ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
УЧРЕЖДЕНИЕ НАУКИ
ИНСТИТУТ ФИЗИКИ им. Л.В. КИРЕНСКОГО
СИБИРСКОГО ОТДЕЛЕНИЯ РОССИЙСКОЙ АКАДЕМИИ НАУК

научно-образовательный курс

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

В НАУКЕ И ПРОИЗВОДСТВЕ

Втюрин А.Н., Крылов А.С.

Красноярск 2013

Модуль 1.
Общие принципы программного управления
внешними устройствами ЭВМ
и автоматизации физического эксперимента

Раздел 1.
Принципы и средства автоматизации физического эксперимента

Лекция 1.
Введение в дисциплину. Предпосылки применения компьютеров в экспериментальной физике

План лекции.
Усложнение экспериментальной техники
Совершенствование ЭВМ
Новые возможности, предоставляемые автоматизацией эксперимента

Усложнение экспериментальной техники

Проведение экспериментальных научных исследований можно представить в форме диалога экспериментатора с природой. Природа отвечает на поставленные человеком вопросы, предполагающие поиск зависимостей между явлениями, результатами целенаправленных наблюдений и измерений, и теоретическими понятиями. Каждый исторический этап в развитии науки требует и приводит к созданию новых технических средств ведения этого диалога. Современный этап развития экспериментальной физики характеризуется чертами, которые зачастую делают невозможным проведение измерений без использования комплекса средств, образующих понятие автоматизации физического эксперимента.

Так, характерной чертой современного эксперимента является огромное количество получаемой в его ходе информации, накопление, хранение и обработка которой возможны только с использованием вычислительной техники. Высокие скорости поступления информации и необходимость ее обработки не после завершения эксперимента, а непосредственно в ходе измерений, требуют прямой связи ЭВМ с измерительной аппаратурой, что, в свою очередь, существенно расширяет возможности экспериментальной установки и повышает качество результата.

Еще одна характерная черта современного эксперимента – необходимость контроля и управления большим числом служебных параметров в ходе эксперимента, что тоже наиболее эффективно может быть

осуществлено с помощью ЭВМ. Использование управляющей ЭВМ позволяет оперативно в ходе эксперимента изменять настройку аппаратуры, руководствуясь результатами предварительной обработки получаемых экспериментальных данных. Во многих случаях это существенно снижает затраты времени на проведение эксперимента и повышает точность измерений.

Совершенствование ЭВМ

В последние десятилетия произошел стремительный прогресс вычислительной техники. Скорости вычислений и объемы обрабатываемой информации выросли за последние 30–40 лет в сотни тысяч раз; в десятки тысяч раз снизились энергопотребление компьютеров, их размеры и цена. Это обусловило появление и массовое распространение персональных ЭВМ, предоставляющих пользователю те же, а часто и большие, ресурсы, которыми обладали в 60-х годах большие ЭВМ, а в 70-х – миникомпьютеры.

В 60-х годах расходы по приобретению и эксплуатации больших ЭВМ были сопоставимы со стоимостью наиболее крупных экспериментальных установок атомной и ядерной физики, и обслуживались десятками, а то и сотнями высококвалифицированных специалистов. Разумеется, именно этими областями и ограничивалось, главным образом, их применение в физике.

Развитие элементной базы, появление транзисторов, а затем – интегральных схем, провело к появлению микро-ЭВМ, обладающих, при сравнимой производительности, существенно меньшими габаритами, стоимостью, более высокой надежностью и существенно меньшими требованиями к обслуживанию. Это сделало возможным их массовое применение в качестве не только вычислительных, но и управляющих ЭВМ, в том числе – в системах физического эксперимента средней сложности. К этому времени относится появление измерительно-вычислительных комплексов – специализированных систем, предназначенных, в том числе, для автоматизации физического эксперимента, начало массовой разработки устройств сопряжения компьютеров и измерительных устройств, создания соответствующего прикладного программного обеспечения.

Появление персональных компьютеров, как нового класса ЭВМ, послужило толчком к революционному перевороту во всех областях человеческой деятельности, в том числе – и в методах экспериментальной физики. Компьютер стал повседневным инструментом физика, столь же привычным, как ранее – калькулятор, а до него – логарифмическая линейка. Массовое распространение прикладного математического обеспечения, в том

числе – специально ориентированного для обработки данных измерений породило естественное желание организовать непосредственный ввод этих данных в компьютер, а осуществление этой потребности приносит принципиально новые возможности, связанные с оперативной обработкой данных и управлением многопараметрическими экспериментами.

Новые возможности, предоставляемые автоматизацией

Применение простых и надежных вычислительных средств в системах автоматизации эксперимента, доступных для использования в лабораториях средних размеров, привело массовой автоматизации рутинных процедур измерений и развитию соответствующего программного и аппаратного обеспечения. Одновременно перед экспериментаторами открылись новые, недоступные ранее возможности. В первую очередь это связано с высокой скоростью производимых операций и возможностью оперативной обработки больших объемов информации. Высокие характеристики и возможности современных ЭВМ и программного обеспечения, видимо, еще не до конца оценены специалистами-экспериментаторами. Временной диапазон процессов, воссоздаваемых, и, следовательно, в принципе контролируемых ЭВМ, охватывает диапазон от 10^{-12} до 10^{8} с, что, видимо, неосуществимо в других технологиях. Очевидным примером является использование ЭВМ в физике элементарных частиц, где современные исследования немыслимы без многопараметрических систем контроля параметров эксперимента и глубокой статистической обработки результатов; в нелинейной оптике и квантовой электронике только системы автоматизированного контроля позволяют проводить экспериментальные исследования процессов взаимодействия сверхкоротких лазерных импульсов; стали практически рутинными методики матричной регистрации изображений и многоканальной спектроскопии, требующие параллельной обработки колоссальных массивов информации.

Литература

1. Ступин Ю. В. Методы автоматизации физических экспериментов с помощью ЭВМ / Ю. В. Ступин – М.: Энергоатомиздат, 1983. – 160 с.
2. Втюрин А. Н. ЭВМ в физическом эксперименте. Учебное пособие / А. Н. Втюрин, А. Г. Агеев, А. С. Крылов – Новосибирск, Изд-во СО РАН, 2003. – 150 с.

Лекция 2.
Области применения автоматизированных систем в экспериментальной физике

План лекции.
Области применения автоматизированных систем
в экспериментальной физике
Блок-схемы связи ЭВМ с экспериментальными установками

Разработка и создание сложных и ресурсоемких экспериментальных измерительных физических комплексов «с нуля» является скорее исключением, чем повседневным занятием большинства физиков-экспериментаторов. Как правило, автоматизация эксперимента средней сложности происходит поэтапно, и производится на основе некоторой уже действующей установки. При этом первым толчком является потребность в ее модернизации – при введении в эксперимент дополнительного контролируемого параметра или новой, более совершенной системы управления. В качестве примера можно привести замену асинхронного двигателя развертки в спектрометрах устаревших моделей на шаговый – это значительно повышает точность позиционирования по спектру и практически исключает механические люфты, но требует введения цифровой системы управления разверткой, то есть той или иной степени компьютеризации управления этим параметром. Таким образом, первым шагом в автоматизации экспериментальной установки становится ***управление отдельными операциями***.

Очевидно, что самой основной операцией любого эксперимента является непосредственное проведение измерений, поэтому, даже начав с автоматизации некоторой другой операции, экспериментатор довольно быстро приходит к мысли о передаче в управляющий компьютер и собственно результатов измерений – разумеется, если его мощность позволяет это сделать. При этом одновременно, в той или иной степени, реализуется ***сбор и обработка данных в ходе эксперимента*** – в зависимости от возможностей используемого компьютера и программного обеспечения это может быть простая визуализация получаемых результатов, предварительная фильтрация данных и пр.

Наличие как собственно измеряемых данных, так и возможностей автоматизированного контроля параметров эксперимента позволяет решить задачу автоматизации еще более полно – осуществить полную ***автоматизацию управления экспериментальной установкой***, когда управляющая ЭВМ контролирует (полностью или частично) условия эксперимента, ведет сбор измеряемых величин, анализирует их «на ходу» и в

зависимости от результатов анализа корректирует условия в соответствии с поставленной экспериментатором задачей.

Как в процессе выполнения эксперимента, так и после его окончания, с полученными данными производятся определенные математические операции – как в целях оптимизации процедуры их получения, так и для того, чтобы извлечь из них наиболее полную физическую информацию. В обоих случаях это предполагает проведение некоторой ***обработки экспериментальных данных.***

Кроме упомянутого выше, автоматизированные системы применяются также при создании баз экспериментальных данных, моделировании эксперимента, проектировании экспериментальных установок теоретической интерпретации экспериментальных данных, разработке и отладке программного обеспечения управления экспериментом и в других областях экспериментальной физики, однако этих разделов в этом курсе мы будем касаться меньше.

Блок-схемы связи ЭВМ с экспериментальными установками

Типичные блок-схемы связи ЭВМ с экспериментальными установками или их отдельными узлами показаны на рис. 1.

Первая схема предполагает, что управляющая ЭВМ является неразрывной частью установки. Для создания подобного комплекса необходимо осуществить разработку специализированного компьютера (или микропроцессорного блока), специально предназначенного для управления данным экспериментом.

Это (особенно случае управления достаточно сложным устройством) предполагает участие высококвалифицированных специалистов-разработчиков и программистов, и далеко не всегда возможно в условиях исследовательской лаборатории. Кроме того, такая схема автоматизации является весьма жесткой; сколь-нибудь заметное изменение экспериментов, проводимых на данной установке, потребует существенной переделки как аппаратного, так и программного обеспечения системы автоматизации. В связи с этим подобные схемы используются, как правило, при автоматизации серийно выпускаемых приборов, предназначенных для рутинных измерений (например, оборудование

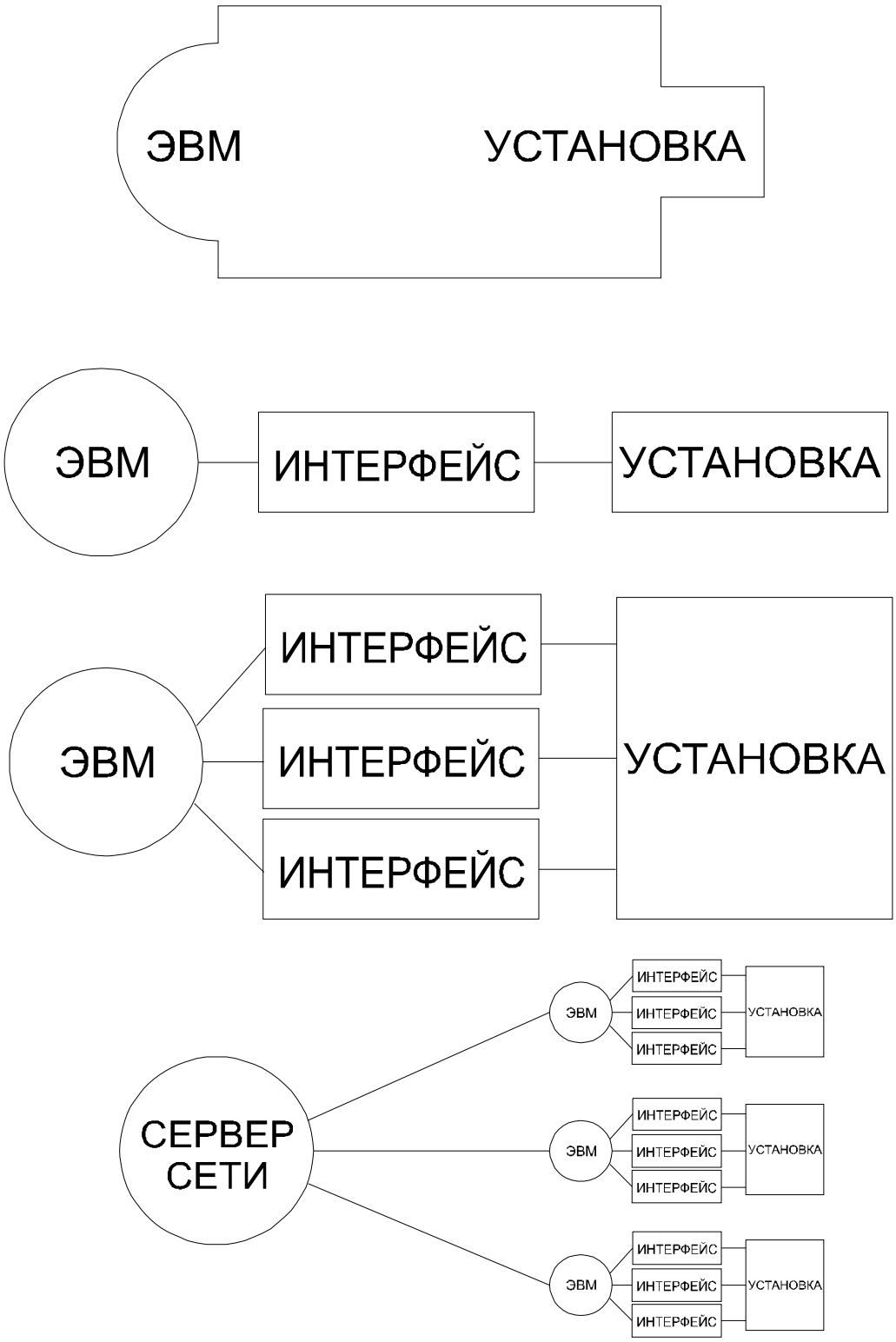

Рис. 1. Блок-схемы связи ЭВМ с экспериментальными установками.

заводских лабораторий), либо для управления простейшими операциями (процессами) в ручном или автоматическом режиме с помощью небольших микропроцессорных систем (системы контроля и управления перемещением, регуляторы температуры).

Существенно более простым является использование в системе автоматизации готового стандартного компьютера. Разумеется, для того, чтобы подключить его к установке, потребуется изготовить или приобрести некоторое интерфейсное устройство, позволяющее вводить данные, получаемые в результате измерений, в компьютер, и преобразовать управляющие сигналы компьютера к тому виду, который требуется управляющим узлам установки. Для его подключения к компьютеру можно воспользоваться либо портами, предназначенными для подключения внешних устройств компьютера (например, параллельные и последовательные порты IBM-совместимых компьютеров), либо внутренними шинами компьютера (ISA, PCI и др. шины IBM PC). Достоинством этого метода подключения является возможность использования стандартного программного обеспечения и развитых средств его разработки, однако здесь по-прежнему отсутствует аппаратная гибкость. Современные экспериментальные установки обеспечивают выдачу широкого набора измерительных сигналов самых разнообразных форматов, и требуют не менее разнообразных управляющих сигналов. Все это приводит к чрезвычайной сложности таких интерфейсных устройств, а внесение любых изменений в установку – требует коренной переработки и аппаратной и программной частей этого интерфейса.

При наличии относительно небольшого числа разнородных информационных и управляющих каналов имеет смысл использовать раздельное интерфейсное устройство для каждого из них. Если при их разработке используется некий стандартный подход, то модернизация установки приводит просто к добавлению еще одного или нескольких интерфейсных модулей и сравнительно небольшой коррекции управляющей программы. Однако при этом возникает ряд проблем, связанных с ростом нагрузки на управляющую ЭВМ при обслуживании большого количества разнородных внешних устройств. В частности, необходимо иметь большое количество каналов ввода-вывода данных в компьютер. В то же время в наиболее распространенных на сегодня компьютерах IBM PC количество таких каналов, как правило, весьма ограничено. В принципе имеется возможность объединения нескольких каналов ввода-вывода в рамках одного интерфейсного устройства. При таком объединении обычно исходят из общего функционального назначения объединяемых каналов связи (например, канал измерения температуры и ее регулировки), либо сходства

осуществляемого ими преобразования (многоканальные аналого-цифровые либо цифро-аналоговые преобразователи). При этом предполагается возможность некоторой предварительной обработки поступающих данных самим объединенным интерфейсным устройством – по меньшей мере, оно должно уметь коммутировать преобразуемые сигналы.

Следующим шагом в этом направлении является использование модульных интерфейсов (КАМАК, ВЕКТОР, PXI, VXI, ряд разработок фирм, выпускающих сложное измерительное оборудование для научных исследований). Основной идеей модульного интерфейса является использование одного устройства – контроллера, связанного непосредственно с управляющей ЭВМ, которое осуществляет коммутацию всех сигналов между остальными интерфейсными модулями и компьютером. При этом от ЭВМ требуется только один канал для подключения контроллера, и вся установка может быть представлена как одно, хотя и достаточно сложное, внешнее устройство. Для того, чтобы все остальные интерфейсные модули можно было подключить к контроллеру, требуется высокая степень стандартизации – механической, электрической, программной. Разумеется, в некоторых случаях это усложняет конструкцию отдельных модулей (и, как следствие, увеличивает первоначальные затраты при создании установки), но существенно упрощает разработку управляющего программного обеспечения и последующую модернизацию установки. Даже замена управляющей ЭВМ требует, в худшем случае, только замены контроллера – все остальные интерфейсные модули остаются теми же.

С целью снижения нагрузки на управляющую ЭВМ можно использовать также стандартные сетевые решения. При этом, в случае автоматизации сравнительно небольшой компактно расположенной установки сеть используется просто для передачи части нагрузки на сервер (например, для хранения и обработки больших объемов данных), тогда как при обслуживании сложных распределенных измерительных систем могут создаваться управляющие многомашинные комплексы, в которых в сеть объединяются ЭВМ, контролирующие работу отдельных узлов установки. Организация подобных сетей достаточно подробно описана в многочисленных литературных источниках и выходит за рамки данной книги.

Приступая к созданию автоматизированной установки, при выборе схемы автоматизации необходимо решить следующие вопросы:

– Четко сформулировать задачи автоматизации.

– Построить алгоритм работы автоматизированной установки.

– Проработать систему измерений, определить набор необходимых интерфейсных модулей.

– Предусмотреть перспективы развития установки, вопросы надежности работы.

– Учесть финансовые возможности.

Требования, определяющие параметры используемого компьютера:

– Объем накапливаемых и обрабатываемых данных.

– Размер программ управления и обработки.

– Скорость приема/передачи данных.

– Скорость обработки данных.

– Необходимое периферийное оборудование.

Из средств вычислительной техники, используемых в эксперименте, наиболее популярны:

– Микропроцессорные наборы – как упоминалось выше, они используются для управления отдельными простыми операциями.

– Компьютеры типа PDP-11 (СМ, Электроника, ДВК): – 16-разрядные ЭВМ, выпускавшиеся в 80-х годах и специально разработанные для управления сложными процессами. В настоящее время они не производятся, однако имеется большое количество действующего оборудования, использующего их в качестве управляющих. Хотя эти компьютеры и обладают сравнительно низкими вычислительными ресурсами, их архитектура позволяет достаточно просто подключать десятки внешних устройств.

– Персональные компьютеры типа IBM PC: наиболее распространены, выпускается большое число интерфейсных устройств для подключения к ним через стандартные порты, внутренние шины и с помощью модульных интерфейсов, имеется развитое программное обеспечение. В то же время их архитектура не рассчитана на большое число каналов ввода-вывода информации.

Литература

3. Певчев Ю. Ф. Автоматизация физического эксперимента / Ю. Ф. Певчев, К. Г.Финогенов – М.: Энергоатомиздат, 1986. – 254 с.
4. Новиков Ю. В. Разработка устройств сопряжения / Ю. В. Новиков, О. А. Калашников, С. Э. Гуляев – М., ЭКОМ, 1997.– 224 с.
5. Соломенчук В. Аппаратные средства персональных компьютеров. / В. Соломенчук – СПб.: БХВ-Петербург, 2003. – 512 с.
6. Ан П. Сопряжение ПК с внешними устройствами / Пей Ан – М.: ДМК Пресс, 2004. – 320 с.

7. Смит Дж. Сопряжение компьютеров с внешними устройствами. Уроки реализации / Дж. Смит – М.: Мир, 2000. – 266 с.
8. Гук М. Аппаратные средства IBM PC. Энциклопедия. / М. Гук – СПб, Питер, 2006. – 1072 с.
9. Курочкин С. С. Системы КАМАК–ВЕКТОР / С. С. Курочкин – М.: Радио и связь., 1981. – 160 с.

Раздел 2.
Понятие архитектуры ЭВМ, основные узлы компьютера. Стандартное программное обеспечение управляющих ЭВМ. Принципы программного управления внешними устройствами ЭВМ

Лекция 3.
Архитектура ЭВМ

План лекции.
Представление данных в ЭВМ
Организация памяти ЭВМ
Команды процессора, управление последовательностью операций

Основная функция любой ЭВМ – обработка информации, представленной в цифровом виде. Для этого необходимо иметь возможности:

– Представить любые данные в заданном цифровом виде (как правило, это двоичные числа).

– Записывать и считывать эти данные по мере необходимости.

– Обрабатывать эти данные – т. е. иметь возможность описания тех операций, которые с ними возможно и необходимо совершать.

Представление данных в ЭВМ

Целые положительные числа – представляются в *двоичной системе исчисления*. Для упрощения записи и восприятия пользователем часто в качестве промежуточных используются восьмеричная или шестнадцатеричная системы.

Целые числа со знаком – представляются в виде *двоичного дополнительного кода*. При этом старший двоичный разряд определяет знак числа (0 – плюс, 1 – минус). Для положительных чисел далее приводится его двоичное значение (например $011_2 = +3_{10}$). Для отрицательного числа n в этих разрядах приводится двоичное значение $n_{\max} - |n| + 1$, где $n_{\max}$ – наибольшее целое число, представимое в пределах заданного количества

двоичных разрядов. Например, если пользоваться 3-разрядными числами (старший разряд – знаковый, т. е. $n_{max} = 011_2 = +3_{10}$), то:

Десятичные числа	–3	–2	–1	0	1	2	3
Дополнительный код (3 разряда)	101	110	111	000	001	010	011

При такой кодировке, несмотря на то, что смысл старшего разряда отличается от остальных, тем не менее сохраняется возможность выполнения арифметических операций обычным путем (с учетом того, что разряды, вышедшие за пределы длины данного представления, теряются). Например, для 3-разрядного представления:

$$-3_{10} + 3_{10} = 0_{10},\ 101_{2д} + 011_{2д} = (1)000_{2д},$$
$$-2_{10} + 3_{10} = 1_{10},\ 110_{2д} + 011_{2д} = (1)001_{2д},$$

здесь в скобках показаны потерянные разряды.

В цифровых измерительных устройствах, широко применяемых в качестве интерфейсов, часто используется *двоично-десятичный код*. Как и в предыдущем случае, здесь старший разряд является знаковым. Остальные разряды делятся на группы по четыре; каждая группа используется для кодировки отдельной цифры десятичного числа:

Десятичное число	+	4	9	7	8
двоично-десятичное представление	0	0100	1001	0111	1000

Исторически первая стандартизованная *кодировка знаковой информации* – код ASCII. Он содержит 128 символов, каждому из которых присвоен свой код, 7-разрядное двоичное число. Первоначально код ASCII включал только латинский алфавит. Затем на его основе был создан код ДКОИ-7; в нем строчные латинские буквы были заменены на заглавные буквы кириллицы. В последствии код ASCII был расширен, введена кодировка ASCII-8, содержащая 256 символов. Дополнительные 128 кодов отведены первоначально для представления псевдографики и спец. символов, но очень скоро их стали использовать для представления символов других языков, в том числе кириллицы (код ДКОИ-8). Современные операционные системы ЭВМ могут использовать несколько отличающиеся кодировки на базе ASCII-8, например приложения ранних версий Windows – ANSI. В этой кодировке первая 128-знаковая страница одинакова и стандартизована, а вторая может заменяться, в зависимости от используемого языка (команды country, code-page операционных систем DOS, Windows 3.11, Windows-95). В настоящее время фирмой MicroSoft в качестве нового стандарта вводится многостраничная кодировка символов (Unicode), в которой первая 256-

знаковая страница совпадает с ANSI, а остальные содержат коды региональных символов, однако в настоящее время здесь возникают проблемы совместимости – приложения, ориентированные на использование только этой кодировки символов, вызывают появление ошибок при взаимодействии с другими, ориентированными на использование двухстраничной кодировки.

Организация памяти

Информация хранится в различных устройствах компьютера (оперативной памяти, регистрах процессора, гибких и жестких дисках и т. д.) в виде двоичных чисел.

Основные типы памяти:

Оперативная память.

Память для чтения и записи
с произвольным доступом (RAM).

Память только для чтения (ROM).

Память с однократной записью (WORM).

Внешняя память.

Память на жестких дисках.

Память на гибких дисках.

Память на компакт-дисках.

Память на магнитооптических дисках.

Память на лентах.

Элементарная ячейка памяти, место для записи одного двоичного разряда (нуля или единицы) – бит. Биты объединяются в группы по восемь разрядов – байты. 1024 (2^{10}) байта – килобайт, 2^{20} – мегабайт, 2^{30} – гигабайт. В большинстве компьютеров (в том числе и в IBM PC) оперативная память организована в виде одномерного массива байтов – ячеек памяти. В них содержатся как данные, так и коды команд, выполняемых процессором. Каждой ячейке присвоен (двоичный) номер – *адрес*. Методы указания этих номеров, допускаемые данным компьютером – *методы адресации*. Кроме простого способа – прямого указания номера требуемой ячейки (абсолютной адресации) – используются указание сдвига номера относительно некоторого заранее заданного (относительная адресация), а также другие методы вычисления требуемого адреса. Полный набор реализованных методов адресации определяется конструкцией ЭВМ и является одной из основных характеристик архитектуры компьютера. Общее количество возможных адресов оперативной (RAM) памяти – *адресное пространство* (не обязательно, что все оно используется в конкретном компьютере).

Первоначально байт являлся основной информационной единицей компьютеров, с которой процессор работал как с единым числом или командой. В настоящее время большинство процессоров (включая процессоры IBM PC) имеют возможность одновременной работы с несколькими байтами памяти, как с единым числом. Группа байтов, которую процессор воспринимает как одно число (одну команду) – машинное *слово*. Зачастую предполагается, что длина слова равна двум байтам (16 битам), хотя современные IBM-совместимые компьютеры имеют возможность работы с 32- и 64-разрядными числами.

Кроме вышеупомянутых, используются еще следующие единицы памяти: *параграфы* (участки по 16 байт, выровненные по началу памяти), *банки* (участки по 64 Кбайт, выровненные по началу памяти), *сегменты* (участки по 64 Кбайт, но начинающиеся с произвольного адреса), *страницы* (участки памяти, которыми компьютер может манипулировать как единым целым: например, перемещать из оперативной памяти на диск или наоборот, отображать на экране дисплея и т. п.).

Команды процессора

Устройством, которое выполняет операции с числами, является процессор. Набор операций, которые может выполнять данный процессор, или его система команд, – одна из основных компонент архитектуры ЭВМ. Каждой команде ставится в соответствие двоичное число – ее код. Исполняемая программа – это упорядоченный набор кодов команд, который должен выполнить процессор. Обычно программы хранятся в устройствах внешней памяти (на дисках, лентах) и по мере необходимости, для их выполнения, перемещаются в оперативную память.

Процессор содержит также несколько специализированных ячеек памяти (*регистров процессора*), предназначенных для хранения служебной информации. Среди них всегда имеется один регистр – *программный счетчик* (PC, program counter) – в котором хранится адрес команды, выполняемой процессором в настоящий момент. По окончании выполнения текущей команды содержимое программного счетчика увеличивается, и процессор переходит к выполнению следующей команды, код которой находится в следующем слове оперативной памяти (разумеется, если содержимое программного счетчика не было изменено в результате выполнения этой команды).

Кроме программного изменения содержимого программного счетчика, существует еще одна возможность управления последовательностью выполнения операций – *прерывание*. Первоначально прерывания

использовались для получения возможности приостановления работы процессора по команде внешнего устройства; в настоящее время они применяются и в других случаях, например, для обработки ошибок, возникающих при работе программ. Работа прерывания заключается в следующем. При поступлении в процессор *запроса прерывания* (interruption request, IRQ), от внешнего устройства или сгенерированного программно, выполнение основной программы прекращается. Процессор обращается к ячейкам памяти, поставленным в соответствие данному запросу (их адреса – *вектор прерывания*, эти адреса устанавливаются изготовителем процессора и являются одной из основных его характеристик), где хранится адрес размещения в оперативной памяти *программы обработки прерывания* и, в некоторых случаях, минимальный набор необходимых для ее выполнения данных. Адрес программы обработки прерывания переносится в программный счетчик, и начинается ее выполнение. На время выполнения программы обработки прерывания адрес команды основной программы, выполнение которой было прервано, размещается в векторе прерывания. По окончании выполнения программы выполнения прерывания восстанавливается исходное значение программного счетчика и вектора прерывания, и выполнение основной программы продолжается с того места, где она была прервана. Конструкция компьютеров предусматривает несколько прерываний (как минимум – по одному на каждое внешнее устройство, способное прерывать работу процессора), и соответственно, несколько адресов прерываний. Для их размещения в оперативной памяти предусматривается *область векторов прерываний*, расположение которой определяется, в конечном счете, разработчиками компьютера. Установка в компьютер устройства, способного прерывать работу процессора, сопровождается (по крайней мере, должна сопровождаться) размещением в оперативной памяти программы обработки соответствующего ему прерывания, и размещением начального адреса этой программы в соответствующем векторе прерывания.

В принципе возможно написание программ процессора (как обычных, так и программ обработки прерываний), используя исключительно коды команд процессора (программирование в кодах). Однако чисто технически это неудобно, поэтому принято каждой команде ставить в соответствие некий буквенный код (например, MOVE – перенос числа в указанную ячейку памяти). Набор таких текстовых команд, для которых существует взаимно-однозначное соответствие с командами данного процессора, образует язык программирования *Ассемблер*. Очевидно, что этот язык индивидуален для каждого данного типа процессоров, и позволяет выполнять любые операции, на которые данный процессор способен. Остальные *языки высокого уровня*

(Fortran, Basic, Pascal, C и др.), относительно независимые от типа процессора, таким взаимно-однозначным соответствием не обладают; одна команда такого языка преобразуется перед выполнением в набор команд процессора (иногда довольно большой), причем такое преобразование может быть неполным (некоторые команды процессора могут быть невыполнимы в рамках данного языка) и неоднозначным (одна и та же команда языка может быть преобразована в различные наборы команд процессора). Существуют специализированные программы, осуществляющие преобразование команд языка программирования в коды процессора. Они делятся на интерпретаторы (осуществляют выполнение полученного кода сразу после выполнения перевода очередной команды) и трансляторы (производят перевод в коды сразу всей программы; этот код запоминается во внешней памяти и переносится в оперативную память для исполнения по мере необходимости).

Набор характеристик:

– Длина машинного слова.

– Система команд процессора.

– Регистры процессора.

– Векторы прерываний.

– Методы адресации памяти.

характеризует архитектуру компьютера.

Компьютеры с одинаковой архитектурой *программно совместимы*, т. е. на них (в принципе) могут исполняться одни и те же программы. Поддержание принципа программной совместимости при разработке новых компьютеров удобно, т. к. обеспечивает преемственность программного обеспечения, но тормозит развитие новых элементов их архитектуры. Поэтому обычно при совершенствовании вычислительной техники используется принцип совместимости сверху вниз: сохранение всех элементов старой архитектуры и введение новых, расширяющих возможности компьютера. Это позволяет использовать и старое программное обеспечение в полном объеме, и новые возможности. Именно такая совместимость была реализована в ряду машин типа PDP-11, а затем – в ряду IBM-совместимых компьютеров.

Литература

10. Соломенчук В. Аппаратные средства персональных компьютеров. / В. Соломенчук – СПб.: БХВ-Петербург, 2003. – 512 с.
11. Гук М. Аппаратные средства IBM PC. Энциклопедия. / М. Гук – СПб, Питер, 2006. – 1072 с.

Лекция 4.
Особенности архитектуры IBM-совместимых компьютеров

План лекции.
Особенности архитектуры IBM-совместимых компьютеров.
Организация оперативной памяти.
Обработка прерываний.
Организация ввода вывода.

Особенности архитектуры IBM-совместимых компьютеров.

Структура организации IBM-совместимых компьютеров показана на рис. 2.

Любой из этих компьютеров включает:

1. Процессор, совместимый с семейством x86 Intel.

2. Единую систему распределения адресов памяти.

3. Отделенное от памяти унифицированное пространство адресов ввода-вывода, с фиксированным положением и совместимым программным управлением обязательных портов.

3. Минимальный набор системных устройств и интерфейсов ввода-вывода.

4. Единую систему аппаратных прерываний.

5. Унифицированные шины расширения (ISA, PCI и др.), состав которых может варьироваться в зависимости от модели компьютера.

6. Базовую систему ввода-вывода (BIOS), выполняющую начальное тестирование компьютера, загрузку операционной системы и включающую набор функций, обслуживающих системные устройства ввода-вывода.

7. Систему прямого доступа к памяти (DMA), позволяющую производить обмен данными между внешними устройствами и оперативной памятью с минимальным участием процессора.

К *обязательным стандартизованным* средствам ввода-вывода относятся:

– Счетчик.
– Интерфейс клавиатуры и управления.
– Канал управления звуком (динамиком).
– Графический адаптер.

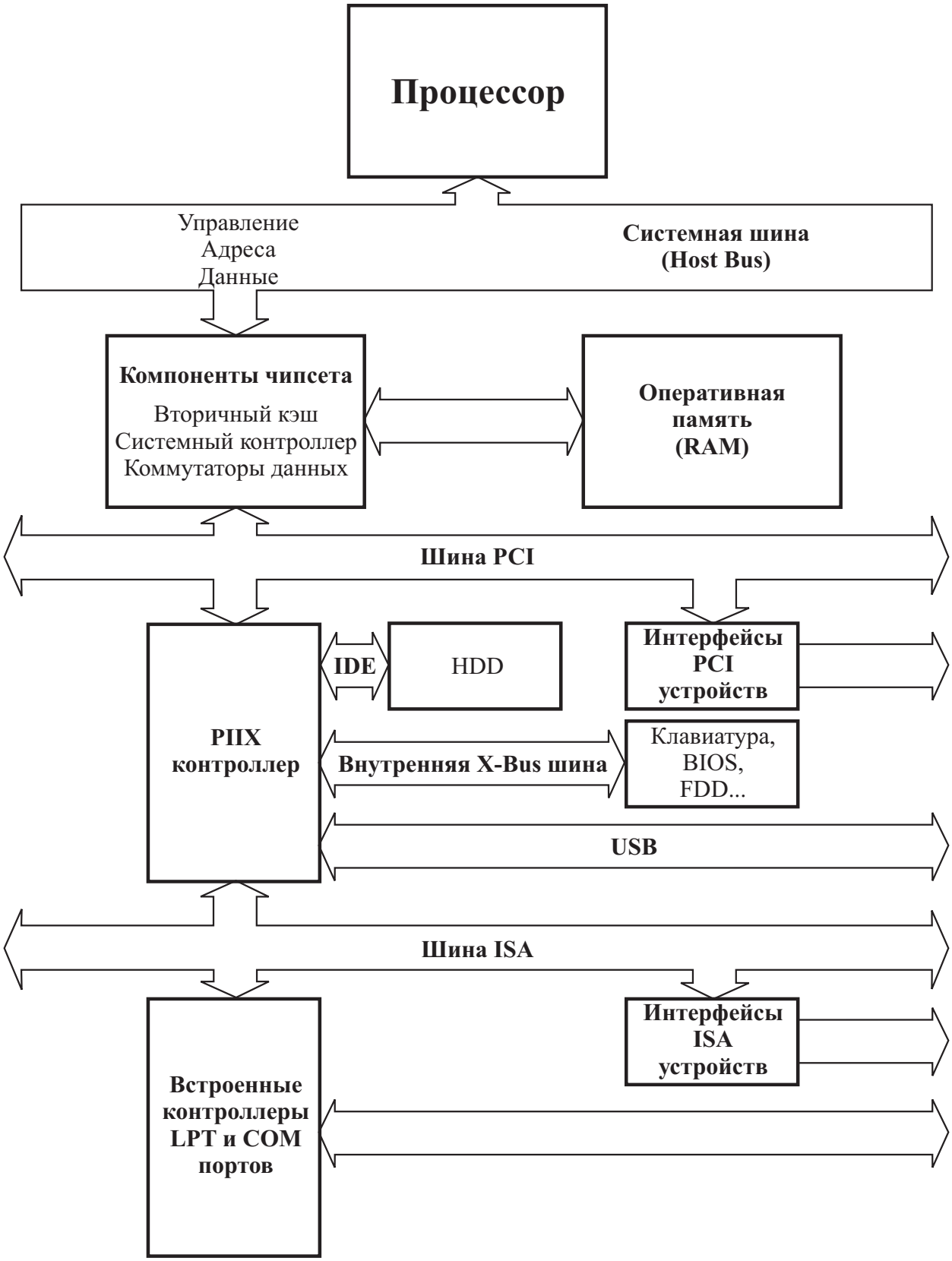

Рис. 2. Структура IBM-совместимых компьютеров.

Основы организации обязательных стандартизованных средств не изменяются от модели к модели (хотя они могут дополняться новыми деталями, что хорошо известно на примере графического адаптера). К *системным* устройствам (то есть обязательно присутствующим в конфигурации компьютера и поддерживаемым BIOS) первоначально относились клавиатура, видеоадаптер с монитором, параллельный (первоначально – один) и последовательные (первоначально – два) порты. Со временем в их число попала дисковая система (сначала – гибкие, затем – жесткие диски, позже – CD-ROM, в некоторых случаях – другие виды накопителей). В некоторых моделях компьютеров (материнских плат) в число системных включают также мышь и цифровой аудиоканал.

Организация оперативной памяти

Первые процессоры семейства x86 – 8086/88 – использовали 16-разрядную адресацию памяти. Шина, используемая для указания адреса памяти, была 20-разрядной, что обеспечивало адресное пространство в 1 Мб, а для вычисления реального адреса на шине (линейного адреса) использовалось два 16-разрядных числа (сегмент и смещение; $Addr = Seg \times 10000_2 + Offset$). В дальнейшем шина адреса была расширена (до 24 бит у 286 и до 32 – начиная с 386 процессоров, что соответствует 4 Гб пространству памяти), и появилась возможность использования 32-разрядной адресации памяти, однако в целях совместимости поддержка изначального метода адресации была сохранена – так называемый реальный режим (*Real mode*) работы процессора. В отличие от него режим работы с использованием 32-разрядных адресов называется защищенным (*Protected mode*).

Распределение памяти показано на рис. 3.

Область 00000_{16}–$9FFFF_{16}$, 640 Кб – *базовая, или стандартная, память*, доступная программам в реальном режиме.

Она, в свою очередь, разделяется следующим образом:

00000_{16}–$003FF_{16}$ – 256 векторов прерываний, по два 16-разрядных слова каждый.

00400_{16}–$004FF_{16}$ – область, используемая BIOS.

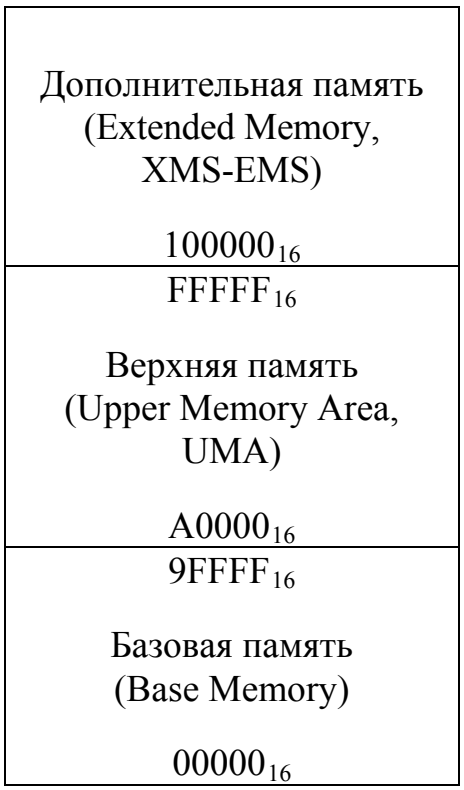

Рис. 3. Распределение памяти IBM-совместимого компьютера.

00500_{16}–$00xxx_{16}$ – область, используемая операционной системой (верхняя граница не фиксирована и определяется операционной системой).

$00xxx_{16}$–$9FFFF_{16}$ – область, предоставленная пользователю.

Область $A0000_{16}$– $FFFFF_{16}$, 384 Кб – *верхняя память*, зарезервированная для системных нужд. В ней размещаются области буферной памяти системных адаптеров (например, видеопамять), и BIOS с расширениями, предоставленными операционной системой.

Верхняя память распределяется следующим образом.

$A0000_{16}$–$BFFFF_{16}$ – 128 К видеопамяти.

$C0000_{16}$–$DFFFF_{16}$ – 128 К, зарезервированных для других системных адаптеров, использующих собственные модули BIOS.

$E0000_{16}$–$EFFFF_{16}$ – свободные 64 К, иногда используемые системным BIOS.

$F0000_{16}$–$FFFFF_{16}$ – 64 К, в которых размещена базовая система ввода-вывода BIOS. Физически эти ячейки памяти находятся в ROM на системной плате.

Как правило, эта область памяти используется имеющимися адаптерами и BIOS не полностью, и операционная система предоставляет пользователю средства использования свободной части верхней памяти. Тем самым увеличивается размер памяти в первом мегабайте, доступном пользователю в реальном режиме работы процессора – но отсюда же зачастую возникают

конфликты использования одной и той же области памяти несколькими устройствами или программами.

Область свыше 100000_{16} – дополнительная (расширенная) память, доступная только в защищенном режиме. Ее нижнюю часть, до $10FFEF_{16}$ (HMA) некоторые операционные системы (например, DOS старших версий) используют для размещения своего ядра с целью экономии пространства базовой памяти.

Обработка прерываний

Аппаратные прерывания обеспечивают реакцию компьютера на события, происходящие независимо от выполняемой им программы, то есть являются необходимой частью программ управления и контроля за внешними процессами. Процессоры x86 поддерживают до 256 прерываний и программ их обработки и различают четыре вида прерываний:

1. Внутренние прерывания процессора.
2. Немаскируемые внешние прерывания.
3. Маскируемые внешние прерывания.
4. Программно-вызываемые прерывания.

Последние не являются прерываниями в точном смысле этого слова, а представляют собой использование механизма прерываний для вызова подпрограмм – не по их адресу, а по номеру приписанного им прерывания.

При поступлении сигнала прерывания процессор завершает выполнение текущей операции, сохраняет в стеке информацию о своем текущем состоянии, в том числе – адрес следующей операции, отменяет разрешение прерываний и вызывает *программу обработки прерывания* – ее адрес (сегмент-смещение, два 16-разрядных слова) хранится в векторе поступившего прерывания в нижних адресах оперативной памяти. Программа обработки прерывания выполняет указанные в ней действия, и процессор, используя хранящиеся в стеке данные, возобновляет выполнение прерванной программы. По умолчанию программа обработки прерывания не может быть вложенной; однако это ограничение легко можно обойти, возобновив разрешение прерываний в начале программы обработки прерывания. Рекомендуется также максимально сокращать время работы программ обработки прерываний – системное время компьютера также определяется по прерываниям от внутреннего таймера, и длительные процедуры обработки других прерываний при установленном запрете приводят к потере системного времени.

Внутренние прерывания процессора возникают при возникновении особых условий выполнения команд. Под их векторы первоначально были

зарезервированы первые 32 из всей таблицы прерываний, однако в настоящее время это выполняется не всегда.

Немаскируемые прерывания обрабатываются процессором независимо от того, установлено или нет разрешение прерываний. Их вызывают системы контроля состояния памяти, управления энергопотреблением, другие внутренние системы ЭВМ.

Маскируемые прерывания – именно тот механизм, который используется для управления большинством внешних устройств. В первых XT компьютерах использовалось 8 маскируемых прерываний с 8-разрядными векторами прерывания. В более поздних AT машинах был установлен дополнительный ведомый контроллер прерываний, который перехватывал прерывание IRQ2 и использовал его для обработки прерываний с IRQ8 по IRQ15. Эта конфигурация сохраняется и до сих пор. Принятые назначения этих прерываний приведены в табл. 1.

Назначения прерываний должны поддерживаться с обеих сторон: адаптер, использующий прерывание, должен быть сконфигурирован на использование установленного номера прерываний (в адаптерах XT использовалась жесткая аппаратная настройка, в современных системах это конфигурирование может устанавливаться джамперами на платах, с помощью специальных программных утилит, либо автоматически); программное обеспечение, управляющее этим адаптером, должно использовать именно это прерывание. Системная шина ISA ориентирована на статическое распределение прерываний и использование одного и того же прерывания несколькими устройствами, подключенными к ней, приводит к ошибкам в работе ЭВМ.

Более современная шина PCI имеет четыре независимых линии запросов прерывания, что дает возможность их разделяемого использования; это осуществляется при настройке опций BIOS и системой Plug&Play.

Таблица 1.

Аппаратные прерывания IBM PC.

Номер прерывания	Вектор прерывания	Назначение в IBM XT	Назначение в IBM AT
IRQ0	8_{16}	таймер	таймер
IRQ1	9_{16}	клавиатура	клавиатура
IRQ2	A_{16}	резерв	2-й контроллер прерываний
IRQ3	B_{16}	COM1	COM2, COM4
IRQ4	C_{16}	COM2	COM1, COM3
IRQ5	D_{16}	HDD	LPT2, звук, резерв
IRQ6	E_{16}	FDD	FDD
IRQ7	F_{16}	LPT1	LPT1
IRQ8	70_{16}	не используется	часы реального времени
IRQ9	71_{16}	не используется	резерв
IRQ10	72_{16}	не используется	резерв
IRQ11	73_{16}	не используется	резерв
IRQ12	74_{16}	не используется	резерв
IRQ13	75_{16}	не используется	сопроцессор
IRQ14	76_{16}	не используется	HDD
IRQ15	77_{16}	не используется	резерв

Организация ввода-вывода

Важной особенностью IBM-совместимых компьютеров является раздельная работа с оперативной памятью, процессором и интерфейсами внешних устройств. Как видно из рис. 2, процессор связан с памятью и интерфейсами несколькими шинами. Любая из них имеет выделенные линии передачи сигналов управления, адреса и данных. Шины могут иметь различную разрядность как данных, так и адреса, и обращение к ним осуществляется различными командами процессора. По выставленной команде элементы чипсета идентифицируют нужную шину, по установленному адресу идентифицируется интерфейсная плата, к которой адресована команда, данные пересылаются или считываются из специализированных ячеек памяти – *регистров* – установленных на интерфейсных платах. Таким образом, задачей интерфейсной платы, установленной на шине, является преобразование внешних данных в формат, определяемый стандартом данной шины, и прием/передача этих данных и команд в соответствии с принятым для данной шины стандартом.

Архитектура этих компьютеров предусматривает возможность взаимодействия с внешними устройствами (т. е. чтения и записи регистров) несколькими способами:

1. Напрямую взаимодействуя со встроенными регистрами интерфейсов.
2. Используя функции базовой системы ввода-вывода (BIOS).
3. Используя встроенные функции операционной системы.

Однако в наиболее распространенных на сегодня операционных системах, основанных на Windows NT (Windows 2000, Windows XP, Windows Vista) поддерживается только третий способ.

Программы могут обмениваться данными с внешними устройствами, обрабатывая прерывание, используя непосредственное управление шиной, либо используя команды обращения к стандартным портам ввода/вывода, если внешнее устройство подключено через такой порт).

Программно-управляемый обмен подразумевает следующую последовательность операций:

1. Чтение регистра состояния устройства для анализа его готовности.
2. Зацикливание предыдущего шага – ожидание готовности.
3. Собственно обмен данными.

Инициатором обмена может быть как основная программа, так и периферийное устройство. Программа ожидает некоторое событие в устройстве (например, установление бита готовности в регистре состояния), периодически считывая содержимое этого регистра. Этот способ называется *обменом по опросу готовности.* При этом время реакции может быть сведено до долей микросекунды (при условии, что программа занимается этим опросом монопольно). Однако при этом процессор во время ожидания оказывается занят бесполезной работой. Другой подход – использование аппаратного прерывания, вырабатываемого устройством при событии, требующем вмешательства управляющей программы. Время реакции на событие в этом случае сильно зависит от режима работы процессора в момент поступления прерывания (так как требуется сохранение информации о работе процессора на момент вызова прерывания), и может составлять от долей микросекунд до десятков и сотен миллисекунд (при работе с виртуальной памятью).

Используется также комплексное решение – опрос готовности устройств не на каждом шагу основной программы, а в моменты времени, определяемые некоторым периодическим процессом – например, прерываниями системного таймера. Периодически опрашиваются все подключенные устройства, и те из них, для которых установлена готовность, обслуживаются. Классическим примером такой работы является утилита фоновой печати PRINT.

Специально для подключения различных адаптеров внешних устройств предназначены *шины расширения.* В компьютерах IBM PC шины расширения начали свое развитие с шины ISA, возникшей в ее первом варианте еще на PC XT. Открытость организации этой шины обеспечила появление широкого спектра плат-адаптеров, позволивших применять эти компьютеры для решения широкого класса задач управления и автоматизации. С появлением AT-286 шина ISA была расширена, что увеличило число подключаемых адаптеров и скорость работы с ними (до 8 Мб/с). Затем, как отклик на потребности в высокопроизводительном обменен данными на серверах возникла еще одна модификация – EISA (33 Мб/с). С появлением процессора 486 возникла потребность в высокопроизводительной работе с графическим адаптером, что привело к появлению специализированной шины VLB (132 Мб/с). Однако узкая специализация и принципиальная привязка к архитектуре процессора 486 привели к быстрому исчезновению этой шины, и развитию новой – PCI (также 132 Мб/с). В настоящее время эта шина стала фактически стандартом, используемым не только в IBM PC, но и ряде других семейств ЭВМ и управляющих устройств. Ее современным развитием стали специализированный графический порт AGP и шина Express PCI.

Литература

1. Соломенчук В. Аппаратные средства персональных компьютеров. / В. Соломенчук – СПб.: БХВ-Петербург, 2003. – 512 с.
2. Смит Дж. Сопряжение компьютеров с внешними устройствами. Уроки реализации / Дж. Смит – М.: Мир, 2000. – 266 с.
3. Гук М. Аппаратные средства IBM PC. Энциклопедия. / М. Гук – СПб, Питер, 2006. – 1072 с.

Лекция 5.
Шины и порты IBM-совместимых компьютеров.
Часть 1

План лекции.
Шины ISA, EISA.
Локальная шина PCI.
Шина PCMCIA.
Шина SCSI.

Шины ISA, EISA

ISA (Industry Standard Architecture) – первая ставшая стандартом шина расширения IBM PC. В компьютерах XT использовалась ее ранняя версия (ISA-8), с разрядностью данных 8 бит и адреса – 20 бит. В компьютерах AT она была расширена до 16 бит данных и 24 – адреса. Типичный вид интерфейсной платы для этой шины показан на рис. 4. Платы подключаются к шине с помощью двух щелевых разъемов. ISA-8 использует только первый из них (группы контактов A, B), в ISA-16 используются дополнительно группы C, D, содержащие по 18 контактов. Количество пар разъемов колеблется от 2–3 до 8–10 для различных материнских плат. Кроме того, на материнской плате имеются интегрированные устройства (например, системный таймер), также использующие эту шину для обмена данными с процессором.

Адреса регистров управления и передачи данных для стандартных устройств находятся в базовой системе ввода/вывода BIOS. Она хранится в ПЗУ компьютера и при загрузке компьютера копируется в нижнюю часть оперативной памяти, сразу после векторов прерываний. Традиционное распределение этих адресов приведено в табл. 2 (хотя для некоторых моделей компьютеров эти адреса могут отличаться).

Шина поддерживает обмен процессора и интерфейсов внешних устройств одно- и двухбайтными данными. Адресное пространство устройств ввода-вывода составляет 64 Кб (758 адресов 8-битных устройств), но практически все существующие интерфейсные платы этого стандарта ограничиваются первым килобайтом этого пространства.

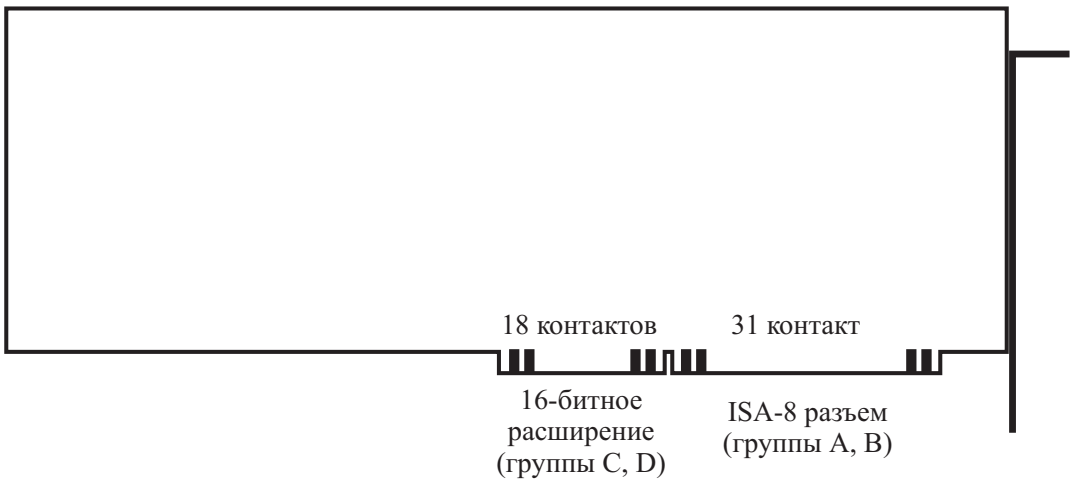

Рис. 4. Вид интерфейсной платы ISA.

Таблица 2.

Стандартные адреса регистров устройств ввода-вывода.

Адреса	Назначение
000_{16}–$01F_{16}$	Контроллер прямого доступа к памяти № 1
020_{16}–$03F_{16}$	Контроллер прерываний № 1
040_{16}–$05F_{16}$	Программируемый таймер
060_{16}–$06F_{16}$	Контроллер клавиатуры
070_{16}–$07F_{16}$	Таймер реального времени
080_{16}–$09F_{16}$	Контроллер страниц памяти
$0C0_{16}$–$0DF_{16}$	Контроллер прерываний № 2
$0F0_{16}$–$0FF_{16}$	Сопроцессор
170_{16}–177_{16}	Жесткий диск HDD2
$1F0_{16}$–$1F7_{16}$	Жесткий диск HDD1
200_{16}–207_{16}	Джойстик
278_{16}–$27F_{16}$	Параллельный порт LPT2
$2C0_{16}$–$2DF_{16}$	Видеоадаптер EGA № 2
$2F8_{16}$–$2FF_{16}$	Последовательный порт COM2
300_{16}–$31F_{16}$	Резерв
320_{16}–$32F_{16}$	Жесткий диск для XT
360_{16}–$36F_{16}$	Резерв
370_{16}–377_{16}	Гибкий диск FDD2
378_{16}–$37F_{16}$	Параллельный порт LPT1
380_{16}–$38F_{16}$	Контроллер обмена № 2
$3A0_{16}$–$3AF_{16}$	Контроллер обмена № 1
$3B0_{16}$–$3DF_{16}$	Видеоадаптер VGA
$3B0_{16}$–$3BF_{16}$	Монохромный видеоадаптер
$3C0_{16}$–$3CF_{16}$	Видеоадаптер EGA № 1
$3D0_{16}$–$3DF_{16}$	Видеоадаптер CGA
$3F0_{16}$–$3F7_{16}$	Гибкий диск FDD1
$3F8_{16}$–$3FF_{16}$	Последовательный порт COM1

Как уже обсуждалось, шина ISA–8 предоставляет внешним устройствам до 6 линий запросов прерываний, для ISA-16 их число увеличено до 11. Шина не имеет механизма автоконфигурирования, и задачей пользователя является бесконфликтное распределение ее ресурсов между интерфейсами. При этом подразумевается:

– Каждая интерфейсная плата при операциях чтения должна выдавать информацию только по своим адресам. Адресные пространства интерфейсов не должны пересекаться.

– В пассивном состоянии плата должна удерживать линию запроса прерывания на низком уровне и переводить на высокий уровень только для

активации запроса. Одной линией запроса может пользоваться только одно устройство.

Назначение контактов плат ISA показано в табл. 3.

Таблица 3.

Назначение контактов интерфейсных плат ISA.

Контакт	Обозначение	Назначение
A1	I/O CH CK	Проверка канала ввода-вывода. Вырабатывается при ошибке, вызывает прерывание.
A2–A9	SD7-SD0	Разряды данных в 8-разрядные регистры интерфейсов либо в младший байт 16-разрядных.
A10	I/O CH RDY	Готовность принять информацию. Снимается интерфейсной платой, если она не успела закончить предыдущую операцию.
A11	AEN	Выставляется процессором при работе в режиме прямого доступа к памяти. При этом все не занятые устройства должны прекратить работу с шиной.
A12-A31	SA19-SA0	Разряды устройств ввода-вывода (практически все устройства используют только SA0-SA9).
B1	GND	Заземление
B2	RESET DRV	Сигнал сброса в начальное состояние. Вырабатывается при включении питания, нажатии кнопки RESET.
B3	+5 В	Питание
B4	IRQ9	Запрос прерывания № 9
B5	–5 В	Питание
B6	DRQ2	Запрос № 2 устройства на прямой доступ к памяти.
B7	–12 В	Питание
B8	0WS	Информирует процессор о необходимости прямого обмена данными. Используется редко.
B9	+12 В	Питание
B10	GND	Заземление
B11	SMEMW	Синхронизация записи данных в память.
B12	SMEMR	Синхронизация чтения данных из памяти.
B13	IOW	Синхронизация записи данных в устройство
B14	IOR	Синхронизация чтения данных из устройства
B15	DACK3	Разрешение прямого доступа к памяти для устройства № 3.
B16	DRQ3	Запрос № 3 устройства на прямой доступ к памяти.
B17	DACK1	Разрешение прямого доступа к памяти для устройства № 1.
B18	DRQ1	Запрос № 1 устройства на прямой доступ к памяти.
B19	REFRESH	Сигнал обмена данными между процессором и памятью. Процессор занят.

Контакт	Обозначение	Назначение
B20	SYSCLK	Сигнал тактового генератора (обычно 8 МГц).
B21–B25	IRQ7–IRQ3	Запросы прерываний.
B26	DACK2	Разрешение прямого доступа к памяти для устройства № 3.
B27	T/C	Окончание работы в режиме прямого доступа к памяти.
B28	BALE	Стробирование разрядов адреса.
B29	+ 5 В	Питание
B30	OSC	Импульсы кварцевого генератора с частотой 14.31818 МГц. Синхронизация работы интерфейсов.
B31	GND	Заземление
C1	SBHE	Тип данных (8- или 16-разрядные).
C2–C8	LA23–LA17	Используются при работе с доп. памятью. При вводе/выводе устанавливаются в нуль.
C9	MEMR	Синхронизация записи данных в память (дополнительные страницы).
C10	MEMW	Синхронизация чтения данных из памяти (дополнительные страницы).
C11–C18	SD8–SD15	Разряды данных в старший байт 16-разрядных регистров интерфейсов.
D1	MEM CS16	Сообщение, что память использует 16-разрядный обмен.
D2	I/O CS16	Сообщение устройства, что оно использует 16-разрядный обмен (по умолчанию – 8-разрядный).
D3–D7	IRQ10–IRQ14	Запросы прерываний
D8	DACK0	Разрешение прямого доступа к памяти для устройства № 0.
D9	DRQ0	Запрос № 0 устройства на прямой доступ к памяти.
D10	DACK5	Разрешение прямого доступа к памяти для устройства № 5.
D11	DRQ5	Запрос № 5 устройства на прямой доступ к памяти.
D12	DACK6	Разрешение прямого доступа к памяти для устройства № 6.
D13	DRQ6	Запрос № 6 устройства на прямой доступ к памяти.
D14	DACK7	Разрешение прямого доступа к памяти для устройства № 7.
D15	DRQ7	Запрос № 7 устройства на прямой доступ к памяти.
D16	+5 В	Питание
D17	MASTER	Сообщение устройства о готовности работы в режиме ДМА (после DRQ и DACK).
D18	GND	Заземление

EISA (Extended ISA) – стандартизованное расширение шины ISA. Геометрические размеры плат этих интерфейсов совпадают с ISA, а четыре дополнительные группы контактов расположены в промежутках и выше контактов ISA, так что плату ISA можно вставить в разъем EISA – она просто

не будет касаться дополнительных контактных площадок. Дополнительные контакты позволили увеличить разрядность данных до 32 бит, за счет чего увеличилась скорость обмена информацией; рабочее адресное пространство при этом не изменилось. Допускается разделяемое использование запросов прерываний, и предусмотрена возможность программно-настраиваемых интерфейсов, для чего на системной плате дополнительно устанавливается энергонезависимая память конфигурации (NVRAM), где хранится информация о конфигурации каждой из установленных интерфейсных плат. В отличие от режима Plug&Play не предусмотрена динамическая перестройка конфигурации – настройка интерфейсов производится специальной утилитой ECU (EISA configuration utility), по окончании работы которой производится перезагрузка компьютера. В целом, это дорогая архитектура, разработанная для применения в системах с объемным обменом данными (типа файл-серверов).

В настоящее время выпускается достаточно большой ассортимент интерфейсных плат ISA, ориентированных на задачи передачи данных в ЭВМ и управления. Как правило, на них монтируется несколько однотипных преобразователей сигнала (АЦП, ЦАП, преобразователей кодов и т. п.). Кроме того, выпускаются *карты-прототипы* (*Prototype Card*), представляющие собой печатные платы с крепежной скобой, на которых размещены обязательные интерфейсные цепи (дешифратор адреса, регистр данных и пр.), и оставлено место для размещения макетного варианта интерфейсного устройства, что удобно для отладки и монтажа единичных экземпляров интерфейсов собственной разработки.

Локальная шина PCI

Локальная шина PCI (Peripheral Component Interconnect), появившаяся впервые на 486 машинах, стала мостом между системной шиной процессора и медленной «классической» шиной ISA. Для действующего в настоящее время стандарта PCI при частоте 20–33 МГц теоретическая максимальная скорость обмена по ней составляет 132 (264) Мбайт/с для 32 или 64-разрядных данных. Допускается также использование частоты 66 МГц, если все установленные на шине устройства поддерживают эту частоту.

Слоты (разъемы) для подключения адаптеров к этой шине расположены на материнской плате несколько дальше от задней панели компьютера. Соединительные контакты на интерфейсных платах расположены на них с двух сторон, образуя группы A и B.Существуют две версии шины, отличающиеся напряжениями питания (5 В и 3.3 В); На платах разных версий имеются прорези на месте 12–13 или 50–51 контактов, в

соответствующих местах слотов – ключи; это не позволяет установить плату в слот с неверным напряжением питания. Имеются, также универсальные слоты и платы, поддерживающие оба варианта питания. 32-разрядные платы имеют 62 пары контактов, 64-разрядные – 94. Наименования сигналов шины приведены в табл. 4, назначение выводов универсального разъема – в табл. 5, 4-разрядные коды команд – в табл. 6.

Таблица 4.

Сигналы шины PCI

Сигнал	Назначение
AD[31:0]	Address/Data – 32 линии адреса/данных. Адрес передается в начале транзакции, в последующих тактах передаются данные.
C/BE[3:0]#	Command/Byte Enable – команда/разрешение обращения к байтам. Команда, определяющая тип очередного цикла шины (чтение/запись памяти, ввод/вывод или конфигурационное чтение-запись, подтверждение прерывания и другие), задается четырехбитным кодом в фазе адреса (см. табл. 6).
FRAME#	Кадр. Введением сигнала отмечается начало транзакции (фаза адреса), снятие сигнала указывает на то, что последующий цикл передачи данных является последним в транзакции.
DEVSEL#	Device Select – устройство выбрано (ответ устройства-приемника на адресованный к нему запрос)
IRDY#	Initiator Ready – готовность устройства-передатчика к обмену данными
TRDY#	Target Ready – готовность устройства-приемника к обмену данными
STOP#	Запрос устройства-приемника к передатчику на остановку транзакции
LOCK#	Используется для установки, обслуживания и освобождения захвата ресурса на шине PCI
REQ[3:0]#	Request – запрос от PCI-мастера (передатчика) на захват шины (разрешен для слотов 3:0)
GNT[3:0]#	Grant – предоставление передатчику управления шиной
PAR	Parity – общий бит паритета для линий AD[31:0] и C/BE[3:0]
PERR#	ParityError – сигнал об ошибке паритета (от устройства, ее обнаружившего)
RST#	Reset – сброс всех регистров в начальное состояние
IDSEL#	Initialization Device Select – выбор устройства в процессах считывания и записи их конфигурации
SERR	System Error – системная ошибка, активизируется любым устройством PCI и вызывает NMI
REQ64#	Request 64 bit – запрос на 64-битный обмен
ACK64#	Подтверждение 64-битного обмена

Сигнал	Назначение
INTRA# INTRB# INTRC# INTRD#	Interrupt A, B, C, D – линии запросов прерывания, циклически сдвигаются в слотах и направляются на доступные линии IRQ. Допускается разделяемое использование линий
CLK	Clock – тактовая частота шины, должна лежать в пределах 20–33 МГц, начиная с PCI 2.1 допустима до 66,6 МГц
M66EN	66MHz_Enable – разрешение частоты синхронизации до 66 МГц, если все абоненты ее допускают (введено начиная с PCI 2.1)
SDONE	Snoop Done – сигнал завершенности цикла слежения для текущей транзакции. Необязательный сигнал, используется только устройствами с кэшируемой памятью
SBO#	Snoop Backoff – попадание текущего обращения к памяти абонента шины в модифицированную строку кэша. Необязательный сигнал, используется только устройствами с кэшируемой памятью
TCK	Test Clock – синхронизация тестового интерфейса JTAG
TDI	Test Data Input – входные данные тестового интерфейса JTAG
TOO	Test Data Output – выходные данные тестового интерфейса JTAG
TMS	Test Mode Select – выбор режима для тестового интерфейса JTAG
TRST	Test Logic Reset – сброс тестовой логики

Таблица 5.

Назначение контактов разъема шины PCI

Ряд B	№	Ряд A	Ряд B	№	Ряд A
–12 B	1	TRST#	GND/M66EN[1]	49	AD 9
TCK	2	+12 B	GND/Ключ 5 B	50	GND/Ключ 5 B
GND	3	TMS	GND/Ключ 5 B	51	GND/Ключ 5 B
TDO	4	TDI	AD 8	52	C/BEO#
+5 B	5	+5 B	AD 7	53	+3.3 B
+5 B	6	INTRA#	+3.3 B	54	AD 6
INTRB#	7	INTRC#	AD 5	55	AD 4
INTRD#	8	+5 B	AD 3	56	GND
PRSNT 1#	9	Reserved	GND	57	AD 2
Reserved	10	+VI/O	AD 1	58	AD 0
PRSNT 2#	11	Reserved	+VI/O	59	+VI/O
GND/Ключ 3.3 B	12	GND/Ключ 3.3 B	ACK64#	60	REQ64#
GND/Ключ 3.3 B	13	GND/Ключ 3.3 B	+5 B	61	+5 B
Reserved	14	Reserved	+5 B	62	+5 B
GND	15	RST#	Конец 32-битного разъема		

Ряд B	№	Ряд A	Ряд B	№	Ряд A
Clock	16	+VI/0	Reserved	63	GND
GND	17	GNT#	GND	64	C/BE7#
REQ#	18	GND	C/BE6#	65	C/BE5#
+V I/O	19	Reserved	C/BE4#	66	+V I/O
AD 31	20	AD 30	GND	67	PAR64
AD 29	21	+3.3 B	AD 63	68	AD 62
GND	22	AD 28	AD 61	69	GND
AD 27	23	AD 26	+VI/O	70	AD 60
AD 25	24	GND	AD 59	71	AD 58
+3.3 B	25	AD 24	AD 57	72	GND
C/BE3#	26	IDSEL#	GND	73	AD 56
AD 23	27	+3.3 B	AD 55	74	AD 54
GND	28	AD 22	AD 53	75	+V I/O
AD 21	29	AD 20	GND	76	AD 52
AD 19	30	GND	AD 51	77	AD 50
+3.3 B	31	AD 18	AD 49	78	GND
AD 17	32	AD 16	+V I/O	79	AD 48
C/BE2#	33	+3.3 B	AD 47	80	AD 46
GND	34	FRAME#	AD 45	81	GND
IRDY#	35	GND	GND	82	AD 44
+3.3 B	36	TRDY#	AD 43	83	AD 42
DEVSEL#	37	GND	AD 41	84	+VI/O
GND	38	STOP#	GND	85	AD 40
LOCK#	39	+3.3 B	AD 39	86	AD 38
PERR#	40	SDONE#	AD 37	87	GND
+3.3 B	41	SBOFF#	+V I/O	88	AD 36
SERR#	42	GND	AD 35	89	AD 34
+3.3 B	43	PAR	AD 33	90	GND
C/BE1#	44	AD 15	GND	91	AD 32
AD 14	45	+3.3 B	Reserved	92	Reserved
GND	46	AD 13	Reserved	93	GND
AD 12	47	AD 11	GND	94	Reserved
AD 10	48	GND	Конец 64-битного разъема		

[1] Сигнал M66EN определен только начиная с модификации PCI 2.1.

Таблица 6.

Коды команд шины PCI

C/BE[3:0]	Тип команды
0000	Interrupt Acknowledge — подтверждение прерывания
0001	Special Cycle — специальный цикл
0010	I/O Read — чтение из устройства на шине
0011	I/O Write — запись в устройство
0100	Резерв
0101	Резерв
0110	Memory Read — чтение памяти
0111	Memory Write — запись в память
1000	Резерв
1001	Резерв
1010	Configuration Read — конфигурационное считывание
1011	Configuration Write — конфигурационная запись
1100	Multiple Memory Read — множественное чтение памяти
1101	Dual Address Cycle — двухадресный цикл

С целью ускорения обмена информацией по шине все процессы обмена данными (транзакции) предполагаются пакетными. На шине выставляется сигнал FRAME#, по 32 линиях AD выставляется адрес приемника, а на четырех линиях C/BE – команда (табл. 6). Указанное устройство-приемник команды отзывается сигналом DEVSEL#, передатчик подтверждает готовность данных сигналом IRDY#, приемник подтверждает готовность к приему сигналом TRDY#. При наличии на шине двух последних сигналов начинается передача данных – по тем же линиям AD; этим обеспечивается синхронная работа, если скорости приема и передачи данных у приемника и передатчика не совпадают. Общий объем передаваемых данных заранее не определен; обмен прекращается, когда передатчик снимет сигнал FRAME#. Обмен может быть прекращен также из-за неготовности приемника (нет сигнала DEVSEL#), либо при поступлении с приемника сигнала STOP#.

С целью повышения надежности обмена на шине введены линии контроля четности (число бит в адресах и данных должно быть четным, в противном случае на шине выставляется сигнал PERR#).

Регистры PCI-устройств могут быть 8- или 16-битными. Для их адресации, в принципе, можно использовать все 32 бита линий AD, но процессоры x86 используют только младшие 16 (шина используется и на других тапах процессоров). Конфигурирование устройств (выбор адресов регистров, запросов прерываний и т. д.) поддерживается средствами BIOS и ориентировано на технологию Plug&Play. Согласно стандарту, каждое PCI-устройство может иметь до 256 8-разрядных регистра, доступ к которым

осуществляется по специальным командам конфигурирования. После перезагрузки системы PCI-устройства не отвечают на обращения к их регистрам данных и доступны в этот момент только для операций конфигурирования. Во время загрузки системы происходит опрос устройств и настройка шины, и только после этого возможно обращение к устройствам для чтения/записи данных.

Для запросов прерываний используются четыре линии INTR A, B, C, D. Подключение линий к определенному слоту может регулироваться программно и первоначально также устанавливается в процессе конфигурации шины. Прерывания, занятые шиной PCI, становятся недоступными для ISA устройств.

Организация вызова прерываний позволяет установить на одной шине PCI не более четырех устройств; для ее расширения либо для соединения с другими шинами материнской платы (шины процессора, ISA) используются *мосты шины PCI*, которые программируются производителями таким образом, чтобы обеспечить однозначную адресацию подключенных к ним устройств. Не распознанные шиной PCI запросы по умолчанию передаются на шину ISA.

Шина PCI является второй по популярности применения после ISA; для нее также выпускаются как специализированные интерфейсные карты, так и карты-прототипы, ориентированные на самостоятельную разработку периферийных интерфейсов. Разумеется, подобная разработка в данном случае представляет более сложную задачу, что связано как с более сложным протоколом обмена данными, так и более высокими частотами работы.

Шина PCMCIA (PC Card)

Шина стандартизована ассоциацией Personal Computer Memory Card International Association для устройств расширения блокнотных компьютеров. Шина позволяет адресовать до 4080 слотов внешних устройств, разрядность данных 16 бит, частота 33 МГц, ориентирована на программное конфигурирование адаптеров. Большинство выпускаемых адаптеров для этой шины используют технологию Plug&Play и предусматривают возможность «горячего» (без выключения машины) подключения. С этой целью контакты линий питания имеют большую длину, чем сигнальные, а контакты Card Detect короче остальных. Тем самым при подключении адаптера сначала на него подается питание, затем подаются управляющие сигналы, и только потом происходит распознавание платы компьютером. Отключение происходит в обратном порядке.

Все устройства PC Card имеют минимальное энергопотребление. В некоторых случаях эта шина устанавливается в качестве дополнительной и в настольных PC.

Шина SCSI

SCSI (Small Computer System Interface) – стандартизованный интерфейс системного уровня, предназначенная для подключения внутренних и внешних периферийных устройств, требующих высокопроизводительного обмена данными. В отличие от рассмотренных выше жестких шин расширения, эта шина реализована в виде кабельного шлейфа, который допускает соединение «один в один» (последовательно) до восьми устройств. Одно устройство, *хост-адаптер*, связывает шину с системной шиной компьютера, семь других могут устанавливаться пользователем.

Каждое установленное на шине устройство имеет свой идентификатор номера (SCSI ID, аналог номера прерывания ISA шины), значение которого передается по восьми линиям шины (отсюда и ограничение на число устройств). Шина имеет большое количество вариантов – механических, электрических и логических, отличающихся как разрядностью данных, так и скоростью их передачи:

SCSI-1 – 8 бит, 18 команд, 5 МГц.

SCSI-2 – 8 бит, добавлены специальные команды управления CD-ROM, 5 МГц. Устройства могут выполнять наборы (цепи) до 256 команд и обмениваться данными без участия центрального процессора.

Fast SCSI-2 – частота повышена до 10 МГц.

Wide SCSI-2 – 16- и 32-битные данные.

Ultra SCSI – частота повышена до 20 МГц.

SCSI-3 – внедряемый сейчас вариант шины, предусматривающий увеличение числа устройств на шине, поддержку Plug&Play, увеличение частоты до 100 МГц, возможность использования последовательного оптоволоконного кабеля вместо шлейфа.

Ввиду большого разнообразия вариантов организации шины и возможности как внутренней, так и внешней установки устройств, существует большое разнообразие кабелей и разъемов для их подключения. Из-за жестких требований к адаптерам и, как следствие, их высокой стоимости, для целей управления и сбора информации от нестандартных устройств (экспериментальных установок) эта шина используется крайне редко.

Литература

1. Соломенчук В. Аппаратные средства персональных компьютеров. / В. Соломенчук – СПб.: БХВ-Петербург, 2003. – 512 с.
2. Смит Дж. Сопряжение компьютеров с внешними устройствами. Уроки реализации / Дж. Смит – М.: Мир, 2000. – 266 с.
3. Гук М. Аппаратные средства IBM PC. Энциклопедия. / М. Гук – СПб, Питер, 2006. – 1072 с.
4. Гук М. Аппаратные интерфейсы ПК. Энциклопедия. / М. Гук – СПб, Питер, 2003. – 528 с.

Лекция 6.
Шины и порты IBM-совместимых компьютеров.
Часть 2

План лекции.
Параллельные порты.
Последовательные порты.
Порт (шина) USB.

Параллельные порты

В первых моделях компьютеров семейства IBM PC не была заложена возможность для пользователя изменять их конфигурацию; единственным путем подключения дополнительных внешних устройств были *параллельные* и *последовательные порты.*

Основным назначением параллельных портов (стандарт Centronics; обычно обозначаются LPT) является подключение к компьютеру принтеров на расстоянии до 1.8 м. Поэтому распределение контактов разъемов этого порта и соответствующего кабеля, назначение сигналов, стандартные программы работы с этими портами (включая программы обработки соответствующих прерываний) ориентированы именно на это использование. Тем не менее широко распространено использование этих портов и для других целей.

Порты предназначены для параллельной передачи восьмибитных групп данных и управляющих сигналов. Назначение контактов показано в табл. 7.

Таблица 7.

Сигналы интерфейса Centronics

№ контакта на компьютере	№ контакта на кабеле	I/O	Наименование и назначение сигнала
1	1	O	STROBE, стробирование, сигнал начала передачи данных
2–9	2-9	O	D0–D7, биты данных
10	10	I	ACK, подтверждение принятия данных и готовности к приему следующих
11	11	I	BUSY, отсутствие готовности к приему
12	12	I	PE, конец бумаги
13	13	I	SLCT, готовность к работе
14	14	O	AUTO FD, перевод строки
15	32	I	ERROR ошибка принтера (и неготовность к работе)
16	31	O	INIT, сброс содержимого памяти принтера
17	36	O	SLCT IN, предупреждение перед передачей данных
18–25	остальные	–	GND, заземление

Таблица 8.

Назначение битов регистров параллельного порта

Регистр	Бит	Назначение
BASE	0 – 7	Данные
BASE+1	3	ERROR, ошибка принтера
	4	SLCT, принтер включен
	5	PE, конец бумаги
	6	ACK, запрос на прерывание
	7	BUSY, запрет передачи данных по неготовности
BASE+2	0	STROBE, стробирующий сигнал
	1	AUTO FD, перевод строки на принтере
	2	INIT, сброс текущего состояния принтера
	3	SLCT IN, предупреждение о передаче данных
	4	разрешение прерывания

Управление и передача данных осуществляется через три регистра интерфейса, объемом по одному байту каждый, с адресами BASE (регистр данных), BASE+1 (регистр состояния), BASE+2 (регистр команд). Стандартная конфигурация компьютера предусматривает

два параллельных порта: LPT1 и LPT2. Базовые адреса (BASE) интерфейсов управления находятся в BIOS и при загрузке копируются в оперативную память: в ячейку 408_{16} для LPT1 и $40A_{16}$ для LPT2. Как правило, это 378_{16} для LPT1 и 278_{16} для LPT2. Назначение битов этих регистров приведены в табл. 8.

Процедура передачи данных на порт состоит из следующих шагов:

1. Передача данных в регистр данных.
2. Проверка готовности (отсутствие сигнала BUSY). Зацикливается до появления сообщения о готовности принтера.
3. По получению готовности – запись сигнала STROBE (начало передачи), затем – его сброс (конец передачи).

Ввиду большого числа операций, необходимых для передачи даже одного байта данных, скорость обмена через параллельный порт относительно невысока и редко превышает 100–150 КБ/с.

Последовательные порты

Основная функция последовательных портов (стандарт RS232C, обычно обозначаются COM) – подключение стандартных внешних устройств (модема, мыши, некоторых видов принтеров и сканеров), а также связи компьютеров между собой. Основное отличие от параллельного порта – передача данных в виде последовательных серий импульсов (битов); при этом каждый байт «обрамляется» стартовым и стоповым битами. В результате сильно упрощается конструкция соединительного кабеля (в некоторых случаях достаточно трехпроводной линии), увеличивается (до десятков метров) расстояние, на котором возможна связь, но заметно снижается скорость передачи и усложняется управление.

Стандартная конфигурация компьютера предполагает возможность установки до четырех последовательных портов. На компьютере имеются 25-контактные (DB25P) или (чаще) 9-контактные (DB9P) разъемы этих портов. Назначение их контактов показано в табл. 9.

Таблица 9.

Назначение контактов последовательного порта

№ контакта DB9P	№ контакта DB25P	I/O	Наименование и назначение сигнала
1	–	–	PG, защитное заземление (экран)

2	3	O	TD, передача данных из компьютера
3	2	I	RD, прием данных компьютером
4	7	O	RTS, запрос на передачу данных
5	8	I	CTS, сброс для передачи, готовность к приему
6	6	I	DSR, готовность приемника
7	5	–	SG, схемное заземление, сигнальный нуль
8	1	I	DCD, детектирование приема сигнала
20	4	O	DTR, готовность передатчика
22	9	I	RI, индикатор вызова

При передаче данных через последовательный порт каждому байту данных предшествует старт-бит (логический ноль), сигнализирующий о начале передачи, затем – биты данных, в некоторых случаях – бит контроля четности, и завершает передачу стоп-бит. Следующий старт-бит может посылаться через произвольный интервал времени; старт-биты обеспечивают синхронизацию приемника по сигналам передатчика. При этом подразумевается, что скорости приема и передачи битов совпадают. Для контроля скорости передачи используется внутренний счетчик приемника, который генерирует последовательность стробирующих импульсов, запуская ее в момент получения старт-бита. Такой механизм приема накладывает высокие требования на стабильность частоты приема-передачи и форму фронтов передаваемых импульсов, что резко ограничивает возможную частоту обмена.

Интерфейс каждого последовательного порта имеет восемь 8-разрядных регистров. Их адреса: для COM1 $3F8_{16}$ – $3FF_{16}$, для COM2 $2F8_{16}$ – $2FF_{16}$, для COM3 $3E8_{16}$ – $3EF_{16}$, для COM4 $2E8_{16}$ – $2EF_{16}$. Интерфейсные устройства параллельных портов осуществляют преобразование параллельного кода в последовательный и наоборот, формирование обрамляющих битов и контроль их правильности при приеме данных, прием и передачу данных, формирование управляющих сигналов и контроль сигналов состояния внешних устройств.

На примере COM1 рассмотрим назначение отдельных битов этих регистров (у остальных портов они аналогичны). Начнем с $3FB_{16}$ – это управляющий регистр, и его содержимое определяет назначение остальных.

$3F8_{16}$ – регистр данных

Если старший бит управляющего регистра $3FB_{16}$ установлен в нуль, то этот регистр используется для чтения/записи данных. Если

старший бит регистра $3FB_{16}$ установлен в единицу, то содержимое этого регистра – младший байт кода частоты передачи данных (см. далее).

$3F9_{16}$ – регистр управления прерываниями

Если старший бит регистра $3FB_{16}$ установлен в нуль, то этот регистр используется для управления прерываниями.

№ бита	Назначение
0	1 – разрешение прерывания по окончании приема 0 – запрещение прерывания
1	1 – разрешение прерывания по окончании передачи 0 – запрещение прерывания
2	1 – разрешение прерывания при сбое 0 – запрещение прерывания
3	1 – разрешение прерывания при изменении управляющих сигналов 0 – запрещение прерывания

Если старший бит регистра $3FB_{16}$ установлен в единицу, то содержимое этого регистра – старший байт кода частоты передачи данных:

Код	Частота (бит/с)	Код	Частота (бит/с)
0410_{16}	110	0018_{16}	4800
0300_{16}	150	$000C_{16}$	9600
0180_{16}	300	0006_{16}	19200
$00C0_{16}$	600	0003_{16}	38400
0060_{16}	1200	0002_{16}	57600
0030_{16}	2400	0001_{16}	115200

$3FA_{16}$ – регистр идентификации прерывания

№ бита	Назначение
0	1 – нет прерывания 0 – есть прерывание
1, 2	Причина (тип) прерывания: 00 –сбой, ошибка 01 –окончание передачи 10 – окончание приема 11 – изменение управляющих сигналов

$3FB_{16}$ – управляющий регистр

№ бита	Назначение
0, 1	Количество бит в группе передаваемых данных 00 – 5, 01 – 6, 10 – 7, 11 – 8
2	Количество стоп-битов: 0 – один стоп-бит, 1 – два
3, 4	Метод контроля четности: 00 или 01 – нет контроля, 10 – контроль на нечетность, 11 – контроль на четность

5	Установка контрольного бита: 1 – контрольный бит всегда равен 0 при контроле на четность или 1 при контроле на нечетность
6	1 – передача нуль-сигнала, 0 – нормальная работа
7	Бит, управляющий назначением остальных регистров

$3FD_{16}$ – регистр состояния линии (принимающего устройства)

№ бита	Назначение
0	данные получены компьютером
1	переполнение
2	ошибка четности
3	ошибка синхронизации (нет стоп-бита)
4	получен запрос на прерывание передачи (постоянный нуль-сигнал)
5	готовность принять следующий байт
6	передача закончена
7	перерыв в передаче

Использование остальных регистров определяется типом и характеристиками внешнего устройства, с которым осуществляется связь.

Порт (шина) USB

USB (Universal Serial Bus — универсальная последовательная шина) является промышленным стандартом расширения архитектуры PC, ориентированным на интеграцию с телефонией и устройствами бытовой электроники. Версия стандарта 1.0 была опубликована в начале 1996 года, большинство устройств поддерживают стандарт 1.1, который вышел осенью 1998 года, — в нем были устранены обнаруженные проблемы первой редакции. Весной 2000 года опубликована спецификация USB 2.0, в которой предусмотрено 40-кратное увеличение пропускной способности шины. Первоначально (в версиях 1.0 и 1.1) шина обеспечивала две скорости передачи информации: *полная скорость, FS* (full speed) — 12 Мбит/с и *низкая скорость, LS* (low speed) — 1,5 Мбит/с. В версии 2.0 определена еще и *высокая скорость, HS* (high speed) — 480 Мбит/с, что позволяет существенно расширить круг устройств, подключаемых к шине. В одной и той же системе могут присутствовать и одновременно работать устройства со всеми тремя скоростями. Шина позволяет с использованием промежуточных хабов соединять устройства, удаленные от компьютера на расстояние до 25 м. Подробную информацию по USB можно найти на сайте http://www.usb.org. Разработку устройств и их классификацию и стандартизацию координирует *USB-IF* (USB Implementers Forum, Inc.).

Шина USB обеспечивает обмен данными между хост-компьютером и множеством периферийных устройств (ПУ). USB является единой централизованной аппаратно-программной системой массового обслуживания множества устройств и множества прикладных программных процессов. Связь программных процессов со всеми устройствами обеспечивает хост-контроллер с многоуровневой программной поддержкой. Этим USB существенно отличается от традиционных периферийных интерфейсов (портов LPT, COM, GAME, клавиатуры, мыши и т. п.), сравнение этих типов подключений приводится в табл. 10.

Архитектура USB допускает четыре базовых *типа передач* данных между хостом и периферийными устройствами:

Изохронные передачи (isochronous transfers) — потоковые передачи в реальном времени, занимающие предварительно согласованную часть пропускной способности шины с гарантированным временем задержки доставки. На полной скорости (FS) можно организовать один канал с полосой до 1,023 Мбайт/с (или два по 0,5 Мбайт/с), заняв 70 % доступной полосы (остаток можно занять и менее емкими каналами). На высокой скорости (HS) можно получить канал до 24 Мбайт/с (192 Мбит/с). Надежность доставки не гарантируется – в случае обнаружения ошибки изохронные данные не повторяются, недействительные пакеты игнорируются. Шина USB позволяет с помощью изохронных передач организовывать синхронные соединения между устройствами и прикладными программами. Изохронные передачи нужны для потоковых устройств: видеокамер, цифровых аудиоустройств (колонки USB, микрофон), устройств воспроизведения и записи аудио- и видеоданных (CD и DVD). Видеопоток (без компрессии) шина USB способна передавать только на высокой скорости.

Прерывания (interrupts) – передачи спонтанных сообщений, которые должны выполняться с задержкой не более, чем того требует устройство. Предел времени обслуживания устанавливается в диапазоне 10–255 мс для низкой и 1–255 мс для полной скорости. На высокой скорости можно заказать и 125 мкс. Доставка гарантирована, при случайных ошибках обмена выполняется повтор, правда, при этом время обслуживания увеличивается. Прерывания используются, например, при вводе символов с клавиатуры или для передачи сообщений о перемещениях мыши. Прерываниями можно передавать данные, и к устройству (как только устройство сигнализирует о потребности в данных, хост своевременно их передаёт). Размер сообщения может составлять 0–8 байт для низкой скорости, 0–64 байт – для полной и 0–1024 байт – для высокой скорости передачи.

Таблица 10.

Сравнение шины USB с традиционными интерфейсами

Традиционные интерфейсы (COM, LPT, Game...)	Шина USB
Подключение каждого устройства в общем случае требует присутствия собственного контроллера (адаптера)	Все устройства подключены через один хост-контроллер
Каждый контроллер занимает свои ресурсы (области в пространстве памяти, ввода/ вывода, а также запросы прерывания)	Ресурсы занимает только хост-контроллер
Малое количество устройств, которые возможно одновременно подключить к компьютеру	Возможность подключения до 127 устройств
Драйверы устройств могут обращаться непосредственно к контроллерам своих устройств, независимо друг от друга	Драйверы устройств обращаются только к общему драйверу хост-контроллера
Независимость драйверов оборачивается непредсказуемостью результата одновременной работы с множеством устройств, отсутствием гарантий качества обслуживания (возможность задержек и уменьшения скорости передачи) для различных устройств	Централизованный планируемый обмен обеспечивает гарантии качества обслуживания, что позволяет передавать мультимедийные изохронные данные наряду с обычным асинхронным обменом
Разнообразие интерфейсов, разъемов и кабелей, специфичных для каждого типа устройств	Единый удобный и дешевый интерфейс для подключения устройств всех типов. Возможность выбора скорости работы устройства (1,5–15–480 Мбит/с) в зависимости от потребности
Отсутствие встроенных средств обнаружения подключения/отключения и идентификации устройств, сложность поддержки PnP	Возможность «горячего» подключения/ отключения устройств, полная поддержка PnP, динамическое конфигурирование
Отсутствие средств контроля ошибок	Встроенные средства обеспечения надежной передачи данных
Отсутствие штатного питания устройств	Возможность питания устройств от шины, а также наличие средств управления энергопотреблением

Передачи массивов данных (bulk data transfers) — это передачи без каких-либо обязательств по своевременности доставки и по скорости. Передачи массивов могут занимать всю полосу .пропускания шины, свободную от передач других типов. Приоритет этих передач самый низкий, они могут приостанавливаться при большой загрузке шины. Доставка гарантированная — при случайной ошибке выполняется повтор. Передачи массивов уместны для обмена данными с принтерами, сканерами, устройствами хранения и т. п.

Управляющие передачи (control transfers) используются для конфигурирования устройств во время их подключения и для управления устройствами в процессе работы. Протокол обеспечивает гарантированную доставку данных и подтверждение устройством успешности выполнения управляющей команды. Управляющая передача позволяет подать устройству команду (запрос, возможно, и с дополнительными данными) и получить на него ответ (подтверждение или отказ от выполнения запроса и, возможно, данные). Только управляющие передачи на USB обеспечивают синхронизацию запросов и ответов; в остальных типах передач явной синхронизации потока ввода с потоком вывода нет.

Аппаратная часть USB включает:

Периферийные устройства USB, несущие полезные функции (USB-functions).

Хост-контроллер (Host Controller), обеспечивающий связь шины с центром компьютера, объединенный с *корневым хабом* (Root Hub), обеспечивающим точки подключения устройств USB. Существует два варианта хост-контроллеров USB 1.x – UHC (Universal Host Controller) и OHC (Open Host Controller), поддерживающие скорости FS/LS; высокую скорость шины USB 2.0 (HS и только) поддерживает EHC (Enhanced Host Controller).

Хабы USB (USB Hubs), обеспечивающие дополнительные точки подключения устройств.

Кабели USB, соединяющие устройства с хабами.

Программное обеспнечение (ПО) USB включает:

Клиентское ПО (CSw, Client Software) — драйверы устройств USB, обеспечивающие доступ к устройствам со стороны прикладного ПО. Эти драйверы взаимодействуют с устройствами только через программный интерфейс с общим драйвером USB (USBD). Непосредственного обращения к каким-либо регистрам аппаратных средств драйверы устройств USB не выполняют.

Драйвер USB (USBD, USB Driver), «заведующий» всеми USB-устройствами системы, их нумерацией, конфигурированием,

предоставлением служб, распределением пропускной способности шины, мощности питания и т. п.

Драйвер хост-контроллера (HCD, Host Controller Driver), преобразующий запросы ввода/вывода в структуры данных, размещенные в коммуникационной области оперативной памяти, и обращающийся к регистрам хост-контроллера. Хост-контроллер выполняет физические транзакции, руководствуясь этими структурами данных.

Драйверы USBD и HCD составляют хост-часть ПО USB; спецификация USB очерчивает круг их задач, но не описывает интерфейс между ними.

Физическое устройство USB должно иметь интерфейс USB, обеспечивающий полную поддержку протокола USB, выполнение стандартных операций (конфигурирование и сброс) и предоставление информации, описывающей устройство. Физические устройства USB могут быть *комбинированными* (compound devices): включать в себя несколько устройств-функций, подключенных к внутреннему хабу, а также предоставлять своим внутренним хабом дополнительные внешние точки подключения.

Работой всех устройств шины USB управляет *хост-контроллер* (host controller), являющийся программно-аппаратной подсистемой *хост-компьютера.* Хост-контроллер является интеллектуальным устройством шины PCI или составной частью хаба (моста) системной платы, интенсивно взаимодействующим с оперативной памятью.

Физическая топология шины USB — многоярусная звезда. Ее вершиной является хост-контроллер, объединенный с *корневым хабом* (root hub). Хаб является устройством-разветвителем, он может служить и источником питания для подключенных к нему устройств. К каждому порту хаба может непосредственно подключаться периферийное устройство или промежуточный хаб; шина допускает до пяти уровней (ярусов) каскадирования хабов (не считая корневого). Поскольку комбинированные устройства содержат внутри себя хаб, их подключение к хабу пятого яруса уже недопустимо. Каждый промежуточный хаб имеет несколько *нисходящих* (downstream) портов, для подключения периферийных устройств (или нижележащих хабов) и один *восходящий* (upstream) порт для подключения к корневому хабу или нисходящему порту вышестоящего хаба.

Логическая топология USB — звезда. Хабы (включая корневой) создают иллюзию непосредственного подключения каждого логического устройства к хост-контроллеру. В этой звезде устанавливаются сугубо подчиненные отношения по системе опроса-ответа: хост-контроллер по своей инициативе передает данные к выбранному устройству или принимает их. Устройство по своей инициативе передавать данные не может; непосредственные передачи

данных между устройствами невозможны. Устройство по своей инициативе может лишь сигнализировать о *«пробуждении»* (wakeup), для чего используется специальная сигнализация, но не передача данных.

Физический интерфейс USB прост и изящен. Информационные сигналы и напряжение питания устройств (5 В) передаются по четырехпроводному кабелю. На кабелях используются два типа разъемов. Разъем типа A предназначен для подключения к восходящим (лежащим по линии ближе к компьютеру) коммутаторам, и обычно устанавливаются на кабелях, не отсоединяемых от устройств. Разъем типа B устанавливается на устройствах, от которых соединительный кабель может отсоединяться, и предназначен для подсоединения к нисходящим устройствам. Разъемы отличаются механически, что исключает их неправильное подключение. Конструкция разъемов обеспечивает позднее соединение и раннее отсоединение сигнальных линий по сравнению с линиями питания. Хабы и устройства обеспечивают возможность «горячего» подключения и отключения с сигнализацией об этих событиях хосту. Назначение контактов разъемов: 1 – питание (+5 В), 2 и 3 – линии передачи данных (обычно обозначаются D– и D+), 4 – заземление.

При планировании соединений следует учитывать способ питания устройств: устройства, питающиеся от шины, как правило, подключают к хабам, питающимся от сети. К хабам, питающимся от шины, подключают лишь маломощные устройства – так, к клавиатуре USB, содержащей внутри себя хаб, подключают мышь USB и другие устройства-указатели (трекбол, планшет).

Логическое устройство USB представляет собой набор независимых *конечных точек* (Endpoint, EP – аналог регистров интерфейса внешнего устройства), с которыми хост-контроллер (и клиентское ПО) обменивается информацией. Каждому логическому устройству USB (как функции, так и хабу) конфигурационная часть ПО хоста назначает свой адрес (1–127), уникальный на данной шине USB. Каждая конечная точка логического устройства идентифицируется своим номером (0–15) и направлением передачи *(IN* – передача к хосту, *OUT* – от хоста). Набор конечных точек зависит от устройства, но всякое устройство USB обязательно имеет *двунаправленную конечную точку* 0 *(EP0),* через которую осуществляется его общее управление. Для прикладных целей используются конечные точки с номерами 1–15 (1–2 для низкоскоростных устройств). Адрес устройства, номер и направление конечной точки однозначно идентифицируют приемник или источник информации при обмене хост-контроллера с устройствами USB. Каждая конечная точка имеет набор характеристик, описывающих поддерживаемый тип передачи данных (изохронные данные, массивы,

прерывания, управляющие передачи), размер пакета, требования к частоте обслуживания.

Литература

5. Гук М. Аппаратные средства IBM PC. Энциклопедия. / М. Гук – СПб, Питер, 2006. – 1072 с.
6. Гук М. Шины PCI, USB и FireWire. Энциклопедия. / М. Гук – СПб, Питер, 2005. – 544 с.
7. Гук М. Аппаратные интерфейсы ПК. Энциклопедия. / М. Гук – СПб, Питер, 2003. – 528 с.

Модуль 2.
Устройства сопряжения ЭВМ и экспериментальных установок. Оперативная обработка данных эксперимента

Раздел 3.
Устройства сопряжения ЭВМ и экспериментальных установок

Лекция 7.
Модульный интерфейс КАМАК. Часть 1

План лекции.
Введение.
Историческая справка.
Основные структуры КАМАК.
Виды модулей КАМАК.

Система КАМАК

С целью обмена данными между контрольно-измерительными приборами и компьютерами было разработано большое количество специализированных интерфейсных устройств. Однако при их использовании в физическом эксперименте возникают следующие проблемы. Во-первых, конфигурация ЭВМ (в частности, IBM PC) предусматривает подключение ограниченного количества интерфейсных устройств, тогда как современные экспериментальные установки требуют работы с достаточно большим числом разнородных каналов связи. Во-вторых, проведение экспериментальных исследований предполагает достаточно частое

изменение конфигурации установки. Использование же разнородных интерфейсов приводит к тому, что каждая такая реконструкция измерительной части установки повлечет за собой радикальную переработку аппаратного и программного обеспечения ее автоматизации. В связи с этим широкое распространение получили модульные интерфейсные системы, примером которых, принятым Академией наук в качестве внутреннего стандарта, является система КАМАК.

Историческая справка

В 60-х годах развитие вычислительной техники привело к необходимости стандартизировать не только размеры модулей, но и каналы связи между ними.

Система КАМАК (CAMAC – Computer Automated Measurement And Control) была разработана и предложена совместно европейским комитетом ESONE – European Standards of Nuclear Electronics и американским комитетом US NIM. Она представляет собой модульную систему, предназначенную для связи измерительных устройств с цифровой аппаратурой обработки данных (в большинстве случаев его роль выполняет компьютер). Первоначально концепция системы была принята пятью лабораториями, находившимися во Франции, ФРГ, Италии, Бельгии, Нидерландах, с целью выбора общего стандарта и определения общих потребностей. В 1966 г. началась работа над системой стандартов КАМАК специалистами ведущих европейских институтов для оснащения сложных экспериментов, например, на ускорителях атомных частиц. К этому времени в комитет уже входило 26 лабораторий, включая лаборатории Швейцарии, Англии Австрии и Югославии. С 1974 г. членами комитета являются 29 лабораторий и организаций, среди них: ZERN, Hauell, Saclay, Grenoble, DESI, Frascatti.

В течение 1967–1970 гг. рабочие группы комитета разработали подробные спецификации и выпустили основные стандарты:

1. Стандарт EUR 4100e (1969 г.), в котором рассматриваются конструктивы, источники питания, цифровые сигналы и магистраль крейта;
2. Стандарт EUR 4600e (1970 г.) – предварительный вариант; охватывает магистраль ветви и контроллер типа А;
3. Стандарт EUR 5100e (1970 г.) – предварительный вариант; распространяется на аналоговые сигналы.

В 1972 г. документ EUR 4100e был пересмотрен и дополнен новыми требованиями и рекомендациями, вытекающими из опыта применения системы.

Через три–четыре года после публикации стандарта десятки фирм в разных странах выпускали модули КАМАК более трехсот типов как для экспериментов, так и для контроля и управления технологическими процессами на производствах. В Советском Союзе специалисты в экспериментальной ядерной физике сразу же оценили новую систему, и уже в 1970 году ведущие лаборатории: Объединенный институт ядерных исследований, Ленинградский и Новосибирский институты ядерной физики – начали разработку модулей в стандарте КАМАК для своих нужд. В СССР так же выпускалась модульная аппаратура «Вектор» с логическим протоколом КАМАК, но в модулях собственной конструкции. Несоответствие размеров модулей международным стандартам МЭК-297 и EUR-6100 привело к тому, что «Вектор» не выдержал конкуренции с системой КАМАК и более ста типов разработанных модулей были практически не востребованы. Наличие двух типов модульных систем: КАМАК и Вектор – затянуло стандартизацию системы КАМАК в СССР (соответствующий ГОСТ 26.201-80 был принят лишь в 1980 году в чрезвычайно искаженном виде) и воспрепятствовала промышленному выпуску аппаратуры КАМАК, которую производили лишь институты ядерных исследований, а в основном закупали в Польше и Венгрии.

Создатели системы КАМАК в конце 60-х годов начали применять только что появившиеся интегральные микросхемы в своих разработках, однако у них не хватило смелости предположить, что в 1972 году в электронике начнется революция – появится микропроцессор. Неудобства магистрали КАМАК заставили электронщиков искать решения, позволяющие эффективно использовать качественно новую интегральную схему. Введение микропроцессоров в модули превращало их в микрокомпьютеры, а крейты – в многопроцессорные системы, которые нуждаются в емкой памяти с большим количеством адресов. 16 адресов в модуле КАМАК оказались совершенно недостаточными, поэтому ведущие электронные фирмы Motorola и Intel к середине 70-х годов создали модульные системы третьего поколения Versabus Module Europe bus (VME bus) и Multibus, магистрали которых содержали 16, а затем и 20 адресных линий, что обеспечивало емкость системы примерно 1 млн адресов. В дальнейшем появились и другие модульные системы (PXI, VXI, Multibus-II, и GPIB), которые обладают большей пропускной способностью шины данных и более высоким быстродействием.

Электронные модульные системы долговременны. Если модули достаточно широко распространились и их количество превзошло некоторый критический уровень, то даже не самую современную аппаратуру оказывается выгодным эксплуатировать. Изменить устоявшиеся стандарты

модулей достаточно сложно, и они продолжают использоваться в рамках своих возможностей. Такая ситуация сложилась и с системой КАМАК: несмотря на то, что ей уже более 30 лет, она все еще широко используется как с персональными ЭВМ, так и с микропроцессорами, встроенными непосредственно в контроллер. Большой парк накопившихся разнообразных модулей позволяет в течение нескольких дней, а то и часов, скомпоновать систему с новыми характеристиками.

Основные структуры системы

Основой системы КАМАК является *крейт* с 25 ячейками (*станциями*), в которые по направляющим вставляются модули, включая контроллер. Модуль может занимать одну или несколько ячеек. Обмен данными между модулями происходит по внутренним шинам крейта – горизонтальной магистрали и организуется контроллером.

В электронной системе модулем является печатная плата с узкой передней панелью и плоским многоконтактным разъемом на противоположной стороне платы. Модули вставляют в каркас с направляющими, в которых скользит плата. Задняя стенка каркаса выполнена также в виде платы с ответными частями разъемов, которые соединены печатными или навесными проводниками, образующими электрические линии магистрали для передачи кодированной информации. По специально назначенным проводникам в модули подается электрическое питание.

Все размеры модулей и каркасов строго стандартизованы: габариты и толщина печатных плат модулей, ширина канавок в направляющих и расстояние между ними, расположение контактов и так далее. Определены длительности и амплитуды электрических сигналов, а также напряжения питания модулей. Стандартизованы не только размеры, но и логический протокол – правила передачи информации по линиям магистрали

Существует несколько конфигураций системы, обусловленных выбранным способом управления крейтом и организацией его связи с управляющей ЭВМ.

1. АВТОНОМНАЯ СИСТЕМА КАМАК. Возможны два основных варианта ее реализации. Первый – на основе программируемого управления крейтом с использованием простого ориентированного на пользователя языка для организации часто проводимых операций, таких, как например периодический опрос содержимого счетчиков импульсов и передача результатов счета в память. Второй вариант автономной системы базируется на управлении крейтом программируемым компьютером, который встраивают в контроллер крейта (микропроцессорный контроллер) или в

качестве вспомогательного контроллера размещают в одной из ячеек крейта и связывают со стандартным контроллером крейта дополнительной магистралью.

Непосредственное управление от компьютера в этом случае не предусмотрено, однако допускается обмен с внешней ЭВМ – передача ей сжатых данных измерений и прием системной информации для управления.

2. ПОДСИСТЕМА КАМАК. В системах с компьютером, в которых используется один или несколько крейтов, может оказаться целесообразным выход контроллера крейта непосредственно связывать с каналом ввода-вывода данных машин. Преимущества такого подхода – низкие затраты на сопряжение. Из-за условий ограниченности длины линий связи крейты в этом случае необходимо размещать непосредственно вблизи машины, а для каждого крейта должен быть выделен отдельный программно управляемый канал ввода-вывода данных компьютера.

3. СИСТЕМА С ВЕРТИКАЛЬНОЙ МАГИСТРАЛЬЮ. Обмен информацией между несколькими крейтами (максимально до семи) и компьютером может осуществляться через так называемую вертикальную магистраль с параллельной передачей данных. Подобная структура из-за больших затрат на организацию кабельной магистрали с параллельными линиями оказывается целесообразной для средних и больших пространственно ограниченных систем. Скорость передачи данных в магистрали может превышать 10^7 бит/с. При определенных условиях к компьютеру может быть присоединено несколько вертикальных магистралей.

3. ПРОСТРАНСТВЕННО-РАСПРЕДЕЛЕННАЯ СИСТЕМА. Для систем, элементы которых удалены друг от друга на значительные расстояния, используется канал с последовательной передачей данных между компьютером и крейтами. Канал представляет собой однонаправленную замкнутую цепь (кольцевую магистраль), в которую последовательно друг за другом включают до 62 крейтов. Двоичные данные передаются поразрядно или пословно (побайтно) со скоростью, обусловленной характеристиками каналов связи. Так, например, при использовании телефонных линий связи скорость передачи оказывается существенно меньше в сравнении с параллельной передачей данных в вертикальной магистрали.

Виды модулей КАМАК

1. Счетчики.
2. Регистры ввода-вывода.
3. Таймеры.

4. Приводы.
5. Интерфейсы.
6. Блоки цифровой обработки сигналов:
    6.1. Логические преобразователи.
    6.2. Преобразователи кодов.
    6.3. Память.
    6.4. Специализированные процессоры.
7. Блоки аналоговой обработки сигналов:
    7.1. Преобразователи напряжений.
    7.2. Аналого-цифровые преобразователи.
    7.3. Цифро-аналоговые преобразователи.
    7.4. Мультиплексоры.
    7.5. Ключи.
    7.6. Прочие.
8. Контрольное оборудование:
    8.1. Индикатор магистрали.
    8.2. Ручной контроллер.

Литература

9. Певчев Ю. Ф. Автоматизация физического эксперимента / Ю. Ф. Певчев, К. Г.Финогенов – М.: Энергоатомиздат, 1986. – 254 с.
10. Курочкин С. С. Системы КАМАК–ВЕКТОР / С. С. Курочкин – М.: Радио и связь., 1981. – 160 с.
11. CAMAC — A Modular Instrumentation System for Data Handling. Revised Description and Specification. Report. EUR 4100e, CEC, Luxembourg 1972; deutsche Ubersetzung; EUR 4100d; revised form: IEEE Standard 583-1975, IEC Recommendation 516; Bloc Transfers in CAMAC Systems, Supplement EUR 4100e, CEC, Luxembourg 1977; IEEE Standard 683-1976.
12. CAMAC — Organization of Multi-Crate System. Specification of the Branch Highway and CAMAC Crate Controller Typ A, Report EUR 4600e, CEC, Luxembourg 1972; revised form: IEEE Standard 596-1976.
13. CAMAC — A Modular Instrumentation System for Data Handling, Specification of Amplitude Analogue Signals, Report EUR 5100e, CEC, Luxembourg 1972; CAMAC Bulletin No. 8, 28 (1973).
14. Государственный стандарт Союза ССР. Единая система стандартов приборостроения. Система КАМАК. Крейт и сменные блоки. Требования к конструкции и интерфейсу. ГОСТ 26.201–80.

15. Науман Г. Стандартные интерфейсы для измерительной техники / Г. Науман, В. Майлинг, А. Щербина – М.: Мир, 1982. – 230 с.

Лекция 8.
Модульный интерфейс КАМАК. Часть 2

План лекции.
Организация горизонтальной магистрали (шины КАМАК)
Линии N, A, F
Данные записи W и считывания R
Сигналы состояния B, X, Q
Запрос на прерывание L
Общие сигналы управления Z, C, I

Организация горизонтальной магистрали (шины КАМАК)

Горизонтальная магистраль состоит из 86 линий, каждый из которых соединен с контактом приемной части разъема крейта. Назначение линий:

Команды

5 линий типа N – номер модуля, которому предназначена команда.

4 линии типа A – адрес внутри модуля (в случаях, если модуль объединяет в себе несколько устройств).

5 линий типа F – номер функции, которую следует выполнить.

Состояние

X – готовность.

Q – ответ модуля на запрос.

L – запрос модуля.

B – шина занята.

Управление

Z – пуск.

C – сброс содержимого всех регистров.

I – остановка операций.

S1, S2 – стробирующие сигналы.

Передача

24 линии чтения информации R.

24 линии записи информации W.

Все линии можно разделить на магистральные, соединяющие между собой все соответствующие контакты разъемов станций, и индивидуальные, которые соединяют контакты разъемов нормальной станции с контактами разъемов управляющей станции. К индивидуальным относятся 24 линии

адреса или **N**–линии, по которым сигнал от управляющей станции инициирует работу модуля, и линии запроса **L**, по которым сигнал от модуля выставляет требование на обработку данных, например, по завершению какой-либо операции в модуле.

Остальные линии являются магистральными. Из них 14 линий питания и 17 управляющих проходят через контакты всех 25 станций, 24 линии записи, 24 линии чтения и две дополнительные соединяют контакты разъемов только нормальной станции. По управляющим линиям команды и синхронизирующие импульсы поступают на нормальную станцию, от станции выставляется сигнал о состоянии модуля, о ходе и завершении операции.

По линиям записи данные поступают на разъемы нормальной станции, по линиям чтения данные передаются к месту назначения.

Линии питания обеспечивают различные диапазоны напряжений. В качестве обязательного набора в стандарте приняты напряжения питания ±5 В при максимальном токе 25 А на крейт и 2 А на станцию, и ±24 В при максимальном токе 6 А на крейт и 1 А на станцию. В качестве дополнительного набора приняты напряжения ±12 В, ±200 В и переменное напряжение 117 В. Рассеиваемая мощность не должна превышать 200 Вт в крейте и 8 Вт в станции.

Линии N, A, F

АДРЕС ЯЧЕЙКИ **N**. Каждая ячейка крейта (ячейка модуля) модуля адресуется контроллером по отдельной соответствующей линии выборки **N** (ячейки на лицевой панели крейта нумеруются слева направо от **N** = 1 до **N** = 23). При использовании дополнительного регистра в контроллере может быть организовано параллельное адресное обращение к нескольким ячейкам (модулям) крейта.

СУБАДРЕСА **A**. Один модуль может иметь несколько источников/приемников информации. Это могут быть несколько независимых устройств, собранных в виде единого модуля, либо единое устройство, разделенное на аппаратные или логические блоки (регистры). Обращение к одному из них задается кодом субадреса. Четыре субадресные линии для передачи двоичного кода (линии **A8**, **A4**, **A2**, **A1**) позволяют выбирать по адресу до 16 различных узлов и частей внутри самого модуля. Адрес узла можно задавать любым от **A(0)** до **A(15)**.

ФУНКЦИИ (ОПЕРАЦИИ) **F**. Каждый элемент модуля может выполнять до 32 различных операций. Для задания одной из функций от **F(0)** до **F(31)**

используют пять функциональных линий **F16**, **F8**, **F4**, **F2** и **F1**. Значения этих функций и выборки указаны в таблице далее.

В выбранном по адресу **N** модуле коды субадреса и функции дешифруются. Допускается также разделение отдельных разрядов кода выборки функций на группы с последующим частичным дешифрированием для выделения дополнительных признаков. Так, например, командами **F16** = 0 и **F16** = 1 разделяют функции чтения и записи соответственно.

Таблица 11.

Команды системы КАМАК

Номер команды	Назначение	F16	F8	F4	F2	F1	Использование линий данных
F(0)	Чтение содержимого регистра группы 1	0	0	0	0	0	Используются линии R
F(1)	Чтение содержимого регистра группы 2	0	0	0	0	1	
F(2)	Чтение и сброс регистра группы 1	0	0	0	1	0	
F(3)	Чтение дополнения содержимому регистра группы 1	0	0	0	1	1	
F(4)	Нестандартная	0	0	1	0	0	
F(5)	Резервная	0	0	1	0	1	
F(6)	Нестандартная	0	0	1	1	0	
F(7)	Резервная	0	0	1	1	1	
F(8)	Контроль требований	0	1	0	0	0	Не используются
F(9)	Сброс регистра группы 1	0	1	0	0	1	
F(10)	Гашение сигнала требования	0	1	0	1	0	
F(11)	Сброс регистра группы 2	0	1	0	1	1	
F(12)	Нестандартная	0	1	1	0	0	
F(13)	Резервная	0	1	1	0	1	
F(14)	Нестандартная	0	1	1	1	0	
F(15)	Резервная	0	1	1	1	1	
F(16)	Перепись содержимого регистра группы 1	1	0	0	0	0	Используются линии W
F(17)	Перепись содержимого регистра группы 2	1	0	0	0	1	
F(18)	Селективная установка регистра группы 1	1	0	0	1	0	
F(19)	Селективная установка регистра группы 2	1	0	0	1	1	
F(20)	Нестандартная	1	0	1	0	0	
F(21)	Селективный сброс регистра группы 1	1	0	1	0	1	
F(22)	Нестандартная	1	0	1	1	0	
F(23)	Селективный сброс регистра группы 2	1	0	1	1	1	
F(24)	Блокирование	1	1	0	0	0	Не используются
F(25)	Исполнение	1	1	0	0	1	
F(26)	Деблокирование	1	1	0	1	0	
F(27)	Проверка состояния	1	1	0	1	1	

F(28)	Нестандартная	1	1	1	0	0
F(29)	Резервная	1	1	1	0	1
F(30)	Нестандартная	1	1	1	1	0
F(31)	Резервная	1	1	1	1	1

При инициализации модуль полностью дешифрует субадрес и команду и подает в магистраль сигнал **X**. При определенных командах модуль может выработать сигнал **Q**. Эти сигналы принимаются контроллером крейта по стробу **S1**.

Для операций чтения определены 4 команды: **F(0)**, **F(1)**, **F(2)**, **F(3)**. По этим командам содержимое регистров, к которым произошло обращение, выставляется на линии чтения R, и по стробу **S1** переписывается в регистр-приемник. Сброс регистра командой **F(2)** происходит по стробу **S2**. Команды **F(4)**, **F(6)** – нестандартные и при разработке модуля могут использоваться их по своему усмотрению. Команды **F(5)**, **F(7)** зарезервированы для дальнейших расширений. Цикл в команде модуля может быть больше цикла КАМАК, в этом случае модуль после окончания операции выработает и выставит на шину **L** запрос.

Для операций записи определены 6 команд **F(16)**, **F(17)**, **F(18)**, **F(19)**, **F(21)**, **F(23)**. По этим командам содержимое регистра-источника (либо преобразованный код регистра-источника) выставляется на линии **W** и по стробу **S1** переписывается в регистр модуля. Команды **F(20)**, **F(22)** нестандартные, т.е. разработчики модулей могут использовать их по своему усмотрению.

Команды **F(9)**, **F(11)** сбрасывают содержимое модуля.

Содержимое регистров 2 группы **A(12)** – регистр состояния, **A(13)** маски – регистр маски, **A(12)** – регистр запроса можно прочитать или заменить командами чтения или записи. При наличии большого количества источников в модуле рекомендуется пользоваться этими командами. В этом случае каждый источник привязан к конкретному разряду регистров состояния **A(12)**, **A(13)**, **A(14)** и наличие запроса от конкретного источника обнаруживается значением соответствующего разряда.

Каждый модуль может генерировать сигнал **L**-запрос на обработку. Линии, по которым передается этот сигнал, являются индивидуальными, как и **N**-линии. Адресуемый модуль не должен выставлять **L**-сигнал до конца текущей операции. Неадресуемый модуль может устанавливать **L**-сигнал в любое время. Когда модуль, который генерирует **L** = 1, принимает команду, заставляющую его устранить этот вызов, он должен запретить **L** сигнал или сбросить **L** запрос.

Команды **F(8)** – **F(15)** шины **R** и **W** не используют. С помощью команды **F(8)** может проверить наличие запроса от конкретного источника адресуясь к соответствующему разряду регистра запроса **A(14)**. Субадрес команды **F(8)** можно интерпретировать как номер разряда регистра **A(14)**. Например, команда **F(8)A(23)** проверяет наличие запроса от источника, который соответствует разряду 23 запроса. Команда вырабатывает ответный сигнал

Q = 0, если разряд в состоянии 0 и **Q** = 1, если разряд в состоянии 1. Команда запрос не сбрасывает.

В качестве примера можно привести следующую часто возникающую ситуацию. Например в работе используется модуль «таймер», который должен отмерять длительные (по сравнению с собственными временами обработки команд КАМАК) промежутки времени. Необходимо приостановить работу программы до окончания заданного промежутка времени. В момент окончания промежутка времени модуль «таймер» выставляет на шины сигнал **Q**. В этом случае программируется цикл состоящий из запроса **F(8)** к интересующему нас модулю. Условием выхода из цикла является получение сигнала **Q**.

Команда **F(10)** сбрасывает запрос от источника, указанного в субадресе команды. При наличии регистра запроса **A(12)** эквивалентна сбросу соответствующего разряда регистра.

Команды **F(24)** – **F(31)** шины **R** и **W** не используют. Команда **F(24)** запрещает какую-либо функцию модуля или маскирует **L** сигнал. Элемент модуля, функции которого запрещаются, задается субадресом команды, при наличии регистра маски **A(13)**. Действие команды начинается по **S1** или **S2**.

Команда **F(25)** инициирует исполнение какой-либо функции, ее начало или окончание. Команда используется, если команды **F(24)** и **F(26)** непригодны. Элемент, который инициализируется командой, задается субадресом команды. Субадрес может интерпретироваться как задание конкретного действия из множества возможных действий. Действие может начинаться по **S1** или **S2**. Например в модуле «счетчик $СЧ_6$10» эта команда используется для увеличения на единицу количества отсчетов.

Команда **F(26)** разрешает какую-либо функцию элемента или снимает маску **L**-сигнала. При наличии регистра маски выполнение команды эквивалентно установке соответствующего разряда регистра **A(13)**. Эта команда обратная к команде **F(24)**. Действие начинается по **S1** или **S2**.

Команда **F(27)** вырабатывает на **Q**-линии ответ, соответствующей состоянию выбранной части модуля по субадресу команды. Характеристика, которая выбирается субадресом, может статусной, что при наличии регистра состояния **A(12)** эквивалентно проверке соответствующего разряда **A(12)**.

Команды **F(28)**, **F(30)** не стандартизованы. Команды **F(29)**, **F(31)** зарезервированы для дальнейших расширений.

Данные записи W и считывания R

Информация, передаваемая по линиям чтения **R** или записи **W**, представляет собой коды численных значений данных, сообщения о

состоянии модулей и сигналов или сигналы, которые в модулях используются для управления.

ЛИНИИ ЗАПИСИ W1–W24. По этим линиям контроллер передает модулю 24-разрядный параллельный двоичный код. Необходимыми условиями передачи являются завершение установление передаваемых сигналов до посылки контроллером строб-сигнала **S1** и сохранение сигналов данных неизменными во время действия строб-сигнала **S1**. Использование для этих целей сигнала **S2** допустимо лишь в специальных случаях.

ЛИНИИ ЧТЕНИЯ R1–R24. По этим линиям модуль передает данные контроллеру. Сигналы этих данных должны устанавливаться до появления строб-сигнала **S1** и сохранятся неизменными до завершения операции. Операции по приему данных контроллер выполняет во время существования строб-сигнала **S1**.

Ответ – сигнал занятости **B** = 1 о состоянии данных в линиях **W** и **R** модуль должен формировать только при командных операциях и операциях по передаче данных. Допускается обмен двоичными словами, содержащими меньше 24 разрядов, однако для контроллера рекомендуется использовать всю разрядность.

Сигналы состояния B, X, Q

Сигнал занятости **B** = 1 указывает на выполнение в данное время модулем операции (адресной или безадресной).

Сигнал **X** = 1 о восприятии команды выдает модуль, оповещая тем самым о возможности самостоятельного выполнения этой команды или о ее выполнении совместно со связанным внешним прибором. Сигнал **X** должен быть установлен до появления строб-сигнала **S1**. Во время действия сигнала **S1** выполняется анализ значения **X**, которое должно сохранятся неизменным до завершения строб-сигнала **S2**. Нулевое значение сигнала **X** = 0 указывает на наличие серьезной ошибки, например, на отсутствие самого модуля в ячейке, питания модуля, на отсутствие требуемого внешнего прибора и др. Если в интерфейсной части системы используется процессор, то с получением сигнала **X** = 0 он показывает выполнение текущей программы и начинает реализацию программы диагностики ошибок.

Сигнал подтверждения **Q** используется в ряде случаев. Так, выбранный по адресу модуль во время выполнения им операции посылкой **Q** может сигнализировать о своем соответствующем состоянии. Значение сигнала в линии **Q** контроллер оценивает во время действия строб-сигнала **S1**. При выполнении операций чтения и записи адресованный модуль должен установить нулевое или единичное значение **Q** до появления строб-сигнала

S1 и сохранять его неизменным по крайней мере до завершения действия строб-сигнала **S2**.

Запрос на прерывание L

Сигналом **L** модуль (через контроллер) посылает заявку в процессор о необходимости прервать текущую программу и начать выполнение программы обслуживания этого модуля. Сигнал **L** в контроллер передается по линии выборки **L**, каждая из которых нумеруется идентично номеру соответствующей ячейки: L1–L23.

В модуле может быть несколько источников, требующих обслуживания с прерыванием. Соответствующие сигналы этих источников обозначают LAM. Для идентификации источников можно присваивать им субадреса (при небольшом числе источников) или использовать специальный регистр заявок на обслуживание.

Общие сигналы управления Z, C, I

Сигналы подготовки **Z** и гашения **C** формируются при выполнении безадресных операций при передаче данных и должны воздействовать на все устройства, связанные общими линиями этих сигналов, до появления строб-сигнал **S2**. Эти сигналы требуют одновременного с ними действия сигнала **B** = 1. Сигнал **Z** имеет абсолютный приоритет по отношению ко всем прочим сигналам, и при воздействии **Z** = 1 все регистры сбрасываются в свои начальные состояния. Сигналом **C** = 1 сбрасываются в нулевое состояние только выбранные пользователем регистры и отдельные триггеры.

Сигнал блокировки **I** может быть сформирован в произвольный момент времени, блокируя в модуле реализацию всех предусмотренных функций.

Литература

16. Певчев Ю. Ф. Автоматизация физического эксперимента / Ю. Ф. Певчев, К. Г.Финогенов – М.: Энергоатомиздат, 1986. – 254 с.
17. Курочкин С. С. Системы КАМАК–ВЕКТОР / С. С. Курочкин – М.: Радио и связь., 1981. – 160 с.
18. CAMAC — A Modular Instrumentation System for Data Handling. Revised Description and Specification. Report. EUR 4100e, CEC, Luxembourg 1972; deutsche Ubersetzung; EUR 4100d; revised form: IEEE Standard 583-1975, IEC Recommendation 516; Bloc Transfers in CAMAC Systems, Supplement EUR 4100e, CEC, Luxembourg 1977; IEEE Standard 683-1976.

19. CAMAC — Organization of Multi-Crate System. Specification of the Branch Highway and CAMAC Crate Controller Typ A, Report EUR 4600e, CEC, Luxembourg 1972; revised form: IEEE Standard 596-1976.
20. CAMAC — A Modular Instrumentation System for Data Handling, Specification of Amplitude Analogue Signals, Report EUR 5100e, CEC, Luxembourg 1972; CAMAC Bulletin No. 8, 28 (1973).
21. Государственный стандарт Союза ССР. Единая система стандартов приборостроения. Система КАМАК. Крейт и сменные блоки. Требования к конструкции и интерфейсу. ГОСТ 26.201–80.
22. Науман Г. Стандартные интерфейсы для измерительной техники / Г. Науман, В. Майлинг, А. Щербина – М.: Мир, 1982. – 230 с.

Лекция 9.
Модульный интерфейс КАМАК. Часть 3

План лекции.
Организация установки
Управление крейтом КАМАК от IBM PC: Последовательный интерфейс

Организация установки

Управление работой крейта и, как правило, организация связи крейта с компьютером возлагается на контроллер крейта, который должен занимать управляющую и одну нормальную станцию: они соответствуют двум правым станциям (24, 25). В этом случае он выполняет роль основного контроллера крейта. Возможна ситуация, когда контроллер будет установлен в любую другую станцию крейта. В этом случае он будет выполнять функции вспомогательного контроллера. Такое соединение применяется, если необходимо соединить несколько крейтов. Основной контроллер связан с компьютером, а остальные крейты связаны между собой с помощью вспомогательных контроллеров. Оставшиеся станции занимают исполнительные модули. Они связываются с контролером крейта через магистраль.

Контроллер крейта играет роль управляющего модуля, в большинстве случаев он играет роль пассивного приемника команд КАМАК, которые ему передает компьютер. После получения команды контроллер дешифрует адрес модуля и генерирует сигнал на индивидуальных линиях **N**.

Для подключения контроллера крейта к компьютеру служат специализированные интерфейсные платы, устанавливаемые внутри компьютера и подключаемые к его шине. Применяя различные интерфейсные платы, можно подключать разные компьютеры к одной и той

же установке, при этом необходимо лишь минимально модифицировать программное обеспечение, управляющее процессом обмена данными между процессором и интерфейсной платой.

Данные в контроллер крейта могут передаваться параллельно или последовательно, в зависимости от вида интерфейсной платы.

Управление крейтом КАМАК от IBM PC. Последовательный интерфейс

Интерфейсная плата устанавливается внутри системного блока компьютера в один из разъемов шины ISA (EISA), и работает как последовательный (COM) порт. В качестве базового адреса выступает один из базовых адресов COM порта (какой конкретно – определяется установками перемычек на плате). Контроллер имеет 4 внутренних 16-разрядных регистра: CSR, DMR, DHR и регистр управления магистралью КАМАК (рассмотрение ведется на примере контроллера К-16). Все регистры контроллера – 16-разрядные, поэтому работа с ими осуществляется, как правило, с помощью команд для 16-разрядных слов, и их адреса – четные (нечетные адреса соответствуют младшим байтам этих слов).

Базовый адрес крейта – это адрес управляющего *регистра команд и состояний* (*Command and Status Register, CSR*). Назначение его разрядов:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q	X*		I*	S	X	Z	C	D	DE	I	F16	F8	F4	F2	F1

Разряды 0–4 используются для указания двоичного номера функции, которую необходимо выполнить.

Пятый разряд I вызывает появление сигнала запрета операций на линии I шины контроллера.

Шестой разряд DE разрешает контроллеру осуществлять прерывание работы процессора и требовать обработки прерывания.

Седьмой разряд D – требование такого прерывания, выставляется контроллером.

Восьмой разряд C устанавливается компьютером и вызывает появление сигнала C на шине КАМАК, что приводит к сбросу всех команд и значений на линиях чтения/записи.

Девятый разряд Z устанавливается компьютером и вызывает появление сигнала Z на шине КАМАК; приводит к воспроизведению начальной установки состояния крейта.

Разряд X – разрешение работы с LAM запросами.

Разряд S используется для установки режима стробирования операций КАМАК. В большинстве крейтов его программная установка приводит к запрету выполнения операций.

Разряд I* используется для контроля за состоянием линии I на магистрали крейта. Программно не изменяется, может только считываться.

Разряд 13 не используется.

Разряд X* – предназначен только для считывания; его значение определяется контроллером и информирует о появлении LAM запроса от одной из станций (если работа с этими запросами была разрешена разрядом X). Если при этом была разрешена работа с прерываниями, то контроллер одновременно формирует запрос на прерывание.

Разряд Q выставляется контроллером и информирует об окончании выполнения команды и готовности принять следующую.

Следующий управляющий регистр – *регистр запросов и маски* (*Demand and Mask Register, DMR*). Этот регистр управляет разрешениями прерываний от отдельных станций, и служит для передачи запросов на прерывания. По умолчанию подобные запросы могут выставлять восемь станций (5–12), хотя программно эти назначения могут меняться (функция F(8)). Младший байт регистра служит для выставления разрешений на прерывание для каждой из этих станций (байт маски); старший – для индикации поступивших запросов (байт запросов). При возникновении незамаскированного запроса контроллер выставляет запрос на прерывание в CSR, а по состоянию регистра DMR можно определить, какая из станций послужила источником запроса.

Третий служебный регистр – *регистр старшего байта* (*Data High bite Register, DHR*) – предназначен для приема/передачи старшего третьего байта данных. Появление этого байта обусловлено тем, что шина КАМАК 24-разрядная, тогда как регистры данных станций – 16-разрядные. Не вошедший старший байт данных с шины КАМАК передается в DHR.

Четвертый регистр предназначен для определения адреса станции в крейте и приема/передачи данных. В него сначала направляется адрес станции в виде

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				C4	C2	C1	N16	N8	N4	N2	N1	A8	A4	A2	A1

(разряды C1–C4 используются при работе с несколькими крейтами одновременно) и затем он же используется для приема/передачи слова данных.

Работа с КАМАК представляет собой, в конечном итоге, процессы записи или чтения данных из соответствующих регистров.

Чтение данных из КАМАК:

1. Если операция вызвана прерыванием – установить, какая станция запросила прерывание (по содержимому DMR).
2. Установить одну из функций чтения в младших битах CSR.
3. В четвертом регистре указать, к какой станции относится команда.

4. Прочитать поступившие данные.

Запись данных в КАМАК:

1. Проверить готовность КАМАК к принятию данных (бит Q в CSR должен быть равен нулю, т. е. двоичное число, содержащееся в этом регистре – положительно).
2. Установить одну из функций записи в младших битах CSR.
3. В четвертом регистре указать, к какой станции относится команда.
4. Передать данные в четвертый регистр (при необходимости и в DHR).

Литература

5. Певчев Ю. Ф. Автоматизация физического эксперимента / Ю. Ф. Певчев, К. Г.Финогенов – М.: Энергоатомиздат, 1986. – 254 с.
6. Курочкин С. С. Системы КАМАК–ВЕКТОР / С. С. Курочкин – М.: Радио и связь., 1981. – 160 с.
7. CAMAC — A Modular Instrumentation System for Data Handling. Revised Description and Specification. Report. EUR 4100e, CEC, Luxembourg 1972; deutsche Ubersetzung; EUR 4100d; revised form: IEEE Standard 583-1975, IEC Recommendation 516; Bloc Transfers in CAMAC Systems, Supplement EUR 4100e, CEC, Luxembourg 1977; IEEE Standard 683-1976.
8. CAMAC — Organization of Multi-Crate System. Specification of the Branch Highway and CAMAC Crate Controller Typ A, Report EUR 4600e, CEC, Luxembourg 1972; revised form: IEEE Standard 596-1976.
9. CAMAC — A Modular Instrumentation System for Data Handling, Specification of Amplitude Analogue Signals, Report EUR 5100e, CEC, Luxembourg 1972; CAMAC Bulletin No. 8, 28 (1973).
10. Государственный стандарт Союза ССР. Единая система стандартов приборостроения. Система КАМАК. Крейт и сменные блоки. Требования к конструкции и интерфейсу. ГОСТ 26.201–80.
11. Науман Г. Стандартные интерфейсы для измерительной техники / Г. Науман, В. Майлинг, А. Щербина – М.: Мир, 1982. – 230 с.

Лекция 10.
Модульный интерфейс КАМАК. Часть 4

План лекции.

Управление крейтом КАМАК от IBM PC: Параллельный интерфейс

Управление крейтом КАМАК от IBM PC.
Параллельный интерфейс

В качестве примера интерфейсного устройства параллельного доступа рассмотрим плату производства СП «GEOSOFT».

Контроллер крейта также подключается к шине ISA или EISA IBM PC. Он имеет двенадцать восьмиразрядных регистров интерфейса ввода-вывода.

Линия адреса ввода/вывода соответствует настройке программных средств на диапазон адресов $240_{16} – 24F_{16}$. Назначение регистров приведено в таблице12.

Таблица 12

Назначение регистров интерфейсной платы контроллера КАМАК производства СП «GEOSOFT»

Регистр	Адрес	Разряды								Назначение
		7	6	5	4	3	2	1	0	
0	240_{16}	W24	W24	W22	W21	W20	W19	W18	W17	Запись
1	241_{16}	W16	W15	W14	W13	W12	W11	W10	W9	Запись
2	242_{16}	W8	W7	W6	W5	W4	W3	W2	W1	Запись
3	243_{16}					A8	A4	A2	A1	Субадрес станции
4	244_{16}				F16	F8	F4	F2	F1	Функция
5	245_{16}				N16	N8	N4	N2	N1	Адрес станции
6	246_{16}		AX4	AX3	AX2	AX1	I	C	Z	Адрес крейта и состояние
7	247_{16}									не используется
8	248_{16}	LAM	L16	L8	L4	L2	L1	X	Q	LAM запросы и состояние
9	249_{16}	R24	R23	R22	R21	R20	R19	R18	R17	Чтение
10	$24A_{16}$	R16	R15	R14	R13	R12	R11	R10	R9	Чтение
11	$24B_{16}$	R8	R7	R6	R5	R4	R3	R2	R1	Чтение
12	$24C_{16}$						AO	AL	AC	Состояние контроллера
15	$24F_{16}$									DMA

Разряды AX1–AX4 используются для указания адреса крейта в системах с несколькими последовательно соединенными крейтами, регистр $24C_{16}$ – для контроля состояния интерфейсной платы, $24F_{16}$ – для организации работы режима прямого доступа к памяти. В остальном работа этого контроллера тоже осуществляется аналогично. Программный модуль, реализующий управление этим контролером крейта КАМАК, приведен в Приложении.

Литература

1. Певчев Ю. Ф. Автоматизация физического эксперимента / Ю. Ф. Певчев, К. Г.Финогенов – М.: Энергоатомиздат, 1986. – 254 с.
2. Курочкин С. С. Системы КАМАК–ВЕКТОР / С. С. Курочкин – М.: Радио и связь., 1981. – 160 с.
3. CAMAC — A Modular Instrumentation System for Data Handling. Revised Description and Specification. Report. EUR 4100e, CEC, Luxembourg 1972; deutsche Ubersetzung; EUR 4100d; revised form: IEEE Standard 583-1975, IEC Recommendation 516; Bloc Transfers in CAMAC Systems, Supplement EUR 4100e, CEC, Luxembourg 1977; IEEE Standard 683-1976.
4. CAMAC — Organization of Multi-Crate System. Specification of the Branch Highway and CAMAC Crate Controller Typ A, Report EUR 4600e, CEC, Luxembourg 1972; revised form: IEEE Standard 596-1976.
5. CAMAC — A Modular Instrumentation System for Data Handling, Specification of Amplitude Analogue Signals, Report EUR 5100e, CEC, Luxembourg 1972; CAMAC Bulletin No. 8, 28 (1973).
6. Государственный стандарт Союза ССР. Единая система стандартов приборостроения. Система КАМАК. Крейт и сменные блоки. Требования к конструкции и интерфейсу. ГОСТ 26.201–80.
7. Науман Г. Стандартные интерфейсы для измерительной техники / Г. Науман, В. Майлинг, А. Щербина – М.: Мир, 1982. – 230 с.

Лекция 11.
Модульный интерфейс PXI. Часть 1

План лекции.
Общее описание интерфейса.
Механический стандарт.

Система PXI

Модульная система интерфейсов PXI (PCI extension for instrumentation) была разработана компанией National Instruments с целью объединить знакомое пользователю и потенциальным разработчикам программное и аппаратное обеспечение с разработанными в последние годы технологиями. В качестве механического стандарта использован стандарт интерфейса Евромеханика (EuroCard), разработанного для применений в промышленных условиях, – это обеспечивает механическую надежность системы и устойчивость электрических соединений. Обмен информацией между интерфейсами осуществляется по шине, аналогичной шине PCI с

расширенной адресацией. Это дает возможность установки до семи станций в крейте, и допускает ее дальнейшее расширение с использованием стандартных PCI–PCI мостов.

Использование стандартной шины PCI дает возможность широкого использования существующего программного обеспечения – начиная от средств операционных систем, до драйверов устройств и программ обработки данных и графического представления результатов. PXI легко интегрируется в операционные системы на базе Windows NT: Windows NT4, Windows 2000, Windows XP, Windows Vista. Для нее разработана архитектура VISA (Virtual Instrument Software Architecture), позволяющая объединять крейты PXI с модульными системами VXI и GPIB.

Механический стандарт

На рис. 5 схематично показан вид крейта PXI. Крейт представляет собой шасси, в задней части которого расположены разъемы для подключения станций – контроллера и интерфейсов внешних устройств. Контроллер устанавливается в слот № 1; справа от него располагаются слоты для подключения интерфейсных модулей (до семи для крейтов с тактовой частотой 33 МГц, и до 3 – для 66 МГц), слева могут располагаться дополнительные слоты для установки дополнительных плат-расширений контроллера.

Стандарт PXI допускает два типоразмера интерфейсных модулей (рис. 6) – формат 3U, размером 100×160 мм с двумя разъемами для подключения к шине крейта (J1 и J2), и формат 6U, размером 233.35×160 мм, с четырьмя разъемами. Разъем J1 полностью аналогичен стандартному слоту PCI-32, разъем J2 содержит контакты, расширяющие шину данных PCI до 64 бит, и выполняющие дополнительные функции PXI, назначение контактов двух других разъемов не стандартизовано.

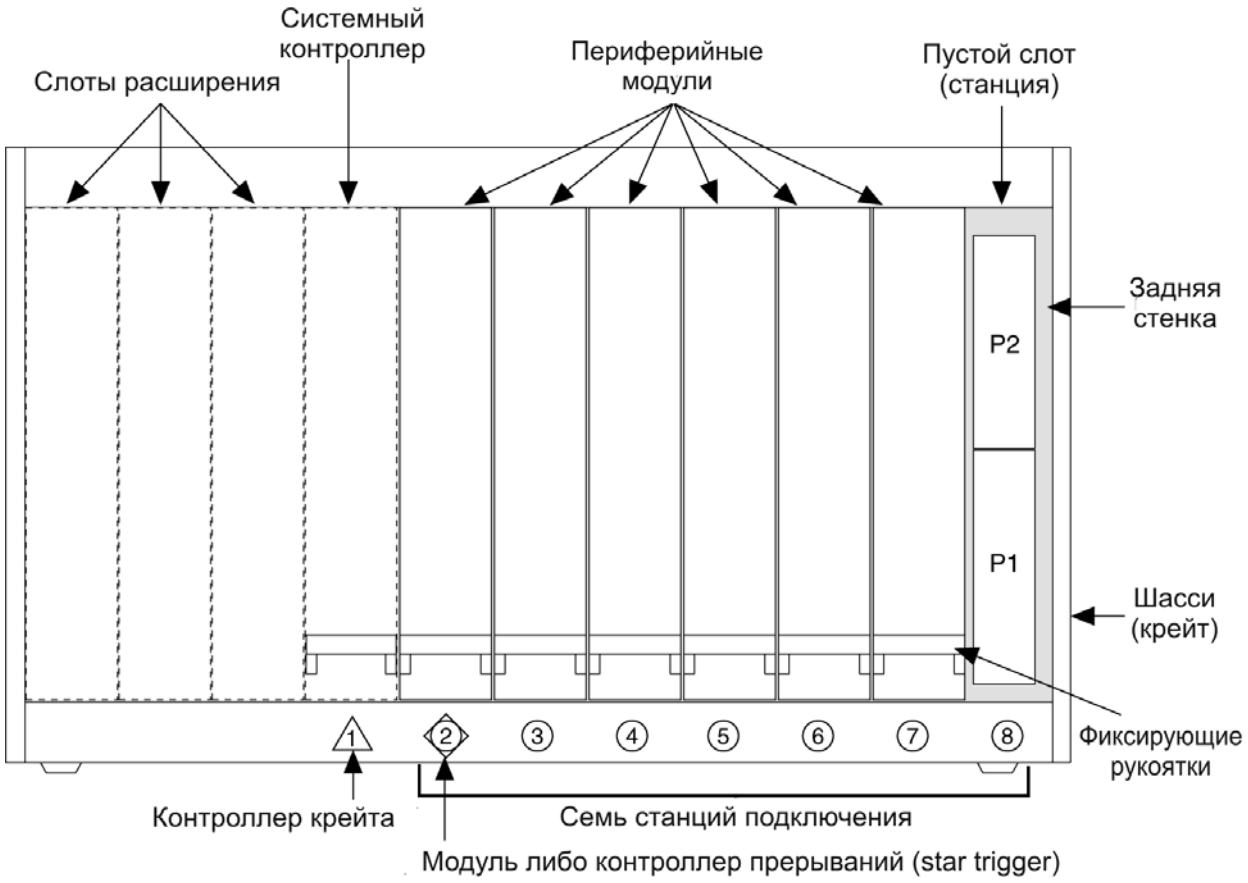

Рис. 5. Схематический вид крейта PXI.

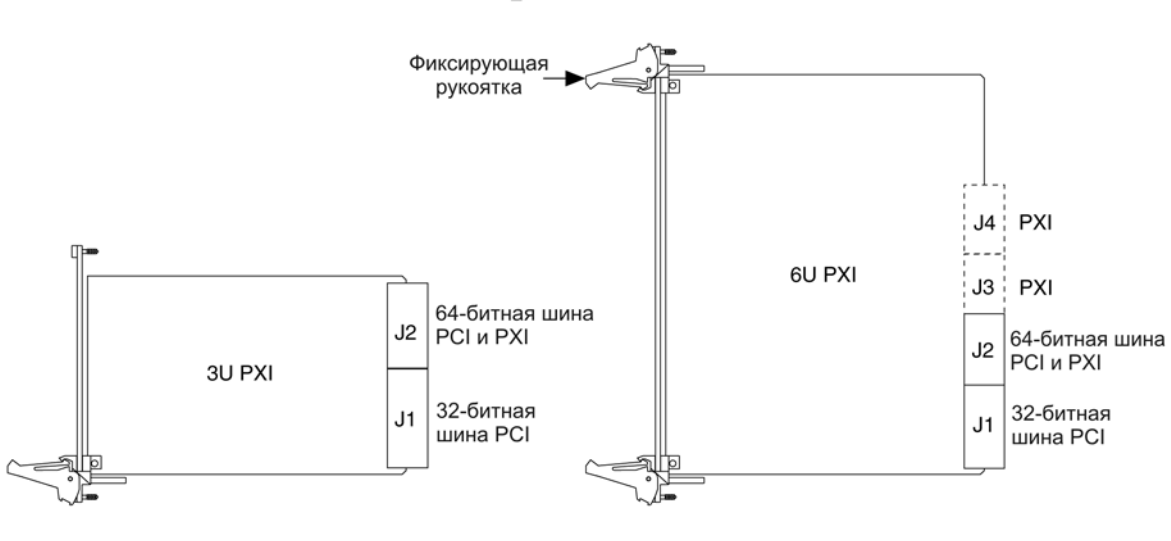

Рис. 6. Типоразмеры плат PXI.

Литература

1. http://www.ni.com/pxi/

2. http://www.pxisa.org/
3. http://www.pxionline.com/

Лекция 12.
Модульный интерфейс PXI. Часть 2

План лекции.
Электрический и логический стандарт PXI.

Электрический стандарт

Основные электрические характеристики стандарта PXI:
– Тактовые частоты 33/66 МГц.
– Возможность подключения до 7 (33 МГц) или 4 (66 МГц) интерфейсных модулей в одном крейте.
– Передача 32- и 64-битных слов данных.
– Пиковая скорость обмена от 132 Мб/с (32 бита, 33 МГц) до 528 Мб/с (64 бита, 66 МГц).
– Возможность подключения дополнительных крейтов через PCI-PCI мост.
– Линии питания 3.3 В.
– Поддержка режима конфигурации Plug & Play.

В дополнение к обычным линиям PCI шины, стандарт PXI включает дополнительные линии, образующие локальную шину PXI. Каждый из интерфейсных модулей соединен 13 линиями этой шины со своим правым и левым соседом в крейте; кроме этого, ближайший к контроллеру крейта слот (№ 2) звездообразно соединяется с остальными шестью (рис. 7). 13 линий локальной шины могут использоваться как для передачи цифровых сигналов в стандарте TTL, так и для аналоговых сигналов напряжением до 42 В, что определяется конструкцией отдельных модулей и должно учитываться при разработке соответствующего программного обеспечения.

Контроллер PXI имеет встроенный таймер с тактовой частотой 10 МГц, который используется для синхронизации процессов в интерфейсных модулях.

Наличие звездообразного соединения модулей(триггер-линии, аналог линий LAN системы КАМАК), как и в КАМАКе, позволяет передавать информацию непосредственно от интерфейсного модуля к контроллеру. Эти же линии могут использоваться для синхронизации процессов в интерфейсных модулях. При наличии специализированного модуля – контроллера триггеров возможна передача сигналов по триггер-линиям от одного модуля к другому, минуя контроллер, и, таким образом, организация

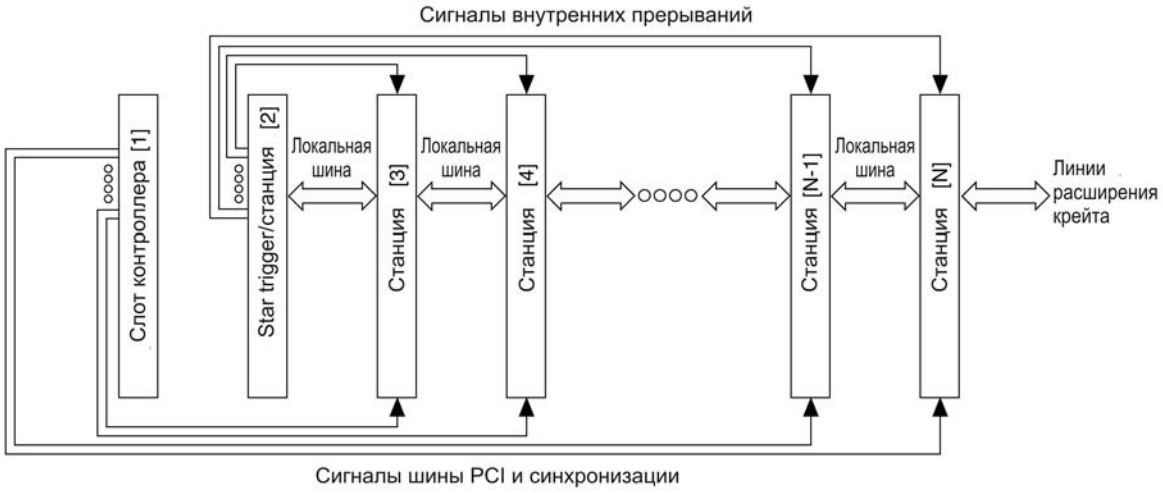

Рис. 7. Локальная шина PXI.

достаточно сложных последовательностей операций, выполняемых интерфейсными модулями без участия контроллера крейта и управляющей ЭВМ.

Как компанией-разработчиком, так и другими фирмами (этому способствует то, что PXI является открытой архитектурой) выпускается широкий ассортимент крейтов, контроллеров и интерфейсных плат этого стандарта, назначение которых аналогично интерфейсам КАМАК; в тоже время более высокие требования к электрическим и механическим параметрам компонент приводят к заметно более высоким ценам этих устройств.

Литература

1. http://www.ni.com/pxi/
2. http://www.pxisa.org/
3. http://www.pxionline.com/

Лекция 13.
Модульный интерфейс VXI

Модульная система VXI (VME Extensions for Instrumentation), аналогично PXI, основана на использовании шины, первоначально разработанной для обмена данными между компонентами компьютера. В данном случае использована шина VMEbus, разработанная совместно компаниями Motorola, Mostek и Signetics. Шина и весь стандарт разрабатывались для применений в промышленных управляющих

компьютерах, поэтому, подобно PXI, с самого начала были ориентированы на работу в жестких эксплуатационных условиях. В качестве механического стандарта использован тот же формат Евромеханика (EuroCard). Первая версия этой шины, известная также как VERSabus, была использована Motorola Corporation в 1979 году при создании промышленных многопроцессорных систем на базе микропроцессора Motorola 68000. Разработка оказалась весьма удачной, главным образом, за счет того, что имела большой запас ресурсов: не была привязана к одному типу процессоров, допускала существенное расширение разрядности данных, повышение рабочих частот; кроме того, благодаря открытой архитектуре ее использовало большое число разработчиков промышленной электроники. В результате доработок и совершенствования архитектуры шины наиболее распространенный сейчас стандарт VME64 поддерживает обмен 64-разрядными данными с со скоростью 80 Мб/с (160 Мб/с для принятого недавно VME64x, свыше 500 Мб/с для разрабатываемого VME320), семь уровней внутренних прерываний, до 21 процессора, архитектуру Plug & Play.

В дополнение к этому в системах VXI (аналогично PXI) добавлено 10 триггерных линий для синхронизации процессов, выполняемых отдельными модулями, введена 12-разрядная внутренняя шина обмена данными между модулями. В целом система во многом аналогична PXI, но предполагает использование более высокоинтеллектуальных модулей, как правило, включающих собственный управляющий процессор. IBM-совместимые контроллеры VXI, по сути, сами являются достаточно мощными компьютерами – они включают процессор семейства Pentium, жесткий диск, оперативную память, допускают прямое подключение монитора и клавиатуры и вполне могут выполнять функции управляющей ЭВМ. Все это, несомненно, расширяет функциональные возможности систем этого стандарта, но значительно увеличивает их стоимость.

Литература

1. http://www.vxitech.com/
2. http://www.vxibus.ru/index.shtml?2_1
3. http://www.vxipnp.org/

Раздел 4.
Программное обеспечение автоматизации эксперимента

Лекция 14.
Среда прикладного графического программирования LABVIEW

План лекции.
Программная среда LabVIEW
Понятие виртуального прибора
Функции LabVIEW
Потоки данных LabVIEW

LabVIEW или *Laboratory Virtual Instrument Engineering Workbench* (Среда разработки лабораторных виртуальных приборов) представляет собой среду графического программирования, которая широко используется в промышленности, образовании и научно-исследовательских лабораториях в качестве стандартного инструмента для сбора данных и управления приборами. LabVIEW – мощная и гибкая программная среда, применяемая для проведения измерений и анализа полученных данных. LabVIEW – многоплатформенная среда: вы можете использовать ее на компьютерах с операционными системами Windows, MacOS, Linux, Solaris и HP-UX на различных аппаратных платформах: на персональных и промышленных компьютерах, в распределенных системах. Производит ее общепризнанный мировой лидер технологии виртуальных приборов компания *National Instruments,* которая уже несколько десятков лет производит аппаратное и программное обеспечение, позволяющее создавать системы измерения, управления и диагностики. Программное обеспечение National Instruments (NI) включает в себя среды для разработки приложений LabVIEW, LabWindows/CVI и Measurement Studio, драйверы приборов и различного оборудования, а также высокоуровневые средства управления тестами и обработкой данных.

LabVIEW находит применение в самых разнообразных сферах человеческой деятельности. В соответствии со своим названием он первоначально использовался в исследовательских лабораториях, да и в настоящее время является наиболее популярным программным пакетом как в лабораториях фундаментальной науки (например, Lawrence Livermore, Argonne, Batelle, Sandia, Jet Propulsion Laboratory, White Sands и Oak Ridge в CILIA, CERN в Европе), так и в отраслевых промышленных лабораториях. Все более широкое применение LabVIEW находит в образовании – в университетских лабораторных практикумах, особенно по курсам электротехники, механики и физики.

Распространение LabVIEW за пределами лабораторий пошло по всем направлениям: вверх (на борту космических аппаратов), вниз (на подводных лодках) и по горизонтали (от буровых установок в Северном море до промышленных предприятий в Новой Зеландии). В связи с ростом возможностей Internet сфера применения LabVIEW стала расширяться не только в географическом, но и в виртуальном пространстве. Созданы виртуальные приборы, допускающие удаленное управление и наблюдение через Internet. Измерительные системы на основе виртуальных приборов отличаются своей многофункциональностью, гибкостью и низкой стоимостью как с точки зрения оборудования, так и с точки зрения затрат времени на разработку. Нужно ли удивляться, что они стали столь популярны?

Пожалуй, лучшим способом объяснить причины столь широкого распространения пакета LabVIEW будет обобщение способов его использования. Во всех видах человеческой деятельности существуют области, где не обойтись без определенных видов измерений – очень часто это температурные измерения, например в печах, холодильниках, парниках, технологических помещениях. Кроме температуры, часто измеряют давление, силу, пространственное смещение, механическое напряжение, *pH* и многое другое. Сейчас персональные компьютеры проникли практически во все сферы жизнедеятельности. LabVIEW ускоряет внедрение компьютера в измерительные системы – и не только потому, что облегчает проведение измерений, он также дает возможность проанализировать измеренные величины, отобразить их на графиках и в отчетах и при желании опубликовать.

Персональные компьютеры являются более гибкими инструментами, чем традиционные измерительные приборы, поэтому создание собственной программы на LabVIEW, или *виртуального прибора* (ВП), является довольно несложным делом, а интуитивно понятный пользовательский интерфейс в среде LabVIEW делает разработку программ и их применение весьма интересным и увлекательным занятием.

Кратко особенности среды LabVIEW сформулировать следующим образом:

– функционально полный язык графического программирования, позволяющий создавать программу в форме наглядной графической блок-схемы, которая традиционно используется радиоинженерами;

– встроенные программные средства для сбора данных, управления приборами и оборудованием, обработки сигналов и экспериментальных данных, генерации отчетов, передачи и приема данных и т. д.;

– мощное математическое обеспечение, возможность интеграции программ, написанных в среде математического пакета Matlab;

– наличие более 2000 программ (драйверов), позволяющих сопрягать разработанную программу с разнообразными приборами и оборудованием различных фирм через стандартные интерфейсы;

–наличие большого количества шаблонов приложений, а также свыше 1000 примеров, позволяющих быстро создавать собственные программы, внося в них небольшие коррекции;

– высокая скорость выполнения откомпилированных программ;

– возможность работы LabVIEW под управлением операционных систем Windows 2000/NT/XP, Mac OS X, Linux и Solaris.

Программная среда LabVIEW

Концепция LabVIEW сильно отличается от последовательной природы традиционных языков программирования, предоставляя разработчику легкую в использовании графическую оболочку, которая включает в себя весь набор инструментов, необходимых для сбора данных, их анализа и представления полученных результатов. С помощью графического языка программирования LabVIEW, именуемого G (Джей), можно программировать задачу из графической блок-диаграммы, которая *компилирует* алгоритм в машинный код. Являясь превосходной программной средой для бесчисленных применений в области науки и техники, LabVIEW позволяет решать задачи различного типа, затрачивая значительно меньше времени и усилий, по сравнению с написанием традиционного программного кода.

LabVIEW является *средой программирования,* с помощью которой вы можете создавать приложения, используя графическое представление всех элементов алгоритма, что отличает ее от обычных *языков программирования,* таких как C, C++ или Java, где программируют, используя текст. Однако LabVIEW представляет собой значительно большее, чем просто алгоритмический язык. Это среда разработки и исполнения приложений, предназначенная для исследователей – ученых и инженеров, для которых программирование является лишь частью работы. Создание законченного приложения с помощью обычных языков программирования может отнять очень много времени – недели или месяцы, тогда как с LabVIEW требуется лишь несколько часов, поскольку пакет специально разработан для программирования различных измерений, анализа данных и оформления результатов. Так как LabVIEW имеет гибкий графический интерфейс и прост для программирования, он также отлично подходит для моделирования процессов, презентации идей, создания

приложений общего характера и просто для обучения современному программированию.

Понятие виртуального прибора

По существу, ***технология виртуальных приборов,*** позволяющая создавать системы измерения, управления и диагностики различного назначения практически любой произвольной сложности, включая математическое моделирование и тестирование этих систем, в свое время стала революцией в области лабораторный измерений. Суть этой технологии состоит в компьютерной имитации функций реальных физических приборов, измерительных и управляющих систем. Программная среда LabVIEW является именно таким инструментарием технологии виртуальных приборов.

Слово *«виртуальный»* не должно вводить в заблуждение, поскольку приборы, реализованные по этой технологии, на самом деле являются *реальными,* работающими с реальными физическими входными сигналами. Виртуальность здесь понимается в смысле виртуальной имитации функций прибора математическими и программными методами. Например, *виртуальный осциллограф* по функциям эквивалентен реальному осциллографу, поскольку имеет физический вход для электрического сигнала. Преобразование сигнала в цифровой сигнал осуществляется аналого-цифровым преобразователем (АЦП). Дальнейшая обработка и управление сигналом, его отображение для наблюдения осуществляются программным способом. Такой осциллограф имеет виртуальный экран, виртуальные ручки управления (усиление, синхронизация, развертка и др.), графически отображаемые на экране монитора компьютера. Ручки, переключатели, кнопки виртуального прибора управляются с клавиатуры или посредством мыши.

Другим простым пояснительным примером может служить *виртуальный генератор сигналов.* Такой виртуальный генератор имеет реальный электрический выход, реальные входы для синхронизации, а также виртуальные ручки управления по функциям, аналогичным обычному генератору. Выходные электрические сигналы (гармонический, пилообразный, прямоугольный, случайный и т. д.) формируются цифро-аналоговым преобразователем (ЦАП). Генерация сигналов различной формы осуществляется программно-математическими методами. Например, если для генерации синусоидального сигнала в реальном генераторе используется колебательный контур, включенный в цепь обратной связи усилителя, то в виртуальном генераторе гармонический сигнал получается математически непосредственно по соответствующей тригонометрической формуле для

синусоиды. Ясно, что в этом случае генерируется почти идеальный синусоидальный сигнал без нелинейных искажений, с очень стабильной частотой и амплитудой, а также с известной начальной фазой. В реальном генераторе такие метрологические параметры практически недостижимы.

Виртуальный прибор состоит из трех основных частей:

– *лицевая панель* (Front Panel) представляет собой интерактивный пользовательский интерфейс виртуального прибора и названа так потому, что имитирует лицевую панель традиционного прибора. На ней могут находиться ручки управления, кнопки, графические индикаторы и другие *элементы управления* (controls), которые являются средствами ввода данных со стороны пользователя, и *элементы индикации* (indicators) – выходные данные из программы. Пользователь вводит данные, используя мышь и клавиатуру, а затем видит результаты действия программы на экране монитора;

– *блок-диаграмма* (Block Diagram) является исходным программным кодом ВП, созданным на языке графического программирования LabVIEW, G (Джей). Блок-диаграмма представляет собой реально исполняемое приложение. Компонентами блок-диаграммы являются: *виртуальные приборы более низкого уровня, встроенные функции* LabVIEW, *константы* и *структуры управления* выполнением программы. Для того чтобы *задать поток данных* между определенными объектами или, что то же самое, *создать связь* между ними, вы должны нарисовать соответствующие *проводники* (wires). Объекты на лицевой панели представлены на блок-диаграмме в виде соответствующих *терминалов* (terminals), через которые данные могут поступать от пользователя в программу и обратно;

– для того, чтобы использовать некоторый ВП в качестве подпрограммы (подприбора) в блок-диаграмме другого ВП, необходимо определить его *иконку* (icon) и *соединительную панель* (connector). Виртуальный прибор, который применяется внутри другого ВП, называется *виртуальным подприбором* (ВПП, SubVI), который аналогичен *подпрограмме* в традиционных алгоритмических языках. Иконка является однозначным *графическим представлением* ВП и может использоваться в качестве *объекта* на блок-диаграмме другого ВП. Соединительная панель представляет собой механизм передачи данных в ВП из другой блок-диаграммы, когда он применяется в качестве подприбора – ВПП. Подобно аргументам и параметрам подпрограммы, соединительная панель определяет входные и выходные данные виртуального прибора.

Виртуальные приборы являются *иерархическими* и *модульными* (modular). Вы можете использовать их как самостоятельные приложения (top-level programs), так и в качестве виртуальных подприборов. Согласно этой логике, Lab-

VIEW следует концепции *модульного программирования* (modular programming). Вначале вы разделяете большую прикладную задачу на ряд простых подзадач. Далее создаете виртуальные приборы для выполнения каждой из подзадач, а затем объединяете эти ВП на блок-диаграмме прибора более высокого уровня, который выполняет прикладную задачу в целом.

Технология модульного программирования очень хороша, потому что вы можете работать с каждым ВПП по отдельности, что облегчает отладку приложения. Более того, ВПП низкого уровня часто выполняют задачи, типичные для нескольких приложений, и поэтому могут использоваться независимо во многих отдельных приложениях.

Преимущество технологии виртуальных приборов состоит в возможности программным путем, опираясь на мощь современной компьютерной техники, создавать разнообразные приборы, измерительные системы и программно-аппаратные комплексы, легко их адаптировать к изменяющимся требованиям, уменьшить затраты и время на разработку.

Измерительная система, созданная в LabVIEW, имеет большую гибкость по сравнению со стандартным лабораторным прибором, потому что она использует многообразие возможностей современного программного обеспечения. И именно пользователь, а не изготовитель оборудования, определяет функциональность создаваемого прибора. Компьютер, снабженный встраиваемой измерительно-управляющей аппаратной частью, и LabVIEW составляют полностью настраиваемый *виртуальный прибор* для выполнения поставленных задач. С помощью LabVIEW допустимо создать необходимый тип виртуального прибора при очень малых затратах по сравнению с обычными инструментами. При необходимости внести в него изменения можно буквально за минуты.

Функции LabVIEW

После *измерения* и *анализа* какой-либо величины следующим логическим шагом часто является *управление,* то есть изменение определенных параметров в зависимости от полученных результатов. Например, измерив температуру объекта, можно включить устройство для его охлаждения либо нагрева. И вновь LabVIEW значительно облегчает решение этой задачи: *мониторинг* и *управление процессами* являются основными функциями этого программного продукта. Управление процессами может быть прямым или осуществляется через специальные *программируемые логические контроллеры* (programmable logical controllers – PLC), что принято называть *диспетчерским управлением и сбором данных* (supervisory control and data acquisition – SCADA).

LabVIEW создан для облегчения работы по программированию ваших задач. Для этой цели имеется расширенная библиотека функций и готовых к использованию подпрограмм, которые реализуют большое число типичных задач программирования и тем самым избавляют вас от рутинной возни с указателями, распределением памяти и прочего шаманства, присущего традиционным языкам программирования. В LabVIEW также содержатся специальные библиотеки виртуальных приборов для *ввода/вывода данных со встраиваемых аппаратных* средств (data acquisition – DAQ), для работы с *каналом общего пользования* (КОП, General Purposes Interface Bus – GPIB), управления устройствами через последовательный порт *RS-232,* программные компоненты для анализа, представления и сохранения данных, взаимодействия через сети и Internet. Библиотека *анализа* (Analysis) содержит множество полезных функций, включая генерирование сигнала, его обработку, различные фильтры, окна, статистическую обработку, регрессионный анализ, линейную алгебру и арифметику массивов.

Благодаря своей графической природе LabVIEW – это пакет эффективного отображения и представления данных. Выходные данные могут быть показаны в любой форме, какую вы пожелаете. Диаграммы, графики стандартного вида, а также оригинальная пользовательская графика (user-defined graphics) составляют лишь малую часть возможных способов отображения выходных данных.

Программы LabVIEW легко переносятся на другие платформы: вы можете создать приложение на Macintosh, а затем запустить его в Windows, для большинства приложений практически ничего не меняя в программе. Приложения, созданные на LabVIEW, качественно улучшают работу во многих сферах деятельности человека – как в автоматизации технологических процессов, так и в биологии, сельском хозяйстве, психологии, химии, физике, образовании и множестве других.

Потоки данных LabVIEW

Разработка приложений в среде LabVIEW отличается от работы в средах на основе C или Java одной очень важной особенностью. Если в традиционных алгоритмических языках программирование основано на вводе текстовых команд, *последовательно* образующих *программный код,* в LabVIEW используется *язык графического программирования,* где алгоритм создается в графической иконной форме (pictorial form), образующей так называемую *блок-диаграмму* (block-diagram), что позволяет исключить множество синтаксических деталей. Применяя этот метод, вы можете

сконцентрировать внимание лишь на программировании потока данных; упрощенный синтаксис теперь не отвлекает вас от анализа самого алгоритма.

В LabVIEW используется терминология, рисунки иконок и основные идеи, знакомые ученым и инженерам. Этот язык базируется на графических символах, а не на тексте для описания программируемых действий. Основополагающий для LabVIEW принцип *потока данных* (dataflow), согласно которому функции выполняются лишь тогда, когда они получают на вход необходимые данные, однозначно определяет порядок исполнения алгоритма.

Литература

1. Автоматизация физических исследований и эксперимента: компьютерные измерения и виртуальные приборы на основе LabVIEW 7 / Под ред. Бутырина П. А. – М.: ДМК Пресс, 2005. – 264 с.
2. Тревис Дж. LabVIEW для всех / Джеффри Тревис – М.: ДМК Пресс, 2004. – 544 с.
3. Евдокимов Ю. К. LabVIEW для радиоинженера: от виртуальной модели до реального прибора. / Ю. К. Евдокимов, В. Р. Линдваль, Г. И. Щербаков – М.: ДМК Пресс, 2007. – 512 с.
4. Суранов А. Я. LabVIEW 7: справочник по функциям / А. Я. Суранов – М.: ДМК Пресс, 2007. – 512 с.
5. Каратаев В. Автоматизация физических исследований и эксперимента: компьютерные измерения и виртуальные приборы на основе LabVIEW 7 / В. Каратаев, Т. Васильковская, П. А. Бутырин – М.: ДМК Пресс, 2004 – 480 с.
6. Батоврин В. К. LabVIEW: Практикум по основам измерительных технологий / В. К. Батоврин, А. С. Бессонов, В. В. Мошкин, В. Ф. Популовский – М.: ДМК Пресс, 2005. – 208 с.
7. http://www.ni.com/labview
8. http://www.ni.com/russia

Лекция 15.
Фильтрация случайных шумов в ходе эксперимента

План лекции.
Метод «ворот»
Метод выборки

Известно, что нормальный (гауссов) шум можно достаточно успешно отфильтровать, путем простого усреднения сигнала. При этом среднеквадратичная ошибка результата:

$$\langle \delta \rangle \propto \frac{1}{\sqrt{N}}, \qquad (1)$$

где N – число измерений, по которым выполняется усреднение.

Отсюда видно, что если одно из измерений оказалось случайным выбросом, который на порядок превышает истинное значение сигнала (и соответственно увеличивает ошибку измерений), то устранения его влияния необходимо увеличить количество измерений на два порядка. Очевидно, что это сильно увеличивает время проведения эксперимента и не всегда осуществимо. Этим определяется необходимость их предварительной фильтрации. Особенностью всех применяемых здесь методов является то, что фильтрация выполняется в ходе эксперимента, когда полный набор данных отсутствует, и имеется только крайне ограниченное число уже полученных измерений.

Метод «ворот»

В этом методе предполагается, что изменения измеряемого сигнала от точки к точке достаточно невелико, и имеются априорные (заданные оператором) данные о возможной величине этих изменений (ширина ворот).

Насколько начальных точек рассматриваются как корректные и по ним выполняется аппроксимация сигнала к точке очередного измерения. На рис. 8 показан простейший случай – линейная аппроксимация, когда достаточно двух начальных точек (1 и 2 на рисунке). Если экспериментально измеренное значение лежит на расстоянии, меньшем установленного допустимого отклонения, то она считается истинной (точка 3 на рисунке), и дальнейшая аппроксимация выполняется с учетом полученного значения (в данном случае – по точкам 2 и 3). Если новое измеренное значение выпадает за пределы допустимого отклонения от экстраполированного значения (точка 4 на рисунке), то ее значение рассматривается как случайный выброс, и оно отбрасывается. В идеальном случае, если значение измеряемой функции в этой точке необходимо, следует повторить измерения. Если это невозможно в силу характера проводимых измерений, значение 4 заменяется на экстраполированное значение (точка 4’).

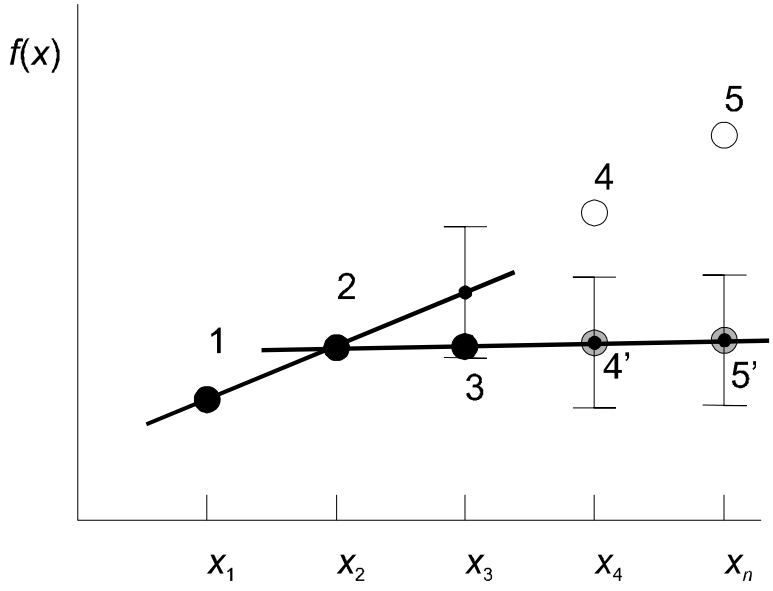

Рис. 8. Фильтрация методом «ворот».

Подобная процедура замены опасна тем, что экстраполированная линия может теперь заметно отличаться от действительной, и высока вероятность того, что последующие правдоподобные точки также будут выпадать за пределы экстраполяции. В результате все дальнейшие экспериментальные значения будут утеряны, независимо от того, являются они выбросами или нет. Чтобы избежать этого, в алгоритмах таких измерений обычно предусматривается условие, запрещающее производить повторные замены по экстраполяции; причиной их возникновения обычно является установление слишком узкого доверительного интервала «ворот».

К недостаткам этого метода, таким образом, следует отнести: необходимость произвольного вмешательства оператора при установлении ширины «ворот»; опасность утери данных из-за неудачно установленных «ворот»; замену реально измеренных значений на экстраполированные в «подозрительных» точках.

Влияние этих недостатков, в какой-то мере, можно ослабить, используя более сильные, нелинейные методы экстраполяции. Однако они, как правило, требуют использования большего количества точек для экстраполяции. Это повышает вероятность попадания случайного выброса среди начальных опорных точек, что приведет к заведомо неверному результату экстраполяции.

Метод выборки

В какой-то мере свободен от этих недостатков метод выборки. Для пояснения его работы введем понятие *робастного среднего*.

Пусть у нас имеется набор из N случайных величин (для простоты будем считать N нечетным). Упорядочим их по величине (по возрастанию или убыванию) и пронумеруем. Значение величины с номером $(N + 1)/2$ (лежащее в середине этой упорядоченной последовательности) называется робастным средним. Существует теорема, доказывающая, что для случайно (гауссово) распределенных величин робастное среднее стремится к «обычному» среднему при $N \to \infty$. В то же время очевидно, что если среди значений выборки окажется случайный выброс, он не повлияет на величину робастного среднего, так как окажется на одном из краев упорядоченной последовательности значений. На этом факте и основан рассматриваемый метод.

Пусть мы имеем первые три (в общем случае – любое нечетное число) измерений экспериментальной последовательности $f(x)$. Упорядочим их по величине и найдем среднюю (по номеру) точку. В показанном на рис. 9 примере это точка № 3 из первых трех. Считаем ее средним значением измеряемой величины $f(x)$ на интервале $x1$–$x3$; на рисунке эта усредненная последовательность обозначена $f'(x)$. Далее берем точки $x2$–$x4$ и повторяем процедуру: средним значением опять оказывается точка № 3. Повторяем это процедуру до конца измерений. Как видно из рисунка, получившаяся в результате фильтрации последовательность $f'(x)$ выглядит более гладкой и не содержит минимальных и максимальных значений последовательности $f(x)$, которые могли быть случайными выбросами.

Достоинствами этого метода являются: простота (не требуется никаких математических действий, только операции сравнения и обмена данными между массивами); отсутствие вычисленных значений измеряемых величин – все значения в отфильтрованном массиве получены экспериментально; отсутствие необходимости ввода априорных данных об измеряемых величинах (за исключением числа точек, по которым следует производить фильтрацию).

Легко убедиться, что фильтрация с выборкой по трем точкам, как в данном примере, не позволяет удалить два случайных выброса, если они следуют друг за другом; один из них проникает в отфильтрованную последовательность. Чтобы отфильтровать n последовательных выбросов, необходима фильтрация по $2n + 1$ точкам. При увеличении числа точек качество фильтрации улучшается (на выходе образуется более гладкая последовательность), но увеличивается количество отброшенных значений. В любом случае, фильтрация по $2n + 1$ точкам приводит к потере n точек в начале и в конце массива, что, несомненно, является недостатком метода.

Отметим, что данный алгоритм является эффективным средством диагностики случайных выбросов, и поэтому часто применяется в качестве предварительного фильтра в сочетании с другими. Так, при многократных измерениях в повторяющихся условиях, когда предполагается установить

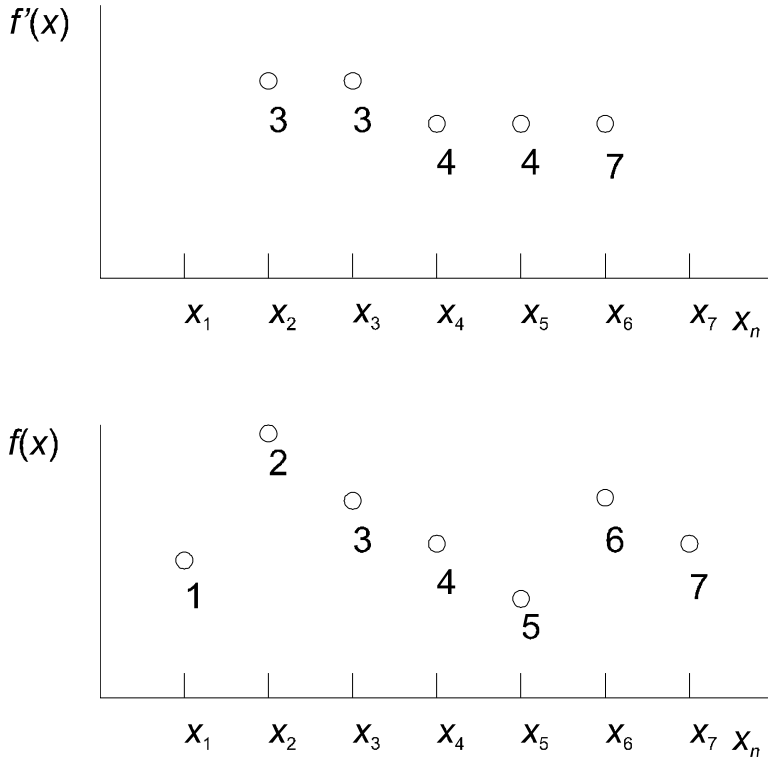

Рис. 9. Фильтрация методом выборки.
Снизу – экспериментально измеренные значения,
сверху – результат фильтрации.

истинное значение зашумленного сигнала, его можно использовать для предварительного отделения заведомых выбросов, после чего использовать обычное усреднение.

Литература

1. Компьютеры в оптических исследованиях / Под ред. Б. Фридена. – М.: Мир, 1983. – 320 с.
2. Отнес Р. Прикладной анализ временных рядов / Р. Отнес, Л. Эноксон – М., Мир, 1982. – 424 с.

Лекция 16.
Аппроксимация экспериментальных данных с помощью аналитических функций

План лекции.
Интерполяция с помощью полиномов
Интерполяционная формула Лагранжа
Интерполяционная формула Ньютона
Интерполяция с помощью кубических сплайнов

Полученные в ходе эксперимента данные, как правило, представляют собой набор точек. В то же время для удобства их отображения, а также для последующей интерпретации желательно представить их в виде непрерывной зависимости, лучше всего – аналитической. Для этого разработаны и используются многочисленные алгоритмы. Здесь описаны наиболее простые из них, реализация которых не требует значительных затрат машинного времени или больших ресурсов используемого компьютера.

В простейшем случае задача аппроксимации формулируется следующим образом. В результате эксперимента в точках x_1–x_N найдены (приближенные) значения y_1–y_N неизвестной функции $y = f(x)$. Пользователем или программистом задан некий набор функций $\{F(x, \mathbf{p})\}$, где $\mathbf{p} = (p_1, \ldots\ p_\alpha)$ – набор параметров этих функций. Задача состоит в том, чтобы выбрать из этого набора функцию $F(x, \mathbf{p})$, удовлетворяющую заданному критерию близости к $f(x)$. В зависимости от выбора этого класса функций и критерия близости можно построить самые разнообразные алгоритмы такого поиска, позволяющие решать различные практические задачи.

Одной из простейших критериев близости является совпадение $f(x)$ и $F(x, \mathbf{p})$ во всех экспериментальных точках; такая задача называется *задачей интерполяции*, а точки, через которые должна проходить искомая функция – *узлами интерполяции*. Такая задача может быть решена в том случае, если имеется достаточно большой набор варьируемых параметров **p**, а ошибки

измерений настолько малы, что их можно не принимать во внимание (либо сама задача носит промежуточный характер – например, выполняется для удобства визуализации полученных результатов измерений).

В противном случае за критерий близости принимается менее жесткое условие, например минимизация выражения:

$$S(\mathbf{p}) = \sum_{1}^{N} (y_i - F(x_i, \mathbf{p}))^2 ; \tag{2}$$

соответствующие алгоритмы являются разновидностями *метода наименьших квадратов*.

Наконец, в некоторых случаях в качестве критерия близости выбирается условие минимакса (*минимаксный критерий*), то есть минимизация функции:

$$Z(\mathbf{p}) = \max|y_i - F(x_i, \mathbf{p})| . \tag{3}$$

Интерполяция с помощью полиномов

Задача интерполяции с помощью полиномов формулируется следующим образом. На некотором отрезке [a, b] заданы N точек $y_i(x_i)$. Необходимо найти полином:

$$P_n(x) = \sum_{\alpha=0}^{n} a_\alpha x^\alpha, \; x \in [a,b] ; \tag{4}$$

для которого

$$P_n(x_i) = y_i, \; i = 1,...N . \tag{5}$$

Коэффициенты полинома a_α являются искомыми параметрами; их можно найти, решая систему уравнений (5).

В общем случае, если $n + 1 < N$, эта система не имеет решений. Если $n + 1 = N$, то определителем этой системы является определитель Вандермонда

$$D = \prod_{i>j}^{N} (x_i - x_j), \tag{6}$$

и система имеет единственное решение, если он не равен нулю, что, достаточно просто обеспечить (это означает, что для каждой экспериментальной точки имеется единственное значение экспериментально измеряемой величины y).

Отсюда следует, что по заданному набору N экспериментальных точек можно построить интерполяционный полином степени $N - 1$, причем он является единственным.

Интерполяционная формула Лагранжа

Предположим, что удалось построить систему полиномов $\varphi_i(x)$, каждый из которых равен единице в точке x_i и равен нулю во всех точках измерений. Тогда искомый полином степени $N - 1$ имеет вид:

$$P_{N-1}(x) = \sum_{i=1}^{N} y_i \varphi_i(x). \tag{7}$$

Этот набор полиномов $\varphi_i(x)$ называется *фундаментальной системой полиномов.* Поскольку этот полином единственный и обращается в нуль во всех точках измерений, кроме x_i, его можно представить в виде:

$$\varphi_i(x) = C_i \prod_{k \neq i}^{N} (x - x_k), \tag{8}$$

где C_i – некоторая постоянная. Найдя ее из условия $\varphi_i(x_i) = 1$ и подставив в (8), получим:

$$\varphi_i(x) = \prod_{k \neq i}^{N} (x - x_k) \Big/ \prod_{k \neq i}^{N} (x_i - x_k). \tag{9}$$

Подставляя в (7), получим интерполяционный полином Лагранжа:

$$P_{N-1}(x) = \sum_{i=1}^{N} y_i \prod_{k \neq i}^{N} (x - x_k) \Big/ \prod_{k \neq i}^{N} (x_i - x_k). \tag{10}$$

Чтобы представить это выражение в более традиционном и компактном виде, введем вспомогательную функцию:

$$\Pi(x) = \prod_{k=1}^{N} (x - x_k), \tag{11}$$

производная которой равна

$$\Pi'(x) = \sum_{m=1}^{N} \prod_{k \neq m}^{N} (x - x_k). \tag{12}$$

Тогда:

$$P_{N-1}(x) = \sum_{i=1}^{N} y_i \frac{\Pi(x)}{(x - x_i)\Pi'(x_i)} = \Pi(x) \sum_{i=1}^{N} \frac{y_i}{(x - x_i)\Pi'(x_i)}. \tag{13}$$

Для равноотстоящих точек $x = x_1 + (t - 1)h$:

$$P_{N-1}(x) = \prod_{m=1}^{N} (t - m) \sum_{k=1}^{N} \frac{(-1)^{N-k} y_k}{(t - k)(k - 1)!(N - k)!}. \tag{14}$$

Аппроксимация с помощью этого алгоритма является достаточно эффективной, когда аппроксимируемые функции достаточно гладкие, а число точек невелико (и, соответственно, невелика степень аппроксимирующего полинома). Большинство распространенных библиотек имеют готовые программные модули, реализующие этот алгоритм. Рост количества точек и степени полинома приводит к быстрому усложнению выражений (10), (13), (14), и зачастую неоправданному возрастанию требований к ресурсам ЭВМ. Еще одним слабым местом этого алгоритма является то, что для его реализации необходимо знание всех точек аппроксимируемой функции, то есть его сложно реализовать в ходе измерений. Последний недостаток преодолен в следующем алгоритме.

Интерполяционная формула Ньютона

Введем понятие *разделенных разностей*. Определим разделенную разность первого порядка как:

$$\Delta(x_i, x_{i-1}) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}. \tag{15}$$

Разность второго порядка:

$$\Delta(x_i, x_{i-1}, x_{i-2}) = \frac{\Delta(x_i, x_{i-1}) - \Delta(x_{i-1}, x_{i-2})}{x_i - x_{i-2}}, \tag{16}$$

и в общем случае разность n-того порядка:

$$\Delta(x_i, \ldots x_{i-n}) = \frac{\Delta(x_i, \ldots x_{i-n+1}) - \Delta(x_{i-1}, \ldots x_{i-n})}{x_i - x_{i-n}}. \tag{17}$$

Отметим, что эти разности являются разностным аналогом производных n-того порядка.

Рассмотрим разность первого порядка:

$$\Delta(x, x_1) = \frac{f(x) - y_1}{x - x_1}. \tag{18}$$

Отсюда:

$$f(x) = y_1 + (x - x_1)\Delta(x, x_1). \tag{19}$$

Выразив разность первого порядка из (16) и подставив в (19), получим:

$$f(x) = y_1 + (x - x_1)\Delta(x_1, x_2) + (x - x_1)(x - x_2)\Delta(x, x_1, x_2), \tag{20}$$

и, повторяя эту процедуру для разностей более высоких порядков,

$$f(x) = P_{N-1}(x) + R_{N-1}(x), \tag{21}$$

где полином

$$\begin{aligned} P_{N-1}(x) = {} & y_1 + (x - x_1)\Delta(x_1, x_2) + \\ & + (x - x_1)(x - x_2)\Delta(x_1, x_2, x_3) + \ldots \\ & + (x - x_1)\ldots(x - x_{N-1})\Delta(x_1, x_2, \ldots x_{N-1}) \end{aligned} \tag{22}$$

является интерполяционным, так как

$$P_{N-1}(x_i) = y_i\,, \tag{23}$$

а остаточный член

$$R_{N-1}(x) = (x - x_1)...(x - x_N)\Delta(x, x_1,...x_N) \tag{24}$$

определяет точность интерполяции. Выражение (22) является разностным аналогом формулы Тейлора разложения функции в ряд.

Так как, как уже отмечалось выше, интерполяционный полином для заданного набора точек является единственным, то путем перегруппировки членов полинома Ньютона можно получить полином Лагранжа, и наоборот.

Преимущество данного представления заключается в том, что *n*-тое слагаемое полинома Ньютона зависит только от *n* первых экспериментальных точек, и последующие точки приводят только к появлению новых слагаемых, без изменения первоначальных. Таким образом, построение этого полинома может идти непрерывно в процессе измерений.

В то же время этот метод не свободен от другого недостатка метода Лагранжа: как и в предыдущем случае, сложность вычислений, и, соответственно, требования к ресурсам компьютера быстро возрастает с увеличением числа экспериментальных точек. Чтобы избежать этого, возникает идея разбить весь исследуемый интервал на отдельные участки и выполнять интерполяцию на каждом из них раздельно. Но тогда интерполирующая функция будет иметь особенности на границах участков интерполяции; здесь возникнут точки перегиба, т.е. производные этой функции здесь будут иметь разрывы. Этого недостатка лишен следующий метод.

Интерполяция с помощью кубических сплайнов

Разобьем область экспериментальных точек [a, b] на участки $\delta\text{-}_I = [x_{i\text{-}1}, x_i]$. Интерполирующая функция $F(x)$ должна проходить через все экспериментальные точки: $F(x_i) = y_i$; она сама, ее первая и вторая производные должны быть непрерывны на отрезке [a, b]. Естественно искать эту функцию, интерполируя экспериментальные точки на каждом из отрезков $\delta\text{-}_I = [x_{i\text{-}1}, x_i]$ полиномом. Тогда низшей степенью такого полинома может быть только третья – иначе его производные превратятся в нули. Введем $C_i = F''(x_i)$. Тогда для кубического сплайна:

$$\frac{F''(x)-C_{i-1}}{x-x_{i-1}}=\frac{C_i-C_{i-1}}{x_i-x_{i-1}}, \tag{25}$$

откуда

$$\frac{F''(x)-C_{i-1}}{x-x_{i-1}}=\frac{C_i-C_{i-1}}{x_i-x_{i-1}}. \tag{26}$$

Выражая отсюда F'' и дважды интегрируя полученное выражение, получим:

$$\begin{aligned} F(x) = {} & C_i\frac{(x_i-x)^3}{6(x_i-x_{i-1})}+C_i\frac{(x-x_{i-1})^3}{6(x_i-x_{i-1})}+ \\ & +\frac{(x_i-x)}{(x_i-x_{i-1})}\left(y_{i-1}-\frac{C_{i-1}(x_i-x_{i-1})^2}{6}\right)+ \\ & +\frac{(x-x_{i-1})}{(x_i-x_{i-1})}\left(y_i-\frac{C_i(x_i-x_{i-1})^2}{6}\right). \end{aligned} \tag{27}$$

Легко убедиться, что данная функция отвечает поставленным условиям.

Коэффициенты сплайна C_i можно найти, исходя из условий непрерывной дифференцируемости (27) во всех точках x_i:

$$F'(x_i-0)=F'(x_i+0),\ i=2,\ N-1. \tag{28}$$

Выражение (28) после подстановки туда (27) дает систему из N – 2 линейных уравнений для N неизвестных коэффициентов сплайна C. Недостающие два уравнения необходимо получить, поставив граничные условия на интерполирующую функцию или ее производные на границах участка [a, b].

Литература

1. Носач В. В. Решение задач аппроксимации с помощью персональных компьютеров / В. В. Носач – М.: МИКАП, 1994. – 382 с.

2. Статистическая обработка результатов экспериментов на микро-ЭВМ / А. А. Костылев, П. В. Миляев, Ю. Д. Дорский и др. – Л.: Энергоатомиздат, 1991. – 304 с.
3. Живописцев Ф. А. Регрессионный анализ в экспериментальной физике / Ф. А. Живописцев, В. А. Иванов – М.: Изд-во МГУ, 1995. – 208 с.

Лекция 17.
Аппроксимация экспериментальных данных методом наименьших квадратов.
Часть 1. Методы нулевого порядка

План лекции.
Введение.
Минимизация методом прямого поиска.
Метод покоординатного спуска.
Симплексный метод.
Случайный поиск.

Аппроксимация экспериментальных данных методом наименьших квадратов

Как правило, экспериментально измеряемые величины y_i содержат ошибки, а их зависимость от параметра эксперимента x_i не является полиномиальной. Поэтому требование прохождения интерполирующей кривой через каждую экспериментальную точку оказывается чрезмерно жестким и не несет физического смысла. В то же время часто, исходя из физических соображений, удается найти предполагаемое аналитическое выражение для зависимости $y_i = F(x_i,\mathbf{p})$, содержащее некоторое ограниченное число неизвестных параметров p_α. В этом случае целесообразно и физически значимо аппроксимировать эту зависимость, используя такое выражение, и определять неизвестные параметры физической модели *методом наименьших квадратов* (2), минимизируя функционал $S(\mathbf{p}) = \sum_1^N (y_i - F(x_i,\mathbf{p}))^2$.

Простейшей ситуацией при этом является случай *линейной модели*, когда

$$F(x_i,\mathbf{p}) = \sum_\alpha p_\alpha F(x_i). \qquad (29)$$

В этом случае поиск параметров p_α сводится к решению системы линейных уравнений

$$\partial S(\mathbf{p}) \big/ \partial p_\alpha = 0, \qquad (30)$$

и проблемы возникают только при плохой обусловленности этой системы.

Но, как правило, физические модели исследуемых процессов не сводятся к простым выражениям вида (29), уравнения (30) оказываются нелинейными, и возникает задача минимизации этого функционала в пространстве нескольких переменных p_α.

Общих методов решения таких задач в настоящее время не существует; они сильно отличаются как по своей природе, так и по математическим особенностям. В зависимости от характера доступной информации о *целевой функции* (2) или (3) различают методы нулевого порядка (имеется только информация о значениях функции в конечном числе точек), первого порядка (известны также значения первых производных целевой функции в этих точках) и второго порядка (используются также сведения о вторых производных).

Большинство этих методов сводится к вводу некоторого начального приближения $p_\alpha{}^o$ для значений искомых параметров, затем по определенному алгоритму находится направление **d** поиска минимума $S(\mathbf{p})$ в многомерном пространстве параметров p_α. В этом направлении осуществляется одномерная минимизация целевой функции. Найденные в результате значения $p_\alpha{}^1$ принимаются в качестве новых начальных значений параметров, и процедура повторяется до достижения заранее поставленных условий остановки. В качестве условий остановки обычно выбираются достижение достаточно малого значения целевой функции, ее производных, выполнение достаточно большого количества итераций, либо некоторую комбинацию этих условий.

Методы нулевого порядка:
минимизация методом прямого поиска

Вообще говоря, методы нулевого порядка, к которым относятся методы прямого поиска, обладают самой медленной скоростью сходимости. Однако, когда число варьируемых параметров является большим, а получение значений производных целевой функции невозможно, либо требует значительных вычислительных мощностей, эти методы могут дать приемлемые результаты.

Достоинством этих методов является также то, что их можно применять не только в методе наименьших квадратов, но и в задачах минимакса, когда целевая функция (3) заведомо недифференцируема.

Методы нулевого порядка:
метод покоординатного спуска

Метод покоординатного спуска является одним из самых простых методов нулевого порядка. Если $\mathbf{e}_\alpha$ – единичные векторы пространства параметров p_α, то на первом шаге в качестве направления поиска минимума принимается $\mathbf{e}_1$ (достигнутое при этом значение параметров обозначим $\mathbf{p}^1$), затем $\mathbf{e}_2$ ($\mathbf{p}^2$), и так далее, до последнего единичного вектора $\mathbf{e}_\omega$ ($\mathbf{p}^\omega$). Затем процедура повторяется. Алгоритм имеет свойство «застревать», если в пространстве параметров имеется направление, в котором целевая функция изменяется особенно сильно (линии равных значений целевой функции имеют вид сильно вытянутого оврага). Чтобы избежать этого, Хуком и Дживсом было предложено дополнить процедуру. После итерации с некоторым заданным номером ω они предложили выполнять поиск минимума в направлении, соединяющем точки $\mathbf{p}^0$ и $\mathbf{p}^\omega$. После этого покоординатный поиск продолжается. Такая модификация существенно повышает эффективность метода.

Существуют также модификации метода координатного спуска, отличающиеся тем, что система единичных векторов $\mathbf{e}_\alpha$ после каждого цикла тем или иным образом модифицируется таким образом, чтобы избежать «застревания» алгоритма и ускорить сходимость метода.

Методы нулевого порядка:
симплексный метод

Этот метод чаще всего используется для минимизации сложных либо недифференцируемых целевых функций с большим числом параметров. По определению *симплексом* называется многогранник в ω-мерном пространстве, имеющий $\omega + 1$ вершину с координатами, равными столбцам матрицы:

$$B_{\omega,\omega+1} = \begin{pmatrix} 0 & b_1 & b_2 & \dots & b_2 \\ 0 & b_2 & b_1 & \dots & b_2 \\ 0 & b_2 & b_2 & \dots & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & b_2 & b_2 & \dots & b_1 \end{pmatrix},$$

$$b_1 = \frac{a}{\omega\sqrt{2}}(\sqrt{\omega+1}+\omega-1), \quad , \qquad (31)$$

$$b_2 = \frac{a}{\omega\sqrt{2}}(\sqrt{\omega+1}-1),$$

a – расстояние между вершинами (сторона) многогранника.

Координаты вершин такого многогранника выбираются в качестве начальных значений параметров $p_\alpha{}^{\text{o}}$.

В каждой из вершин находится значение целевой функции, и точка, в которой значение этой функции максимально, отбрасывается. После этого, используя тот или иной алгоритм, строится новый симплекс.

В стандартном методе симплекса новый многогранник является отражением исходного в плоскости грани, противоположной отбрасываемой точке. В его различных модификациях используются более сложные алгоритмы, допускающие изменение размеров симплекса (увеличение после удачных итераций и сжатие после неудачных), либо его деформацию.

Имеется большое количество библиотечных подпрограмм, реализующих различные варианты этого метода.

Методы нулевого порядка:
случайный поиск

Эффективность методов поиска минимума целевой функции сильно зависит от того, насколько удачно выбрано начальное приближение $p_\alpha{}^{\text{o}}$. В некоторых случаях начальные значения параметров можно выбрать из физических соображений, однако иногда такая возможность отсутствует. Неудачный выбор начальных параметров поиска может привести к тому, что алгоритм окажется в окрестности локального минимума целевой функции, где большинство из них «застревает». Метод случайного поиска в некоторой степени позволяет обойти эту трудность.

Для большинства вариантов этого метода последовательность наборов параметров определяется соотношением:

$$\mathbf{p}^{i+1} = \mathbf{p}^{i} + \rho_i \mathbf{r}, \qquad (32)$$

где ρ_i – величина шага, **r** – случайный вектор. Поиск заключается в генерации заданного числа этих случайных векторов направления поиска, определении соответствующих значений целевой функции, и выбору минимального из них. Соответствующая точка принимается далее за исходную для начала следующей итерации. В простейших вариантах метода величина шага при этом остается неизменной, в более сложных – варьируется, аналогично длине стороны многогранника в предыдущем методе. В алгоритмах *случайного поиска с адаптацией* при последующих генерациях случайных векторов увеличивается вероятность генерации векторов в направлении последнего удачного поиска.

Литература

4. Компьютеры в оптических исследованиях / Под ред. Б. Фридена. – М.: Мир, 1983. – 320 с.
5. Измаилов А. Ф. Численные методы оптимизации: Учебное пособие / А. Ф. Измаилов, М. В. Солодов – М.: Физматлит, 2003. – 304 с.
6. Живописцев Ф. А. Регрессионный анализ в экспериментальной физике / Ф. А. Живописцев, В. А. Иванов – М.: Изд-во МГУ, 1995. – 208 с.
7. Пантелеев А. В. Методы оптимизации в примерах и задачах: Учебное пособие / А. В. Пантелеев, Т. А. Летова – М.: Высшая школа, 2002. – 544 с.

Лекция 18.
Аппроксимация экспериментальных данных методом наименьших квадратов.
Часть 2. Методы первого порядка

План лекции.
Градиентный метод (метод Коши).
Овражный метод
Метод сопряженных градиентов (метод Флетчера-Ривза)

Методы первого порядка:
градиентный метод (метод Коши)

По определению, градиентом функции является вектор ее частных производных по координатам:

$$\mathrm{grad}S(\mathbf{p}) = \left(\frac{\partial S}{\partial p_1}, ..., \frac{\partial S}{\partial p_\omega} \right). \quad (33)$$

Из математического анализа известно, что вектор градиента задает направление наискорейшего возрастания функции. Тогда вектор $-\mathrm{grad}S$ определяет направление ее наискорейшего убывания. Градиентный метод заключается в поиске очередного набора параметров целевой функции, исходя из соотношения:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \rho_i \mathrm{grad}S(\mathbf{p}^i), \quad (34)$$

где величина шага ρ_I на каждой итерации находится одним из методов одномерной минимизации. Различные версии градиентного метода отличаются, главным образом, выбором алгоритма одномерной минимизации.

Градиентный метод является одним из самых медленных среди методов первого порядка и имеет свойство «застревать» в «оврагах» целевой функции (областях со слабой зависимостью целевой функции в одном из направлений пространства параметров), особенно если точность одномерного поиска недостаточно высока. Следующий метод разработан для коррекции этого недостатка.

Методы первого порядка:
овражный метод

В этом методе, в случае застревания градиентного поиска в некоторой точке $\mathbf{p}^1$, производится повторный поиск из другой, близкой к исходной, начальной точки. После того, как метод снова застрянет в некоторой точке $\mathbf{p}^2$, осуществляется одномерный поиск минимума вдоль прямой, проходящей через точки $\mathbf{p}^1$ и $\mathbf{p}^2$. Далее процедура повторяется.

Методы первого порядка:
метод сопряженных градиентов (метод Флетчера-Ривза)

В градиентном методе при выборе очередного направления одномерного поиска не учитывается информация о целевой функции, полученная на предыдущих итерациях. Данный метод осуществляет такой учет. Направление поиска минимума здесь задается соотношением:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \rho_i(\mathbf{g}^i + \lambda_i \mathbf{g}^{i-1}),$$
$$\mathbf{g}^i = \mathrm{grad} S(\mathbf{p}^i), \qquad (35)$$
$$\lambda_i = (\mathbf{g}^i \mathbf{g}^{i-1} - (\mathbf{g}^i)^2)/(\mathbf{g}^{i-1})^2.$$

Используются и другие методы задания параметра сопряжения λ.

Метод сопряженных градиентов сходится быстрее, чем обычный градиентный метод; он особенно эффективен на завершающих этапах минимизации, когда зависимость целевой функции от варьируемых параметров становится слабой.

Литература

8. Компьютеры в оптических исследованиях / Под ред. Б. Фридена. – М.: Мир, 1983. – 320 с.
9. Измаилов А. Ф. Численные методы оптимизации: Учебное пособие / А. Ф. Измаилов, М. В. Солодов – М.: Физматлит, 2003. – 304 с.
10. Живописцев Ф. А. Регрессионный анализ в экспериментальной физике / Ф. А. Живописцев, В. А. Иванов – М.: Изд-во МГУ, 1995. – 208 с.
11. Пантелеев А. В. Методы оптимизации в примерах и задачах: Учебное пособие / А. В. Пантелеев, Т. А. Летова – М.: Высшая школа, 2002. – 544 с.

Лекция 19.
Аппроксимация экспериментальных данных методом наименьших квадратов.
Часть 3. Методы второго порядка

План лекции.
Метод Ньютона-Рафсона.
Метод Ньютона-Гаусса.
Метод Дэвидона-Флетчера-Пауэлла.

Методы второго порядка:
метод Ньютона-Рафсона

Разложим целевую функцию в ряд вблизи начальной точки поиска и ограничимся членами второго порядка малости:

$$S(\mathbf{p}) \approx S(\mathbf{p}^0) + (\mathbf{p} - \mathbf{p}^0)\mathbf{g}^0 + (\mathbf{p} - \mathbf{p}^0)\mathbf{H}(\mathbf{p}^0)(\mathbf{p} - \mathbf{p}^0). \qquad (36)$$

Здесь **H** – матрица Гессе, образованная вторыми производными целевой функции по ее параметрам (иногда матрицей Гессе называют матрицу, обратную матрице вторых производных).

Условием минимума целевой функции является равенство нулю ее градиента, то есть:

$$0=\mathrm{grad}S(\mathbf{p}^{\min}) \approx \mathbf{g}^0 + \mathbf{H}(\mathbf{p}^0)(\mathbf{p}^{\min} - \mathbf{p}^0), \tag{37}$$

откуда

$$\mathbf{p}^{\min} \approx \mathbf{p}^0 - [\mathbf{H}(\mathbf{p}^0)]^{-1}\mathbf{g}^0. \tag{38}$$

Отсюда следует рекуррентное соотношение поиска минимума методом Ньютона:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - [\mathbf{H}(\mathbf{p}^i)]^{-1}\mathbf{g}^i. \tag{39}$$

Метод Ньютона сходится только вблизи точек минимума целевой функции и при положительно определенных матрицах Гессе. Кроме того, вычисление новой матрицы Гессе на каждой итерации требует значительных вычислительных затрат. В связи с этим в практике обычно применяется *обобщенный метод Ньютона*. Матрица Гессе используется для определения направления поиска:

$$\mathbf{d}^i = -[\mathbf{H}(\mathbf{p}^i)]^{-1}\mathbf{g}^i, \tag{40}$$

которое затем используется для одномерной минимизации, аналогично градиентным методам:

$$\mathbf{p}^{i+1} = \mathbf{p}^i + \rho_i \mathbf{d}^i. \tag{41}$$

Различные варианты обобщенного метода Ньютона отличаются используемыми алгоритмами одномерной минимизации.

Метод Ньютона является одним из наиболее эффективных алгоритмов многомерной минимизации. Для целевых функций, являющихся квадратичными формами параметров, он сходится за один шаг. Для сложных целевых функций, с различной зависимостью от входящих в них параметров,

скорость его сходимости может оказаться в 100–1000 раз выше, чем, например, у градиентного метода или его аналогов. Однако недостатком метода является необходимость расчета и обращения громоздкой матрицы Гессе на каждой итерации.

Методы второго порядка:
метод Ньютона-Гаусса

С учетом выражения (2) для целевой функции ее градиент и матрицу Гессе можно представить в виде:

$$g_\alpha = -2\sum_{i=1}^{N}(y_i - F(x_i,\mathbf{p}))\frac{\partial F(x_i,\mathbf{p})}{\partial p_\alpha}, \tag{42}$$

$$\begin{aligned} H_{\alpha\beta} = 2\sum_{i=1}^{N}\frac{\partial F(x_i,\mathbf{p})}{\partial p_\alpha}\frac{\partial F(x_i,\mathbf{p})}{\partial p_\beta} - \\ -2\sum_{i=1}^{N}(y_i - F(x_i,\mathbf{p}))\frac{\partial^2 F(x_i,\mathbf{p})}{\partial p_\alpha \partial p_\beta}. \end{aligned} \tag{43}$$

Введем матрицу:

$$P_{i\alpha} = \frac{\partial F(x_i,\mathbf{p})}{\partial p_\alpha} \tag{44}$$

размера $N \times \omega$. Тогда (43) можно представить в виде:

$$\begin{aligned} &\mathbf{H} = \mathbf{H}^{(1)} - \mathbf{H}^{(2)}, \\ &\mathbf{H}^{(1)} = 2\mathbf{P}^T\mathbf{P}, \\ &H^{(2)}_{\alpha\beta} = 2\sum_{i=1}^{N}(y_i - F(x_i,\mathbf{p}))\frac{\partial^2 F(x_i,\mathbf{p})}{\partial p_\alpha \partial p_\beta}. \end{aligned} \tag{45}$$

Если мы находимся недалеко от минимума, а вторые производные F'' не слишком велики, то элементы матрицы $\mathbf{H}^{(2)}$ малы и $\mathbf{H} \approx \mathbf{H}^{(1)}$. С учетом этого, подставляя (43–35) в (39), получим рекуррентное соотношение для поиска минимума целевой функции методом Ньютона-Гаусса:

$$\mathbf{p}^{i+1} = \mathbf{p}^i + [\mathbf{P}^{iT}\mathbf{P}^i]^{-1}\mathbf{P}^{iT}(\mathbf{Y} - \mathbf{F}(x_i,\mathbf{p}^i)), \tag{46}$$

где **Y** и **F** – векторы, составленные из экспериментально измеренных величин, и рассчитанных значений модельной функции *F* в этих точках при заданных параметрах. По объему вычислений этот метод значительно лучше, чем метод Ньютона, так как не требует вычисления матрицы Гессе, а по скорости сходимости мало ему уступает, особенно вблизи минимума. Для ускорения сходимости этот метод можно модифицировать аналогично методу Ньютона, введя переменную длину итерации:

$$\mathbf{p}^{i+1} = \mathbf{p}^{i} + \rho_i [\mathbf{P}^{iT}\mathbf{P}^{i}]^{-1}\mathbf{P}^{iT}(\mathbf{Y} - \mathbf{F}(x_i, \mathbf{p}^{i})), \tag{47}$$

величина которой определяется методом одномерной минимизации.

Методы второго порядка:
метод Дэвидона-Флетчера-Пауэлла

Другим методом использования сведений о форме целевой функции при сокращении вычислительных затрат на расчеты и обращение матрицы Гессе является ее оценка одним методов конечных разностей на основе сведений, полученных при расчете целевой функции и ее первых производных в нескольких точках. Метод Дэвидона-Флетчера-Пауэлла является одной из реализаций этой идеи. Для оценки матрицы, обратной матрице Гессе, здесь используется выражение:

$$(H)^{-1i+1}_{\alpha\beta} = (H)^{-1i}_{\alpha\beta} - \frac{(p_\alpha^{i+1} - p_\alpha^{i})d_\beta^{i}}{\sum_\gamma (g_\gamma^{i+1} - g_\gamma^{i})d_\gamma^{i}} - \frac{\sum_\gamma (H)^{-1i}_{\alpha\gamma}(g_\gamma^{i+1} - g_\gamma^{i})\sum_\gamma (H)^{-1i}_{\beta\gamma}(g_\gamma^{i+1} - g_\gamma^{i})}{\sum_{\gamma\delta} (g_\gamma^{i+1} - g_\gamma^{i})(H)^{-1i}_{\gamma\delta}(g_\delta^{i+1} - g_\delta^{i})}, \tag{48}$$

и далее используется итерационная процедура, аналогичная (40–41). На первом шаге в качестве оценки матрицы, обратной матрице Гессе, используется единичная матрица. Таким образом, на первом шаге этот метод совпадает с градиентным (34), а затем, по мере приближения к точке минимума, из-за уменьшения длины одномерных итераций ρ разностная оценка (48) становится все более точной и метод сводится к обобщенному методу Ньютона. Существуют разновидности этого метода, отличающиеся, видом разностной схемы, которая используется для оценки матрицы Гессе,

что приводит к несколько иному виду выражения (48); все эти алгоритмы признаны надежным методом решения задач оптимизации, хотя и требуют достаточно больших вычислительных ресурсов на каждом шаге итерации.

Литература

1. Компьютеры в оптических исследованиях / Под ред. Б. Фридена. – М.: Мир, 1983. – 320 с.
2. Измаилов А. Ф. Численные методы оптимизации: Учебное пособие / А. Ф. Измаилов, М. В. Солодов – М.: Физматлит, 2003. – 304 с.
3. Живописцев Ф. А. Регрессионный анализ в экспериментальной физике / Ф. А. Живописцев, В. А. Иванов – М.: Изд-во МГУ, 1995. – 208 с.
4. Пантелеев А. В. Методы оптимизации в примерах и задачах: Учебное пособие / А. В. Пантелеев, Т. А. Летова – М.: Высшая школа, 2002. – 544 с.

Приложение 1

Пример библиотеки для работы с системой КАМАК с контроллером, использующем параллельный интерфейс.

```
unit CAMAC;

interface

procedure CamO(const n, F, a:byte; const d:Word);
procedure CamO24(const n, F, a:byte; const d:longint);
procedure CamI(const n, F, a:byte; var d:Word);
procedure CamI24(const n, F, a:byte; var d:longint);
procedure CamDrv(const n, F, a:byte);
procedure CamCl(const i:byte);
procedure CamL(var l:byte);
Function CamQ: boolean;
Function CamX: boolean;

implementation
{$R-,S-}
procedure CamO(const n, F, a:byte; const d:Word);
begin
   portw[$0240]:=$000000;
   portw[$0241]:=(d and $00FF00) shr 8;
   portw[$0242]:=d and $0000FF;
   portw[$0243]:=a;
   portw[$0244]:=F;
   portw[$0245]:=n;
   portw[$0247]:=1;
end;

procedure CamO24(const n, F, a:byte; const d:longint);
begin
   portw[$0240]:=(d and $FF0000) shr 16;
   portw[$0241]:=(d and $00FF00) shr 8;
   portw[$0242]:=d and $0000FF;
   portw[$0243]:=a;
   portw[$0244]:=F;
   portw[$0245]:=n;
   portw[$0247]:=1;
end;
procedure CamI(const n, F, a:byte; var d:Word);
var c : integer;
begin
```

```
   portw[$0243]:=a;
   portw[$0244]:=F;
   portw[$0245]:=n;
   portw[$0247]:=1;
      d:=((portw[$024A] and
      $0000FF)*$100)+(portw[$024B] and $0000FF);
end;

procedure CamI24(const n, F, a :byte; var d:longint);
var c,e : longint;
begin
   portw[$0243]:=a;
   portw[$0244]:=F;
   portw[$0245]:=n;
   portw[$0247]:=1;
      d:=((portw[$0249] and
      $0000FF)*$10000)+((portw[$024A] and
      $0000FF)*$100)+( portw[$024B] and $0000FF);
end;

procedure CamDrv(const n, F, a:byte);
var c,e:longint;
begin
   portw[$0243]:=a;
   portw[$0244]:=F;
   portw[$0245]:=n;
   portw[$0247]:=1;
end;

procedure CamCl(const i:byte);
begin
   portw[$0246]:=i;
end;

procedure CamL(var l:byte);
begin
   l:=(portw[$0248] and $00007C) shr 2;
end;

Function CamQ: boolean;
begin
   CamQ:=(portw[$0248] and $000001) = 1;
end;

Function CamX: boolean;
```

```
begin
   CamX:=((portw[$0248] and $000002) shr 1) = 1;
end;

end.
```

Приложение 2

Пример библиотеки для работы с системой КАМАК с контроллером, использующем последовательный интерфейс.

```
unit camac;
interface

procedure GTW;
function F0(K,N,A,F: Byte): integer;
                { Функция типа «Ввод” }

function CTRLQ(K: byte): boolean;

procedure F16(K,N,A,F: Byte; W: Word);
                { Процедура типа «Вывод” }
                { K – номер канала }
                { N – номер модуля }
                { A – субадрес }
                { F – функция }

procedure Zero;

implementation
uses CRT;

const           {Адреса регистров драйвера.}
   RGAS=$310;
   RGAM=$314;
   RGDS=$312;
   RGDM=$316;
   RGZ=$318;
var
   AdrS, AdrM: byte;

procedure GTW; {Проверка конца передачи по посл. каналу.}
var
   GOT, N :integer;
begin
   N:=0;
   repeat
      GOT:=port[RGZ];
      N:=N+1;
   until (GOT and 16=16)  {Выход если готов }
   or (N>100);    {или отсутствует связь}
```

```
   if N>100 then WRITELN  ('Нет связи с контроллером.')
end;

function F0(K,N,A,F: Byte): integer;
var
   STB,MLB: integer;

begin
   AdrS:=(8 shl K)+((N and 24) shr 3)+4;
   AdrM:=((N and 7) shl 5)+(A shl 1);
   port[RGAS]:=(8 shl K)+8;
   port[RGAM]:=0;
   port[RGDS]:=0;
   port[RGDM]:=F;
   GTW;

   port[RGAS]:=AdrS;
   port[RGAM]:=AdrM;
   GTW;

   STB:=port[RGDS];
   MLB:=port[RGDM];
   F0:=(STB shl 8)+MLB
end;

function CTRLQ(K: byte): boolean; { Проверка состояния Q. }
begin
   CTRLQ:=false;
   port[RGAS]:=(8 shl K)+4;
   port[RGAM]:=0;
   GTW;
   if (port[RGDS] and $80) = $80 then CTRLQ:= true
end;

procedure F16(K,N,A,F: Byte; W: Word);
begin
   port[RGAS]:=(8 shl K)+8;
   port[RGAM]:=0;
   port[RGDS]:=0;
   port[RGDM]:=F;
   AdrS:=(8 shl K)+((N and 24) shr 3)+8;
   AdrM:=((N and 7) shl 5)+(A shl 1);
   port[RGAS]:=AdrS;
   port[RGAM]:=AdrM;
   port[RGDS]:=Hi(W);
   port[RGDM]:=Lo(W)
end;
```

```
procedure Zero;
var
   K:  byte;
begin
   for K:=1 to 4 do
   begin
      port[RGAS]:=(8 shl K)+8;
      port[RGAM]:=0;
      port[RGDS]:=1;
      port[RGDM]:=0
   end;
   delay(100)
end;

end.
```

Приложение 3

Заголовочный файл и пример реализации некоторых функций библиотеки «gen_func.dll» для работы с КАМАК под управлением контроллера с параллельным интерфейсом через драйвер «**genport»,** для операционных систем Windows NT/2000/XP.

С полными текстами приведенных библиотек можно ознакомится по адресу http://www.kirensky.ru/master/camac/.

«main.h»

```
void __declspec(dllexport) __stdcall Camac_Port_Init();
// Функция, инициализирующая работу с системой КАМАК,
открытие «файла драйвера устройства» на чтение и на
запись.

bool __declspec(dllexport) __stdcall Ca-
mac_Port_Close();
// Функция, закрывающая «файл драйвера устройства» от
чтения и записи. В случае успешного выполнения этих
операций возвращает True, в противном случае Fasle.

void __declspec(dllexport) __stdcall CamO(const un-
signed char n, const unsigned char f, const unsigned
char a, long d);
// Функция, отправляющая модулю команду и данные.
// В качестве аргументов передается адрес ячейки
// модуля (n), субадрес (a), функция (f) и данные (d).
// Используются шины записи.

long __declspec(dllexport) __stdcall CamI(const un-
signed char n, const unsigned char f, const unsigned
char a, long d);
// Функция, отправляющая модулю команду и получающая от
// него данные. В качестве аргументов передается адрес
// ячейки модуля (n), субадрес (a), функция (f) и
// данные (d). Используются шины чтения.

void __declspec(dllexport) __stdcall CamDrv (const un-
signed char n, const unsigned char f, const unsigned
char a);
// Функция, отправляющая модулю управляющую команду без
// использования шин даннх. В качестве аргументов
```

```
// передается адрес ячейки модуля (n), субадрес (a) и
// функция (f).

void __declspec(dllexport) __stdcall Waiting(const un-
signed char n);
// Функция приостановки работы программы до получения
// сигнала Q от модуля расположенного в станции с
// номером n.

void __declspec(dllexport) __stdcall WaitingExt(const
unsigned char n, const unsigned char a);
// Функция приостановки работы программы до получения
// сигнала Q от модуля расположенного в станции с
// номером n при проверке субадреса a.

void __declspec(dllexport) __stdcall SetC();
// Функция, вызывающая передачу сигнала гашения.

void __declspec(dllexport) __stdcall SetZ();
// Функция, вызывающая передачу сигнала подготовки.

void __declspec(dllexport) __stdcall SetI();
// Функция, вызывающая передачу сигнала блокировки.

unsigned int __declspec(dllexport) __stdcall CamL();
// Функция, вызывающая проверку сигнала требования на
обслуживание. Возвращает адрес модуля выставившего
требование. Наличие сигнала L означает необходимость
прервать текущую программу и начать выполнение
программы обслуживания модуля.

bool __declspec(dllexport) __stdcall CamQ();
// Проверка сигнала подтверждения Q.

bool __declspec(dllexport) __stdcall CamX();
// Проверка сигнала X. Наличие сигнала обозначает, что
// модуль принял адресованную ему команду и готов к ее
// исполнению.

long __declspec(dllexport) __stdcall CamI(const un-
signed char n, const unsigned char f, const unsigned
char a);
// Проверка сигнала блокировки I.
```

«main.cpp»

```
HANDLE hndFile_Write;
// Переменная устройства для записи.
HANDLE hndFile_Read;
// Переменная устройства для чтения.


void __stdcall Camac_Port_Init()
{
   hndFile_Read = CreateFile(
                  "\\\\.\\GpdDev",
                  GENERIC_READ,
                  FILE_SHARE_READ,
                  NULL,
                  OPEN_EXISTING,
                  0,
                  NULL);

   if (hndFile_Read == INVALID_HANDLE_VALUE)
   {
      throw(/*CAMAC_Except::err=
           */"Unable to open the device.");
      exit(1);
   }
   hndFile_Write = CreateFile(
                     "\\\\.\\GpdDev",
                     GENERIC_WRITE,
                     FILE_SHARE_WRITE,
                     NULL,
                     OPEN_EXISTING,
                     0,
                     NULL);
   if (hndFile_Write == INVALID_HANDLE_VALUE)    {
      throw(/*CAMAC_Except::err=
           */"Unable to open the device.");
      exit(1);
   }
}

bool __stdcall Write_To_Port(GENPORT_WRITE_INPUT
NewInformation)
{
   BOOL                 IoctlResult;
   LONG                 IoctlCode;
```

```
   ULONG               DataLength;
   ULONG               ReturnedLength;

   IoctlCode = IOCTL_GPD_WRITE_PORT_UCHAR;
   DataLength = offsetof(GENPORT_WRITE_INPUT, CharData)
   +sizeof(NewInformation.CharData);
   IoctlResult = DeviceIoControl(
          hndFile_Write,
          IoctlCode,
          &NewInformation,
          DataLength,
          NULL,
          0,
          &ReturnedLength,
          NULL);
   if (IoctlResult)
   {
      return true;
   }
   else
   {
      return false;
   }
}

void __stdcall CamO(const unsigned char n, const
unsigned char f, const unsigned char a, long d)
{
   GENPORT_WRITE_INPUT InputBuffer;

   InputBuffer.PortNumber = 0x0;
   InputBuffer.CharData = (d & 0xFF0000) >> 16;
   Write_To_Port(InputBuffer);

   InputBuffer.PortNumber = 0x1;
   InputBuffer.CharData = (d & 0x00FF00) >> 8;
   Write_To_Port(InputBuffer);

   InputBuffer.PortNumber = 0x2;
   InputBuffer.CharData = d & 0x0000FF;
   Write_To_Port(InputBuffer);

   InputBuffer.PortNumber = 0x3;
   InputBuffer.CharData = a;
   Write_To_Port(InputBuffer);
```

```
   InputBuffer.PortNumber = 0x4;
   InputBuffer.CharData = f;
   Write_To_Port(InputBuffer);

   InputBuffer.PortNumber = 0x5;
   InputBuffer.CharData = n;
   Write_To_Port(InputBuffer);

   InputBuffer.PortNumber = 0x7;
   InputBuffer.CharData = 0x1;
   Write_To_Port(InputBuffer);
}

void __stdcall SetZ()
{
   GENPORT_WRITE_INPUT InputBuffer;

   InputBuffer.PortNumber = 0x6;
   InputBuffer.CharData = 0x1;
   Write_To_Port(InputBuffer);
}

bool __stdcall CamQ()
{
   ULONG   PortNumber;
   UCHAR   Data;

   PortNumber = 0x8;
   Data = Read_From_Port(PortNumber);
   return (Data & 0x000001) == 1;
}


void __stdcall Waiting(const unsigned char n)
{
   do
   {
      CamDrv(n, 8, 0);
      Sleep(1);
   } while (!CamQ());
}
```

Литература

1. Ступин Ю. В. Методы автоматизации физических экспериментов с помощью ЭВМ / Ю. В. Ступин – М.: Энергоатомиздат, 1983. – 160 с.
2. Певчев Ю. Ф. Автоматизация физического эксперимента / Ю. Ф. Певчев, К. Г.Финогенов – М.: Энергоатомиздат, 1986. – 254 с.
3. Воробьев В. И. Математическое обеспечение ЭВМ в науке и производстве / В. И. Воробьев – Л.: Машиностроение, 1988. – 158 с.
4. Новиков Ю. В. Разработка устройств сопряжения / Ю. В. Новиков, О. А. Калашников, С. Э. Гуляев – М., ЭКОМ, 1997.– 224 с.
5. Соломенчук В. Аппаратные средства персональных компьютеров. / В. Соломенчук – СПб.: БХВ-Петербург, 2003. – 512 с.
6. Ан П. Сопряжение ПК с внешними устройствами / Пей Ан – М.: ДМК Пресс, 2004. – 320 с.
7. Кулаков В. Программирование на аппаратном уровне. Специальный справочник / В. Кулаков – М.: Питер, 2003. – 848 с.
8. Смит Дж. Сопряжение компьютеров с внешними устройствами. Уроки реализации / Дж. Смит – М.: Мир, 2000. – 266 с.
9. Гук М. Аппаратные средства IBM PC. Энциклопедия. / М. Гук – СПб, Питер, 2006. – 1072 с.
10. Гук М. Шины PCI, USB и FireWire. Энциклопедия. / М. Гук – СПб, Питер, 2005. – 544 с.
11. Гук М. Аппаратные интерфейсы ПК. Энциклопедия. / М. Гук – СПб, Питер, 2003. – 528 с.
12. Курочкин С. С. Системы КАМАК–ВЕКТОР / С. С. Курочкин – М.: Радио и связь., 1981. – 160 с.
13. CAMAC — A Modular Instrumentation System for Data Handling. Revised Description and Specification. Report. EUR 4100e, CEC, Luxembourg 1972; deutsche Ubersetzung; EUR 4100d; revised form: IEEE Standard 583-1975, IEC Recommendation 516; Bloc Transfers in CAMAC Systems, Supplement EUR 4100e, CEC, Luxembourg 1977; IEEE Standard 683-1976.
14. CAMAC — Organization of Multi-Crate System. Specification of the Branch Highway and CAMAC Crate Controller Typ A, Report EUR 4600e, CEC, Luxembourg 1972; revised form: IEEE Standard 596-1976.
15. CAMAC — A Modular Instrumentation System for Data Handling, Specification of Amplitude Analogue Signals, Report EUR 5100e, CEC, Luxembourg 1972; CAMAC Bulletin No. 8, 28 (1973).
16. Государственный стандарт Союза ССР. Единая система стандартов приборостроения. Система КАМАК. Крейт и сменные блоки. Требования к конструкции и интерфейсу. ГОСТ 26.201–80.

17. Науман Г. Стандартные интерфейсы для измерительной техники / Г. Науман, В. Майлинг, А. Щербина – М.: Мир, 1982. – 230 с.
18. http://www.ni.com/pxi/
19. http://www.pxisa.org/
20. http://www.pxionline.com/
21. http://www.vxitech.com/
22. http://www.vxibus.ru/index.shtml?2_1
23. http://www.vxipnp.org/
24. Автоматизация физических исследований и эксперимента: компьютерные измерения и виртуальные приборы на основе LabVIEW 7 / Под ред. Бутырина П. А. – М.: ДМК Пресс, 2005. – 264 с.
25. Тревис Дж. LabVIEW для всех / Джеффри Тревис – М.: ДМК Пресс, 2004. – 544 с.
26. Евдокимов Ю. К. LabVIEW для радиоинженера: от виртуальной модели до реального прибора. / Ю. К. Евдокимов, В. Р. Линдваль, Г. И. Щербаков – М.: ДМК Пресс, 2007. – 512 с.
27. Суранов А. Я. LabVIEW 7: справочник по функциям / А. Я. Суранов – М.: ДМК Пресс, 2007. – 512 с.
28. Каратаев В. Автоматизация физических исследований и эксперимента: компьютерные измерения и виртуальные приборы на основе LabVIEW 7 / В. Каратаев, Т. Васильковская, П. А. Бутырин – М.: ДМК Пресс, 2004 – 480 с.
29. Батоврин В. К. LabVIEW: Практикум по основам измерительных технологий / В. К. Батоврин, А. С. Бессонов, В. В. Мошкин, В. Ф. Популовский – М.: ДМК Пресс, 2005. – 208 с.
30. http://www.ni.com/labview
31. http://www.ni.com/russia
32. Компьютеры в оптических исследованиях / Под ред. Б. Фридена. – М.: Мир, 1983. – 320 с.
33. Отнес Р. Прикладной анализ временных рядов / Р. Отнес, Л. Эноксон – М., Мир, 1982. – 424 с.
34. Носач В. В. Решение задач аппроксимации с помощью персональных компьютеров / В. В. Носач – М.: МИКАП, 1994. – 382 с.
35. Измаилов А. Ф. Численные методы оптимизации: Учебное пособие / А. Ф. Измаилов, М. В. Солодов – М.: Физматлит, 2003. – 304 с.
36. Статистическая обработка результатов экспериментов на микро-ЭВМ / А. А. Костылев, П. В. Миляев, Ю. Д. Дорский и др. – Л.: Энергоатомиздат, 1991. – 304 с.

37. Живописцев Ф. А. Регрессионный анализ в экспериментальной физике / Ф. А. Живописцев, В. А. Иванов – М.: Изд-во МГУ, 1995. – 208 с.

38. Пантелеев А. В. Методы оптимизации в примерах и задачах: Учебное пособие / А. В. Пантелеев, Т. А. Летова – М.: Высшая школа, 2002. – 544 с.

39. Втюрин А. Н. ЭВМ в физическом эксперименте. Учебное пособие / А. Н. Втюрин, А. Г. Агеев, А. С. Крылов – Новосибирск, Изд-во СО РАН, 2003. – 150 с.

Оглавление