**Курсовая работа**

**"Микроконтроллеры для начинающих. И не только"**

## Введение

**Микроконтро́ллер** (англ. *Micro Controller Unit, MCU*) – микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает в себе функции процессора и периферийных устройств, может содержать ОЗУ и ПЗУ. По сути, это однокристальный компьютер, способный выполнять простые задачи. Большая часть выпускаемых в современном мире процессоров – микроконтроллеры.

Об однокристальных микро-ЭВМ или микроконтроллерах в журнале «Радио» писалось не раз. Это – и цикл статей об однокристальных микро-ЭВМ 8048, 8051, 8052, и отдельные обзорные статьи о микроконтроллерах новых семейств MCS-96, MCS-151, MCS-152, PIC16CХХ, Z8 и т.д., и описания программно-аппаратных средств отладки микроконтроллерных устройств. Видимо, пришла пора систематизировать сведения об этих интереснейших и всемогущих изделиях электронных фирм, познакомить читателей с микроконтроллерами, выпускаемыми в мире сегодня, рассказать о том, что они могут и как это реализуется на практике, о современных средствах программирования и отладки устройств на основе микроконтроллеров, в частности, о том, что можно сделать, не имея средств отладки или пользуясь программным симулятором-отладчиком, который предполагается разместить на сайте журнала «Радио» в Интернете, и т.д. Данный цикл статей адресован в первую очередь читателям, ранее не имевшим дела с микроконтроллерами, но, несомненно, будет полезен и тем, кто уже применял их в своих конструкциях.

## 1. Первое знакомство

Вначале несколько слов для тех, кому тема цикла, если судить по его названию, кажется априорно неинтересной или «чужой». Возможно, Вы в своих конструкциях до сих пор не применяли микроконтроллеры (далее для краткости МК) и считаете, что и в обозримом будущем сможете обходиться без них. Возможно также, Вы предполагаете, что создание микроконтроллерной системы для решения Вашей задачи будет слишком обременительным и экономически нецелесообразным. Не спешите: специально для Вас мы хотим привести несколько фактов и немного статистики.

Для примера возьмём ближайшего родственника МК – персональный компьютер – и сравним интенсивности их применения. По данным аналитической компании Loewenbaum & Co. Inc. (США), число персональных компьютеров, выпущенных в мире в 1997 г., достигло примерно 20 млн. шт. Согласитесь, это очень много. А теперь представьте, что это гигантское число составляет всего лишь 0,2% от мирового объёма выпуска МК. По данным аналитической компании IC Insights Inc. (США) мировой рынок в 1998 г. поглотил их более 13,5 млрд. шт.!

Вывод напрашивается сам. Если уже сегодня трудно найти область деятельности человека, где бы эффективно не использовался компьютер, то что же тогда говорить о МК? Почему они стали такими популярными и буквально незаменимыми? Ответ кроется в самой структуре микроконтроллера. В качестве первого приближения к определению этого понятия можно считать, что МК – это компьютер, разместившийся в одной микросхеме. Отсюда и его основные привлекательные качества: малые габариты, потребление, цена; высокие производительность, надёжность и способность быть адаптированным для выполнения самых различных задач.

МК отличается от микропроцессора тем, что помимо центрального процессора (ЦП) содержит память и многочисленные устройства ввода / вывода: аналого-цифровые преобразователи, последовательные и параллельные каналы передачи информации, таймеры реального времени, широтно-импульсные модуляторы (ШИМ), генераторы программируемых импульсов и т.д. По своей структуре и принципу функционирования МК, в сущности, не отличается от персонального компьютера. Поэтому слова микроконтроллер и микро-ЭВМ являются синонимами. Однако первый термин (от английского слова control – управлять) более распространён, поскольку отражает его основное назначение – использование в системах автоматического управления, встроенных в самые разные устройства: кредитные карточки, фотоаппараты, сотовые телефоны, музыкальные центры, телевизоры, видеомагнитофоны и видеокамеры, стиральные машины, микроволновые печи, системы охранной сигнализации, системы зажигания бензиновых двигателей, электроприводы локомотивов, ядерные реакторы и многое, многое другое. Встраиваемые системы управления стали настолько массовым явлением, что фактически сформировалась новая отрасль экономики, получившая название Embedded Systems (встраиваемые системы – англ.).

В настоящее время в мире выпускаются тысячи разновидностей МК. Они поставляются в корпусах с числом выводов от 8 до 356, работают при температуре от –55 до +125 °C на частотах от 32 кГц до 200 МГц, способны функционировать при напряжении питания от 1,2 В, потребляя при этом ток, не превышающий единицы микроампер. Цена изделий также непрерывно снижается. Некоторые восьмиразрядные МК уже сегодня стоят не дороже 50 центов, что сопоставимо со стоимостью одной микросхемы «жёсткой логики». Все это привело к тому, что сегодня всё труднее найти область человеческой деятельности, где бы МК не нашли применения. И процесс их распространения имеет лавинообразный характер.

Надеемся, что приведённые факты уже настроили Вас на почтительное отношение к главному герою нашего повествования. Действительно, МК стал событием мирового масштаба, вторгшимся практически во все виды человеческой деятельности.

Что же обеспечило такой бурный рост популярности этих изделий, появившихся немногим более 25 лет назад? Что это за устройства, и каковы их возможности и перспективы?

Если Вы до сих пор в своей деятельности не использовали МК или системы на их основе, то, может быть, настало время подумать об этом? А если Вы решились применить МК, то какова должна быть последовательность Ваших действий? Какие трудности могут Вас поджидать, что может Вам помочь на этом пути?

На эти вопросы мы и попытаемся ответить.

## 2. Закон Мура и первый МК

Ещё в 1965 г. Гордон Мур (Gordon Moore), один из будущих основателей могущественной корпорации Intel, обратил внимание на интереснейший факт. Представив в виде графика рост производительности запоминающих микросхем, он обнаружил любопытную закономерность: новые модели микросхем появлялись каждые 18–24 месяца, а их ёмкость при этом возрастала каждый раз примерно вдвое. Если такая тенденция продолжится, предположил Г. Мур, то мощность вычислительных устройств экспоненциально возрастёт на протяжении относительно короткого промежутка времени.

Предвидение Г. Мура впоследствии блестяще подтвердилось, а обнаруженная им закономерность наблюдается и в наши дни, причём с поразительной точностью, являясь основой для многочисленных прогнозов роста производительности. За 28 лет, истекшие с момента появления микропроцессора 4004 (1971 г.), число транзисторов на кристалле выросло более чем в 12 000 раз: с 2 300 до 28 000 000 в микросхеме Coppermine.

Ну а в 1976 г. экспоненциальное развитие полупроводниковой технологии привело к созданию фирмой Intel первого МК – 8048. Помимо ЦП, в его состав входила память программ, память данных, восьмибитный таймер и 27 линий ввода / вывода. Сегодня 8048 является уже достоянием истории, а вот следующее изделие, выпущенное Intel в 1980 г., живёт и здравствует поныне. Это – МК 8051.

**Архитектура МК 8051**

Этот МК можно считать классическим образцом, по образу и подобию которого позднее было создано множество других изделий. Его структурная схема представлена на **рис.****1.** ЦП – главный узел МК. С ним связано такое важнейшее понятие, как система команд.

**Система команд** – это уникальный, характерный для данного ЦП набор двоичных кодов, определяющих перечень всех его возможных операций. Каждый такой код определяет одну операцию и называется **кодом операции** или командой. Чем больше кодов используется в системе команд, тем больше операций способен выполнить ЦП. МК 8051 – восьмиразрядный, поэтому коды операций у него имеют размер 8 бит. Теоретически может быть всего 256 восьмибитных кодов операций. В 8051 используются 255.

В зависимости от числа использованных кодов операций, системы команд подразделяют на две группы: **CISC** и **RISC**. Термин CISC означает сложную систему команд и является аббревиатурой английского определения Complex Instruction Set Computer. Аналогично термин RISC означает сокращённую систему команд и происходит от английского Reduced Instruction Set Computer. Систему команд МК 8051 можно отнести к типу CISC.

Однако, несмотря на широкую распространённость этих понятий, необходимо признать, что сами названия не отражают главного различия между системами команд CISC и RISC. Основная идея RISC-архитектуры – эго тщательный подбор таких комбинаций кодов операций, которые можно было бы выполнить за один такт тактового генератора. Основной выигрыш от такого подхода – резкое упрощение аппаратной реализации ЦП и возможность значительно повысить его производительность.

Первоначально реализовывать такой подход удавалось, лишь существенно сократив набор команд, отсюда и родилось название RISC. Например, система команд МК семейства Microchip PIC16 включает в себя всего 35 инструкций и может быть отнесена к типу RISC. Очевидно, что в общем случае одной команде CISC-архитектуры должны соответствовать несколько команд RISC-архитектуры. Однако обычно выигрыш от повышения быстродействия в рамках RISC-архитектуры перекрывает потери от менее эффективной системы команд, что приводит к более высокой эффективности RISC-систем в целом по сравнению с CISC. Так, самая быстрая команда МК 8051 выполняется за 12 тактов. Даже если для каждой инструкции потребуется выполнить три инструкции RISC-контроллера, то в итоге RISC-архитектура обеспечит четырёхкратное увеличение производительности.

Попутно RISC-архитектура позволяет решить ещё ряд задач. Ведь с упрощением ЦП уменьшается число транзисторов, необходимых для его реализации, следовательно, уменьшается площадь кристалла. А с этим связано снижение стоимости и потребляемой мощности.

В этом месте можно было бы воскликнуть: будущее – за RISC-архитектурой! Однако в настоящее время грань между этими двумя понятиями стремительно стирается. Например, МК семейства AVR фирмы Atmel имеют систему команд из 120 инструкций, что соответствует типу CISC. Однако большинство из них выполняется за один такт, что является признаком RISC-архитектуры. Сегодня принято считать, что основным признаком RISC-архитектуры является выполнение команд за один такт тактового генератора. Число команд само по себе значения уже не имеет.

**Тактовый генератор** вырабатывает импульсы для синхронизации работы всех узлов устройства. Частоту их следования могут задавать кварцевый резонатор или RC-цепь, подключаемые к выводам МК. В некоторых МК предусмотрен режим работы тактового генератора без применения внешних элементов. В этом случае частота тактовых импульсов зависит от параметров кристалла, определяемых в процессе его производства.

**ПЗУ** – постоянное запоминающее устройство, предназначенное для хранения программ, поэтому часто эту память называют кодовой или памятью программ. До недавнего времени существовало две основных разновидности ПЗУ **масочные** и **программируемые**.

В масочные ПЗУ информацию заносят в процессе изготовления МК с помощью технологических шаблонов – масок. Изменить её после окончания производственного цикла невозможно.

Такие ПЗУ используют лишь в случаях, когда качество программы не вызывает сомнений и существует массовая потребность в МК именно с этой программой. Достоинство масочных ПЗУ – самая низкая стоимость при массовом производстве (от нескольких тыс. шт.).

В программируемые ПЗУ информацию записывают с помощью устройства, называемого программатором. МК с такими ПЗУ бывают двух типов: **однократно** и **многократно программируемые** (перепрограммируемые). Первые, как говорит само название, допускают только однократное программирование, после чего стереть информацию уже невозможно (МК с OTP-памятью – от англ. One Time Programmable). Используют их в мелкосерийном производстве (до 1000 шт.), когда применение масочных МК экономически не оправдано.

Многократно программируемые микросхемы подразделяются на МК, оснащённые ПЗУ со стиранием ультрафиолетовым облучением (выпускаются в корпусах с «окном»), и МК с электрически перепрограммируемой памятью. Недостаток МК с ПЗУ со стиранием ультрафиолетовым облучением – очень высокая стоимость и относительно небольшое число циклов записи / стирания (зависит от суммарной дозы облучения кристалла и обычно не превышает 15…20).

В настоящее время все более популярной становится новая технология реализации ПЗУ – **Flash**-память. Её главное достоинство в том, что она построена на принципе электрической перепрограммируемости, то есть допускает многократное стирание и запись информации с помощью программаторов. Минимальное гарантированное число циклов записи / стирания обычно превышает несколько тысяч. Это существенно увеличивает жизненный цикл и повышает гибкость МК-систем, так как позволяет вносить изменения в программу МК как на этапе разработки системы, так и в процессе его работы в реальном устройстве.

**ОЗУ** – оперативное запоминающее устройство, используемое для хранения данных, поэтому эту память называют еще памятью данных. Число циклов чтения и записи в ОЗУ не ограничено, но при отключении питающего напряжения вся информация теряется.

Архитектура МК 8051 предполагает раздельное использование памяти программ и данных и носит название **гарвардской**. Обычно такую архитектуру используют для повышения быстродействия системы за счёт разделения путей доступа к памяти программ и данных, но в 8051 она была применена с целью получения памяти программ и данных, не требующих одинакового размера. Антипод гарвардской – **архитектура фон** **Неймана** – предполагает хранение программ и данных в общей памяти и наиболее характерна для микропроцессоров, ориентированных на использование в компьютерах. Примером могут служить микропроцессоры семейства х86.

**Таймеры Т0, Т1** – шестнадцатиразрядные программируемые таймеры / счётчики, которые могут быть запрограммированы на выполнения целого ряда функций. Их можно использовать для точного формирования временных интервалов, подсчёта импульсов на выводах МК, формирования последовательности импульсов, тактирования приёмопередатчика последовательного канала связи. Таймеры/счётчики способны вырабатывать запросы прерываний, переключая ЦП на их обслуживание по событиям и освобождая его от необходимости периодического опроса состояния таймеров. Поскольку основное применение МК находят в системах реального времени, таймеры / счётчики являются их обязательным элементом. В некоторых модификациях число таймеров достигает 32.

**Последовательный порт** – канал информационного обмена МК с внешним миром. Такие каналы связи занимают минимальное число выводов кристалла, обеспечивая связь на значительные расстояния с минимальными аппаратными затратами, В 8051 реализован универсальный асинхронный последовательный приёмопередатчик (UART), поддерживающий протокол стандарта RS-232C, что обеспечивает возможность организации связи этого МК и персональным компьютером. Кроме RS-332C, популярными протоколами в мире встраиваемых систем являются RS-485, I2C (двухпроводная двунаправленная шина), SPI (последовательный периферийный трёхпроводный интерфейс), Bitbus (последовательная магистраль управления), CAN (межконтроллерный сетевой интерфейс), USВ (универсальная последовательная шина) и некоторые другие. Практически для любого типа последовательного канала сегодня можно найти МК, имеющий в своем составе соответствующий последовательный порт.

**Параллельные порты** ввода / вывода – также обязательная часть любого МК. Обычно их используют для связи с ближайшим окружением – датчиками и исполнительными механизмами.

Важная особенность параллельных портов МК – возможность программирования на выполнение нескольких функций. Например, в 8051 выводы портов P0 и P2 могут использоваться либо как обычные статические регистры ввода / вывода, либо в качестве шины адреса и данных для подключения внешних устройств, таких как дополнительная память программ, память данных, устройства ввода / вывода. Это придаёт МК архитектурную гибкость. Порт P3 может либо использоваться как статический регистр ввода / вывода, либо выполнять специальные функции для работы последовательного канала, таймеров, контроллера прерываний и т.д. Возможность перепрограммирования позволяет с максимальной эффективностью задействовать все выводы МК в проектируемом устройстве.

**Система прерываний** – одна из важнейших частей МК. Особенность систем реального времени заключается в том, что для них чрезвычайно важным параметром является время реакции на внешние события. Поясним на простом примере. Когда Вы производите математический расчёт на компьютере, то обычно запускаете программу, предназначенную для выполнения этих расчётов, и после того, как она загрузится в память компьютера, вводите условие задачи и ждёте результата. Время ожидания в таком случае не имеет принципиального значения (в пределах разумного, конечно) – медленная работа компьютера может раздражать, но на результате это не скажется. Система реального времени предполагает совершенно конкретную, рассчитываемую на этапе разработки скорость реакции системы управления на внешние события. Задержки сверх расчётных здесь просто недопустимы – они могут приводить к катастрофическим последствиям.

Проблемы быстрой реакции на события решаются организацией системы прерываний. Она подразумевает, что для каждого такого события разрабатывается отдельный «кусок» кода, который формирует реакцию МК на него. Этот «кусок» кода называют подпрограммой обработки запроса на прерывание (для краткости часто используют термин **подпрограмма прерывания**) и размещают в памяти программ по известному адресу. В момент возникновения заданного события сигнал об этом поступает на вход **контроллера прерываний**. Последний представляет собой устройство, устанавливающее однозначное соответствие между входным сигналом о происшедшем событии и адресом программной памяти, по которому размещена точка входа в подпрограмму обработки запроса прерывания от данного события. Контроллер прерывает выполнение ЦП текущей программы и инициирует его переход на выполнение подпрограммы обработки прерывания. Время, прошедшее с момента возникновения события до начала выполнения первой инструкции подпрограммы прерывания, называют временем реакции МК на событие. После окончания обработки ЦП автоматически возвращается к выполнению прерванной программы.

Другая функция контроллера прерываний – установка приоритетов событий. Понятие **приоритет** означает, что выполняемая подпрограмма прерывания может быть прервана другим событием только при условии, что оно имеет более высокий приоритет, чем текущее. В противном случае ЦП перейдет к обработке нового события после окончания обработки предыдущего. Контроллер прерываний, входящий в состав МК 8051, имеет пять входов событий: два от внешних устройств, два от таймеров и один от последовательного канала.

Обычно, когда говорят о каком-либо МК, то всегда упоминают **семейство**, к которому он принадлежит. К одному семейству относят изделия, имеющие одинаковое **ядро**, под которым понимают совокупность таких понятий, как система команд, циклограмма работы ЦП, организация памяти программ и памяти данных, система прерываний и базовый набор периферийных устройств. Фактически на рис. 1 представлено ядро, ставшее основой для создания сотен других модификаций семейства 8051.

Отличия между его различными представителями заключаются, в основном, в составе периферийных устройств и объёме памяти программ или данных. Поскольку диапазон задач, решаемых МК, чрезвычайно широк, их производители стараются выпустить столько модификаций, чтобы удовлетворить самые разнообразные запросы потребителей. Во многих семействах число модификаций приближается к сотне или даже превышает это значение.

Наиболее важная особенность семейства – программная совместимость на уровне двоичного кода всех входящих в него МК. Это позволяет разработчикам систем заменять одни МК семейства другими без потери наработок своего программного обеспечения. Естественно, чем большее число разновидностей входит в семейство, тем больше шансов выбрать оптимальный вариант, тем привлекательнее это семейство для разработчика. Вопрос правильного выбора семейства МК для новой разработки является стратегическим, так как проблема переноса программного обеспечения между изделиями разных семейств чрезвычайно сложна и даже использование языков высокого уровня не всегда позволяет решить её без больших потерь. К вопросу о критериях выбора мы вернемся в следующих статьях цикла.

Разработка программы – один из наиболее важных этапов в создании устройства на базе МК. Без неё он «мёртв», не реагирует на внешние воздействия и не выдаёт управляющих сигналов.

При включении питания МК немедленно начинает исполнять программу, находящуюся в подключенной к нему памяти программ (обычно это ПЗУ). Её выполнение начинается с некоторого фиксированного адреса, чаще всего нулевого. Адрес – это просто номер ячейки ПЗУ. Процесс осуществляется следующим образом: МК считывает число, хранящееся в памяти программ, и в зависимости от его значения, называемого машинным кодом, выполняет определённые действия над содержимым регистров АЛУ, памяти, портов и т.д. Например, прочитав из памяти программ число 32Н, МК «соображает», что нужно считать значение из входного порта номер 2 и поместить его в регистр-аккумулятор. Часто одного байта для описания действия МК не хватает, и тогда МК считывает из памяти дополнительные байты.

После выполнения действия МК считывает значение из следующей по порядку ячейки памяти и т.д. Совокупность байтов, описывающих одно выполняемое МК действие, называют машинной командой (инструкцией), а совокупность таких команд, которые «понимает» МК, – его системой команд или набором инструкций (Instruction Set). МК разных семейств имеют разные системы команд, то есть машинные коды у них имеют разные значения, хотя и выполняют похожие действия.

Итак, программа для МК представляет собой последовательность чисел, значения которых указывают ему, какие действия выполнять. Результатом разработки программы является компьютерный файл, содержащий эти машинные коды. С помощью программатора ПЗУ его заносят («зашивают») в память программ МК.

Каким же образом составляется эта последовательность машинных кодов – программа для МК? Неужели разработчику необходимо помнить значения машинных кодов и вручную задавать их последовательность? Первые программы для МК создавались именно так, и называлось это программированием в машинных кодах. Ясно, что такой способ разработки программ очень трудоёмок и неэффективен.

Первым шагом в облегчении процесса создания программ была компьютерная программа – так называемый транслятор с языка ассемблера. Идея состояла в том, чтобы выражать выполняемые МК действия на более понятном человеку языке и затем преобразовывать эти выражения в машинные коды. В приведённом выше примере машинной инструкции, которая считывает значение порта 2 и помещает его в аккумулятор, выполняемые действия можно условно обозначить как MOV A, P2.

Здесь слово MOV (от англ. move), называемое мнемоникой инструкции, обозначает пересылку значения, а A и P2, именуемые операндами, указывают, откуда взять значение и куда его поместить. Система подобных обозначений называется языком ассемблера. Программа, написанная на нём, обрабатывается транслятором, который преобразует конструкции языка ассемблера в машинные коды.

Программирование на ассемблере широко распространено по сей день. Трансляторы с языка ассемблера для всех популярных семейств микроконтроллеров бесплатны.

Несмотря на очевидные преимущества программирования на ассемблере перед программированием в машинных кодах, во многих случаях ассемблер недостаточно эффективен для реализации задач разработчика. Дело в том, что МК способен выполнять лишь простейшие действия вроде арифметических операций над целыми числами, пересылок, сравнений и т.п. Для более сложных задач, например, для операций над числами с плавающей запятой, разработчикам приходилось писать специальные подпрограммы, пользоваться которыми неудобно и громоздко. Следующим шагом в разработке программ для МК стало создание специальных компьютерных программ – трансляторов с языков программирования высокого уровня, или компиляторов. Наибольшее распространение получил язык программирования Си.

С появлением трансляторов разработка программ для МК резко упростилась. Если, например, нужно сложить в программе два числа, то теперь достаточно просто написать

a = b + c;

а транслятор преобразует это выражение в необходимую последовательность машинных команда зависимости от типов переменных a, b и c.

Использование языка высокого уровня позволяет разработчику отвлечься от системы команд конкретного МК и оперировать более простыми и понятными человеку категориями. От разработчика требуется только знание общей архитектуры МК, принципов работы необходимых для решения поставленной задачи встроенных периферийных устройств и навыки программирования на языке Си. Функциональное наполнение программы реализуется с помощью средств языка Си, который содержит большое число разнообразных подпрограмм (функций): арифметических, для работы с символьными строками и многих других.

Рассмотрим процесс создания программы для МК на языке Си. В процессе разработки потребуется персональный компьютер.

После уяснения поставленной задачи разработчик пишет исходный текст своей программы на языке Си с помощью любого текстового редактора. Затем он запускает программу-транслятор с языка Си, которая преобразует исходный текст в промежуточный объектный файл. Транслятор управляется с помощью набора ключей (их описание можно найти в его документации), которые указываются в его командной строке. Если при написании программы разработчик допустил синтаксические ошибки, транслятор выдаёт на экран их список с указанием для каждой номера строки в файле исходного текста. Разработчик должен исправить все ошибки. После успешной трансляции объектные файлы нужно обработать редактором связей (линкером), который и генерирует файл программы в машинных кодах.

При использовании языка высокого уровня возникает одна проблема. Преобразование конструкций языка в машинные коды возложено на компилятор, а выполнить это преобразование можно с различной степенью эффективности. Критериями эффективности являются размер машинного кода (чем он меньше, тем, естественно, лучше) и скорость машинного кода. Задача генерации компактного и быстрого кода весьма сложна, и от её решения зависит общее качество компилятора. Современные компиляторы Си используют многоуровневую оптимизацию, особенности архитектуры конкретного МК, позволяют создавать смешанные программы, в которых часть подпрограмм написана на ассемблере.

Описанный процесс выглядит довольно громоздким: разработчик должен вручную запускать разнообразные программы (текстовый редактор, компилятор Си, линкер), помнить управляющие ключи, искать ошибки в программе по номерам строк в файле. Последним на сегодняшний день шагом в облегчении труда разработчика программ для МК стало появление интегрированных сред разработки (Integrated Development Environment, IDE). Интегрированная среда разработки – это компьютерная программа, связывающая воедино все этапы разработки программы. Она совмещает в себе текстовый редактор для написания исходных текстов, трансляторы с ассемблера и Си, линкер, отладчик, справочную информацию по МК и другие средства, необходимые разработчику. Настройка трансляторов, линкера и других компонентов производится не методом указания ключей в командной строке, а в виде диалоговых окон, где нужно только расставить «галочки» в нужных местах. Преобразование исходных текстов программ в файл машинных кодов запускается одной клавишей.

Появление интегрированных сред разработки программ ещё больше повысило эффективность создания программ для МК, позволило разработчику сосредоточиться на сути решаемой задачи и отвлечься от конкретных деталей её реализации.

Интегрированные пакеты для разработки программ выпускают несколько фирм. Пакеты разных производителей схожи между собой по функциям, но различаются предоставляемыми сервисными возможностями, удобством работы и качеством генерируемого машинного кода.

Основные характеристики наиболее популярных пакетов средств разработки приведены в **таблице**.

## 4. Символьная отладка программ для МК

За редким исключением программы для МК из-за содержащихся в них ошибок не начинают работать с первого раза и требуют отладки. К вопросам отладки разработчики относятся по-разному. Некоторые из них считают, что достаточно внимательно проанализировать исходный текст, посмотреть с помощью осциллографа, что происходит на выводах МК, и можно исправить все ошибки. Такой способ применим, если разработчик имеет большой опыт, отлично знает применяемый МК и располагает транслятором, который всегда генерирует правильный код (обычно это ассемблер), и достаточным временем.

Другие используют в своей практике самодельные отладочные мониторы – наборы специальных подпрограмм, загружаемых в МК вместе с основной программой. Последняя вызывает в контрольных точках подпрограммы монитора, а те выдают информацию о состоянии ресурсов МК. Таким способом можно отладить практически любую программу, но у него есть недостатки, которые могут оказаться существенными. Во-первых, отладочному монитору необходимо предоставить для работы часть ресурсов МК: как минимум – часть адресного пространства кода и некоторое число ячеек стека, а как максимум – ещё часть ОЗУ и периферийные устройства МК, используемые монитором для отображения информации. Выделить ресурсы отладочному монитору бывает непросто, если основная программа сама активно загружает МК. Например, у МК PIC16C5x (Microchip) всего две ячейки стека, и использовать вызовы подпрограмм отладочного монитора затруднительно. Во-вторых, вызовы монитора отнимают время у основной программы и, следовательно, его нельзя вызывать из критичных ко времени частей программы. В-третьих, создание отладочного монитора, само по себе, требует времени.

Самый эффективный способ отладки программ для МК – применение специализированных профессиональных отладочных средств, к которым следует отнести отладчики-симуляторы и внутрисхемные эмуляторы.

Прежде чем рассказывать о возможностях, предоставляемых такими отладчиками, необходимо коснуться выбора компилятора, с помощью которого исходные тексты программ преобразуются в машинный код. В подавляющем большинстве случаев предпочтительно программирование на языке высокого уровня. Использование ассемблера необходимо, если к размеру и быстродействию генерируемого кода предъявляются очень жёсткие требования, В настоящее время таких случаев становится всё меньше, так как практически всегда можно взять более «быстрый» МК с большим объёмом памяти. Кроме того, современные пакеты кросс-средств позволяют легко писать смешанные программы, где часть модулей написана на Си, а наиболее критичные к быстродействию части – на ассемблере. Компиляторы Си позволяют также вставлять в исходные тексты ассемблерные инструкции.

Каковы же преимущества программирования на Си по сравнению с программированием на ассемблере? Вкратце они заключаются в следующем:

* отпадает необходимость заботиться об операциях с числами большой разрядности. Компилятор автоматически сгенерирует правильный код для операции a + b, если a и b будут 8-, 16-, 32-битными числами, числами с плавающей запятой и даже числами разных типов;
* в комплекте с компилятором поставляется обширная библиотека функций (подпрограмм), реализующих различные математические операции (тригонометрические функции, возведение в степень и т.п.), работу с символьными строками, форматированный ввод / вывод и т.д.;
* многие ошибки программиста диагностируются компилятором: он, например, не позволит передать функции неверное число параметров или параметры неверных типов, забыть поставить оператор возврата и т.п.;
* исходный текст, написанный на Си, гораздо легче читается, он компактнее, легче модифицируется;
* программы, написанные на Си, легче переносятся на MК других семейств.

Чтобы эффективно отлаживать программы, написанные на языке высокого уровня, разработчик должен иметь в своём распоряжении отладочные средства, предоставляющие адекватные возможности по отображению используемых в программе данных, а также по отслеживанию выполнения программы по её исходному тексту. Для обеспечения таких возможностей необходимы два условия:

* компилятор должен предоставлять достаточную информацию о структуре программы и используемых ею данных. Эту информацию называют символьной (отладочной);
* отладчик должен уметь интерпретировать эту информацию.

Все современные компиляторы и ассемблеры в том или ином виде генерируют символьную информацию, но в настоящее время ещё не разработано универсального формата, и каждый компилятор генерирует её в собственном формате. Это создаёт дополнительные трудности для отладчиков, которые должны уметь «понимать» несколько символьных форматов.

Теперь рассмотрим, как отладчик должен интерпретировать символьную информацию, и какие возможности должны в связи с этим предоставляться пользователю.

**Отслеживание выполнения программы по её исходному тексту**

В общем случае, одна строка исходного текста преобразуется компилятором в несколько машинных команд. Даже ассемблерная программа почти всегда содержит макросы, разворачивающиеся при трансляции в несколько инструкций процессора. Отлаживать такую программу по дизассемблеру её кода неудобно, поэтому компиляторы вставляют в отладочную информацию таблицу номеров строк. Она содержит информацию о соответствии номеров строк исходного текста и имён файлов исходного текста абсолютным адресам кода программы. Отладчик отображает на экране исходный текст программы и, следуя этой таблице, может выполнять программу «по строкам», выполняя за один шаг все машинные команды, сгенерированные компилятором для текущей строки.

Таблица номеров строк также позволяет производить контекстные действия с текстом программы, например, выполнять её «до курсора», то есть до указанного пользователем места в исходном тексте, ставить точки останова на указанные строки и т.п. Контекстные действий удобны тем, что разработчику не нужно знать адреса, соответствующие строкам исходного текста: отладчик сам определит их по таблице. Отладчик должен «знать» адреса подпрограмм, функций и меток кода и уметь находить исходный текст функции по её имени.

**Отображение данных, используемых в отлаживаемой программе**

Для полноценной отладки разработчику необходимо иметь возможность в любой момент просмотреть данные, которыми манипулирует программа. Отладчик должен «уметь» отображать любые используемые программой данные в наиболее подходящем виде.

Как правило, разработчики используют в программах именованные данные, то есть каждому объекту, который используется в программе, присваивается имя. Объекты могут быть разной сложности – от простых ячеек памяти до сложных конструкций языков высокого уровня типа структур, массивов и т.п.

**Данные в ассемблерных программах**

В ассемблерных программах используются в основном простые данные, то есть ячейки памяти. Применяются также массивы. Для правильного отображения простых данных отладчику нужно «знать»:

* имя объекта;
* адрес объекта в памяти;
* адресное пространство МК, в котором располагается объект. Многие МК имеют более одной области данных. Например, в МК семейства MCS-51 есть внутренняя память данных, внешняя память данных и битовое пространство;
* разрядность объекта, то есть занимаемое им число байт. 16-битные МК, такие как представители семейства MCS-96. «умеют» оперировать 8-, 16-, 32-битными данными. Здесь необходимо отметить один существенный момент. Для разработчика важно, какой логический размер имеет объект. Например, восьмиразрядные МК семейства PIC (Microchip) оперируют только байтами. Если же необходимо иметь в программе, например, 16-битный счётчик, то манипулировать каждым байтом приходится в отдельности. Но программисту при отладке хотелось бы видеть не каждый байт счётчика в отдельности, а оба байта сразу, в виде 16-битной переменной. Популярные кросс-ассемблеры такой возможности не предоставляют. Исключение – кросс-ассемблер PASM-PIC фирмы «Фитон», который позволяет объявлять в программе данные размером байт, слово, двойное слово, а также массивы таких объектов. При отладке программ, написанных с помощью PASM-PIC, все объекты отображаются в виде, соответствующем их логическому размеру и структуре;
* область видимости объекта. Если программа состоит из нескольких модулей, у программиста есть возможность локализовать область видимости имени в пределах одного модуля. Таким образом, в разных модулях могут существовать объекты с одинаковыми именами, но разными остальными атрибутами. Отладчик должен «разбираться», когда какой объект активен, и правильно отображать его. Заметим, однако, что практика использования одинаковых имён в разных модулях часто приводит к путанице и ошибкам. В случае если объект объявлен глобальным (PUBLIC) и виден во всех модулях, трудностей с интерпретацией не возникает.

Обладая вышеизложенной информацией, отладчик должен, получив от пользователя имя объекта, отобразить его значение в соответствии с типом. Наиболее «продвинутые» отладчики дополнительно могут отображать остальные атрибуты объекта.

**Данные в программах на языках высокого уровня**

Отображать объекты, применяемые в языках высокого уровня, значительно сложнее ввиду разнообразия структуры объектов, способов их размещения в памяти и областей видимости. Для примеров будем использовать язык Си, как наиболее популярный у разработчиков.

## 5. Структура объектов

Помимо простых переменных разной разрядности, в программах на Си используются также переменные с плавающей запятой, структуры (struct), объединения или союзы (union), указатели, одномерные и многомерные массивы. Последние могут состоять как из простых объектов, так и из сложных (структур, союзов, указателей).

Использование сложных объектов в программах, безусловно, удобно. Однако ввиду сложности их структуры крайне желательно иметь возможность её адекватного отображения на этапе отладки. В отладчиках фирмы «Фитон» сложные объекты могут отображаться как в сжатом (список значений элементов), так и в развёрнутом виде с указанием адреса, значения и типа каждого элемента массива и / или члена структуры. Реализация указателей в разных компиляторах различна. То, что МК обычно имеет несколько адресных пространств, создает дополнительные трудности, так как при работе с указателем должно быть известно, помимо адреса, и адресное пространство, куда указывает указатель. В некоторых реализациях идентификатор адресного пространства является составной частью значения указателя, в других компилятор заранее «знает» это и генерирует соответствующий код.

Кроме этого, компонент адреса в указателе может быть размером от 8 до 32 бит. При отображении значений указателей отладчик должен «знать» все детали их реализации в каждом компиляторе.

**Способы размещения объектов в памяти**

Помимо статических объектов, адреса которых не изменяются за время выполнения программы, в программе, написанной на языке высокого уровня, могут существовать так называемые автоматические объекты, память под которые временно отводится в стеке МК. Адреса таких объектов не абсолютны, а определяются динамически на этапе выполнения программы. Обычно они отсчитываются от текущего значения некоторой статической переменной, называемой указателем фрейма стека (Base Pointer или BP). Так как значение BP формируется программой динамически на этапе выполнения, значения автоматических объектов доступны только в пределах их области видимости, то есть при правильном значении BP. Отладчик при отображении значений автоматических объектов должен «знать» способ, которым определяются адреса, а также отслеживать правильность значения BP.

Возможно также временное размещение переменных в регистрах МК. В этом случае отладчик должен «знать», какие переменные в каких регистрах размещены и в течение какого времени. И, наконец, часто встречается ситуация, когда один и тот же объект за время своей жизни меняет способ размещения в памяти, причём не один раз. Это может происходить, например, когда функция получает один или несколько параметров в регистрах, а затем перемещает их в стек.

**Область видимости объекта**

Как и в ассемблерных, в программах на Си существуют глобальные объекты, доступные по имени из любого модуля, и объекты, локализованные в модуле (эти объекты объявляются как static). Однако автоматические и регистровые переменные создают отладчикам дополнительные трудности при отображении их значений. Дело в том, что, во-первых, время жизни автоматического объекта ограничено его областью видимости, а во-вторых, охватывающие области видимости могут иметь свои автоматические объекты с теми же именами. Проиллюстрируем это на примере функции, имеющей несколько вложенных областей видимости:

**void** f (**int** а)

{

**long** b;

**if** (a == **0**) a++;

**for** (b = a \* **2**; b < **100**; b++)

{

**long** с;

**long** a = b / **3**;

**if** (a == 0) a++;

**for** (c = a; с < **10**; c++)

{

**char** a = с == **9**? **1**: **0**;

fl(a);

}

f2 (a);

}

f3 (a);

}

Переменная с именем «а» существует всё время, пока выполняется функция f, но в зависимости от того, какая часть функции выполняется, имя «а» обозначает разные переменные. При трассировке функции f отладчик должен в зависимости от того, какая переменная активна, правильно показывать её значение.

Создавая программу, разработчик не заботится о деталях реализации понятий, которые он использовал в программе. Оперируя «само собой разумеющимися» категориями, он зачастую не подозревает, как сложно было реализовать их разработчикам компиляторов и отладчиков. Последним приходится решать задачи совмещения в одной оболочке одновременно простого и интуитивного интерфейса, богатства функциональных возможностей и детальной проработки всего, что связано с реализацией особенностей архитектуры и функционирования конкретного МК. Если отладчик не предоставляет разработчику средств отладки, адекватных сложности решаемой задачи, то разработчик неизбежно теряет в производительности. Кому из нас не приходилось тратить часы и дни в поисках досадной ошибки или опечатки в исходном тексте?!

В процессе разработки и создания микропроцессорной системы рано или поздно наступает момент, когда она, наконец, воплощена в «железе» и начинает подавать признаки жизни. Однако в большинстве случаев эти признаки оказываются непредсказуемыми, система начинает жить «своей» жизнью. Многие программисты, наверное, согласятся, что каждая новая программа содержит ошибки. Отчасти поэтому новый МК на первых порах ведёт себя как «чёрный» ящик.

Для того чтобы облегчить процесс отладки систем, разработан целый класс инструментальных средств. Их основное назначение – сделать процесс функционирования отлаживаемого МК «прозрачным», то есть легко контролируемым, произвольно управляемым и модифицируемым по воле разработчика. Хороший профессиональный инструментальный комплекс дополнительно может предоставить разработчику множество сервисных услуг, тем самым существенно облегчая его труд, избавляя от рутинных операций.

К числу основных инструментальных средств отладки относятся внутрисхемные эмуляторы, программные симуляторы, платы развития (оценочные платы), мониторы отладки и эмуляторы ПЗУ. Существуют и комбинированные устройства и наборы.

## 6. Внутрисхемные эмуляторы

Внутрисхемный эмулятор (ВСЭ) – программно аппаратное средство, способное замещать собой эмулируемый процессор в реальном устройстве. ВСЭ – наиболее мощное и универсальное отладочное средство.

Функционально ВСЭ делятся на стыкуемые с внешней вычислительной машиной (обычно это – IBM-совместимый ПК) и функционирующие автономно. Последние имеют собственные вычислительные ресурсы и средства ввода / вывода, поэтому при равных возможностях стоят значительно дороже первых, а при одинаковой цене существенно уступают им в функциональных и сервисных возможностях.

С отлаживаемой системой ВСЭ обычно соединяют кабелем со специальной эмуляционной головкой. Относительно недавно появились модели ВСЭ, у которых такая головка конструктивно совмещена с основным блоком и вставляется в отлаживаемую систему вместо МК. Если последний невозможно удалить (выводы впаяны в плату), использование ВСЭ допустимо при условии, что этот МК имеет отладочный режим, в котором все его выводы находятся в третьем (высокоимпедансном) состоянии. В этом случае для подключения ВСЭ используют специальный адаптер-клипсу. подсоединяемый непосредственно к выводам эмулируемого МК.

Как минимум, ВСЭ содержит отладчик, узел эмуляции МК, эмуляционную память и подсистему точек останова, В состав более совершенных ВСЭ могут дополнительно входить трассировщик, процессор точек останова, профилировщик (анализатор эффективности программного кода), таймер реального времени, программно-аппаратные средства, позволяющие читать и модифицировать ресурсы эмулируемого процессора «на лету», программно-аппаратные средства, обеспечивающие синхронное управление и необходимые для эмуляции в мультипроцессорных системах, интегрированную среду разработки.

**Отладчик** является своеобразным мостом между разработчиком и отладочным средством. Хороший отладчик обеспечивает загрузку отлаживаемых программ в память системы, вывод на монитор состояний и содержимого всех регистров и памяти (а при необходимости – и их модификаций), управление процессом эмуляции.

Более мощные отладчики (обычно их называют высокоуровневыми или High-Level Debuggers) позволяют также:

* вести символьную отладку (благодаря тому, что отладчик, используя специальную информацию, поставляемую компилятором, «знает» адреса всех символьных переменных, массивов и структур). При этом пользователь может оперировать более приемлемыми для человека символьными именами, не утруждая себя запоминанием их адресов,
* контролировать и анализировать не только дизассемблированный текст, но и исходный текст программы, написанной на языке высокого уровня, и даже с собственными комментариями.

Такой отладчик даёт возможность пользователю одновременно контролировать ход выполнения программы и видеть соответствие между исходным текстом, образом программы в машинных кодах и состоянием всех ресурсов эмулируемого микроконтроллера.

Следует отметить, что высокоуровневый отладчик обеспечивает выполнение всех своих функций только в том случае, если используется кросс-компилятор, поставляющий полную и правильную отладочную информацию (не все компиляторы, особенно их пиратские версии, способны на это), и при этом формат её представления «знаком» отладчику.

**Эмуляционную память** используют в процессе отладки вместо ПЗУ разрабатываемой системы. Более того, она позволяет отлаживать программу в отсутствие реальной системы или её макета. Если необходимо внести изменения в отлаживаемую программу, достаточно загрузить новую или модифицированную программу в память эмулятора, а не заниматься перепрограммированием ПЗУ.

Существуют ВСЭ, которые позволяют пользователю «подставлять» эмуляционную память вместо ПЗУ не только целиком, но и поблочно (в некоторых моделях минимальный размер блока – 1 байт), в порядке, определённом пользователем. Для этого ему достаточно задать распределение памяти данных и памяти программ, в соответствии с которым процессор будет получать доступ и к содержимому ПЗУ в отлаживаемой системе, и к содержимому эмуляционной памяти ВСЭ, Такую память обычно называют памятью с возможностью мэппинга.

**Трассировщик** представляет собой логический анализатор, работающий синхронно с процессором и фиксирующий поток выполняемых инструкций и состояния выбранных внешних сигналов. Существуют ВСЭ, позволяющие трассировать не только внешние сигналы, но и состояния внутренних ресурсов МК, например, регистров. В таких устройствах используют специальные версии МК (эмуляционные кристаллы).

**Процессор точек останова** даёт возможность останавливать исполнение программы или производить иные действия (например, запускать или останавливать трассировщик) при выполнении заданных пользователем условий. В отличие от механизма обычных точек останова, процессор позволяет на аппаратном уровне формировать и отслеживать условия практически любой сложности, при этом эмулируемый процесс не выводится из масштаба реального времени. В некоторых моделях ВСЭ процессор точек останова может дополнительно использоваться для динамического управления трассировщиком.

**Профилировщик** (анализатор эффективности программного кода) позволяет по результатам прогона отлаживаемой программы получить информацию о числе обращений к различным участкам программы и времени, затраченном на их выполнение. Анализ статистической информации, поставляемой профилировщиком, даёт возможность выявлять «мёртвые» или перенапряженные участки программ и в результате оптимизировать структуру отлаживаемой программы.

**Интегрированная среда разработки** – это совокупность программных средств, поддерживающая все этапы разработки программного обеспечения от написания исходного текста программы до её компиляции и отладки и обеспечивающая простое и быстрое взаимодействие с программным отладчиком-симулятором и программатором.

Наличие в программной оболочке ВСЭ встроенного редактора, менеджера проектов и системы управления существенно облегчает работу разработчика, избавляя его от множества рутинных действий. Для него стирается грань между написанием программы, её редактированием и отладкой. Переход от редактирования исходного текста к отладке и обратно осуществляется «прозрачно» и синхронно с активизацией соответствующих окон. Менеджер проектов по мере необходимости автоматически запускает компиляцию и активизирует соответствующее окно программного интерфейса. Столь же просто можно перейти к отладке проекта с помощью имеющегося отладчика-симулятора или приступить к «прошивке» ПЗУ отлаженной программой,

Некоторые ВСЭ предоставляют пользователям и другие дополнительные возможности. Среди них особенно следует отметить одну, хотя и довольно специфическую, но в ряде случаев имеющую принципиальное значение, возможность построения многоэмуляторных комплексов, необходимых для отладки мультипроцессорных систем. Отличительная особенность такого комплекса – синхронное управление (с одного компьютера) несколькими эмуляторами.

В общем же случае возможности ВСЭ по контролю и управлению функционированием отлаживаемых устройств могут быть ограничены (например, некорректной обработкой прерываний в пошаговом режиме, запретом на использование последовательного порта и т.п.). Также необходимо помнить, что у каждой модели ВСЭ – свой перечень поддерживаемых МК и компиляторов.

Однако для большинства популярных МК разработаны ВСЭ, не имеющие ограничений по использованию ресурсов отлаживаемых кристаллов. Возможности такого ВСЭ проиллюстрируем на примере модели PICE-51 фирмы «Фитон».

PICE-51 – устройство, созданное с применением программируемых логических ИС (ПЛИС). Это позволило резко уменьшить размеры ВСЭ, минимизировать отклонения его электрических и частотных характеристик от характеристик эмулируемого МК и тем самым добиться максимальной точности эмуляции на частотах до 33 МГц при напряжениях питания от 3,3 до 5 В. Перезагружаемая аппаратная структура PICE-51 обеспечивает эмуляцию практически всех МК семейства MCS-51. Программная поддержка работает в среде Windows.

PICE-51 состоит из основной платы, сменного адаптера под конкретную группу МК и сменной эмуляционной головки также под конкретный тип корпуса. На основной плате собраны трассировщик и процессор точек останова, на плате сменного адаптера – эмулирующий процессор под конкретный тип МК. Эмуляционные головки обеспечивают установку прибора в розетки DIP и PLCC на плате пользователя. Питание осуществляется от блока с выходным напряжением +5 В (0,5 А) или от отлаживаемого устройства. Связь с компьютером – по гальванически развязанному каналу RS-232C при скорости 115 кБод.

Остальные характеристики и возможности PICE-51 следующие:

* точная эмуляция – отсутствие каких-либо ограничений на использование программой пользователя ресурсов МК;
* до 256 Кбайт эмулируемой памяти программ и данных. Поддержка банкированной модели памяти. Распределение памяти между ВСЭ и устройством пользователя с точностью до 1 байта;
* до 512К аппаратных точек останова по доступу к памяти программ и данных;
* аппаратная поддержка для отладки программ на языках высокого уровня;
* трассировка восьми произвольных внешних сигналов;
* четыре выхода синхронизации аппаратуры пользователя;
* трассировщик реального времени с буфером объёмом от 16 до 64К фреймов (массивов) по 64 бита с доступом «на лету». Трассировка адреса, данных, сигналов управления, таймера реального времени и восьми внешних сигналов пользователя;
* программируемый фильтр трассировки;
* аппаратный процессор точек останова с возможностью задания сложного условия останова эмуляции по комбинации сигналом адреса, данных, управления, восьми внешних сигналов, таймера реального времени, счётчиков событий и таймера задержки;
* четыре комплексных точки останова, которые могут быть использованы независимо или в комбинациях по условиям AND/OR/IF-THEN;
* 48-разрядный таймер реального времени;
* «прозрачная» эмуляция – доступ «на лету» к эмулируемой памяти, точкам останова, процессору точек останова, буферу трассировки, таймеру реального времени;
* управляемый генератор тактовой частоты для эмулируемого МК. Возможность плавного изменения её от 500 кГц до 40 МГц;
* встроенная система самодиагностики аппаратуры ВСЭ, Поддерживается разработка программ на уровне ведения проектов для макроассемблера MCA-51 («Фитон»/ «Микрокосм»), а также для пакетов кросс-средств фирм Keil Software и IAR Systems;
* поддержка полнофункциональной символьной отладки программ, созданных с помощью следующих компиляторов: ассемблера ASM51 фирмы Intel, компилятора PL/M фирмы Intel, ассемблеров и компиляторов Си фирм Avocet Systems, Hi-Tech, Tasking Software;
* автоматическое сохранение и загрузка файлов конфигурации аппаратуры, интерфейса и опций отладки. Обеспечена совместимость файлов конфигурации с симулятором PDS-51 и переносимость проектов между PICE-51 и симулятором PDS-51;
* возможность настройки цветов, шрифтов и других параметров для всех окон одновременно и для каждого окна в отдельности.

Столь широкий набор функциональных возможностей делает ВСЭ наиболее мощным и универсальным средством отладки.

**Симуляторы**

Симулятор – программное средство, способное имитировать работу МК и его памяти. Как правило, оно состоит из отладчика, модели центрального процессора и памяти. Более совершенные устройства содержат в своём составе модели встроенных периферийных устройств (таймеров, портов, АЦП и систем прерываний).

Симулятор должен «уметь» загружать файлы программ во всех популярных форматах, максимально полно отображать информацию о состоянии ресурсов симулируемого МК, а также предоставлять возможности по симуляции выполнения загруженной программы в различных режимах. В процессе отладки модель выполняет программу, и на экране монитора компьютера отображается текущее состояние модели.

Загрузив программу в симулятор, пользователь может запускать её в пошаговом или непрерывном режиме, задавать условные или безусловные точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров симулируемого МК. Симулятор позволяет быстро проверить логику выполнения программы, правильность выполнения арифметических операций.

В зависимости от класса используемого отладчика некоторые модели симуляторов поддерживают высокоуровневую символьную отладку программ.

Симулятор может содержать и ряд дополнительных программных средств, например интерфейс внешней среды. Наличие такого интерфейса позволяет создавать и гибко использовать модель внешней среды МК, функционирующую и воздействующую на отлаживаемую программу по заданному алгоритму.

В реальной системе МК обычно «занимается» считыванием информации с подключенных к нему внешних устройств (датчиков), обработкой её и выдачей управляющих сигналов на исполнительные устройства. Для того чтобы в простом симуляторе смоделировать работу датчика, нужно вручную изменять текущее состояние модели периферийного устройства, к которому в реальной системе подключен датчик. Если, например, при приёме байта через последовательный порт устанавливается некоторый флажок, а сам байт попадает в определённый регистр, то оба эти действия нужно производить в симуляторе вручную. В некоторых моделях эта проблема решена: симуляторы имеют встроенные средства для создания моделей подключенных к МК внешних устройств, включая средства графического отображения информации.

Очевидная особенность программных симуляторов в том, что загруженные в них программы выполняются в масштабе времени, отличном от реального. Однако низкая цена, возможность отладки даже при отсутствии макета отлаживаемого устройства делают программные симуляторы весьма привлекательным средством отладки. Необходимо также отметить, что существует целый класс ошибок, которые можно обнаружить только с помощью симулятора.

## 7. Отладочные мониторы

Отладочный монитор – специальная программа, загружаемая в память отлаживаемой системы. Она вынуждает МК выполнять, кроме прикладной задачи, ещё и отладочные функции:

* загрузку прикладных кодов пользователя в свободную от монитора память;
* установку точек останова;
* запуск и останов загруженной программы в реальном времени;
* проход программы пользователя по шагам;
* просмотр, редактирование содержимого памяти и управляющих регистров.

Программа монитора работает «в связке» с компьютером или пассивным терминалом, на которых и происходят визуализация и управление процессом отладки. Достоинство этого подхода – очень малые затраты при сохранении возможности вести отладку в реальном масштабе времени, главный недостаток – отвлечение ресурсов МК на отладочные и связные процедуры (монитор занимает некоторый объём памяти, прерывания, последовательный канал). Недавно появились программы, которые практически не занимают аппаратных ресурсов МК (о них будет рассказано в разделе «Эмуляторы ПЗУ»).

**Платы развития**

Платы развития, или, как принято их называть в зарубежной литературе, оценочные платы (Evaluation Boards), – своеобразные конструкторы для макетирования прикладных систем. В последнее время многие фирмы-производители, выпуская новые модели МК, предлагают и соответствующие платы развития. Обычно это печатная плата с установленными на ней МК и всеми необходимыми для его нормальной работы элементами, а также системами связи с компьютером. Как правило, на плате предусмотрено свободное место для монтажа разрабатываемого устройства пользователя. Иногда имеется и готовая «разводка» для установки рекомендуемых фирмой дополнительных устройств (ПЗУ ОЗУ, ЖК-дисплея, клавиатуры, АЦП и др.). Платы, доработанные пользователем, выгодно использовать в качестве одноплатных, контроллеров, встраиваемых в мелкосерийную продукцию (5…20 шт.).

Для удобства пользователя платы развития комплектуются ещё и простейшим отладочным средством на базе монитора отладки. Здесь проявились два разных подхода: один используется для МК, имеющих внешнюю шину, а второй – для МК, не имеющих её.

В первом случае отладочный монитор поставляется в виде микросхемы ПЗУ, которую устанавливают в специальную розетку на плате развития. Плата также имеет ОЗУ для программ пользователя и канал связи с компьютером или терминалом, Примером может служить плата развития, разработанная фирмой Intel для МК семейства МCS-51.

Во втором случае плата развития содержит встроенные системы программирования внутреннего ПЗУ МК, которые управляются компьютером. Программу монитора заносят в ПЗУ МК вместе с прикладной, подготовленной соответствующим образом (в нужные места вставляют вызовы отладочных подпрограмм монитора). Затем осуществляется пробный прогон. Чтобы внести исправления в отлаживаемую программу, её стирают из ПЗУ и записывают в него откорректированную. Готовую прикладную программу получают из отлаженной путём удаления монитора и всех вызовов его функций. На такой алгоритм отладки рассчитаны платы развития для МК семейств PICmicro (Microchip), 80С750 (Philips), 89С2051 (Atmel).

Платы развития иногда комплектуются программами отладки, которые запускают на внешнем компьютере «в связке» с монитором. Эти программы в последнее время заметно усложнились и зачастую имеют высокопрофессиональный набор отладочных функций (например, отладчик-симулятор) или различные элементы, присущие в чистом виде только интегрированным средам разработки. В состав комплектов могут входить и программы прикладного характера, наиболее часто встречающиеся на практике.

Отладочные возможности комплекта «плата развития плюс монитор» не столь универсальны, как возможности ВСЭ, к тому же некоторая часть ресурсов МК в процессе отладки отбирается для работы монитора. Тем не менее, наличие законченного набора готовых программно-аппаратных средств, позволяющих без потери времени приступить к монтажу и отладке прикладной системы, во многих случаях является решающим фактором. Особенно если учесть, что стоит такой комплект в несколько раз меньше, чем более универсальный эмулятор.

**Эмуляторы ПЗУ**

Эмулятор ПЗУ – программно-аппаратное средство, позволяющее заменять ПЗУ отлаживаемого устройства на ОЗУ, в которое можно загрузить программу с компьютера через один из стандартных каналов связи. Оно позволяет пользователю избежать многократных циклов перепрограммирования ПЗУ. Эмулятор ПЗУ используют только для отладки программ МК, которые способны обращаться к внешней памяти программ. По сложности и стоимости это устройство сравнимо с платами развития. У него одно большое достоинство – универсальность. Эмулятор ПЗУ может работать с любыми МК.

Первые эмуляторы ПЗУ позволяли только загружать программу, запускать её и останавливать, используй общий сброс. Затем появились усложнённые модели с аппаратной выработкой сигналов трассировки на осциллограф по достижении определённого адреса. Эмулируемая память в таких изделиях была доступна для просмотра и модификации, однако контроль за внутренними управляющими регистрами МК был до недавнего времени невозможен.

В последнее время появились так называемые интеллектуальные эмуляторы ПЗУ. Они позволяют «заглядывать» внутрь МК на плате пользователя и по управлению отладкой похожи на ВСЭ. Фирма Cactus даже представляет свой фактически интеллектуальный эмулятор ПЗУ, как ВСЭ ряда МК, настолько невозможно отличить работу с тем и другим. В действительности же процессор в этом случае не замещается, а используется тот, что стоит на плате пользователя.

Интеллектуальный эмулятор ПЗУ – это гибрид обычного эмулятора ПЗУ, отладочного монитора и системы быстрого переключения шины с одного на другой. Этим создаётся эффект, как если бы монитор отладки был установлен на плате пользователя, и при этом он практически не занимает у МК аппаратных ресурсов, кроме небольшой (примерно 4 Кбайт) зоны программных шагов. Подобный эмулятор разработан, например, фирмой «Фитон» для всех существующих и будущих МК, которые имеют ядро 8051, но дополнительно насыщены различными устройствами ввода / вывода. Изделие поддерживает множество самых разных МК фирм Philips, Siemens, OKI.

## 8. Интегрированные среды разработки

Строго говоря, интегрированные среды разработки не относятся к числу средств отладки, тем не менее, обойти вниманием данный класс программных средств, существенно облегчающих и ускоряющих процесс разработки и отладки микропроцессорных систем, было бы неправильно.

При традиционном подходе начальный этап написания программы строится следующим образом. Исходный текст набирают с помощью какого-либо текстового редактора. По завершении набора работа с текстовым редактором прекращается и запускается кросс-компилятор. Как правило, новая программа содержит синтаксические ошибки, и компилятор сообщает о них на консоль оператора. Затем вновь запускается текстовый редактор, и оператор ищет и устраняет выявленные ошибки. При этом сообщения об их характере, выведенные компилятором, уже не видны, так как экран занят текстовым редактором.

Этот цикл может повторяться не один раз. И если программа относительно сложна, собирается из различных частей, подвергается редактированию или модернизации, то даже этот начальный этап может потребовать очень много сил и времени программиста.

Избежать большого объёма рутинной работы и там самым существенно повысить производительность труда программиста позволяют появившиеся и быстро завоёвывающие популярность так называемые интегрированные среды (оболочки) разработки (Integrated Development Environment – IDE).

Как правило, хорошая интегрированная среда объединяет имеющиеся средства отладки (внутрисхемный эмулятор, программный симулятор, программатор) и обеспечивает работу программиста с текстами программ в стиле «турбо».

Интегрированная среда позволяет:

* использовать встроенный многофайловый текстовый редактор, специально ориентированный на работу с исходными текстами программ;
* наблюдать одновременно (в многооконном режиме) диагностику выявленных при компиляции ошибок и исходный текст программы, доступный редактированию;
* вести параллельную работу над несколькими проектами. Менеджер проектов позволяет использовать любой проект в качестве шаблона для вновь создаваемого. Опции используемых компиляторов и список исходных файлов проекта устанавливаются в диалоговых меню и сохраняются в рамках проекта, устраняя необходимость работы с неудобными batch-файлами:
* подвергать перекомпиляции только редактировавшиеся модули;
* загружать отлаживаемую программу в имеющиеся средства отладки и работать с ними без выхода из оболочки;
* подключать к оболочке практически любые программные средства.

В последнее время функции интегрированных сред разработки становятся принадлежностью программных интерфейсов наиболее «продвинутых» эмуляторов и отладчиков-симуляторов. Такие функциональные возможности в сочетании с дружественным интерфейсом существенно ускоряют работу программиста.

Таким образом, выбирая инструментальные средства отладки, целесообразно принимать в расчёт следующий комплекс показателей: перечень поддерживаемых МК, ограничения на ресурсы эмулируемых / симулируемых МК, возможность символьной отладки, перечень поддерживаемых компиляторов и, наконец, сервисные возможности.

## Заключение

В последнее время функции интегрированных сред разработки становятся принадлежностью программных интерфейсов наиболее «продвинутых» эмуляторов и отладчиков-симуляторов. Такие функциональные возможности в сочетании с дружественным интерфейсом существенно ускоряют работу программиста.

Таким образом, выбирая инструментальные средства отладки, целесообразно принимать в расчёт следующий комплекс показателей: перечень поддерживаемых МК, ограничения на ресурсы эмулируемых / симулируемых МК, возможность символьной отладки, перечень поддерживаемых компиляторов и, наконец, сервисные возможности.

**Литература**

1. Бродин В.Б., Калинин А.В. Системы на микроконтроллерах и БИС программируемой логики. – М.: ЭКОМ, 2002.
2. Микушин А. Занимательно о микроконтроллерах. – М.: БХВ-Петербург, 2006. – ISBN 5–94157–571–8
3. Фрунзе А.В. Микроконтроллеры? Это же просто!. – М.: ООО «ИД СКИМЕН», 2002. – Т. 1.
4. Фрунзе А.В. Микроконтроллеры? Это же просто!. – М.: ООО «ИД СКИМЕН», 2002. – Т. 2.
5. Фрунзе А.В. Микроконтроллеры? Это же просто!. – М.: ООО «ИД СКИМЕН», 2003. – Т. 3.
6. Голубцов. М.С. Микроконтроллеры AVR: от простого к сложному – М.: СОЛОН-Пресс, 2003. 288 с. – (Серия «Библиотека инженера»).
7. Журнал «Радио» №2 за 2000 г.