Полный текст функционального блока Бейсик-сценария, снабженный подробными комментариями, вполне ясен:

```
' Вывод из стратегии (приложения) в электронную таблицу с
'   использованием DDE-технологии
Sub SCR1( )
    ' Определение объектов
    Dim index As Tag
    Dim mytag As Tag
    Dim i As Integer
```

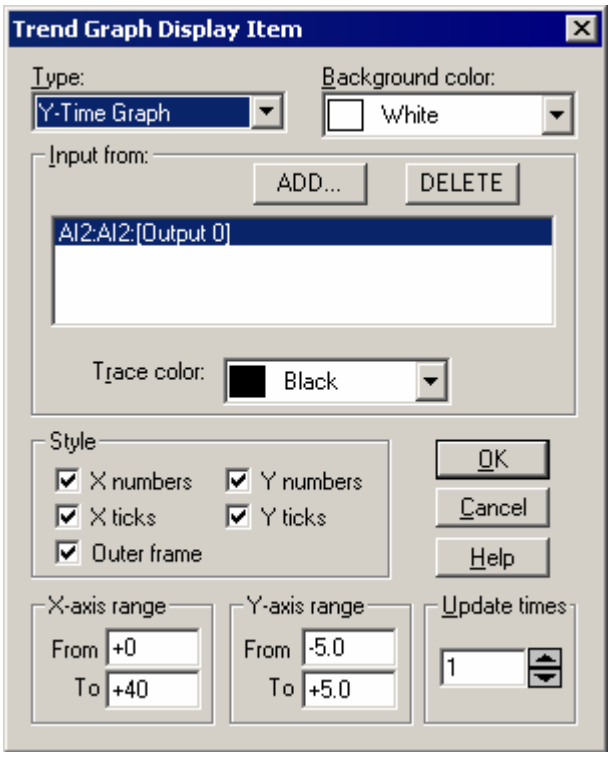

Рис. 4.134. Конфигурация графика для блока аналогового ввода задачи **TASK2**

```
' Инициализация DDE связи с электронной таблицей (Excel)
NumCan% = DDEInitiate( "Excel", "[Book1.xls]Лист1" )
' Выше:
'   использование NumCan% (уникальный номер канала)
'     эквивалентно Dim NumCan as Integer;
'   "Excel" задает приложение, с которым устанавливается
'     связь;
'   "[Book1.xls]Лист1" задает файл (Book1.xls), с
'     которым будет работать электронная таблица и его
'     текущую вкладку (Лист1)
' Получение значения объекта для виртуального тега VT1
Set index = GetTag( "VIRTASK", "VT1" )
' Получение значения объекта для функционального блока
'   аналогового ввода AI1
Set mytag = GetTag( "TASK1", "AI1" )
' Сохранение данных из AI1 в электронной таблице
DDEPoke NumCan%, "R1"+"C"+Format$( index.value ), _
    Format$( mytag.value )
' Выше:
'   Format$( index.value ), Format$( mytag.value )
'     преобразуют значение своего аргумента в строку;
```

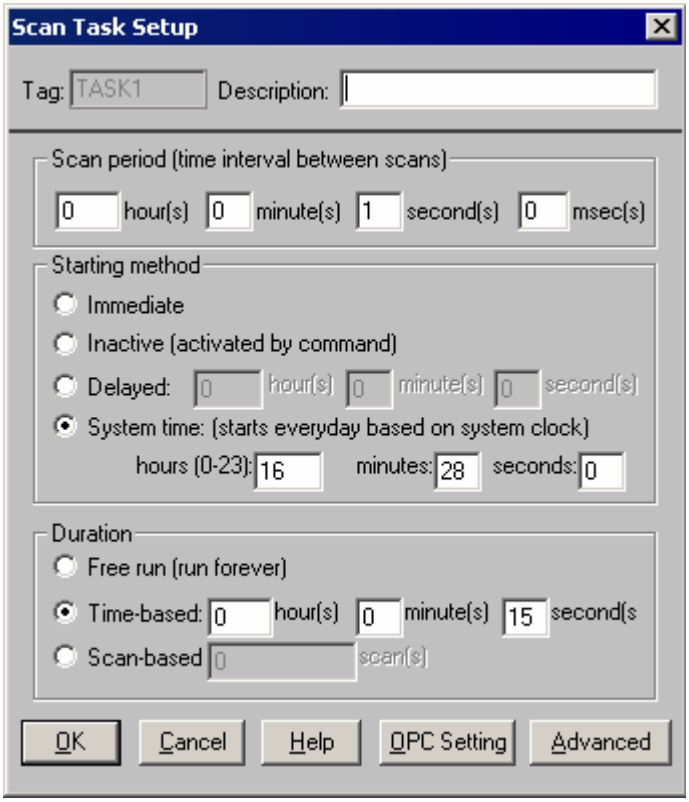

Рис. 4.135. Конфигурирование параметров выполнения задачи **TASK1**

```
    '   "R1" задает вывод в строку 1 файла электронной
    '     таблицы (Row 1);
    '   "C"+Format$( index.value ) задает номер столбца (Col)
    '     электронной таблицы, + операция конкатенации строк;
    '   Format$( mytag.value ) задает значение, выводимое в
    '     ячейку электронной таблицы
    ' Разрыв DDE связи
    DDETerminate NumCan%
    ' Изменение значения index, определяющего номер столбца в
    '   файле электронной таблицы
    i = index.value
    If index.value >= 10 Then
        index.value = 1
    Else
        index.value = i + 1
    End If
End Sub
```

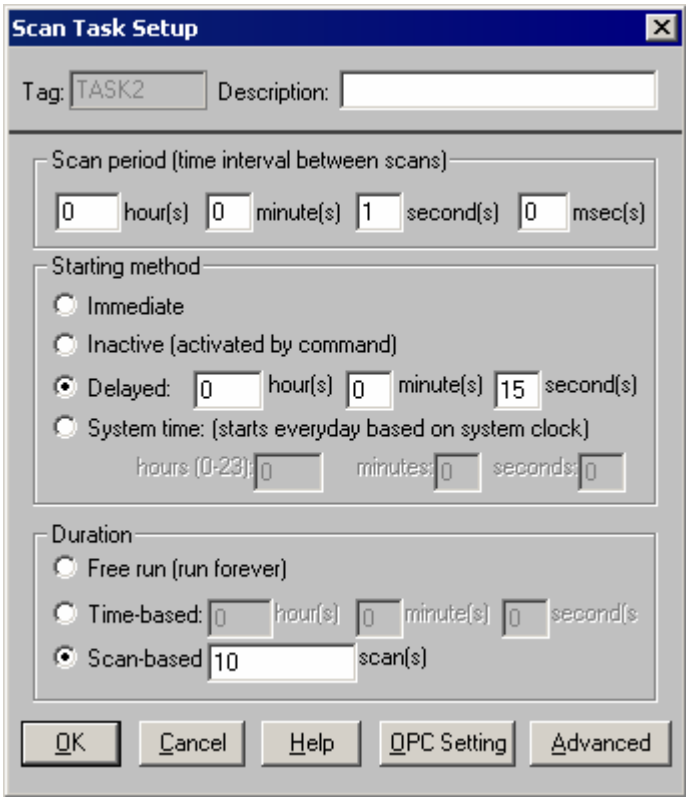

Рис. 4.136. Конфигурирование параметров выполнения задачи **TASK2**

3. *Третье действие*. Задайте временные параметры запуска задачи **TASK1**. Выполните команду **Setup |Task Properties... (Ctrl+T)** задайте параметры задачи, соответствующие умолчанию (период повторения 1 секунда).
4. *Четвертое действие*. Задайте предварительный и пост сценарии задачи в соответствии с рис. 4.122 и 4.123. Полный текст предварительного сценария задачи, снабженный подробными комментариями, также вполне ясен и приведен далее.

```
' Пост-сценарий задачи
Sub POST_TASK1( )
    ' Определение объекта
    Dim excel As Object
    ' Получение значения объекта - электронной таблицы
    Set excel = GetObject( , "Excel.Application" )
    ' Закрытие файла book1.xls с его сохранением и
    '   завершение работы электронной таблицы
    excel.Workbooks( "book1.xls" ).Close SaveChanges := True
    excel.Quit
End Sub
```

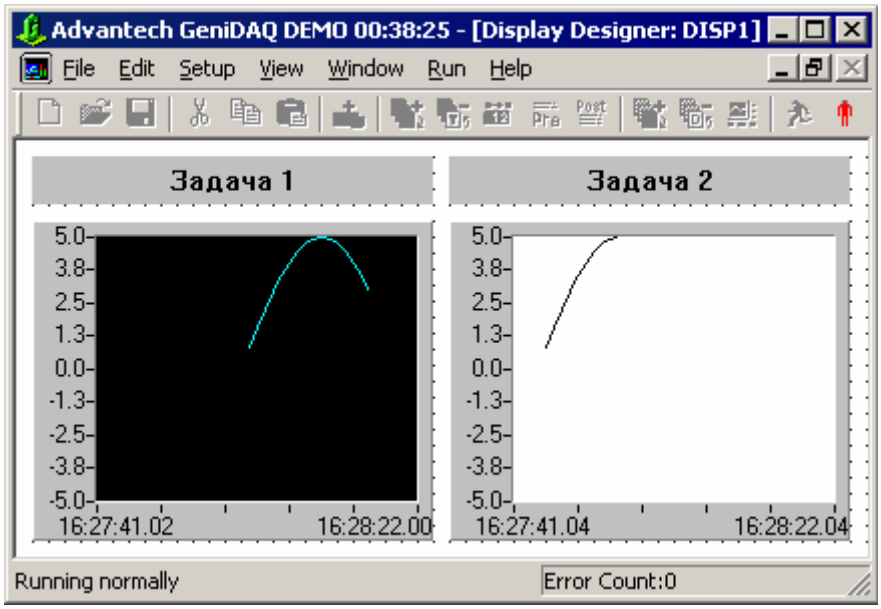

Рис. 4.137. Демонстрация работы проекта в окне отображения

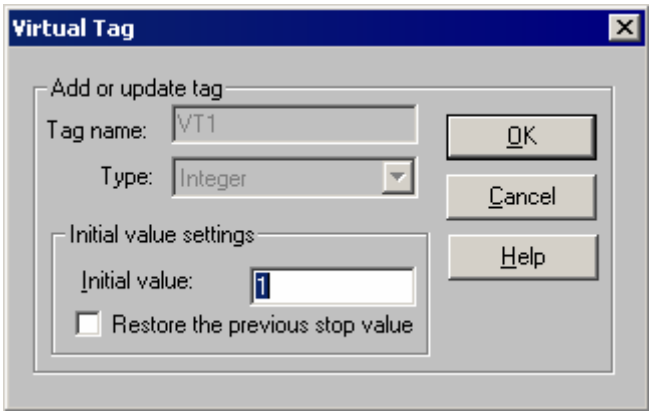

Рис. 4.138. Конфигурирование виртуального тега

5. *Пятое действие*. С помощью приложения **Microsoft Excel** создайте файл Book1.xls в папке проекта.
6. *Шестое действие*. Сохраните созданный проект (стратегию) аналогично тому, как это делалось на занятии 1.
7. *Седьмое действие*. После сохранения файлов созданного проекта запустите созданный проект с помощью кнопки **Start** на панели инструментов (рис. 4.143). Обратите внимание на то, что первом плане отображается электронная таблица в

процессе ее заполнения. Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов, предварительно перейдя в окно SCADA системы.

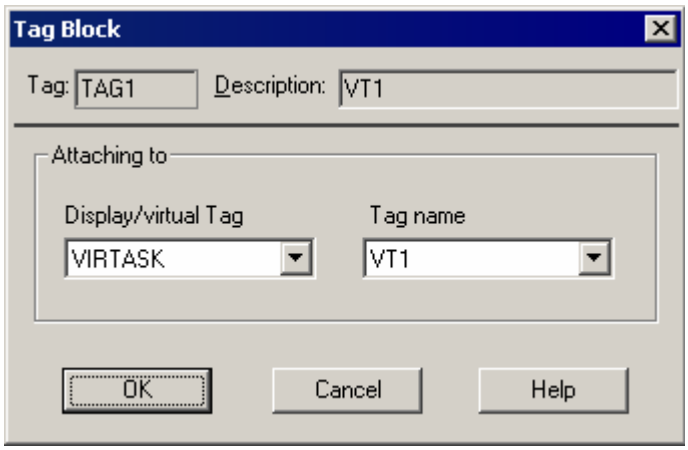

Рис. 4.139. Конфигурирование тега

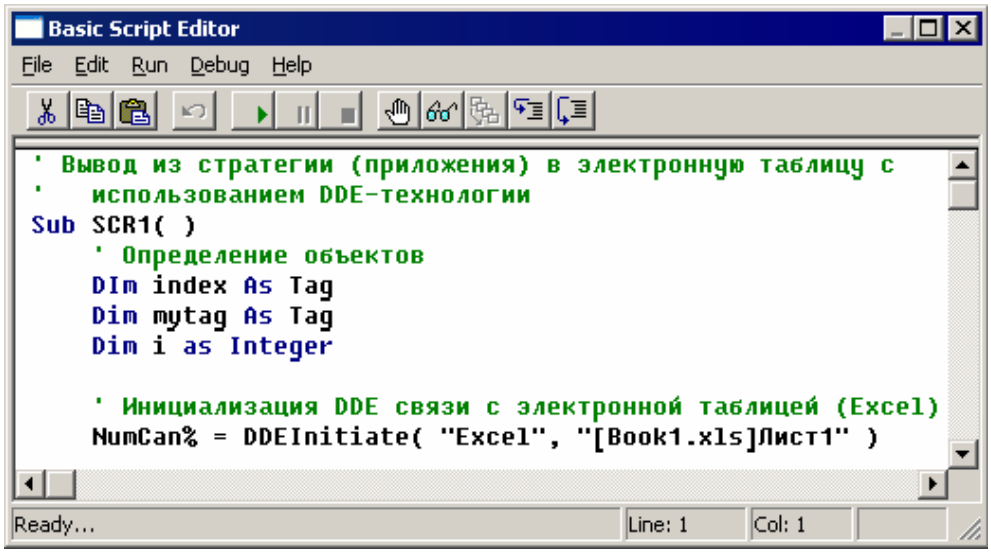

Рис. 4.140. Функциональный блок Бейсик-сценария

8. *Восьмое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

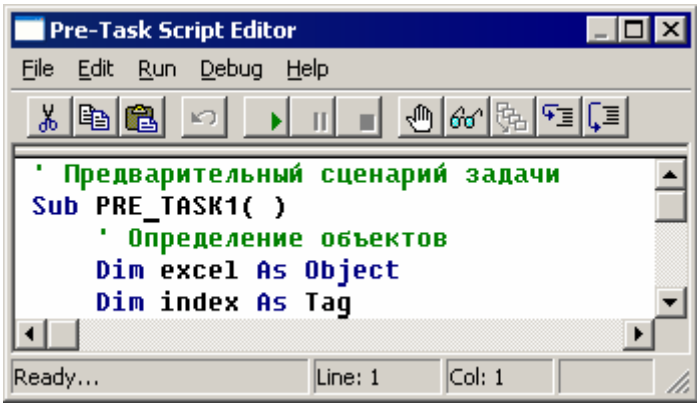

Рис. 4.141. Предварительный сценарий задачи **TASK1**

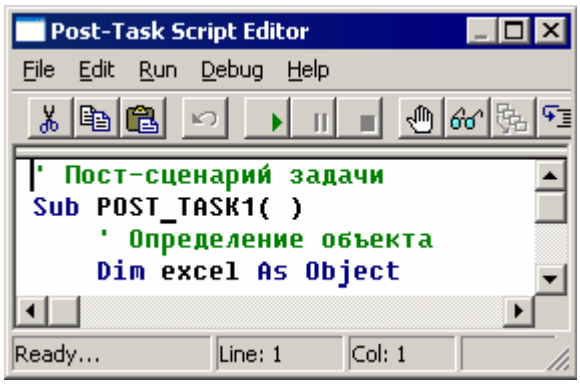

Рис. 4.142. Пост сценарий задачи **TASK1**

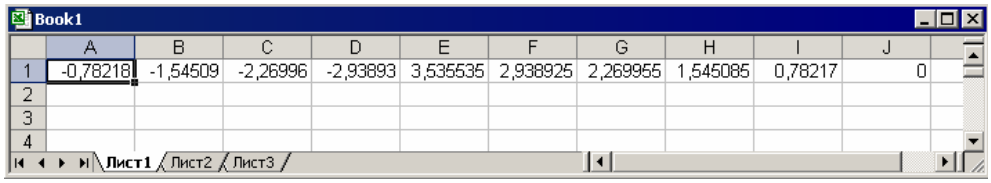

Рис. 4.143. Горизонтальное заполнение электронной таблицы

4.11.2. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.29. Повтор создания и тестирования приложения "Занятие 10"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 10**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Упражнение 4.30

Разработайте стратегию, обладающую следующими возможностями. Стратегия (приложение) должна выводить в электронную таблицу (файл "Пример.xls") выходной сигнал блока аналогового ввода, использующий эмулятор пилообразного сигнала. Вывод в электронную таблицу выполнить циклически по главной диагонали (15 значений по кольцу).

Упражнение 4.31

Разработайте стратегию, обладающую следующими возможностями. Стратегия (приложение) должна выводить в электронную таблицу (файл "Прямоугольник.xls") выходной сигнал блока аналогового ввода, использующий эмулятор прямоугольного сигнала. Вывод в электронную таблицу выполнить циклически по вспомогательной диагонали (12 значений по кольцу).

Упражнение 4.32

Разработайте стратегию, обладающую следующими возможностями. Стратегия (приложение) должна выводить в электронную таблицу (файл "Пила.xls") выходной сигнал блока аналогового ввода, использующий эмулятор пилообразного сигнала. Вывод в электронную таблицу выполнить циклически по сторонам квадрата размером 8 по часовой стрелке.

Упражнение 4.33

Разработайте стратегию, обладающую следующими возможностями. Стратегия (приложение) должна выводить в электронную таблицу (файл "Пила.xls") выходной сигнал блока аналогового ввода, использующий эмулятор гармонического сигнала. Вывод в электронную таблицу выполнить циклически по сторонам квадрата размером 10 против часовой стрелки.

4.12. Занятие 11 "Работа с таймером"

Цель занятия состоит в изучении правил работы с *таймером*. С этой целью создается стратегия, использующая *функциональные блоки таймера (***Timer***)* и *метки времени (***Time Stamp***)* редактора задач.

4.12.1. Используемый инструментарий

Новыми инструментами на данном занятии являются функциональные блоки таймера и метки времени.

Функциональный блок таймера редактора задач предназначен для реализации таймеров различных типов и имеет вход сброса. Таймер может выполнять функции формирования абсолютных или относительных временных интервалов с разрешением 0.1 секунды или 1 секунда. Цикл таймера до сброса с последующим возобновлением работы может составлять от 1 минуты до 1 года. Содержимое таймера (относительное или абсолютное время) может быть передано другому функциональному блоку стратегии. Выходное значение блока таймера представляется в виде длинного целого значения в диапазоне от 0 до 4294967295. Данный блок является весьма удобным средством для реализации различных алгоритмов управления, выполнение которых основывается на интервалах времени. Диалоговое окно настройки параметров блока показано на рис. 4.144. Единицы, в которых выражается значение на выходе блока таймера, могут быть выбраны с помощью переключателя *разрешающая способность* (**Resolution**). Таким образом, разрешающая способность таймера может составлять 1 тик (0.1 секунды) или 1 секунда. Получение более высокого разрешения в рамках операционной системы Windows представляется весьма трудной задачей. Цикличность (периодичность) работы таймера может быть выбрана с помощью переключателя *цикл* (**Cycle**) диалогового окна настройки параметров блока. По истечении периода времени, выбранного с помощью указанного переключателя, содержимое таймера сбрасывается в ноль и работа таймера возобновляется. Например, если выбран цикл, равный 1 минуте (**By minute**), а разрешение (**Resolution**) — 1 секунда, то содержимое таймера в каждом цикле будет увеличиваться от 0 до 59 с последующим возобновлением. Данная функция блока таймера в текущей версии пакета работает только при использовании разрешающей способности, равной 1 секунде. Содержимое блока таймера может быть сброшено в 0 в процессе его работы путем подачи на вход блока дискретного сигнала с уровнем логической единицы. Для возобновления работы таймера следует подать на вход блока дискретный сигнал с уровнем логического нуля.

При использовании таймера, отсчитывающего абсолютное время (переключатель *тип таймера* — **Timer type** — установлен в положение *прошедшее время* — **Elapsed time**), после запуска стратегии на исполнение алгоритм работы таймера объясняется следующим примером. Если разрешающая способность таймера составляет 1 секунду, а в качестве цикла таймера выбрана 1 минута, то при запуске стратегии на исполнение в 11:23:17 (по часам реального времени компьютера) работа таймера начнется со значения, равного 17, и будет продолжаться 59 секунд последующим сбросом и дальнейшим возобновлением работы. Нулевые начальные значения содержимого таймера при использовании указанного режима работы приведены в табл. 4.19.

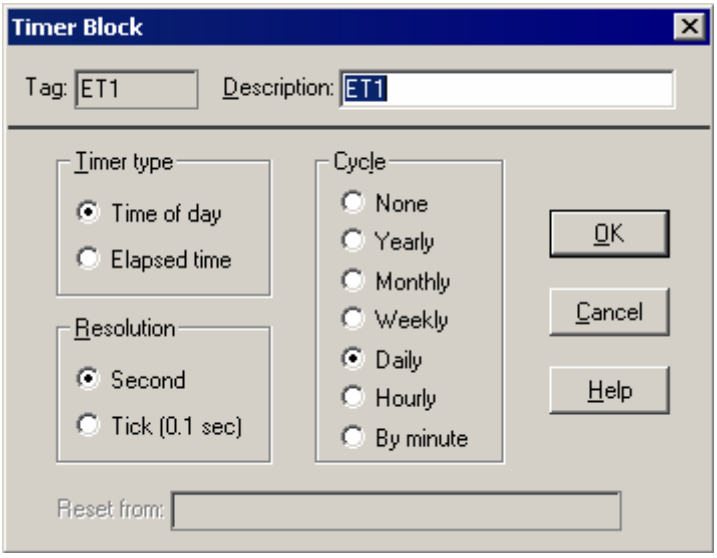

Рис. 4.144. Диалоговое окно для конфигурирования функционального блока таймера

Таблица 4.19. *Нулевые начальные значения функционального блока таймера*

Цикл таймера	**Значение системного времени для нулевого начального содержимого таймера**
Год (yearly)	00:00:00, 1 января каждого года
Месяц (monthly)	00:00:00, в первый день каждого месяца
Неделя (weekly)	00:00:00, в воскресенье каждой недели
Сутки (daily)	00:00:00, в начале каждых суток
Час (hourly)	Каждый час
Минута (by minute)	Каждую минуту

Таким образом, имеется возможность вычисления текущего времени. Например, таймер с циклом *неделя* и разрешением 1 секунда при запуске стратегии в 0:00 в понедельник будет иметь начальное содержимое, равное 86400, а в полдень (12:00) пятницы его содержимое увеличится до 518400. В результате появляется возможность автоматического отключения какой-либо единицы контролируемого оборудования в течение выходных. Блок таймера имеет один дискретный вход, предназначенный для сброса содержимого таймера с последующим возобновлением

его работы, и один выход, по которому выводится абсолютное или относительное значение времени (длинного целого типа) в заданных единицах разрешения таймера.

Функциональный блок метка времени имеет единственный выход, присоединение которого к элементу отображения или блоку архивации данных позволяет получать значение системного времени в виде строки символов. Имеется возможность использования различных форматов представления системного времени (рис. 4.145).

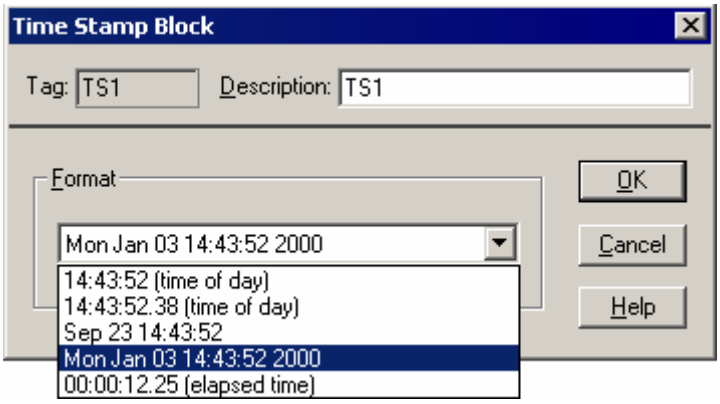

Рис. 4.145. Диалоговое окно для конфигурирования *метки времени*

Входная связь блока отсутствует — при попытке присоединения к блоку метки времени проводника от другого функционального блока на экран монитора будет выведено сообщение **Вход недоступен** (**Cannot accept input**). Выходная связь блока позволяет передавать другим функциональным блокам стратегии значение системного времени в виде строки символов или в ином формате (рис. 4.146).

4.12.2. Проектирование приложения

1. Выполните *первое действие* из занятия 1. Создаваемое приложение назовите **Занятие 11**.
2. *Второе действие.* Сделайте активным окно редактора задач и поместите в него функциональные блоки таймера, реализации простой операции деления, Бейсик-сценария и метки времени, соединив их в соответствии с рис. 4.129. Настройте функциональные блоки в соответствии с рис. 4.144, 4.148 — 4.153 и 4.145. Такая настройка блоков обеспечивает на выходе блока деления **SOC1** значение текущего часа, на выходе **SOC2** — текущей минуты, на выходе **ET3** — текущей секунды, а на выходе **ET4** — текущего тика текущей секунды.
3. *Третье действие.* Активизируйте окно редактора форм отображения. Добавьте в это окно элементы отображения в соответствии с рис. 4.154. Сконфигурируйте их в соответствии с рис. 4.155 — 4.159.

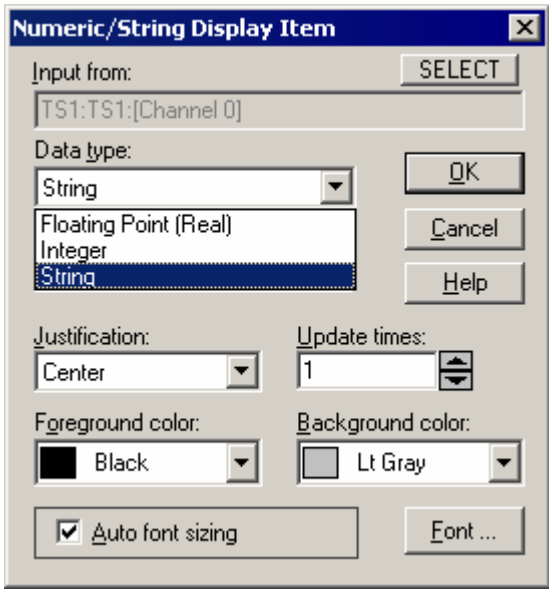

Рис. 4.146. Форматы выдачи информации из блока метки времени в другие блоки

4. *Четвертое действие*. Задайте временные параметры запуска задачи — непрерывный режим работы с периодом перезапуска 100 миллисекунд.
5. *Пятое действие*. Сохраните созданный проект (стратегию) аналогично тому, как это делалось на занятии 1.
6. *Шестое действие*. После сохранения файлов созданного проекта запустите созданный проект с помощью кнопки <*Start*> на панели инструментов (рис. 4.160). Остановить работу проекта можно с помощью кнопки <*Stop*> на панели инструментов.
7. *Седьмое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.

4.11.2. Упражнения

> ***Совет***
> Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.34. Повтор создания и тестирования приложения "Занятие 11"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 11**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

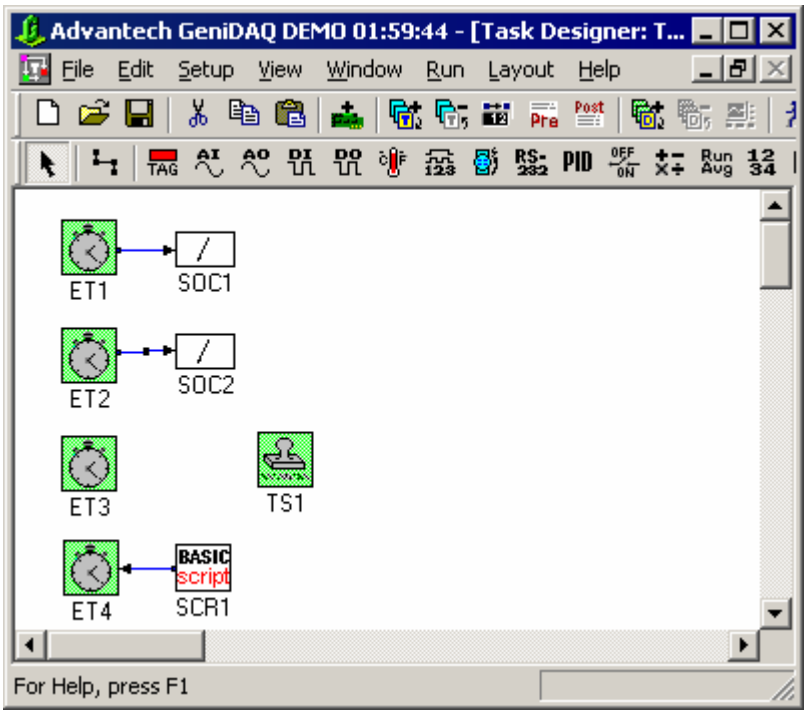

Рис. 4.147. Окно редактора задач с функциональными блоками

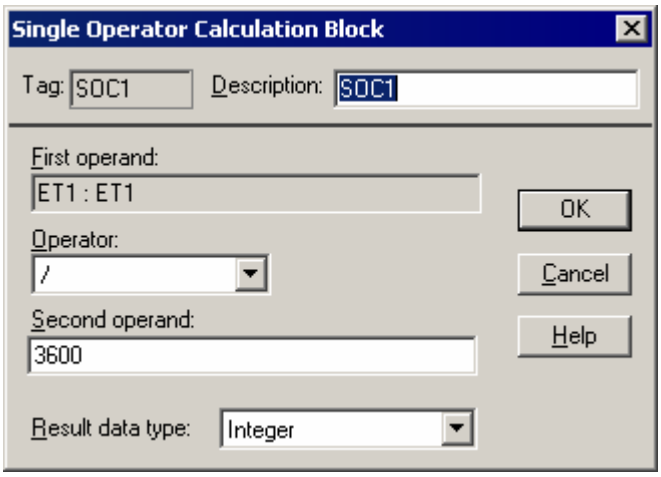

Рис. 4.148. Конфигурирование блока деления **SOS1**

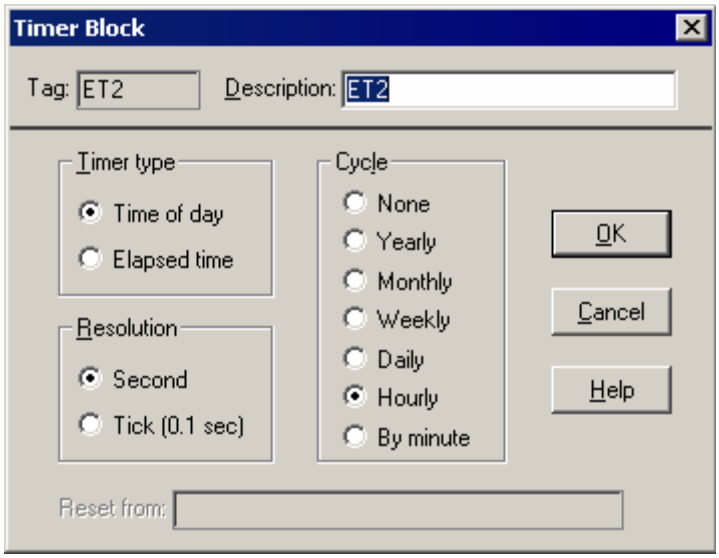

Рис. 4.149. Конфигурирование блока таймера **ET2**

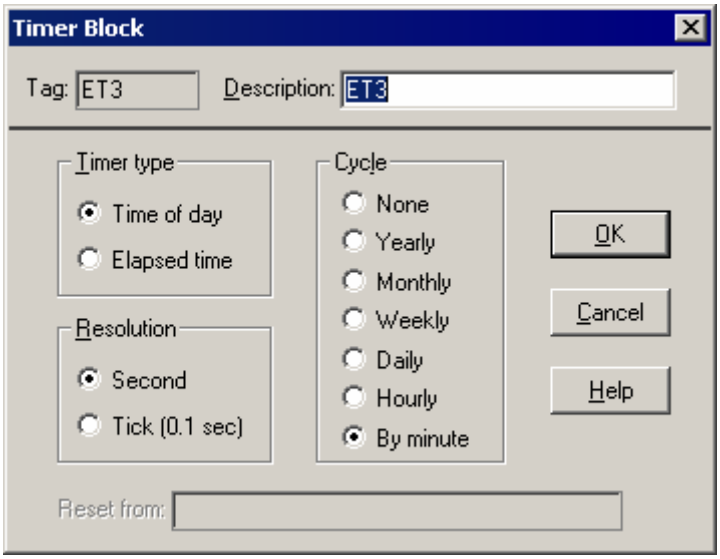

Рис. 4.150. Конфигурирование блока таймера **ET3**

Упражнение 4.35

Разработайте стратегию, обладающую следующими возможностями. Стратегия должна реализовать секундомер (секунды-десятые доли секунды). При нажатии

кнопки секундомер запускается, а при нажатии другой — фиксируется. Повторное нажатие кнопки сбрасывает показание, вновь запускает секундомер и т. д.

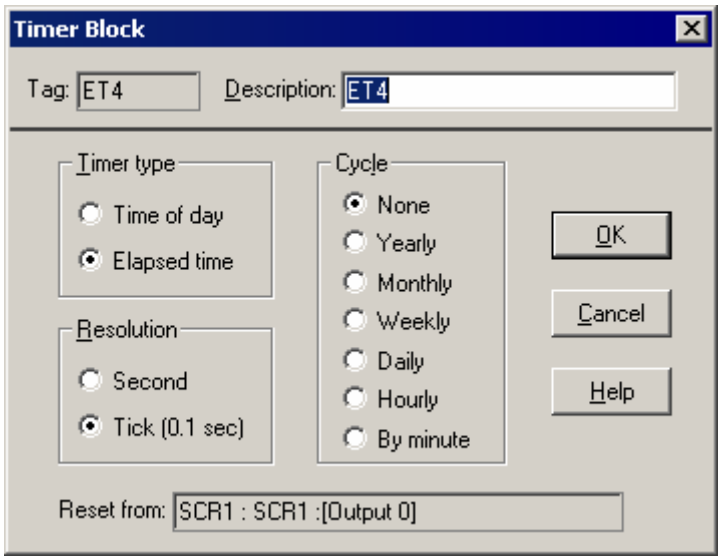

Рис. 4.151. Конфигурирование блока таймера **ET4**

Упражнение 4.36

Разработайте стратегию, обладающую следующими возможностями. Стратегия должна реализовать секундомер (секунды-десятые доли секунды). При нажатии кнопки секундомер запускается, а при ее отжатии — фиксируется. Сброс показания секундомера — от отдельной кнопки.

Упражнение 4.37

Разработайте стратегию, обладающую следующими возможностями. Стратегия должна реализовать секундомер (секунды-десятые доли секунды). При нажатии кнопки секундомер запускается, а при ее отжатии — фиксируется. Использовать один цифровой индикатор.

Замечание

В следующей далее главе приводится методика информационной интеграции SCADA-системы GeniDAQ и промышленного контроллера, базирующаяся на использовании OPC-стандарта. В качестве промышленного контроллера используется контроллер типа **7188E2 фирмы ICP DAS (Тайвань)**, содержащий устройства связи с объектом (УСО), а в качестве OPC-сервера рассматривается OPC-сервер **OpenLabOPCSvr.AiVT.2**, разработанный на кафедре автоматики и вычислительной техники Санкт-Петербургского государственного политехнического университета с участием автора. Изложенная методика интересна тем, что может быть использована для информационной связи SCADA-системы GeniDAQ и имеющегося в распоряжении любого другого промышленного контроллера, снабженного OPC-сервером. Другой

пример информационной интеграции SCADA-системы **Vijeo Citect** (компаний Citect, Австралия и Schneider Electric, Франция) и промышленного контроллера **Twido** (Schneider Electric), снабженного групповым OPC-сервером **OPC Factory Server** (Schneider Electric), рассматривается далее во второй части учебного пособия.

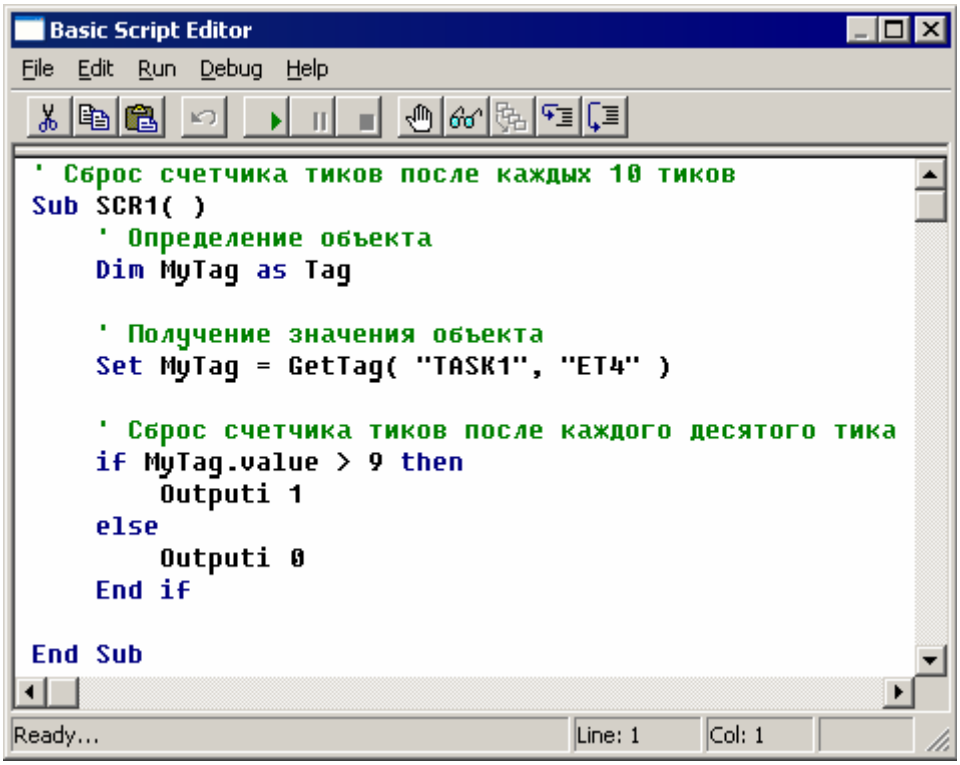

Рис. 4.152. Конфигурирование блока Бейсик-сценария

Глава 5. OPC-технология информационной интеграции SCADA-системы и промышленного контроллера

В главе последовательно приводится краткое описание промышленного контроллера **7188E2 фирмы ICP DAS (Тайвань)**, рассматриваются интерфейс OPC-сервера **OpenLabOPCSvr.AiVT.2** и его конфигурирование, настройка компьютеров локальной вычислительной сети и методика информационного обмена между функциональными блоками SCADA-системы и промышленного контроллера.

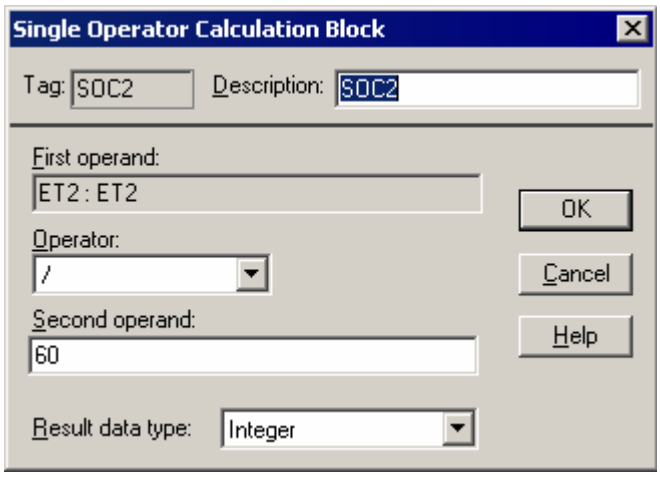

Рис. 4.153. Конфигурирование блока деления **SOS2**

5.1. Промышленный контроллер 7188E2 фирмы ICP DAS (Тайвань)

Промышленный контроллер типа 7188E2 фирмы ICP DAS (Тайвань) используется для взаимодействия с объектами управления и содержит устройства ввода-вывода аналоговой и дискретной информации. В настоящее время большинство новых моделей производимых контроллеров снабжено интерфейсным выходом на Ethernet. К подобным контроллерам обычно присоединяются пассивные модули ввода/вывода. Обращение из host-компьютера к модулям ввода/вывода происходит косвенно через контроллер. Именно такая конфигурация используется в рассматриваемом случае в качестве одного из возможных решений (рис. 5.1).

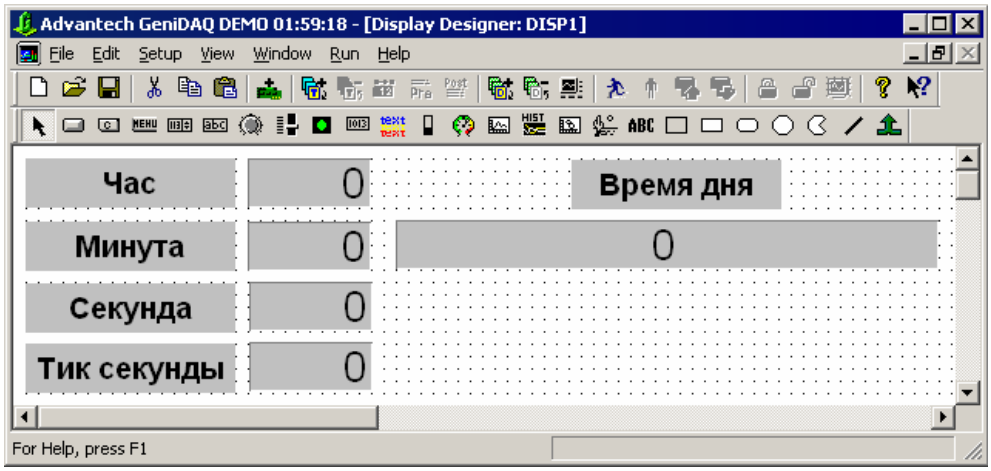

Рис. 4.154. Конфигурация окна отображения

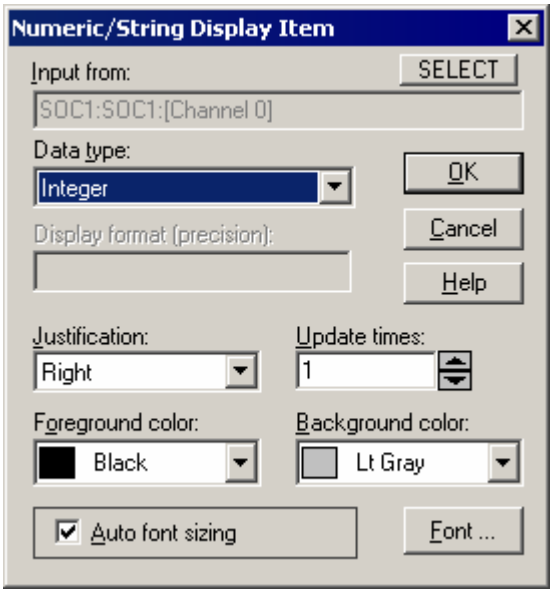

Рис. 4.155. Конфигурация цифрового индикатора для блока **SOS1**

Промышленный контроллер подсоединяется к host-компьютеру через выход Ethernet компьютера. В сети содержатся два восьмиканальных *модуля аналогового ввода I-7017* (±10 В), два четырехканальных *модуля аналогового вывода I-7024* (±10 В) и один *модуль дискретного ввода/вывода I-7050* с семью входами, восемью выходами и семью счетчиками для дискретных входов.

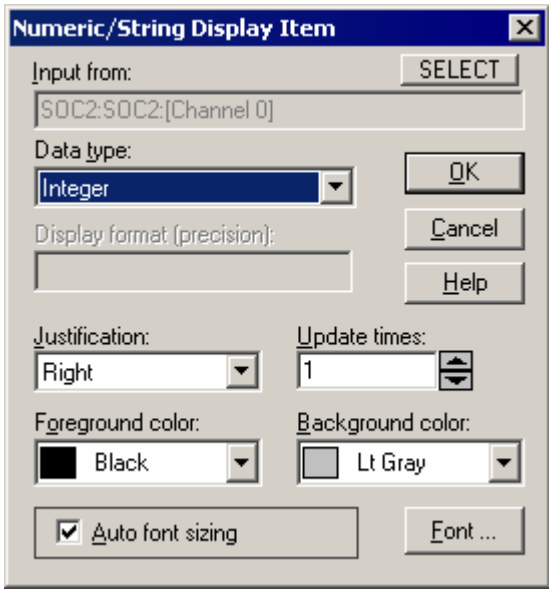

Рис. 4.156. Конфигурация цифрового индикатора для блока **SOS2**

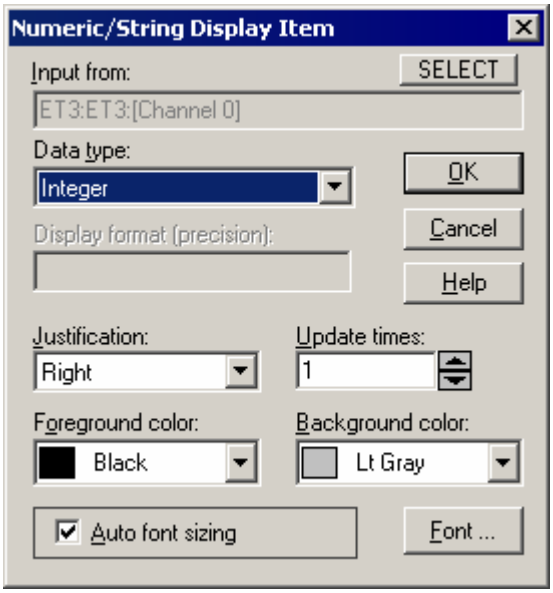

Рис. 4.157. Конфигурация цифрового индикатора для блока **ET3**

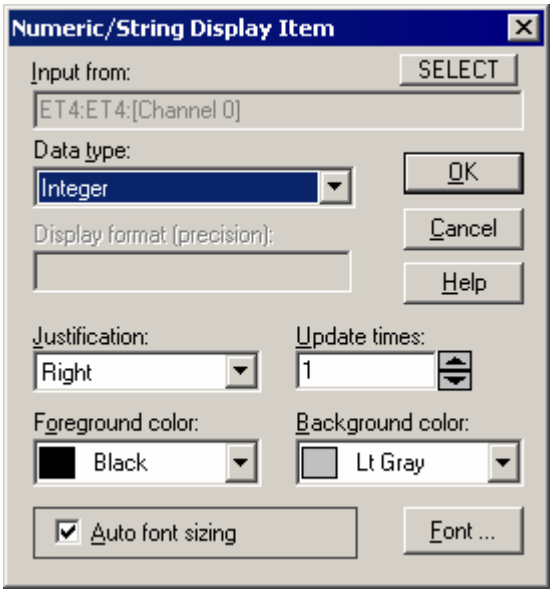

Рис. 4.158. Конфигурация цифрового индикатора для блока **ET4**

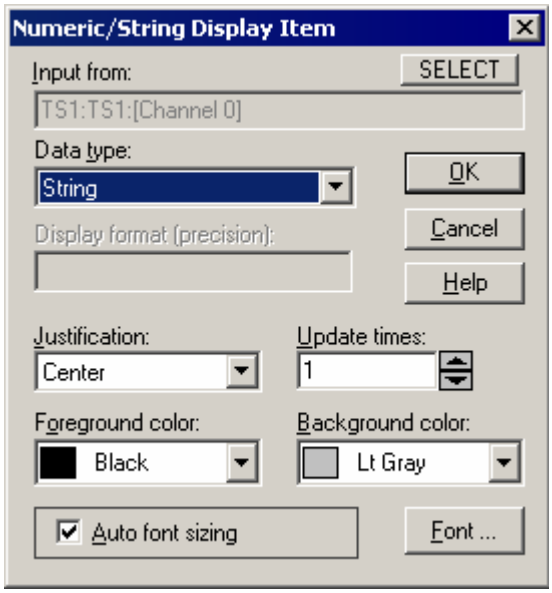

Рис. 4.159. Конфигурация цифрового индикатора для блока **TS1**

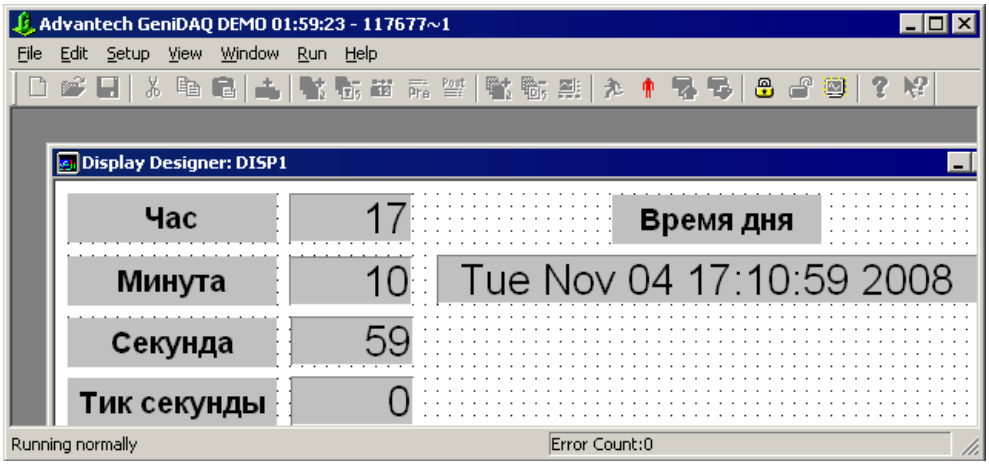

Рис. 4.160. Вид окна отображения при работе приложения

Каждый модуль имеет уникальный адрес (см. рис. 5.1). Для обеспечения доступа к контроллеру используется сервер VxComm, который создает в host-компьютере множество виртуальных COM-портов. Запросы ввода/вывода от OPC-сервера по интерфейсу RS-232 поступают в сервер VxComm и сервер, в свою очередь, преобразует сообщения RS-232 в пакеты TCP/IP и передает контроллеру 7188E2. При этом программисту нет необходимости программировать обмен по сети, например, знать протокол TCP/IP. Ему надо владеть лишь традиционным протоколом RS-232 и обладать навыками традиционного программирования в области промышленной автоматизации. Время доступа к данным контроллера и каналам ввода/вывода определяется скоростью передачи данных используемого виртуального COM-порта. Промышленный контроллер обладает следующими техническими характеристиками.

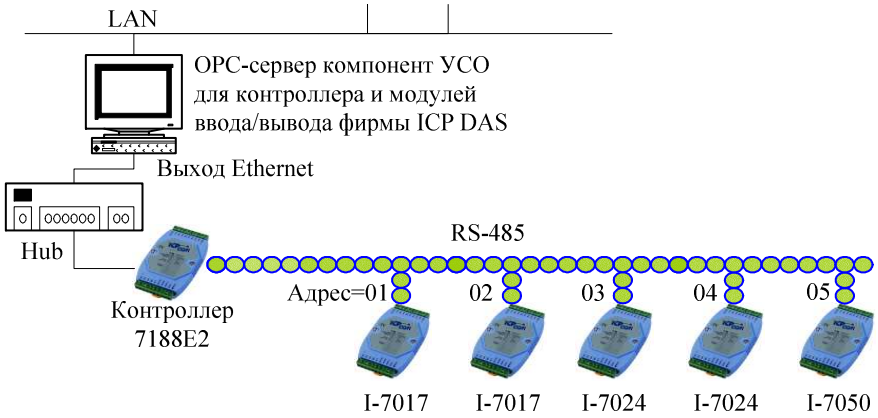

Рис. 5.1. Физическая структура системы мониторинга и управления модулями ввода-вывода промышленного контроллера типа 7188E2 фирмы ICP DAS

Процессор AMD80188ES с тактовой частотой 40 МГц. Операционная система реального времени MiniOS7, разработанная специально для контроллеров. Из нее исключены неиспользуемые для контроллеров функции, упрощена файловая система, добавлена поддержка модулей ввода-вывода и слотов расширения, устройств дополнительной памяти, функций самодиагностики, управления Flash-ROM, ускорена процедура загрузки и др. Flash-память и RAM-память по 256 Ком. Имеются встроенные часы реального времени и двойной сторожевой таймер.

Модуль ввода аналоговых сигналов I-7017 имеет восемь входных канала ±10 Вольт, 16 разрядов, скорость опроса 10 сигналов в секунду. В составе контроллера используются два модуля.

Модуль вывода аналоговых сигналов I-7024 имеет четыре выходных канала ±10 Вольт, 12 разрядов, скорость опроса 10 сигналов в секунду. В составе контроллера используются два модуля.

Модуль дискретного ввода-вывода I-7050 имеет семь входных канала для сигналов *TTL*-уровня (логический 0 соответствует диапазону 0-1 Вольт, логическая 1 – диапазону 3.5-30 Вольт), семь счетчиков для входных сигналов (входная частота до 100 Гц) и восемь выходных каналов с открытым коллектором (30 Вольт, 30 мА). Скорость опроса 10 сигналов в секунду. В составе контроллера используется один модуль.

5.2. Интерфейс OPC-сервера OpenLabOPCSvr.AiVT.2 и его конфигурирование

После включения компьютера, на котором реализован OPC-сервер, и промышленного контроллера на экране компьютера появляется запрос на ввод атрибутов пользователя (имя, пароль, права администратора), которые следует получить от преподавателя. Для запуска OPC-сервера с подключенным эмулятором тестовых сигналов достаточно запустить файл Системный диск\Program Files\ OpenLabOPCSvr_7188E&Emulator\OpenLabOPCSvr.exe. Для этой цели удобнее всего воспользоваться ярлыком **7188E2&Emulator**, который следует предварительно разместить на рабочем столе. В результате этого на экране появляется окно, представленное на рис. 5.2.

В этом окне на переднем плане появляется диалоговое окно, информирующее о способе задания параметров эмулируемых сигналов. Нажмите в диалоговом окне кнопку **OK**.

Для управления работой и конфигурирования OPC-сервера и эмулятора тестовых сигналов в окне, представленном на рис. 5.2, имеется строка меню и панель инструментов. По умолчанию, в левом поле окна выбран эмулятор тестовых сигналов **Emulator**, а в правом поле отображается информация о составе и параметрах тестовых сигналов. Если в левом поле выбрать контроллер **7188E2** и нажать кнопку **Monitoring** на панели инструментов, то в правом поле будет отображена информация об уже сконфигурированных тегах контроллера (рис. 5.3).

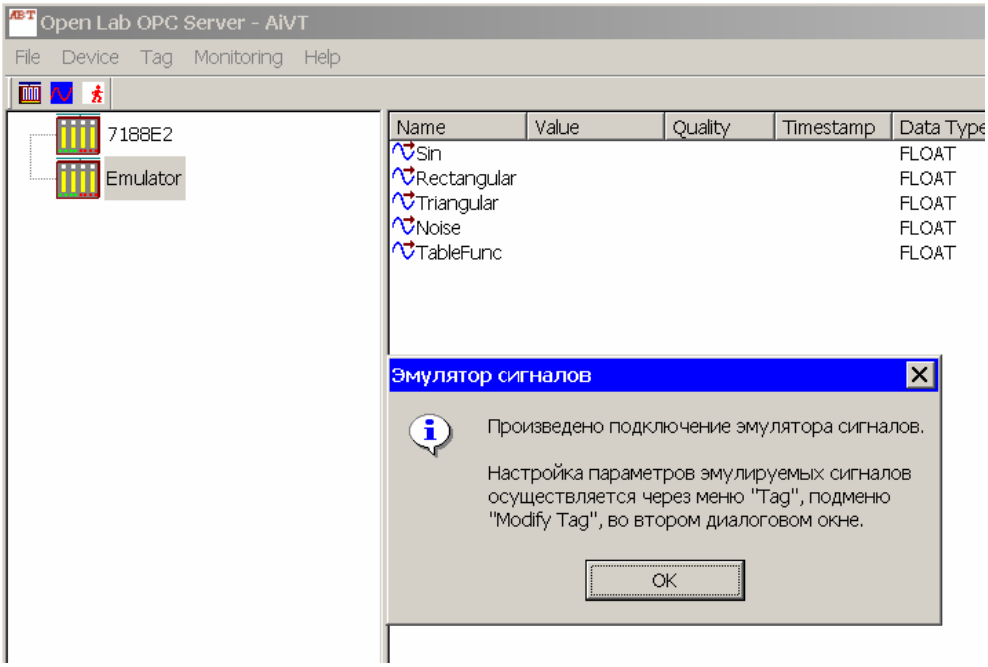

Рис. 5.2. Вид окна после запуска OPC-сервера **OpenLabOPCSvr.AiVT.2** с эмулятором тестовых сигналов

Интерфейс OPC-сервера с эмулятором представлен строкой меню с пятью меню, панелью инструментов с тремя кнопками, двумя названными ранее окнами и строкой состояния (статуса). Меню **File** содержит команды **Open Config** (акселератор **Ctrl+O**) — выбор и загрузка из файла ранее созданной конфигурации, **Save Config** (акселератор **Ctrl+S**) — сохранение в файле текущей конфигурации, **Exit** — завершение работы сервера. Меню **Device** содержит команды **Load New Device** (акселератор **Ctrl+L**, кнопка **New Device** на панели инструментов) — создание нового устройства, **Modify Device** (акселератор **Ctrl+M**) — модификация выбранного устройства, **Unload Device** (акселератор **Ctrl+U**) — удаление из конфигурации выбранного устройства. На приведенном ранее рис. 5.3 текущая конфигурация содержит устройства **7188E2** (контроллер) и **Emulator**. Меню **Tag** содержит команды **Add New Tag** (акселератор **Ctrl+A**, кнопка **New Tag** на панели инструментов) — добавление в устройство нового тега, **Modify Tag** (акселератор **Ctrl+F**) — модификация выбранного тега, **Delete Tag** (акселератор **Del**) — удаление из конфигурации выбранного тега. С помощью команды **Tag | Modify Tag** можно модифицировать тестовый сигнал эмулятора. Например, если выбрать тестовый сигнал **Sin**, и выполнить для него команду **Tag | Modify Tag** и нажать кнопку **OK**, то появится диалоговое окно, представленное на рис. 5.4.

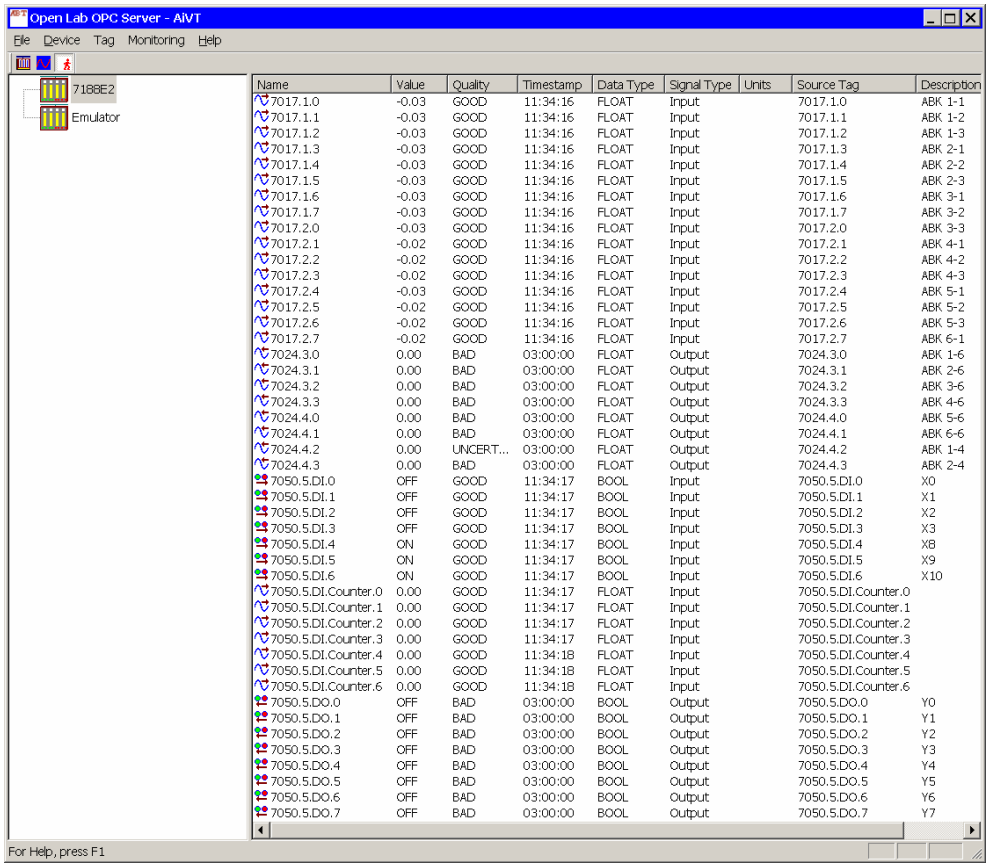

Рис. 5.3. Мониторинг сконфигурированных тегов контроллера

В этом окне можно посмотреть текущие значения параметров тестового сигнала или изменить их. Для выхода из диалогового окна следует нажать кнопку **OK**. Меню **Monitoring** содержит команды **Toolbar** — отображение или скрытие панели инструментов, **Status Bar** — отображение или скрытие строки состояния (статуса), **Tag Monitoring** (кнопка **Monitoring**) — мониторинг тегов выбранного устройства. И, наконец, команда **Help | About OPC Server…** с помощью диалогового окна отображает информацию об OPC-сервере.

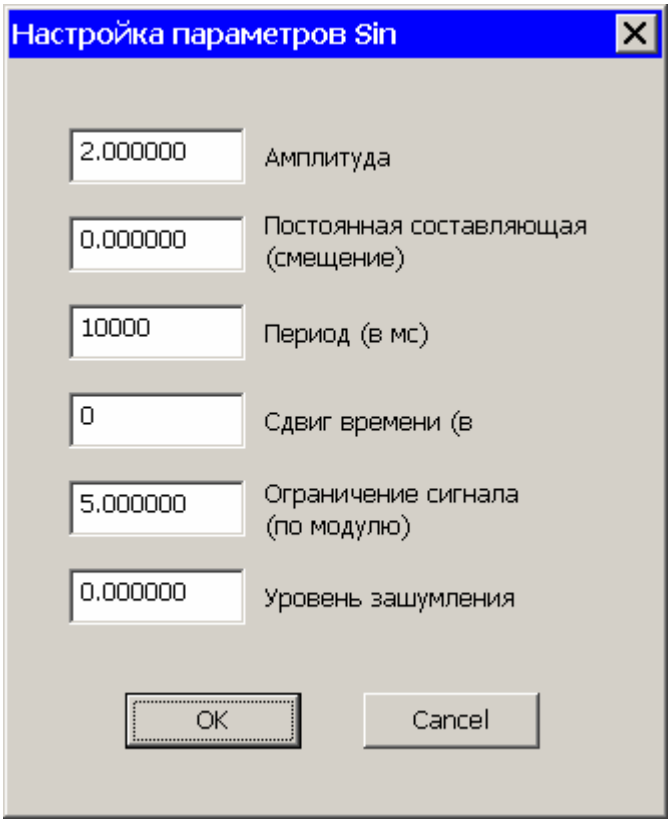

Рис. 5.4. Модификация тестового сигнала **Sin** эмулятора

5.3. Настройка компьютеров локальной вычислительной сети

Для обеспечения доступа к OPC-серверу из любого компьютера локальной вычислительной сети следует выполнить дополнительную настройку компьютеров, которая несколько отличается для различных операционных систем.

5.3.1. Настройка ОС Windows2000 для работы с OPC-серверами (на примере OPC-сервера OpenLabOPCSvr.AiVT.2)

1. При регистрации входа на компьютере *обязательно* используйте имя пользователя и пароль как у компьютера, на котором работает OPC-сервер, и с правами администратора.

2. В папке **..\Program Files** создайте папку **OpenLabOPCSvr** (можно использовать и другое имя папки, но лучше, чтобы имя папки совпадало с именем OPC-сервера). Из ЭВМ, где работает OPC-сервер (пусть, например, такой компьютер имеет имя **Comp1_2003**) из аналогичной папки в созданную папку копируем файлы **OPCProxy.dll** и **Setup.reg**.
3. Запустите на выполнение скопированный файл **Setup.reg**. В появляющихся дважды диалоговых окнах нажмите кнопку **OK**.
4. Зарегистрируйте **OPCProxy**, для чего в командной строке выполните команду

```
regsvr32  OPCProxy.dll
```

В появившемся диалоговом окне нажмите кнопку **OK**.

5. Сконфигурируйте DCOM. С этой целью выполните команду **Пуск | Выполнить...**, в появившемся диалоговом окне (рис. 5.5) введите **dcomcnfg** и нажмите кнопку **OK**. В ответ на появившийся запрос нажмите кнопку **Да**.

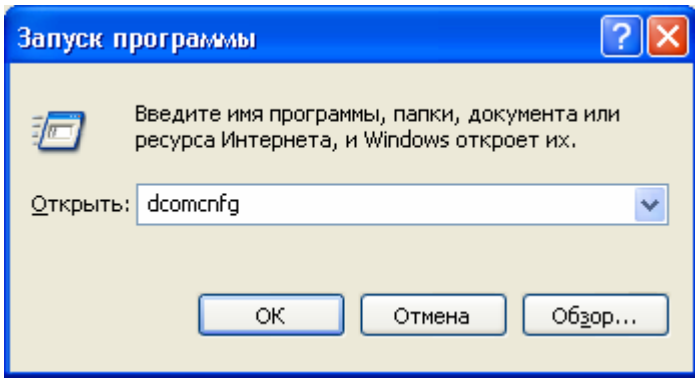

Рис. 5.5. Конфигурирование **DCOM**

6. В появившемся диалоговом окне (рис. 5.6) выберите OPC-сервер (в нашем примере это **Opened Lab OPC Server**) и нажмите кнопку **Свойства...**
7. В появившемся диалоговом окне выберите вкладку **Свойства** и настройте ее в соответствии с рис. 5.7. Здесь, в качестве примера, **Comp1_2003** является именем компьютера, на котором работает OPC-сервер. После настройки дважды нажмите кнопку **ОК**.

5.3.2. Настройка ОС Windows XP для работы с OPC-серверами (на примере OPC-сервера OpenLabOPCSvr.AiVT.2)

1. При регистации входа на компьютере *обязательно* используйте имя пользователя и пароль как у компьютера, на котором работает OPC-сервер, и с правами администратора.

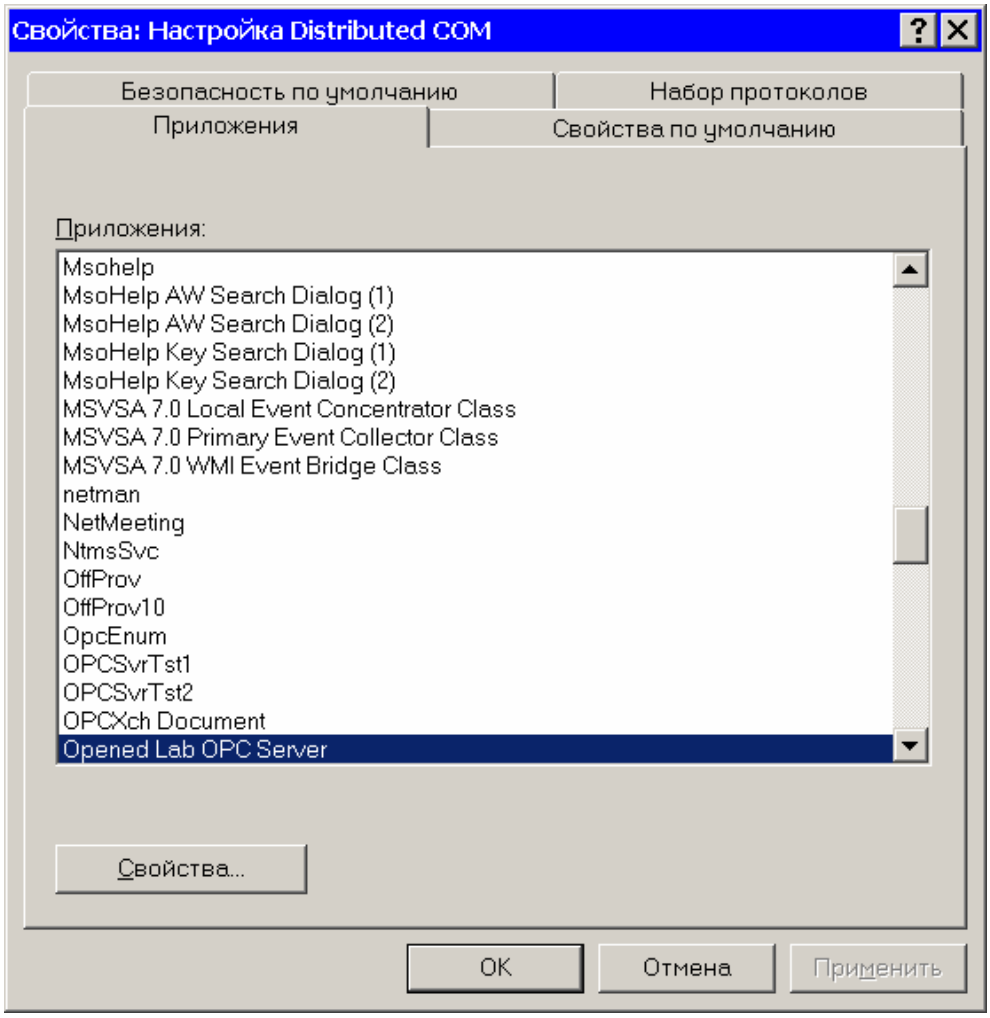

Рис. 5.6. Выбор OPC-сервера

2. В папке **..\Program Files** создайте папку **OpenLabOPCSvr** (можно использовать и другое имя папки, но лучше, чтобы имя папки совпадало с именем OPC-сервера). Из ЭВМ, где работает OPC-сервер (пусть, например, такой компьютер имеет имя **Comp1_2003**) из аналогичной папки в созданную папку копируем файлы **OPCProxy.dll** и **Setup.reg**.
3. Запустите на выполнение скопированный файл **Setup.reg**. В появляющихся дважды диалоговых окнах нажмите кнопку **OK**.
4. Зарегистрируйте **OPCProxy**, для чего в командной строке выполните команду

```
regsvr32  OPCProxy.dll
```

В появившемся диалоговом окне нажмите кнопку **OK**.

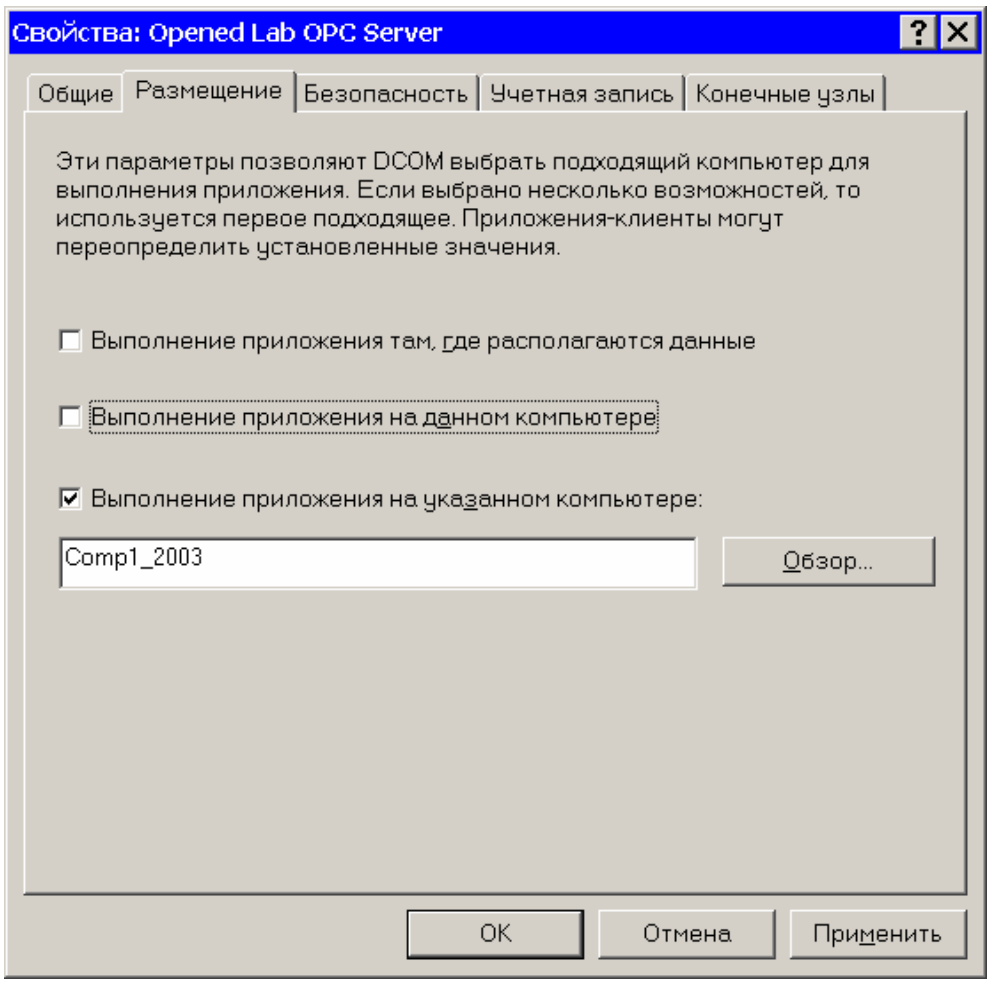

Рис. 5.7. Указание размещения OPC-сервера

5. Для выбранного сервера правой кнопкой мыши вызовите контекстное меню, выполните команду **Свойства**, в появившемся окне **Свойства** выберите вкладку **Расположение** и настройте ее в соответствии с рис. 5.9. Здесь, в качестве примера, **Comp1_2003** является именем компьютера, на котором работает OPC-сервер. После настройки дважды нажмите кнопку **ОК**.
6. Сконфигурируйте DCOM. С этой целью выполните команду **Пуск | Выполнить...**, в появившемся диалоговом окне (см. приведенный ранее рис. 5.5) введите **dcomcnfg** и нажмите кнопку **OK**.

7. В появившемся диалоговом окне **Службы компонентов** откройте последовательно кнопки **Службы компонентов**, **Компьютеры**, **Мой компьютер**, **Настройка DCOM** и выберите OPC-сервер (рис. 5.8, в нашем примере это **Opened Lab OPC Server**).

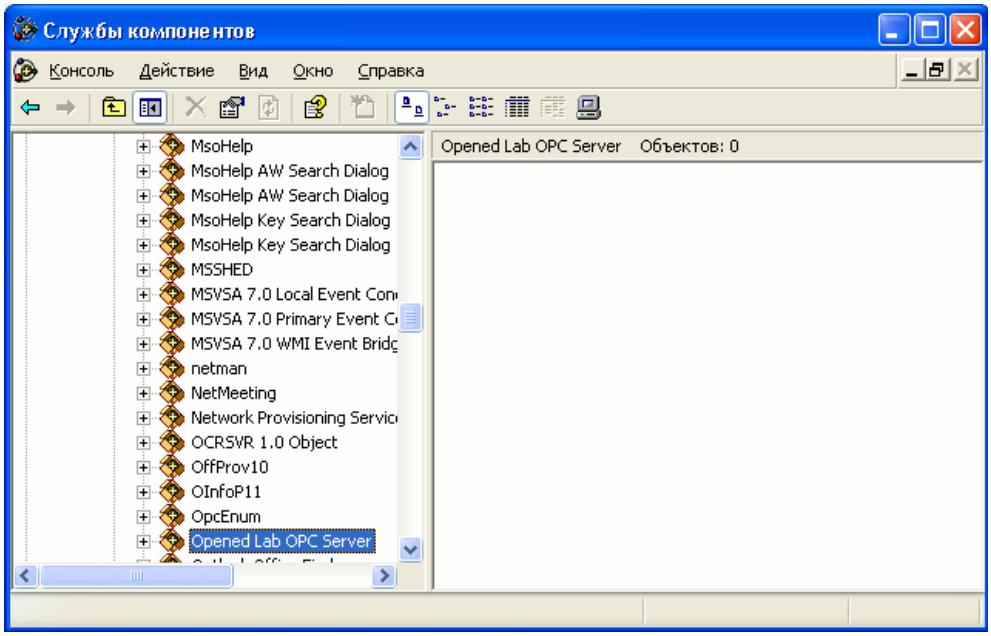

Рис. 5.8. Выбор OPC-сервера

8. Установите низкий уровень безопасности, для чего выполните команду **Пуск | Настройка | Панель управления | Брандмауэр Windows | Вкладка Общие** и выберите радиокнопку **Выключить**.

5.4. Занятие 12 "Информационный обмен между SCADA-системой и промышленным контроллером"

Цель занятия состоит в демонстрации средств, предоставляемых GeniDAQ, для ввода-вывода аналоговой и дискретной информации в реальном масштабе времени и ее отображения. В отличие от занятия 1 вместо программного эмулятора сигналов **Advantech I/O** используется OPC-сервер (**OpenLabOPCSvr.AiVT.2**), блоки аналогового и дискретного ввода-вывода и различные индикаторы.

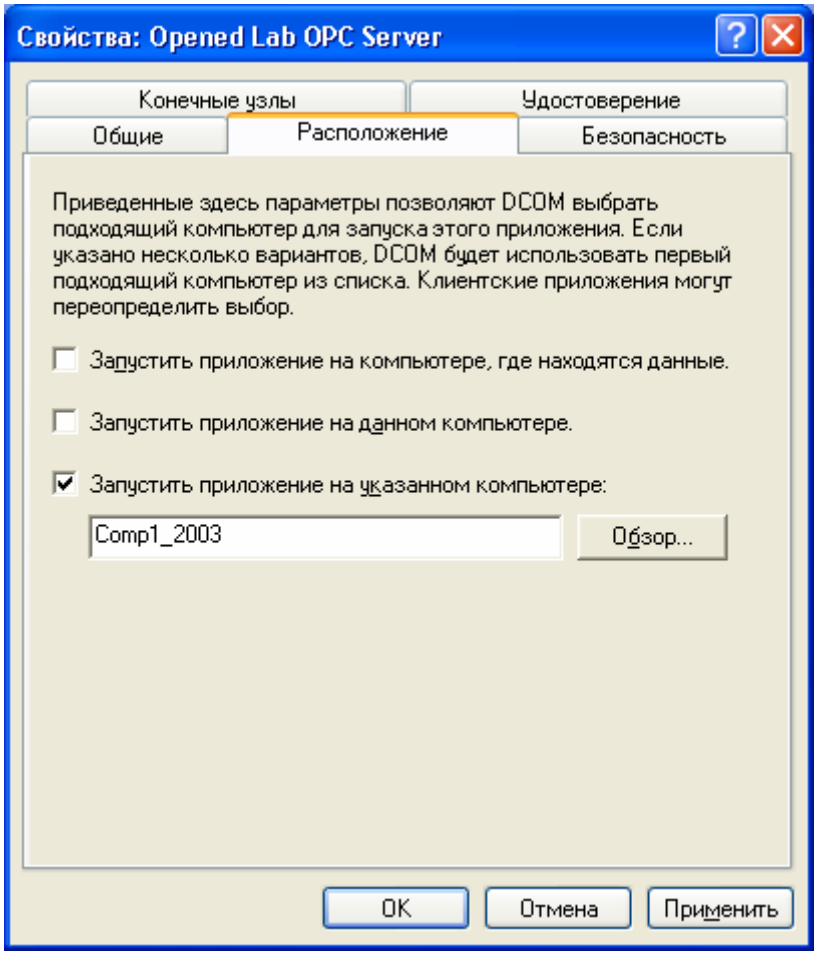

Рис. 5.9. Указание размещения OPC-сервера

5.4.1. Используемый инструментарий

Новыми инструментами, используемыми на данном занятии, являются блок аналогового вывода (редактор задач); блок дискретного ввода (редактор задач); блок дискретного вывода (редактор задач); OPC-сервер (рассмотрен ранее в начале главы); промышленный контроллер типа 7188E2 фирмы ICP DAS, содержащий УСО (рассмотрен ранее в начале главы); аналоговые вычислительные комплексы АВК-31, используемые для имитации аналоговых сигналов и моделирования объектов управления; имитатор для получения входных дискретных сигналов и приема выходных дискретных сигналов.

Блок аналогового вывода предназначен для передачи информации, получаемой от других функциональных блоков, элементов отображения или других приложений

Windows с использованием технологии COM/DCOM, устройствам, имеющим подсистему вывода аналоговых сигналов. Двойной щелчок левой клавишей мыши на пиктограмме **AO** блока аналогового вывода приводит к появлению диалогового окна для настройки параметров блока (рис. 5.10). Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров блока аналогового вывода будут рассмотрены далее.

Блок дискретного ввода предназначен для приема информации, в том числе с использованием технологии COM/DCOM, от устройств, имеющих подсистему ввода дискретных сигналов, и передачи указанных сигналов другим функциональным блокам и элементам отображения. Двойной щелчок левой клавишей мыши на пиктограмме **DI** блока дискретного ввода приводит к появлению окна диалогового окна для настройки параметров блока (рис. 5.11). Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров блока дискретного ввода будут рассмотрены далее.

Блок дискретного вывода предназначен для передачи информации, получаемой от других функциональных блоков, элементов отображения или других приложений Windows с использованием технологии COM/DCOM, устройствам, имеющим подсистему вывода дискретных сигналов. Двойной щелчок левой клавишей мыши на пиктограмме **DO** блока дискретного вывода приводит к появлению диалогового окна для настройки параметров блока (рис. 5.12). Информацию о параметрах настройки этого окна можно получить с помощью кнопки **Help**. Конкретные настройки параметров блока дискретного вывода будут рассмотрены далее.

Соединитель блоков (проводник). В учебных примерах, рассмотренных ранее, вы уже неоднократно пользовались проводником. Соберем воедино все, что мы знаем об этом блоке. Пиктограмма этого инструмента, предназначенного для установления видимых связей между пиктограммами функциональных блоков задачи, расположена второй слева на панели инструментов редактора задач (см. приведенный ранее рис. 4.1). При выборе данного инструмента активизируется режим установления связей между функциональными блоками редактора задач и курсор мыши принимает вид катушки с нитками. Соединение функциональных блоков в рамках задачи является основным типом организации взаимодействия между блоками. Для соединения двух блоков следует поместить курсор в виде катушки с нитками на пиктограмму блока, данные которого предполагается передать в другой блок, и щелкнуть левой клавишей мыши.

Если блок, являющийся источником данных для второго блока, имеет несколько выходов, то в появившейся диалоговой панели со списком всех выходов блока потребуется выбрать номер соединяемого выхода. Далее необходимо поместить курсор на пиктограмму функционального блока, являющегося приемником данных и произвести щелчок левой клавишей мыши.

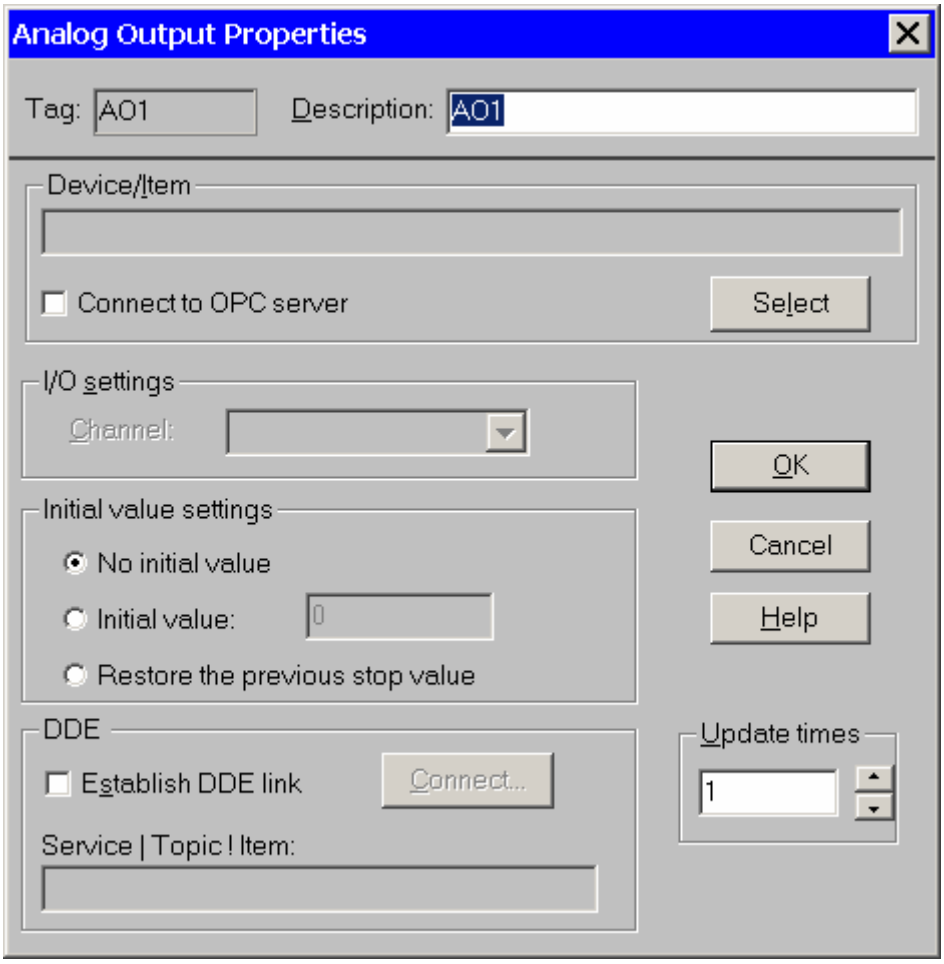

Рис. 5.10. Диалоговое окно настройки блока аналогового вывода

Если блок, являющийся приемником данных, имеет несколько входов, то в появившейся диалоговой панели со списком всех входов блока потребуется выбрать номер соединяемого входа. Если графическое изображение устанавливаемого соединения необходимо выполнить в виде ломаной линии, то для фиксации каждого отрезка ломаной следует производить щелчок левой клавишей мыши в точке, которая должна служить началом очередного отрезка ломаной. В зависимости от того, на каком блоке был произведен первый щелчок левой клавишей мыши при установлении связи, информационный поток через установленную логическую связь будет различным. Например, если требуется установить связь между блоком аналогового ввода и блоком усреднения, следует сначала произвести щелчок левой клавишей мыши на пиктограмме блока аналогового ввода, а затем на пиктограмме блока усреднения.

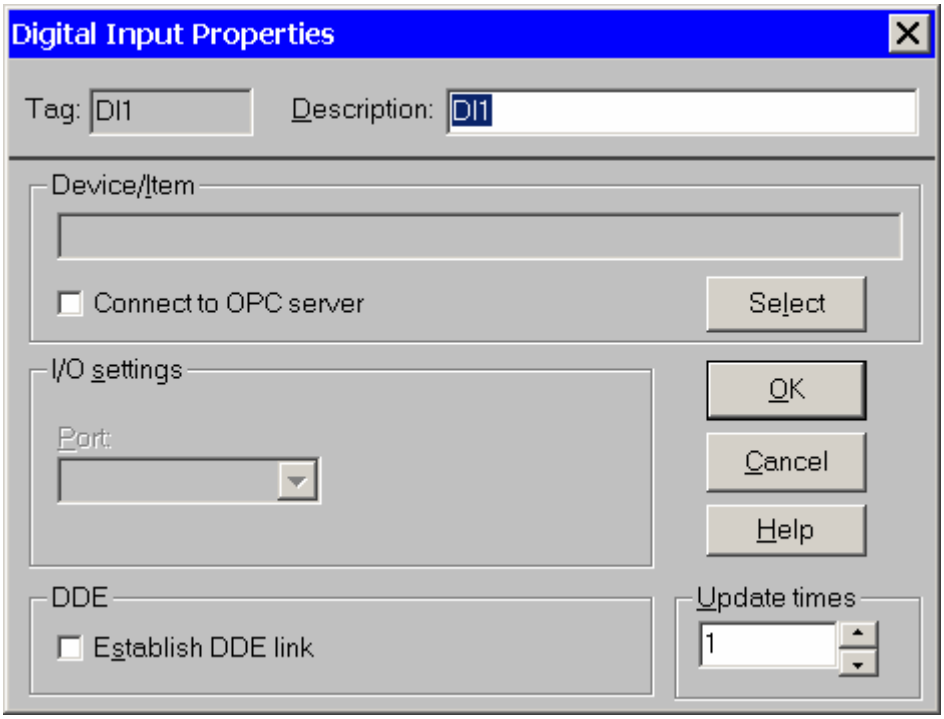

Рис. 5.11. Диалоговое окно настройки блока дискретного ввода

При этом место присоединения проводника к пиктограмме блока усреднения будет отмечено стрелкой, направленной в сторону блока усреднения. Направление стрелки указывает, что значение с выхода функционального блока аналогового ввода поступает на вход блока усреднения. Некоторые блоки допускают передачу данных только в одном направлении. Обратите внимание, что при попытке установления связи между блоком аналогового ввода и блоком ввода данных из файла на экран монитора будет выведено сообщение "Вход недоступен". Блок ввода данных из файла имеет только выходные каналы и не позволяет принимать данные.

Аналоговые вычислительные комплексы АВК-31 используются для имитации аналоговых входов, приема аналоговой информации и моделирования объектов управления. В каждом из АВК по три гнезда на коммутационном поле с маркировкой **Внешние цепи** (1, 2, 3) связаны с входами модулей ввода аналоговых сигналов промышленного контроллера. Аналогичным образом, выходы модулей аналогового выхода промышленного контроллера соединены с гнездами 4 и 6 на коммутационном поле АВК.

Имитатор для получения входных дискретных сигналов и приема выходных дискретных сигналов содержит по восемь тумблеров (X0-X7) и кнопок (X8-X15) для имитации входных дискретных сигналов и восемь светодиодов для индикации выдаваемых дискретных сигналов.

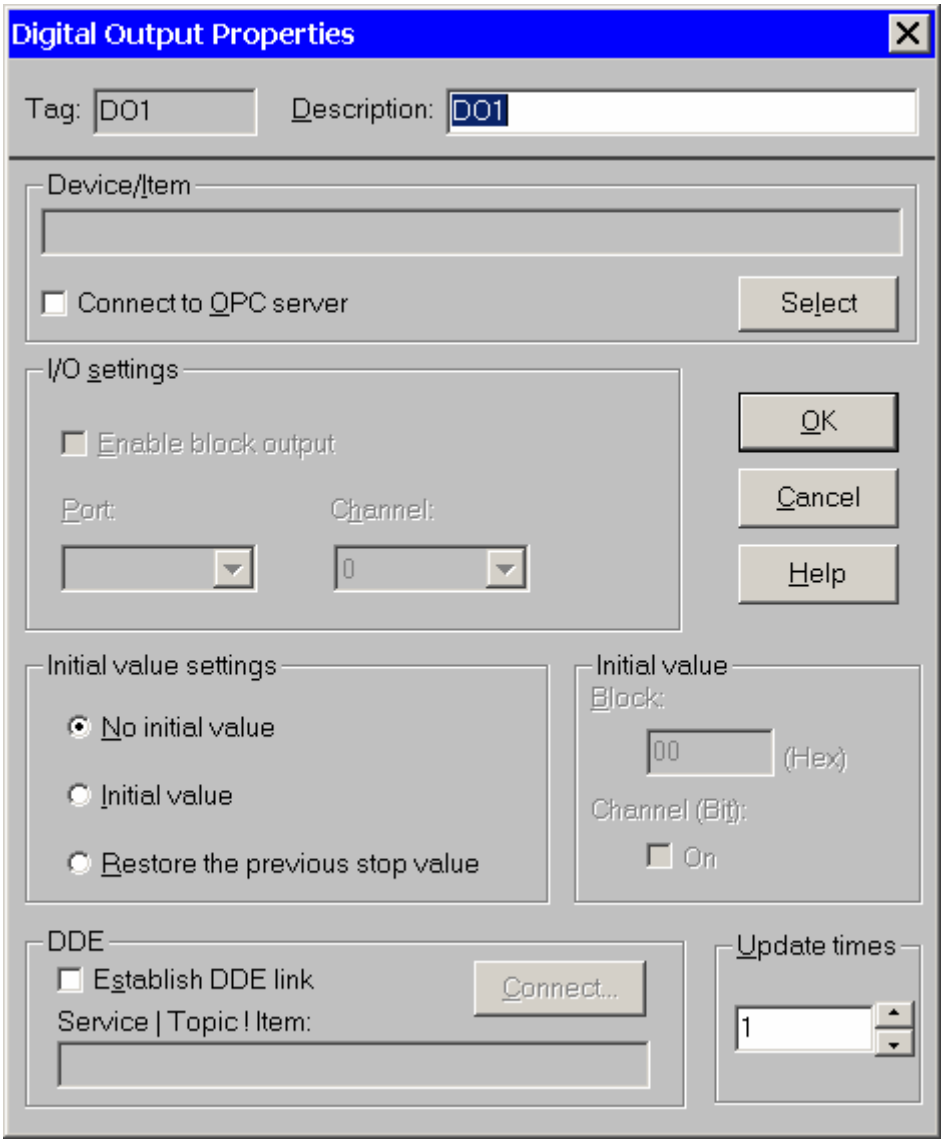

Рис. 5.12. Диалоговое окно настройки блока дискретного вывода

Имитатор для получения входных дискретных сигналов и приема выходных дискретных сигналов содержит по восемь тумблеров (X0-X7) и кнопок (X8-X15) для имитации входных дискретных сигналов и восемь светодиодов для индикации выдаваемых дискретных сигналов.

5.4.2. Проектирование приложения

1. *Первое действие* — включение OPC-сервера и промышленного контроллера – выполняет преподаватель (см. приведенный ранее подразд. 5.2).
2. *Второе действие*. Включите рабочую станцию или перегрузите ее, при регистрации обязательно используйте *те же самые имя и пароль*, которые использовались при регистации на компьютере С запущенным OPC-сервером. Запустите построитель приложений и настройте приложение аналогично рассмотренному ранее. Назовите приложение **Занятие** 12.
3. *Третье действие*. Задайте временные параметры запуска приложения аналогично тому, как это делалось на занятии 1. В нашем примере задан период повторения задачи 100 миллисекунд.
4. *Четвертое действие*. В окне редактора задач на панели инструментов выберите кнопку (пиктограмму) с мнемоникой **AI** и перетащите ее в окно редактора задач. При этом в окне редактора задач появится блок аналогового ввода с позиционным обозначением **AI1** (см. занятие 1, рис. 4.9). Произведите двойной щелчок левой кнопкой мыши над блоком аналогового ввода и появится диалоговое окно его настройки. Введите в этом окне в поле **Description:** значение **AI61** (последние две цифры означают следующее: 6 — номер рабочей станции, возможный диапазон номеров 1…6, а 1 — номер аналогового входа на рабочей станции, возможный диапазон номеров 1…3). Для выбора источника аналогового сигнала — OPC-сервера — следует включить флажок **Connect to OPC Server**. Далее нажмите кнопку **Select**, в появившемся окне диалога **OPC Item Select** выберите сервер **OpenLabOPCSrv.AiVT.2**, нажмите кнопку **Connect…**, откройте **OpenLabOPCSrv.AiVT.2** и **7188E2**, выберите **7017.2.7** и нажмите кнопку **OK** (рис. 5.13). Тем самым, для подключения аналогового сигнала было выбрано гнездо 1 **Внешние цепи** аналогового вычислительного комплекса (АВК) рабочей станции №6 (см. приведенный ранее рис. 5.3). Диалоговое окно конфигурирования блока аналогового ввода приобретает вид, представленный нп рис. 5.14. Для сохранения значений параметров блока аналогового ввода нажмите кнопку **OK**.
5. *Пятое действие*. Аналогичным образом поместите в окно редактора задач блок аналогового вывода. Для этого на панели инструментов выберите кнопку с мнемоникой **AO** и перетащите ее в окно редактора задач. При этом в окне редактора задач появится блок аналогового вывода с позиционным обозначением **AO1**. Произведите двойной щелчок левой кнопкой мыши над блоком вывода аналоговой информации и появится диалоговое окно его настройки. Введите в этом окне **Description:** значение **A066** (последние две цифры означают следующее: 6 — номер рабочей станции, возможный диапазон номеров 1…6, а 6 — номер аналогового выхода на рабочей станции, возможный диапазон номеров 4 и 6). Для выбора источника аналогового сигнала — OPC-сервера — следует включить флажок **Connect to OPC Server**. Далее нажмите кнопку **Select**, в появившемся окне диалога **OPC Item Select** выберите сервер **OpenLabOPCSrv.AiVT.2**, нажмите кнопку **Connect…**, откройте

OpenLabOPCSrv.AiVT.2 и **7188E2**, выберите **7024.4.1** и нажмите кнопку **OK** (рис. 5.15). Тем самым, для подключения аналогового сигнала было выбрано гнездо 6 **Внешние цепи** аналогового вычислительного комплекса (АВК) рабочей станции №6 (см. приведенный ранее рис. 5.3). Диалоговое окно конфигурирования блока аналогового ввода приобретает вид, представленный нп рис. 5.16. Для сохранения значений параметров блока аналогового ввода нажмите кнопку **OK**.

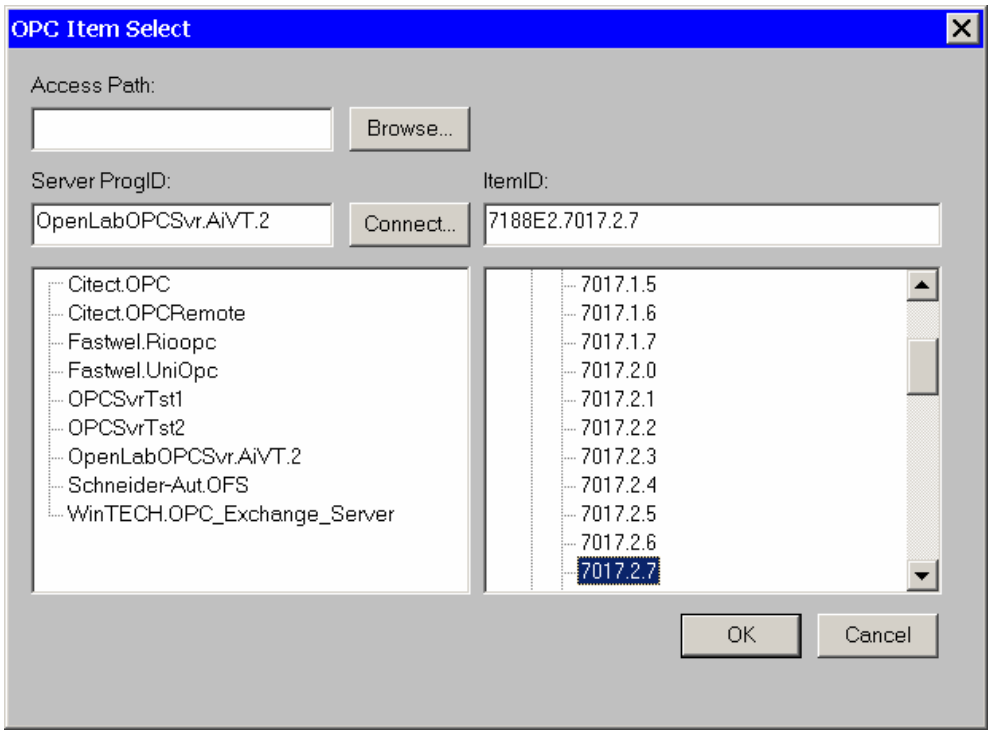

Рис. 5.13. Диалоговое окно выбора параметров OPC-сервера

6. *Шестое действие*. Теперь выполните соединение выхода блока аналогового ввода с входом блока аналогового вывода. Для этого выберите кнопку **Connection wire** на панели инструментов и поместите "нить" на блок **AI61** в окне редактора задач. В появившемся диалоговом окне выберите выходной канал **Output 0** (при работе с OPC-сервером правильным всегда будет только такой выбор), нажмите кнопку **OK** и положите "нить" на блок **AO66** в окне редактора задач. В результате выполненных действий информация будет передаваться с выхода блока аналогового ввода на вход блока аналогового вывода (рис. 5.17).

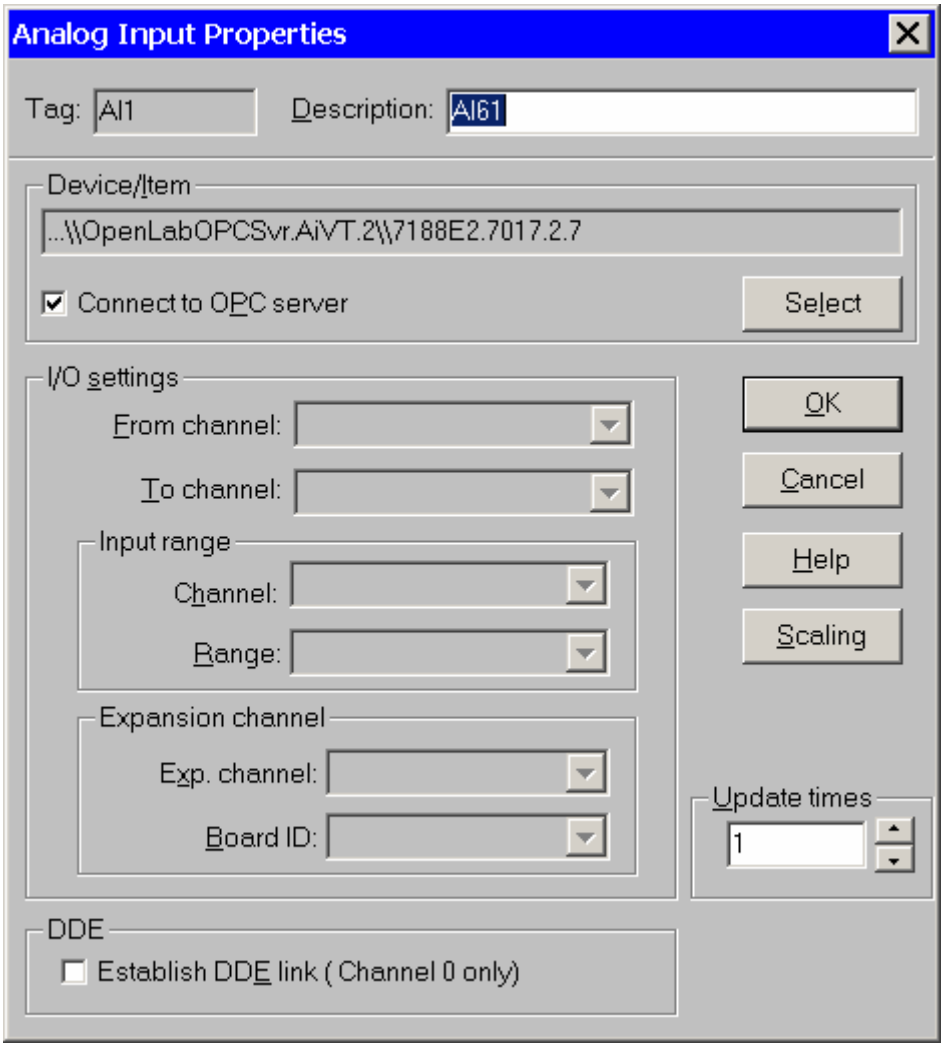

Рис. 5.14. Диалоговое окно конфигурирования блока аналогового ввода

7. *Седьмое действие*. Поместите в окно редактора задач блок дискретного ввода. Для этого на панели инструментов выберите кнопку с мнемоникой **DI** и перетащите ее в окно редактора задач. При этом в окне редактора задач появится блок аналогового вывода с позиционным обозначением **DI1**. Произведите двойной щелчок левой кнопкой мыши над этим блоком и аналогичным образом настройте его в соответствии с рис. 5.18. В позиционном обозначении **DI_X0** два последних символа являются обозначением тумблера — источника дискретного сигнала. На приведенном ранее рис. 5.3 приведены обозначения тегов в OPC-сервере, соответствующих имитаторам входных дискретных сигналов.

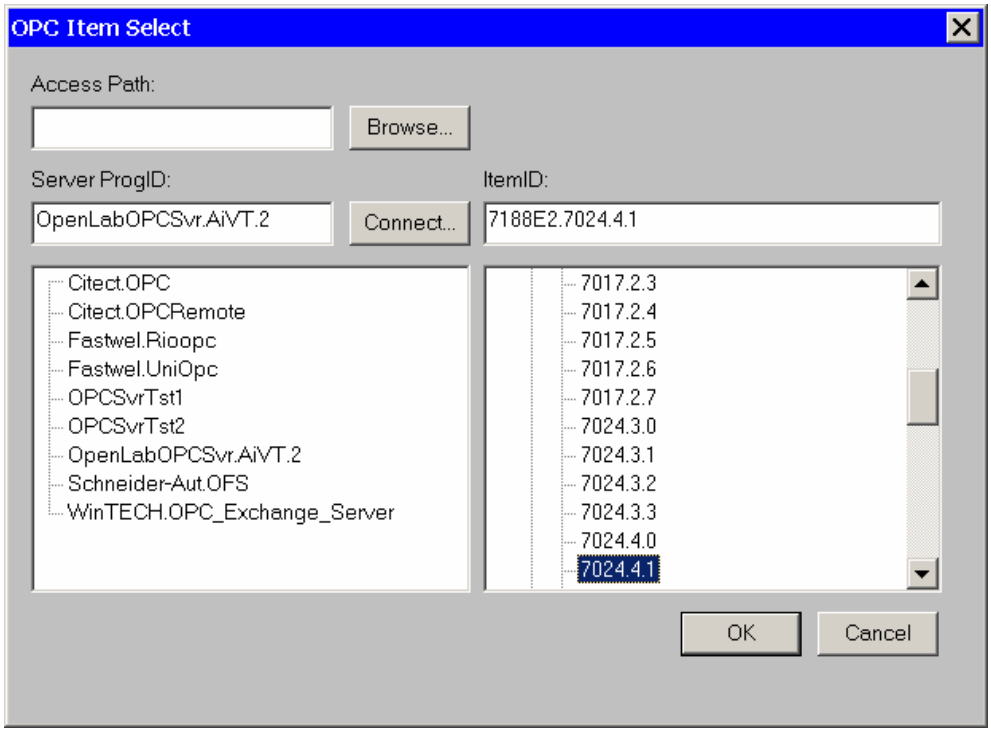

Рис. 5.15. Диалоговое окно выбора параметров OPC-сервера

8. *Восьмое действие*. Поместите в окно редактора задач блок дискретного вывода. Для этого на панели инструментов выберите кнопку с мнемоникой **DO** и перетащите ее в окно редактора задач. При этом в окне редактора задач появится блок аналогового вывода с позиционным обозначением **DO1**. Произведите двойной щелчок левой кнопкой мыши над этим блоком и аналогичным образом настройте его в соответствии с рис. 5.19. В позиционном обозначении **DO_Y0** два последних символа являются обозначением светодиода — приемника дискретного сигнала. На приведенном ранее рис. 5.3 приведены обозначения тегов в OPC-сервере, соответствующих имитаторам выходных дискретных сигналов.
9. *Девятое действие*. Аналогично указанному ранее выполните соединение выхода блока дискретного ввода с входом блока дискретного вывода таким образом, чтобы информация передавалась с блока дискретного ввода на блок дискретного вывода. Таким образом, с помощью указанных блоков дискретного ввода и дискретного вывода будет обеспечена передача информации от тумблера на светодиод.
10. *Десятое действие*. Поместите в окно редактора задач блок дискретного ввода. Для этого на панели инструментов выберите кнопку с мнемоникой **DI** и перетащите ее в окно редактора задач. При этом в окне редактора задач появится

блок аналогового вывода с позиционным обозначением **DI2**. Произведите двойной щелчок левой кнопкой мыши над этим блоком и аналогичным образом настройте его в соответствии с рис. 5.20. В позиционном обозначении **СчетчикDI_X8** два последних символа являются обозначением кнопки — источника дискретного сигнала, для которой ведется подсчет количества нажатий. На приведенном ранее рис. 5.3 приведены обозначения тегов в OPC-сервере, соответствующих имитаторам входных дискретных сигналов.

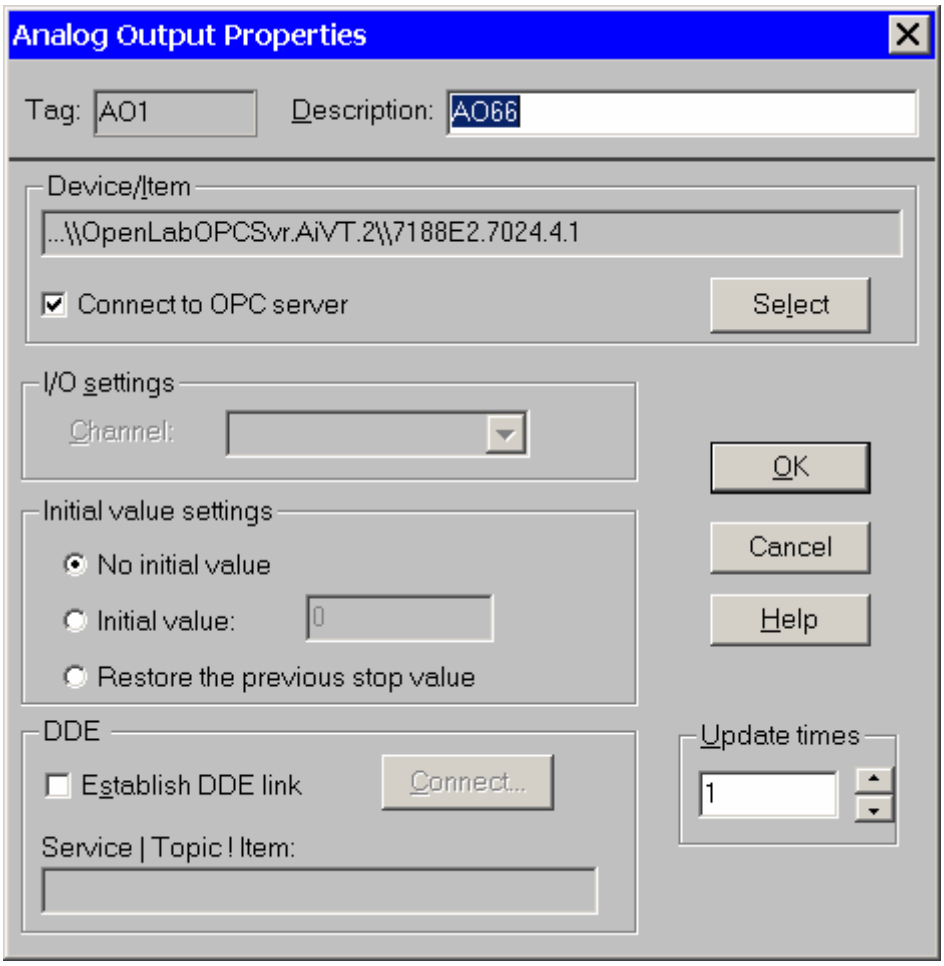

Рис. 5.16. Диалоговое окно конфигурирования блока аналогового вывода

11. *Одиннадцатое действие.* В результате выполненных действий окно редактора задач примет вид, указанный на рис. 5.21.
12. *Двенадцатое действие.* Сделайте активным окно **DISP1** редактора форм отображения. На панели инструментов выберите кнопку **RealTime Trend Graph**

(временной график) и с помощью мыши перетащите ее и положите во внутреннюю область окна редактора форм отображения и, не отпуская левой кнопки мыши, установите нужный размер графика. Для настройки свойств временного графика щелкните дважды левой кнопкой мыши над пиктограммой временного графика. В результате появится диалоговое окно настройки. Для привязки графика к источнику информации в этом окне нажмите кнопку **ADD** и в появившемся окне выполните необходимую настройку для индикации выходного сигнала блока аналогового вывода (рис. 5.22). Затем с помощью кнопки **OK** закройте окно.

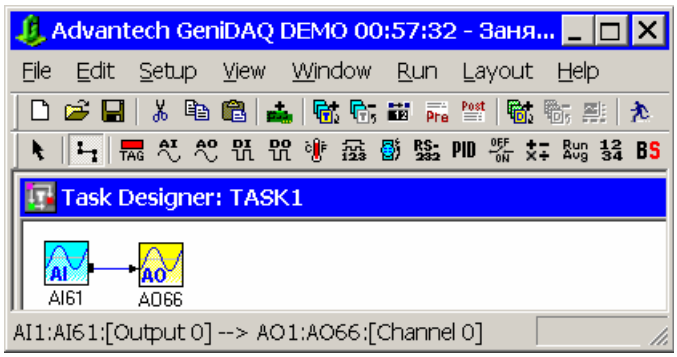

Рис. 5.17. Окно редактора задач

Для отображения содержимого счетчика нажатий кнопки **X8** поместите в окно редактора форм отображения блок цифровой индикации (**Numeric String**) и настройте его в соответствии с рис. 5.23. Снабдите элементы отображения соответствующими надписями (рис. 5.24).

13. *Тринадцатое действие*. Сохраните созданный проект аналогично тому, как это было сделано на занятии 1. После сохранения проекта имеется возможность немедленного запуска созданного проекта с помощью кнопки **Start** на панели инструментов (рис. 5.25). Остановить работу проекта можно с помощью кнопки **Stop** на панели инструментов. В процессе выполнения стратегии рекомендуем выполнить следующие эксперименты. На АВК № 6 подавать на гнездо 1 **Внешние цепи** различные напряжения с выхода источника эталонных напряжений (входной аналоговый сигнал) и измерять напряжение на гнезде 6 (выходной аналоговый сигнал) — эти напряжения должны совпадать. Выполнять переключение тумблера **X0** на имитаторе дискретных сигналов и наблюдать за состоянием светодиода **Y0**. Выполнять нажатие кнопки **X8** на имитаторе дискретных сигналов и наблюдать за состоянием цифрового индикатора.
14. *Пятнадцатое действие*. Завершите работу приложения. Для этого выполните команду **File | Exit**.
15. *Шестнадцатое действие — выполняет преподаватель*. На компьютере, где работает OPC-сервер, командой **File | Exit** завершите его работу и выключите

компьютер. Выключите питание промышленного контроллера. Выключите включенные АВК.

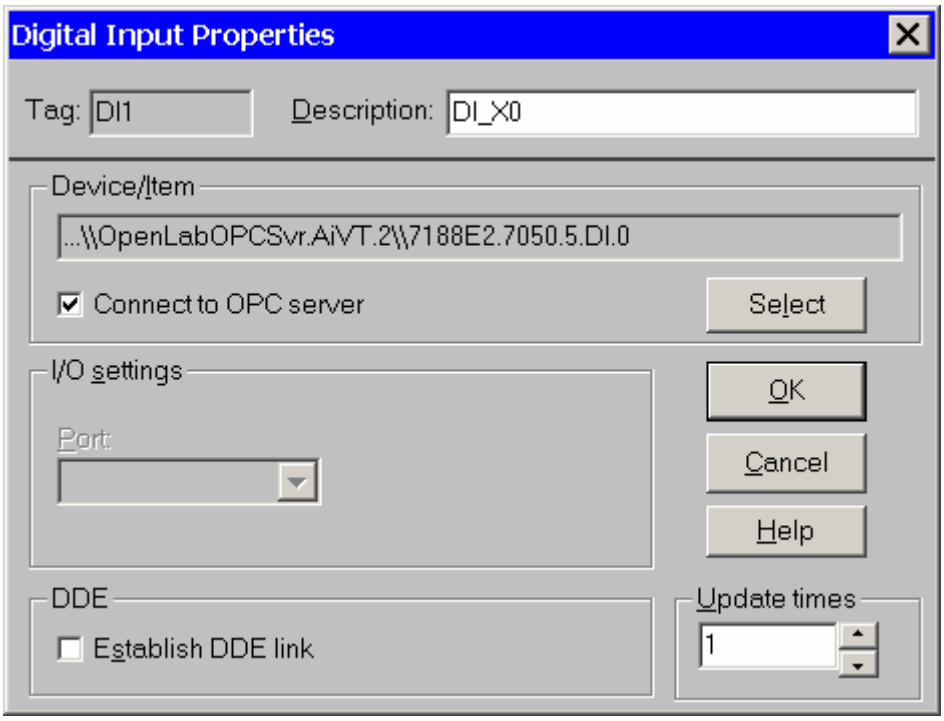

Рис. 5.18. Диалоговое окно конфигурирования блока дискретного ввода

5.4.3. Упражнения

Совет

Обязательно выполните приводимые далее упражнения — это очень важно для практического освоения рассмотренного материала.

Упражнение 4.38. Повтор создания и тестирования приложения "Занятие 12"

Повторите создание и тестирование рассмотренного ранее приложения **Занятие 12**. Для сокращения затрат учебного времени упражнение выполняйте параллельно с преподавателем.

Упражнение 4.39

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование дискретных входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS. В окне

редактора форм отображения снабдить элементы отображения поясняющими надписями.

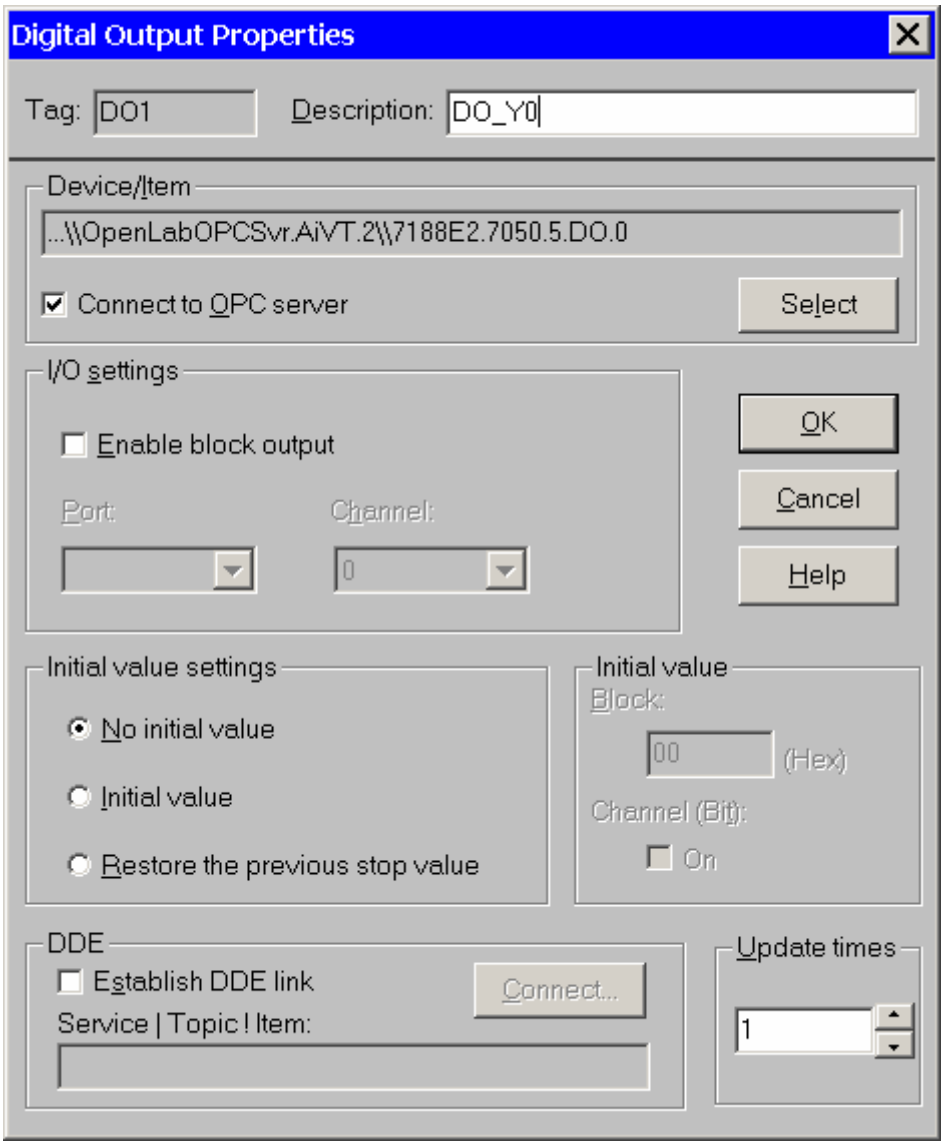

Рис. 5.19. Диалоговое окно конфигурирования блока дискретного вывода

Упражнение 4.40

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование аналоговых

входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS для АВК-1. В окне редактора форм отображения снабдить элементы отображения поясняющими надписями.

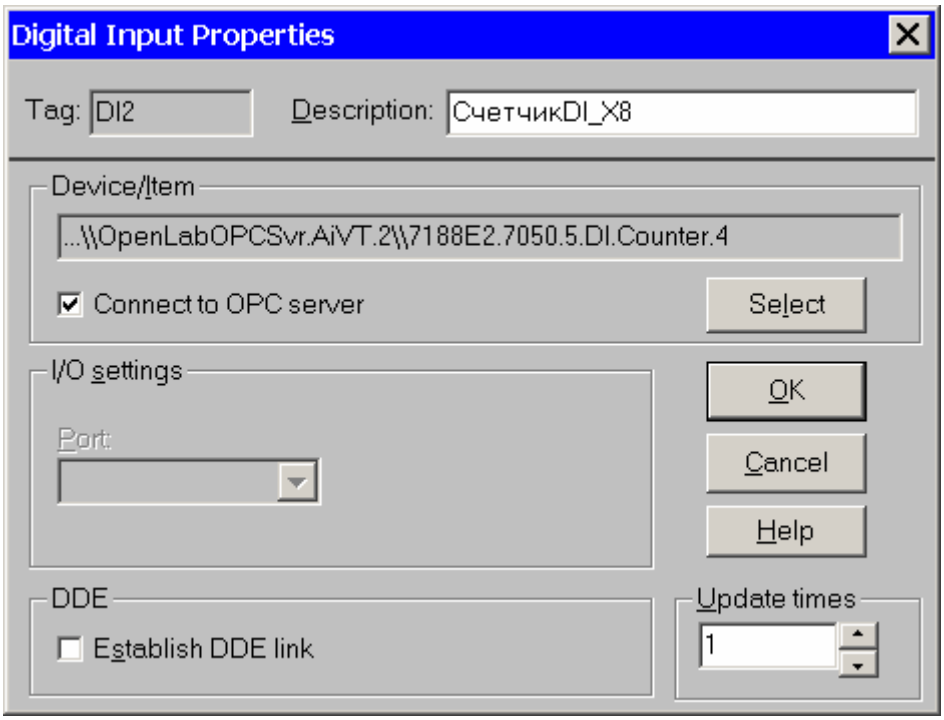

Рис. 5.20. Диалоговое окно конфигурирования блока дискретного ввода

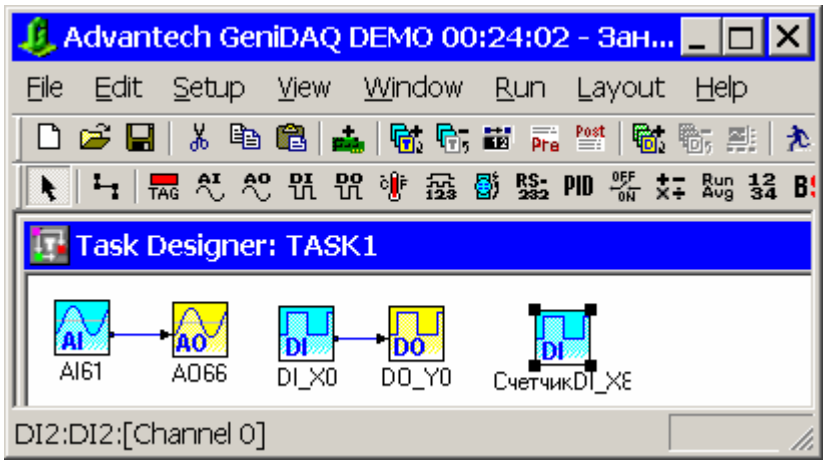

Рис. 5.21. Окно редактора задач

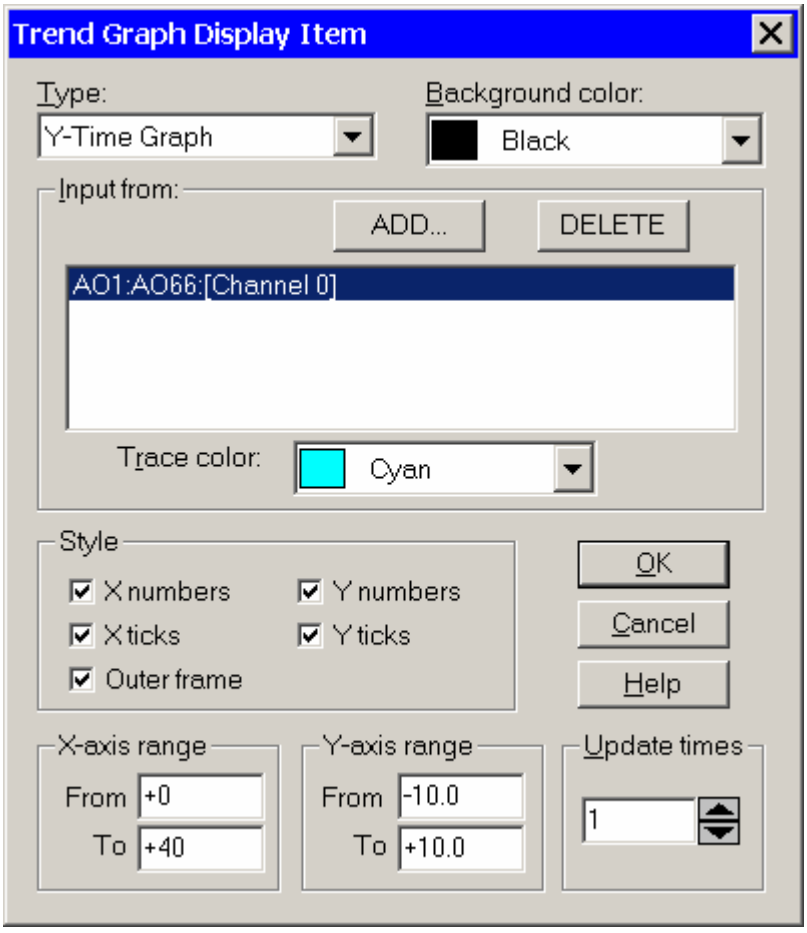

Рис. 5.22. Настройка графика времени

Упражнение 4.41

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование аналоговых входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS для АВК-2. В окне редактора форм отображения снабдить элементы отображения поясняющими надписями.

Упражнение 4.42

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование аналоговых входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS для АВК-3. В

окне редактора форм отображения снабдить элементы отображения поясняющими надписями.

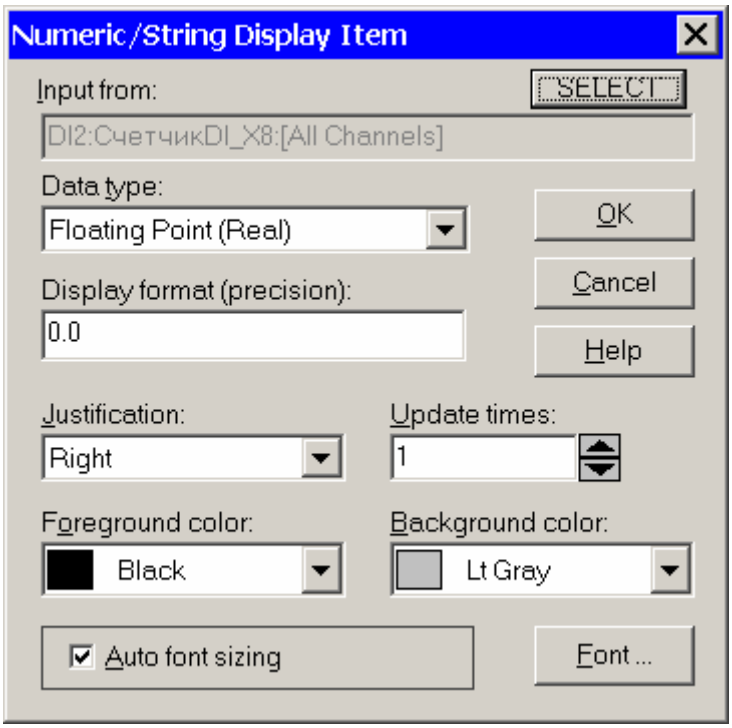

Рис. 5.23. Настройка блока цифровой индикации

Упражнение 4.43

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование аналоговых входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS для АВК-4. В окне редактора форм отображения снабдить элементы отображения поясняющими надписями.

Упражнение 4.44

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование аналоговых входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS для АВК-5. В окне редактора форм отображения снабдить элементы отображения поясняющими надписями.

Упражнение 4.45

Разработать стратегию, обладающую следующими возможностями. Задать период запуска задачи 2 секунды. Стратегия должна обеспечивать тестирование аналоговых входов-выходов промышленного контроллера **7188E2** фирмы ICP DAS для АВК-6. В окне редактора форм отображения снабдить элементы отображения поясняющими надписями.

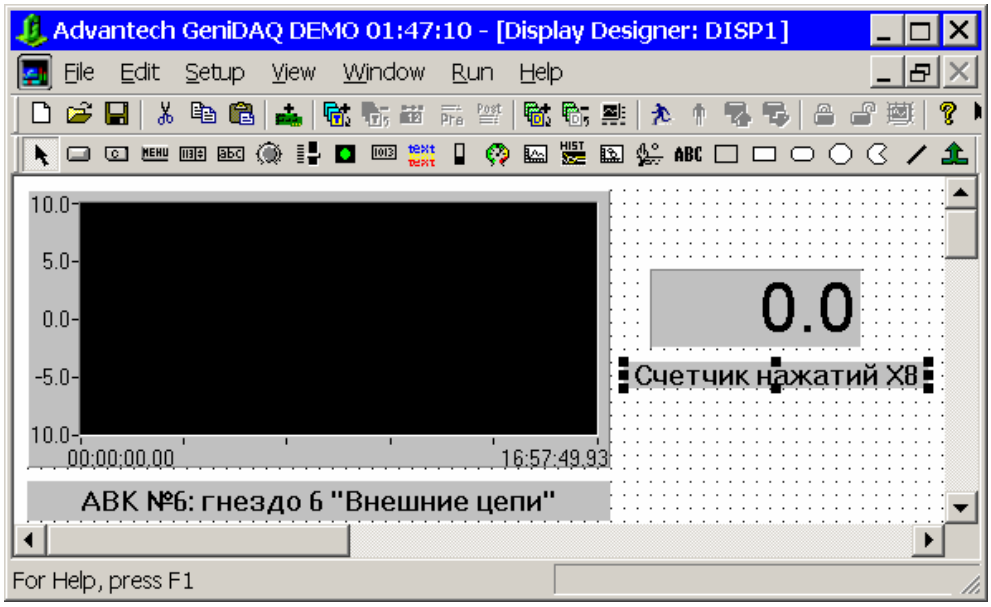

Рис. 5.24. Вид окна редактора форм отображения

Совет

Для лучшего освоения изучаемого материала советуем вам дополнить указанные ранее материалы курсовым проектированием. Варианты заданий для курсового проектирования приводятся в приложении 1.

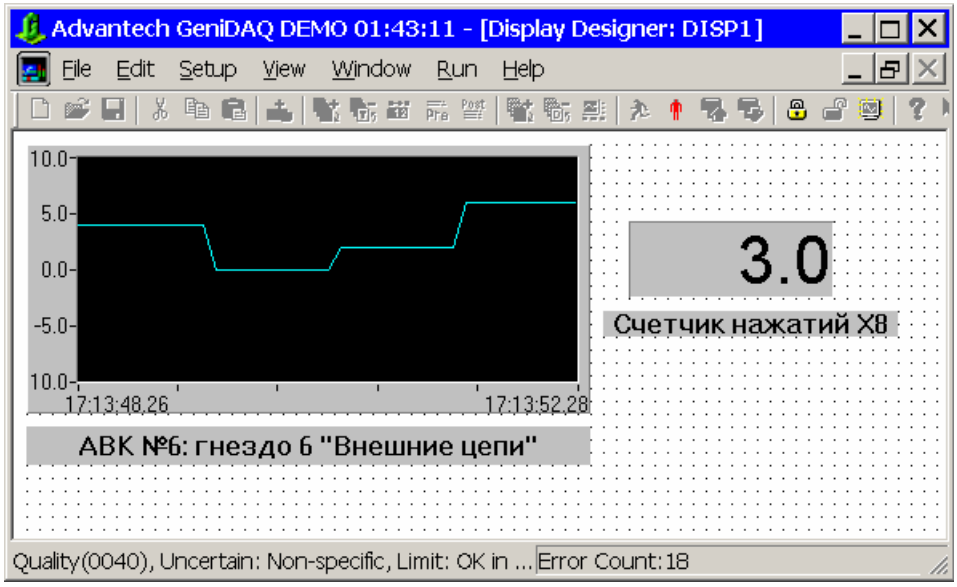

Рис. 5.25. Вид окна редактора форм отображения

Приложение 1. Курсовое проектирование. Варианты заданий

В приводимых ниже вариантах заданий в зависимости от объема выполненной работы, предусмотрено три уровня выполнения:

- уровень 1 (соответствует оценке "удовлетворительно");
- уровень 2 (соответствует оценке "хорошо");
- уровень 3 (соответствует оценке "отлично").

Для уровней 2 и 3 предусмотреть в режиме выполнения два подрежима — задания параметров и рабочий режим. После запуска проекта начальным подрежимом д. б. подрежим задания параметров.

В процессе выполнения курсового проекта надлежит разработать и документировать проект, реализующий решение заданной задачи. Результат представить в электронном виде аналогично оформлению описания занятий (уроков) в первой части учебного пособия.

Вариант П1.1

Проект для реализации алгоритма трехпозиционного управления. В качестве объекта управления использовать математическую модель инерционного объекта первого порядка с настраиваемыми начальными условиями и временем переходного процесса.

- Уровень 1 — параметры алгоритма фиксированы.
- Уровень 2 — параметры алгоритма задаются в режиме выполнения, величина управляющих воздействий постоянна.
- Уровень 1 — то же, что и в уровне 2, но предусматривается два режима — работа с постоянными величинами управляющих воздействий и формирование управлений по пропорционально-интегрально-дифференциальному закону (ПИД-закону).

Вариант П1.2

Проект для генерации гармонического (амплитуда, период, смещение, ограничения максимальной и минимальной величин) процесса.

- Уровень 1 — параметры алгоритма фиксированы.
- Уровень 2 — параметры процесса задаются в режиме выполнения.
- Уровень 1 — дополнительно предусматривается масштабирование по времени и уровню.

Вариант П1.3

Проект для генерации пилообразного (амплитуда, период, смещение, ограничения максимальной и минимальной величин, скважность) процесса.

- Уровень 1 — параметры алгоритма фиксированы.
- Уровень 2 — параметры процесса задаются в режиме выполнения.
- Уровень 1 — дополнительно предусматривается масштабирование по времени и уровню.

Вариант П1.4

Проект для генерации прямоугольного сигнала (амплитуда, период, смещение, ограничения максимальной и минимальной величин, скважность).

- Уровень 1 — параметры алгоритма фиксированы.
- Уровень 2 — параметры процесса задаются в режиме выполнения.
- Уровень 1 — дополнительно предусматривается масштабирование по времени и уровню.

Вариант П1.5

Проект для ПИД-регулятора и его частных разновидностей (оригинальный, разработанный и из имеющихся в SCADA-системе). Модель объекта математическая (первый порядок, предусмотреть задание времени переходного процесса и начальных условий).

- Уровень 1 — параметры алгоритма фиксированы.
- Уровень 2 — параметры процесса задаются в режиме выполнения.
- Уровень 3 — дополнительно предусматривается использование ПИД-регулятора, встроенного в SCADA-систему.

Вариант П1.6

Проект для экстремальной системы, использующей для поиска экстремума по двум координатам метод Гаусса-Зейделя. Поверхность поиска и возмущающее воздействие моделируются математически.

- Уровень 1 — параметры алгоритма, поверхности поиска и возмущения фиксированы.
- Уровень 2 — указанные параметры задаются в режиме выполнения.
- Уровень 3 — обеспечивается оптимизация параметров алгоритма поиска.

Вариант П1.7

Проект для экстремальной системы, использующей для поиска экстремума по двум координатам метод наискорейшего спуска. Поверхность поиска и возмущающее воздействие моделируются математически.

- Уровень 1 — параметры алгоритма, поверхности поиска и возмущения фиксированы.
- Уровень 2 — указанные параметры задаются в режиме выполнения.
- Уровень 3 — обеспечивается оптимизация параметров алгоритма поиска.

Вариант П1.8

Проект для экстремальной системы, использующей для поиска экстремума по двум координатам градиентный метод. Поверхность поиска и возмущающее воздействие моделируются математически.

- Уровень 1 — параметры алгоритма, поверхности поиска и возмущения фиксированы.
- Уровень 2 — указанные параметры задаются в режиме выполнения.
- Уровень 3 — обеспечивается оптимизация параметров алгоритма поиска.

Вариант П1.9

Проект для фильтрации методом экспоненциального сглаживания и путем вычисления скользящего среднего.

- Уровень 1 — параметры фильтров фиксированы.
- Уровень 2 — указанные параметры задаются в режиме выполнения.
- Уровень 3 — дополнительно предусматриваются в режиме выполнения по кнопке выбор вида фильтрации.

Вариант П1.10

Проект для поддержания заданного уровня жидкости в резервуаре. Резервуар имеет три трубы. По одной из них жидкость подается в резервуар (кран на этой трубе управляется автоматически), а по другой — вытекает из резервуара (кран на этой трубе управляется пользователем). Третья труба является дренажной — она должна открываться только при двух других закрытых трубах и уменьшении уставки.

- Уровень 1 — проект с фиксированными настройками, краны двухпозиционные (полностью открыты или полностью закрыты).
- Уровень 2 — краны на трубах могут занимать любое положение от полностью открытого до полностью закрытого. Задание требуемого уровня в резервуаре и

управление краном на выводящей трубе должны производиться в режиме выполнения.

- Уровень 3 — дополнительно предусматриваются зона нечуствительности и ПИД-закон управления.

Вариант П1.11

Проект для поддержания заданного уровня жидкости в резервуаре. Резервуар имеет четыре трубы. По одной из них жидкость подается в резервуар (кран на этой трубе управляется автоматически), а по двум другим — вытекает из резервуара (краны на этих трубах управляется пользователем). Четвертая труба является дренажной — она должна открываться только при остальных закрытых трубах и уменьшении уставки.

- Уровень 1 — проект с фиксированными настройками, краны двухпозиционные (полностью открыты или полностью закрыты).
- Уровень 2 — краны на трубах могут занимать любое положение от полностью открытого до полностью закрытого. Задание требуемого уровня в резервуаре и управление краном на выводящей трубе должны производиться в режиме выполнения.
- Уровень 3 — дополнительно предусматриваются зона нечуствительности и ПИД-закон управления.

Вариант П1.12

Проект для поддержания заданного уровня смеси жидкостей и процентного состава компонентов в миксере. Резервуар имеет четыре трубы. По двум трубам жидкости-компоненты подаются в резервуар (краны на этих трубах управляется автоматически). По третьей трубе "коктейль" вытекает из миксера (кран на этой трубе управляется оператором). Четвертая труба является дренажной — она должна открываться при остальных закрытых трубах и уменьшении уставки. При изменении процентного состава компонентов предварительно содержимое резервуара должно сливаться через дренажную трубу.

- Уровень 1 — проект с фиксированными настройками, краны двухпозиционные (полностью открыты или полностью закрыты).
- Уровень 2 — краны на трубах могут занимать любое положение от полностью открытого до полностью закрытого. Задание требуемого уровня в резервуаре и управление краном на выводящей трубе должны производиться в режиме выполнения.
- Уровень 3 — дополнительно предусматриваются зона нечуствительности и ПИД-закон управления.

Вариант П1.13

Проект для генерации процесса, заданного таблично — в файле на магнитном диске с ограничением максимальной и минимальной величин.

- Уровень 1 — параметры процесса фиксированы.
- Уровень 2 — параметры процессов задаются в режиме выполнения, производится контроль корректности файла.
- Уровень 3 — дополнительно предусматривается масштабирование по времени и уровню.

Литература

1. Чыонг Д.Т. Взаимодействие открытых систем промышленной автоматизации — состояние и проблемы // Информационно-управляющие системы, 2003. №2-3.
2. Системы диспетчерского управления и сбора данных (SCADA-системы) // Мир компьютерной автоматизации, 1999. №1.
3. Куцевич Н.А. SCADA-системы. Взгляд со стороны // Промышленные контроллеры и АСУ, 1999. №1.
4. Куцевич Н.А. SCADA-системы, или муки выбора // Мир компьютерной автоматизации, 1999. №1.
5. Genie 3.0. Руководство пользователя, Prosoft Ltd., 1998.
6. Куцевич Н.А. Citect - новая SCADA-система на российском рынке и новые возможности // Промышленные контроллеры и АСУ, 2000. №1.
7. Калядин А.Ю. Scada-система Citect — что внутри? // PCWeek, 1999. №48.
8. SCADA-продукты на российском рынке // Мир компьютерной автоматизации, 1999. №1.
9. Давыдов В.Г., Чыонг Д.Т. OPC-серверы с открытой архитектурой — средства взаимодействия компонентов в промышленной автоматизации // Автоматизация в промышленности, 2003. №7.
10. Теркель Д. OLE for Process Control — свобода выбора // Современные технологии автоматизации, 1999. №3.
11. Кузнецов А. SCADA-системы: программистом можешь ты не быть // Современные технологии автоматизации, 1996. №1.
12. Advantech GeniDAQ User’s Manual. Advantech Co., 2000.
13. Давыдов В.Г., Чыонг Д.Т. Проектирование компьютерных систем управления на основе SCADA-систем: Учеб. пособие. СПб.: Изд-во СПбГПУ, 2004. 152 с. (современные компьютеризованные образовательные технологии). ISBN 5-7422-0573-2.
14. Гарнаев А.Ю. Самоучитель VBA. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2004. — Ил.
15. Карпов Б. VBA: специальный справочник. — СПб: Питер, 2002. — 416 с.: ил.