В.Л. СОСОНКИН, Г.М. МАРТИНОВ,
доктора техн. наук, профессора,
М.М. ПЕРЕПЕЛКИНА, аспирант

Концепция управления электроавтоматикой станков с ЧПУ по типу виртуальных контроллеров SoftPLC

На очередном витке эволюции программируемых контроллеров появилась и получила заслуженную популярность идея их программной реализации (*SoftPLC*). Наибольший эффект подобная идея дает в системах ЧПУ, где ПО виртуального контроллера *SoftPLC* работает в одной операционной среде с ПО систем ЧПУ. В этой связи возникает необходимость построения хорошо организованного и обозримого математического обеспечения виртуального контроллера на основе объектно-ориентированного подхода. В статье рассмотрены те проблемы построения ядра виртуального контроллера, которые не нашли своего отражения в известной литературе.

At the just another step of PLC evolution, there appeared and became popular the idea of virtual controllers SoftPLC; this idea meant PLC program realization. It is mostly effective in Numerical Control, where PLC works in the same operating-system environment, as NC software. Thus, it is necessary to implement well-organized and observable software of virtual controllers, based on the object-oriented approach. Here there are analyzed those problems of developing the virtual-controller kernel, which haven't been discussed in well-known literature.

Введение

Сегодня появляется реальная возможность программной реализации управления электроавтоматикой станков в рамках общего ПО систем ЧПУ без привлечения дополнительной аппаратуры и системного ПО программируемых контроллеров, которые являются неотъемлемой частью практически любой современной системы ЧПУ (далее предполагаются системы ЧПУ, построенные на базе ПК [1]). Подобные программные системы управления электроавтоматикой получили наименование виртуальных контроллеров *SoftPLC*. Указанный подход позволяет снизить стоимость системы управления при одновременном получении ряда преимуществ. В их числе: упрощение общего ПО, уменьшение ошибок системного программирования, возможность отладки управляющих программ электроавтоматики в рамках самой системы ЧПУ, гибкость конфигурирования электроавтоматики, возможность использования различных коммерческих библиотек.

Далее предлагается объектно-ориентированный подход для построения виртуальных контроллеров электроавтоматики применительно к станкам с системами ЧПУ типа *PCNC* (*Personal Computer Numerical Control*).

Объектно-ориентированный подход при организации математического обеспечения виртуальных контроллеров

В основе технологии создания ПО электроавтоматики лежат обычные для объектно-ориентированного программирования понятия класса и объекта. При этом класс описывает тип оборудования, а объект – конкретный экземпляр. Таким образом, при объявлении класса, согласно принципу инкапсуляции, создаются шаблоны структур данных и методы, которые будут работать с этими данными. В объекте класса по шаблону выстраиваются конкретные данные и приводится ссылка на обслуживающий их процесс.

При появлении нового типа оборудования, благодаря механизму наследования, разработчик не нуждается в том, чтобы заново разрабатывать новый класс: достаточно выбрать наиболее близкий и реализовать отличия в новом классе. Тем самым обеспечивается простота модификаций, сокращаются затраты времени на разработку, снижается общая стоимость разработки.

Наиболее важен тот факт, что объектный подход позволяет создавать хорошо структурированные сложные системы управления электроавтоматикой. Основные преимущества, приобретаемые при этом, состоят в следующем.

- Повышается уровень унификации разработки; для повторного использования пригодны не только управляющие программы, но и проекты в целом, что служит хорошей основой для построения среды разработки. Снижаются затраты времени и средств на создание нового проекта.
- Возникает возможность повторного использования собственных функциональных модулей и готовых модулей других разработчиков, что делает систему управления открытой. Уменьшается вероятность ошибок при разработке сложных систем, увеличивается уверенность в правильности принимаемых решений.

Все эти достоинства обеспечиваются благодаря лежащим в основе объектно-ориентированной технологии принципам наследования, инкапсуляции и полиморфизма.

Архитектура виртуального контроллера

В системе ЧПУ виртуальный контроллер работает в среде ОС *Windows NT* с расширением РВ *RTX* (*Real Time Extension*) фирмы *VentureCom* [2]. Проектирование контроллера предполагает последовательное рассмотрение его модели на трех уровнях абстракции: на уровне модели потоков (структуры потоков), на уровне функциональных модулей и на уровне программной реализации (рис. 1).

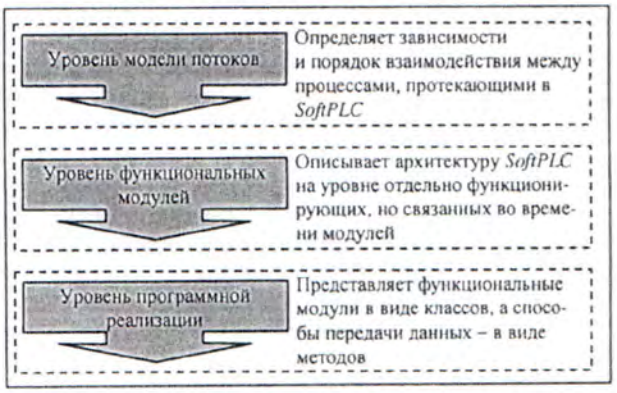

Рис. 1. Последовательная трансформация модели виртуального контроллера SoftPLC

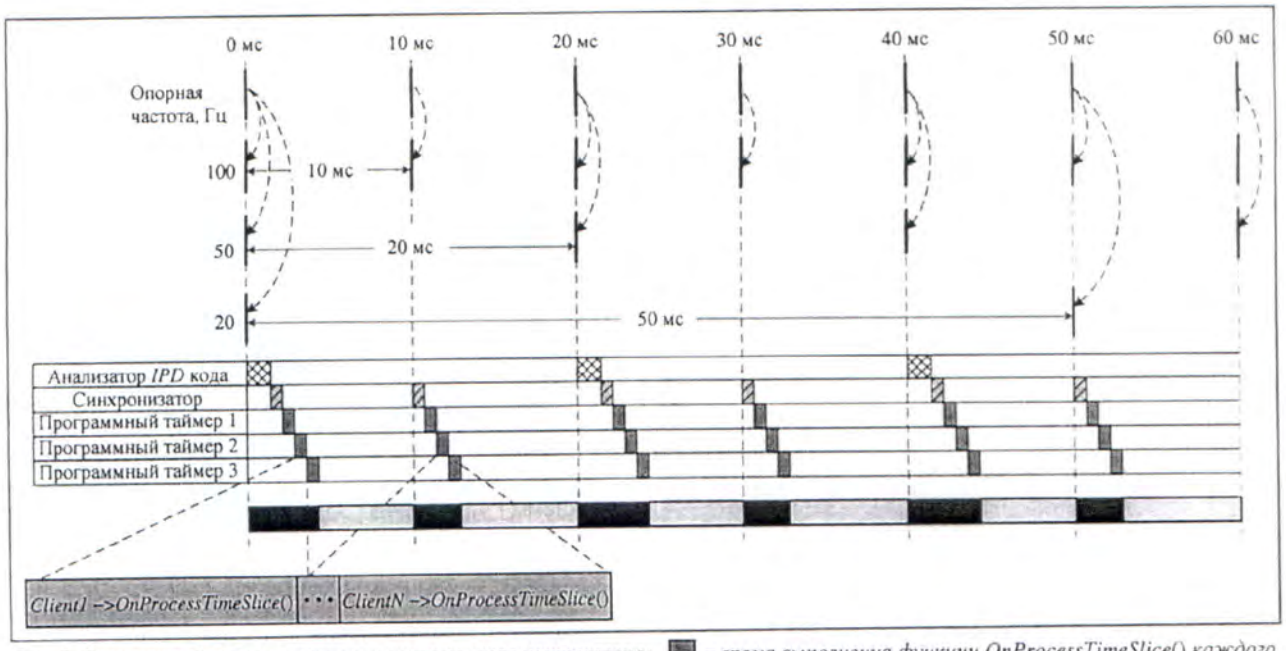

Рис. 2. Временная диаграмма потоков виртуального контроллера: – время выполнения функции OnProcessTimeSlice() каждого таймера; – время работы SoftPLC; – время работы системы; – время работы анализатора IPD кода; – время работы синхронизатора; ---► – вызов программных таймеров

Сначала рассмотрим *структуру потоков*. Основная задача контроллера состоит в одновременном выполнении нескольких команд и параллельной обработке внешних сигналов. Каждый процесс контроллера, который нуждается в выделении отдельного потока, выполняется в рамках основного процесса виртуального контроллера, запущенного под *RTX*. Процессорное время, выделяемое ОС основному процессу, должно быть распределено между потоками. Далее воспользуемся идеей псевдо-многопоточности на основе механизма выделения квантов (разделения времени).

Процессорное время выделяется потокам отдельными квантами с помощью внутренних механизмов виртуального контроллера. В каждом кванте может выполняться только один поток. Все потоки разделены на группы по приоритетам, причем управление группой осуществляется отдельным программным таймером. Программный таймер аналогичен системному, реализованному в ОС, но не генерирует прерывания, а его обработчик запускается планировщиком (в нашем случае – модулем синхронизации). Выделение нескольких групп потоков в виртуальном контроллере связано с тем, что различные его задачи требуют разного времени реакции на внешнее воздействие: чем меньше время реакции, тем выше приоритет потока, обслуживающего задачу.

Более высокий приоритет потока означает более высокую частоту выделения квантов времени. Различные частоты поддерживаются в системе несколькими таймерами, каждый из которых активизируется на своей частоте и выделяет кванты времени своим потокам. Модуль синхронизации осуществляет синхронную активизацию таймеров.

На временной диаграмме потоков (рис.2) рассмотрен случай, когда виртуальный контроллер работает с тремя группами псевдо-потоков. Каждая группа получает кванты времени на выделенной частоте, что соответствует трем приоритетам. Таймер опорной частоты запускается на максимальной частоте сканирования входных регистров. Частоты программных таймеров формируются путем деления опорной частоты в обработчике событий таймера опорной частоты.

На временной диаграмме (рис. 2) опорная частота равна 100 Гц, модуль синхронизации настроен на использование частот 100, 50 и 20 Гц. Для каждой частоты назначен программный таймер. Интервалы времени выделены: для анализатора *IPD* кода (*InterPolator Data*, данные на входе интерполятора), синхронизатора, каждого таймера в отдельности. Таким образом, в интервалах времени, кратных 10 мс, будут работать все три таймера. С целью более равномерного распределения нагрузки в интервалах времени, задачи второго и третьего программных таймеров разделены, соответственно, на две и четыре подгруппы. Это позволяет запускать подгруппы поочередно в каждом интервале времени.

Модель контроллера *на уровне функциональных модулей* показана на рис.3.

Виртуальный контроллер имеет пять составных частей (модулей):

- анализатор, читающий *IPD*-данные из входного буфера и преобразующий эти данные во внутренний формат виртуального контроллера с учетом входных и выходных регистров электроавтоматики;
- синхронизатор, поддерживающий механизм назначения квантов времени, генерирующий синхросигналы для всех процессов виртуального контроллера;
- исполняемые модули, служащие для отработки команд, поступающих в виртуальный контроллер, таких как опрос датчиков аварийного останова и конечных выключателей, включение/выключение подачи охлаждающей жидкости, зажим/разжим патрона, запуск/останов шпинделя, опрос датчиков температуры и т.д.;

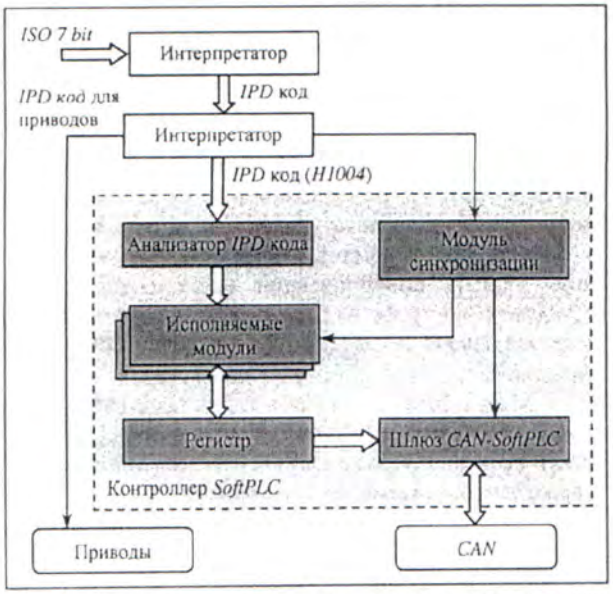

Рис. 3. Архитектура виртуального контроллера на уровне функциональных модулей

• регистр, используемый для обмена информацией между системой ЧПУ и виртуальным контроллером;

• шлюз, предназначенный для отображения информации, передаваемой по *CAN*-магистрали в регистр.

Взаимодействие модулей осуществляется следующим образом. В результате интерпретации управляющей программы ЧПУ формируется промежуточный *IPD* код [3], представляющий собой универсальный бинарный код, не зависящий от используемой платформы. *IPD* код содержит траекторную информацию об относительных перемещениях инструмента и детали, а также и информацию о вспомогательных *M*-командах. Модуль интерполятора читает *IPD* код, отделяя те данные, которые относятся к командам контроллера. Выделенная команда направляется соответствующему исполняемому модулю, внутри которого есть все необходимое для выполнения команды контроллером. Исполняемый модуль представлен в виртуальном контроллере в виде отдельной задачи.

Допускается параллельное выполнение нескольких *M*-команд. Механизм назначения квантов времени, генерирующий синхросигналы для всех процессов контроллера, обеспечивает синхронное выполнение команд. После отработки внутреннего алгоритма контроллер передает интерполятору информацию о своем состоянии.

Обмен данными между контроллером и системой ЧПУ осуществляется через разделяемую память, называемую в нашем проекте "регистром". В регистре выделены три блока: блок специальных маркеров *SM* (*Special Marker*), блок входных данных виртуального контроллера и блок выходных данных контроллера.

Маркер *SM* используется для оповещения о нерегулярных ситуациях, возникающих в системе ЧПУ и в контроллере. В каждом кванте времени блок *SM* анализируется на наличие в нем признаков сбоев (отказов аппаратуры и механизмов и др.) или признаков особо важных сигналов (при воздействии оператора на внешние органы управления). Например, аварийный останов принудительно прекращает работу всей системы. Информация о поступлении аварийного сигнала (при нажатии на кнопку аварийного останова) распространяется по всем объектам контроллера через определенную ячейку в блоке *SM*.

Блоки входных и выходных данных предназначены для организации двустороннего обмена с объектом управления: контроллер считывает информацию из блока входных данных и записывает информацию в блок выходных данных. Передача информации между регистром и *CAN*-магистралью осуществляется посредством шлюза. Модель виртуального контроллера *на уровне программной реализации* рассмотрим в отдельном разделе.

Программная реализация виртуального контроллера

Виртуальный контроллер представляет собой некоторую систему с *DLL*-интерфейсом, работающую в отдельном *RT*-процессе PB (*Real Time*). Система имеет единственный экспортируемый класс *CNcMParser*, содержащий набор базовых функций управления и общедоступные экземпляры входных и выходных данных. Остальные механизмы системы защищены от совместного доступа и не контролируются пользователем.

Объекты представляют собой экземпляры классов, описывающих электроавтоматику системы ЧПУ. В рамках модульной архитектуры виртуального контроллера каждый отдельный класс отвечает за свой объект управления на станке (рис. 4). Так, класс *CNcSpindle* отвечает за управление шпинделем; класс *CNcClnt* – за управление механизмом подачи смазочно-охлаждающей жидкости, и т.д.

Благодаря модульной архитектуре виртуальный контроллер обладает высокой степенью гибкости, позволяющей использовать его на станках различных групп и типов. Конфигурация контроллера для заданного типа станка, в процессе ее инициализации, состоит в создании такого набора связанных объектов, который воспроизводит конфигурацию станка. Например, если в станке имеются два шпинделя, то будут созданы два объекта класса *CNcSpindle*, и т.п.

При создании объектов осуществляется их взаимное связывание. На уровне языка *C++* процесс связывания состоит в инициализации специальных указателей. Например, для привязывания патрона к шпинделю

Рис. 4. Фрагмент диаграммы классов в нотации UML, реализующих исполняемые модули виртуального контроллера

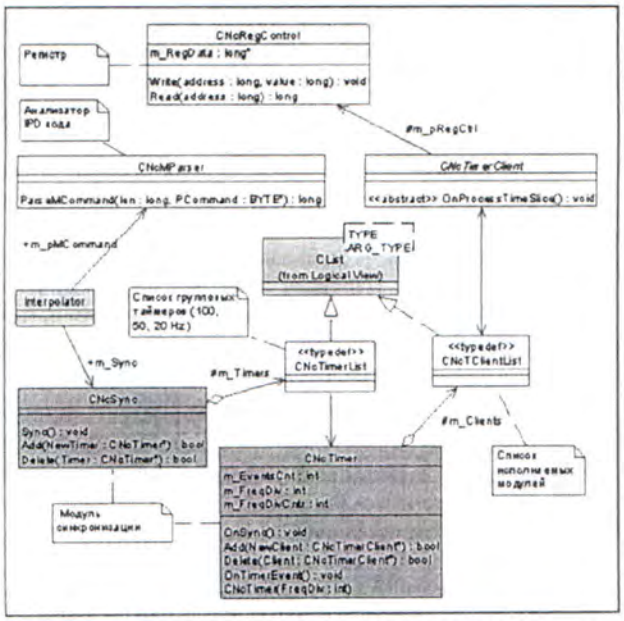

Рис. 5. Диаграмма основных классов, реализующих основные механизмы виртуального контроллера

в классе *CNcChuck* инициализируется указатель *m_ParentSpindle*. Указатели инициализируются в конструкторах классов, т.е. информация о связях между объектами обновляется только при создании объектов.

После создания объектов и при необходимости настройки дополнительных параметров виртуальный контроллер переходит в состояние готовности выполнения команд.

Рассмотрим программную реализацию функциональных модулей, входящих в состав виртуального контроллера (рис. 5).

Интерполятор вызывает *анализатор IPD кода* частотой 50 Гц; при этом анализатору передается некоторый объем *IPD* кода с информацией о командах, которые контроллер должен выполнить. Анализатор описан отдельным классом *CNcMParser*. "Приемником" *IPD* кода, поступающего со стороны интерполятора по указателю на массив, служит метод *ParserMCommand* (*long len, BYTE* PCommand*). Объем кода указан в параметре "*len*". Метод *ParserMCommand()* интерпретирует *IPD* код, выделяя из него команды и их параметры. После выделения очередной команды разыскивается объект, для которого она предназначается (шпиндель, патрон, пиноль и т.д.); затем активизируется команда с заданными параметрами в соответствующем объекте. Активизация команды не означает мгновенного ее запуска: выполнение команды начнется в ближайшем кванте времени, выделенном объекту, и продолжится во всех последующих квантах вплоть до завершения выполнения команды.

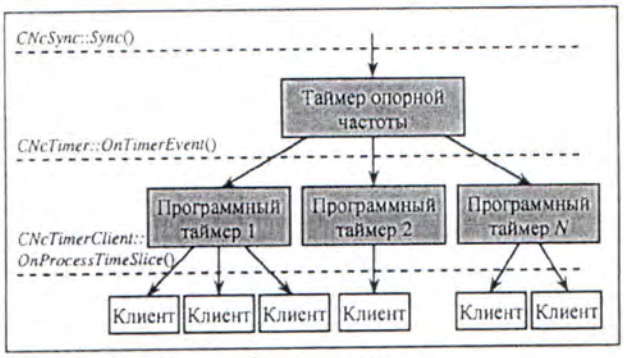

Рис. 6. Схема работы модуля синхронизации

Модуль синхронизации (МС) предназначен для генерации в контроллере необходимого набора внутрисистемных частот, используемых для псевдо-параллельного выполнения команд с помощью механизма квантов. Модуль синхронизации представлен классом *CNcSync* и построен на базе списка программных таймеров *m_Timers*. В класс *CNcSync* включены следующие методы.

- Метод *Sync()*, который вызывается с каждым тактом опорной частоты. Метод, в свою очередь, периодически вызывает методы OnSync каждого таймера, в которых осуществляется деление опорной частоты.
- Метод *Add(CNcTimer *NewTimer)*, предназначенный для добавления нового таймера в список. Каждый таймер содержит две переменные: время счета и счетчик времени. Эти переменные инициализируются при создании таймера, и таймер подключается к синхронизатору при помощи метода *CNcSync::Add()*.

Синхросигнал опорной частоты передается с помощью метода *CNcSync::Sync()* модуля синхронизации (рис. 6). Метод *CNcSync::Sync()* просматривает список программных таймеров и для каждого элемента списка осуществляет вызов метода *OnSync()*. В методе *CNcTimer::OnSync()* увеличивается содержимое счетчика времени. Если значение счетчика времени становится равным времени счета, то счетчик времени обнуляется, после чего вызывается метод *CNcTimer::OnTimerEvent()*.

Синхросигнал передается всем объектам, входящим в состав контроллера. Для этого в *CNcTimer*, на этапе инициализации контроллера, создается список всех объектов *m_Clients*. Элементами списка служат указатели на объекты класса *CNcTimerClient*, порождающего все классы виртуального контроллера. Передача синхросигнала объекту (выделение кванта времени) осуществляется вызовом перегруженного метода *OnProcessTimerSlice()*: из списка *m_Clients* последовательно извлекаются все указатели на объекты, каждый извлеченный указатель вызывает метод *OnProcessTimeSlice()*, в котором команды непосредственно отрабатываются.

Регистр, представляющий собой разделяемую память, поддерживает весь информационный обмен между аппаратными средствами и некоторыми объектами виртуального контроллера. Например, сигналы с датчиков отображаются в ячейках разделяемой памяти. Работа с регистром виртуального контроллера абстрагирована в специальном классе *CNcRegControl*. Методы *CNcRegControl::Read()* и *CNcRegControl::Write()* осуществляют чтение-запись регистра. Разделяемая память состоит из набора последовательно расположенных байтов, отображающих как текущее состояние физических объектов, подключенных к виртуальному контроллеру, так и вектор управления физическими объектами. Синхронизацию процессов чтения и записи осуществляет семафор.

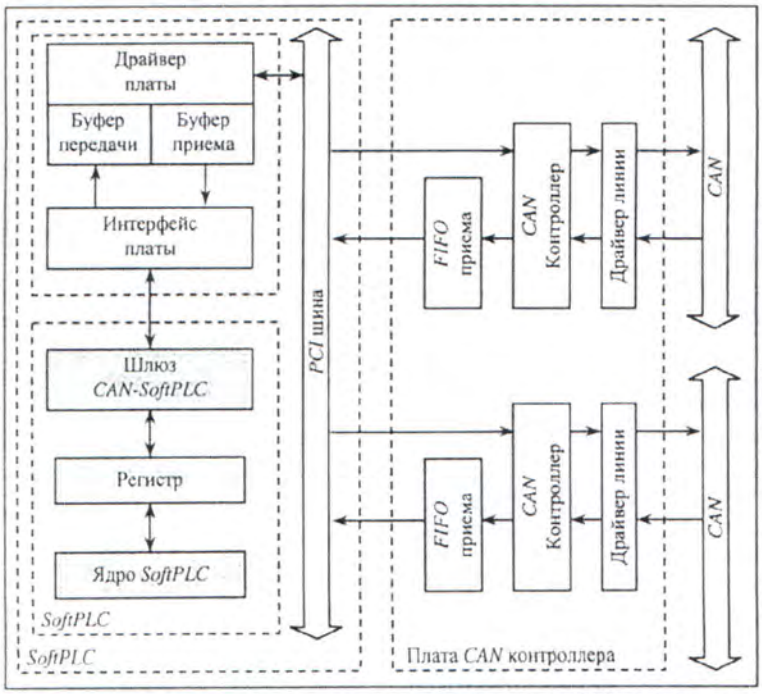

Рис. 7. Схема взаимодействия виртуального контроллера с CAN-магистралью

Шлюз CAN-PLC предназначен для отображения информации, передаваемой по магистрали *CAN* к регистру. Шлюз работает с интерфейсными функциями *CAN*-контроллера, считывает пакеты и направляет их в *CAN*-магистраль. Если некоторый флаг в регистре отображает состояние конечного переключателя, то шлюз будет принимать пакеты от конечного переключателя, анализировать пакеты и обновлять этот флаг в регистре. Если некоторый байт регистра отвечает за управление устройством, подключенным к *CAN*-магистрали, то задачей шлюза будет слежение за изменениями этого байта и передача устройству его значения.

CAN интерфейс

CAN (*Controller Area Network*) представляет собой последовательную асинхронную шину, использующую витую пару в качестве среды передачи. Существуют две версии *CAN* интерфейса: в версии *A* (*BasicCAN*) для идентификации сообщений выделены 11 бит, при этом система обслуживает до 2048 сообщений; в версии *B* (*FullCAN*) для идентификации сообщений выделены 29 бит, при этом система обслуживает до 536 млн. сообщений. *CAN* интерфейс используют в распределенных системах для объединения интеллектуальных датчиков, интеллектуальных приводов и систем управления. Технологию *CAN* поддерживает и развивает некоммерческая международная группа *CiA* (*CAN in Automation*) [4].

В нашем случае *CAN* интерфейс осуществляет передачу данных между разделяемой памятью и устройствами станка. В состав *CAN* интерфейса входят контролер, драйверы, сеть и удаленные устройства ввода-вывода (рис. 7).

Если один из абонентов сети передает информационный пакет, то прочитать его могут любые другие абоненты. Если *CAN*-контроллер обнаруживает признак начала передачи пакета, то считывает его во внутренний *FIFO*-буфер, из которого впоследствии пакет будет направлен в систему через драйвер контроллера. *FIFO*-буфер необходим, поскольку пакеты могут поступать быстрее, чем их считывает драйвер. Как только *FIFO*-буфер принимает очередной пакет, драйвер копирует его в буфер приема, откуда затем пакет может быть считан и проанализирован с помощью *API*-функций *CanRcvMsg()* (*Application Program Interface*). Если необходимо отправить пакет в сеть, то будет вызвана соответствующая *API*-функция *CanSendMsg()* *CAN* интерфейса, которая сформирует пакет на основе полученных ею параметров и скопирует его в буфер передачи в памяти драйвера. Если драйвер обнаружит в буфере передачи новый пакет, то передаст его в контроллер и далее в сеть. Диаграмма последовательности считывания и записи данных в *CAN* со стороны исполнительного модуля в нотации *UML* [5] представлена на рис. 8.

Скорость передачи данных в физическом канале *CAN* достигает 1 Мбит/с. Фактическая скорость зависит от расстояния между абонентами, уровня помех и времени работы математического обеспечения канала. К передаваемым данным добавляется служебная информация, позволяющая идентифицировать пакет в узлах сети. Работа сети строится по принципу "*каждый слышит каждого*". При этом возможны два варианта информационного обмена (на примере датчика).

1. Датчик передает информацию о своем состоянии в сеть с некоторой частотой (например, 100 Гц). Контроллер обрабатывает поступающие данные по мере необходимости. Если они в данный момент не нужны, то игнорируются. Таким образом, ненужные данные все равно поступают в сеть, засоряя канал.

2. Датчик передает данные в сеть только по запросам, формируемым устройством. В нормальном режиме датчик следит за изменением своих параметров, но передает данные в сеть только при необходимости, т.е. после получения запроса. Запрос может быть однократным или "групповым".

Наиболее удачный вариант состоит в том, чтобы найти оптимальную (различную) частоту отребности в

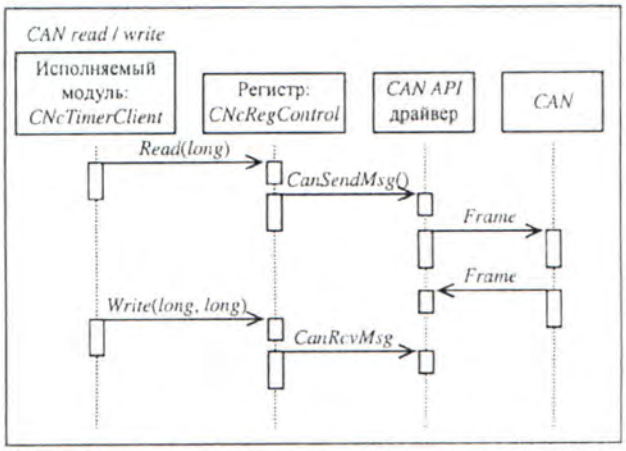

Рис. 8. Диаграмма последовательности считывания и записи информации в CAN интерфейс

данных от каждого датчика. В ряде случаев поток данных может достигать скорости, граничащей с пропускной способностью канала. Такие потоки лучше настраивать на "групповую передачу" по запросам.

Описанные методы относятся к низкоуровневым *CAN*-протоколам. Существуют протоколы и более высокого уровня, реализованные поверх нижнего, например, *CANopen*. При их использовании возрастает сервис, но падает эффективная пропускная способность.

CAN-контроллеры могут содержать несколько изолированных *CAN*-портов. В случае недостаточной пропускной способности одного порта можно использовать другие, и тем самым повысить общую пропускную способность. Таким образом, создается несколько *CAN*-сетей, в каждой из которых могут находиться различные датчики. Недостаток заключается в том, что невозможен прямой обмен пакетами между сетями, т.е. абоненты, находящиеся в разных сетях, не могут обмениваться информацией без посредника. В качестве посредника (моста) выступает виртуальный контроллер, который транслирует пакеты из сети отправителя пакетов в сеть получателя.

Имеет смысл группировать датчики по сетям. Критериями могут служить: интенсивность передачи информации датчиками, приоритет информации, принадлежность датчиков отдельным узлам оборудования.

Заключение

Идея построения виртуального контроллера *SoftPLC* на базе ПК чрезвычайно плодотворна и перспективна. Имеется весьма скудная информация о том, как строить ядро такого контроллера. Нами предложен объектный подход, благодаря которому удается достичь высокой степени обозримости системы, сократить затраты времени на разработку ПО и упростить процесс отладки. В рассматриваемой системе виртуальный контроллер *SoftPLC* имеет модульную архитектуру, в которой отдельный класс отвечает за свой объект управления на станке. Благодаря этому, виртуальный контроллер обладает высокой степенью гибкости, позволяющей использовать его для станков различных групп и типов. Основная задача виртуального контроллера *SoftPLC*, заключающаяся в одновременном выполнении нескольких управляющих команд и параллельной обработке внешних сигналов в режиме РВ, была решена при помощи идеи псевдо-многопоточности. Эта идея использует механизм разделения времени (выделения квантов), а также дополнительную возможность работы с приоритетами. Для информационного обмена с аппаратными средствами и между некоторыми объектами виртуального контроллера предложено использовать разделяемую память.

Работа выполнена на кафедре "Компьютерные системы управления" Московского государственного технического университета "Станкин".

Контактный телефон (095) 972-94-40.

E-mail: mail@ncsystems.ru

Список литературы

1. *Сосонкин В.Л., Мартинов Г.М.* Концепция числового программного управления мехатронными системами: реализация логической задачи управления // Мехатроника. 2001. № 2.
2. *Мартинов Г.М., Сосонкин В.Л.* Концепция числового программного управления мехатронными системами: проблема реального времени // Мехатроника. 2000. № 3.
3. *Сосонкин В.Л., Мартинов Г.М.* Принципы построения систем ЧПУ с открытой архитектурой // Приборы и системы управления. 1996. № 8.
4. *http: // www.can-cia.de.*
5. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя / Пер. с англ. М.: ДМК Пресс, 2001. (Сер. "Для программистов").

Новинка

Конфигурируемая система управления

Фирмой *Berstorff GmbH* (Германия) представлена новая система управления *Process Control*, выполненная на базе высокопроизводительного процессора *Pentium* или *Celeron* и снабженная плоским 15-дюймовым графическим *TFT*-дисплеем с сенсорным экраном, что позволяет пользователю прямо управлять процессом в РВ и обеспечивает непосредственный доступ к данным процесса. Современная технология управления и визуализации создает дружественные пользователю условия работы с системой управления. Предусмотрена возможность дистанционного обслуживания системы при помощи модема (телесервис), что уменьшает затраты на сервис и сокращает время простоя. В зависимости от структуры управляемой машины осуществляется индивидуальное конфигурирование системы при помощи конфигурационного менеджера без дополнительного программирования. Для этой цели данные блоков управляемой машины или установки загружаются в банк данных, откуда передаются конфигурационному менеджеру в виде соответствующих модулей. www. *berstorff.de*

Источник: Kunstoffe. 2002. Bd. 92. № 12.