САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Е. Г. Павловский, В. А. Жвариков, А.А. Кузьмин

ОСНОВЫ ОРГАНИЗАЦИИ МИКРОКОНТРОЛЛЕРОВ

Учебное пособие и методические указания
к лабораторному практикуму

Санкт-Петербург
2014

УДК 004.3

Е. Г. Павловский, В. А. Жвариков, Кузьмин А.А. **Основы организации микроконтроллеров:** Учебное пособие и методические указания к лабораторному практикуму. — СПб.; Санкт-Петербургский государственный политехнический университет, 2014. — 205 с.

В пособии излагаются основные принципы организации и функционирования микроконтроллеров семейства MCS-51. Приводится описание основных подсистем микроконтроллеров. Рассматриваются используемые способы адресации и особенности системы команд, а также вопросы взаимодействия микроконтроллера с памятью и внешними устройствами ввода/вывода. Теоретический материал, изложенный в пособии, поддержан лабораторным практикумом и иллюстрируется примерами решения типовых задач, реализуемых микроконтроллерными системами. Пособие предназначено для студентов, обучающихся по направлениям 220400 «Управление в технических системах», 230100 «Информатика и вычислительная техника».

Печатается по решению кафедры компьютерных систем и программных технологий Санкт-Петербургского государственного политехнического университета.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Микропроцессорные средства появились как естественный этап развития технологии производства СБИС. Использование этих средств позволило существенно расширить сферы применения вычислительной техники. В частности они оказали влияние на развитие таких важных отраслей промышленности, как приборостроительная, радиотехническая, электронная, машиностроительная и др. Уже первые внедрения микропроцессоров (МП) определили два основных направления их применения: использование МП в качестве центральных процессоров вычислительных машин (ВМ) и реализацию на базе МП встраиваемых систем управления различными объектами. Число типов МП для ВМ ограничено. Напротив, из-за бесконечно большого разнообразия объектов управления число встраиваемых систем и МП, на основе которых они строятся, чрезвычайно велико.

Основой для реализации алгоритмов управления в технических системах является универсальность средств вычислительной техники, поскольку они обеспечивают возможность решения любой алгоритмически разрешимой задачи.

Чтобы лучше понять принципы построения, организацию и основные характеристики средств вычислительной техники (ВТ) для встраиваемых систем управления объектами рассмотрим структуру типовой системы управления (контроллера). В ее состав (рис. 1.1) входят:

- объект управления (ОУ), содержащий собственно объект или процесс, исполнительные устройства (ИУ) и систему датчиков контролируемых параметров объекта;
- устройство сопряжения с объектом, обеспечивающее согласование сигналов (как управляющих, так и информационных) микропроцессорной системы и объекта управления;
- пульт управления, предоставляющий оператору возможность контролировать параметры процесса управления и вносить при необходимости в него коррективы;
- цифровой регулятор, обеспечивающий управление объектом в соответствии с решаемой задачей и информацией поступающей от датчиков и пульта управления системой.

Цифровой регулятор получает необходимую информацию о текущем состоянии объекта от измерительных преобразователей – датчиков. Управляющие воздействия поступают на объект через исполнительные устройства. Для связи цифрового регулятора с датчиками и исполнительными устройствами используются специальные электронные схемы – *устройства сопряжения с объектом*.

Конкретная реализация цифрового регулятора определяется типом используемой системы обработки и зависит от сложности объекта

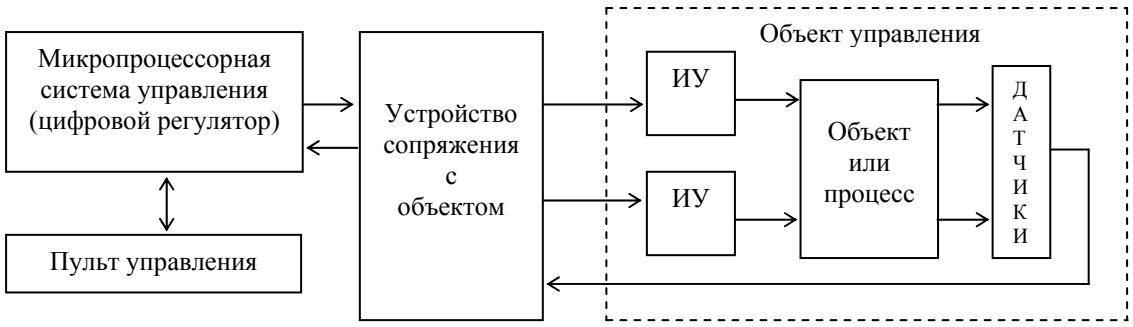


Рис. 1.1. Обобщенная структура типовой системы управления (контроллера)

управления. Сложные системы требуют использования сложных регуляторов, которые могут быть построены как на основе ВМ общего назначения, например, класса персональных компьютеров (ПК) в промышленном исполнении, так и на основе вычислительных комплексов (ВК) или вычислительных систем (ВС). Однако во многих случаях функциональные возможности ВМ (ВК, ВС) для решения задач управления оказываются избыточными и большинство цифровых регуляторов чаще всего реализуют в виде микропроцессорных систем.

Существенной особенностью работы цифровых регуляторов в системах управления, как на базе ВМ, так и при микропроцессорной реализации, является требование выполнения всех операции, связанных с управляемым процессом, *в реальном масштабе времени.* В большинстве случаев в системах существуют несколько параллельных процессов. Требования к времени выполнения различных процессов различны, и поскольку для их выполнения используются общие ресурсы, выполнение процессов упорядочивается с использованием абсолютных приоритетов для каждого из процессов.

В основу организации управления заложена модель реально протекающего дискретного процесса [1]. Процесс в модели представляется в дискретном времени. Реальное непрерывное время делится на кванты τ. Длительность кванта задается в зависимости от инерционности объекта с помощью таймера (на рис. 1.1 не показан). Кванты нумеруется: $\tau = 0, 1, 2, \ldots$ Для программного управления объектом при работе с моделью минимально необходимыми являются три компонента:

- текущее состояние модели q (i) на основе показаний датчиков;
- расчеты, связанные с определением следующего состояния $q(i+1)$;
- формирование управляющих воздействий на исполнительные устройства, обеспечивающих переход в состояние $q(i+1)$.

Работа системы тактируется. По сигналу от таймера, отмечающему начало очередного кванта, происходит прерывание программы, выполняемой в фоновом режиме, и осуществляется переход к программам идентификации состояния объекта $q(i)$, расчета следующего состояния $q(i+1)$ и формирования управляющих воздействий на объект. Выполнение

этих программ должно завершиться раньше, чем закончится квант. В оставшуюся часть кванта времени выполняются программы, имеющие более низкий приоритет. Процессы периодически повторяются.

Для управления сложными, распределенными в пространстве объектами организуют аппаратно-программные управляющие комплексы с иерархической (двух или трехуровневой) структурой. На верхнем уровне находятся ВМ в промышленном исполнении. На нижних уровнях, где непосредственно осуществляется сбор информации от датчиков, ее обработка и формирование управляющих воздействий, часто используют встраиваемые системы на базе микропроцессоров. Обмен информацией осуществляется с помощью цифровых и аналоговых сигналов. Для передачи сигналов требуются специальные каналы связи (отдельные жилы кабеля, оптоэлектронные развязки и пр.). При такой организации информационного взаимодействия с объектами, особенно, если они большой протяженности, возникает ряд проблем, связанных с обеспечением помехоустойчивости кабельных каналов связи, относительно высокой стоимостью кабельных соединений и их громоздкостью. Для преодоления этих трудностей часто используют встраиваемые системы на базе микроконтроллеров. С их помощью осуществляется первичная обработка сигналов в непосредственной близости от объекта, и реализуются алгоритмы локального управления без выхода на ВМ. При этом углубляется иерархия организации системы управления и перераспределяются функции между уровнями. Обмен информацией между микроконтроллерами и ВМ более высокого уровня осуществляется с использованием передачи сообщений, а не отдельных сигналов и, как правило, выполняется с использованием стандартизированных каналов связи, благодаря чему существенно упрощается система кабельных соединений.

Принципы организации микроконтроллерных систем

Под *встраиваемой системой управления (контроллером)* будем понимать систему управления, пространственно приближенную к датчикам и исполнительным устройствам и конструктивно интегрированную в оборудование – механизм, технологическую установку, робот и пр. Такая система управления может быть реализована на основе одноплатной ВМ с необходимым набором интерфейсов, обеспечивающих функции связи с объектом и взаимодействие с оператором. Важным компонентом такой системы является интерфейс с системой управления более высокого уровня. Благодаря его наличию возможно решение задач комплексной автоматизации с использованием заданного числа единиц технологического оборудования, объединенного в единую распределенную систему.

Микропроцессорные реализации цифровых регуляторов имеют ряд специфических характеристик. В первую очередь это необходимость разработки для каждой решаемой задачи собственного программного обеспечения (ПО). Действительно, из-за многообразия задач управления и ограниченности ресурсов микропроцессорных систем при решении конкретных задач использование стандартного ПО, в отличие от ВМ общего пользования, практически исключается. Кроме того, «уникальность» программного обеспечения любой задачи, не изменяемого после его отладки в течение всего времени эксплуатации системы, позволяет использовать для хранения ПО энергонезависимую память – ПЗУ (постоянное запоминающее устройство). К отличительным характеристикам микропроцессорных реализаций цифровых регуляторов можно отнести:

- относительно невысокие требования к вычислительным ресурсам (в большинстве случаев требуется реализация сравнительно простых алгоритмов управления**);**
- многообразие источников и приемников обрабатываемой информации (входные и выходные сигналы для цифровых регуляторов могут быть как дискретными, так и аналоговыми и для их обработки в составе микропроцессорной системы часто предусматривают устройства, преобразующие аналоговые сигналы в цифровую форму и обратно);
- требуется работа микропроцессорных систем управления в «реальном времени». При этом необходимо обеспечивать совершенно конкретную, рассчитываемую на этапе разработки, скорость реакции системы управления на внешние события. Задержки, превышающие расчетные, в таких системах недопустимы, так как могут приводить к потере управляемости и, как следствие, к повреждению или разрушению объекта управления. «Реальное время» потребовало включения в состав микропроцессорных систем управления таких устройств, как таймеры, контроллеры прерываний и др.;
- повышенные требования к надежности и стойкости к воздействиям окружающей среды. Встраиваемые системы управления должны работать без сбоя и ошибок в необслуживаемом режиме и, как правило, в сложных условиях эксплуатации.
- в отличие от универсальных компьютеров к управляющим контроллерам, как правило, не предъявляются высокие требования к производительности и программной совместимости.

По мере совершенствования цифровых технологий у разработчиков СБИС появилась возможность разместить основные подсистемы цифровых регуляторов (процессор, память, периферийные устройства) на одном кристалле. Первая реализация подобной системы, получившая название однокристальная ВМ или *микроконтроллер* (МК), была представлена фирмой Intel в 1978 в виде устройства 8048.

Микроконтроллером *называется программируемое однокристальное вычислительное устройство с встроенным набором средств для ввода и вывода, применяемое для решения задач управления и первичной обработки данных в технических системах.*

Микроконтроллер - это разновидность микропроцессорной системы, ориентированной на реализацию алгоритмов управления техническими устройствами и технологическим оборудованием.

Встраиваемые средства на базе МК относительно просты, имеют низкую стоимость, малые размеры. На них обычно возлагают следующие функции: сбор, первичную обработку и преобразование сигналов датчиков объекта в сообщения стандартного формата, передачу их в ВМ; прием, накопление и трансляцию команд управления, передаваемых от ВМ; самодиагностику и первичную диагностику объекта.

Вычислительная мощность и состав внутренней периферии современных МК позволяет строить централизованные системы управления даже для таких сложных объектов, как бортовые системы воздушных и морских кораблей, транспортные системы и многие другие. В подобных системах регулируемые величины нередко являются сильно связанными между собой, в связи с чем требуется их совместное регулирование. Централизованные системы, выполненные на базе одного МК, упрощают реализацию сложных алгоритмов управления для таких объектов, но имеют недостаток: в них приходится передавать информацию о состоянии объекта и управляющие воздействия на относительно большие расстояния, при этом часто такая передача должна осуществляться в условиях сильных помех. В ряде случаев стоимость помехоустойчивых средств сопряжения с объектом для подобных систем оказывается соизмеримой со стоимостью самого регулятора. Для устранения отмеченного недостатка при реализации систем управления сложными объектами широко используют *рассредоточение средств регулирования* – каждая подсистема выполняется на отдельном МК и располагается вблизи регулируемого объекта. Обеспечение взаимодействия распределенных подсистем регулирования достигается с помощью ВМ верхнего уровня управления и специальных коммуникационных каналов связи. В англоязычной литературе архитектура многоуровневого управления, обеспечивающая *интеграцию отдельных подсистем,* обозначают термином CAN (Controller Area Network – локальная управляющая сеть).

Обмен данными в управляющей системе, содержащей несколько совместно работающих МК, в большинстве случаев осуществляется по последовательному каналу. Для разделения информационных посылок с адресами абонентов и данными используют специальные приемы.

В зависимости от сложности задач встраиваемые системы управления (англ. - embedded application) бывают одноуровневыми и много-уровневыми. В последнем случае на нижнем уровне управления решаются

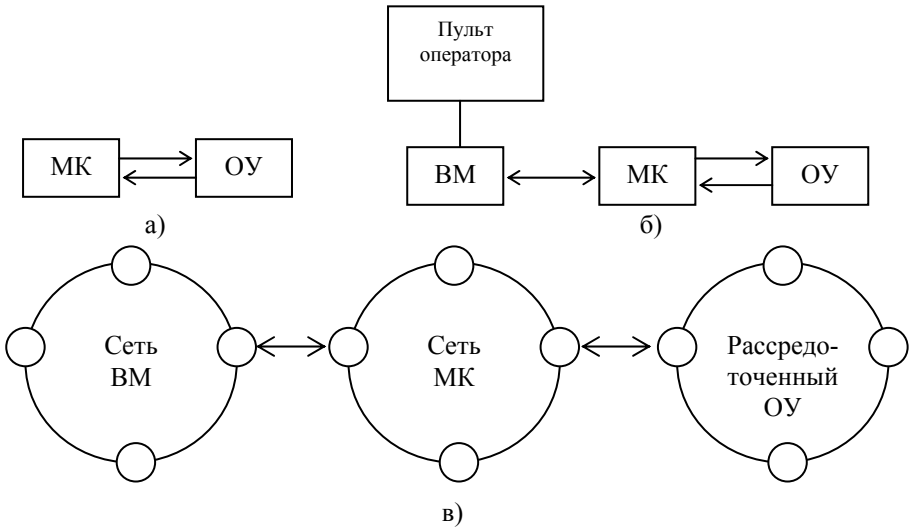


Рис. 1.2. Типы микроконтроллерных систем:
а) автономная, б) локальная, в) сетевой конфигурации

задачи локального управления отдельными подсистемами, а на более высоких – общие задачи управления системой в целом. Одноуровневые системы управления, как правило, реализуют на базе микроконтроллеров, а при построении двух- и более уровневых систем на верхних уровнях управления предполагается использование ВМ в промышленном исполнении, обеспечивающих интерфейс с человеком-оператором. Микроконтроллерные системы соответствующего типа - автономные (без ВМ), локальные (автономные с ВМ) и сетевой конфигурации - представлены на рис. 1.2. Автономные системы управления являются самыми простыми и дешевыми, но их возможности ограничены. Более сложные локальные системы обладают широкими возможностями при управлении объектами различной природы и сложности. Наиболее сложными системами управления являются системы сетевой конфигурации с распределенным многоуровневым управлением и распределенным (рассредоточенным) объектом управления.

Из-за многообразия и противоречивости требований, предъявляемых к микропроцессорным реализациям встраиваемых систем, развитие и совершенствование микроконтроллеров идет по пути специализации. Количество различных моделей микроконтроллеров, производимых разными фирмами в настоящее время, очень велико.

1. Типовая структура микроконтроллера

Микроконтроллеры, реализованные в соответствии с общими архитектурными принципами, объединяют в семейства. МК семейства характеризуются совокупностью таких понятий, как процессорное ядро, система команд, организация памяти программ и памяти данных, базовый набор внутренних периферийных устройств и система прерываний.

Наиболее важной особенностью семейства является программная совместимость на уровне двоичных кодов всех входящих в него МК. Благодаря этой особенности возможна замена одних МК другими без изменения ранее разработанного программного обеспечения. Отличия между отдельными представителями одного семейства в основном заключаются в составе периферийных устройств и объеме памяти программ и данных. Сегодня в мире выпускаются тысячи типов МК. В группе лидеров такие компании, как Atmel, Intel, Motorola, Texas Instruments, Toshiba, Microchip Technology Inc., Zilog, Mitsubishi Electronics, Hitachi Semiconductor, Philips Semiconductors, Siemens и др. Функциональная схема МК представлена на рис. 1.3.

Микроконтроллер содержит в своем составе процессорное ядро с развитой системой команд, память программ IROM, регистровый файл данных RRAM (и может содержать небольшую внутреннюю память IRAM), а также набор программируемых интерфейсных схем, выполняющих ряд важных функций внутри МК и обеспечивающих связь с внешней средой. Большинство из перечисленных блоков являются обязательными компонентами любого МК, некоторые, например IROM, в структуре конкретного МК могут отсутствовать.

Состав и назначение интегрированных на кристалл периферийных устройств определяется областью преимущественного применения МК и может широко варьироваться в зависимости от типа МК. С целью повышения гибкости использования аппаратных средств, интегрированных на кристалле, и внешних выводов кристалла (для систем

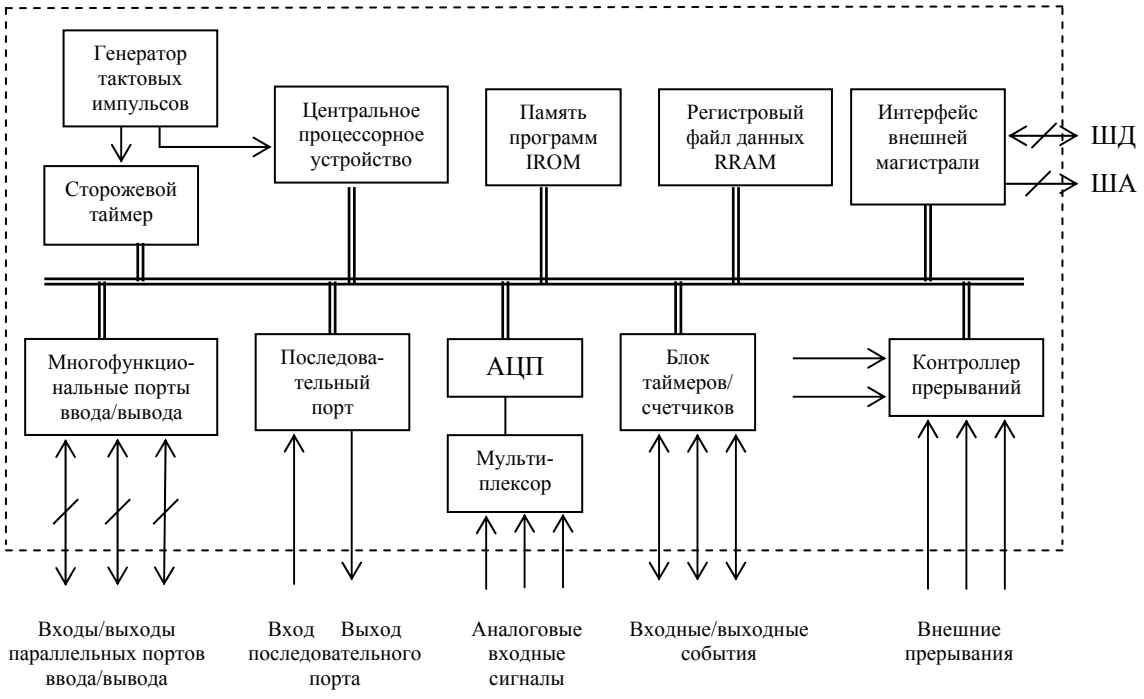


Рис. 1.3. Функциональная схема МК

различного назначения) эти средства делаются многофункциональными с программной настройкой на тот или иной режим работы. При инициализации МК информация о типе настройки заносится в специальные регистры и в процессе дальнейшей работы МК обычно остается неизменной. Это позволяет упростить оперативное управление встроенными средствами МК и расширить возможности их использования. Как правило, в число устройств встроенной периферии входят таймеры/счетчики, последовательные и параллельные порты ввода/вывода, контроллер прерываний, многоканальный аналого-цифровой преобразователь (АЦП), сторожевой таймер и пр. Рассмотрим назначение и функционирование отдельных блоков МК подробнее.

1.1. Центральное процессорное устройство

Центральный процессор ЦП (Central Processor Unit) современных МК представляет собой n-разрядный микропроцессор с фиксированными разрядностью и списком команд. Сегодня наибольшая доля мирового рынка МК принадлежит 8-разрядным устройствам (около 50 % в стоимостном выражении). За ними следуют 16-разрядные МК и цифровые сигнальные процессоры DSP (Digital Signal Processor). Каждая из названных групп занимает примерно 20 % рынка. DSP ориентированы на использование в системах цифровой обработки сигналов. Оставшаяся часть рынка распределена между мощными 32-разрядными МК и маломощными МК с ограниченным числом выводов.

Основой ЦП является операционный блок, включающий арифметико-логический блок (АЛУ) и файл регистров общего назначения (ОЗУ МК). Взаимодействие отдельных блоков МК, как внутри ЦП, так и с блоками внутренней периферии, осуществляется по внутренней шине. Связь МК с внешними устройствами реализуется через внешние выводы параллельных и последовательных портов.

При реализации современных МК используется весь спектр архитектурных решений, реализованных в МП [1]. Ранние модели МК, широко применяемые и в настоящее время, содержат одноадресный процессор с главным регистром аккумулятором. В подобном процессоре один из исходных операндов арифметических и логических операций по умолчанию размещается в аккумуляторе и в него же помещается результат. Второй операнд, адрес которого указывается в команде, является содержимым одного из регистров блока РОН или оперативной памяти. На основе одноадресной аккумуляторной архитектуры реализуются широко распространенные 8-разрядные МК семейства MCS-51 и МК других семейств. Аккумуляторная архитектура ЦП, обладая простотой структуры, позволяет сократить формат команд и благодаря этому уменьшить время их выборки из программной памяти. Однако в ряде случаев из-за дополнительных операций загрузки первого операнда в

аккумулятор и пересылки результата из аккумулятора в регистр назначения простота одноадресного ЦП снижает его производительность.

Более производительными являются многоадресные (двух- и трех-адресные) ЦП, в которых оба операнда и результат могут размещаться в любой ячейке интегрированного на кристалл регистрового файла RRAM. Архитектуру ЦП МК подобного типа, например семейства MCS-96, иногда называют *многоаккумуляторной архитектурой* или *регистр-регистровой архитектурой* [4]. Естественно, *многоаккумуляторная* архитектура требует более сложного формата команд. Ее характерными особенностями являются:

- необязательная предварительная загрузка одного из операндов в аккумулятор перед выполнением операции, вследствие чего уменьшение числа операций по пересылке данных и повышение скорости вычислений;
- возможность сохранять неизменным или модифицировать содержимое источника данных в результате выполнения операции;
- возможность расширения системы команд трехоперандными высокопроизводительными командами.

В середине 90-х годов разработаны высокопроизводительные МК, ЦП которых представляет собой RISC-процессор. Напомним, что в RISC-процессоре, благодаря использованию простых команд одной длины, исполняемых, как правило, за 1 такт, достигается существенное упрощение блока управления ЦП и повышается его производительность. RISC-микроконтроллерами являются МК семейства AVR фирмы Atmel. и МК семейства PICmicro фирмы Microchip. В целом ЦП различных типов МК характеризуются широким диапазоном производительности и зависящей от нее стоимостью.

Внутренняя память данных МК обычно представлена в виде регистрового файла RRAM. Объем этой памяти зависит от типа МК и может существенно отличаться не только в МК различных семейств, но и в отдельных моделях МК одного семейства (96 байт в AVR-МК, 256 байт в МК семейства MCS-51, 232-1000 байт в МК MCS-96).

Система команд типового МК ориентирована на решение задач управления, алгоритмы которых большей частью обеспечивают контроль состояния объекта и реализацию функций логического управления. Каждое семейство МК обладает собственной системой команд, которая характеризуется списком команд и их форматом. Форматы команд позволяют определить тип выполняемой операции, входные и выходные операнды, а также адрес команды, подлежащей выполнению на следующем шаге программы. Для задания операндов в большинстве случаев применяют практически все основные способы адресации – прямую, непосредственную, неявную, косвенную и ее разновидности.

Доля вычислительных алгоритмов в задачах управления относительно невелика. Наряду с традиционными группами команд пересылок,

арифметических и логических операций, условной и безусловной передачи управления, система команд МК обычно расширяется командами операций с отдельными битами (булевыми переменными). Отдельные программно доступные биты могут быть установлены, сброшены или заменены инверсным значением и, кроме того, могут пересылаться, анализироваться и использоваться в логических вычислениях. Реализация побитовой обработки наделяет ЦП МК свойствами булевого процессора, весьма эффективного для задач логического управления. Специализированные команды, облегчающие программирование специфических функций, например, вычисление цифровой свертки, в системы команд типовых МК, обычно не включаются. С особенностями организации ЦП, системой команд и способами адресации операндов конкретных МК можно познакомиться по материалам фирменных описаний МК.

1.2. Особенности организации памяти

Отличительной архитектурной особенностью МК является то, что в них для хранения команд и данных используются разделенные физически, а часто и логически блоки памяти. Это обусловлено тем, что программы МК после отладки обычно не меняются и их размещают в ПЗУ. В отличие от команд переменные при выполнении программы могут модифицироваться, поэтому для их размещения используют ОЗУ (оперативную память).

Архитектура ВМ, в которой программа и данные хранятся в логически и физически разделенных областях памяти, получила название Гарвардской архитектуры. В ВМ с Гарвардской структурой (рис. 1.4) доступ процессора к командам и данным осуществляется через две независимые раздельные шины – шину доступа к командам и шину доступа к данным.

Физическое и логическое разделение памяти программ и памяти данных позволяет организовывать независимое и параллельное во времени обращение к ним, обеспечивая возможность совмещать выполнение текущей команды с выборкой и дешифрацией следующей команды. Благодаря этому достигается повышение скорости работы процессора и облегчается реализация конвейерных методов работы аппаратуры

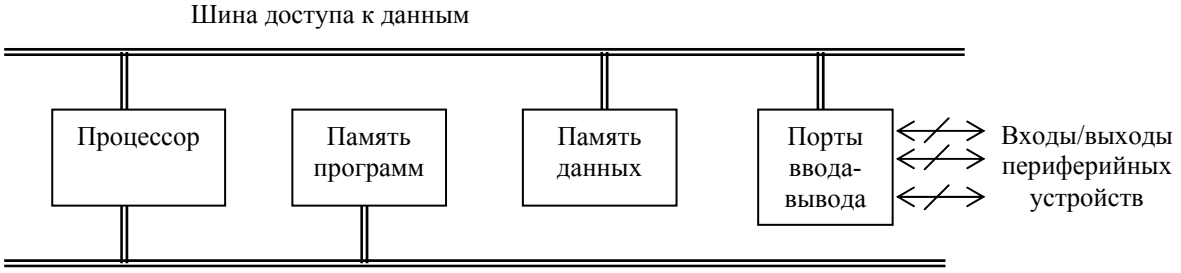


Рис. 1.4. Гарвардская структура ВМ

управления памятью и вычислителя. В соответствии с Гарвардской структурой реализуются МК семейства MCS-51 и x51-совместимые МК, производимые различными фирмами (Siemens, Atmel), МК AVR фирмы Atmel и др.

В отличие от Гарвардской архитектуры некоторые семейства МК, например семейство MCS-96, реализуются по Принстонской (фон неймановской) архитектуре, предполагающей размещение команд и данных в едином адресном пространстве памяти. В МК Принстонской архитектуры память команд и память данных разделены только физически (используются различные БИС памяти: ПЗУ - для хранения команд и ОЗУ - для хранения данных).

Объем внутренней памяти программ IROM в МК разных типов может варьироваться в достаточно широких пределах – от 1 Кбайт до 32 Кбайт, при этом в некоторых моделях, например в МК SAB 80C535, память IROM вообще может отсутствовать. МК без внутренней памяти программ используются в микроконтроллерных системах с внешней программной памятью, которые в большинстве случаев применяются на ранних этапах разработки и отладки новых систем. Большинство МК допускают расширение памяти программ за счет подключение дополнительных БИС внешней памяти через внешнюю магистраль МК, для организации которой обычно используют линии многофункциональных параллельных портов ввода/вывода. В современных МК в качестве IROM могут использоваться различные разновидности ПЗУ. МК с масочным (непрограммируемым пользователем) ПЗУ применяются при массовом (серийном) изготовлении МК с вполне определенной программой. Более распространены МК с программируемым ПЗУ. Производятся МК как с однократно программируемым ПЗУ (МК с OTP памятью – One Time Programmable), так и с перепрограммируемым ПЗУ (PROM – Programmable Read Only Memory). В МК с PROM памятью используется либо память с ультрафиолетовым стиранием EPROM (Erasable Programmable ROM) либо более дешевая, разработанная позже Flash-память EEPROM (Electrical Erasable Programmable ROM).

Регистровый файл данных RRAM в большинстве случаев объединяет две области памяти – оперативную память общего назначения, используемую пользователем по своему усмотрению, и область регистров специальных функций SFR (Special Function Registers). Набор регистров блока SFR определен для каждого типа МК и может существенно отличаться в МК разных семейств. Регистры специальных функций управляют внутренними периферийными устройствами МК, фиксируют их состояние и выполняют некоторые другие функции. Формат и назначение управляющих бит этих регистров обычно рассматривается при изучении особенностей функционирования и задании режимов работы внутренних устройств МК. Использование набора программируемых регистров SFR

позволило наделить МК свойством программной настройки внутренней периферии на требуемый режим работы, Это обеспечило многофункциональное использование узлов МК и решило проблемы выполнения широкого спектра требований, предъявляемых к МК как устройству общего назначения.

МК некоторых семейств, например семейство MCS-96, кроме регистрового файла данных RRAM, дополнительно содержат небольшую внутреннюю память IRAM (128 – 512 байт), в которой могут храниться как данные, так и команды. При размещении команд программы в IRAM появляется возможность модификации программы в процессе ее выполнения. При необходимости большинство МК допускают расширение памяти данных за счет подключения внешних БИС памяти с произвольным доступом.

Многие МК в своем составе содержат специальные схемы выборки кристаллов, представляющие собой программируемые селекторы адресов. Благодаря наличию таких схем, внешним устройствам памяти выделяются конкретные области адресного пространства и упрощается их подключение к МК. Особенности организации памяти МК рассматриваются ниже на примере микроконтроллерной системы с МК SAB 80C515 фирмы Infineon - клона МК семейства MCS-51.

1.3. Средства связи с объектом управления, другими МК и ЭВМ

Для связи МК с объектом управления (датчиками и исполнительными устройствами) в структуре МК обычно содержится широкий спектр специальных средств, обеспечивающих ввод-вывод как дискретных, так и аналоговых сигналов. В число таких средств входят устройства параллельного и последовательного ввода-вывода дискретных сигналов, а также устройства ввода-вывода аналоговых сигналов.

Порты ввода-вывода параллельной информации. В МК отдельные линии ввода/вывода дискретных сигналов объединяются в параллельные n-разрядные (обычно 8-разрядные) порты. Параллельные порты позволяют непосредственно вводить в МК и выводить из него данные в виде параллельного цифрового кода. Обычно их используют для связи с датчиками и исполнительными устройствами объекта управления, имеющего входные или выходные сигналы, представленные многоразрядными двоичными кодами. Сигналами подобного типа являются входные сигналы блока жидкокристаллических индикаторов, выходные сигналы аналого-цифровых преобразователей, входные сигналы цифроаналоговых преобразователей и др. Число портов ввода/вывода в современных микроконтроллерах непосредственно связано с числом выводов корпуса и, как правило, не превышает 4-6. Параллельные порты обеспечивают как ввод (передачу информации от внешнего устройства в МК), так и вывод (передачу информации от МК во внешнее устройство).

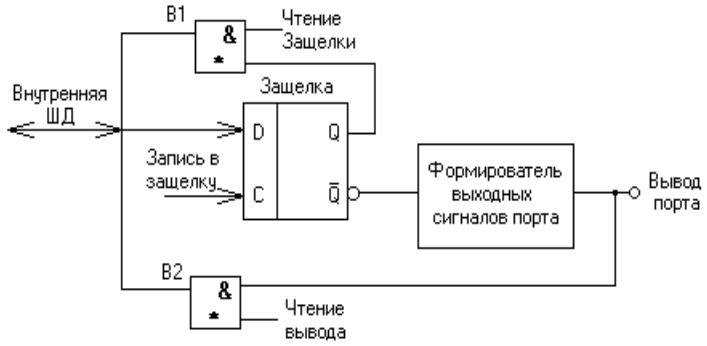


Рис. 1.5. Базовая структура одного разряда двунаправленного порта

Базовая структура одного разряда двунаправленного порта (на примере порта МК SAB 80C515) показана на рис. 1.5.

Каждый из разрядов порта ввода-вывода состоит из триггера-защелки, вентиля B1, формирователя выходных сигналов порта и входного буфера B2. Внутренний сигнал «запись в защелку» формируется при выводе, например, при выполнении команд *mov* P_n, *A*, в которых порт P_n является приемником (здесь *n* – номер порта).

При *выводе* данные с внутренней шины записываются в триггер-защелку порта и из него поступают на внешние выводы порта. Выход триггера Q через вентиль B1 может быть подключен к внутренней шине с помощью внутреннего сигнала «чтение защелки». Благодаря такой организации разряда порта обеспечивается программное чтение защелки и возможность побитовой обработки выходных сигналов порта с последующей записью результата обратно в защелку. Сигнал «чтение защелки» формируется в режиме «чтение - модификация - запись» при выполнении команд типа *anl direct,A, orl direct,#data, inc direct*, в которых *direct* – адрес порта. В этом режиме сначала читается содержимое адресуемого порта, затем выполняется заданная в команде операция и после этого осуществляется запись результата операции в регистр порта с выводом содержимого порта на внешний вывод МК.

При *вводе* данных работу порта поддерживает вентиль B2, управляемый внутренним сигналом «чтение вывода». При формировании сигнала «чтение вывода» в командах *mov A,*P_n данные с внешнего вывода порта через вентиль B2 поступают на внутреннюю шину МК и далее обрабатываются ЦП. Для правильной работы порта при вводе выходы формирователя выходных сигналов (рис. 1.5) должны отключаться от внешнего вывода (контакта) порта.

Все порты МК являются двунаправленными. Некоторые порты, например, порт, используемый для организации мультиплексируемой шины адрес/данные, должны обеспечивать возможность динамического переключения из режима вывода в режим ввода и обратное переключение.

Другие порты, а таких большинство, обычно используются для передачи сигналов в одном, задаваемом программой, направлении. Действительно, трудно представить конкретную микроконтроллерную систему, в которой физические устройства, подключенные к портам МК, во время работы изменяли бы свои функции и одновременно являлись бы датчиком и исполнительным устройством объекта управления. По этой причине такие порты МК реализуются как квазидвунаправленные, т.е. с программно задаваемой и неизменяемой во время работы функцией ввода-вывода. Особенности использования параллельных портов учитываются при построении формирователей выходных сигналов двунаправленного и квазидвунаправленного портов. Выходные каскады квазидвунаправленных портов выполнены на полевых транзисторах с внутренним нагрузочным резистором. Аналогичные каскады двунаправленного порта реализованы на транзисторах с открытым стоком.

Важной особенностью параллельных портов является возможность программирования отдельных разрядов портов для выполнения альтернативных функций. Каждый вывод многофункциональных портов может использоваться либо для выполнения стандартного ввода/вывода, либо для выдачи/приема специальных функциональных сигналов, поддерживающих работу внутренних периферийных устройств или управляющих внешними устройствами.

Последовательные порты. Обмен данными между МК и удаленными датчиками и исполнительными устройствами, как правило, осуществляется по последовательному каналу с помощью встроенных в структуру МК последовательных портов SP (Serial Port). Обычно в микроконтроллеры обязательно встраивается универсальный асинхронный последовательный приемопередатчик UART, поддерживающий протокол стандарта RS-232C. Кроме RS-232C, популярными протоколами информационного обмена для встраиваемых систем являются протоколы RS-485, RS-422, CAN (межконтроллерный сетевой интерфейс) и некоторые другие. Особенности реализации последовательных протоколов рассмотрены в [1].

Последовательная передача данных представляет собой реализацию трех последовательных процессов: преобразования параллельных данных источника информации в последовательный формат, передачи последовательной посылки по каналу связи и последующего преобразования приемником принятой посылки в параллельный формат. Последовательная передача характеризуется более низкой скоростью передачи, чем параллельная. Однако она обеспечивает связь на большие расстояния, и для ее реализации требуется меньшее число линий в физическом канале связи, что снижает стоимость. Вместо 3-5 метров при параллельных обменах при последовательной передаче обеспечивается надежная связь на расстояния более 15 метров.

При передаче цифровой информации по последовательному каналу связи она кодируется. Значения двоичных разрядов передаваемых данных в большинстве случаев представляются двумя уровнями напряжений. Двоичные разряды передаются в линию последовательно один за другим.

При организации последовательной связи на расстояния до 50-ти метров приемник и передатчик можно соединить друг с другом с помощью двух пар линий связи: по одной осуществляется передача информации от источника к приемнику, а по другой – обратная передача. При значительном удалении передатчика и приемника подобный способ организации последовательного обмена оказывается экономически дорогим. Более того, из-за искажающего влияния помех непосредственная передача логических сигналов (абсолютных уровней напряжений) по длинным линиям связи вообще невозможна. Поэтому для передачи на большие расстояния используют промежуточное частотное преобразование информационных сигналов. Такое преобразование осуществляется с помощью модемов. (Слово модем является аббревиатурой от слов модулятор/демодулятор, модуляция – это способ наложения переменного сигнала на другой сигнал постоянной несущей частоты). Модем на передающей стороне преобразует цифровой код последовательной посылки в колебания соответствующей звуковой частоты и передает этот сигнал по двухпроводной линии связи. Например, логическая 1 преобразуется в сигнал 1270 Гц, а логический 0 – в 1070 Гц. На приемной стороне с помощью принимающего модема выполняется обратное преобразование частотного сигнала в последовательный информационный сигнал, который непосредственно обрабатывается получателем. Передачу информации в обратном направлении можно осуществлять, используя ту же линию связи, но в другом частотном диапазоне, например, преобразуя логическую 1 в сигнал 2225 Гц, а логический 0 – в 2025 Гц. Таким образом, при использовании модемов достаточно всего одного канала связи, в частности обычной телефонной линии, для обмена информацией в обоих направлениях. Отмеченная особенность является важным достоинством способа последовательной передачи, так как позволяет использовать существующие системы связи для обмена информацией между удаленными устройствами. Скорость обмена информацией, при которой модем обеспечивает надежные передачу и прием, ограничивается возможностями телефонной линии связи, полоса пропускания которой составляет 300 – 3000 Гц.

В общем случае для организации последовательного обмена требуются два последовательных порта, которые должны быть установлены на входе и выходе последовательного канала. При передаче данных последовательный порт на передающей стороне преобразует параллельный код данных в последовательность двоичных символов.

Принимающий последовательный порт выполняет обратное преобразование последовательного кода в параллельный. Различают два основных способа передачи последовательных данных: асинхронный и синхронный.

В режиме *асинхронной передачи* источник и получатель данных не имеют общей синхронизации: синхронизация приемника и передатчика осуществляется от разных физических генераторов, которые должны быть настроены на близкие, в идеале одинаковые, частоты. Проблему синхронизации при асинхронном обмене решают путем ограничения длительности передаваемых посылок данных и использования стандартных частот работы приемо-передающих схем. В этом режиме процессор посылает и принимает отдельные байты информации, при этом канал работает асинхронно как в отношении передаваемых слов, так и в отношении отдельных битов. При асинхронной передаче последовательность двоичных символов данных дополняется служебной информацией, которая используется для синхронизации асинхронно работающих приемника и передатчика, а также контроля обмена. Каждая информационная посылка обрамляется стартовым и стоповыми битами (рис.1.6) , которые передаются вместе с битами данных. Пока по каналу нет передачи, он находится в состоянии логической 1 (стоповому биту, завершающему передачу слова, также соответствует значение логической 1). Передача данных всегда начинается со стартового бита (сигнала логического 0). Переход из 1 в 0, соответствующий началу стартового бита, может быть распознан приемником как признак начала слова. Передний фронт стартового бита используется для внутренней (принудительной) синхронизации приемника, благодаря чему обеспечивается правильное определение моментов поступления отдельных битов принимаемого слова. При правильной синхронизации значение принимаемого бита должно опрашиваться приемником в средине временного интервала, где искажения импульсов наименее влияют на величину считываемого уровня. После стартового бита следуют информационные биты в виде последовательности 0 и 1 (сигналов высокого и низкого уровня). В большинстве случаев число информационных бит в посылке равно 8. Далее в последовательной посылке может следовать контрольный бит, используемый для обнаружения ошибок. Его значение определяется содержимым передаваемого байта и тем, какой из двух режимов контроля установлен – контроль четности или контроль нечетности. Завершает передачу стоповый бит. Длительность стопового бита, определяющая временной интервал между отдельными передаваемыми словами, ограничений сверху не имеет. Минимальная длительность стопового бита в современных протоколах связи обычно программируется и не может быть меньше длительности 1, 1,5 или 2 периодов трансляции одного бита.

Передатчик может повторять стоп-бит до тех пор, пока приемник не будет готов получить следующий байт данных. Вид информационной посылки при асинхронных обменах показан на рис. 1.6.

Заметим, что вычислительное устройство (как источник, так и приемник цифровых данных) оперируют только данными и не имеют никакого отношения к служебным сигналам, передаваемым в информационной посылке. Последовательный порт на передающей стороне при передаче дополняет информационную посылку необходимыми служебными сигналами, а последовательный порт на приемной стороне после обработки удаляет их и передает цифровому приемнику только данные. Асинхронная передача данных отличается гибкостью, она не требует задания жестких моментов времени посылки очередных блоков данных.

При *синхронных* обменах прием и передача информационных бит сопровождается передачей специальных синхронизирующих импульсов по дополнительной линии связи. И источник, и приемник интерпретируют один и тот же синхроимпульс как указание на наличие в канале очередного бита. Передатчик, «договорившись» с приемником о параметрах связи, пересылает данные сплошным потоком без всякого разделения на блоки (байты). Синхронная передача отличается высокой скоростью, но хуже защищена от помех. Она широко применяется при обменах данными между блоками обработки данных и внешними ПУ на расстояния от нескольких сантиметров до нескольких метров, требуя минимального физического интерфейса.

Наиболее распространенным стандартом последовательной связи является стандарт RS-232C.

Важным назначением последовательного порта является не только обеспечение связи с пространственно удаленными датчиками и исполнительными механизмами собственного контроллера, но и его использование для организации связи с другими локальными регуляторами и ВМ верхнего уровня управления. «Мощные» МК, как правило, содержат несколько последовательных портов, в том числе специальный порт, поддерживающий такую связь.

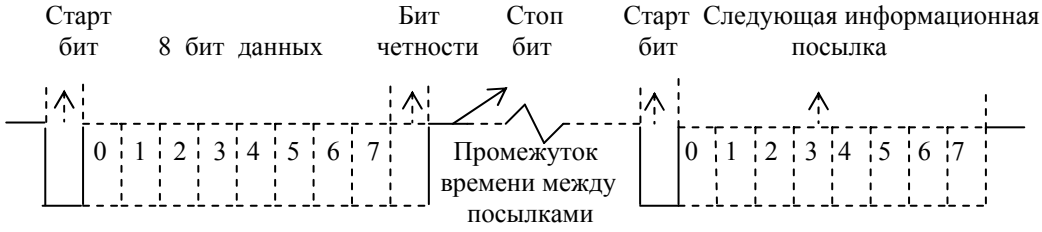


Рис. 1.6. Вид информационной посылки при асинхронной передаче

Встроенные средства ввода и вывода аналоговых сигналов, размещенные на кристалле МК, обеспечивают связь МК с аналоговыми датчиками и исполнительными устройствами объекта управления. С их помощью входные аналоговые сигналы от датчиков преобразуются в цифровой код для последующей обработки процессором, а также производится формирование аналоговых управляющих воздействий.

Для преобразования аналоговых сигналов в цифровой код используются аналого-цифровые преобразователи АЦП. Современные МК обычно содержат в своем составе 8-канальный АЦП, обеспечивающий преобразование аналоговых сигналов напряжения в диапазоне от 0 до 5,12 В в цифровой 8-ми или 10-ти разрядный код. Запуск преобразования в канале может производиться программно или аппаратно. Некоторые преобразователи могут работать в режиме сканирования входов.

Преобразование аналоговых сигналов в цифровой код в МК чаще всего выполняется методом поразрядного уравновешивания. В соответствии с этим методом осуществляется последовательное сравнение входного аналогового сигнала с рядом формируемых эталонных напряжений и выбирается такое эталонное напряжение, которое наиболее близко к значению входного сигнала.

Упрощенная функциональная схема АЦП поразрядного уравновешивания (рис. 1.7) содержит аналоговый компаратор, выполняющий сравнение формируемого эталонного напряжения $U_э$ с входным измеряемым $U_{изм}$, выходной регистр результата и внутренний цифро-аналоговый преобразователь (ЦАП). Результат аналогово-цифрового преобразования фиксируется в регистре результата. ЦАП, подключенный к выходу регистра результата, преобразует цифровой n-разрядный код регистра в эталонное напряжение сравнения $U_э$.

Уравновешивание начинается со старшего разряда регистра результата. В этом разряде вначале устанавливается «1», и ЦАП вырабатывает напряжение $U_э/2$. Это напряжение сравнивается с входным и оценивается знак разности. Если выясняется, что уравновешивающий сигнал меньше преобразуемого, то установленная в старшем разряде «1» сохраняется, если больше – «1» сбрасывается, и в старшем разряде будет храниться «0». Далее аналогично проверяется, нужна ли «1» в следующем, более

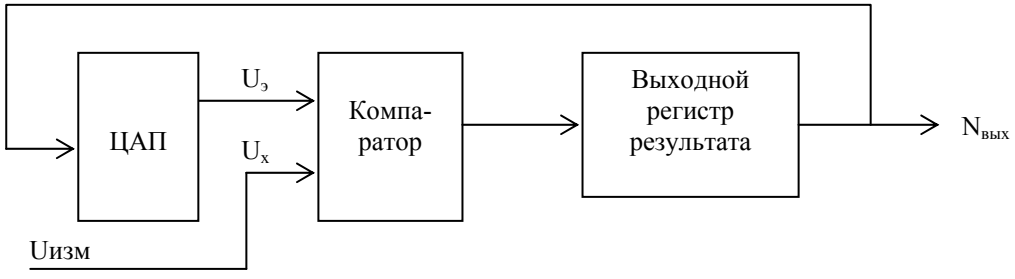


Рис. 1.7. Упрощенная функциональная схема АЦП

младшем разряде. Уравновешивание продолжается, пока не будут опрошены все разряды регистра, включая самый младший.

Преобразование цифровых данных в аналоговый сигнал может быть выполнено с помощью внешних цифроаналоговых преобразователей. Однако такой способ преобразования требует использования значительного числа выходных линий параллельного порта для выдачи преобразуемого цифрового кода. Более простым является способ преобразования цифровых данных в аналоговый сигнал с помощью широтно-импульсного модулятора (ШИМ) и внешнего фильтра нижних частот (ФНЧ), рассматриваемый в п. 2.6.3.

1.4. Средства тайминга и формирование времязадающих функций

Задачи управления различным оборудованием в реальном времени предопределили обязательное наличие и использование в МК таймеров/счетчиков. С их помощью осуществляется не только счет внешних событий, но и существенно упрощается формирование типовых управляющих сигналов в функции времени. В большинстве случаев таймеры обеспечивают реализацию следующих функций:

- деление внешней или внутренней частоты;
- счет событий;
- генерация импульсов заданной длительности с программным и/или аппаратным запуском;
- регистрация и генерация событий;
- функции сторожевого таймера.

Блок таймеров/счетчиков МК, как правило, содержит в своем составе несколько таймеров общего назначения, сторожевой WDT (Watchdog Timer) таймер и набор устройств, с помощью которых реализуются функции быстрого ввода/вывода.

Таймеры общего назначения используются для отсчета интервалов реального времени и привязки к нему отдельных программных событий. Обычно они реализуются на базе 16-разрядных счетчиков и имеют несколько программируемых режимов работы. Изменяя входную тактовую частоту таймера, можно задавать требуемую разрешающую способность таймера (минимальный квант времени таймера равен одному периоду тактовой частоты). В состав упрощенной структуры физической модели таймера (рис. 1.8) включены следующие основные блоки:

- 16-разрядный регистр для приема начальных значений счета;
- программно недоступный 16-разрядный счетчик TH:TL, осуществляющий счет входных импульсов и формирующий выходной сигнал τ_{ov};
- 16-разрядный выходной буферный регистр, отображающий текущее содержимое счетчика TH:TL.

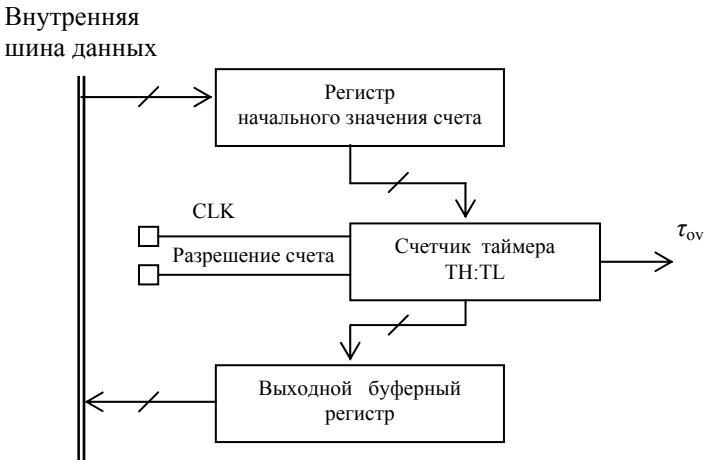
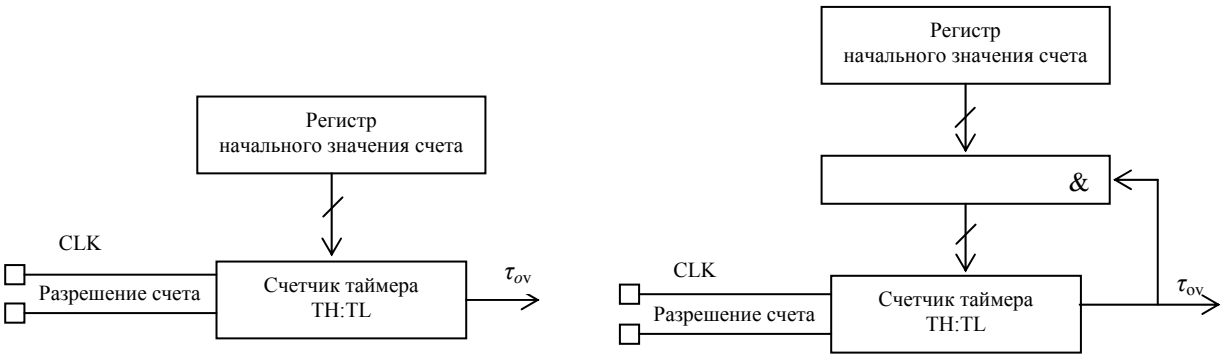


Рис. 1.8.Упрощенная структура таймера

При разрешенном счете на вход счетчика таймера TH:TL от внешнего или внутреннего источника поступают счетные импульсы, каждый из которых увеличивает содержимое TH:TL на 1. При переполнении счетчика на выходе таймера формируется выходной сигнал τ_{ov}. Обычно таймер имеет несколько режимов работы, обеспечивающих реализацию различных времязадающих функций. Настройку таймера на заданный режим работы осуществляют с помощью управляющих бит регистра управления (на рис. 1.8 не показан).

«Режим программируемой задержки» (рис. 1.9,а) реализуется при однократной загрузке счетчика TH:TL данными из регистра начального значения счета. При запуске таймера в этом режиме обеспечивается аппаратное формирование выходного импульса τ_{ov} через заданное время.

В режиме «управляемого генератора» (рис.1.9,б) на выходе таймера генерируются импульсы программируемой частоты. Данный режим реализуется при использовании импульсов τ_{ov} для циклической



Структура таймера в «режиме программируемой задержки»

а)

Структура таймера в «режиме управляемого генератора»

б)

Рис. 1.9. Структуры таймера в различных режимах работы

перезагрузки счетчика TH:TL данными из регистра начального значения счета

В режиме «счетчика» в TH:TL осуществляется подсчет внешних событий (входных импульсов на входе CLK). Этот режим обеспечивается при загрузке счетчика TH:TL нулевым значением и последующим разрешением счета. Значение счетчика контролируется при чтении содержимого выходного буферного регистра.

16-разрядный счетчик TH:TL не имеет непосредственной связи с внутренней шиной данных МК, а, следовательно, и с его CPU. Программирование параметров формируемых сигналов таймера и чтение содержимого счетчика TH:TL осуществляется через программно доступные буферные регистры – регистр начального значения счета и выходной буферный регистр. Названные регистры обычно имеют одинаковое имя TH:TL. Это не приводит к ошибкам, поскольку при программном доступе к ним возможность неправильного обращения исключена: команда записи в регистр TH:TL осуществляет пересылку данных из CPU в регистр начального значения счета, а при выполнении команды чтения содержимого TH:TL реализуется пересылка данных из выходного буферного регистра таймера в CPU. Рассмотренные особенности организации таймера учитывается в его программной модели. Для программиста таймер представляется в виде одного TH:TL.

В современных МК в дополнение к стандартному вводу/выводу, реализуемому с помощью параллельного и последовательного портов, обработка входных и формирование выходных дискретных сигналов часто осуществляется с помощью специальных периферийных устройств, называемых блоками быстрого ввода-вывода. Отличительной особенностью использования таких устройств является то, что быстрый ввод и быстрый вывод дискретных сигналов с их помощью выполняется без непосредственного участия процессора.

Блоки быстрого ввода/вывода. Практика применения МК диктовала необходимость использования таймеров и в качестве часов «реального времени». Блок быстрого ввода/вывода HSIO (High Speed Input-Output), реализованный фирмой Intel в МК 8xC51FX, стал первым устройством, предназначенным для регистрации входных и генерации выходных событий в реальном времени. Под событием здесь понимается изменение входного или выходного сигнала на внешнем выводе МК. В большинстве случаев блок HSIO (рис. 1.10) реализуется на основе 16-разрядного таймера/счетчика TH:TL, который связан с массивом программируемых счетчиков PCA (Programmable Counter Array) через модули выборки и сравнения событий. Блок HSIO содержит несколько каналов. В состав канала входит 16-разрядный многофункциональный регистр захвата/сравнения CCL/CCH, а также 16-разрядные схемы сравнения (CMP) и логического совпадения И.

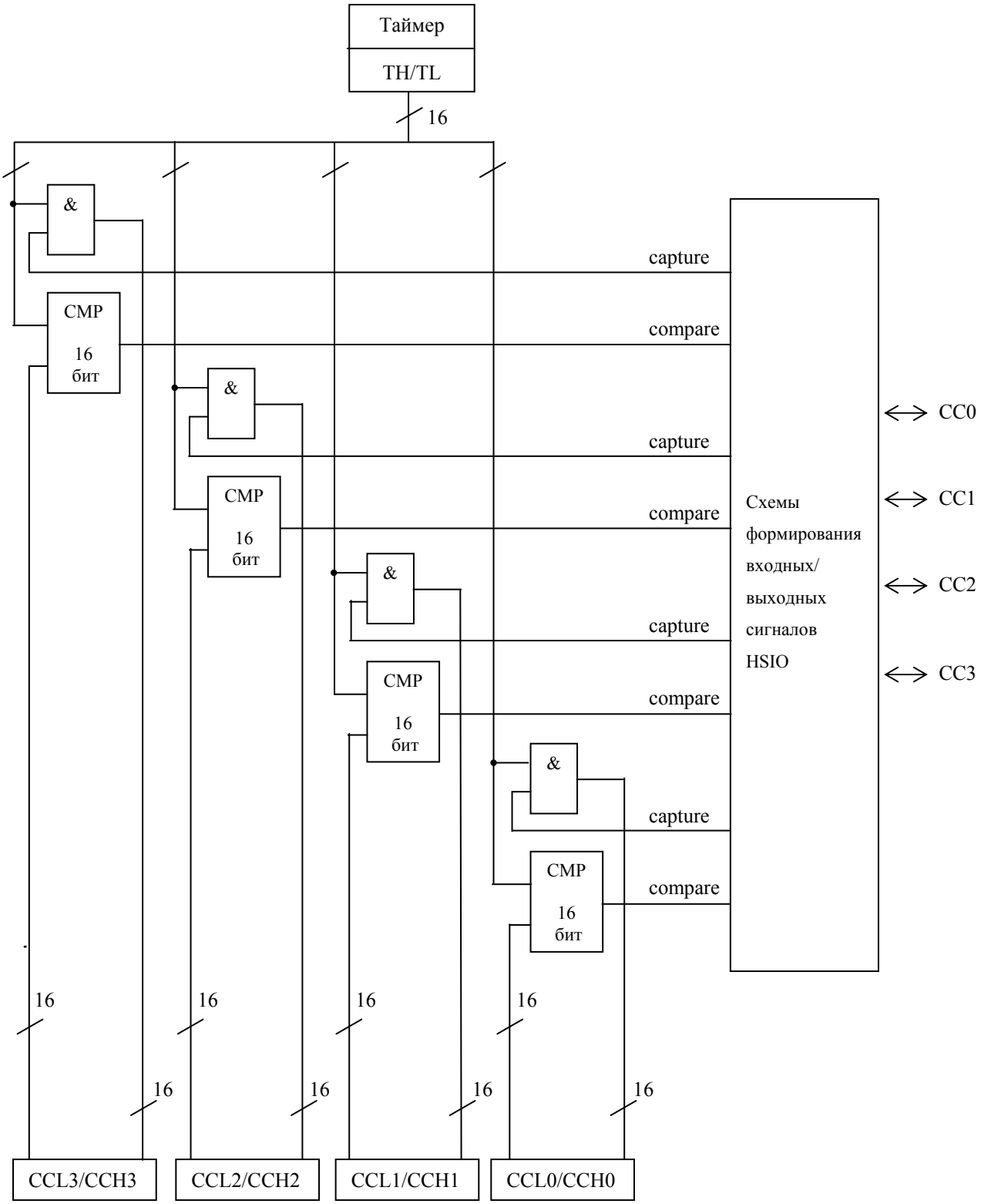


Рис. 1.10. Структурная схема блока HSIO

Быстрый ввод заключается в обнаружении входного события - сигнала заданного (программируемого) вида и запоминании времени его наступления в заданной системе отсчета времени. Обычно МК может идентифицировать одно из следующих типов событий: формирование

положительного перепада входного сигнала (переход из 0 в 1), формирование отрицательного перепада входного сигнала (переход из 1 в 0), наличие любого перепада входного сигнала (из 0 в 1 или из 1 в 0). Время наступления события заданного вида, полученное из таймера, фиксируется в регистре CCL/CCH и называется *захватом события*. Захват события сопровождается запросом прерывания центрального процессора на обслуживание высокоскоростного ввода

В режиме *высокоскоростного вывода* блок HSIO осуществляет сравнение текущего времени, формируемого таймером, с заданным в регистре CCL/CCH. В момент равенства значений обеспечивается генерация выходного события - сигнала определенного, программируемого вида. Выходным событием может быть не только внешнее событие, формируемое на выходе МК, но и внутреннее событие. Внутренние события не связаны с изменением потенциала на каком-либо выводе МК и применяются, например, для запуска встроенного аналого-цифрового преобразователя в требуемый момент времени. Важным подрежимом быстрого вывода является формирование ШИМ-сигналов, рассматриваемое ниже.

В обоих режимах высокоскоростного ввода/вывода текущее значение времени (сравниваемое или регистрируемое) формируется на выходе таймера/счетчика, который осуществляет счет входных импульсов от внутреннего генератора МК или от внешнего источника.

Все функции блока HSIO (определение момента наступления внешнего события, формирование сигналов заданного вида в требуемые моменты времени, генерирование ШИМ-сигналов) выполняются на аппаратном уровне, не требуя участия ЦП. Благодаря этому повышается общая пропускная способность микроконтроллерной системы, точность измерения и отработки сигналов и снижается время реакции МК на внешние события, что особенно важно для систем реального времени. Для реализации функций захвата и сравнения базовый таймер дополняют схемами захвата внешних событий и каналами сравнения.

Конкретная организация блока HSIO и особенности его функционирования рассматриваются в п.2.4.5 на примере блока HSIO МК SAB80C515/535.

Сторожевой таймер WDT. Для контроля правильности функционирования программного обеспечения большинство современных МК имеют в своем составе сторожевой таймер WDT. Таймер WDT предназначен для борьбы с программными сбоями системы. С его помощью контролируется длительность интервала τ между некоторыми событиями. Если ожидаемое событие не наступает в пределах интервала τ, таймер вызывает прерывание программы, при этом формируется внутренний сигнал сброса МК и осуществляется рестарт программы. В условиях сильных промышленных помех (электромагнитных,

электростатических и других), когда возможны программные сбои и искажения при считывании данных, WDT обеспечивает защиту от нештатных ситуаций, которые трудно предусмотреть заранее, но которые могут сделать программу неработоспособной. Примером такой нештатной ситуации может быть зависание программы вследствие ожидания события, которое по какой-либо причине не может произойти, или зацикливание программы из-за сбоя.

Таймер WDT обычно реализуется на базе 16-битного счетчика, который инкрементируется с определенной частотой. После запуска WDT возможно только его программное обнуление. При правильном функционировании системы пользовательское ПО принудительно обнуляет счетчик WDT непосредственно в ходе выполнения программы, не допуская его переполнения. Если программное обнуление счетчика WDT не выполняется, то WDT переполняется и формируется внутренний сигнал аппаратного сброса МК, вызывающий рестарт программы.

Аппаратные средства контроля и защиты. Микроконтроллеры, как правило, не содержат встроенных средств тестирования внутренних устройств. Для контроля правильности функционирования программного обеспечения, защиты внутренней памяти программ, контроля допустимого уровня снижения частоты тактового генератора и пр. в составе МК могут использоваться специальные аппаратные средства контроля и защиты. Кроме таймера WDT МК некоторых типов, например 8XC51GB фирмы Intel, имеют специальную схему обнаружения падения частоты тактовых импульсов. При снижении частоты ниже определенного уровня такая схема вырабатывает внутренний сигнал сброса МК.

Для защиты внутренней памяти программ от несанкционированного доступа (программного пиратства) в ПЗУ некоторых типов МК размещаются специальные программируемые пользователем байты шифрации, которые изначально не запрограммированы (все «1»). При чтении (верификации) программного кода процедура верификации выполняется как обычно за исключением того, что над каждым байтом кода перед чтением выполняется операция «Исключающее ИЛИ-НЕ» с одним из байтов защиты. Поэтому чтобы прочитать программный код, необходимо знать закодированные байты шифрации в их правильной последовательности. Если область шифрации оставить незапрограммированной, при чтении любой байт программного кода остается неизмененным.

Встроенная система прерываний. Наряду с таймерами работу встраиваемых систем управления в реальном времени поддерживает система прерываний МК. Прерывания являются универсальным средством организации управления, поскольку обеспечивают автоматический запуск различных процедур обслуживания внешних или внутренних устройств

микроконтроллерной системы по их запросу. Встроенная система прерываний позволяет отказаться от непрерывного контроля состояния различных устройств, который требует существенных затрат процессорного времени. Система прерываний МК реализуется с помощью размещенного на кристалле контроллера прерываний. Последний представляет собой устройство, однозначно определяющее адрес подпрограммы обработки поступившего запроса. Другой функцией контроллера прерываний является задание приоритетов обработки поступающих запросов прерываний. Понятие приоритет означает, что выполняемая программа обработки прерывания может быть прервана другим запросом прерывания только при условии, что этот запрос имеет более высокий приоритет по сравнению с выполняемым. В противном случае обработка нового запроса возможна только после окончания обработки текущего.

Реакция процессора на запросы прерывания от любого источника идентична: в конце выполнения каждой команды процессор опрашивает регистр запросов и выбирает наиболее приоритетный из поступивших. После идентификации источника запроса процессор сохраняет минимальный контекст текущей программы (по крайней мере, адрес возврата) и вызывает обработчик прерывания. За каждым прерыванием закрепляется адрес программной памяти - вектор прерывания. При вызове обработчика управление программой передается по адресу вектора прерывания. Во время обработки текущего прерывания все прерывания равного и меньшего приоритета запрещаются. Программа обработки завершается обязательной командой RETI, при исполнении которой обеспечивается возврат к продолжению прерванной программы и восстановление логики прерываний. В течение времени обработки другие запросы прерывания одинакового или меньшего приоритета продолжают фиксироваться. Пример организации и особенности реализации системы прерывания МК SAB 80C515 рассматривается в подразд. 2.8.

1.5. Управление потребляемой мощностью.

Для приложений, где потребление энергии критично (в большинстве случаев это устройства с батарейками, аккумуляторным питанием), важными требованиями являются малое энергопотребление и возможность программного управления потребляемой мощностью. Для удовлетворения указанных требований СБИС современных МК реализуют по КМОП-технологии, которая обеспечивает малое потребление энергии, и используют встроенные программно-аппаратные средства для управления режимами энергопотребления.

В МК имеется два режима пониженного энергопотребления – *режим холостого хода Idle* и *режим микропотребления (Power Down mode)*.

В режиме Idle (этот режим также называют режимом ожидания) работа ЦП блокируется, но работа тактового генератора и всей внутренней и внешней периферии не запрещается. За счет прекращения работы ЦП энергопотребление уменьшается. В режиме Idle обеспечивается нормальная работа таймеров, поддерживается работа АЦП и при незавершенных приеме или передаче последовательного порта продолжается его работа. Реагируя на входные сигналы, микроконтроллер в этом режиме обеспечивает захват внешних событий и некоторые другие функции. Уменьшение энергопотребления, которого можно достигнуть в Idle режиме, зависит от числа работающей периферии. Минимум потребления достигается, если все таймеры остановлены, АЦП и последовательный порт не работают.

В режиме микропотребления, иногда называемом режимом выключенного напряжения питания, генератор тактовых импульсов МК отключается, прекращая тем самым работу всех блоков МК, однако внутренняя память IRAM сохраняет свое содержимое, потребляя очень маленький ток (на 2 – 3 порядка меньший, чем в рабочем режиме). В отличие от режима холостого хода, в котором производительность МК снижается, но не прекращается его работа, в режиме *микропотребления* МК останавливается, и никакие операции не выполняются.

Завершая рассмотрение назначения и особенностей функционирования основных блоков МК, отметим следующее. Несмотря на непрерывное развитие и появление большого числа новых МК, функциональная организация МК практически не меняется. Однако каждое семейство МК и даже каждый МК одного семейства имеют некоторые специфические характеристики и особенности использования. Построение конкретных микроконтроллерных систем должно выполняться в строгом соответствии с фирменными справочными руководствами, в которых приводятся подробные описания каждого МК и рекомендуемые схемы его включения. В рамках учебного пособия невозможно рассмотреть все известные типы МК, но в этом и нет необходимости в виду общности подходов, применяемых при построении МК разных фирм.

Особенности организации и функционирования отдельных подсистем МК рассмотрим на примере одного из распространенных МК семейства MCS-51 – МК SAB 80C515 фирмы Infineon (дочерней фирмы компании Siemens). Выбор указанного МК для рассмотрения определен следующим:

1. Существует очень большое количество приложений, для которых использование 8-разрядных микроконтроллеров обеспечивает требуемые функции управления при сравнительно низкой стоимости и приемлемых массогабаритных характеристиках.

2. 8-разрядные МК семейства MCS-51 фирмы Intel хорошо известны разработчикам в нашей стране, так как некоторые аналоги младших членов

семейства выпускаются отечественной промышленностью [3]. Простота структуры, относительно низкая стоимость и наличие средств поддержки проектирование определяют популярность МК семейства MCS-51.

3. Наиболее существенным в выборе указанного МК является наличие лабораторных стендов, реализованных на основе МК SAB 80C515, и методической поддержки лабораторных работ, выполняемых студентами кафедры КСПТ СПбГПУ при изучении курса «Микропроцессорные системы».

2. Управляющий восьмиразрядный МК SAB 80C515

Основные характеристики 8-битного МК SAB 80C515 следующие:

- 8-разрядное АЛУ и схемы аппаратной реализации умножения и деления;
- внутреннее ПЗУ (IROM) программ и констант объемом 8 Кбайт;
- возможность расширения памяти программ до 64 Кбайт за счет подключения внешних микросхем ПЗУ;
- регистровый файл данных RRAM объемом 256 байт;
- возможность подключения внешнего ОЗУ данных объемом до 64 Кбайт;
- блок регистров специальных функций SFR;
- шесть программируемых параллельных портов ввода-вывода (P0–P5);
- дуплексный последовательный порт с фиксированной и переменной скоростью обмена;
- 8-входовой порт ввода аналоговых сигналов (P6);
- 8-канальный аналого-цифровой преобразователь со встроенным блоком программируемых эталонных напряжений;
- три программируемых 16-битных таймера/счетчика;
- 4-канальный блок быстрого ввода-вывода внешних событий, обладающий дополнительными возможностями формирования ШИМ-сигналов;
- сторожевой таймер (WDT);
- 4-уровневая система прерываний от 12 источников прерываний;
- два режима пониженного энергопотребления: режим холостого хода (ожидания) Idle и режим микропотребления (Power Down)
- внутренний, моделируемый в памяти RRAM стек глубиной до 256 байт;
- минимальное время выполнения команд – 1 мкс при F_T = 12 МГц;
- аппаратные умножение и деление, выполняемые за 4 мкс.

Функциональная и структурная схемы МК SAB 80C515 приведены на рис. 2.1 и рис. 2.2. Микроконтроллер кроме 48 выводов параллельных портов и 8 выводов входных аналоговых сигналов имеет всего 5 выводов управления:

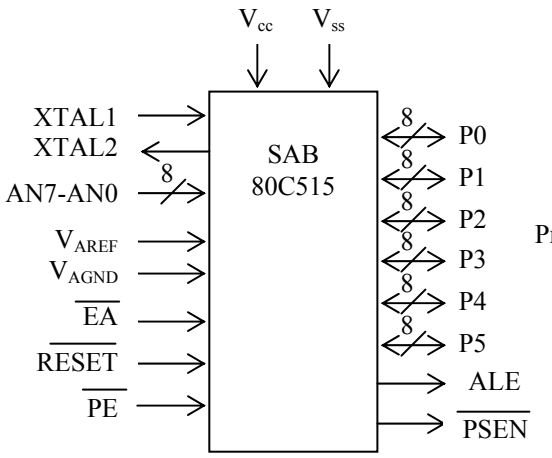


Рис. 2.1.Функциональная схема МК SAB 80C515

Reset (начальная установка), PE (разрешение программирования) и выводы EA (Enable Address), ALE (Address Latch Enable) и PSEN (Program Store ENable), используемые при работе с внешней памятью программ (см. рис. 2.1). Структурная схема МК SAB 80C515 содержит практически те же блоки, что и функциональная схема типового МК (рис. 1.3). В состав МК входит центральное процессорное устройство CPU, память программ IROM, память данных RRAM, а также набор периферийных устройств, выполняющих ряд важных функций внутри МК и обеспечивающих связь с внешней средой.

2.1. Центральное процессорное устройство МК SAB 80C515

CPU МК 80C515 представляет собой 8-разрядный микропроцессор с фиксированными разрядностью и списком команд. Основу CPU составляют арифметико-логический блок (блок АЛУ) и устройство управления, содержащее регистр команд, дешифратор команд, счетчик команд PC и схему синхронизации. Взаимодействие отдельных блоков как внутри CPU, так и с периферийными внутренними устройствами МК осуществляется по 8-разрядной внутренней шине данных. Связь МК с внешними устройствами осуществляется через выводы двунаправленных портов ввода/вывода P0 – P5. Расширение системы за счет подключения внешней памяти и дополнительных устройств ввода-вывода, реализуется с помощью мультиплексируемой шины адрес/данные, организуемой с использованием линий порта P0 и P2

Арифметико-логический блок предназначен для выполнения арифметических и поразрядных логических операций, а также операций с отдельными битами. В состав блока АЛУ входят: собственно АЛУ, реализованное на базе комбинационного сумматора с последовательным переносом, регистр результата (аккумулятор) ACC, дополнительный регистр B (расширение аккумулятора для некоторых команд), регистры временного хранения операндов БР1 и БР2 и регистр признаков результата PSW (регистр слова состояния программы).

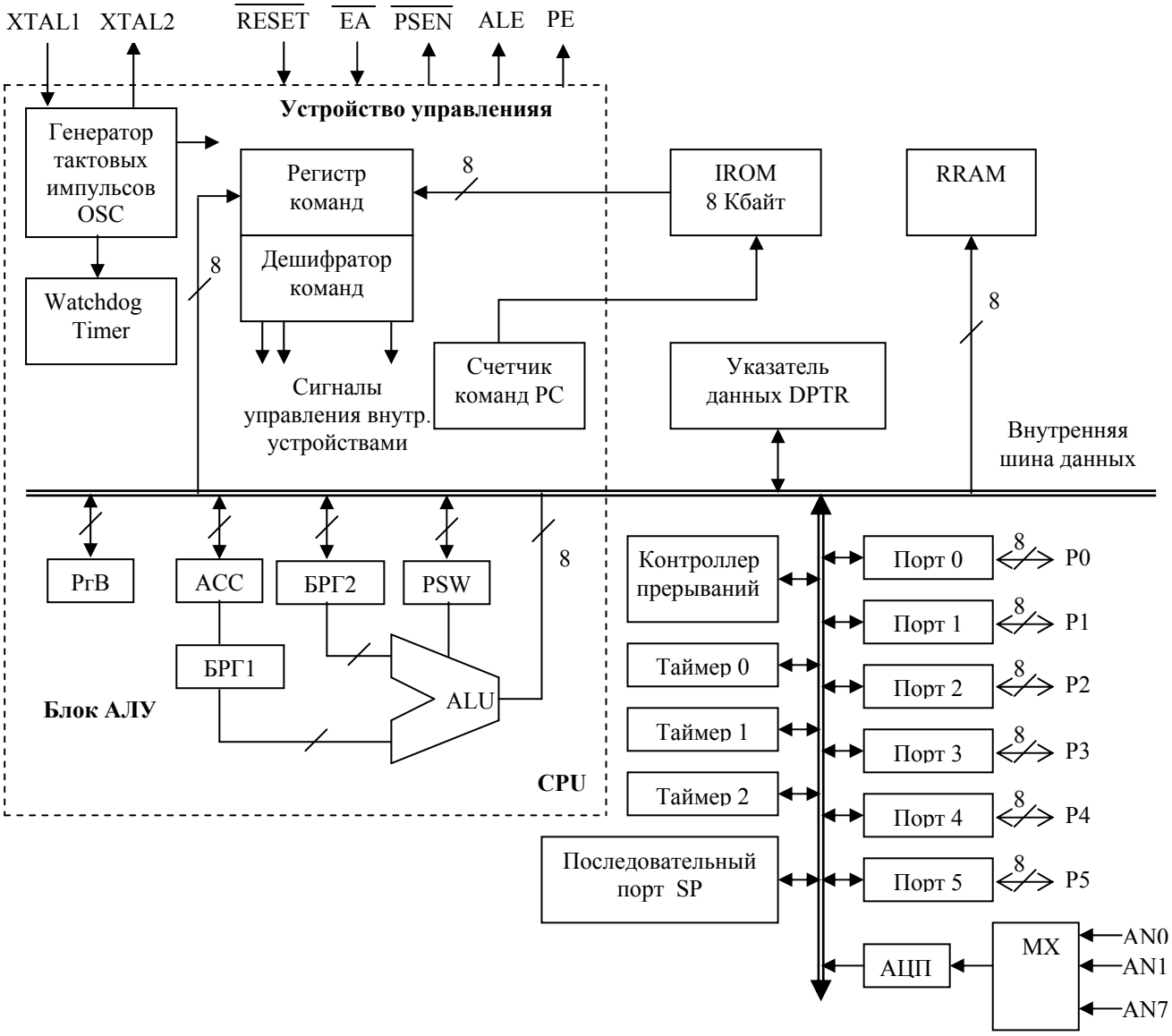


Рис. 2.2. Структурная схема МК SAB 80C515

CPU реализован по схеме одноадресного вычислителя с главным регистром - аккумулятором ACC. Все арифметические и логические операций реализуются с использованием аккумулятора. При их выполнении аккумулятор является источником одного из операндов и приемником результата. Второй операнд в командах арифметических и логических операций может поступать из памяти RRAM. Большинство команд пересылок также выполняется с участием аккумулятора. В них, как и во всех других командах, в которых участвует аккумулятор, он адресуется неявно. В мнемонике команд аккумулятор обозначается *A*.

8-разрядный регистр *B* используется в качестве регистра-расширителя ACC при выполнении операций умножения и деления. В других командах регистр *B* может использоваться как дополнительный регистр блока РОН.

При выполнении арифметических и логических команд на выходе АЛУ формируются признаки (флаги) результата выполненной операции: флаг переноса C, флаг межтетрадного переноса AC, флаг переполнения OV, флаг четности (паритета) P и флаг нуля Z. Первые четыре флага фиксируются в регистре PSW. Флаг нуля Z в регистре PSW не фиксируется.

Регистр PSW (рис. 2.3) также содержит два флага состояния F0 и F1, которые могут программироваться пользователем, и два бита RS1 и RS0 указателя рабочего банка регистров общего назначения. В составе CPU МК SAB80C515 имеется четыре банка рабочих регистров по восемь регистров (R0 – R7) в каждом. Банки регистров размещаются во внутренней памяти данных RRAM (см. п. 2.2).

C	AC	F0	RS1	RS0	OV	F1	P

Рис. 2.3. Формат регистра PSW

К программно доступным регистрам, поддерживающим работу CPU, относятся три регистра блока специальных функций SFR: 16-разрядный счетчик команд PC, 8-разрядный указатель стека SP (на рис. 2.2 не показан) и 16-разрядный регистр-указатель данных DPTR.

16-разрядный счетчик команд PC предназначен для формирования текущего адреса программной памяти (внутренней и внешней). После чтения (выборки) очередного байта команды из программной памяти содержимое PC инкрементируется на 1.

С помощью 16-разрядного указателя данных DPTR обеспечивается косвенная адресация операндов при обращении к ячейкам внешней памяти данных и считывание констант из программной памяти. Механизм обращения к внешней памяти программ и внешней памяти данных рассматривается ниже.

В МК SAB 80C515 используется стек, моделируемый во внутренней памяти данных IRAM. 8-битный указатель стека SP может адресовать любую область RRAM и не может превышать 256 байт. В реальных системах фактическая глубина стека определяется непосредственно пользователем, который должен позаботиться о том, чтобы стек не перекрывал другие области данных программы, так как при загрузке стека эти данные будут испорчены. При загрузке данных в стек он растет вверх (в область старших адресов) и всегда готов к чтению. При инициализации в SP записывается значение 07, указывающее на вершину стека.

Блок управления CPU. С помощью этого блока вырабатывается последовательность управляющих сигналов как внутренних, так и внешних, обеспечивающих выполнение любой команды. Блок управления состоит из программно недоступного регистра команд, дешифратора команд и схемы формирования управляющих сигналов. Регистр команд

предназначен для запоминания и последующей передачи первого байта команды в дешифратор команд. Команда в зависимости от ее типа может состоять из одного или нескольких байт, однако первый байт любой команды всегда запоминается в регистре команд без учета того, какая его часть отведена для кода операции. Дешифратор команд перекодирует содержимое регистра команд в управляющее слово.

Синхронизация CPU осуществляется импульсами синхронизации внутреннего генератора тактовых импульсов. Блок синхронизации формирует как внутренние синхросигналы (машинные циклы, состояния, фазы), так и внешние управляющие сигналы, используемые при обращении к внешним БИС памяти. Понятие машинного цикла в МК семейства MCS-51 отличается от определения этого понятия для других МП. В системах с МП 8080 машинный цикл – это время выполнения одного внешнего обмена. В МК семейства MCS-51 машинный цикл – это аппаратно формируемый (жестко заданный) интервал времени, используемый для синхронизации операций при их выполнении.

Машинные циклы (рис. 2.4) имеют фиксированную длительность (12 периодов тактовой частоты) и содержат 6 состояний (S1, ..., S6), каждое из которых, в свою очередь, состоит из двух фаз P1 и P2. Длительность фаз равна периоду следования импульсов синхронизации. Машинные циклы начинаются фазой S1P1 и заканчиваются фазой S6P2. В фазе P1, как правило, выполняется типовая арифметическая или логическая операция, а в фазе P2 осуществляется межрегистровая передача.

В каждом машинном цикле состояния S1 и S4 зарезервированы для выполнения операций чтения программной памяти. В эти моменты времени в зависимости от того, к какой памяти программ (внутренней или внешней) осуществляется обращение, МК формирует соответствующий сигнал чтение: при доступе к внутренней памяти программ формируется внутренний сигнал RD Opcode, а при доступе к внешней памяти – выходной сигнал $\overline{\text{PSEN}}$ (Program Store ENable).

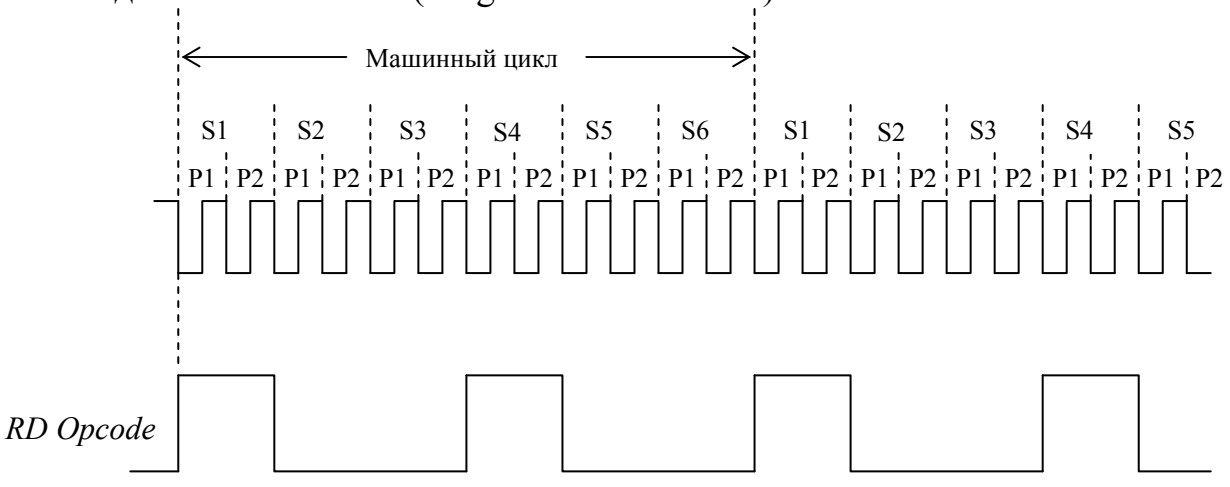


Рис. 2.4 Машинный цикл микроконтроллера семейства MCS-51

Время выполнения команд МК зависит как от длины команды (1, 2 или 3 байта), так и от ее типа. Большинство команд МК выполняется за один или два машинных цикла. Исключение составляют однобайтные команды умножения *MUL AB* и деления *DIV AB*, время выполнения которых составляет 4 машинных цикла.

2.2. Организация памяти МК SAB 80C515

Гарвардская архитектура микроконтроллерных систем на основе МК SAB 80C515 предполагает использование физически и логически разделенных устройств памяти программ и памяти данных. Полное пространство памяти МК SAB показано на рис. 2.5. Оно объединяет память программ и память данных.

Память программ объемом до 64 Кбайт доступна только по чтению. В МК SAB80C515 младшие 8 Кбайт программной памяти IROM размещаются на кристалле МК. В расширенной конфигурации МК память программ может быть представлена либо только внешней памятью XROM, либо в виде внешней (XROM) и внутренней (IROM) памяти. Как правило, элементы программной памяти адресуются с помощью 16-разрядного счетчика команд PC, но в некоторых командах, например в командах чтения констант типа *movc A,@A+dptr,* могут косвенно адресоваться с использованием 16-разрядного указателя DPTR.

Доступ к внешней памяти программ (ВПП) разрешен, если активизирован вход отключения внутреннего ПЗУ EA (EA = 0), либо, если содержимое счетчика команд PC превышает значение 1FFFh, т.е. при обращении к программной памяти вне первых 8 Кбайт. Пользователь может разрешить или запретить обращение к IROM, управляя сигналом EA на одноименном входе МК. При активном сигнале EA (EA = 0) все выборки команд, начиная с нулевого адреса, будут происходить из внешней памяти программ.

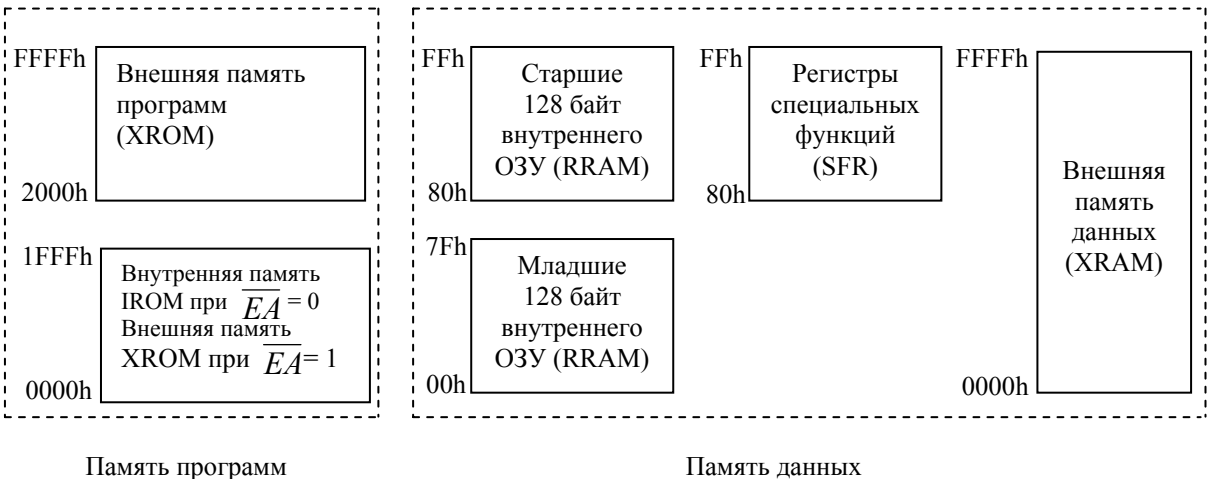


Рис. 2.5. Полное пространство памяти МК SAB 80C515

В МК семейства MCS-51 отсутствуют специальные выводы сигналов адреса и данных, предназначенные для подключения внешней памяти. Для организации шин адреса и данных внешней памяти программ или данных используют линии многофункциональных параллельных портов ввода/вывода. В МК семейства MCS-51 в качестве внешних шин адреса и данных используются линии портов P0 и P2, при этом с помощью линий порта P0 реализуется мультиплексированная внешняя шина адрес/данные.

16-разрядный адрес внешней памяти формируется на линиях порта P0 (младший байт) и порта P2 (старший байт). Для демультиплексирования адреса на выходах порта P0 используется выходной сигнал ALE (Address Latch Enable). С его помощью младший байт адреса, формируемый на выходе порта P0, фиксируется во внешнем регистре адреса. При обращении к внешней памяти программ чтение ее содержимого осуществляется с помощью сигнала PSEN (Program Store ENable). Упрощенная функциональная схема включения МК SAB 80C515 и внешней памяти программ показана на рис. 2.6.

Модель МК SAB 80C535 в отличие от МК SAB 80C515 не содержит внутренней памяти программ IROM и может использоваться только с внешней памятью XROM.

Память данных МК SAB 80C515 может объединять внутреннюю IRAM и внешнюю XRAM области. В отличие от памяти программ, пространства памяти RRAM и XRAM являются независимыми: внешняя память XRAM объемом до 64 Кбайт используется как дополнительная к внутренней памяти IRAM.

Внутренняя память данных IRAM, размещенная на кристалле МК, представлена двумя физически разделенными блоками – регистровым файлом RRAM емкостью 256 байт и блоком регистров специальных функций SFR емкостью 128 байт (рис. 2.7). Логически регистровый файл RRAM удобно разделить на две области - младшие 128 байт с адресами 00 – 7Fh и старшие 128 байт с адресами 80h – FFh.

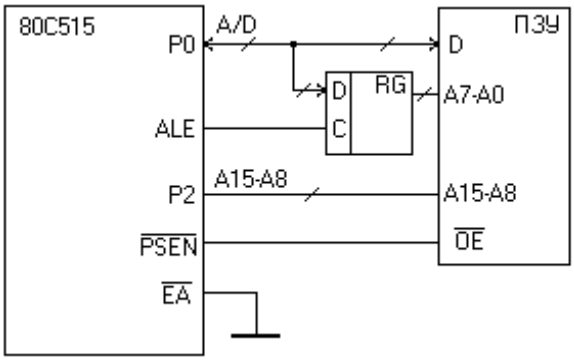
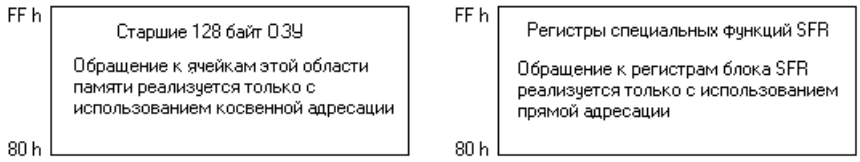


Рис.2.6. Схема включения МК SAB 80C515 и внешней памяти программ

Младшие 128 байт внутренней памяти данных в свою очередь подразделяются на три области, обладающие специальными свойствами. Первые 32 байта внутреннего памяти RRAM с адресами 00 – 1Fh сгруппированы в четыре банка по 8 регистров общего назначения с именами R0 – R7 в каждом. Отличительной особенностью блока рабочих регистров является возможность использования, наряду с прямой и косвенной адресацией байт, прямой регистровой адресации. Команды с прямой регистровой адресацией регистров R0 – R7 короче команд с прямой адресацией.



80 h 30 h	80 байт ОЗУ								Область ОЗУ, допускающая прямую и косвенную адресацию байт,
2F h	7F	7E	7D	7C	7B	7A	79	78	Область ОЗУ, допускающая прямую и косвенную адресацию байт, а также прямую адресацию отдельных бит. Адрес бита указан внутри квадрата, соответствующего выделенному биту
2E h	77	76	75	74	73	72	71	70	
.									
.	.	.	.	.	.	.	.	.	
.									
22 h	17	16	15	14	13	12	11	10	
21 h	0F	0E	0D	0C	0B	0A	09	08	
20 h	07	06	05	04	03	02	01	00	
1F h 18 h						Банк 3			Область ОЗУ, допускающая использование прямой, косвенной и регистровой адресации при обращении к ячейкам памяти
17 h 10 h						Банк 2			
0F h 08 h						Банк 1			
07 h		R7				Банк 0			
		R6							
		R5							
		R4							
		R3							
		R2							
		R1							
00 h		R0							

Рис. 2.7. Карта адресуемых байт и адресуемых бит внутренней памяти данных

Выбор рабочего (текущего) банка регистров осуществляется программно с помощью бит RS0 и RS1 в слове состояния PSW. Наличие четырех банков рабочих регистров обеспечивает быстрое переключение контекста программы при использовании ограниченного числа подпрограмм или обработчиков внешних прерываний. При инициализации (по сбросу и при включении питания) CPU настраивается на работу с банком регистров 0. 16 ячеек внутренней памяти данных с адресами 20h – 2Fh образуют область, зарезервированную для хранения 128 программно-управляемых пользовательских флагов (битовых данных). Каждый из указанных флагов имеет прямой 8-битовый адрес в диапазоне 00 – 7Fh. Адреса битовых данных указаны на рис. 2.7. В командах побитовой обработки используется только прямая адресация бит. Если область битовых данных в программе не используется, то при необходимости указанную область пользовательских флагов можно использовать как область ОЗУ, допускающую прямую и косвенную адресации 8-битных ячеек памяти. Оставшиеся 80 байт из области младших 128 байт RRAM и вся область старших 128 байт – это область обычной оперативной памяти, для адресации ячеек которой допускается использование косвенной адресации всех данных и прямой адресации 80 байт из области младших 128 байт.

Блок регистров специальных функций

Регистры блока SFR имеют такие же адреса 80h – FFh, как и старшие 128 байт RRAM. Поэтому при обращении к регистрам этих областей используют разные способы адресации. Доступ к старшим 128 байтам RRAM возможен только с использованием косвенной адресации, а при обращении к регистрам SFR используется только прямая адресация.

Регистры специальных функций SFR – это набор регистров, в основном используемых для управления работой и отображения состояния периферийных устройств, размещенных на кристалле МК. В состав любого периферийного устройства, будь то порт ввода-вывода, таймер, контроллер прерывания и пр., входит сравнительно большое число регистров. Программно доступные регистры периферийных устройств как раз и являются регистрами блока SFR. С помощью управляющих бит этих регистров осуществляется выбор режима работы периферийного устройства и индикация его состояния. Программируя регистры блока SFR, пользователь имеет возможность настроить внутреннюю периферию МК на выполнение требуемых функций. Состав и назначение регистров SFR приводится при рассмотрении конкретных устройств МК. В прил.2 для справки представлены все регистры SFR МК SAB 80C515.

В заключение отметим, что все обращения к ячейкам внутренней памяти IRAM (RRAM и SFR) всегда осуществляются с использованием короткого 8-разрядного адреса.

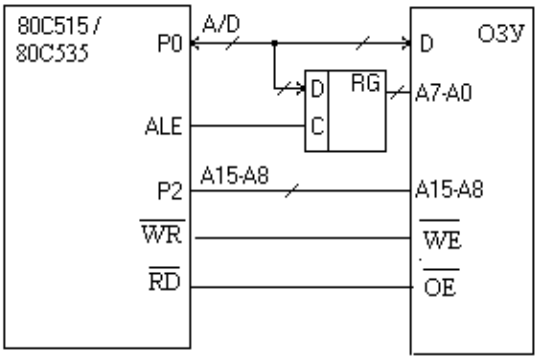


Рис. 2.8. Функциональная схема включения МК SAB 80C515/ 80C535 и внешней памяти данных

Внешняя память данных. МК SAB 80C515/535 допускают расширение памяти данных путем подключения внешних БИС оперативной памяти емкостью до 64 Кбайт. Для доступа к внешней памяти данных блок управления МК формирует управляющие сигналы, запись WR и чтение RD и использует рассмотренный выше механизм формирования внешних шин адреса и данных с помощью линий портов P0 и P2. Обращение к внешней памяти данных выполняется командами с мнемоникой *movx*. При обращении к данным внутренней памяти сигналы RD и WR не формируются.

Команды пересылок данных при обращениях к внешней памяти рассматриваются в п.2.3.

Схема включения внешней памяти данных в системе с МК SAB 80C515/ 80C535показана на рис. 2.8.

2.3. Система команд МК SAB 80C515 полностью совпадает с системой команд МК семейства MCS-51 фирмы Intel и включает в себя 111 кодов команд. Синтаксис большинства ассемблерных команд состоит из мнемонического обозначения функции команды, вслед за которым могут размещаться операнды, указывающие методы адресации и типы данных. Машинные коды команд МК представляются одним, двумя или тремя байтами. Команды выполняются за один или два машинных цикла. Исключение составляют команды умножения и деления, длительность исполнения которых 4 машинных цикла. В качестве операндов могут использоваться биты, 4-битные слова (тетрады), байты и 16-битные слова, при этом большинство операндов команд являются байтами.

Для адресации операндов-источников используют следующие типы адресации: неявную, регистровую, прямую, непосредственную, косвенную и разновидность косвенной – базово-индексную. При обращении к приемникам непосредственную и базово-индексную адресации не используют.

Обозначение типов данных и способов адресации в мнемониках команд МК семейства MCS-51 приводится в прил.1.

Систему команд МК SAB 80C515 по функциональному признаку удобно подразделить на нескольких групп:

- команды пересылок;
- команды арифметических и логических операций с байтовыми переменными;
- команды битового процессора;
- команды передачи управления.

Группа команд пересылок (передачи данных). В число команд этой группы включены команды с мнемониками *mov, movc, movx, push, pop* и команды обмена *swap, xchd, xch*. В системе команд отсутствуют специальные команды ввода/вывода. Ввод/вывод реализуют обычными пересылками при обращениях к регистрам портов ввода/вывода в составе блока SFR. Команды пересылок не модифицируют флаги результата.

15 команд пересылок с мнемоникой *mov d, s* обеспечивают передачу байта данных из источника S (source) в приемник D (destination). Источником S может выступать аккумулятор A, регистр Rn блока рабочих регистров, прямо адресуемый операнд из области младших 128 байт RRAM или из области SFR, косвенно адресуемый по содержимому регистра Ri (i=0,1) операнд RRAM, непосредственный операнд #data. Перечисленные операнды, за исключением непосредственных данных #data, могут указываться в качестве приемника D. Формат команд этой группы зависит от типа операнда и используемых способов адресации и может составлять от 1 до 3 байт.

В систему команд входит одна команда пересылки 16-битных данных - команда загрузки регистра DPTR непосредственными данными.

Пересылки данных при обращениях к внешней памяти реализуются только через аккумулятор. Мнемоники команд пересылок данных при обращениях к внешней памяти программ и к внешней памяти данных различны. С помощью команд *movc* выполняются пересылки из внешней программной памяти в аккумулятор. Для адресации операнда во внешней памяти программ используется базово-индексная адресация. Адрес выбираемого байта вычисляется путем суммирования 8-битного беззнакового содержимого ACC (индекса) с содержимым 16-битного регистра (базы), которым может быть либо DPTR, либо PC.

Для организации обменов между внешней памятью данных и аккумулятором используются команды с мнемоникой *movx*. Операнд во внешней памяти данных может адресоваться только косвенно через регистр DPTR или через регистр Ri (i=0,1). Команды *movx @Ri,A* и *movx A,@Ri* (*Ri* = 0,1) выполняют пересылки данных между аккумулятором и внешней памятью, организованной в виде 256-байтных страниц. В указанных командах содержимое регистра Ri (i=0,1) является 8-битным

адресом, мультиплексированным с данными на линиях порта P0. Выбор конкретной страницы реализуется с помощью выходных линий любого порта ввода/вывода, кроме порта P0, соединенных с входами «выбор кристалла» CS соответствующих СБИС памяти данных. Управление линиями выбора внешних страниц должно выполняться командой, стоящей перед *movx*.

Команды *movx @DPTR,A* и *movx A,@DPTR* обеспечивают непосредственное обращение к операндам всего 64-Кбайтного адресного пространства внешней памяти данных с использование косвенной адресации через 16-битный регистр DPTR. Порт P2 выводит старшие 8 бит адреса (содержимое DPH), а порт P0 мультиплексирует младшие 8 бит адреса (DPL) с данными. С помощью указанных команд обеспечивается более быстрое обращение к ячейкам внешней памяти данных, так как в этом случае не требуются дополнительные команды для переключения старших линий адреса.

Команды загрузки в стек *push direct* и извлечения из стека *pop direct* загружают/извлекают из стека содержимое прямо адресуемой ячейки RRAM.

В группу команд пересылок также входят команды обмена содержимого аккумулятора с содержимым регистра Rn или ячейкой внутренней памяти RRAM (команды XCH) и команды межтетрадного обмена (XCHD и SWAP). Команда обмена тетрадами внутри аккумулятора может рассматриваться как команда четырехбитного циклического сдвига. Коды команд пересылок и выполняемые ими функции приведены в прил.1.

Группа команд арифметических и логических операций. Указанная группа команд является наиболее многочисленной и фактически определяет вычислительные возможности CPU. Система команд включает команды выполнения простейших арифметических операций (сложения *add*, сложения с переносом *addc*, вычитания с заемом *subb*, инкремента *inc*, декремента *dec)* и более сложных операций умножения *mul ab* и деления *div ab*. Команды логических операций реализуют основные логические функции *И (anl)*, *ИЛИ (orl)*, *ИСКЛЮЧАЮЩЕЕ ИЛИ (xrl)*, инверсии аккумулятора. В эту же группу команд входят команды циклического сдвига влево и вправо на один разряд.

Практически все команды арифметических и логических операций выполняются в арифметико-логическом блоке CPU. Комбинационное АЛУ реализует различные действия над целыми числами без знака. При выполнении большинства арифметических и логических команд аккумулятор является источником одного из операндов и приемником результата. Вторым операндом может быть регистр Rn (n=0-7) или ячейка внутренней памяти RRAM. Многие команды этой группы в отличие от команд других групп модифицируют признаки результата в регистре PSW.

Выполнение операций сложения и вычитания чисел со знаком, как и одноименных операций с беззнаковыми числами, выполняется в АЛУ. Однако правильность их выполнения в этом случае необходимо программно контролировать с помощью флага переполнения OV. Установленный флаг OV указывает на ошибочный результат операции со знаковыми числами.

С помощью команды *add* и дополнительной команды коррекции *da a* реализуются операции сложения двоично-десятичных чисел в формате BCD (binary-coded-decimal). Алгоритм указанной операции выполняется в два этапа. Сначала операнды в упакованном двоично-десятичном формате складываются как двоичные числа, а затем с помощью команды *da a* производится коррекция в общем случае неправильного результата сложения двоично-десятичных чисел. Действие команды *da a* следующее. Восьмибитное число результата в аккумуляторе рассматривается как две четырехбитные двоично-десятичные цифры. Коррекция результата суммирования выполняется по правилам:

1. Если значение младших четырех бит больше 9 или признак межтетрадного переноса AC равен 1, то к содержимому аккумулятора прибавляется число 06h, образуя правильное BCD-число в младшей тетраде.

2. Если значение старших четырех бит больше 9 или установлен флаг переноса C равен 1, то к содержимому аккумулятора прибавляется число 60h, образуя правильное BCD-число в старшей тетраде.

Перенос, фиксируемый при сложении двух BCD-чисел или при выполнении команды *da a*, указывает, что сумма исходных BCD-чисел больше, чем 100.

В общем случае команда *da a* выполняет двоичное преобразование содержимого аккумулятора в зависимости от значения флага AC в регистре PSW и не может просто преобразовать двоичное число в аккумуляторе в BCD-формат. Команда коррекции двоично-десятичного вычитания отсутствует..

В системе команд МК MCS-51 нет команды обычного вычитания с мнемоникой *sub*. Для выполнения обычного вычитания командой *subb A,*R_n предварительно необходимо обнулить флаг *C*.

С помощью команд *addc* и *subb* на 8-разрядном АЛУ реализуются операции сложения/вычитания с многократной точностью. В частности, используя команды *addc* и *da a* можно выполнять многоразрядные десятичные сложения.

В число команд арифметических операций входят команды инкремента и декремента операнда, указанного в команде.

Операции умножения и деления реализуются встроенным аппаратным блоком умножения/деления с использованием аккумулятора A и регистра B. В команде умножения *mul ab* содержимое A умножается на содержимое

регистра B. 16-битный результат размещается в регистре B (старший байт) и в аккумуляторе (младший байт). При выполнении команды деления *div ab* реализуется деление 8-битового беззнакового числа в аккумуляторе на 8-битовое беззнаковое число в регистре B. Результат операции заноситься в аккумулятор (целая часть частного) и в регистр B (остаток).

В командах циклического сдвига влево (вправо) на один разряд *rl a, rlc a, rr a, rrc a* операндом команд является содержимое аккумулятора. Результат операции сдвига помещается в аккумулятор. Тип команды сдвига определяется тем, что помещается в освобождающийся при сдвиге бит и что заносится во флаг переноса C. В системе команд МК отсутствуют команды арифметического и логического сдвига, однако при необходимости названные операции можно выполнить программным способом, используя имеющиеся команды циклического сдвига.

Отличительной особенностью МК является отсутствие в его системе команд обычной команды сравнения, а также флага нулевого результата Z. Однако это не ограничивает возможности CPU МК по сравнению со стандартными МП. В системе команд МК имеется группа команд с мнемоникой *cjne*, которые осуществляют сравнение двух операндов и при их несовпадения выполняют переход по адресу, указываемому в команде. С помощью команд *cjne* не просто реализуются операция сравнения с формированием признака результата сравнения, но и одновременно выполняется принятие решения (переход) в соответствии с результатом.

Отсутствие программно доступного флага Z (см. комментарии к рис.2.3) компенсируется наличием команд *jz/jnz rel*, с помощью которых результат выполнения любой операции, фиксируемый в аккумуляторе, может быть проверен на нуль, и в соответствии с результатом проверки выполнен переход на соответствующую ветвь продолжения программы.

Полный список команд арифметических и логических операций приведен в прил.1.

Команды битового процессора. Битовый процессор, являющийся частью архитектуры МК, оперирует битовыми операндами и выполняет собственный набор команд. Битовые операнды размещаются в ячейках памяти с адресами 20h – 2Fh RRAM и регистрах блока SFR, адреса которых кратны 8. Для адресации битовых операндов используется только прямая адресация.

Каждый из адресуемых битов может быть установлен в 1, сброшен в 0, инвертирован и проверен. Между любым прямоадресуемым битом и флагом переноса C могут быть произведены логические операции И, ИЛИ с запоминанием результата во флаге C. Любой бит может быть перезаписан в/из флага C. Команда условной передачи управления битового процессора реализуют переходы, если проверяемый бит установлен (со сбросом или без сброса этого бита) или если проверяемый бит не установлен. Команды битового процессора представлены в прил.1.

Команды ветвления и передачи управления *объединяют команды безусловной и условной передачи управления.*

Команды безусловной передачи управления имеют три модификации: *ljmp*, *ajmp* и *sjmp*.

Команда *ljmp* выполняет длинные безусловные переходы в пределах всего 64-Кбайтного адресного пространства памяти программ. В 3-байтной команде *ljmp* содержится полный 16-битный адрес перехода. Однако на практике переходы в пределах всего адресного пространства реализуются редко и чаще используют укороченные команды перехода *ajmp* и *sjmp*, занимающие меньше места в памяти. Команда *ajmp addr11* абсолютного перехода (рис.2.9) обеспечивает безусловную передачу управления в пределах одной страницы памяти программ размером 2 Кбайт.

При выполнении команды *ajmp* содержимое счетчика PC сначала увеличивается на 2, формируя адрес следующей команды, после чего 11 младших бит PC заменяются адресом абсолютного перехода addr11.

Двухбайтная команда *sjmp rel* короткого безусловного перехода выполняет короткий переход по адресу, вычисляемому в результате сложения 8-битного смещения со знаком rel из второго байта команды с содержимым PC. Содержимое PC предварительно инкрементируется дважды для указания адреса следующей по порядку команды. Диапазон назначений команды *sjmp* лежит в пределах от 128 байт, предшествующих команде *sjmp*, до 127 байт, следующих за ней.

В системе команд МК имеется команда безусловного косвенного перехода *jmp @a+dptr*, позволяющая передавать управление по косвенному адресу. Эта команда удобна тем, что предоставляет возможность организации множественных ветвлений по адресу, вычисляемому самой программой и неизвестному при написании исходного текста программы.

Кроме команд безусловной передачи управления, система команд МК содержит команды безусловного вызова подпрограмм *lcall, acall* и команды безусловного возврата *ret* и возврата из прерывания *reti*.

Команды условной передачи управления выполняют переход только в случае выполнения условия, указанного в коде команды. Если условие не выполнено, передача управления не происходит, и выполняется следующая по порядку команда. Все команды условной передачи управления реализуют только короткие относительные переходы, позволяющие передать управление в пределах ±128 байт относительно адреса команды, следующей по порядку за командой условного перехода.

Рис.2.9.Формат команды *ajmp addr11*

Развитая система условных переходов предоставляет возможность осуществлять ветвления по следующим условиям:

– значение результата в *Акк* равно/не равно нулю (команды *jz/jnz*);

- адресуемый операнд не равен задаваемому в команде (команды *cjne*);

– перенос *C* равен/не равен нулю (команды *jc/jnc*);

– адресуемый бит равен/не равен нулю (команды *jb/jnb*).

Команды условных и безусловных переходов приведены в прил.1.

2.4. Встроенные средства ввода-вывода дискретных сигналов МК SAB 80C515

2.4.1. Ввод-вывод параллельной информации

МК SAB 80C515 содержит в своем составе шесть двунаправленных портов ввода-вывода параллельной информации P0 – P5, предназначенных для обмена информацией с внешними устройствами. При соответствующем управлении двунаправленный порт обеспечивает как ввод (передачу информации от внешнего устройства в МК), так и вывод (передачу информации из МК во внешнее устройство). Программно доступными регистрами портов ввода-вывода параллельной информации являются одноименные регистры P0-P5 блока SFR. Особенностью портов ввода-вывода P0-P5 является то, что для них разрешена побитовая обработка содержимого отдельных разрядов порта. Обращение к отдельным разрядам порта выполняется командами побитовой обработки *mov bit*,*C*, *cl*r *bit*, *setb bit*„ где операнд *bit* x – адрес разряда порта $P_{x.y}$, x - номер порта (x = 0-5), y – номер разряда порта (y = 0 -7).

Для обмена дискретными сигналами «включено-выключено» используются отдельные линии ввода-вывода параллельных портов.

Большинство портов МК SAB 80C515 (порты P1 – P5) являются квазидвунаправленными, т.е. портами с программно заданной и неизменяемой во время работы функцией ввода-вывода. Порт P0, обеспечивающий в силу специфики при работе с внешней памятью динамическое переключение функций ввода-вывода, является действительно двунаправленным портом. Порт P0 реализует как прием данных из внешней памяти в МК, так и передачу данных из МК во внешнюю память

Двунаправленный и квазидвунаправленный порты отличаются схемотехникой формирователя выходных сигналов порта.

Выходной каскад квазидвунаправленного порта (рис. 2.10) образован транзисторами T1, T2, T. Транзистор T2, включенный в режиме источника тока, используется в качестве резистора, с помощью которого выводы порта подключаются к питанию U_{CC}. Слаботочный транзистор T2 не может обеспечить быстрого переключения сигнала на выводе порта. Для уменьшения времени переключения (при переходе сигнала на выводе

порта из состояния 1 в состояние 0 или из состояния 0 в состояние 1)

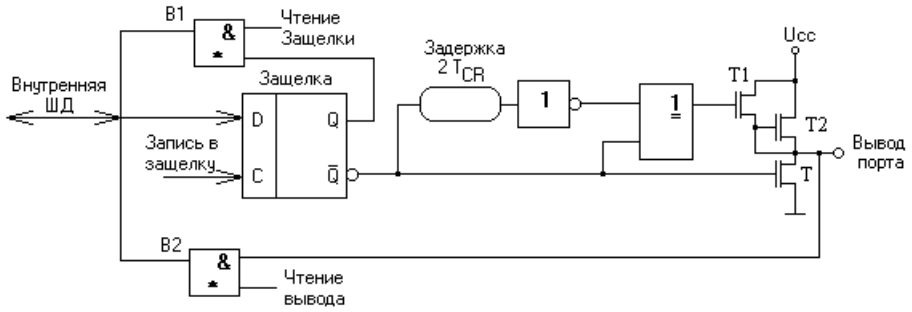


Рис. 2.10. Базовая структура разряда квазидвунаправленного порта

используется дополнительный сильноточный транзистор T1,переключающийся значительно быстрее слаботочного транзистора T2. Транзистор T1 с помощью элемента “Задержка” включается на время, равное двум периодам тактовой частоты $T_{CR.}$ В открытом состоянии ток транзистора T1 ток приблизительно в 100 раз больше, чем ток постоянно открытого транзистора T2.

Выходные каскады портов МК SAB 80C515, за исключением порта P0, реализованы по рассмотренной схеме.

Каждый вывод порта может независимо от других использоваться как вход или как выход. При *выводе* данные с внутренней шины записываются в триггер-защелку порта и из него поступают на внешние выводы порта. Для использования квазидвунаправленного порта для ввода данных необходимо, чтобы выходной транзистор T был закрыт, т.е. защелка должна находиться в состоянии "1". В режиме ввода входные данные через вентиль B2 с помощью внутреннего сигнала «чтение вывода» считываются на внутреннюю шину МК и далее обрабатываются операционным блоком.

Доступ к регистрам портов *Pn* (n=0-5) и контактам портов отличается при записи и чтении. Запись в порт всегда означает запись в регистр и вывод на контакты порта (*mov P0,A; clr* P3.2). При чтении из порта осуществляется чтение контакта (*mov A,P0; mov C,P0.2*). Выход триггера Q через вентиль B1 может быть подключен к внутренней шине с помощью внутреннего сигнала «чтение защелки». Благодаря такой организации разряда порта в режиме «чтение-модификация-запись» обеспечивается программное чтение содержимого регистра и возможность побитовой обработки выходных данных порта с последующей записью результата обратно в регистр порта и вывод на контакт порта. Побитовая обработка выходных данных порта реализуется при выполнении команд логических операций (*anl, orl, xrl)* с прямой адресацией порта. В режиме ввода также могут выполняться операции побитовой обработки. Например, с помощью

команды *jb Px.y,addr* можно проанализировать состояние входной линии *Px.y* и выбрать соответствующую ветвь алгоритма программы.

Действительно двунаправленный порт P0 отличается от рассмотренных. В нем обеспечивается возможность динамического переключения режимов ввода и вывода данных. Порт P0 может работать либо в режиме стандартного ввода/вывода, либо в режиме мультиплексируемой шины адрес/данные. При обращении к внешней памяти на выводах порта P0 осуществляется мультиплексирование во времени передаваемого младшего байта адреса (A7 – A0) и передаваемого или принимаемого байта данных (D7-D0: сначала выводится младший байт адреса памяти (A7 – A0), а затем выдается или принимается байт данных (D7-D0). Выходной каскад порта P0 имеет структуру (рис. 2.11) отличную от рассмотренной структуры квазидвунаправленного порта. Он реализован на транзисторе T с открытым стоком, к которому в режиме мультиплексируемой шины адрес/данные подключается транзистор T1. Включение (открытие) транзистора T1 осуществляется только при обращениях к внешней памяти, когда через порт выводится «1». В остальных случаях транзистор T1 отключен.

Выбор режима работы порта P0 определяется сигналом УПР. При значении УПР=0 порт P0 переводится в режим стандартного ввода/вывода. В этом режиме транзистор T1 всегда закрыт, а состояние транзистора T определяется уровнем сигнала на выходе $\overline{Q}$ триггера. Для правильной работы в режиме стандартного ввода/вывода выходной контакт порта необходимо подсоединить к источнику $+U_{CC}$ через внешний резистор.

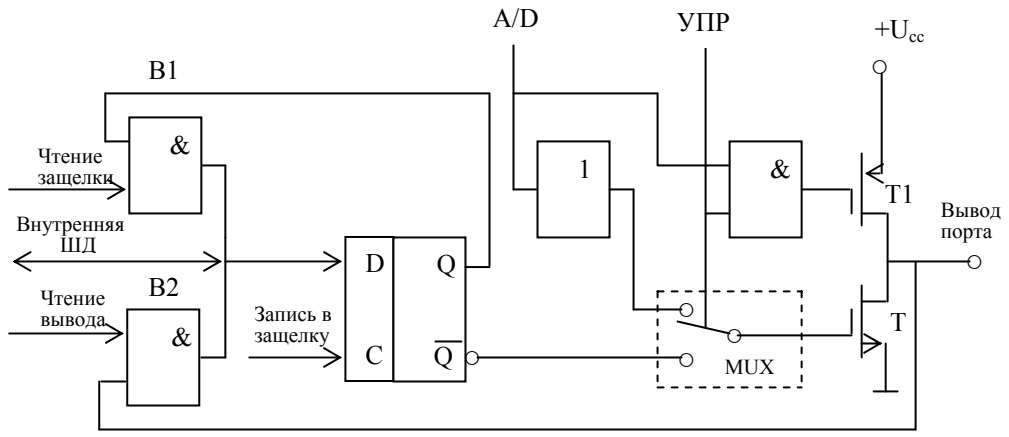


Рис. 2.11. Структура разряда двунаправленного порта P0

При выводе сигнал на выходном контакте порта определяется состоянием транзистора T. При $\overline{Q}$=1 транзистор T открыт. Это состояние соответствует сигналу логического «0». При $\overline{Q}$=0 транзистор T закрыт, и на выходе порта формируется сигнал логической «1».

Для реализации ввода выходной контакт порта P0 должен находиться в состоянии высокого сопротивления, т.е. в состоянии, когда транзисторы T и T1 закрыты. Поскольку транзистор T1 в режиме стандартного ввода/вывода всегда закрыт, для перевода порта P0 в режим ввода достаточно закрыть транзистор T, что обеспечивается при записи «1» в триггер-защелку.

При значении управляющего сигнала УПР = 1 триггер-защелка отключается от выходного каскада порта, и выводы порта P0 используются в качестве внешней мультиплексируемой шины адрес/данные при обращении к внешней памяти. В этом режиме состояние транзисторов T1 и T выходного каскада порта определяется значением внутреннего сигнала адрес/данные, формируемым внутренними схемами блока CPU.

При выводе (передаче младшего байта адреса или данных) выходной каскад в зависимости от состояния сигнала A/D обычным образом формирует на выводе порта значение «0» или «1». При вводе данных внутренние схемы МК устанавливают единичное значение сигнала на входе A/D, при этом транзистор T закрывается (транзистор T1, как отмечено выше, закрыт всегда), и порт работает на ввод, принимая считываемые данные из внешней памяти и передавая их через вентиль B2 на внутреннюю шину МК.

В МК SAB 80C515 многофункциональными являются порты P0, P1, P2 и P3. Каждый вывод этих портов может использоваться либо для выполнения стандартного ввода-вывода, либо для выдачи/приема специальных функциональных сигналов, поддерживающих работу внутренних периферийных устройств или управляющих внешними устройствами (табл.1). Если в конкретном приложении какой-либо специальный сигнал не используется, то соответствующий вывод может быть использован для стандартного ввода-вывода.

Порт P0 и *порт P2*, наряду со стандартным вводом/выводом, при работе с внешней памятью используются для приема/выдачи сигналов внешней мультиплексируемой шины адреса/данных. Для демультиплексирования сигналов адрес/данные на выходе порта P0 устанавливается внешний буферный регистр, фиксирующий младший байт адреса с помощью выходного сигнала ALE (см. п. 2.2). Квазидвунаправленный порт P2 при доступе к внешней памяти используется в качестве формирователя старших разрядов адреса A15-A8.

При программировании памяти программ IROM с помощью порта P0 задаются данные и читается ее содержимое, при этом с помощью порта P1 задается младший байт адреса.

Линии *порта P1* и *порта* P3, наряду с функциями стандартного ввода/вывода, используются для ввода/вывода управляющих сигналов при работе внутренних периферийных устройств (табл.1).

Для реализации альтернативной функции вывода порта в соответствующий D-триггер регистра-защелки необходимо записать «1». При этом выходным каскадом порта управляет внутренний сигнал «альтернативная функция выхода», и на выводе порта формируется соответствующий специальный сигнал.

Имя регистра SFR	Имена управляющих бит								Адрес регистра
	7	6	5	4	3	2	1	0	
Порт P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	80h
Порт P1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	90h
	T2	CLKout	T2E1	INT2	INT6	INT5	INT4	INT3	
					CC3	CC2	CC1	CC0	
Порт P2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	A0h
Порт P3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	B0h
	RD	WR	T1	T0	INT1	INT0	TxD	RxD	
Порт P4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0	E8h
Порт P5	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0	F8h

Таблица 1

2.4.2. Ввод-вывод последовательной информации. Встроенный последовательный порт SP МК SAB80C515 реализован в виде универсального синхронно-асинхронного приемо/передатчика УСАПП. Порт SP используется для организации обмена данными в последовательном коде между МК и устройствами ввода-вывода, а также для организации межпроцессорных коммуникаций в мультипроцессорных системах. Порт SP поддерживает стандартный протокол последовательного обмена RS-232C, но не содержит блок управления модемом.

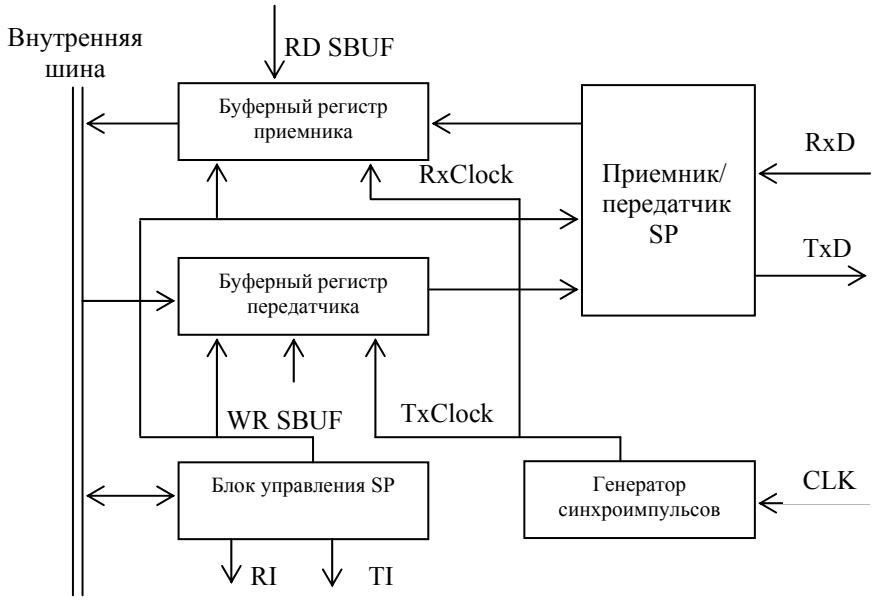


Рис. 2.12. Функциональная схема последовательного порта

В состав порта SP (рис. 2.12) входят буферные регистры приемника и передатчика, собственно приемник/передатчик последовательных данных и блок управления, обеспечивающий работу SP в различных режимах.

В режиме «прием» приемник/передатчик SP осуществляет прием данных в последовательном коде с внешнего вывода P3.0/RxD (Receive Data), формирует параллельный код принятых данных и выполняет запись этого кода в буферный регистр приемника, откуда преобразованные входные данные могут быть считаны процессором. Благодаря такой схеме преобразователя, SP может принимать и преобразовывать следующий байт информации в последовательном коде сразу после загрузки преобразованного байта в выходной буферный регистр приемника, т.е. еще до того, как ранее принятый байт будет считан процессором. Однако если содержимое буферного регистра приемника не будет прочитано до окончания преобразования очередного байта, невостребованные данные в буферном регистре приемника будут потеряны.

Буферный регистр передатчика предназначен для приема параллельной информации с внутренней шины данных и выдачи ее для преобразования в блок приемника/передатчика. В режиме «передача» приемник/передатчик SP обеспечивает прием параллельных данных с выхода буферного регистра передатчика, преобразование их в последовательный код и выдачу его на вывод P3.1/TxD (Transmitted Data). Прием и выдача последовательного кода байта данных начинается с младшего разряда и заканчивается старшим разрядом.

Буферные регистры приемника и передатчика являются независимыми, что позволяет организовывать одновременный прием и передачу данных, другими словами реализовывать дуплексный режим работы последовательного порта. При программном доступе буферные регистры приемника и передатчика имеют одинаковое имя SBUF. При использовании в команде имени SBUF в качестве операнда источника выполняется обращение к буферному регистру приемника. Чтение содержимого SBUF на внутреннюю шину освобождает SBUF для приема следующего байта преобразованных данных.

Генератор синхроимпульсов обеспечивает требуемую скорость приема/передачи последовательных данных. В качестве источников импульсов синхронизации могут использоваться либо генератор тактовых импульсов F_{CR} с выходным делителем, либо импульсы переполнения таймера T/C1. Требуемая скорость приема/передачи устанавливается программно.

Блок управления вырабатывает необходимые сигналы управления другими блоками SP. При завершении приема последовательных данных устанавливается флаг запроса прерывания RI (буфер SBUF заполнен), а при завершении передачи последовательных данных – флаг TI (SBUF пуст

и готов принять новые данные для передачи). Программно доступными регистрами порта SP являются два регистра блока SFR – регистр управления SCON (Serial port CONtrol/status register, адрес 98h) и регистр SBUF (адрес 99h).

Последовательный порт может быть запрограммирован для работы в одном из четырех режимов приема/передачи (одном синхронном и трех асинхронных). Различные режимы работы SP повышают гибкость его использования. Во всех режимах работы SP передача последовательных данных инициируется командой записи в регистр SBUF. Условия приема последовательных данных определяются режимом работы SP. В асинхронных режимах прием инициируется при идентификации старт-бита на входе RxD. В синхронном режиме прием начинается после программного сброса бита RI.

Задание режима работы SP осуществляется программно путем записи управляющего слова в регистр SCON, формат которого и назначение отдельных битовых полей приведены на рис. 2.13. Регистр SCON содержит как управляющую информацию – управляющие биты задания режимов работы SM2, SM1, SM0, бит разрешения приема последовательных данных REN, девятый бит передаваемых данных TB8, так и информацию о состоянии порта - флаги запроса прерывания приемника RI и передатчика TI, девятый принимаемый бит RB8.

9Fh	9Eh	9Dh	9Ch	9Bh	9Ah	99h	98h	SCON
SM0	SM1	SM2	REN	TB8	RB8	TI	RI	Адрес 98h

Бит	Назначение
SM1 SM0	Биты задание режимов работы SP:
00	режим синхронной двухпроводной передачи
01	режим 8-битового УАПП с переменной скоростью передачи
10	режим 9-битового УАПП с фиксированной скоростью передачи
11	режим 9-битового УАПП с переменной скоростью передачи
SM2	Бит разрешение многопроцессорной работы В асинхронных режимах работы при SM2 = 1 флаг RI не активизируется, если принятый бит данных RB8 равен 0 (режимы 2 и 3) или если не принят стоп-бит (режим 1). Бит SM2 не влияет на работу порта SP в синхронном режиме 0.
REN	Бит разрешения приема последовательных данных.
TB8	9-й бит передаваемых данных в режимах 2 и 3.
RB8	9-й бит принимаемых данных в режимах 2 и 3. В режиме 1 при SM2 = 0 RB8 является принятым стоп-битом.
TI	Флаг прерывания передатчика. Бит TI устанавливается аппаратно в конце выдачи 8-го бита в режиме 0 или в начале стоп-бита в других режимах. Сбрасывается программно.
RI	Флаг прерывания приемника. Бит RI устанавливается аппаратно при приеме 8-го бита в режиме 0 или в середине стоп-бита в других режимах. Сбрасывается программно.

Рис. 2.13. Формат регистра SCON и назначение отдельных битовых полей

Синхронный режим приема/передачи (режим 0*)*. В режиме синхронного обмена многофункциональные выводы порта P3 используются для вывода синхросигналов, сопровождающих каждый бит данных синхронной посылки (вывод TxD), и вывода последовательных данных (вывод RxD), принимаемых или передаваемых в полудуплексном режиме. Скорость приема/передачи (частота пересылки одного бита) F_{SP0} в этом режиме максимальна, она постоянна и равна $f_{CR}/12$.

Передача последовательных данных в синхронном режиме инициируется при записи передаваемых данных в SBUF. При передаче 8-го бита данных формируется запрос прерывания передатчика TI и устанавливается одноименный флаг в его регистре SCON.

Прием последовательных данных в синхронном режиме инициируется программно при сбросе флага прерывания приемника RI. После приема 8-го бита данных формируется запрос прерывания приемника RI и устанавливается одноименный флаг в регистре SCON.

Синхронный режим в большинстве случаев применяется при необходимости *расширения адресного пространства параллельных портов ввода/вывода* за счет подключении внешних регистров сдвига. На рис. 2.14 показана упрощенная схема подключения внешних сдвигающих регистров к выводам последовательного порта при его работе в режиме 0.

Для организации дополнительных портов ввода/вывода параллельных данных используются внешние сдвигающие регистры, управляемые сигналами последовательного порта, запрограммированного для работы в синхронном режиме. Обмен данными с внешними регистрами выполняется в полудуплексном режиме

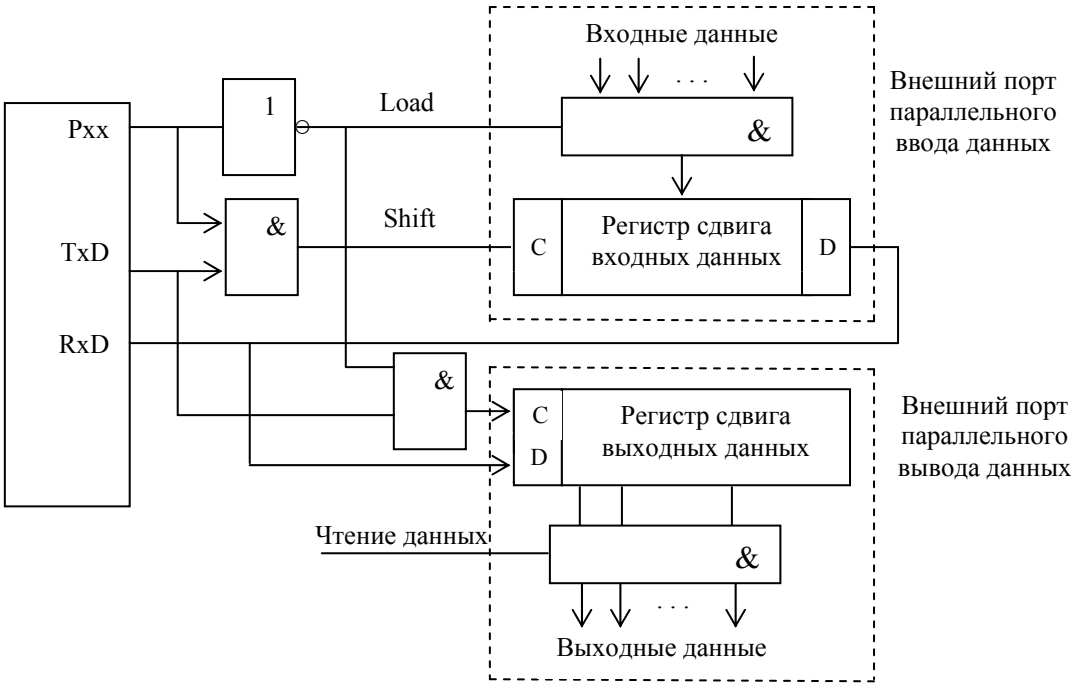


Рис. 2.14. Упрощенная схема подключения внешних сдвигающих регистров для организации дополнительных параллельных портов ввода-вывода

Выбор внешнего регистра ввода или вывода параллельных данных реализуется с помощью сигнала, формируемого на выводе Px.x любого из параллельных портов МК.

При Px.x=0 разрешена работа внешнего порта параллельного вывода данных и осуществляется запись входных данных в регистр сдвига внешнего порта ввода.

При Px.x=1 разрешена работа внешнего порта параллельного ввода данных. При вводе входной параллельный код с помощью синхроимпульсов *Shift* преобразуется в последовательную кодовую посылку, которая поступает на вход R_xD последовательного порта МК. Порт SP, выполняющий обратное преобразование последовательного кода в параллельный, обеспечивает возможность использования внешних параллельных данных внутри МК. На время преобразования при Px.x=1 запись новых входных данных во внешний параллельный порт ввода блокируется.

Вывод параллельных данных осуществляется внешним портом параллельного вывода. Инициализация вывода выполняется при записи байта выходных данных в регистр SBUF порта SP МК. Порт SP выполняет преобразование параллельного кода в последовательный, при этом на линию RxD последовательно выдаются 8 бит данных, которые с помощью внешнего регистра сдвига преобразуются в параллельные выходные данные. Чтение выходных данных внешнего порта разрешается после завершения преобразования, например, по сигналу «чтение данных» при TI = 1.

Режимы асинхронного приема/передачи.

Режимы 1, 2, 3 порта SP – это режимы асинхронного приема/передачи, наиболее часто применяемые на практике. В этих режимах последовательные данные передаются через вывод TxD передатчика, а принимаются с вывода RxD приемника. Асинхронные режимы приема/передачи отличаются друг от друга длиной кодовой посылки и различной скоростью приема/передачи.

Скорость приема/передачи при асинхронных обменах определяется источником импульсов синхронизации и режимом работы SP. Она может быть как переменной, так и постоянной. В качестве источника импульсов синхронизации могут выступать либо тактовый генератор, формирующий синхроимпульсы с частотой F_{CR}, либо импульсы переполнения таймера *T/C1*. Режимы 1 и 3 являются режимами приема/передачи с переменной скоростью, а режим 2 – с постоянной скоростью

Формирователь внутренних синхросигналов *RxClock*, *TxClock* показан на рис. 2.15.

Отличительной особенностью последовательного порта МК SAB 80C515 является наличие в нем подрежима постоянной скорости приема/передачи при работе в режимах 1 и 3. В подрежиме постоянной

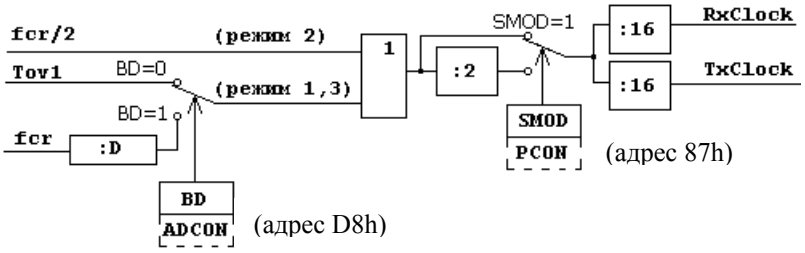
скорости импульсы синхронизации формируются на выходе специального делителя сигналов F_{CR} с коэффициентом пересчета 2500. Выбор источника синхроимпульсов в режимах 1 и 3 программируется с помощью бита *BD* в регистре *ADCON* (см. рис. 2.21).

Еще одной особенностью последовательного порта МК SAB 80C515 является то, что в нем скорость приема/передачи асинхронных режимах может быть изменена в два раза с помощью внутреннего делителя на 2. Включение делителя на 2 в цепочку формирования синхросигналов *RxClock, TxClock* программируется с помощью бита *SMOD* в регистре *PCON* (см. рис. 2.47).

При *BD* = 1 скорость передачи последовательного порта фиксирована: при f_{CR} = 12 МГц и *SMOD* = 0 скорость передачи $f_{SP1,3}$ = 4800 бит/сек, а при *SMOD* = 1 скорость передачи $f_{SP1,3}$ = 9600 бит/сек.

При *BD* = 0 порт SP имеет переменную скорость передачи

Независимо от источника импульсов синхронизации частота внутренних синхросигналов приемника *RxClock* и передатчика *TxClock* делится на 16.



Здесь и далее пунктиром выделены имена регистров SFR, в которых размещаются соответствующие биты управления

Рис. 2.15. Формирователь синхроимпульсов RxClock, TxClock в режимах 1, 2, 3.

Передача последовательных данных во всех режимах работы SP инициируется командой записи в регистр SBUF, а прием - при обнаружении старт-бита на входе RxD приемника. По завершению приема байта данных устанавливается флаг запроса прерывания приемника RI (буфер приемника SBUF заполнен). При завершении передачи байта данных устанавливается флаг запроса прерывания передатчика TI (буфер передатчика SBUF пуст и готов принять данные для передачи новых данных).

Режим 1 является стандартным 8-битовым асинхронным режимом, который, как правило, используется для обычных каналов последовательной связи, например, для связи МК с компьютером верхнего уровня. В режиме 1 последовательная посылка представляется 10 битами: старт-бит (сигнал нулевого уровня), 8 бит данных и стоп-бит (сигнал единичного уровня).

Режимы 2 и 3 используются при повышенных требованиях к достоверности приема последовательных данных, а также для организации

межпроцессорных коммуникаций в распределенных системах микропроцессорного управления. В режимах 2 и 3 длина кодовой последовательности увеличена до 11 бит за счет включения в нее дополнительного 9-го бита данных. Назначение и использование этого бита зависит от режима работы порта SP (прием или передача). При передаче 9-й бит данных *TB8* является программируемым. Например, в качестве бита TB8 может использоваться бит паритета из регистра PSW или специально программируемый бит признака данных в мультипроцессорных системах. При приеме 9-й бит данных, содержащий некоторую дополнительную информацию о принимаемых данных, заносится в разряд *RB8* регистра *SCON* и его значение может быть программно проанализировано.

В общем случае требуемая скорость приема/передачи в режимах 1, 2 и 3 устанавливается программно при задании соответствующих значений бит *BD* и *SMOD*. Частота синхроимпульсов *RxClock*, *TxClock* в этих режимах определяется выражениями:

$$f_2 = \frac{2^{SMOD}}{64} \cdot f_{CR}, \text{ (режим 2)}$$

$$f_{1,3} = \frac{2^{SMOD}}{32} \cdot f_{OV1} \qquad \text{или} \qquad f_{1,3} = \frac{2^{SMOD}}{2500} \cdot f_{CR}, \text{ (режим 1, 3)}$$

Программирование бита *BD* удобно выполнять с помощью команды пересылки с битовой адресацией, например *mov 0DFh, C* с предварительной инициализацией флага C командой *clr C* или командами *clr C* и *cpl C*.

Поскольку к битам регистра *PCON*, содержащего бит *SMOD*, побитовое обращение запрещено, программирование бита *SMOD* можно реализовать следующей последовательностью команд:

```
mov A,87h       ;
anl A,#7Fh      ;
orl A,#data     ; здесь #data = 00h или #data = 80h
mov 87h,A
```

Для обнуления бит *BD* и *SMOD* удобно использовать команды:

```
anl D8h,#7Fh
anl 87h,#7Fh
```

При использовании импульсов T_{OV1} в качестве источника синхросигналов *RxClock* и *TxClock* для таймера *T/C1* обычно назначается режим автозагрузки (режим 2 таймера). В этом случае скорость последовательного обмена определяется по формуле:

$$f_{1,3} = \frac{2^{SMOD}}{32 \cdot (256 - TH1)} \cdot \frac{f_{CR}}{12}$$, где TH1 – десятичный код числа в регистре TH1, $TH1 = 256 - 10^6 / 32\, f_{1,3}$

В табл.2 приведен ряд стандартных скоростей последовательного обмена и соответствующие им значения TH1 для режима автозагрузки таймера T/C1. В принципе, загружаемое значение TH1 может быть любым. В частности может потребоваться установка TH1 специальным значением, которое подбирается при настройке. Программирование таймера T/C1 рассматривается в п.2.6. Если необходим последовательный обмен с очень низкой скоростью, можно задать режим 16-разрядного счетчика таймера 1 (режим 1). При этом прерывания от таймера 1 должны быть разрешены, и они должны использоваться для программной перезагрузки 16-битного значения в подпрограмме обслуживания прерывания.

Стандартные скорости последовательного обмена
с использованием таймера T/C1 Таблица 2

Режим работы SP	Скорость приема/передачи Кбит/сек	f_{CR}, МГц	SMOD	Таймер 1		
				C / $\overline{T}$	Режим	Загружаемое значение TH1
Режим 0	F_{MAKC} = 1000 Кбит/сек	12	x	X	X	X
Режим 2	F_{MAKC} = 375 Кбит/сек	12	1	X	X	X
Режимы 1, 3	62,5	12	1	0	2	FFh
	19,2	11,059	1	0	2	FDh
	9,6	11,059	0	0	2	FDh
	4,6	11,059	0	0	2	FAh
	2,4	11,059	0	0	2	F4h
	1,2	11,059	0	0	2	E8h
	0,110	6,0	0	0	2	72h
	0,110	12,0	0	0	1	TH1 = FEh TL1 = EBh

Передача последовательных данных в асинхронных режимах

Упрощенная функциональная схема передатчика в асинхронных режимах представлена на рис. 2.16. Передача последовательных данных инициируется любой командой записи байта передаваемых данных в регистр сдвига передатчика *SBUF* (сигналом *WR SBUF*). Сдвигающий регистр передатчика *SBUF* дополнен D-триггером, в который сигналом *WR SBUF* загружается «1» в режиме 1 или значение бита *TB8* из регистра *SCON* в ре жимах 2 и 3.

Собственно передача начинается в момент формирования ближайшего после сигнала *WR SBUF* импульса *TxClock*. Подчеркнем, что начало передачи синхронизируется сигналом *TxClock*, а не сигналом *WR SBUF*. Период сигнала *TxClock* определяет время трансляции каждого бита передаваемого байта данных на выводе *TxD*. В момент начала передачи активизируется внутренний сигнал *Send* (посылка), и на выводе *TxD* начинает формироваться старт-бит. Временная диаграмма работы передатчика в асинхронных режимах показана на рис. 2.17.

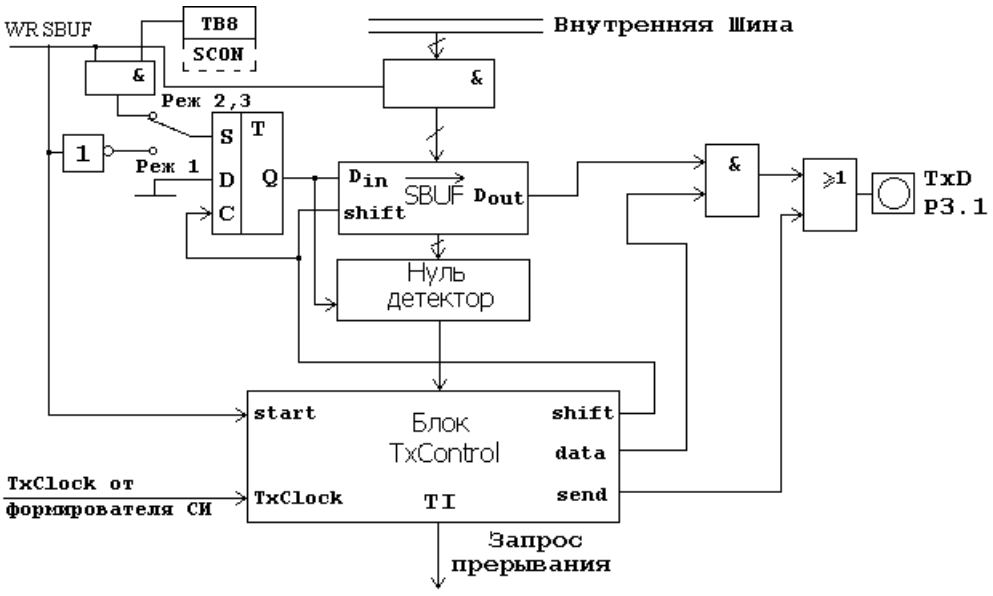


Рис. 2.16. Функциональная схема передатчика в асинхронных режимах

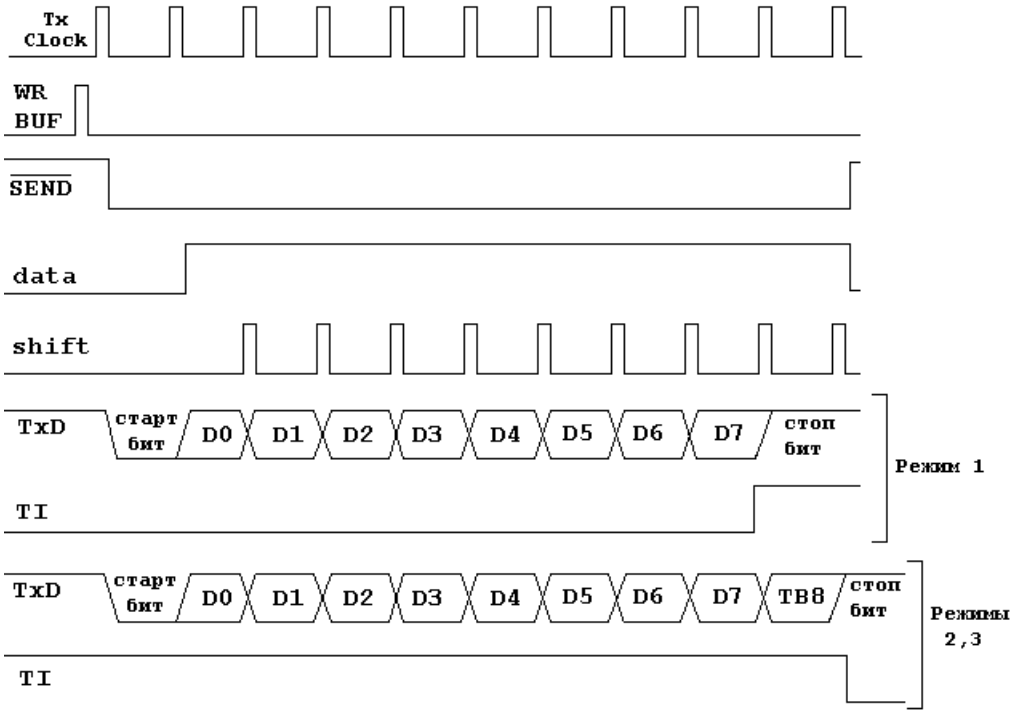


Рис. 2.17. Временные диаграммы работы передатчика в асинхронных режимах

С приходом следующего синхроимпульса *TxClock* завершается формирование старт-бита, и внутренние схемы блока *TxControl* активизируют внутренний сигнал *Data*, который разрешает работу формирователя импульсов сдвига передатчика *Shift*. С этого момента начинается преобразование параллельного кода *SBUF* в последовательный код и выдача его на внешний вывод *TxD*. После выдачи на выход *TxD* младшего бита *D*0 передаваемого байта данных в момент прихода

очередного импульса *TxClock* блок TxControl формирует первый импульс *Shift*. Содержимое *SBUF* сдвигается на один разряд вправо, при этом на вывод *TxD* поступает бит *D*1, а в старший разряд *SBUF* записывается значение 9-го дополнительного разряда («1» или *TB8*), после чего 9-й разряд *SBUF* обнуляется. Всего формируется 9 импульсов *Shift*. Нуль-детектор отслеживает состояние *SBUF* и по завершению выдачи кодовой посылки запрещает дальнейшее формирование импульсов *Shift*.

В режиме 1 на вывод *TxD* выдаются 8 бит данных и стоп-бит.

В режимах 2 и 3 на вывод *TxD* поступают 9-битовая посылка, содержащая 8 информационных бит и дополнительный бит TB8. В этих режимах стоп-бит формируется автоматически. После выдачи всех бит информационной посылки блок *TxControl* устанавливает флаг запроса прерывания передатчика *TI* и деактивизирует сигналы *Send* и *Data*. Отличием режима 1 от режимов 2 и 3 является момент формирования сигнала *TI*, который в режиме 1 выдается по окончании 8 бита данных, а в режимах 2 и 3 по завершении передачи дополнительного бита *TB8*.

Прием последовательных данных в асинхронных режимах.

Упрощенная функциональная схема приемника *SP* в асинхронных режимах представлена на рис. 2.18.

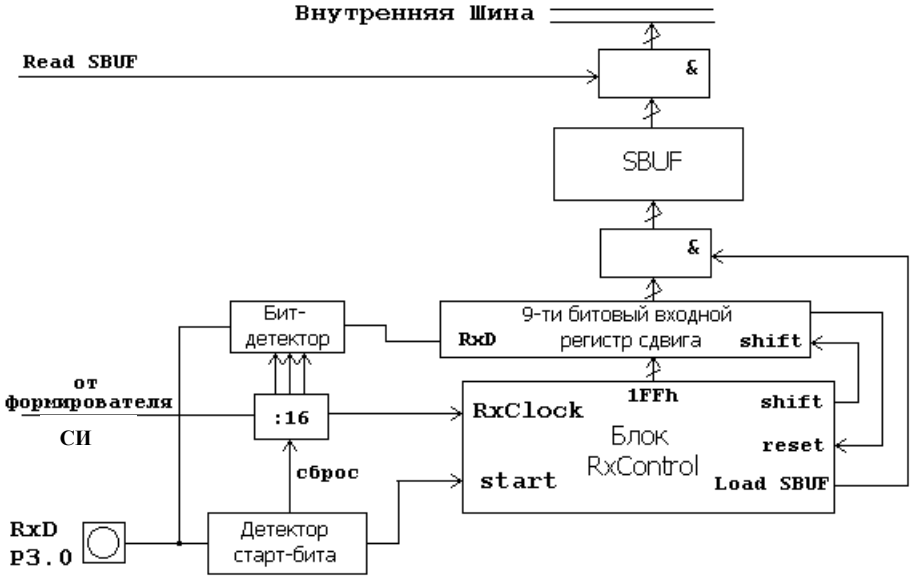


Рис. 2.18.Упрощенная функциональная схема приемника в асинхронных режимах

Для обнаружения входной посылки последовательных данных устройство управления приемника *RxControl* с частотой поступления импульсов *RxClock* опрашивает состояние контакта *RxD*. При обнаружении старт-бита (перехода из 1 в 0), выявленного детектором старт-бита, счетчик-делитель на 16 принудительно синхронизируется

(сбрасывается в 0). Благодаря этому обеспечивается согласование моментов опроса принимаемых бит. Заметим, что возможные сбои на линии связи могут привести к случайному выявлению старт-бита и, как следствие, к ошибочному приему последовательных данных на входе *RxD*.

Для исключения подобных ошибок блок *RxControl* содержит в своем составе специальный бит-детектор, принцип действия которого можно представить следующим образом. Шестнадцать состояний счетчика-делителя на 16 формирователя *RxClock* делят длительность каждого импульса, поступающего на вход *RxD*, на 16 фаз. В фазах 7, 8 и 9 бит-детектор считывает с входа *RxD* три значения принимаемого бита и в соответствии с мажоритарным принципом «2 из 3-х» выбирает истинное значение. Если мажоритарный отбор при приеме старт-бита показывает ненулевое значение, то фиксируется ошибка сбойного старт-бита, при этом все устройства блока *RxControl* сбрасываются и начинается отслеживание (поиск) следующего старт-бита. При выявлении «правильного» старт-бита блок *RxControl* записывает в 9-битовый входной регистр сдвига число 1FFh и формирует первый внутренний импульс *Shift*, по спаду которого содержимое регистра сдвига сдвигается на один разряд, при этом в освобождающийся разряд записывается значение принятого бита. Всего формируется 10 сдвигающих импульсов Shift. В результате, после 10-го импульса в сдвигающем регистре приемника фиксируются биты данных *D*7 – *D*0, а в 9-й дополнительный разряд в зависимости от режима помещается стоп-бит или бит *RB8*. Окончание приема байта данных должно завершаться формированием сигнала запрос прерывания приемника *RI* и переписью принятого байта данных в регистр *SBUF*. Однако эти действия выполняются только при выполнении некоторых условий, которые зависят от режима работы порта *SP*. В режиме 1 сигнал загрузки данных в регистр SBUF формируется в момент генерации 10-го импульса при условии, что флаг *RI* сброшен (*RI* = 0) и либо бит *SM2* в регистре *SCON* равен 0, либо принятый стоп-бит равен 1. В режимах 2 и 3 сигнал загрузки *SBUF* вырабатывается после 10-го импульса *Shift* только при выполнении условий: *RI* = 0 и 9-й принятый бит RB8 при установленном бите *SM2* равен 1 (*RB8* = 1), Одновременно с формированием сигнала загрузки устанавливаются биты *RB8* и *RI* в регистре *SCON*. Если хотя бы одно из приведенных условий не выполняется, принятая посылка безвозвратно теряется, а флаг *RI* не устанавливается.

Во всех асинхронных режимах по окончании приема последовательной посылки независимо от выполнения приведенных условий блок RxControl вновь начинает отслеживать старт-бит на входе RxD. Прием последовательных данных в асинхронных режимах иллюстрируется временными диаграммами (рис. 2.19).

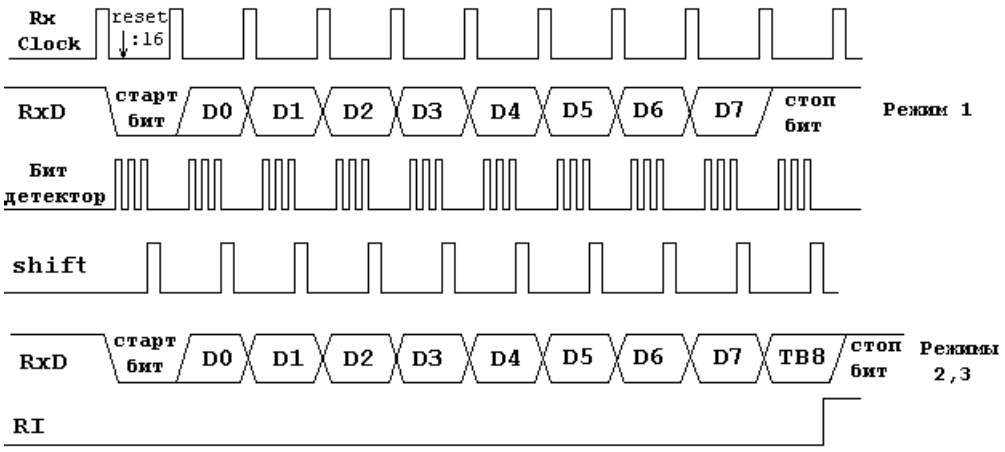


Рис. 2.19. Временные диаграммы работы приемника в асинхронных режимах.

Организация межконтроллерных обменов в локальной сети

При организация межконтроллерных обменов протокол обменов несколько усложняется. Перед передачей информационной посылки данных сначала необходимо выбрать МК – получатель посылки, т.е. сначала необходимо передать адрес МК, и только после этого можно передавать данные. Чтобы реализовать такой протокол, необходимо уметь разделять адресные посылки от посылок данных. Аппаратные средства МК SAB 80C515 обеспечивают такое разделение.

В МК SAB 80C515 мультипроцессорный режим работы задается с помощью управляющего бита SM2 в регистре SCON. В режимах 2 и 3 бит SM2 фактически управляет разрешением прерывания от SP: при SM2=1 прерывания от SP активизируются только при поступлении «единичного» 9-го бита TB8 информационной посылки. Это позволяет всем устройствам, подключенным к последовательному каналу, одновременно принимать так называемый фрейм адреса приемника и определять, с кем из них передатчик желает установить связь.

Программируя бит RB8 в передаваемой информационной посылке (признаком адресной посылки служит установленный «единичный» бит RB8, а признаком информационной посылки данных - сброшенный бит RB8), на приемной стороне в соответствии со значением принимаемого бита TB8 разделить адреса и данные не представляет труда. При SM2=1 порт SP приемника принимает адресную информацию, а при SM2=0 выполняется обмен данными.

В качестве иллюстрации рассмотрим организацию межконтроллерных обменов в мультипроцессорной системе с одним ведущим и несколькими ведомыми МК (рис. 2.20).

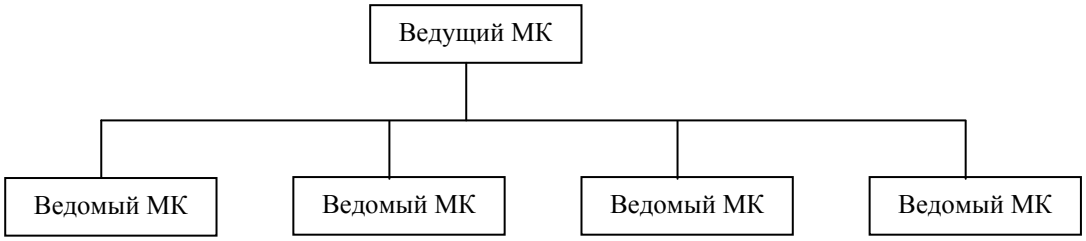


Рис. 2.20. Мультипроцессорная система с одним ведущим и несколькими ведомыми МК

В исходном состоянии все ведомые МК запрограммированы в режим приема адресной информации, т.е. имеют установленный бит SM2. При необходимости передачи данных ведущий МК выдает в канал связи адресную посылку с адресом выбираемого ведомого МК, в которой бит TB8 установлен (TB8=1). Так как в принимаемой посылке бит RB8 установлен, все ведомые МК получат ее, и во всех ведомых МК возникнут прерывания по приему RI. Обработчик прерывания RI осуществляет сравнение принятого адреса с собственным сетевым адресом. При равенстве адресов адресуемый МК идентифицирует себя как получатель. Программа обработки прерывания этого МК сбрасывает бит SM2 в регистре SCON (SM2 = 0), подготавливая МК к приему данных. Остальные ведомые МК оставляют неизменным состояние бита SM2 (SM2 = 1) и возвращают управление собственной основной программе. После передачи адресной информации ведущий МК начинает выдачу в канал связи посылок данных со сброшенным битом TB8. Получателем данных будет только выбранный МК, у которого бит SM2=0. Остальные МК не будут принимать передаваемые данные, поскольку в них бит SM2 = 1, и посылки данных с нулевым битом RB8 не вызывают генерацию запросов прерывания. После завершения сеанса обмена ведущий МК передает ведомому соответствующую команду, устанавливающую МК в режим приема адресной информации.

Рассмотренный протокол больше всего подходит для сети распределенных МК, подключенных по разделяемому моноканалу к одному управляющему компьютеру. Используя рассмотренный подход можно реализовать и более сложные протоколы обмена, в которых функции ведущего МК могут временно захватываться любым из МК.

2.5. Встроенные средства ввода-вывода аналоговых сигналов МК SAB 80C515. МК содержит в своем составе 8-канальный аналого-цифровой преобразователь поразрядного уравновешивания, обеспечивающий преобразование входных аналоговых сигналов напряжения в диапазоне от 0 до 5,12 В в цифровой 8- или 10-разрядный код (рис. 2.21).

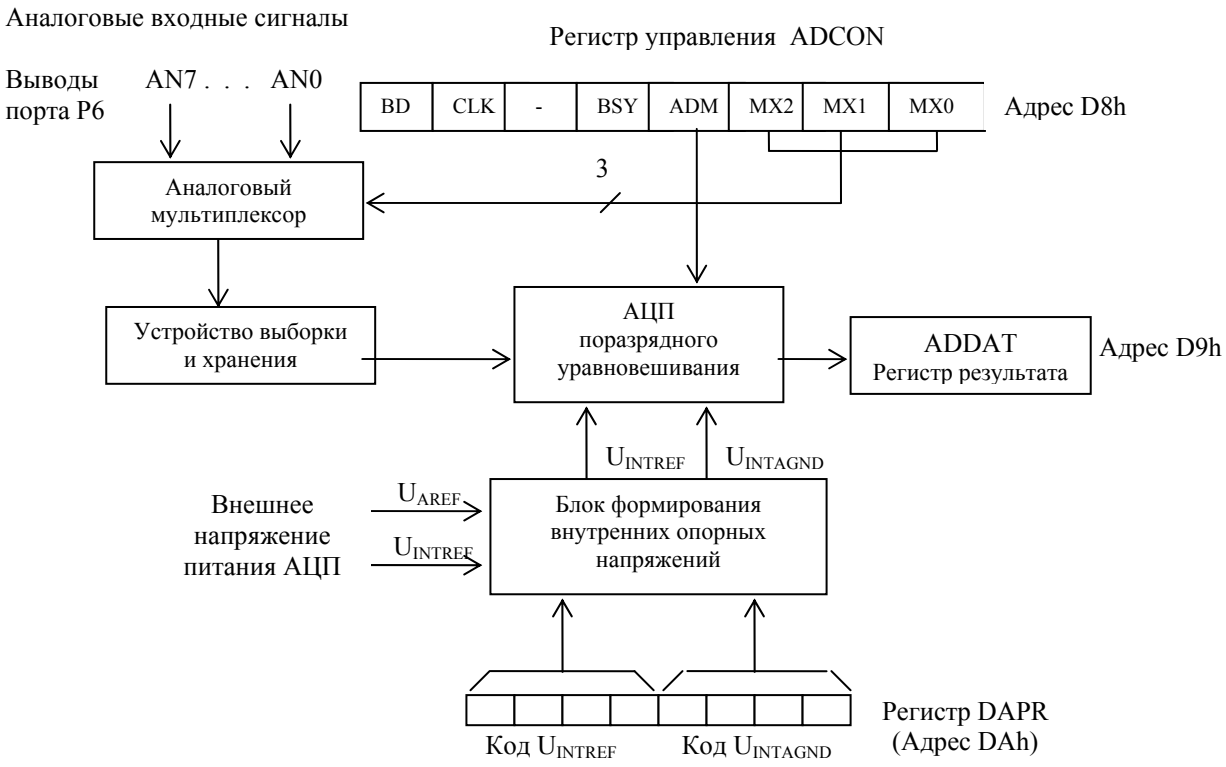


Рис. 2.21. Функциональная схема аналого-цифрового преобразователя МК SAB 80C515

АЦП состоит из 8-канального аналогового мультиплексора, устройства выборки и хранения входного аналогового сигнала, собственно цифроаналогового преобразователя поразрядного уравновешивания, блока формирования внутренних опорных напряжений и устройства управления.

Аналоговый мультиплексор (коммутатор аналоговых сигналов) позволяет использовать АЦП для преобразования аналоговых сигналов от нескольких датчиков. Устройство выборки и хранения формирует из непрерывно меняющегося аналогового сигнала *U(t)* последовательность отдельных выборок *U(n*), которые с помощью АЦП преобразуются в цифровой код. Блок формирования внутренних эталонных напряжений используется для задания опорных напряжений сравнения.

Работу АЦП поддерживают три программно доступные регистра блока SFR: регистр управления *ADCON*, регистр результата *ADDAT* и регистр управления внутренними опорными напряжениями *DAPR*.

Для управления АЦП используются управляющие биты регистра *ADCON*: три бита *MX2 – MX0*, кодирующие номер аналогового входа, подключенного через мультиплексор к входу АЦП, и бит *ADM*.

Бит ADM программирует режим работы АЦП. При *ADM* = 0 задается режим одиночного преобразования, в котором после запуска АЦП выполняется одно преобразование аналогового сигнала, после чего работа

АЦП прекращается. При *ADM* = 1 АЦП переводится в режим непрерывного преобразования. В этом режиме автоматически по завершении очередного преобразования выполняется следующее.

Бит *BSY* (флаг занятости) регистра *ADCON* осуществляет индикацию состояния АЦП: при выполнении аналого-цифрового преобразования бит *BSY* автоматически устанавливается, а по завершении преобразования сбрасывается. При завершении преобразования одновременно со сбросом флага занятости BSY в регистре *IRCON* автоматически устанавливается флаг запроса прерывания *IADC*. Два старших бита *BD* и CLK регистра *ADCON* для управления АЦП не используются.

8-битный регистр *ADDAT* предназначен для запоминания результата аналого-цифрового преобразования. Содержимое регистра *ADDAT* сохраняется в течение всего следующего цикла преобразования и изменяется только при его перезагрузке. Регистр *ADDAT* программно доступен, поэтому если АЦП не требуется конкретным приложениям, то *ADDAT* может быть использован как регистр общего назначения.

Регистр *DAPR* (D/A Converter Program Register) предназначен для программного задания внутреннего опорного напряжения $U_{INTAREF}$ и $U_{INTAGND}$, используемого для формирования эталонного напряжения сравнения. Дискретность преобразования входных аналоговых сигналов определяется значениями напряжений $U_{INTAREF}$ и $U_{INTAGND}$ на выходе блока формирования внутренних опорных напряжений, который, управляемого 8-разрядным регистром *DAPR*.. С помощью резистивной матрицы этого блока опорное напряжение внешнего стабильного источника напряжения U_{AREF} преобразуется во внутренние опорные напряжения $U_{INTAREF}$ и $U_{INTAGND}$. Оба внутренних напряжения могут программно изменяться относительно внешней аналоговой "земли" U_{AGND} с шагом 1/16 от напряжения U_{AREF} (табл.3). Четыре младших разряда регистра *DAPR* управляют формированием внутреннего напряжения $U_{INTAGND}$, а четыре старших – формированием $U_{INTAREF}$. Значения $U_{INTAGND}$ и $U_{INTAREF}$ вычисляются по формулам:

$$U_{INTAGND} = U_{AGND} + \frac{DAPR.3 \div DAPR.0}{16} \cdot (U_{AREF} - U_{AGND})$$

$$U_{INTAREF} = U_{AGND} + \frac{DAPR.7 \div DAPR.4}{16} \cdot (U_{AREF} - U_{AGND})$$

Для правильной работы АЦП необходимо, чтобы минимальная разность между опорным напряжением $U_{INTAREF}$ и напряжением $U_{INTAGND}$ была больше 1,25В. При внешнем опорном напряжении U_{AREF} = 5 В и U_{AGND} = 0 В шаг дискретности задания внутренних опорных напряжений равен 5В /16 = 0,3125 В.

Значения внутренних опорных напряжений
$U_{INTAREF}$ и $U_{INTAGND}$ Таблица 3

DAPR.3 – DAPR.0	$U_{INTAGND}$ (В)	DAPR.7– DAPR.4	$U_{INTAREF}$ (В)
0000	0,0	0000	5,0 В
0001	0,3125	0001	–
0010	0,625	0010	–
0011	0,9375	0011	–
0100	1,25	0100	1,25
0101	1,5625	0101	1,5625
0110	1,875	0110	1,875
0111	2,1875	0111	2,1875
1000	2,5	1000	2,5
1001	2,8125	1001	2,8125
1010	3,125	1010	3,125
1011	3,4375	1011	3,4375
1100	3,75	1100	3,75
1101	–	1101	4,0625
1110	–	1110	4,375
1111	–	1111	4,6875

Задавая различные значения регистра *DAPR* можно программировать внутренние опорные напряжения $U_{INTAREF}$ и $U_{INTAGND}$ индивидуально для каждого диапазона входных сигналов. Благодаря этому обеспечивается возможность изменения шага квантования (дискретности преобразования) в зависимости от диапазона входных сигналов. Например, для диапазона входных сигналов 0 – 5 В ошибка квантования составляет порядка 20 мВ, а для диапазона 0 – 3 В - 12 мВ. Минимальная ошибка квантования обеспечивается при значении Uвх ≤ Uоп =1,25 В и равняется величине $\Delta = 1{,}25\ \text{В} / 256 \cong 5$ мВ. Возможность достижения такой погрешности эквивалентна увеличению точности преобразования входных сигналов номинального диапазона (0 - 5 В) до 10-ти разрядов.

10-разрядное преобразование выполняется за два последовательных цикла преобразования. В первом цикле осуществляется "грубое" измерение входного сигнала и формируется 8-битный результат, старшие четыре разряда которого помещаются в четыре младших разряда регистра *DAPR*. При этом изменяется уровень внутреннего напряжения $U_{INTAGND}$ ($U_{INTAGND}$ приближается к $U_{INTAREF}$). При загрузке *DAPR* результатом "грубого" измерения инициируется повторное преобразование входного сигнала, в котором разность между значением Uвх и его грубой оценкой преобразуется с установленным для этой разности новым значением Uоп

(с более сжатым диапазоном внутренних опорных напряжений). В результате двойного преобразования формируется 10-разрядный результат, 4 старших разряда которого размещаются в регистре *DAPR* (*DAPR*.3 – *DAPR*.0), а шесть младших разрядов - в регистре *ADDAT* (*ADDAT*.7 – *ADDAT*.2). Стандартное 8-разрядное преобразование выполняется за 15 машинных циклов, 10-разрядное - за 22-29 машинных циклов.

В общем случае, порядок работы с аналого-цифровым преобразователем сводится к выполнению следующих действий:

- с помощью устройства управления задается режим работы преобразователя (программируется бит *ADM*) и указывается номер аналогового входа, подключаемого через мультиплексор к входу АЦП;
- в регистр DAPR производится запись константы, определяющей задаваемый диапазон внутренних опорных напряжений. Запись константы в регистр *DAPR* запускает преобразование. Входное напряжение преобразуется в код N в соответствии с выражением N = 256 * Uвх/5.
- по завершению преобразования код преобразуемого напряжения считывается из регистра *ADDAT*, при этом АЦП готов к очередному преобразованию.

При 8-разрядном преобразовании и значении опорного напряжения 0 - 5 В (данный режим задается командой записи константы #00h в регистр *DAPR*) точность преобразования составляет 20 мВ.

Процедура преобразования является сравнительно длительной (до 29 мкс), поэтому перед считыванием результата преобразования используют временную задержку. Текст программы аналого-цифрового преобразования входных аналоговых сигнала приведен на рис.2.22.

```
;Программа adc_read, осуществляющая преобразование входного аналогового сигнала в цифровой код

ADCON:      equ     D8h
ADDAT:      equ     D9h
DAPR:       equ     Dah
U:          equ     40h

        ;Настройка АЦП
        mov A,#data     ; загрузка управляющего слова, определяющего режим работы АЦП
        anl ADCON,#E0h
        orl ADCON,A

; Запуск преобразования (осуществляется при загрузке в регистр DAPR управляющего слова, задаю-
; щего значения внутренних опорных напряжений UAREF и UAGND (см. табл.3).
        mov DAPR,#0

    mov R7,#15  ;выдержка паузы для аналого-цифрового преобразования
m1:     djnz R7,m1
        mov U,ADDAT   ;запоминание результата преобразования
        ret
```

Рис.2.22. Текст программы преобразования аналоговых сигналов в цифровой код

2.6. Блок таймеров/счетчиков МК SAB 80C515

МК SAB 80C515 содержит в своем составе три программируемых 16-разрядных таймера-счетчика – *T/C0*, *T/C1* и *T/C2*. Таймеры реализуются на основе 16-разрядных суммирующих счетчиков. Все три таймера могут работать либо в режиме таймера, либо в режиме счетчика событий. Программно доступными регистрами таймеров *T/C0*, *T/C1* и *T/C2* являются регистры блока SFR: *TH0* и *TL0* (таймер 0), *TH1* и *TL1* (таймер 1), *TH2* и *TL2* (таймер 2).

Таймеры не идентичны. Таймеры *T/C0* и *T/C1* имеют одинаковую внутреннюю структуру. Они могут использоваться в качестве программно управляемого таймера, генератора программируемой частоты, счетчика внешних событий. В дополнение к названным применениям таймер *T/C1* используется для синхронизации работы приемопередатчика порта SP и управления скоростью последовательного обмена.

С помощью таймера *T/C2*, наряду с стандартными функциями управляемого таймера и счетчика, реализуют блок быстрого ввода/вывода.

2.6.1. Особенности организации и использования таймеров T/C0 и T/C1

Обобщенная структура таймеров *T/C0* и *T/C1* приведена на рис.2.23. На схеме таймер представлен своей программной моделью в виде 16-разрядного регистра *TLx:THx* (*x*=0,1).

Входные (счетные) импульсы таймеров *T/Cx* (*x*=0,1) в зависимости от управляющего сигнала $C/\overline{T}$ могут поступать либо от внутреннего источника тактовых импульсов $f_{CR}/12$ (при $C/\overline{T}$ = 0), либо от внешнего источника с входа *T*x (при $C/\overline{T}$=1). Счетчики таймеров работают на сложение. Признаком окончания счета является сигнал переполнения T_{ov}, фиксируемый во флаге TF_x. Сброс флага выполняется программно (обычно в программе обработки прерывания).

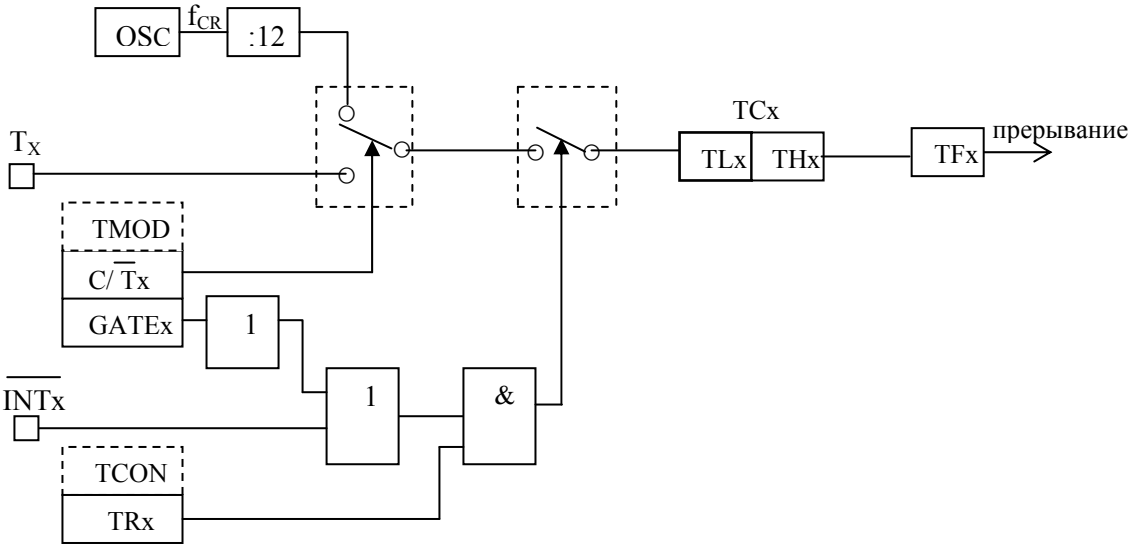


Рис.2.23. Обобщенная структура таймеров *T/C0* и *T/C1*

При работе в режиме таймера ($C/\overline{T}$ = 0) содержимое *TLx:THx* увеличивается в каждом машинном цикле с частотой f_{CR}/12. При работе в режиме счетчика ($C/\overline{T}$=1) содержимое TLx:THx инкрементируется при переходе сигнала на внешнем выводе *T*x из 1 в 0. В этом режиме уровень сигнала на внешнем выводе T_X опрашивается в фазе *S5P2* каждого машинного цикла. Наличие внешнего события фиксируется при обнаружении высокого уровня сигнала в одном цикле и низкого уровня в следующем. Новое значение счетчика формируется в фазе *S3P1* цикла, следующего за тем, в котором был обнаружен переход из 1 в 0. Поскольку на распознавание перехода требуется два машинных цикла, максимальная частота счета входных событий равна f_{CR}/24.

Для управления таймером *T/Cx* (x = 0,1) и задания режимов его работы используются два регистра блока SFR: регистр управления *TCON* (рис. 2.24) и регистр режимов *TMOD* (рис. 2.25).

8Fh	8Eh	8Dh	8Ch	8Bh	8Ah	89h	88h	TCON
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	Адрес 88h

Рис. 2.24. Формат регистра TCON

Регистр *TCON* для каждого таймера *T/Cx* (x=0,1) содержит бит управления разрешением работы *TRx*, с помощью которого осуществляют пуск или останов таймера, и флаг переполнения *TFx*. Установка бита *TRx* является необходимым, но недостаточным условием разрешения счета. Счет разрешен, если *TRx* = 1 и отсутствует блокировка работы *T/Cx* : бит *GATEx* в регистре *TMOD* сброшен или отсутствует запрос прерывания на внешнем входе ($\overline{INTx}$=1). При невыполнении хотя бы одного из приведенных условий работа таймера *T/Cx* блокируется и счет запрещается. Четыре младших бита регистра *TCON* к работе таймера отношения не имеют. Управляющие биты *IT0* и *IT1* специфицируют вид сигнала на входах прерывания *INT0* и *INT1*. При *ITx*=0 запрос прерывания вызывает уровень (логический «0»), а при *ITx*=1 прерывание вызывает спад сигнала на входе *INTx* (переход из «1» в «0»). Биты IE0 и IE1являются флагами внешних прерываний *INT0* и *INT1*. Их состояние определяется наличием/отсутствием запроса прерывания на соответствующем входе.

Регистр *TMOD* содержит набор программно управляемых бит (по четыре бита для каждого таймера), используемых для задания режима *T/C*.

Рассмотрим особенности работы таймеров *T/C0* и *T/C1* в различных режимах.

Режимы работы 0, 1, 2 таймеров одинаковы. В этих режимах таймеры *T/C0* и *T/C1* полностью независимы друг от друга.

Таймер 1				Таймер 0				TMOD
GATE1	$C/\overline{T}1$	M1	M0	GATE0	$C/\overline{T}0$	M1	M0	Адрес 89h

GATEx - бит управления блокировкой таймера T/Cx;

$C/\overline{T}x$ – бит, определяющий функцию таймера/счетчика T/Cx:

таймер ($C/\overline{T}x = 0$) или счетчик ($C/\overline{T}x = 1$);

M1, M0 – биты, задающие режим работы T/Cx (x=0,1).

M1	M0	Режим работы таймера
0	0	8-битный T/C на базе регистра THx (x=0, 1). Регистр TLx используется как 5-разрядный предделитель
0	1	16-битный T/C. Регистры THx и TLx включены последовательно как один 16-битный регистр
1	0	Автогенератор на основе 8-разрядного регистра TLx. Регистр THx содержит значение, которое загружается в TLx при каждом переполнении TLx
1	1	8-битный T/C на базе регистра TL0, управляемый битом TR0, и 8-битный таймер на базе TH0, управляемый битом TR1. Таймер T/C1 остановлен

Рис. 2.25. Формат и назначение отдельных бит регистра TMOD

Режимы 0 и 1 таймеры *T/C0* и *T/C1* используются для счета входных событий и формирования сигналов требуемой длительности (временных задержек). Логику работы *T/Cx* в этих режимах поясняет рис. 2.26.

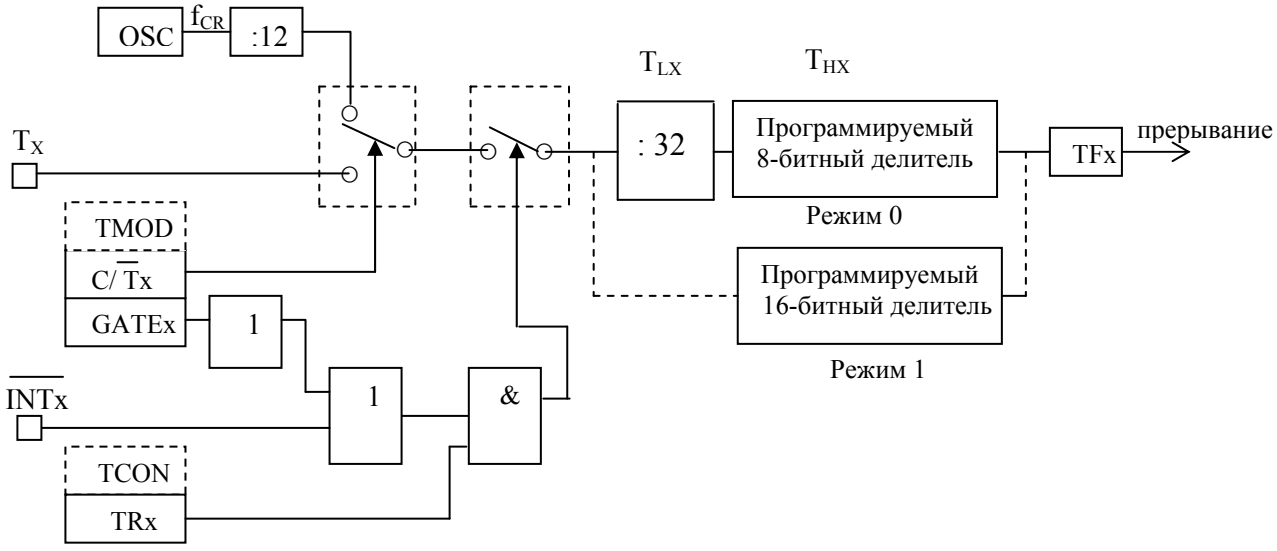


Рис. 2.26. Логика работы таймера T/Cx в режимах 0 и 1

В *режиме счетчика событий* ($C/\overline{T} = 1$) счетчик таймера увеличивает свое значение на 1, когда сигнал на входе *T*x переходит из 1 в 0. Работой *T/Cx* управляет схема управления, логика которой рассмотрена выше.

В *режиме таймера* ($C/\overline{T} = 0$) счетные импульсы $f_{CR}/12$ поступают на вход программируемого делителя (8-разрядного в режиме 0 или 16-разрядного в режиме 1). Разрядность делителя является основным отличием этих режимов.

Программируемым делителем в режиме 0 является 8-разрядный регистр *THx*. На его вход поступают счетные импульсы с выхода предделителя на 32. При f_{CR}=12 МГц счетчик *THx* увеличивает свое значение через каждые 32 мкс, обеспечивая формирование временных задержек в диапазоне от 32 мкс до 8192 мкс.

В режиме 1 в качестве счетчика-делителя используется программируемый 16-битный счетчик *THx:TLx*. При f_{CR} = 12 МГц счетчик-делитель обеспечивает формирование временных задержек в диапазоне от 1 мкс до 65535 мкс. При формировании требуемых временных задержек счетчик таймера предварительно должен быть инициализирован значением (*FFFFh – Nзад)*, где *Nзад* – численное значение формируемой задержки.

Таймер/счетчик *T/Cx* в режимах 0 и 1 можно использовать для измерения длительности входных импульсов, поступающих на вход *GATEx*. При *TRx*=1 и нулевом значении сигнала $\overline{INTx}$ на одноименном входе управление счетом осуществляется с помощью сигнала *GATEx*. Единичное значение сигнала *GATEx* запрещает счет, а нулевое значение - разрешает счет. Число, зафиксированное в счетчике *THx:TLx*, определяет длительность входного импульса в соответствующих временных единицах.

Режим 2. В режиме 2 таймер *T/Cx* программируется для работы в качестве автогенератора на базе 8-разрядного счетчика *TLx*. Частота выходных импульсов автогенератора T_{ov}, определяется при программной установке регистра *THx*. Логику работы *TCx* в режиме 2 поясняет рис. 2.27.

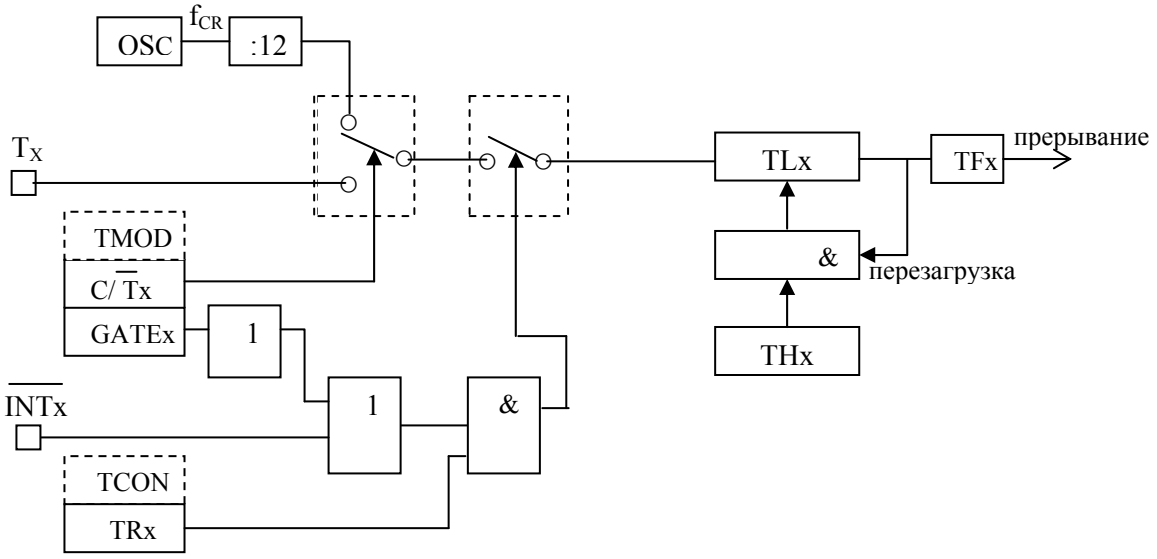


Рис. 2.27. Логика работы таймера T/Cx в режиме 2.

При разрешенном счете выходные импульсы T_{ov}, формируемые при переполнении счетчика *TLx*, не только устанавливают флаг *TFx*, но и автоматически перезагружают регистр *TLx* значением из регистра *THx*, который предварительно должен быть инициирован программным путем. Перезагрузка *TLx* не изменяет содержимое регистра *THx*.

Режим 3. В режиме 3 таймеры *T/C0* и *T/C1* работают по-разному. В режиме 3 работа *T/C0* и *T/C1* различна. При переводе *T/C0* в режим 3 работа *T/C1* блокируется.

Таймер *T/C0* в режиме 3 функционирует как два независимых 8-битных таймера/счетчика на базе счетчиков *TH0* и *TL0* (рис. 2.28), а таймер *T/C1* остановлен. В этом режиме на базе счетчика *TL0* реализован обычный 8-разрядний таймер/счетчик, для управления которым используются биты *TR0*, *GATE0*, и $C/\overline{T}$. Переполнение счетчика *TL0* фиксируется во флаге *TF0* регистра *TCON*. Таймер на базе счетчика *TH0* в режиме 3 может использоваться только как таймер, управляемый битом *TR1*.Его переполнение фиксируется во флаге *TF1*.

При работе таймера *T/C0* в режиме 3, таймер *T/C1* может использоваться в любом режиме, не требующем прерывания. В частности, таймер *T/C1* может работать в режиме 2 и формировать синхросигналы последовательного порта SP. Можно считать, что при переключении *T/C0* в режим 3 МК SAB 80C515 имеет в своем составе три таймера.

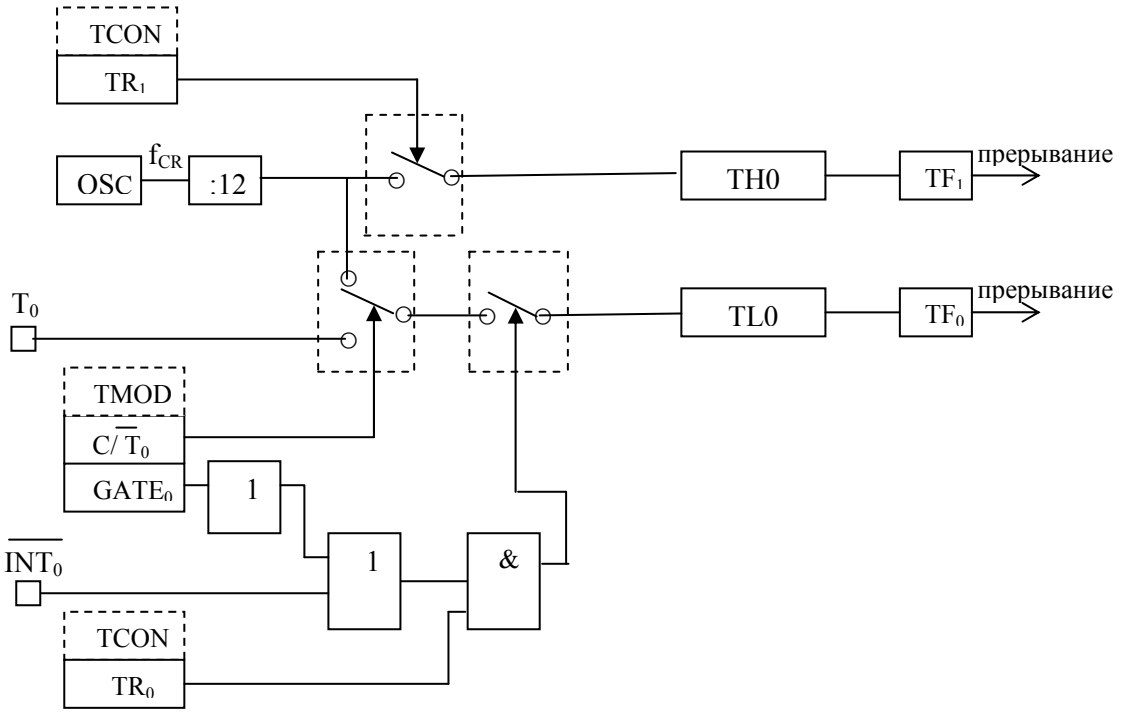


Рис. 2.28. Логика работы таймера T/C0 в режиме 3

Примеры применения таймера

Важнейшим использованием таймера является формирование «тика» - временного интервала программируемой длительности. «Тики» используются в программах реализации различных временных функций: часы реального времени, определение длительности временных интервалов, формирование сигналов требуемой частоты, определение моментов наступления входных событий, генерация выходных событий в заданные моменты времени и т.п. 16-разрядный таймер без цепей перезагрузки позволяет формировать «тики» длительностью 65535 мкс. При формировании временных интервалов произвольной длительности используют предварительную программную загрузку счетчика таймера, при этом таймер программируется для работы либо в режиме неуправляемого делителя частоты с автоперезагрузкой, либо в режиме счетчика с программной перезагрузкой при переполнении счетчика таймера. «Тик» требуемой длительности T формируется из импульсов тактовой частоты в соответствии с выражением $T = N \cdot \tau_0$, где τ_0 – период импульсов тактовой частоты, N – расчетное значение длительности.

Поскольку счетчик таймера работает на сложение, для формирования «тика» T требуемой длительности в счетчик таймера необходимо загружать константу $N_{загр} = (2^{16} - 1) - N$. Для $f_{CR}/12 = 1$МГц в табл.4 приведены константы, соответствующие некоторым значениям формируемых «тиков».

Таблица 4

T, мкс	10000	5000	3333	2500	2000	1666	1428	1250	1111	1000	500	200
$N_{загр}$	55535	60535	62203	63035	63535	63870	64107	64285	64424	64535	65035	65335
$N_{загр\ hex}$	D8EF	EC77	F2FB	F63B	F82F	F97E	FA6C	FB1D	FBA8	FC17	FE0B	FF37

Аппаратный «тик», формируемый таймером, фиксируется при переполнении счетчика таймера. При разрешенных прерываниях в момент формирования импульса переполнения τ_{ov} запускается обработчик прерывания, реализующий функции, определяемые пользователем. Обработчик прерывания должен быть коротким, чтобы не прерывать основную программу на длительное время и не блокировать другие одноуровневые прерывания. «Длинные» функции обработчика, если они необходимы, следует реализовывать в основной (фоновой) программе управления объектом.

В качестве примера реализации временной функции рассмотрим программу генерирования меандра с частотой 100 Гц на выходе одного из разрядов порта, например *P4.0*.

Поскольку в меандре длительность импульса равна длительности паузы для формирования меандра с помощью *T/C0* достаточно сгенерировать

«тик», длительность которого равна полупериоду требуемой частоты (для рассматриваемого примера 5 мс) и проинвертировать разряд порта *P4.0*.

Генерирование «тика» требуемой длительности легко реализуется с помощью обработчика прерывания по переполнению таймера T/C0, который запрограммирован для работы в режиме 16-битного счетчика. Обработчик должен содержать команды перезагрузки таймера и команду инвертирования разряда порта *P4.0* и (см. пример). Константа перезагрузки для «тика» 5 мс равна *EC77h* (см. табл.4).

Схема алгоритма программы timer.asm, обеспечивающей формирование меандра требуемой частоты, и текст программы приведены на рис. 2.29,а и рис. 2.29,б.

а)

```
;timer.asm – программа генерирования меандра требуемой частоты
        org 8100h
        lcall init      ;вызов подпрограмма инициализации таймера T/C0
                        ; и системы прерываний
loop:   sjmp loop       ;пустой бесконечный цикл

init:   anl TMOD,#11110000b     ;инициализация таймера для работы
        orl TMOD,#00000001b     ;в режиме 16-битного счетчика
        mov TH0,#ECh            ;инициализация счетчика T/C0 для
        mov TL0,#77h            ;формирования «тика» 5 мс
        setb ea                 ;разрешение всех прерываний
        setb et0                ;разрешение прерывания T/C0
        setb tr0                ;разрешение счета
        ret
tim0:   mov TH0,#ECh            ;инициализация счетчика T/C0 для
        mov TL0,#77h            ;формирования «тика» 5 мс
        cpl P4.0                ;формирование меандра
        reti
        org 800bh               ;обработчик прерывания T/C0
        ljmp tim0
```

б)

Рис. 2.29. Алгоритм (а) и текст программы формирования меандра требуемой частоты (б)

Существенной особенностью формирования аппаратных «тиков» с помощью таймера с программной перезагрузкой является наличие временных затрат на выполнение команд обработчика, увеличивающих длительность «тика» (каждая из команд обработчика выполняется за два машинных цикла и на перезагрузку счетчика таймера тратится минимум 6 мкс, к этому времени необходимо добавить время вызова обработчика). Временные затраты на программную перезагрузку в обработчике можно скомпенсировать, увеличив константу $N_{загр}$. Учитывая отмеченную особенность формирования аппаратных «тиков», перезагружаемая константа $N_{загр}$ в общем случае должна рассчитываться с учетом временных затрат на выполнение обработчика.

Другим примером применения таймера является его использование в качестве источника счетных импульсов электронных часов.

Электронные часы строят на основе счетчиков (аппаратных и/или программных). С их помощью осуществляется пересчет импульсов определенной длительности («тиков») в секунды, минуты, часы. По существу электронные часы – это многоступенчатый счетчик, число ступеней которого определяется точностью представления времени. Аппаратные «тики» формируются таймером, запрограммированным для работы в режиме 16-битного счетчика. В зависимости от программной настройки можно сформировать аппаратные «тики» длительностью от единиц микросекунд до 65535 мкс. Содержимое счетчиков отображается на электронных индикаторах, например, экране модуля ЖКИ. При заданной длительности «тика» 500 мкс часы, отображающие время с точностью 0,1 с, можно представить схемой (рис. 2.30).

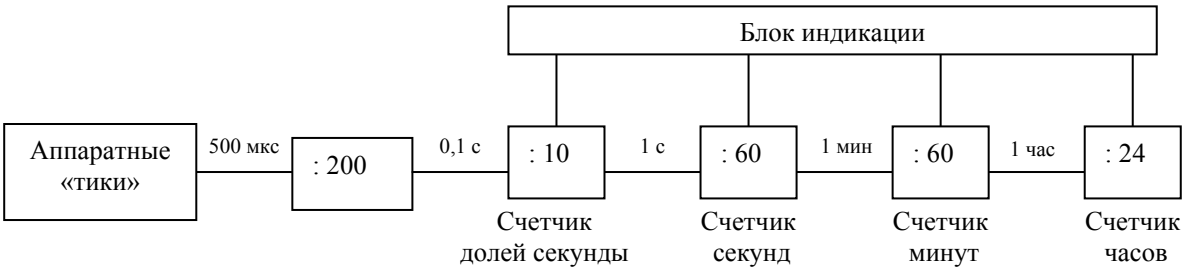


Рис. 2.30. Функциональная схема электронных часов

Счетные временные сигналы (секунды, минуты, часы) формируют программно. Построить программный счетчик с заданным коэффициентом пересчета нетрудно. Например, программный счетчик можно реализовать циклической процедурой инкремента и сравнения со значением коэффициента пересчета. При достижении этого значения осуществляется выход из цикла, при этом программно формируется счетный импульс следующей ступени и счетчик сбрасывается в нуль.

```
;фрагмент программы, реализующей программный счетчик
            inc count
metka:      mov a,count
            cjne a,#const,metka
            inc count2
            mov count,#0
```

Программная обработка аппаратных «тиков» должна реализовываться в «коротком» обработчике прерываний от переполнения таймера. Минимальный набор операций, выполняемый в обработчике, включает перезагрузку счетчика таймера значением константы, определяющей «тик», и инкремент программного счетчика «тиков». Программный пересчет «тиков» в секунды, минуты, часы целесообразно реализовать в

фоновой (циклической) программе, в которой, наряду с преобразованием цифровой информации в счетчиках таймера в ASCII коды и отображением их на экране ЖКИ, могут выполняться какие-либо другие действия.

Выполнение фоновой программы может занимать сравнительно большой временной отрезок, в течение которого счетчик «тиков» в обработчике может зафиксировать не один, а несколько «тиков». Поскольку анализ содержимого счетчика «тиков» («делителя на 200») осуществляется в фоновой программе и производится не после каждого «тика», эту особенность счетчика «тиков» необходимо учитывать при его реализации. Построение «делителя на 200» в виде обычной циклической процедуры, осуществляющей инкремент содержимого счетчика и сравнение со значением коэффициента пересчета, может приводить к ошибке деления при счете. Не исключена ситуация, когда при сравнении содержимое счетчика окажется больше значения коэффициента пересчета. В этом случае в соответствии с действием команды сравнения *cjne a,#const,metka* счетчик продолжит счет, что приведет к ошибке деления и погрешности электронных часов. Для исключения ошибок пересчета можно использовать следующий фрагмент в фоновой программе.

```
    ;реализация «делителя на 200» на базе счетчика count, опрос содержимого которого выполняется
    ; в произвольные моменты времени

time:   mov a,count     ;загрузка аккумулятора содержимым счетчика «тиков»
        clr c
        subb a,#200     ;сравнение со значением коэффициента пересчета
        jc exit         ;контроль наличия импульса переполнения делителя
        mov count,a     ;коррекция значения счетчика
        inc count1      ;счет импульсов переполнения делителя
exit:   ...
```

Счетные импульсы на выходе «делителя на 200» формируются через каждые 100 мс и, поскольку их пересчет производится непосредственно в фоновой программе, остальные счетчики электронных часов обычно реализуют по классической схеме пересчета.

2.6.2. Особенности организации и использования таймеров T/C2

16-разрядный таймер/счетчик *T/C2* (рис.2.31), наряду с реализацией функций счета внешних событий и формирования сигналов требуемой частоты, является базовым таймером устройства быстрого ввода/вывода HSIO, предназначенного для регистрации входных и генерации выходных событий в реальном времени.

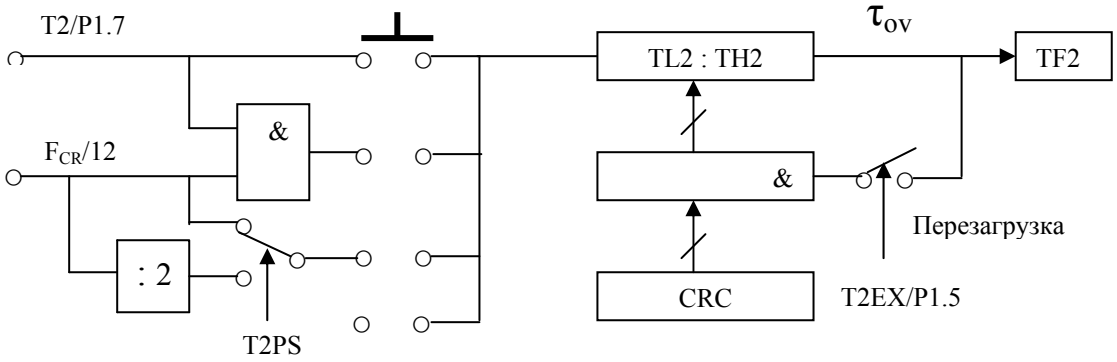


Рис.2.31. Функциональная схема таймера Т/С2

Режим работы таймера *T/C2* задается программно с помощью управляющих бит регистра *T2CON* блока SFR (рис.2.32).

0CFh	0Ceh	0CDh	0CCh	0CBh	0CAh	0C9h	0C8h
T2PS	I3FR	I2FR	T2R1	T2R0	T2CM	T2I1	T2I0

T2CON
Адрес C8h

T2I1	T2I0	Режим работы таймера Т/С2
0	0	Таймер 2 остановлен. Счетные импульсы на вход не поступают.
0	1	Режим неуправляемого таймера. На вход таймера поступают импульсы с частотой f_{CR}/12 (T2PS = 0) или f_{CR}/24 (T2PS = 1)
1	0	Режим управляемого таймера. На вход таймера поступают счетные импульсы с частотой f_{CR}/12 (T2PS = 0) или f_{CR}/24 (T2PS = 1)
1	1	Режим счетчика внешних событий, поступающих на вход T2/P1.7

T2R1	T2R0	Режим перезагрузки
0	X	Перезагрузка таймера 2 запрещена
1	0	Режим 0: автоматическая перезагрузка импульсами переполнения таймера
1	1	Режим 1: перезагрузка Т/С2 по сигналу на входе T2EX/P1.5

Рис.2.32. Формат регистра T2CON и назначение бит управления режимами работы таймера Т/С2

Назначение бит регистра *T2CON* следующее:

Биты *T2I1* и *T2I0* задают режим работы таймера *T/C2* и определяют источник входных счетных импульсов.

Биты *T2R1, T2R0* назначают режим перезагрузки регистров *TH2:TL2* таймера *T/C2*. Режимы перезагрузки могут использоваться для вызова подпрограмм обслуживания, которые должны выполняться с определенной периодичностью.

Бит *T2PS* – бит разрешения работы делителя на 2 частоты внутреннего источника счетных импульсов таймера 2. При *T2PS* = 0 счетные импульсы поступают с частотой f_{CR}/12, при T2PS = 1 – с частотой f_{CR}/24.

Биты *T2CM, I2FR* и *I3FR* не управляют работой таймера *T/C2*. Эти биты назначают один из двух режимов сравнения блока HSIO (бит *T2CM*) или определяют вид входных сигналов внешних прерываний $\overline{INT3}$ и $\overline{INT2}$ (биты *I3FR* и *I2FR*).

Функциональная схема таймера *T/C2* и рассмотрение назначения управляющих бит регистра *T2CON* достаточно прозрачно поясняют работу таймера в различных режимах.

На счетный вход таймера *T/C2* могут поступать импульсы либо от внутреннего источника тактовых импульсов с частотой $f_{CR}/12$ или $f_{CR}/24$ (режим таймера), либо сигналы с внешнего входа *T2/P1.7* (режим счетчика внешних событий). В режиме 0 (*T2I1 T2I0* = 00) таймер остановлен (счетные импульсы на его вход не поступают).

В режиме 1 (неуправляемого таймера) на вход таймера поступают импульсы с частотой $f_{CR}/12$ (T2PS = 0) или $f_{CR}/24$ (*T2PS* = 1). Таймер *T/C2* содержит специальные цепи перезагрузки счетчика *T/C2* содержимым регистра CRC. При выборе режима автоперезагрузки (*T2R1 T2R0* = 10) обеспечивается работа *T/C2* в качестве программируемого автогенератора.

В режиме управляемого таймера (режим 2) разрешением счета импульсов управляет сигнал на внешнем входе *T2/P1.7*. При *T2* = 1 счет разрешен, а при *T2* = 0 – запрещен. Наличие режима управляемого таймера обеспечивает простоту измерения длительности входных импульсов, поступающих на вход *T2/P1.7*.

В режиме счетчика внешних событий (режим 3) счетчик таймера *T/C2* инкрементируется при изменении уровня сигнала на входе *T2/P1.7* из 1 в 0. Внешний вход *T2/P1.7* опрашивается в каждом машинном цикле и для выявления перехода из 1 в 0 требуется два машинных цикла. Максимальная скорость счета внешних событий ограничена величиной $f_{CR}/24$.

Переполнение таймера τ_{ov2} во всех режимах фиксируется установкой флага *TF2*.

2.6.3. Блок быстрого ввода-вывода МК SAB 80C515

Важной характеристикой таймера *T/C2* является возможность его использования в качестве базового таймера 4-канального блока быстрого ввода/вывода HSIO. Структурная схема HSIO рассмотрена ранее (рис.1.10). В состав HSIO, кроме таймера T/C2, входит набор из 4-х многофункциональных 16-разрядных регистров *CC1 – CC3*, *CRC*. Названные регистры используются при выполнении функций сравнения с текущим временем (в режиме быстрого вывода и при формировании ШИМ-сигналов), захвата (при фиксации времени наступления события в режиме быстрого ввода) и перезагрузки таймера *T/C2* (только регистр *CRC*). Блок HSIO имеет 4 входа захвата внешних событий и 4 канала сравнения. Входы-выходы событий одного канала объединены общим

многофункциональным выводом канала. В качестве входов/выходов каналов HSIO используются многофункциональные выводы порта *P1*:

P1.0/ $\overline{INT3}$ /CRC – вход внешнего прерывания 3 / выход 0 HSO / вход 0 HSI
P1.1/INT4/CC1 – вход внешнего прерывания 4 / выход 1 HSO / вход 1 HSI
P1.2/INT5/CC2 – вход внешнего прерывания 5 / выход 2 HSO / вход 2 HSI
P1.3/INT6/CC3 – вход внешнего прерывания 6 / выход 3 HSO / вход 3 HSI

Функционирование блока HSIO поддерживают регистры SFR (табл.5).

Регистры SFR, поддерживающие работу блока HSIO — Таблица 5

Регистр SFR	Назначение регистра SFR	Адрес
TH2/TL2	Старший/младший байт 16-разрядного регистра таймера T/C2	CDh/CCh
T2CON	Регистр управления таймера T/C2	C8h
CCEN	Compare/Capture ENable register (Регистр разрешения сравнения/захвата)	C1h
CC1 CCH1/CCL1	Compare/Capture 1 - 16-разрядный регистр сравнения/захвата 1 Старший (CCH1)/младший (CCL1) байт регистра CC1	C3h/C2h
CC2 CCH2/CCL2	Compare/Capture 2 - 16-разрядный регистр сравнения/захвата 2 Старший (CCH2)/младший (CCL2) байт регистра CC2	C5h/C4h
CC3 CCH3/CCL3	Compare/Capture 3 - 16-разрядный регистр сравнения/захвата 3 Старший (CCH3)/младший (CCL3) байт регистра CC3	C7h/C6h
CRC CRCH/CRCL	Compare/Reload/Capture - регистр сравнения/перезагрузки/захвата Старший (CRCH) /младший байт (CRCL) регистра CRC	CBh/CAh
IRCON	Interrupt control register – регистр управления прерываниями	C0h

Каналы блока HSIO могут работать в двух основных режимах: в режиме захвата и в режиме сравнения. Режимы работы каналов задаются при программировании управляющих бит bit_{2i+1} и bit_{2i} (i=0-3) регистра *CCEN* (рис. 2.33). Кодирование режимов каналов HSIO выполняется в соответствии с табл.6. Режимы работы каналов однозначно определяют назначение и использование многофункциональных 16-разрядных регистров *CC1 – CC3, CRC*.

CC3		CC2		Режим CC1		Режим CC0		CCEN
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	адрес C1h

Рис. 2.33. Формат регистра CCEN

Биты *bit0* и *bit1* задают режим использования регистра CRC, биты *bit2* и *bit3* - регистра *CC1*, биты *bit4* и *bit5* - регистра *CC2*, биты *bit6* и *bit7* - регистра *CC3*.

Кодирование режимов каналов блока HSIO — Таблица 6

bit 2i+1	bit 2i	Режим использования регистров CRC и CCx
0	0	Захват/сравнение запрещены
0	1	Захват внешнего события (быстрый ввод)- режим захвата 1
1	0	Сравнение разрешено (быстрый вывод или ШИМ)
1	1	Режим чтения "на лету" (запись в регистр CC_x) - режим захвата 2

Нетрудно заметить, что биты bit_{2i+1} и bit_{2i} (i=0-3) не определяют конкретного подрежима сравнения. Собственно кодирование режима сравнения осуществляется с помощью бита *T2CM* в регистре *T2CON*, при этом кодирование подрежима является общим для всех каналов блока HSIO.

Каждый из основных режимов работы блока HSIO имеет два подрежима. Подрежимами *захвата* являются режим быстрого ввода и режим «чтения на лету». В подрежимах *сравнения* собственно быстрый вывод и осуществляется формирование ШИМ-сигналов. В обоих режимах высокоскоростного ввода-вывода текущее значение времени (сравниваемое или регистрируемое) формируется на выходе таймера *T/C2*, который осуществляет счет входных импульсов.

Рассмотрим режимы работы блока HSIO более подробно.

Режимы сравнения блока HSIO. В режимах сравнения 16-разрядное значение, загруженное в любой из регистров CRC или CC1 – CC3, сравнивается с текущим содержимым регистров TH:TL таймера. В момент равенства этих значений генерируется внутренний сигнал «compare». В зависимости от способа активизации выходного события на внешнем выводе канала блока HSIO различают два режима сравнения: 0 и 1. Режим сравнения 0 является режимом формирования ШИМ-сигналов, а режим сравнения 1 – режимом формирования внешнего события заданного вида.

Работу канала блока HSIO в режиме сравнения поддерживают 16-разрядный цифровой компаратор и набор управляющих схем, активизирующих требуемое выходное событие на выходе порта P1 в момент равенства значений в регистре CCx и в счетчике T/C2. Структура одного разряда канала сравнения показана на рис. 2.34.

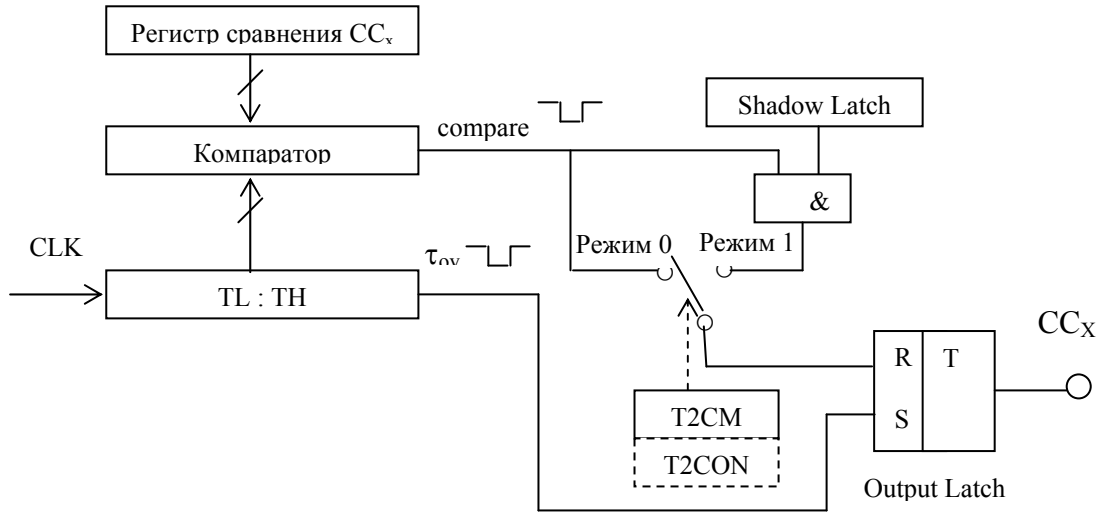
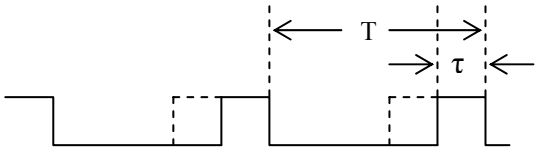


Рис. 2.34. Структура одного разряда канала сравнения

Формирование ШИМ-сигналов (режим сравнения 0) .

ШИМ-сигналы – это сигналы фиксированной частоты с регулируемой скважностью (переменной длительностью сигнала). ШИМ – широтно-импульсная модуляция. Вид ШИМ-сигнала показан на рис. 2.35.



$\frac{T}{\tau}$ — Скважность,

Пунктиром показана регулируемая длительность импульса

Рис. 2.35. ШИМ-сигнал

Формирование ШИМ-сигналов в современных МК чаще всего осуществляется с помощью так называемых ШИМ-генераторов. Основными узлами ШИМ-генератора являются таймер, запрограммированный для работы в режиме программируемого делителя частоты, регистр задания скважности, *n*-разрядный цифровой компаратор и формирователь выходного ШИМ-сигнала (в большинстве случаев – это простейший RS-триггер). Генератор импульсов требуемой частоты можно реализовать на базе таймера T/C2, запрограммировав его для работы в режиме неуправляемого таймера с автоматической перезагрузкой. Структурная схема ШИМ-генератора на базе таймера T/C2 представлена на рис. 2.36. ШИМ-сигнал формируется на выходе P_{1x} соответствующего канала сравнения блока HSIO.

При работе таймера на выходах счетчика таймера TH2:TL2 формируется цифровой пилообразный сигнал (рис. 2.37). С помощью *n*-разрядного цифрового компаратора этот сигнал сравнивается со значением $N_{зад}$, загруженным в регистр задания скважности, и в момент равенства формируется сигнал $\tau_{сравн}$, устанавливающий RS-триггер. Импульс переполнения τ_{ov} таймера сбрасывает RS-триггер. Частота поступления

Рис. 2.36. Структурная схема ШИМ-генератора

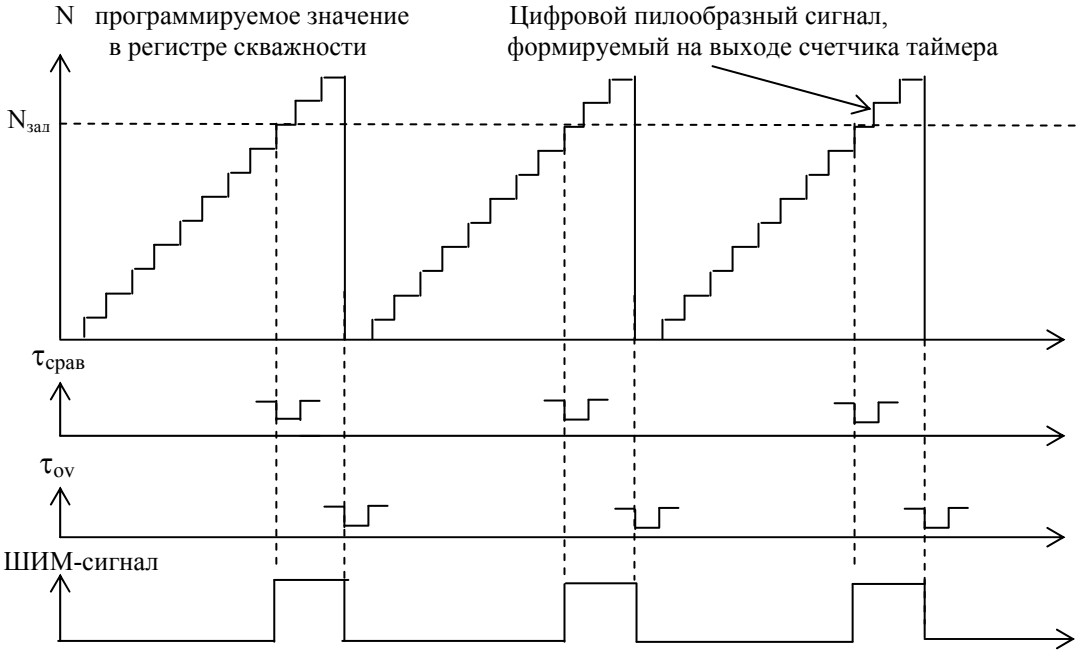


Рис. 2.37. Временная диаграмма работы ШИМ-генератора

импульсов τ_{ov} определяет частоту f_{pwm} формируемого ШИМ-сигнала. Длительность импульса (и скважность) ШИМ-сигнала определяется значением в регистре задания скважности. При изменении содержимого этого регистра скважность формируемых ШИМ-сигналов меняется. Временная диаграмма работы ШИМ генератора показана на рис. 2.37.

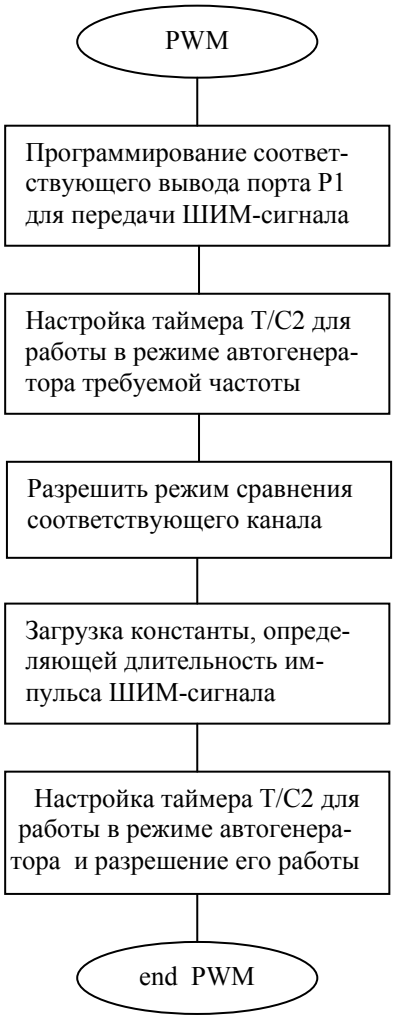
В качестве примера рассмотрим формирование ШИМ-сигнала частоты 125 Гц и длительностью импульса 128 мкс на выходе 3-го канала блока HSIO. Схема алгоритма программы формирования ШИМ-сигнала и ее текст приведены на рис. 2.38.

Формирование ШИМ-сигнала фиксированной скважности, рассматриваемое в примере, имеет ограниченное использование. Значительно более распространены ШИМ-сигналы с переменной скважностью, которые используются при формировании управляющих сигналов с изменяемой амплитудой. Управляя длительностью генерируемых ШИМ-сигналов, нетрудно сформировать управляющие сигналы требуемой интенсивности или мощности. Такие сигналы обычно используют при работе с объектами с аналоговым управлением.

Кодовое (цифровое) управление длительностью ШИМ-сигнала обычно реализуется с помощью задатчиков с цифровым выходом, например, с помощью АЦП. Будем считать, что код управления скважностью $N_{\tau pwm}$, формируемый АЦП, размещается в ячейке с именем *pwm_var*. Поскольку разрядность типовых АЦП обычно равна 8 разрядов, для корректной работы ШИМ-генератора на базе 16-разрядного счетчика содержимое *pwm_var* необходимо масштабировать путем преобразования 8-разрядного значения АЦП (*pwm_var*) в 16-разрядный код управления скважностью $N_{\tau i}$

$$N_{\tau i} = i \cdot \kappa_{м} ,$$

здесь i - управляющий код, формируемый на выходе АЦП (i = 0 – 255),
$\kappa_м$ - масштабирующий множитель.



а)

```
                                org 8100h
pwm:
        orl P1,#00001000b ;Настройка порта P1.3 для вывода ШИМ
        mov T2CON,#0      ;Остановка таймера T/C2
;Настройка таймера T/C2 для работы в режиме автогенератора:
;F_PWM = f_CR/12 /N, где N – расчетное значение делителя. Для суммиру-
;ющего счетчика таймера T/C2 константа перезагрузки регистра CRC
;вычисляется по формуле: N_загр = FFFFh - N = FFFFh - f_CR/12/ f_PWM.
;При f_CR=12МГц для частоты f_PWM = 125 Гц N_загр = E0C0h
;Загрузка константы N_загр в регистр CRC для задания частоты 125Гц
        mov CBh,#E0h
        mov CAh,#C0h
;Разрешение работы 3-го канала HSIO в режиме сравнения
        mov  C1h,#10000000b  ;Программирование регистра CCEN
;Загрузка константы Nзагр1, определяющей длительность τ в регистр
;CC3.  N_загр1 = FFFFh – Nτ = N̄τ. Для τ=128 мкс N_загр1 = FF7Fh;
        mov C7h,#FFh
        mov C6h,#7Fh
;Программирование таймера T/C2 для работы в режиме неуправляемо-
;го таймера с автоматической перезагрузкой по переполнению и
;разрешение его работы
        mov T2CON,#00010001b  ;задание режима генератора
        ret
```

б)

Рис. 2.38. Алгоритм (а) и текст программы (б) формирования ШИМ-сигнала

При вычислении масштабирующего множителя $\kappa_м$ будем учитывать, что минимальное значение кода с выхода АЦП (оно равно 0) должно соответствовать минимально допустимому значению $N_{загрмин}$ (в ШИМ-генераторе $N_{загрмин}$ = FFFFh), а максимальное значение кода на выходе АЦП - значению $N_{загрмакс}$ = FFFFh - $N_{загрf}$, которое не может быть меньше значения $N_{загрf}$, определяющего частоту f_{pwm}.

$$\kappa_м = (N_{загрмин} - N_{загрмакс})/255 = N_{загрf} / 255$$

В общем случае значение $\kappa_м$, вычисленное по этой формуле, является двухбайтным числом, содержащим целую и дробную части. Целая часть

κ_{mint} является старшим байтом, а дробная часть κ_{mfract} – младшим байтом этого 16-битного числа.

Значение $N_{\tau i}$, определяющее длительность формируемого ШИМ-сигнала, является результатом умножения константы управления скважностью *pvm_var*, формируемой на выходе АЦП, на масштабирующий коэффициент $\kappa_{\text{м}}$. При грубом масштабировании исходное значение *pvm_var* умножается на целую часть коэффициента $\kappa_{\text{м}}$ (κ_{mint}). Однако в этом случае возможна большая погрешность масштабирования. Поэтому в большинстве случаев вычисление $N_{\tau i}$ выполняют с использованием как κ_{mint}, так и κ_{mfract}. Общий алгоритм вычисления $N_{\tau i}$ можно представить следующей схемой.

1. Значение *pvm_var* умножается на κ_{mint}. Результат – 16-битовое число является базовым значением вычисляемого параметра $N_{\tau i}$.

2. Значение *pvm_var* умножается на κ_{mfract}. Старший байт 16-разрядного результата является добавкой (поправкой), которую необходимо учитывать при вычислении $N_{\tau i}$.

3. 16-битовое число 1-го произведения суммируется со старшим байтом 2-го произведения. Результат 16-разрядного сложения является значением $N\tau_i$. Инверсное значение $N_{\tau i}$ загружаетсяе в регистр задания скважности CCx (x=1-3).

После инициализации регистра задания скважности (CC1 – CC3) формирование ШИМ-сигнала выполняется блоком HSIO автономно, не требуя для своей поддержки каких-либо ресурсов центрального процессора. Программная перезагрузка регистра задания скважности может выполняться асинхронно, т.е. без привязки к текущему состоянию канала формирования ШИМ-сигнала. Изменение задания скважности ШИМ-сигнала по любому из каналов может выполняться как в фоновой программе, формирующей управляющее воздействие, так и в процедуре обслуживания прерывания по переполнению таймера T/C2 что более предпочтительно.

Для формирования требуемой частоты ШИМ-сигналов можно применить другую реализацию генератора, используя прерывания по переполнению T/C2 для перезагрузки счетчика таймера расчетным значением частоты N_{pwm}. При такой реализации генератора регистр CRC не требуется, и для формирования ШИМ-сигналов возможно использование всех 4-х каналов блока HSIO. Особенности реализации генератора требуемой частоты с использованием прерывания по переполнению таймера для перезагрузки счетчика таймера расчетным значением частоты рассматриваются ниже.

Для цифро-аналогового преобразователя ШИМ-сигнал буферизуется повторителем и фильтруется с помощью фильтра низкой частоты ФНЧ(активного или пассивного). С целью уменьшения выходного сопротивления преобразователя преобразованный непрерывный выходной

сигнал может быть дополнительно пропущен через аналоговый повторитель на базе операционного усилителя. Схема простейшего ЦАП на основе ШИМ-генератора приведена на рис. 2.39.

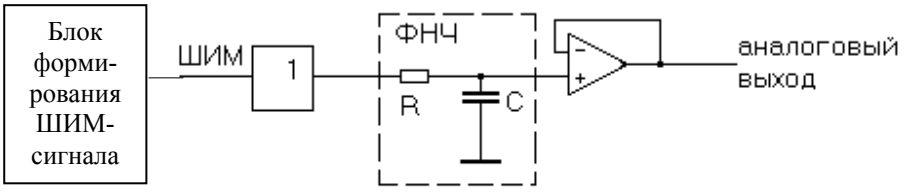


Рис. 2.39. Схема простейшего ЦАП на основе ШИМ-генератора

Области применения ШИМ-сигналов весьма разнообразны. Наиболее простое и в то же время эффективное применение ШИМ-сигнала состоит в реализации с его помощью цифро-аналогового преобразования без использования дорогостоящих схем ЦАП лестничного типа.

ЦАП на основе ШИМ-генератора обладает достаточно высокими точностными характеристиками, но имеет относительно низкое быстро действие. Подобные ЦАП удобно использовать при невысоких требованиях к быстродействию формирования аналоговых сигналов, например для вывода информации о значении некоторой переменной на стрелочный индикатор или для задания медленно меняющихся аналоговых управляющих воздействий. Другим примером широкого использования ШИМ-сигналов является цифровое управление ключами силовых преобразователей для двигателей постоянного и переменного тока.

Режим быстрого вывода (режим сравнения 1). Данный режим работы канала HSIO используется при формировании выходных событий заданного вида на выводе порта P1 в требуемый момент времени.

Режим сравнения 1 устанавливается программно путем кодирования бита T2CM регистра T2CON (T2CM=1) и бит b_{2i+1} , b_{2i} (i=0…3) регистра CCEN. В этом режиме выходные схемы каналов P1.0 – P1.3 переконфигурируются. Они состоят из двух защелок: скрытой (Shadow Latch) и выходной (Output Latch), связанных друг с другом через вентиль, управляемый сигналом сравнения *compare* (см. рис.2.34). После запуска таймера T/C2 содержимое регистров CCx и TH2/TL2 сравнивается, и в момент равенства формируется сигнал сравнения, с помощью которого требуемое значение выходного сигнала из скрытой защелки переписывается в выходную.

Для формирования выходного события заданного вида пользовательская программа должна обеспечить выполнение следующей последовательности действий:

– запретить работу таймера;

– записать в выходную защелку соответствующего канала HSIO исходное значение выходного сигнала;

– инициализировать регистр CCx (x=0-3), задавая время выходного события;

– запрограммировать выбранный канал HSIO для работы в режиме быстрого вывода;

– записать в Shadow Latch требуемое значение выходного сигнала.

Операция записи в скрытую защелку выполняется теми же командами, что и установка или сброс выходной защелки (*setb P1.x* или *clr P1.x*). Скрытая защелка становится приемником этих команд после программирования режима быстрого вывода.

После запуска таймера содержимое регистров *CCx* и *TH2/TL2* сравнивается, и в момент равенства формируется сигнал «compare», с помощью которого требуемое значение выходного сигнала из скрытой защелки переписывается в выходную. Рассмотренный режим сравнения является более быстрым (точным), чем непосредственное переключение вывода порта в программируемые моменты времени.

Режимы захвата блока HSIO

Захват (*capture*) события заданного вида – это фиксация времени его наступления в многофункциональном регистре *CCx* (*x*=0-3). Входное событие может задаваться программно или аппаратно. Выбор режима захвата выполняется программно при соответствующем кодировании управляющих бит bit_{2i} и bit_{2i+1} (*i*=0-3) регистра CCEN. Структура одного канала блока HSIO в режиме захвата представлена на рис. 2.40.

При аппаратном захвате, называемом *быстрым вводом*, в регистре *CCx* (*x* = 0-3) фиксируется время наступления внешнего события. В этом режиме с помощью входного детектора HSIO выявляется событие заданного вида и формируется внутренний сигнал *capture*, реализующий перепись текущего содержимого таймера в регистр CCx соответствующего канала и устанавливающий флаг прерывания *IEXx* (*x*=0-3).

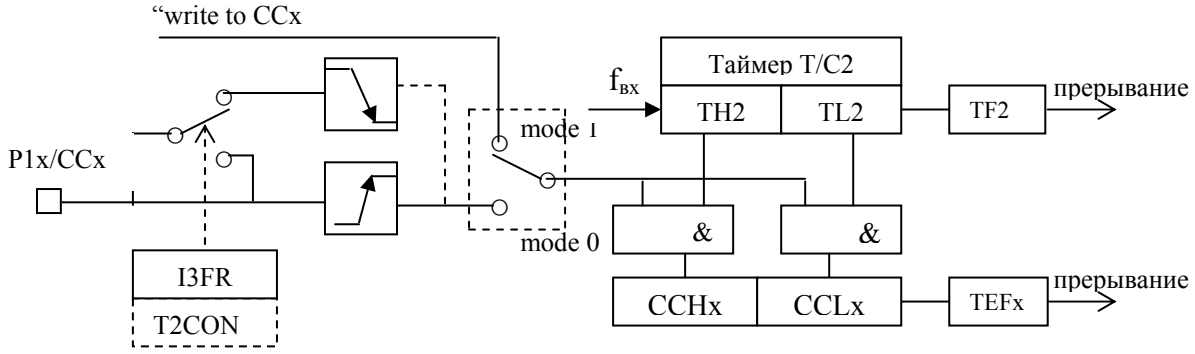


Рис. 2.40. Структура одного канала блока HSIO в режиме захвата

Для фиксации момента поступления внешнего события специальная подпрограмма не требуется, поэтому этот режим и называют быстрым вводом.

Захват события на выводах *P1.1*/*CC1* - *P1.3/CC3* (каналами 1-3) происходит при изменении уровня входного сигнала из 0 в 1. В канале 0 (с регистром *CRC*) тип входного воздействия, определяющего захват, кодируется с помощью бита *I3FR* регистра *T2CON*. Захват внешнего события на выводе *P1.0*/*CC0* может происходить при изменении входного сигнала как из 0 в 1, так и из 1 в 0.

Программный захват, называемый «чтение на лету», происходит при выполнении команды записи 8-битного значения в регистр *CCLx*. В этом режиме в качестве сигнала *capture* используется внутренний сигнал «*write to CCLx*», формируемый при выполнении указанной команды. Конкретное значение пересылаемого байта в регистр *CCLx* в этом режиме значения не имеет, так как команда записи используется только для инициирования сигнала «*write to CCLx*».

Во многих случаях только режим «чтение на лету» обеспечивает правильное чтение 16-битного значения счетчика таймера *T/C*2 без остановки счета. Это связано с тем, что прочитать содержимое таймера *T/C*2 по 8-разрядной внутренней шине можно только с помощью двух последовательно выполняемых команд *mov*. Однако асинхронно работающий таймер в процессе выполнения этих команд может изменить свое значение, и в результате произойдет ошибочное чтение. Для исключения подобных ошибок и предусмотрен режим "чтение на лету".

В режиме «чтение на лету» при инициализации захвата флаг *IEXx* (*x*=0-3) не устанавливается, и запрос прерывания не формируется.

2.6.4. Сторожевой таймер WDT МК SAB 80C515 предназначен для контроля правильности функционирования программного обеспечения и борьбы с программными сбоями системы. WDT-таймер реализован на базе 16-битного счетчика, который инкрементируется с определенной частотой. Функционирование WDT поддерживают отдельные биты управления нескольких регистров SFR – бит *WDT* регистра *IEN0*, бит *SWDT* регистра *IEN1* и бит *WDTS* регистра *IP0*. После аппаратной начальной установки *Reset* работа WDT-таймера (рис. 2.41) запрещена, его счетчик содержит 0000h.

WDT-таймер начинает счет при установке бита *SWDT*, при этом в регистре *IP0* автоматически устанавливается статусный бит *WDTS*. После запуска работу WDT-таймера программно остановить нельзя, но возможно его программное обнуление с помощью двух последовательно выполняемых команд: загрузки бита очистки *wdt* (*IEN0.6*) и стартового бита *swdt* (*IEN1.6*). Двойная операция обнуления WDT-таймера, реализуется командами *setb wdt* и *setb swdt*,.

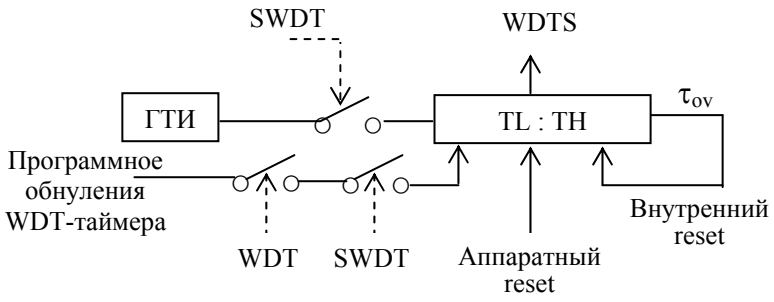


Рис. 2.41. Упрощенная функциональная схема WDT-таймера

При правильной работе ПО она обеспечивает защиту от случайного сброса таймера WDT. При нарушении в работе ПО обнуление WDT-таймера не происходит, и формируемый при переполнении таймера сигнал внутреннего аппаратного сброса осуществляет рестарт программы. В отличие от внешнего аппаратного сброса *Reset* при внутреннем сбросе статусный бит *WDTS* не сбрасывается в течение 4-х машинных циклов. При необходимости значение бита *WDTS* можно проанализировать и использовать для программной идентификации причины сброса.

В режиме холостого хода. (Idle-режиме) сторожевой таймер WDT прекращает свою работу.

2.7. Система прерываний МК SAB 80C515

Работу встраиваемых систем в реальном времени, наряду с таймерами, поддерживает система прерываний. Она является универсальным интерфейсом, обеспечивающим автоматический запуск различных процедур обслуживания по запросу от внешних или внутренних устройств микроконтроллерной системы. Система прерываний МК SAB 80C515 реализована с помощью размещенного на кристалле контроллера прерываний. Последний представляет собой устройство, устанавливающее однозначное соответствие между запросом прерывания и адресом подпрограммы обработки этого запроса.

Система прерываний МК SAB 80C515 объединяет 6 внутренних и 8 внешних источников запросов. Она является многовекторной и многоуровневой. Каждому источнику запроса прерываний определен адрес (вектор) обработчика. Контроллер прерываний устанавливает последовательность обработки одновременно поступающих запросов, обеспечивая приоритетное обслуживание наиболее ответственных. При поступлении очередного запроса прерывания контроллер по адресу обработчика передает управление процедуре его обслуживания.

В многоуровневой системе МК SAB 80C515, в отличие от одноуровневых систем прерываний, возможно прерывание прерывания: запрос с более высоким уровнем приоритета может прервать обработку прерывания более низкого уровня.

Пары источников прерываний и приоритеты прерываний внутри уровня Таблица 6

Источник прерывания со старшим приоритетом в паре	Имя источника прерывания	Адрес вектора прерывания	Источник прерывания с младшим приоритетом в паре	Имя источника прерывания	Адрес вектора прерывания	Приоритет пары источников прерываний	
Внешнее прерывание 0	IE0	0003h	Прерывание АЦП	IADC	0043h	0	Высший
Переполнение таймера 0	TF0	000Bh	Внешнее прерывание 2	IE2	004Bh	1	
Внешнее прерывание 1	IE1	0013h	Внешнее прерывание 3	IE3	0053h	2	
Переполнение таймера 1	TF1	001Bh	Внешнее прерывание 4	IE4	005Bh	3	
Прерывание SP	TI или RI	0023h	Внешнее прерывание 5	IE5	0063h	4	
Переполнение таймера 2 или внешняя перезагрузка	TF2 EXF2	002Bh	Внешнее прерывание 6	IE6	006Bh	5	Низший

Все источники прерываний МК SAB 80C515 объединены попарно, образуя 6 пар. В табл.6 указаны адреса векторов прерывания каждого из источников. Каждой паре источников прерывания программно можно определить один из четырех «глобальных» уровней обслуживания (табл.7). Задание «глобального» уровня обслуживания выполняется путем установки/ сброса пары бит *IP0.x* и *IP1.x* ($x = 0 – 5$) в регистрах управления приоритетами *IP0* и *IP1* (рис. 2.42). Приоритеты пар источников прерываний в пределах одного глобального уровня фиксированы: приоритет первой пары – наивысший, а приоритет шестой пары – самый низкий. Внутри каждой пары источник прерывания, указанный слева имеет более высокий приоритет по сравнению со вторым источником пары. Приоритет любого запроса может быть повышен путем перевода этого запроса на глобальный уровень с более высоким уровнем.

Кодирование глобального уровня приоритета пары источников запросов прерывания Таблица 7

Значения бит		Задаваемый глобальный уровень приоритета
IP1.x	IP0.x	
0	0	Уровень приоритета 0 (низший)
0	1	Уровень приоритета 1
1	0	Уровень приоритета 2
1	1	Уровень приоритета 3 (высший)

IP0

–	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0

адрес A9h

IP1

–	–	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0

адрес B9h

Рис. 2.42. Форматы регистров IP0 и IP1

Внешние прерывания.

Внешние прерывания активизируют систему прерываний при поступлении сигнала заданного вида на соответствующий вход запроса прерывания *INTx* ($x = 0 - 6$). Семь входов внешних прерываний распределены по следующим выводам портов *P1* и *P3*: *INT0/P3.2, INT1/P3.3, INT2/P1.4, INT3/P1.0, INT4/P1.1, INT5/P1.2, INT6/P1.3*. При инициализации соответствующие разряды указанных портов должны быть установлены в 1. Восьмым источником внешнего прерывания является отрицательный перепад напряжения на выводе *P1.5/T2EX*. Сигнал прерывания, формируемый на этом выводе, управляет внешней перезагрузкой таймера 2. При выявлении указанного прерывания устанавливается флаг *EXF2* регистра *IRCON*.

Входы внешних прерываний опрашиваются в каждом машинном цикле. Тип сигнала прерывания (уровень или переход от одного уровня к другому) определяет требования к источнику прерывания.

Для распознавания внешнего прерывания, вызываемого уровнем, необходимо, чтобы источник удерживал активный уровень, по крайней мере, один машинный цикл. Флаг такого прерывания просто повторяет входной сигнал запроса, поэтому последний должен удерживаться до вызова программы обработки прерывания. Для предотвращения повторного вызова программы обработки прерывания сигнал запроса должен сбрасываться до завершения программы обработки прерывания.

Если внешнее прерывание активизируется перепадом, для его идентификации требуется, по крайней мере, два машинных цикла. Флаг прерывания по перепаду устанавливается, если в одном машинном цикле на входе внешнего прерывания фиксируется сигнал исходного уровня, а в следующем – сигнал уровня перепада. Сброс флага прерывания по перепаду выполняется аппаратно (автоматически) при вызове программ обработки прерывания.

Если прерывание разрешено, при поступлении запроса прерывания устанавливается флаг прерывания, и вызывается программа его обработки.

Разработчики микроконтроллера разместили флаги прерываний от внутренних и внешних источников периферийных внутренних устройств в 4-х регистрах блока SFR: *TCON*, *T2CON*, *SCON* и *IRCON*, не всегда придерживаясь функциональной принадлежности флага к соответствующему регистру SFR. Форматы и назначение отдельных бит регистров *TCON*, *T2CON* и *SCON* рассмотрены ранее. Формат регистра *IRCON* представлен на рис. 2.43.

IRCON

EXF2	TF2	IEX6	IEX5	IEX4	IEX3	IEX2	IADC

адрес C0h

Рис. 2.43 Формат регистра IRCON

Внешние прерывания $\overline{INT0}$ и $\overline{INT1}$ устанавливают флаги прерывания *IE0* и *IE1* в регистре *TCON*. Тип сигнала прерывания по этим входам определяется значением бита *ITx* (*x* = 0, 1) регистра *TCON*: при *ITx* = 1 прерывания активизируются низким логическим уровнем, а при *ITx* = 0 - отрицательным перепадом сигнала на соответствующем входе МК. Сброс флагов *IE0* и *IE1* зависит от вида прерывания, задаваемого битом *ITx*.

Внешние прерывания $\overline{INT2}$ и $\overline{INT3}$ устанавливают флаги прерывания *IEX2* и *IEX3* в регистре *IRCON*. Вид входного сигнала, устанавливающего флаг внешнего прерывания *IEXx* (*x* = 2, 3) определяется значением бита *IxFR* (Interrupt *x* Falling/Rising) регистра *T2CON*. При *IxFR* = 0 флаг *IEXx* (*x* = 2, 3) устанавливается по спаду входного сигнала (переходом из 1 в 0), а при *IxFR* = 1 – по фронту входного сигнала (переходом из 0 в 1). Кроме того, в режиме быстрого ввода, флаг IEX3 устанавливается при поступлении на вход CC0/P1.3 сигнала, вид которого определяется значением бита I3FR: при I3FR = 0 установка флага IEX3 и захват события осуществляется по спаду входного сигнала (переходом из 1 в 0), а при I3FR = 1 – по фронту входного сигнала (переход из 0 в 1).

Флаги IEX2 и IEX3 сбрасываются аппаратно при вызове программы обработки прерывания.

Внешние прерывания *P1.1/INT4, P1.2/INT5, P1.3/INT6* активизируются положительным перепадом входного сигнала на одноименном входе МК. Запоминание запросов названных прерываний осуществляется в регистре *IRCON* с помощью флагов *IEX4, IEX5, IEX6*.

В режиме быстрого ввода многофункциональные выводы МК *P1.1/CC1, P1.2/CC2, P1.3/CC3* используются для подключения источников внешних событий, при этом флаги *IEX4, IEX5, IEX6* фиксируют не запрос внешнего прерывания, а факт поступления входного события заданного вида.

Прерывания от внутренних периферийных устройств.

Источниками прерываний от внутренних периферийных устройств являются:

- сигналы переполнения таймеров *T/C0, T/C1* и *T/C2*;
- сигнал завершения аналого-цифрового преобразования *IADC*;
- сигналы прерываний последовательного порта *RI* и *TI,* формируемые при готовности порта к обмену.

Прерывания, обусловленные переполнением таймеров T/C0 и T/C1, фиксируются во флагах *TF0* и *TF1* регистра *TCON*. При обращении к программам обслуживания прерывания по переполнению таймеров *T/C0* и *T/C1* флаги *TF0* и *TF1* сбрасываются аппаратно.

Таймер T/C2 является источником двух типов внутренних прерываний, фиксируемых в регистре IRCON: прерывания, формируемого при установке флага *TF2* при переполнении таймера *T/C2* и прерывания, формируемого при установке флага *EXF2* при поступлении сигнала

внешней перезагрузки на вход *P1.5/T2EX*. Оба прерывания таймера *T/C2* объединены логической схемой ИЛИ и идентифицируются одним вектором прерывания. Флаги *TF2* и *EXF2* сбрасываются только программно при выполнении соответствующего обработчика, при этом сама программа определяет, какой из флагов *TF2* или *EXF2* вызвал прерывание.

Прерывание последовательного порта. Последовательный порт SP является источником двух внутренних прерываний, фиксируемых в регистре SCON: прерывания передатчика TI и прерывания приемника RI. Оба прерывания порта SP объединяются логической схемой ИЛИ и идентифицируются одним вектором прерывания. Флаги TI и RI сбрасываются в программе обслуживания прерывания последовательного порта, которая сама определяет, какой из флагов TI или RI вызвал прерывание.

Прерывание аналого-цифрового преобразователя генерируется при установке флага IADC в регистре IRCON. Сброс флага IADC выполняется программно.

Работу системы прерывания поддерживают два регистра разрешения прерываний IEN0 и IEN1 (рис. 2.44). С помощью установки/сброса отдельных бит указанных регистров каждый из перечисленных источников прерываний может быть индивидуально разрешен или запрещен. Установка/сброс бита EAL разрешает/запрещает все прерывания.

0AFh	0Aeh	0ADh	0Ach	0Abh	0Aah	0A9h	0A8h	IEN0
EAL	WDT	ET2	ES	ET1	EX1	ET0	EX0	Адрес 0A8h

EAL – бит разрешения или запрещения всех прерываний. Если EAL = 1, каждый источник прерывания индивидуально разрешается или запрещается установкой/сбросом бита разрешения собственного прерывания.

EX0, EX1 – биты разрешения внешних прерываний INT0, INT1. При сброшенном бите EXx (x = 0, 1) внешние прерывания INTx (x = 0, 1) запрещены.

ET0, ET1, ET2 – биты разрешения прерываний по переполнению таймеров T/C0, T/C1 и T/C2.

ES – бит разрешения прерываний последовательного порта SP

0BFh	0Beh	0BDh	0BCh	0BBh	0BAh	0B9H	0B8H	IEN1
EXEN2	SWDT	EX6	EX5	EX4	EX3	EX2	EADC	(адрес 0B8h)

Биты EX2 – EX6 – биты разрешения внешних прерываний INT2 – INT6.

Бит EXEN2 – бит разрешения/запрещения прерывания таймера 2 по сигналу внешней загрузки.

Биты WDT и SWDT– биты управления сторожевым таймером

Бит EADC – бит разрешения/запрещения прерывания аналого-цифрового преобразователя ADC

Рис. 2.44. Форматы регистров разрешения прерываний IEN1 и IEN0

Общая схема 4-уровневой системы приоритетных прерываний МК SAB 80C515/535 имеет вид, показанный на рис. 2.45.

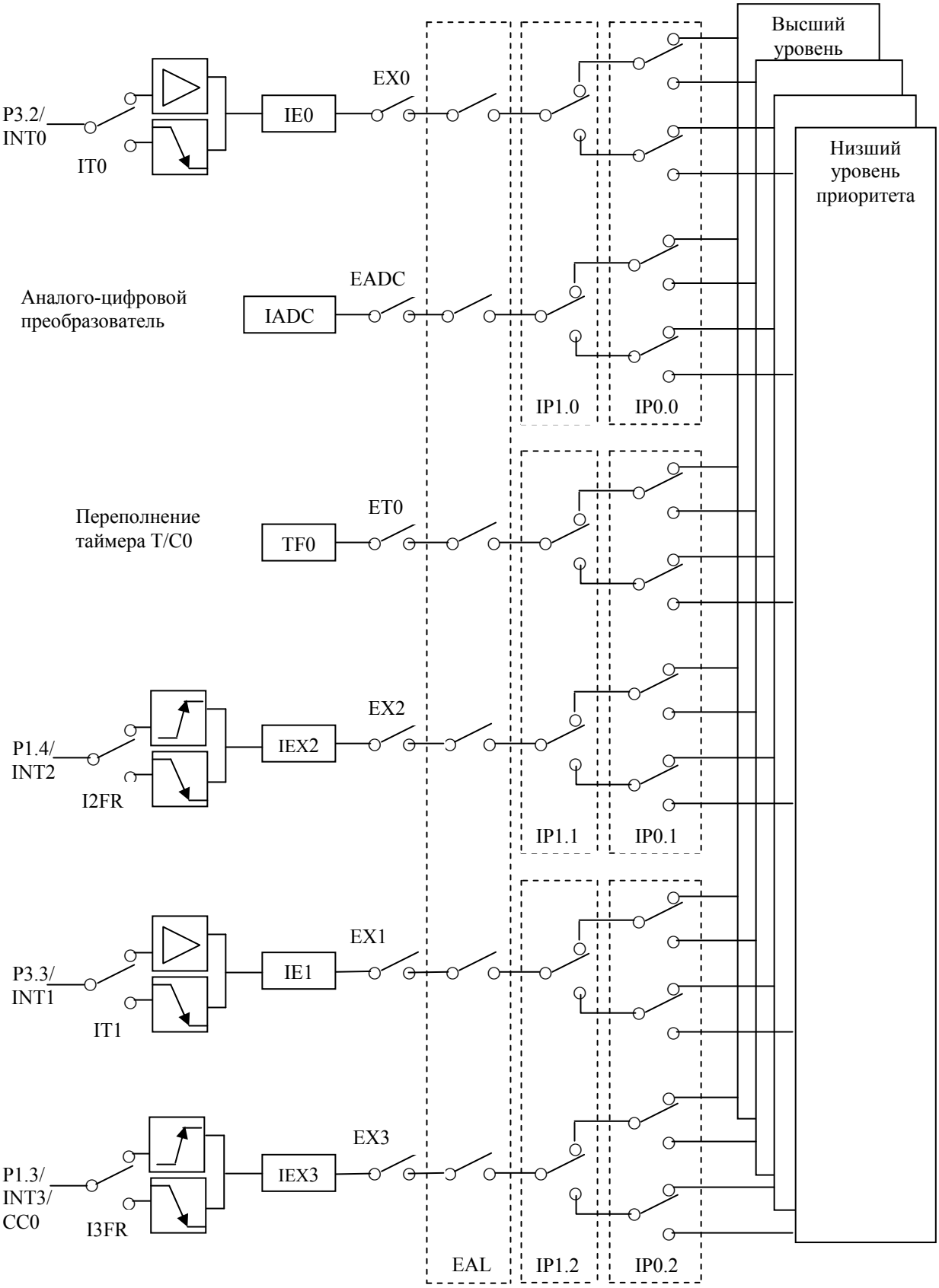


Рис. 2.45. 4-уровневая система приоритетных прерываний МК SAB 80C515/535

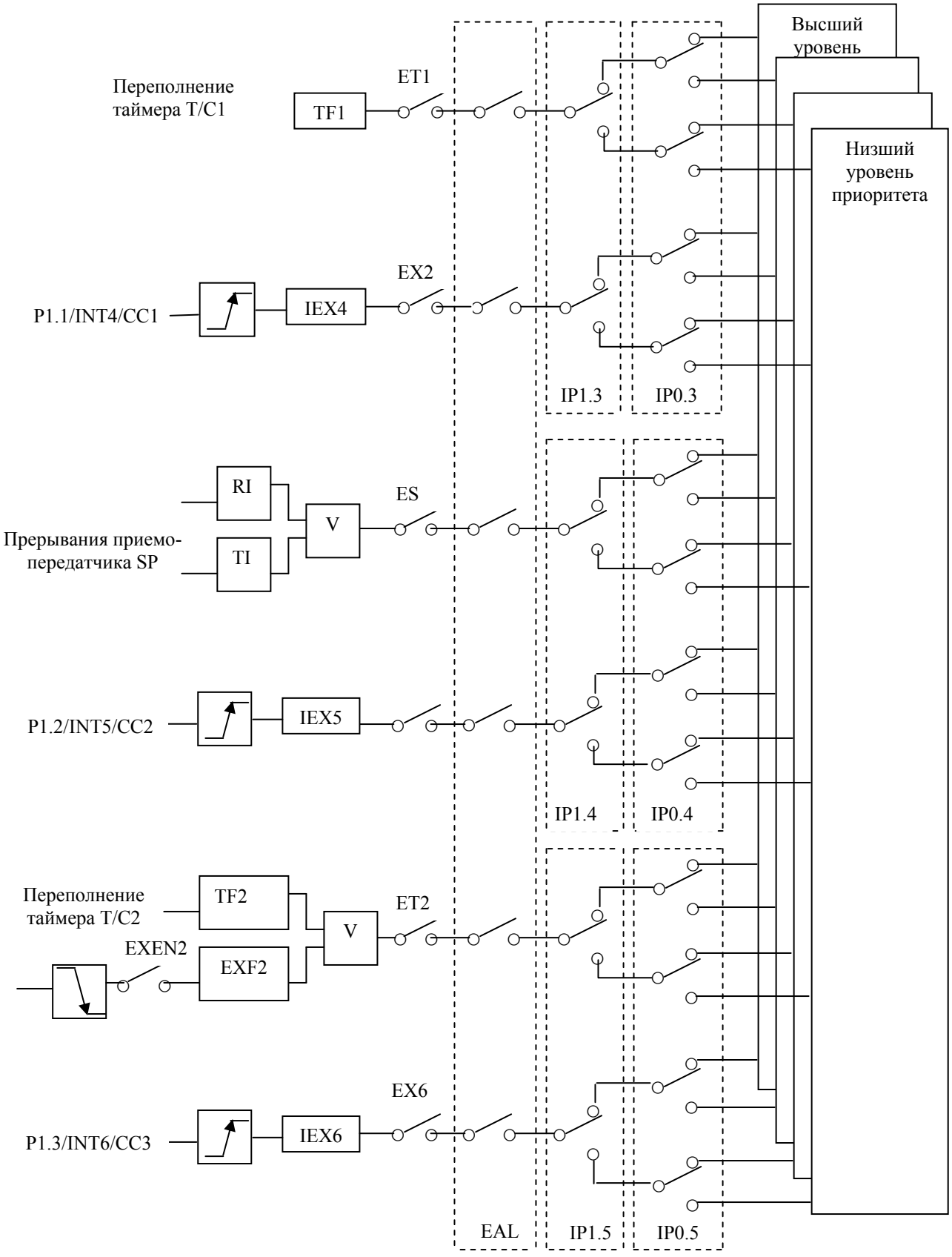


Рис. 2.45. 4-уровневая система приоритетных прерываний МК SAB 80C515/535 (продолжение)

Процедура обработки прерывания и время отклика.

Флаги прерывания опрашиваются в фазе S5P2 текущего машинного цикла, а анализируются в следующем машинном цикле. При наличии установленного флага процессор, обрабатывая запрос разрешенного прерывания, осуществляет аппаратный вызов команды *lcall*, с помощью которой реализуется вызов соответствующей программы обслуживания. Аппаратный вызов команды *lcall* может быть заблокирован любым из следующих условий:

1. В момент аппаратного вызова выполняется обработка прерывания с равным или более высоким уровнем приоритета. Другими словами, поступивший запрос может прервать обработку обслуживаемого прерывания только, если он имеет более высокий приоритет.

2. Машинный цикл опроса флага прерывания не является последним циклом выполняемой команды. Данное условие гарантирует, что выполняемая команда будет завершена до вызова программы обслуживания прерывания.

3. Выполняемая команда является командой *reti* или любой командой записи в регистры *IENx* или *IPx* ($x = 0, 1$). Указанное условие обеспечивает выполнение минимум еще одной команды текущей программы перед вызовом программы обслуживания прерывания.

Флаг прерывания, установленный во время действия блокировки аппаратного вызова команды *lcall* и сброшенный до снятия блокировки, не вызовет обслуживания соответствующего ему прерывания.

Последовательность машинных циклов, включая установку флага прерывания в машинном цикле *C1*, анализ состояния флагов в машинном цикле *C2*, аппаратный вызов команды *lcall* и ее исполнение в машинных циклах *C3* и *C4*, и машинный цикл начала программы обработки прерывания, показана на рис. 2.46.

Машинный цикл фиксации прерывания условно назван машинным циклом С1. В следующем машинном цикле С2 происходит анализ состояния флага прерывания и при отсутствии условий блокировки прерывания подготовка к формированию команды аппаратного вызова *lcall* обработчика прерывания. В машинных циклах С3 и С4 реализуется исполнение команды вызова обработчика по адресу вектора прерывания. Выполнение программы обработки прерывания начинается в машинном цикле С5.

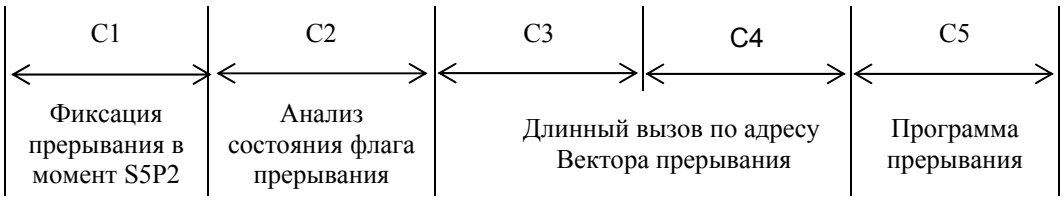


Рис. 2.46. Последовательность машинных циклов обработки запроса прерывания

Если в процессе обработки запроса прерывания до фазы S5P2 машинного цикла C3 поступит запрос более высокого уровня, то его обработка (аппаратный вызов команды *lcall*) будет начата в машинных циклах C5 и C6, а обработка прерывания меньшего приоритета будет заблокирована. На диаграмме (рис. 2.46) приведен самый быстрый возможный отклик на прерывания, когда цикл C2 является последним циклом команды, которая не является командой *reti* или командой записи в регистр *IENx* или *IPx* ($x = 0, 1$).

Аппаратно генерируемая команда *lcall*, предающая управление программе обслуживания прерывания, сохраняет в стеке содержимое программного счетчика PC и загружает в него адрес вектора прерывания. Выполнение программы обслуживания должно завершаться командой *reti.* Команда *reti* загружает в счетчик PC адрес возврата и восстанавливает состояние логики прерывания. Восстановление логики прерывания заключается в следующем: при переходе по вектору на программу обслуживания прерывания автоматически независимо от состояния бит разрешения прерываний запрещаются все прерывания с уровнем приоритета, равным уровню приоритета обслуживаемого прерывания и ниже. Блокировка прерываний сохраняется до выполнения команды *reti*, снимающей этот запрет. В соответствии с логикой работы системы прерываний реализация вложенных прерываний с равным уровнем приоритета невозможна. Заметим, что простая команда *ret* также возвращает управление прерванной программе, но при ее выполнении логика прерываний не восстанавливается, и система прерываний считает, что прерывание еще не закончилось.

Начальные адреса программ обслуживания прерываний расположены последовательно с интервалом 8 байт. Это означает, что если используются последовательно расположенные прерывания, например, внешнее прерывание 0 и переполнение таймера 0, и если первая программа обслуживания прерывания имеет длину более 7 байт, то в ней должен быть выполнен переход в другую область памяти, где эта программа может быть правильно завершена без наложения на начальный адрес следующей программы обслуживания прерываний.

2.8. Управление потребляемой мощностью

МК SAB 80C515 имеет два режима пониженного энергопотребления – *режим холостого хода Idle* и *режим микропотребления (Power Down mode)*, иногда называемый режимом выключенного напряжения питания. Особенности работы МК в этих режимах рассмотрены в подразд. 1.5.

Для организации работы в энергосберегающих режимах МК SAB 80C515 использует как аппаратную, так и программную поддержку. Это обеспечивает высокую эффективность и безопасность использования режимов энергосбережения.

С помощью сигнала $\overline{PE}$, подаваемого на одноименный вывод МК, разрешается ($\overline{PE} = 0$) или запрещается ($\overline{PE} = 1$) перевод МК в энергосберегающий режим. Благодаря наличию вывода $\overline{PE}$ можно исключить случайные переключения в энергосберегающие режимы, что недопустимо для ряда применений.

Вывод $\overline{PE}$ также можно использовать для предупреждения о нарушении работы источника питания или прекращения снабжения энергией. Последовательность действий в таких случаях возможна следующая: сигнал, фиксирующий падение уровня питания ниже допустимого вызывает обработчик, который сохраняет контекст программы в памяти. После этого активизируется сигнал $\overline{PE}$ ($\overline{PE} = 0$), и осуществляется программный перевод в режим энергосбережения.

Для перевода МК в любой из энергосберегающих режимов необходимо, чтобы сигнал $\overline{PE}$ на одноименном входе был равен 0.

Режимы пониженного энергопотребления задаются при установке соответствующих битов в регистре управления мощностью PCON (рис.2.47).

Переключение МК в режим Power-Down реализуется последовательностью из двух команд, следующих непосредственно друг за другом.

```
ORL PCON, #00000010b
ORL PCON, #01000000b
```

Первая команда, устанавливающая флаг *PDE* (*PCON.1*), не должна устанавливать флаг *PDS* (*PCON.6*). Следующая команда должна устанавливать старт-бит *PDS* и не должна воздействовать на флаг *PDE*.

7	6	5	4	3	2	1	0	PCON
SMOD	PDS	IDLS	–	GF1	GF0	PDE	IDLE	адрес 87h

Рис. 2.47. Формат регистра PCON

PDS (Power-Down Start bit) – старт-бит входа в режим микропотребления. Команда *orl pcon*, #01000000b, устанавливающая бит PDS, является последней командой перед входом в режим микропотребления.

PDE (Power-Down Enable bit) – бит разрешения старта режима микропотребления. При PDE = 1 старт режима микропотребления разрешен. Бит PDE можно установить командой *orl pcon*, #00000010b.

*IDLS (*IDle Start bit) – старт-бит входа в Idle-режим. Команда *orl pcon*, #00100000b, устанавливающая бит IDLS, является последней командой перед входом в режим холостого хода.

IDLE (IDle Enable bit) – бит разрешения старта Idle-режима. Бит IDLE можно установить командой, *orl pcon*, #00000001b.

GF1, GF0 – флаги общего назначения. Эти флаги могут использоваться для получения информации о режиме, в котором произошло прерывание (нормальном или Idle-режиме). Например, команда, активизирующая Idle-режим, может также установить один из этих флагов. Если Idle-режим завершается прерыванием, программа обработки прерывания может опросить биты *GF1, GF0*.

SMOD - бит, с помощью которого можно изменить скорость последовательного обмена. При SMOD = 1 скорость обмена последовательного канала в режимах 1, 2, 3 удваивается.

Одновременная установка бит *PDE* и *PDS* не инициирует режим микропотребления. Выход из режима осуществляется только аппаратным сбросом.

Переключение МК в Idle-режим также осуществляется с помощью последовательности двух команд, следующих непосредственно друг за другом.

```
ORL PCON, #00000001b
ORL PCON, #00100000b
```

Сначала должен быть установлен флаг *IDLE* (*PCON*.0), и следующей командой устанавливается флаг *IDLS* (*PCON*.5).

Одновременная установка бит *IDLE* и *IDLS* не инициализирует Idle-режим. Завершение Idle-режима может быть выполнено одним из двух способов: либо активизацией любого разрешенного прерывания и вызова подпрограммы обслуживания прерывания, завершающейся командой RETI (после выполнения команды RETI управление передается команде, следующей за командой установки старт-бита *IDLS*, которая вызвала режим Idle), либо выполнением аппаратного сброса МК, при котором сбрасываются биты IDLE и PDE регистра PCON и возобновляется выполнение программы с начального адреса.

Если вызываются оба режима энергопотребления Idle и Power-Down, осуществляется переход в режим Power-Down.

3. Особенности проектирования систем управления на базе МК. Характеристика инструментальных средств поддержки проектирования и отладки микроконтроллерных систем

Отличительной особенностью МК-систем является то, что в них аппаратные средства и программное обеспечение (ПО) существуют в форме неделимого аппаратно-программного комплекса. Под проектированием МК-системы понимается процесс создания ее прототипа [2,5]. В общем случае цикл разработки системы рассматривается как последовательность следующих этапов проектирования:

- анализ предметной области (анализ задачи);
- определение функций, возлагаемых отдельно на аппаратные средства и отдельно на программное обеспечение;
- выбор или разработка аппаратных средств контроллера;
- разработка прикладного ПО;
- комплексная настройка аппаратных средств и отладка ПО.

Кратко охарактеризуем названные этапы проектирования.

Анализ предметной области и определение функций, возлагаемых на аппаратные средства и отдельно на программное обеспечение. Основной целью этого этапа проектирования является выявление особенности объекта управления и выбор структуры системы управления.

Процесс разработки компьютерных систем не укладывается в простую схему применения оптимальных алгоритмов. Многие инженерные решения основываются на эвристических предположениях и практическом опыте. При их принятии учитываются соотношение стоимости и производительности с одной стороны, а также сложность аппаратного и программного обеспечения – с другой, причем диапазон альтернативных вариантов очень широк.

При проектировании МК-систем приходится решать достаточно сложную задачу оптимизации распределения функций между аппаратными средствами и программным обеспечением. Еще совсем недавно считалось, что аппаратные средства обеспечивают производительность, а программное обеспечение дешевизну устройства. Однако сейчас это правило далеко не всегда выполняется. Известно, что использование специализированных БИС и СБИС упрощает разработку и обеспечивает высокое быстродействие системы в целом, но сопряжено с увеличением стоимости, объема и потребляемой мощности проектируемого контроллера. Большой удельный вес ПО позволяет сократить число электронных компонент и стоимость аппаратных средств, но снижает быстродействие и увеличивает затраты на разработку и отладку ПО. Следует подчеркнуть, что возможность коррекции прикладного ПО продлевает «жизнь» контроллера, в то время как изменение конфигурации аппаратных средств в готовом изделии невозможно, и что аппаратные реализации контроллеров морально устаревают значительно быстрее. После выбора структуры контроллера (проектируемой системы) необходимо выбрать аппаратные средства проектируемой системы.

Выбор или разработка аппаратных средств контроллера.

Использование микропроцессоров, а затем и микроконтроллеров существенно изменило подход при выборе и разработке аппаратных средств. До использования МП/МК практически все устройства управления реализовывались как контроллеры с жесткой структурой. Особое значение при выборе аппаратных средств контроллера имеет выбор типа МК. В настоящее время на рынке представлен широкий спектр микроконтроллеров от 8-выводных PIC-микроконтроллеров до мощных 32-разрядных МК i386EX. Существенное влияние на выбор типа МК оказывает сложность системы управления. В соответствии с тенденциями построения современных систем управления (тенденцией интеграции отдельных подсистем МК и тенденцией рассредоточения аппаратных средств МК) систему управления можно реализовать на базе одного мощного МК или использовать для построения системы несколько менее мощных МК, объединенных в управляющую сеть. В общем случае решение о выборе аппаратных средств микроконтроллерной системы должно приниматься на основании комплексного сравнения основных

технико-экономических параметров МК, важнейшими из которых являются:

- *Разрядность*. Современные МК имеют разрядность 4, 8, 16 , 32 разряда. 4- и 8-разрядные МК в большинстве случаев используются при построении относительно несложных контроллеров и систем, не требующих значительных объемов вычислений. Для решения сложных задач управления производительности 8-разрядных МК и их вычислительной мощности может оказаться недостаточно. Такие задачи решают с помощью более мощных 16- и 32-разрядных МК. В целом, чем больше разрядность МК, тем он мощнее и дороже.
- *Производительность* характеризуется количеством вычислительной работы, выполняемой в единицу времени. Неправильно оценивать производительность МК по числу команд, выполняемых за такт. Такая оценка часто оказывается необъективной, так как не учитывает особенности и состав команд различных МК. В настоящее время для оценки производительности МК используют их тестирование специальными смесями (наборами) команд.
- *Количество линий ввода/вывода*. В принципе, чем больше линий ввода/вывода (портов параллельной информации) в составе МК, тем шире его функциональные возможности при обменах информацией с внешними устройствами. Однако МК с большим числом портов ввода/вывода имеют более высокие массогабаритные показатели и повышенное энергопотребление.
- *Состав периферийных устройств*. При выборе типа МК учитывается, нужна ли сеть (т.е. сколько последовательных портов должен содержать МК), нужны ли устройства быстрого ввода-вывода, нужен ли АЦП и т.п.
- *Потребляемая мощность* и массогабаритные показатели.
- *Стоимость*. Стоимость МК варьируется в зависимости от сложности его организации (разрядности, наличии и типе внутренней памяти программ, составе внутренней периферии и пр.) и лежит в пределах от 1 до нескольких десятков долларов.
- *Диапазон рабочих температур*. Данный параметр зависит от типа исполнения (0…+70°С при коммерческом исполнении, -40°...+85°С при исполнении в индустриальном стандарте, -40°...+125°С для работы в автомобильном стандарте и -55°...+125°С в военном).
- *Наличие средств поддержки разработки* (отладчиков, трансляторов и т.п.).
- *Способ программирования ПЗУ*. Современные МК могут содержать масочное, однократно программируемое или перепрограммируемое внутреннее ПЗУ. В некоторых моделях МК внутреннее ПЗУ может отсутствовать. Среди МК с перепрограммируемым ПЗУ перспективны

модели с памятью типа Flash, дополненные стандартным последовательным интерфейсом SPI. Такие МК допускают возможность многократного изменения программного кода во внутренней памяти программ в изготовленной системе.

При прочих равных условиях, когда основные характеристики микроконтроллеров разных типов удовлетворяют требованиям решаемой задачи, в первую очередь учитываются стоимостные характеристики, а также наличие средств поддержки проектирования и опыт работы разработчика с МК выбранного типа. Нередки случаи, когда разработчик при выборе МК в основном руководствуется своим опытом и возможностями в конкретный момент времени – «выбирает то, что знает, и то, что может выбрать». Только в случае очевидной невозможности использования конкретного типа МК для решения задачи (часто ошибочное решение выявляется только при комплексной отладке системы, которая оказывается неработоспособной) разработчик проводит анализ и целенаправленный выбор типа МК.

В принципе микроконтроллеры позволяют создавать функционально полные устройства управления при минимальном внешнем окружении. В пределе подобные устройства могут состоять лишь из одной микросхемы и кварцевого резонатора, а некоторые и без внешнего кварца. Однако в большинстве случаев разработка аппаратных средств систем управления на основе МК является более сложным процессом, при этом применение микропроцессоров и микроконтроллеров значительно изменило процесс разработки аппаратных средств. Суть заключается в следующем: микроконтроллер представляет собой логический автомат с высокой степенью детерминированности связей, который допускает сравнительно небольшое число вариантов его системного включения. Поэтому для сокращения сроков разработки типовых аппаратных средств ядро любой микроконтроллерной системы (а это сам МК, внешние микросхемы ПЗУ и ОЗУ, интерфейсные БИС, схемы синхронизации и системного управления), часто оформляется в виде универсальной одноплатной конструкции. Такая конструкция легко встраивается в контур управления объектом или процессом. На печатной плате конструкции обычно предусматриваются гнезда для установки дополнительных БИС пользователя. Благодаря их наличию универсальный одноплатный контроллер можно дополнить набором специфических узлов проектируемой системы, например схемами оптронной развязки, микросхемой ЦАП и пр. При использовании универсального одноплатного контроллера процесс разработки аппаратных средств проектируемой системы существенно упрощается и, более того, исключается необходимость подготовки конструкторской и эксплуатационной документации на самую сложную часть контроллера. На разработчика лишь возлагается задача проектирования специализированных схем

сопряжения ядра системы с датчиками и исполнительными механизмами объекта. При необходимости проектирования нетиповых устройств сопряжения с объектом применяются специализированные системы проектирования аппаратных средств, выполненные в виде пакетов программ для ВМ общего назначения. Системами проектирования аппаратных средств являются программные пакеты OrCad, PCAD, Protel, DesignLab, MicroCap, Proteus и др. Пакет Proteus интересен тем, что позволяет проводить моделирование синтезированных схем совместно с моделированием поведения микроконтроллера. Это позволяет выявить и устранить большинство ошибок еще до начала тестирования реальной аппаратуры. Различные пакеты автоматического проектирования отличаются набором предоставляемых пользователю инструментов, их сложностью и ценой. В целом объем трудозатрат на разработку аппаратных средств вновь проектируемых систем управления по отношению к суммарным затратам постоянно уменьшается.

Разработка прикладного ПО проектируемого контроллера. В отличие от этапа разработки аппаратных средств, удельный вес прикладного ПО при проектировании МК-систем имеет устойчивую тенденцию к увеличению.

Технологию проектирования ПО для МК удобно представить алгоритмом, представленным на рис. 3.1.

Рис. 3.1. Последовательность действий при проектировании прикладного ПО

Последовательность действий при проектировании ПО достаточна прозрачна. При обнаружении ошибок проектирования возможен возврат на предыдущие ступени разработки. Особенности отладки ПО рассматриваются ниже.

Многообразие возможных применений МК принципиально изменило сам подход к разработке прикладного ПО. Если при первых разработках МК-систем этап формирования объектного программного модуля считался наиболее трудоемким и ответственным, то в дальнейшем центр тяжести при разработке прикладного ПО переместился с фазы собственно программирования на фазу постановки и формализации задачи. Такое положение естественно неслучайно, так как до 60 % ошибок в прикладных программах вызвано не логическими ошибками программирования и не ошибками кодирования, а ошибочной формализацией прикладной задачи. Устранение подобных ошибок часто требует полного переосмысливания задачи и разработки нового программного обеспечения с иными аппаратными средствами. В настоящее время разработчиками прикладного ПО обычно являются не профессиональные программисты, а специалисты в конкретной области знаний (программирующие профессионалы). В определенной степени это связано с изменением стиля и технологии разработки программ (в условиях быстро дешевеющей памяти экономят уже не память контроллеров, а время разработки ПО).

Сказанное подчеркивает важность и ответственность разработки ПО, но при этом значение этапа выбора или разработки аппаратных средств нисколько не уменьшается. Это естественно и понятно, поскольку ПО проектируется для выбранной аппаратной реализации.

Комплексная настройка аппаратных средств и отладка ПО.

Под *отладкой* понимается процесс поиска, обнаружения и исправления ошибок в разрабатываемой системе. Отладка МК-систем включает стадии отладки отдельно аппаратных и отдельно программных средств и комплексную (совместную) настройку аппаратуры и ПО.

При отладке аппаратуры применяют аппаратные и программные средства отладки. Программные средства моделирования аппаратных средств позволяют исследовать работу аппаратной части МК-системы и выявлять ошибки в функционировании аппаратуры без проведения трудоемкого макетирования. Моделирование работы проектируемого устройства реализуется с использованием его принципиальной схемы и библиотечных данных о применяемых компонентах.

Не менее ответственным и сложным является этап отладки ПО. Его трудоемкость определяется рядом факторов:

- отладка программной и аппаратной частей системы сильно взаимосвязаны;
- отсутствует непосредственный доступ к внутренним ресурсам МК;
- многоразрядность сигналов, сложно распределенных во времени;

- непериодичность или большая скважность контролируемых сигналов.

Простейшим и одновременно самым неэффективным способом отладки разработанной программы МК является «метод проб и ошибок»: тестируемая программа загружается в репрограммируемое внутреннее ПЗУ и предпринимается попытка ее выполнения. При обнаружении ошибок в программе производится их исправление и перепрограммирование программной памяти МК. Процессы модифицирования программы (стирание и запись), являющиеся достаточно медленными, после определенного числа циклов перепрограммирования могут приводить к нарушению работоспособности СБИС МК. В настоящее время «метод проб и ошибок» при отладке программ не используется.

При разработке программ современных МК-систем используют специализированные средства поддержки проектирования. Подобные средства (аппаратные и аппаратно-программные), как правило, реализуют на базе компьютера. Применение средств поддержки проектирования существенно упрощает разработку программ, обеспечивая возможность их написания на языках высокого уровня (С, Паскаль и др.).

Характеристика инструментальных средств поддержки проектирования МК-систем.

Наиболее распространенными инструментальными средствами поддержки проектирования являются:

- программные средства;
- платы развития;
- эмуляторы ПЗУ;
- внутрисхемные эмуляторы;
- интегрированные системы разработки.

К *программным средствам поддержки* проектирования относятся ассемблеры, компиляторы, моделирующие программы - симуляторы (Simulators), программы отладчики (Debuggers).

С помощью программ трансляторов исходная программа, представленная в виде текста на языке программирования низкого или высокого уровня, преобразуется в объектный (машинный) код МК. Результат трансляции отображается на экране инструментальной ВМ в виде сообщения об отсутствии или наличии синтаксических ошибок в исходном тексте программы. При наличии ошибок, анализируется исходный текст программы, выявляются и исправляются ошибки.

Симулятор представляет собой программное средство, способное имитировать работу МК и его памяти. Обычно, симулятор содержит отладчик и программную модель вычислительного ядра МК-системы (процессора и памяти) и может содержать модели блоков внутренней периферии таких, как таймеры, порты ввода/вывода, АЦП. Симулятор

размещается в памяти инструментальной ВМ. Загрузив оттранслированную программу в память симулятора, пользователь имеет возможность запускать программу для исполнения в пошаговом и непрерывном режимах, задавать точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров МК. В процессе отладки модель «выполняет» программу, и на экране инструментальной ВМ в «окне» отладчика отображается текущее состояние модели (содержимое внутренней памяти МК и состояние его внутренних периферийных устройств). Симулятор позволяет проконтролировать логику работы проектируемой программы и устранить логические ошибки, но, к сожалению, он не учитывает реальных характеристик МК и подключаемых к нему внешних устройств. Исполнение программ, загруженных в симулятор, происходит в масштабе времени отличном от реального, поэтому при комплексной проверке системы с реальным объектом возможно выявление ошибок, в том числе и алгоритмических, для исправления которых, по крайней мере, требуется перепрограммирование памяти МК. Однако низкая цена, возможность отладки ПО, даже при отсутствии макета проектируемого устройства, делают программные симуляторы весьма эффективным средством отладки

Отладчик является мостом, связывающим разработчика с отлаживаемой системой. От свойств и качества отладчика зависят объем информации о подсистемах МК, доступность этой информации для контроля и при необходимости для коррекции.

В настоящее время при разработке и комплексной отладке микроконтроллерных систем с использованием реальных датчиков и исполнительных устройств в качестве основных инструментальных средств поддержки проектирования чаще всего применяют более сложные программно-аппаратные средства: платы развития, эмуляторы ПЗУ, внутрисхемные эмуляторы, интегрированные системы разработки.

Платы развития Широко распространенными средствами поддержки проектирования являются *платы развития* или как их называют в зарубежной литературе *Evaluation Boards*. Платы развития являются своеобразными конструкторами прикладных систем. Обычно они разрабатываются фирмами-производителями одновременно с новыми моделями МК. Плата развития представляет собой одноплатную микроЭВМ, содержащую в своем составе микроконтроллер новой модели , внешние микросхемы ПЗУ и ОЗУ, схемы связи с инструментальной ВМ, а также типовой набор периферийных устройств. Обязательной составной частью программно-аппаратных средств поддержки разработки является программа-монитор, которая реализует функции, подобные функциям операционной системы (это микроОС). С помощью монитора осуществляется запуск пользовательских программ и их выполнение в пошаговом и непрерывном режимах. Как правило, на монитор возлагаются

две группы функций: отладочные функции (монитор обеспечивает чтение и модификацию содержимого регистров и ячеек внутренней памяти, задание точек останова и т.д.) и функции связи с инструментальной ВМ. Экран отладчика монитора может быть похож на экран отладчика симулятора, но принципиально отличается тем, что отражает реальные характеристики МК и допускает возможность передачи параметров объекта в МК. Программа монитор может размещаться во внутренней памяти программ МК или загружаться вместе с отлаживаемой программой во внешнее ОЗУ. После завершения отладки дополнительные аппаратные и программные компоненты, используемые для отладки, могут быть удалены из контроллера, при этом достигается удешевление контроллера при его серийном производстве.

С помощью платы развития можно полностью промоделировать работу разрабатываемой МК-системы в реальном времени. Более того, универсальность оборудования плат развития позволяет использовать их в качестве конструктивной основы при изготовлении реальных микроконтроллерных систем. Некоторым недостатком плат развития является то, что для каждой модели МК требуется собственная плата развития (отсутствует их универсальность).

Эмулятор ПЗУ. Простейшим универсальным программно-аппаратным средством поддержки, используемым при отладке ПО, является *эмулятор ПЗУ*, представляющий собой некоторый набор аппаратных средств, соединенных последовательным каналом с инструментальной ВМ. Эмулятор ПЗУ можно использовать только в системах с МК допускающими обращение к внешней памяти программ. По стоимости и сложности эмуляторы ПЗУ сравнимы с платами развития, но они универсальны, так как могут работать с любыми типами МК. Аппаратура эмуляции имеет в своем составе блок оперативной памяти (ОП), который используется вместо внешней памяти программ отлаживаемой системы. ПО, размещенное в инструментальной ВМ, позволяет легко изменять содержимое памяти эмулятора, обеспечивая модификацию рабочих программ МК и запись дополнительных отладочных и тестовых программ. Вид системы с эмулятором ПЗУ показан на рис. 3.2.

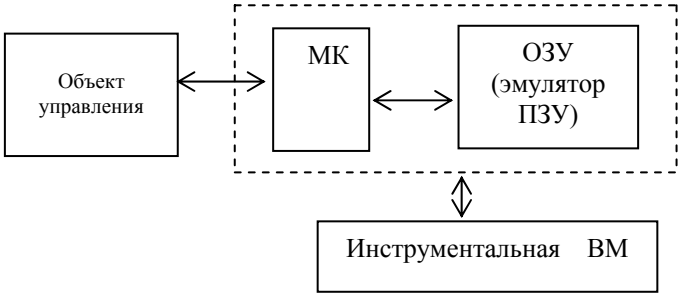


Рис. 3.2. Микроконтроллерная система с эмулятором ПЗУ

В настоящее время появились модели интеллектуальных эмуляторов ПЗУ, которые позволяют контролировать состояние внутренних блоков МК и управлять процессом отладки. Такие эмуляторы ПЗУ очень похожи на внутрисхемные эмуляторы (In-Circuit Emulators).

Внутрисхемные эмуляторы предназначены для отладки прикладного ПО и комплексной настройки аппаратной части системы при наличии изготовленного модуля центрального вычислителя на базе МК. Являясь более сложными и мощными средствами поддержки проектирования МК-систем, они способны заменить эмулируемый МК в реальной системе. В состав внутрисхемного эмулятора обычно входят блок эмуляции МК, эмуляционная память и отладчик. Блок эмуляции МК, как правило, реализуемый на процессоре с более высокими скоростными характеристиками, способен воспроизводить в реальном времени все характеристики эмулируемого МК. С помощью инструментальной ВМ он обеспечивает доступ ко всей информации о состоянии внутренних устройств МК и его памяти.

Внутрисхемный эмулятор может быть выполнен либо в виде стандартной платы, которая устанавливается на свободное место в корпусе инструментальной ВМ, либо в виде отдельного блока. Стыковка внутрисхемного эмулятора МК с отлаживаемой системой производится при помощи «эмуляционного» кабеля со специальной эмуляционной головкой, которая вставляется вместо МК в отлаживаемую систему. Если МК невозможно удалить из платы, все выводы МК переводят в 3-е состояние, а для подключения эмулятора используют специальный адаптер-клипсу, подключаемый к выводам эмулируемого МК.

Интегрированные системы разработки представляют собой совокупность программно-аппаратных средств, поддерживающих все этапы разработки прикладного ПО от написания исходного текста программы до комплексной отладки разрабатываемой системы. Программное обеспечение интегрированных систем разработки обычно объединяет такие инструментальные средства, как редактор исходных текстов, компилятор, программный симулятор, отладчик, справочную подсистему. В состав аппаратных средств интегрированных систем входят внутрисхемные эмуляторы, платы развития и другие аппаратные средства, обеспечивающие удобство программирования и отладки МК-систем. Благодаря использованию интегрированных систем, цикл «разработка – изготовление» новых систем сокращается до нескольких дней или недель, при этом объем изготавливаемой партии изделий может составлять сотни - десятки штук при весьма доступной цене.

Некоторые производители с целью популяризации своих изделий и ускорения их продвижения на рынке разрабатывают и предлагают разработчикам МК-систем так называемые стартовые комплекты разработчика (Project Builder), представляющие собой совокупность

программных и отладочных модулей. Project Builder позволяет решать следующие задачи:

- изучение структуры конкретного семейства микроконтроллеров;
- выбор МК для разрабатываемой системы;
- приобретение навыков настройки и управления периферийными устройствами МК;
- накопление опыта разработки и отладки ПО для МК с использованием ассемблера и языков высокого уровня;

При работе в автономном режиме модуль стартового комплекта разработчика может быть использован в качестве центрального вычислителя разрабатываемой МК-системы.

Характеристики распространенных семейств МК

Многообразие задач, решаемых с помощью МК, позволило сформулировать основные требования к ресурсам, размещаемым на его кристалле, и его производительности. Удовлетворить разнообразные требования при минимальной стоимости кристалла с помощью одной универсальной структуры МК практически невозможно. Производители предлагают широкий спектр модификаций МК, отличающихся друг от друга пользовательскими требованиями, ценой и требованиями, предъявляемыми к ресурсам кристалла.

Можно выделить два больших класса задач, решаемых встраиваемыми системами: управление событиями в реальном времени и управление потоками данных. Каждый класс задач характеризуется своими специфическими требованиями, которые учитываются в наборе функциональных устройств, размещаемых на кристалле МК, а также в его системе команд. К первому классу относятся задачи, требующие быстрой реакции МК-системы на изменение внешних условий (сигналы технологических датчиков, изменение параметров и пр.). Эти задачи требуют применение МК с встроенной памятью программ и данных, большим объемом интегрированной на кристалле периферии, большим числом портов ввода/вывода для подсоединения датчиков и исполнительных устройств, системой команд, включающей битовые операции. Многие из встраиваемых систем управления, такие как системы управления технологическим оборудованием и периферийными устройствами ВМ, системы управления устройствами промышленной автоматики, а также распределенные системы, принадлежат к системам первого класса.

Ко второму классу задач относятся *задачи, требующие быстрой обработки значительных объемов информации*, например задачи, характерные для систем управления бортовым оборудованием, систем обработки видеоизображений и др. В таких системах, как правило, используется встроенный высокопроизводительный 32- или 64-разрядный

процессор, выполняющий множество различных вычислительных операций, в том числе операции с плавающей точкой. Встраиваемыми 16-разрядными и 32-разрядными МК с PC-подобной архитектурой являются МК 80C186, 386EX фирмы Intel, МК семейства C166 фирмы Siemens и др.

Для характеристики современных МК используют различные классификационные признаки, важнейшими из которых являются тип ЦП, его производительность, разрядность обрабатываемых данных, принцип организации и объем адресуемой памяти, состав внутренней периферии, число выводов МК. Основные характеристики МК определяют его функциональные возможности. В настоящее время в дополнение к традиционно используемым 8-разрядным МК общего назначения активно развиваются два направления в развитии микроконтроллерной элементной базы. Они ориентированы на существенное расширение спектра возможных применений МК во встраиваемых системах управления. Первое направление характеризуется развитием МК для сложных встраиваемых систем управления. В таких системах функциональная сложность решаемых задач может быть сопоставима с возможностями ПК. Использование 8-разрядных МК с их производительностью, вычислительной мощностью и объемом встроенной периферии для таких систем может оказаться неэффективным. Это направление связано с усложнением МК, применением 16- и 32-разрядных МК, встраиванием дополнительных аппаратных ресурсов. Второе направление, напротив, направлено на некоторое «упрощение» функциональных возможностей МК. Для некоторых объектов при несложных алгоритмах управления требуется не высокая производительность и большое число устройств внутренней периферии, а малое энергопотребление и компактные массогабаритные характеристики. Более простые МК характеризуются уменьшенным объемом внутренней памяти программ (до 1-2 Кбайт), сокращенным числом портов и выводов микросхемы. Подобные МК находят применение во встраиваемых устройствах управления несложными объектами. Ниже рассматриваются характеристики распространенных семейств МК. Цель такого рассмотрения – наглядно продемонстрировать возможности современных МК и помочь разработчикам МК-систем ориентироваться в многообразии типов МК.

4-разрядные МК являются самыми простыми и дешевыми устройствами для встраиваемых применений. Они предназначены для замены несложных схем на «жесткой» логике. Типичными характеристиками 4-разрядных МК являются:

- объем IROM: < 1кбайт;
- объем IRAM: 16 – 64 4-разрядные ячейки памяти;
- число команд: 30 – 50;
- тактовая частота: 100 кГц – 1 МГц;

- состав внутренней периферии: 2 - 4 четырехразрядных параллельных порта, иногда контроллер жидкокристаллического индикатора ЖКИ.

4-разрядные МК широко применяются в электронных часах, в простых калькуляторах, в несложных устройствах бытовой техники, в игрушках.

8-разрядные МК занимают лидирующее положение на мировом рынке микроконтроллерной техники. Объем продаж 8-разрядных МК примерно в 2,5 раза превышает объем продаж ближайших конкурентов 16-разрядных МК. Наиболее распространенными 8-разрядными МК являются МК семейства MCS-51, получившие свое название от МК 8051, выпущенного фирмой Intel в 1980 году [3]. Удачный набор внутренней периферии, возможность использования внешней или внутренней памяти программ и относительно невысокая стоимость обеспечили применение МК этого семейства для решения самых разнообразных задач. Основные характеристики МК семейства MCS-51 представлены в табл.8.

Важную роль в достижении высокой популярности семейства MCS-51 сыграла открытая политика фирмы Intel, направленная на широкое распространение лицензий на ядро МК 8051 среди ведущих полупроводниковых компаний мира. На сегодня существует более 200 модификаций МК

Характеристика 8-разрядных МК семейства MCS-51 Таблица 8

МК	Максимальная тактовая частота, МГц	ROM/ EPROM, Кбайт	RAM байт	Таймеры	Число параллельных портов
8xC51BH	24	4	128	2	4
8xC52	24	8	256	3	4
8xC54	33	16	256	3	4
8xC58	33	32	256	3	4
8xC51FA	24	8	256	3 +PCA	4
8xC51FB	24	16	256	3 +PCA	4
8xC51FC	24	32	256	3 +PCA	4
8xC51GB	16	8	256	3 +2 PCA+WDT	6
8xC152JA	16	8	256	2	5
SAB 80C515 SAB 80C535	12	8 -	256	3 + PCA WDT	6
AT89 C55	12 - 24	Flash 20Кбайт	256	3	4

Примечание. Все МК семейства содержат в своем составе асинхронный последовательный приемопередатчик UART; в составе МК 8xC51GB и SAB 80C515 имеется 8-канальный АЦП.

семейства MCS-51, выпускаемых почти 20 компаниями. Эти модификации включают в себя кристаллы с широчайшим спектром ресурсов: от 20-выводных устройств с одним таймером и программной памятью 1 Кбайт до сложнейших 100-выводных кристаллов с 10-разрядным АЦП, массивами таймеров/счетчиков, аппаратным 16-разрядным умножителем и программной памятью объемом 64 Кбайт. Ряд фирм с использованием современных технологических достижений создают x51-совместимые МК с более высокими техническими характеристиками. В частности, фирма Atmel производит x51-совместимые МК семейства AT89 с памятью программ, реализованной в виде электрически перепрограммируемой Flash-памяти. Такое решение позволило отказаться от прозрачного окна в корпусе СБИС МК со стиранием УФ облучением и существенно снизить стоимость.

Некоторым недостатком МК семейства MCS-51 является их относительно невысокая производительность (на выполнение одной команды в среднем тратится 12 тактов частоты синхронизации и сравнительно высокое энергопотребление). По этой причине для приложений, требующих повышенного быстродействия и большей вычислительной мощности, часто приходится использовать более производительные МК, например RISC МК. Для устранения отмеченного недостатка разработчики фирм Intel и Philips разработали семейство МК MCS-251 с архитектурой, обеспечивающей аппаратную и программную совместимость с МК MCS-51, но имеющее высокую производительность. МК MCS-251 фактически представляет собой два МК: собственно МК MCS-251 с расширенным набором команд, включающим команды 16-битовой арифметики, и МК, воспроизводящий режим работы МК MCS-51.

Основными характеристиками МК семейства MCS-251 являются:

- линейное адресное пространство памяти объемом до 16 Мбайт;
- двухадресная архитектура;
- аппаратная реализация очереди команд;
- расширенный набор команд, включающий 16-битные арифметические и логические команды;
- выполнение простых команд за два такта;
- расширенный стек объемом до 64 Кбайт.

В режиме МК MCS-251 используются 24-разрядные адреса и адресуется программная память объемом до 16 Мбайт. Младшие 64 Кбайт этой памяти (IROM) могут размещаться на кристалле МК. Обращение к IROM выполняется за 2 такта, при этом обеспечивается считывание двух байтов команды. При обращении к внешней памяти программ считывается байт, а само обращение реализуется за 2 или 4 такта. Объем внутренней памяти RRAM расширен: 512 или 1024 8-разрядных регистра. 56 регистров RRAM допускают обращение с использованием регистровой адресации.

Операнды команд могут быть представлены в формате байт, слово, двойное слово. Любой РОН, пара или четверка регистров может быть источником или приемником. Команды выполняются за 2 такта.

Система команд МК MCS-251 построена на базе двух наборов инструкций: набора команд MCS-51 и расширенного набора команд MCS-251. Перед использованием микроконтроллер семейства MCS-251 необходимо сконфигурировать путем инициализации однократно программируемых конфигурационных регистров. Конфигурация МК определяет, какой из наборов команд станет активным после включения питания. Однако и после конфигурации сохраняется возможность использования обоих наборов команд.

8-разрядные МК с RISC CPU. Ряд производителей современных МК отказались выпускают высокопроизводительные МК с RISC CPU. Напомним, что система команд RISC процессора объединяет только простые команды, выполняемые за такт. Основным достоинством RISC процессоров является существенное упрощение аппаратной реализации CPU и возможность повышения частоты работы и производительности.

8-разрядные МК AVR фирмы Atmel.

Идея создания МК с RISC ядром родилась в Норвегии (1994). Его изобретатели Альф Боген и Бегард Волен предложили фирме Atmel 8-разрядный RISC МК с внутренней Flash-памятью на кристалле. МК-ядро было запатентовано и получило название AVR (**A**lf Bogen + **V**egard Wollen + **R**ISC). Новый МК представлялся перспективным и с 1998 г. началось его активное внедрение. Для AVR МК различными фирмами разработано и выпущено большое количество разнообразных аппаратных и программных средств поддержки проектирования AVR МК-систем. В настоящее время AVR МК является еще одним индустриальным стандартом 8-разрядных МК общего назначения.

AVR-архитектура (рис. 3.3) имеет следующие особенности:

- включает в себя мощный двухадресный гарвардский RISC процессор с раздельным доступом к памяти программ и памяти данных;
- объем внутренней памяти программ, реализованной в виде флэш-памяти варьируется в широких пределах (от 1…2 Кбайт до 32, 64, 128 и 512 Кбайт). Существует тенденция замены всех типов IROM блоками Flash-памяти;
- объем внутренней оперативной памяти SRAM зависит от модели МК и составляет 128 … 512 байт. Большинство моделей МК допускают подключение внешней памяти данных объемом до 64 Кбайт;
- все МК AVR содержат блок энергонезависимой электрически стираемой памяти данных EEPROM объемом 64 … 512 байт, используемой в качестве дополнительного блока ОЗУ для хранения промежуточных данных, различных констант и таблиц перекодировок.

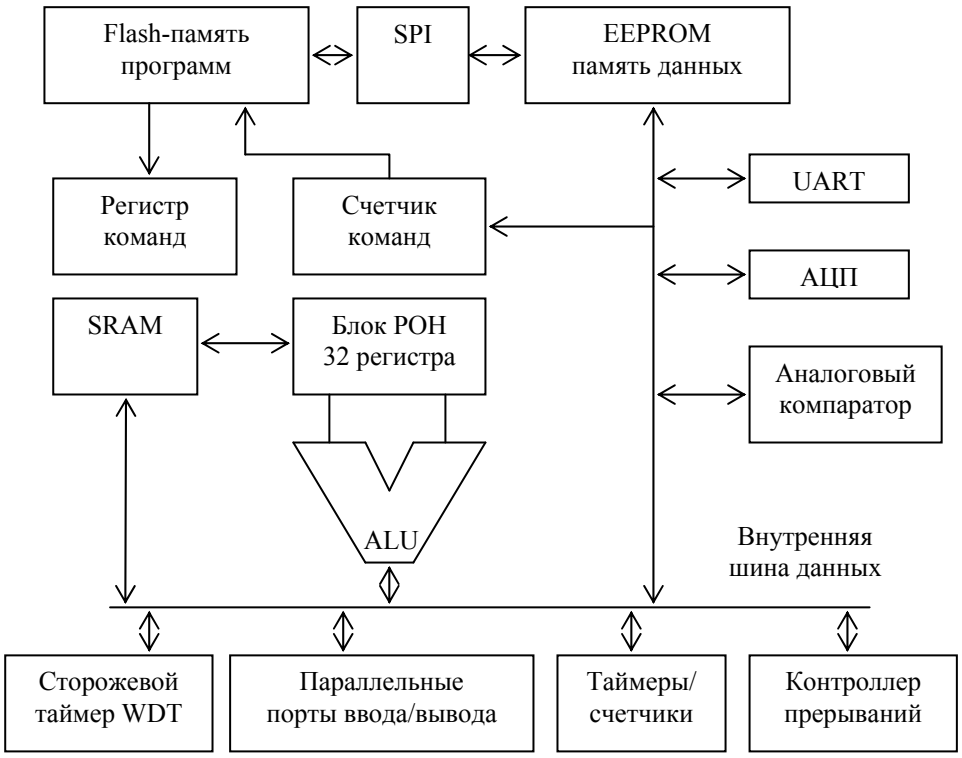


Рис. 3.3. Структурная схема AVR МК

Память EEPROM может быть загружена извне как через SPI интерфейс, так и с помощью обычного программатора. Число циклов перезаписи составляет не менее 100000;

- команды МК AVR имеют одинаковый формат (16 разрядов). Большинство команд МК выполняется за машинный такт в конвейерном режиме. Время выполнения команд при тактовой частоте 20 МГц составляет 50 нс;
- содержит регистровый файл, объединяющий 32 регистра, каждый из которых напрямую связан с АЛУ (за такт из регистрового файла выбираются два операнда, выполняется операция и результат возвращается в регистровый файл). Регистровый файл занимает область ОЗУ с младшими адресами;
- производительность AVR МК существенно выше производительности МК семейства MCS-51, даже при выполнении сложных операций, реализуемых тремя простыми RISC-командами.

Микроконтроллеры AVR разработаны в соответствии с принципами построения языка «Си». Благодаря этому компиляция исходных текстов, написанных на «Си», осуществляется быстро и дает эффективный код.

Важной особенностью AVR МК является объединение в них флэш-технологии памяти IROM со стандартным последовательным интерфейсом SPI. Такое решение обеспечило возможность многократно

модифицировать программный код во внутренней памяти программ в уже изготовленной системе пользователя. Число циклов перезаписи памяти IROM составляет не менее 1000. Два программируемых бита секретности позволяют защитить память программ от несанкционированного считывания.

На рынке МК AVR МК представлены 4-мя подсемействами – Mega AVR (префикс ATmegaXXX), Classic AVR (префикс AT90SXXX) Tiny AVR (префикс ATtinyXXX) и AVR для Smart Cards. МК всех подсемейств имеют единую базовую систему команд и программно совместимы.

МК Mega AVR – самые мощные МК семейства. Они содержат наибольший объем встроенной памяти (до 128 Кбайт), наиболее полный набор устройств встроенной периферии и имеют наибольшее количество выводов (64 вывода).

МК Classic AVR исторически были первыми МК семейства. Состав внутренней периферии и объем встроенной памяти этих МК, как и число выводов микросхемы, варьируется зависимости от модификации. Большинство МК этого семейства содержат память программ типа Flash объемом от 1 до 8 Кбайт, внутреннюю память данных SRAM 128 – 512 байт и дополнительно к ней перепрограммируемую память EEPROM объемом 64 – 512 байт. Практически все МК подсемейства за исключением 8-выводных малюток содержат два таймера (8-битный и 16-битный с формирователем ШИМ и схемами сравнения), асинхронный (UART) и синхронный (SPI) последовательные порты, аналоговый компаратор. В старших моделях МК Classic AVR, например AT90S8535, традиционный набор внутренний периферийных устройств расширен за счет включения 10-разрядного АЦП с мультиплексором на 8 каналов и третьего 8-битного таймера. Этот таймер поддерживает работу блока быстрого ввода/вывода, обеспечивая формирование ШИМ-сигналов и реализуя функции захвата/сравнения. Главной особенностью таймера является то, что он может работать в асинхронном от всей системы режиме, используя дополнительный источник внешней частоты 32,768 кГц, что позволяет вести отсчет реального времени.

МК Tiny AVR – представляют собой наиболее простые и дешевые кристаллы семейства стоимостью порядка 1 $. Они выпускаются в 8-выводном корпусе и способны работать от источника пониженного напряжения (2,7 В). Несмотря на низкие массогабаритные характеристики и малое энергопотребление, МК Tiny AVR содержат в своем составе широкий набор периферийных устройств.

8-разрядные МК PICmicro. Распространенным семейством 8-разрядных RISC МК являются МК PICmicro фирмы Microchip Technology Inc. Из важнейших архитектурных особенностей данного семейства МК можно выделить следующие: гарвардская архитектура с раздельным

доступом к памяти программ и памяти данных, одинаковый формат команд (12 бит), наличие очереди команд, конвейерная обработка. Команды выполняются за 8 тактов (2 машинных цикла), но благодаря конвейерной обработке результаты их исполнения формируются в каждом машинном цикле длительностью 4 такта. Все МК PICmicro имеют внутреннюю память программ. В составе семейства имеются модели с масочной, однократно программируемой (OTP) и многократно программируемой (с ультрафиолетовым стиранием и Flash) памятью. С помощью бита секретности имеется возможность защитить память программ от несанкционированного считывания. Бит секретности устанавливается после записи программного кода в ПЗУ. Попытка чтения ПЗУ с установленным битом секретности приводит либо к чтению бессмысленной (зашифрованной) информации (МК PIC16C5x), либо к полному запрету считывания (МК PIC16C84).

Встроенный тактовый генератор обеспечивает работу МК в широком диапазоне частот (от 32 кГц до 33 МГц). Напряжение питания отдельных моделей МК варьируется от 2 В до 6 В.

Отличительной характеристикой МК PICmicro является низкое энергопотребление, которое существенно зависит от частоты работы МК. Например, МК PIC16C5x на частоте 32 кГц потребляет всего 12 мкА, а на частотах 4 и 20 МГц - 2 мА и 9 мА соответственно. В режиме пониженного энергопотребления ток уменьшается до 0,25 мкА.

В зависимости от состава внутренней периферии МК PICmicro подразделяются на несколько программно совместимых семейств. Полное количество моделей МК PICmicro составляет порядка 500. Родоначальником семейства являются МК PIC16C5x, оперирующие 12-битовыми командами и имеющие минимальную конфигурацию (один 8-битный таймер, сторожевой таймер и 1-3 двунаправленных портов ввода/вывода). МК базового семейства выпускаются в 18-, 20- и 28-выводных корпусах. Максимальная производительность МК базового семейства равна 5 MIPS. 8-выводные МК-«малютки» семейства PIC12Cxx имеют структуру, подобную базовой. В них связь с внешним “миром” осуществляется с помощью 6 линий ввода/вывода.

Развитием базового семейства являются МК PIC16Cxx с расширенным составом внутренней периферии. Отдельные модели этого семейства могут содержать двунаправленные порты ввода/вывода, последовательные порты, поддерживающие стандарты последовательной связи (I2C, SPI, USART), дополнительные таймеры/счетчики, аналоговый компаратор и 8-разрядный АЦП, блоки быстрого ввода/вывода, контроллер жидкокристаллического индикатора, сторожевой таймер и др. В новых моделях МК используется Flash-память программ и добавлена электрически перепрограммируемая память данных EEPROM.

Наиболее популярным и перспективным семейством МК PICmicro является семейство МК PIC18Cxx. При его разработке основное внимание было уделено оптимизации системы команд с целью обеспечения удобства компиляции программ, написанных на языке Си. По сравнению с базовым МК в структуру МК PIC18Cxx внесены значительные изменения:

- максимальная тактовая частота 40 МГц;
- максимальная производительность 10 MIPS;
- адресное пространство памяти программ расширено до 2 Мбайт;
- доработан аппаратный интерфейс внешней памяти программ;
- расширена система команд;
- использован фиксированный формат команд (14 бит);
- увеличено число внутренних периферийных устройств.

МК PIC18Cxx содержат аппаратный умножитель, реализующий умножение байт в течение 1 машинного цикла, и 10-разрядный многоканальный АЦП. В некоторых моделях может использоваться дополнительный USART с поддержкой протокола IEEE-485 или добавляется межконтроллерный интерфейс сетевой поддержки.

16-разрядные МК семейства MCS-96, разработанные фирмой Intel в конце 80-х годов прошлого века, также ориентированы на решение задач управления процессами в реальном времени. В состав семейства входит более 30 моделей МК, имеющих одинаковую базовую структуру и программно совместимую систему команд [4]. Семейство МК MCS-96 динамически развивается, улучшаются его характеристики. За годы развития адресное пространство памяти увеличилось с 64 Кбайт до 1 Мбайт (семейство NT) и даже 6 Мбайт (семейство К17), тактовая частота повысилась с 10 МГц до 50 Мгц, производительность возросла в 16 раз, при этом улучшение характеристик достигнуто при значительном снижении стоимости МК. МК этого семейства фактически стали индустриальным стандартом для 16-разрядных МК общего назначения.

МК семейства MCS-96 имеют классическую структуру МК. Основными блоками МК являются ЦП, память программ (внутренняя и/или внешняя), память данных (внутренняя, а при необходимости и внешняя), а также набор периферийных устройств, размещенных на кристалле вместе с ЦПУ.

Центральное процессорное устройство МК реализовано в соответствии с регистр-регистровой архитектурой. В ней любые ячейки регистрового ЗУ могут использоваться в качестве операндов-источника и в качестве приемника результата операции АЛУ. Такую архитектуру ЦПУ в отличие от архитектуры с одним регистром результата называют многоаккумуляторной. Важной особенностью регистр-регистровой архитектуры является возможность использования 3-адресных команд логических и арифметических операций. Благодаря этому необходимость в дополнительных операциях пересылки данных из аккумулятора и обратно

отпадает и производительность процессора возрастает. В системе команд МК семейства MCS-96 трехадресными являются некоторые команды арифметических операций. Наряду с 3-адресными, широко используются 2-адресные команды, в которых адрес одного из операндов одновременно является адресом приемника результата. По сравнению с одноадресными командами MCS-51 команды MCS-96 являются значительно более мощными.

Объем регистрового файла RRAM в различных МК семейства MCS-96 варьируется и может составлять 232, 488 или 1000 байт. Ячейки RRAM используются только для хранения данных. При обращении к ним применяется прямая регистровая адресация. В дополнение к RRAM регистровый файл МК включает регистры специальных функций, предназначенные для управления работой и отображения состояния ПУ, размещенных на кристалле МК.

Микроконтроллеры семейства MCS-96 имеют единое адресное пространство памяти для команд и данных (Принстонская архитектура). Внутренняя память программ IROM доступна только для чтения. У МК разных типов объем памяти IROM может составлять 8, 12, 16, 24 или 32Кбайт, при этом некоторые модификации МК могут и не содержать внутренней памяти программ. В памяти IROM имеется область, используемая для хранения векторов прерываний, ключа защиты и некоторых других специальных кодов. Для размещения памяти программ также может использоваться внешняя память. Максимальный суммарный объем внешней и внутренней памяти (без регистрового ЗУ) у МК большинства типов составляет 64 Кбайт.

Микроконтроллеры подсемйств KR и NT содержат внутреннюю память IRAM, в которой могут хранится как данные, так и команды. Емкость памяти IRAM в зависимости от типа МК может составлять 128, 256 или 512 байт. При использовании памяти IRAM для размещения команд программы возможна модификация программы в процессе ее выполнения и, кроме того, фрагменты программы в памяти IRAM выполняются быстро, поскольку память IRAM имеет 16-разрядный интерфейс. В задачах с циклическим выполнением небольших фрагментов кода использование памяти IRAM может существенно повысить скорость выполнения программ, при этом основная программа по-прежнему может размещаться в относительно «медленной» 8-разрядной памяти IROM.

МК MCS-96 имеет типовую структуру (рис. 3.4). ЦП обращается к памяти с помощью размещенного на кристалле контроллера памяти, содержащего 4-байтную очередь команд. При обращении к внешней памяти

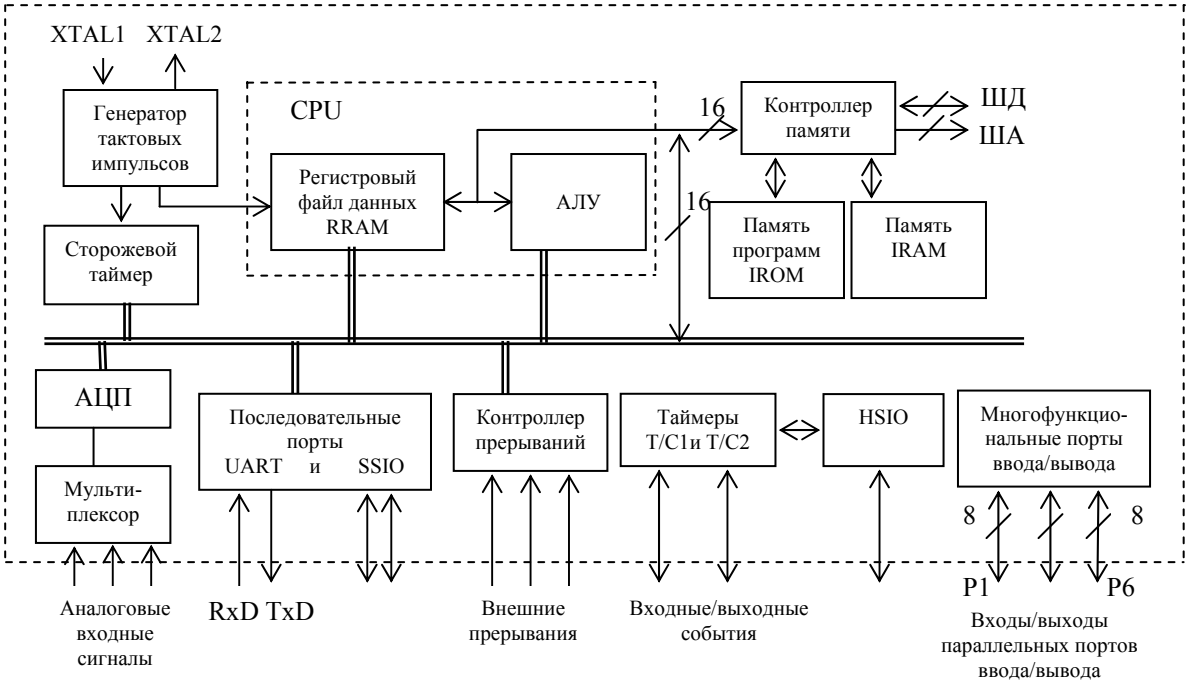


Рис. 3.4. Функциональная схема МК семейства MCS-96

контроллер памяти управляет внешней шиной, ширина которой может динамически переключаться с 8-разрядной на 16-разрядную.

В состав внутренней периферии МК семейства MCS-96 входит классический набор периферийных устройств типового МК:

- набор из 6-8 параллельных портов ввода/вывода, два из которых (P3, P4) могут использоваться для организации внешней мультиплексируемой 16-разрядной шины адрес/данные. Отдельные выводы других портов, наряду со стандартным вводом/выводом, могут выполнять альтернативные функции ввода/вывода сигналов управления внутренней периферией;- многоканальный АЦП, преобразующий аналоговые входные сигналы напряжения в 8-или 10-разразрядный двоичный код;
- два 16-разрядных счетчика/таймера. Кроме функций счета внешних событий и формирования программируемых задержек, таймеры используются блоками быстрого ввода/вывода для генерации выходных событий в заданные моменты времени и регистрации входных событий;
- формирователь ШИМ-сигналов;
- дуплексный последовательный порт. Ряд моделей, наряду с универсальным синхронно-асинхронным приемопередатчиком UART, содержат в своем составе дополнительный синхронный последовательный порт SSIO, предназначенный для одновременной двунаправленной связи двух МК или устройств, содержащих SSIO порт;
- развитая система прерываний.

В МК некоторых типов для обслуживания прерываний, наряду с типовым программируемым контроллером прерываний, используется

контроллер прерываний PTS (Peripheral Transaction Server), обслуживающий прерывания на микропрограммном уровне. Выполнение микропрограмм-обработчиков запросов прерываний совмещается во времени (фактически чередуется) с выполнением основной программы. Под управлением контроллера PTS могут выполняться отдельные и групповые пересылки 8- и 16-разрядных слов между любыми областями памяти, генерироваться ШИМ-сигналы, обслуживаться некоторые другие прерывания.

Основные характеристики наиболее популярных МК семейства MCS-96 приведены в табл.9.

16-разрядные МК семейства MCS-96 Таблица 9

Обозначен. под-семейства	Тип МК	Тактовая частота, МГц/ Быстродействие MIPS	IROM Кбайт	RRAM, байт	Дополн. IRAM, Байт	Число линий ввода/ вывода
BH KB KB	8X96BH/ 8XC196KB 8XC198	12/1 16/2 16/2	8	232	-	48
KC	8XC196KC 8XC196KD	20/2,5 20/2,5	16 32	488 1000	- -	48 48
KR	8XC196KR 8XC196KQ 8XC196JR 8XC196JQ	16/2	16 12 16 12	488 360 488 360	256 128 256 128	56 56 41 41
MC	8XC196MC 8XC196MP	16/2	16 32	488 744	- 256	53 50
NT	8XC196NT 8XC196NP 8XC196NU	20/2,5 25/3 50/6	32 4 -	1000 1000 1000	512 - -	56 56 32
MCS-296	8XC296SA	50/16	2	512	2048	32

*Примечание***.** Все МК содержат в своем составе два таймера/счетчика и последовательный порт UART. МК подсемейства KR и МК 8XC196NT содержат дополнительный последовательный порт SSIO. Число программируемых блоков HSIO в МК разных типов варьируется от 6 до 10. МК семейств NT и MCS-296 не содержат АЦП. Все модели МК, кроме МК подсемейств BH, KB и МК 8XC296SA, оснащены контроллером прерываний PTS.

МК подсемейства MCS-296 *с интегрированным на кристалл процессором цифровой обработки сигналов*. При построении встраиваемых систем управления, выполняющих цифровую обработку сигналов, существенной особенностью является то, что такие системы нельзя реализовать ни отдельным МК с большим набором периферийных устройств, ни отдельным сигнальным процессором DSP. Использование

первых для подобных задач оказывается неэффективным, поскольку их система команд мало пригодна для специфики цифровой обработки сигналов, а DSP не содержат в своем составе необходимого набора внутренней периферии. Как правило, подобные системы строились как двухпроцессорные системы, в которых МК выполнял обработку событий в реальном времени, а дополнительную цифровую обработку сигналов реализовывал DSP. По мере развития микроэлектроники стала возможна интеграция функций этих процессов в одном кристалле. В 1996 году фирма Intel предложила 16-разрядный МК с конвейерной архитектурой 80296SA, который имеет высокую производительность и содержит встроенные средства цифровой обработки сигналов.

Кратко охарактеризуем МК подсемейства MCS-296. В качестве отличительных характеристик МК подсемейства MCS-296 можно отметить:

- увеличение производительности ЦПУ;
- расширение системы команд МК семейства MCS-96 специальными командами цифровой обработки сигналов;
- наличие аппаратных блоков умножения и деления.

Повышение производительности ЦПУ достигается за счет сокращения времени выборки команд из памяти и их конвейерной обработки. Выборка команд и их выполнение реализуются разными блоками. При предварительной выборке команд из памяти они размещаются в специальной очереди длиной 10 байт. В ходе выполнения программы процессор практически не тратит дополнительного времени на считывание команд из памяти, выбирая требуемую команду из очереди. Выборка операндов из памяти более приоритетна по сравнению с выборкой команд. Четырехступенчатый конвейер исполнения команд в пределе позволяет одновременно выполнять четыре команды.

Аппаратные блоки умножения и деления позволяют значительно сократить время исполнения распространенных операций цифровой обработки сигналов, например умножения с накоплением.

В целом МК MCS-296 обеспечивают преимущества относительно дешевых МК с большим количеством интегрированных на кристалле периферийных устройств и DSP-процессоров среднего быстродействия, предоставляя в распоряжение пользователей мощное средство, способное решать широкий спектр задач. Основные характеристики МК 8XC296SA представлены в табл. 9.

32-разрядные МК i386EX. Эти микроконтроллеры фирмы *Intel* предназначены для решения задач быстрой обработки значительных объемов информации. В своем составе они содержат высокопроизводительный 32-разрядный МП, подобный МП i386SX, и уникальный набор периферии, обеспечивающий DOS-совместимость с ПК PC/AT. При сбросе МК i386EX переводится в режим, в котором все внутренние ПУ имеют адреса ячеек и структуру прерываний, идентичные спецификациям шины ISA. При необходимости МК i386EX допускает программное изменение адресации ПУ. Поддержка DOS-совместимости обеспечивает быструю разработку и внедрение встраиваемых высокопроизводительных 32-разрядных МК. Основные особенности архитектуры МК i386EX следующие:

- высокопроизводительный 32-разрядный МП с частотой работы до 25 МГц;
- раздельные шины адреса и данных;
- адресное пространство памяти 4 Мбайт;
- адресное пространство ввода/вывода 64 Кбайт;
- совместимость адресов ввода/вывода и прерываний с PC/AT;
- высокопроизводительный контроллер ПДП (два канала);
- два асинхронных последовательных порта UART;
- дуплексный синхронный последовательный порт;
- два режима пониженного энергопотребления;
- режим системного управления SMM;
- три программируемых таймера/счетчика;
- сторожевой таймер;
- три порта ввода/вывода параллельной информации;
- устройство управления регенерацией динамической памяти;
- отсутствие встроенной памяти программ.

Более полную информацию о МК различных фирм содержат фирменные описания разработчиков МК.

4. Методические указания к лабораторному практикуму

Основной целью лабораторного практикума является углубленное изучение особенностей функционирования внутренних блоков МК (на примере МК SAB 80C535 семейства MCS-51), приобретение навыков программирования этих МК и проектирования встраиваемых микроконтроллерных систем. В рамках лабораторного практикума изучаются внутренняя структура МК SAB 80C535, способы сопряжения МК с внешними устройствами ввода/вывода, организация межпроцессорных обменов, вопросы разработки прикладного ПО, проектирования и отладки микроконтроллерных систем.

Работы выполняются с помощью лабораторного стенда, подключенного по последовательному каналу связи к COM-порту инструментальной ЭВМ. С помощью программ резидентного «монитора» обеспечивается доступ к ресурсам контроллера и контроль результатов выполнения пользовательских программ на экране ЭВМ.

4.1. Описание лабораторного стенда

Лабораторный стенд (рис. 4.1) предназначен для изучения особенностей построения и функционирования встраиваемых микроконтроллерных систем. В состав стенда входят одноплатный контроллер, совокупность устройств дискретного и аналогового ввода/вывода, два генератора тестовых воздействий и коммутационное поле. На плате контроллера размещены МК SAB 80C535, микросхемы внешней памяти программ и данных (ОЗУ объемом 32 Кбайт), микросхемы внешней памяти программ «резидентного» монитора, схемы физического последовательного канала с оптоэлектронной развязкой и набор вспомогательных микросхем

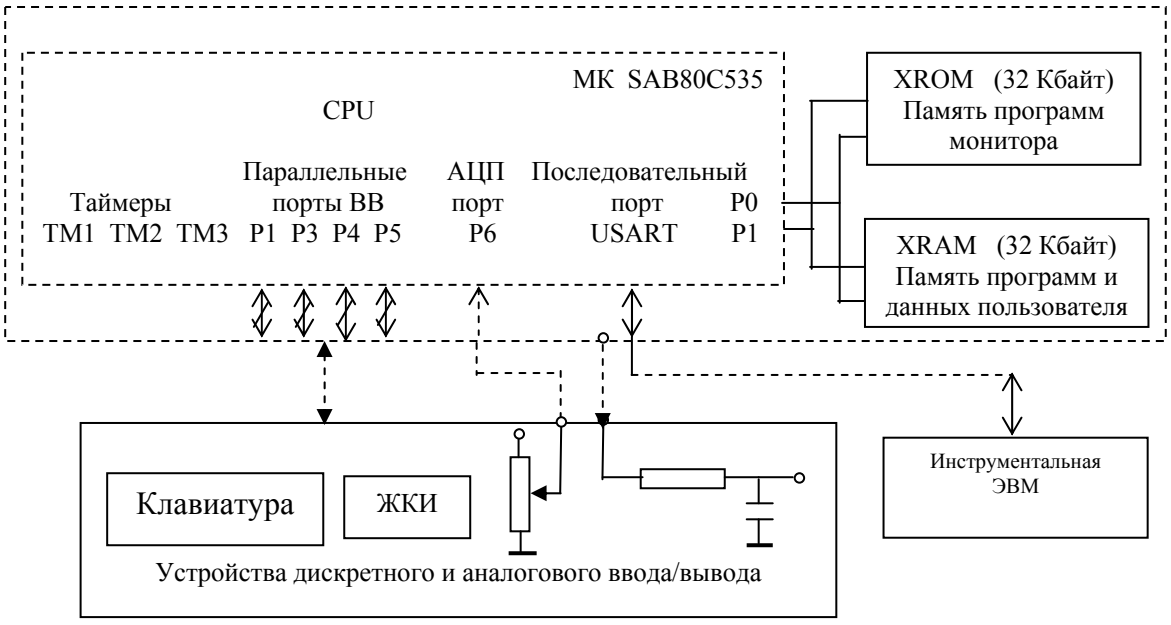


Рис.4.1 Функциональная схема лабораторного стенда

сопряжения МК с внешними устройствами.

Лабораторный стенд работает в составе системы поддержки проектирования МК-систем на базе ПК. С помощью ПК и установленной на нем оболочки проектирования Shell51[1] [5] решаются следующие задачи разработки прикладного ПО МК-систем:

- подготовка исходного текста программ на языке ассемблера;
- трансляция «ассемблерных» программ в машинный код МК;
- обнаружение и коррекция синтаксических ошибок программы;
- запуск и отладка программы на симуляторе;
- загрузка программы в исполняемом формате в МК;
- контроль содержимого внутренней памяти программ и данных МК.

Интерфейс среды проектирования Shell51 предоставляет необходимые средства загрузки и отладки прикладных программ.

Совокупность устройств дискретного ввода-вывода в стенде представлена клавиатурой 4x4 (устройство ввода) и блоком жидкокристаллических индикаторов ЖКИ (устройство вывода). Поскольку и клавиатура, и ЖКИ требуют программного управления, то по отношению к МК они являются объектами управления, что и отражено на схеме в их обозначении. Совокупность устройств аналогового ввода-вывода представлена регулируемым источником постоянного напряжения (потенциометром) и интегрирующим RC-звеном.

В качестве генераторов тестовых воздействий использованы источник гармонического сигнала «С» и источник импульсных сигналов «П» (на схеме 4.1 источники тестовых воздействий не показаны). При формировании гармонических сигналов обеспечена возможность регулирования частоты и амплитуды. Регулируемыми параметрами генератора импульсных сигналов является частота и скважность;

Коммутационное поле позволяет задавать структуру аппаратных связей между элементами стенда и портами ввода-вывода МК. Внешние соединения реализуются с помощью набора проводников (специальных «шин»), прилагаемого к аппаратуре стенда.

4.2. Устройства дискретного ввода-вывода лабораторного стенда

4.2.1. Клавиатура

Элементы ручного управления (кнопки, переключатели, различные клавиатуры) относятся к устройствам ввода цифровой информации в процессорные блоки. Элементы управления располагаются на панели управления и связываются с портом процессорного блока с помощью

[1] Среда проектирования программного обеспечения микропроцессорных систем Shell51 разработана доцентом кафедры КСПТ СПбГПУ Васильевым А.Е.

кабельного соединения. Для подключения каждого переключателя требуется, по крайней мере, одна сигнальная линия и разряд порта. Чем больше элементов ручного управления, тем больше число сигнальных линий, связывающих пульт управления с процессорным блоком. Для сокращения количества сигнальных линий переключатели конструктивно объединяют в клавиатуры. Используют либо простые клавиатуры с механическим переключением, либо сенсорные клавиатуры с электрической фиксацией одиночного касания.

Нажатие клавиш клавиатуры осуществляется в произвольные моменты времени, т.е. является асинхронным событием. На практике используются следующие режимы асинхронного обслуживания клавиатуры:

- ожидание асинхронного ввода;
- периодическое сканирование состояния клавиатуры;
- по прерыванию.

В первом случае предполагается, что сканирование происходит постоянно и является центральной функцией программы, обслуживающей МК-систему. Все операции, реализуемые процессорным блоком, зависят от команд, формируемых оператором с помощью клавиатуры.

В режиме периодического сканирования через определенные промежутки времени осуществляется опрос клавиатуры. Период сканирования должен быть коротким ($Tc \leq 0{,}1c$), но достаточным для фиксации нажатия клавиши с учетом дребезга и удержания. При большом периоде сканирования ($Tc > 0{,}1c$) для получения необходимой реакции может потребоваться неоднократное нажатие клавиши. При обнаружении нажатой клавиши определяется ее номер и выполняется соответствующая команда обслуживания.

В режиме обслуживания клавиатуры по прерываниям при нажатии клавиши выполняется обслуживание, как в предыдущем случае. Но поскольку обслуживание не связано с фиксированным периодом сканирования, в промежутках между запросами прерывания процессор может выполнять любые действия, предусмотренные программой.

При выполнении лабораторных работ данного цикла предполагается использование первого режима обслуживания клавиатуры - режима ожидания асинхронного ввода.

Используемый в лабораторном стенде вариант схемы объединения 16-ти клавиш (кнопок) пульта в клавиатуру показан на рис.4.2. Клавиши объединены в группы по четыре в каждой группе. Общий вывод каждой группы клавиш через диод соединен с одной из четырех входных шин клавиатуры. Отдельные выводы клавиш группы соединены с соответствующими выводами клавиш других групп и подключены к выходной шине клавиатуры.

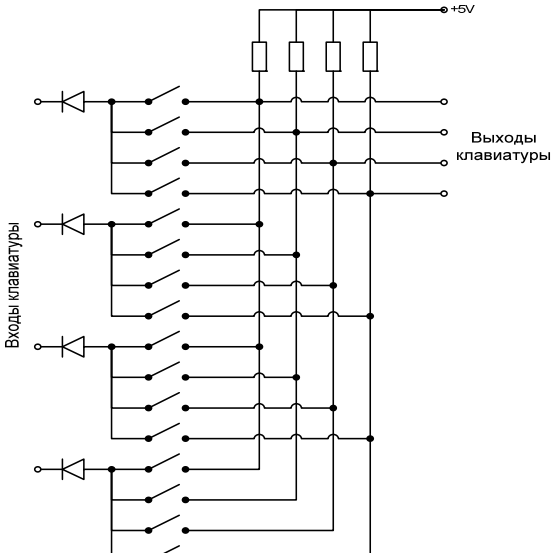


Рис.4.2. Схема объединения отдельных кнопок (клавиш) в клавиатуру

Для определения состояния клавиатуры (было ли нажатие какой-либо клавиши) необходимо сформировать сигналы опроса клавиатуры и считать ее состояние. Опрос клавиатуры удобно выполнить, организовав «бегущий нуль» по строкам клавиатуры (входное воздействие), с одновременным считыванием состояния столбцов (выходную реакцию). Результат опроса клавиатуры записывается во внутреннюю память МК, например, по адресам 30h - 33h. Указанную область памяти МК можно назвать картой памяти клавиатуры.

Входные воздействия и выходная реакция при нажатии только одной клавиши показаны в табл.10.

Для подключения клавиатуры к МК необходимо использовать два порта МК: порт ввода, подсоединяемый к выходной шине клавиатуры и порт вывода, соединяемый с входной шиной клавиатуры. Вариант схемы подключения клавиатуры к МК показан на рис.4.3. Входная шина клавиатуры соединена с выходным портом P4 МК, а выходная шина клавиатуры соединена с портом ввода P1. Через порт P4 на вход клавиатуры подается входное воздействие, а с помощью порта P1 фиксируется ее состояние.

Таблица 10

Входной код	Опрашиваемая клавиша	Выходной код
0111 xxxx	0, 1, 2 или 3	xxxx 0111 - нажата клавиша левого столбца
1011 xxxx	4, 5, 6 или 7	xxxx 1011- нажата клавиша среднего левого столбца
1101 xxxx	8, 9, 10 или 11	xxxx 1101- нажата клавиша среднего правого столбца
1110 xxxx	12, 13, 14 или 15	xxxx 1110 - нажата клавиша правого столбца xxxx 1111 - не нажата ни одна клавиша
		Оставшиеся коды свидетельствуют, что нажаты две или большее количество клавиш

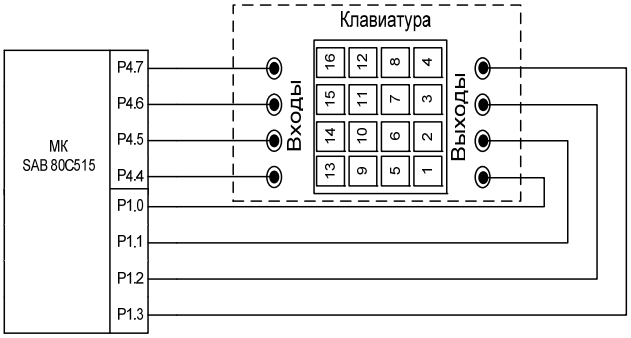


Рис.4.3. Вариант схемы подключения клавиатуры к МК

Тестовая программа klav.asm позволяет проиллюстрировать программный способ определения состояния клавиатуры. Схема программы klav.asm приведена на рис.4.4, а ее текст – на рис.4.5.

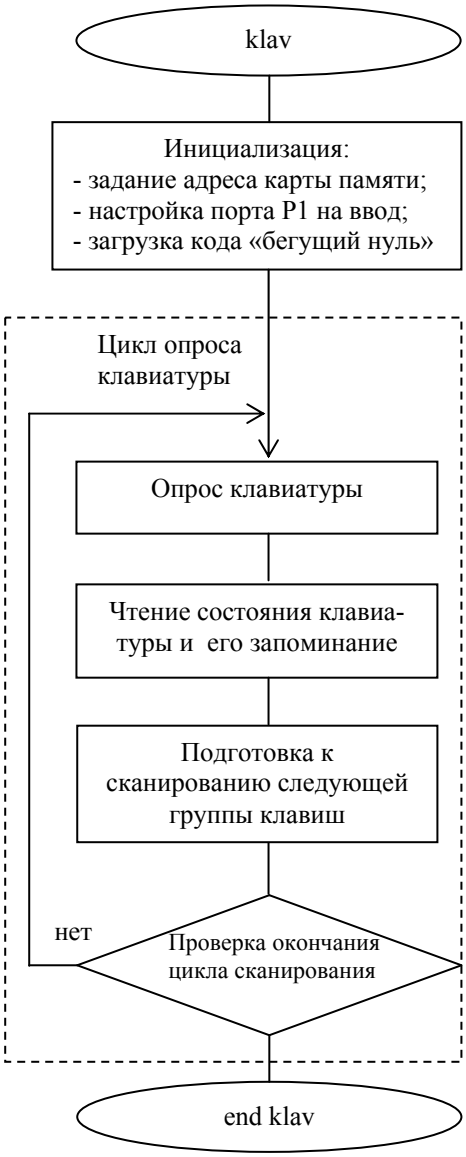


Рис.4.4. Схема программы опроса клавиатуры

```
      ;Программа klav.asm

            org 8100h
            p4: equ 0E8h
klav:       mov r0, #30h      ; инициализация адреса карты
                              ; памяти клавиатуры

            orl P1, #0fh      ;настройка порта P1 на ввод.

            mov a, #7fh       ; загрузка исходного кода
                              ; «бегущий нуль»

            ;цикл сканирования клавиатуры

m1:         mov p4, a         ; загрузка и временное запоминание
            mov r2, a         ; кода "бегущий нуль"

            mov a, p1         ; чтение состояния клавиатуры
            anl a, #0fh

            mov @r0,a         ; заполнение карты памяти

            inc r0            ; формирование следующего адреса
                              ; карты памяти
            mov a,r2          ; формирование нового значения
            rr a              ; кода «бегущий нуль»

            cjne a, #f7h, m1  ; Проверка окончания цикла сканиро-
                              ; вания и продолжение опроса в
                              ;цикле, если сканирование не
                              завершено

            ret
```

Рис.4.5. Текст программы опроса

Для определения номера нажатой клавиши необходимо проанализировать содержимое карты памяти клавиатуры и выполнить преобразование считанных кодов состояния клавиатуры. В тестовой программе klav.asm такая процедура не выполняется.

4.2.2. Алфавитно-цифровой модуль жидкокристаллических индикаторов

Индикация как элемент внешнего интерфейса МК-системы предназначена для вывода цифровых значений результатов вычислений, а также диагностических и информационных сообщений. Среди разнообразных средств индикации важное место занимают жидкокристаллические индикаторы (ЖКИ), обеспечивающие унифицированный интерфейс отображения различной информации. В настоящее время номенклатура ЖК-индикаторов достаточно разнообразна. Популярны ЖК-индикаторы фирм Seiko, Power Tip, Electronic Semiconductor и др. Алфавитно-цифровые ЖК-индикаторы с встроенным контроллером управления типа HD44780 и унифицированной системой команд имеют напряжение питания 5В, потребляют ток порядка 200 мА, обеспечивают управление подсветкой. Конструктивно модуль ЖКИ выполняется в виде дисплейного табло, которое содержит 1-, 2- или 4-строчный набор индикаторов с матрицей символа 5x7 или 5x10 точек и длиной строки 16-40 символов. В лабораторный стенд встроен модуль ЖКИ фирмы Sanyo - модель DM2021 (рис.4.6).

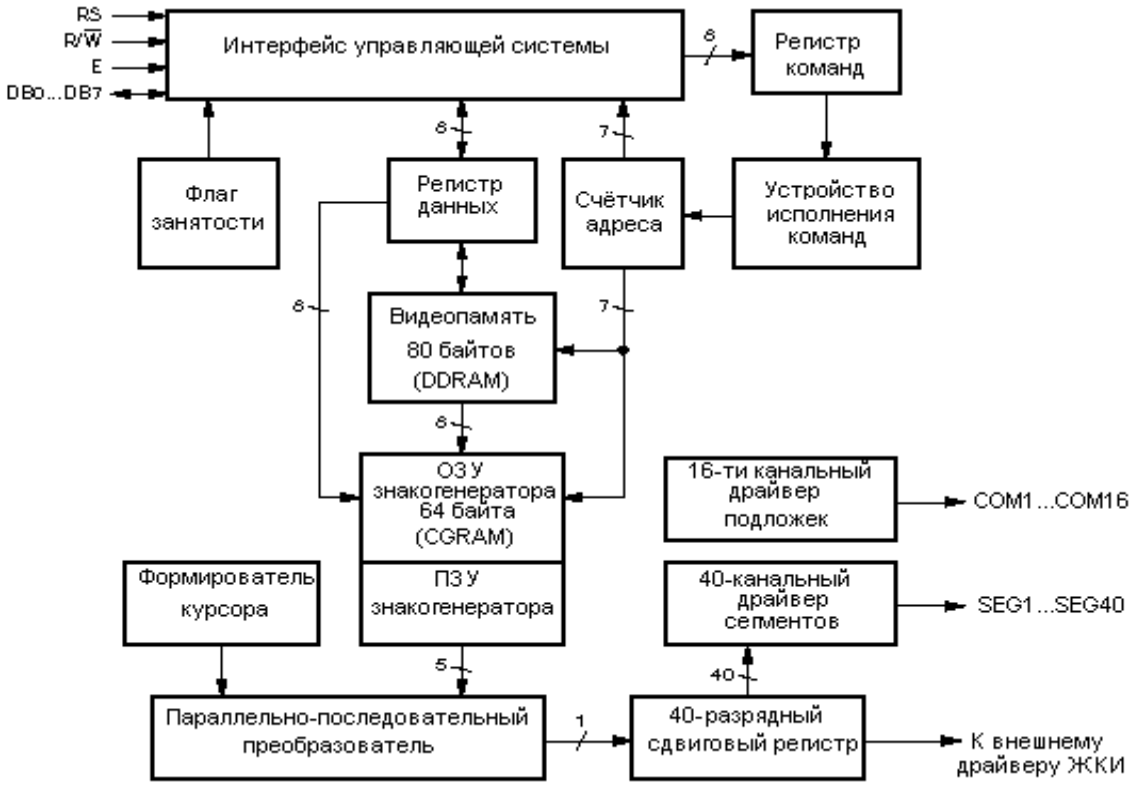


Рис.4.6. Упрощённая структурная схема контроллера управления модулем ЖКИ

Дисплей этой модели имеет двустрочное табло ЖК-индикаторов по 20 символов в каждой строке. Для соединения модуля ЖКИ с МК используются 8-разрядная двунаправленная шина DB7-DB0 и 3 линии сигналов управления физическим интерфейсом модуля (линии *RS*, $R/\overline{W}$ и *E*). Кроме линий управляющей шины, имеются две линии для подачи напряжения питания 5 В и линия напряжения *VO*. При регулировке *VO* изменяется угол поворота жидких кристаллов, благодаря чему обеспечивается требуемая контрастность изображения для некоторого преимущественного угла наблюдения (снизу вверх или сверху вниз). Основными объектами взаимодействия при управлении модулем ЖКИ являются регистры данных (DR) и команд (IR), счётчик адреса памяти AC, флаг занятости BF.

Отображаемые на ЖК-дисплее символы представляются в виде битовых матриц 5x7 точек. Коды отображаемых символов хранятся в памяти кодов символов (ПЗУ знакогенератора). ПЗУ хранит информацию о 192-х символах (табл.11). Каждый символ идентифицируется собственным HEX-кодом (номером). Младшая половина кодов таблицы символов содержит коды букв латинского алфавита, десятичных цифр, знаков препинания и некоторых специальных символов (коды 00h-7Fh). HEX-коды названных символов совпадают с их ASCII-кодами. Старшая половина кодов таблицы хранит коды символов национальных алфавитов, дополнительных знаков препинания и символов псевдографики. В этой части таблицы хранятся не все символы кириллицы, а только те, которые нельзя представить схожими по написанию символами латиницы.

Для вывода информации на экран ЖК-дисплея коды отображаемых символов необходимо переписать в видеопамять DDRAM или ОЗУ знакогенератора CGRAM. Процедура вывода отображаемого символа запускается при загрузке HEX-кода символа в регистр DR, при этом в зависимости от значения HEX-кода схемы управления подключают ЖК-индикаторы к видеопамяти DDRAM или к ОЗУ знакогенератора CGRAM.
Видеопамять DDRAM организована в виде двух строк по 40 байт в каждой строке. Для адресации памяти DDRAM используются 7-битные адреса счетчика AC. При выводе очередного символа содержимое AC увеличивается или уменьшается на 1. Организация видеопамяти в виде двух строк по 40 байт в каждой строке является жёсткой и не зависит от числа строк дисплея конкретного модуля ЖКИ. Особенностью видеопамяти DDRAM модуля ЖКИ DN2021, используемого в лабораторном стенде, является то, что она представляется как бы разрывной: элементы первой строки имеют адреса в диапазоне 00 . . . 27h, а элементы второй строки - в диапазоне 40h . . . 67h. Значения элементов DDRAM в диапазоне адресов 28h . . .3Fh и 88h . . .7Fh являются неопределенными.

HEX-коды отображаемых символов Таблица 11

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p			Б	Ю	ч	.	Д	Ч
1			!	1	A	Q	a	q			Г	Я	ш	ı	Ц	Ъ
2			"	2	B	R	b	r			Ё	б	ъ	‼	Щ	Ч
3			#	3	C	S	c	s			Ж	в	ы	‼	Д	Ъ
4			$	4	D	T	d	t			Э	г	ь	Ў	ф	Ы
5			%	5	E	U	e	u			И	ё	э	ж	Ц	¯
6			&	6	F	V	f	v			й	ж	ю	ў	Щ	¶
7			'	7	G	W	g	w			Л	з	я	I	´	Ь
8			(	8	H	X	h	x			П	и	«	II	¨	‡
9			)	9	I	Y	i	y			У	й	»	↑	~	⁞
A			*	:	J	Z	j	z			Ф	к	„	↓	é	Ъ
B			+	;	K	[	k	10			Ч	л	”	Ж	ç	‡
C			,	<	L	¢	l	12			Ш	м	№	Ж	ї	Ж
D			−	=	M	]	m	15			Ъ	н	¿	Ж	♣	§
E			.	>	N	^	n	↵			Ы	п	ѓ	Ў	°	¶
F			/	?	O	_	o	←			Э	т	€	·	o	■

Память CGRAM емкостью 64 байт предназначена для запоминания 8-битных кодов графических символов, которые могут формироваться самим пользователем. Обращение к памяти CGRAM происходит при загрузке в регистр DR HEX-кодов 00 – 07. Для адресации элементов памяти CGRAM используются 6 младших разрядов счетчика AC

Модуль ЖКИ имеет несколько режимов отображения. Назначение режима отображения осуществляется путем программирования управляющих бит (флагов) регистра IR. Комбинации битов этого регистра (табл. 12) определяют список команд управления модулем. При инициализации ЖКИ соответствующие команды управления записываются в регистр IR.

Типовыми командами управления модулем ЖКИ являются:

- команда 0Ch, выбирающая режим отображения экрана без индикации курсоров;
- команда 38h, задающая режим отображения, при котором устанавливается 8-битовый режим обмена с выводом обеих строк с размером матрицы символов 5*7 точек;

Команды управления модулем ЖКИ Таблица 12

D7	D6	D5	D4	D3	D2	D1	D0	Выполняемая командой функция
0	0	0	0	0	0	0	1	Команда очистки экрана, AC=0, AC адресует DDRAM
0	0	0	0	0	0	1	-	Команда адресации DDRAM: AC=0, AC адресует DDRAM, начало дисплейной строки адресуется в начале DDRAM, сдвиги сброшены
0	0	0	0	0	1	I/D	S	Команда выбора режима смещения содержимого экрана (флаг S) и режима увеличения или уменьшения содержимого счетчика AC на 1 (флаг I/D): I/D=1 – режим увеличения AC на 1, I/D=0 - режим уменьшения AC на 1. При S=0 сдвиг изображения не производится, при S=1 сдвиг изображения вправо при I/D=0 или влево при I/D=1.
0	0	0	0	1	D	C	B	Команда выбора режима отображения: D - флаг включения изображения (D=0 – выключено, D=1 – включено), C - флаг курсора в виде подчеркивания (C=0 – выключено, C=1– включено), B - флаг курсора в виде мерцающего знакоместа (B=0 – выключено, B=1 – включено)
0	0	0	1	S/C	R/L	-	-	Команда сдвига курсора/экрана (флаг S/C) и направление сдвига (флаг R/L): S/C = 0 – сдвигается курсор, S/C = 1 – сдвигается экран; R/L = 0 – сдвиг влево, R/L = 1 – сдвиг вправо.
0	0	1	D/L	N	F	-	-	Команда задания параметров развертки (флаг N), ширины шины данных (флаг D/L) и размера матрицы символов (флаг F): D/L=0 – 4 разряда, D/L=1 – 8 разрядов; N=0 - одна строка, N=1 – две строки; F=0 – 5x8 точек, F=1 - 5x10 точек
0	1	AG5	AG4	AG3	AG2	AG1	AG0	Команда присвоения счетчику AC адреса в области CGRAM
1	AD6	AD5	AD4	AD3	AD2	AD1	AD0	Команда addr_dram присвоения счетчику AC адреса в области DDRAM

- команда *addr_dram*, обеспечивающая присвоение счетчику AC адреса в области DDRAM. С помощью команды *addr_dram* выбирается место размещения первого отображаемого символа на экране ЖКИ. Часто используемыми кодами команды *addr_dram* являются коды 80h и C0h. Команда 80h загружает в счетчик AC адрес 00h, адресуя нулевую ячейку первой строки DDRAM, а команда C0h загружает в счетчик AC адрес 40h, адресуя нулевую ячейку второй строки. Чтобы произвести переустановку размещения текста на нужную позицию экрана необходимо присвоить AC требуемое значение, т.е. выполнить соответствующую команду *addr_dram*.

После инициализации модуль ЖКИ подготовлен к выводу информации.

Процедура вывода заданного текста на экран ЖКИ включает инициализацию модуля ЖКИ (пересылку команд управления в регистр IR) и собственно отображение символьной информации (пересылку HEX-кодов символов в регистр DR и их индикацию). Для организации шины DB7-DB0 назначается специальный порт МК. Управление обменом по шине DB7-DB0 осуществляется по трехпроводной шине управления физическим интерфейсом модуля. Сигналы управления интерфейсом $R/\overline{W}$, *RS* и *E* формируются на выводах еще одного порта МК. Сигнал $R/\overline{W}$ указывает направление передачи информации: при $R/\overline{W}$=1 осуществляется чтение данных из ЖКИ, при $R/\overline{W}$=0 выполняется запись данных в ЖКИ. С помощью сигнала RS производится выбор регистра ЖКИ, участвующего в обмене. При *RS*=0 адресуется регистр команд *IR*: на модуль поступают команды или из него считывается байт состояния. При *RS*=1

осуществляется обмен данными между регистром данных DR и МК. Поскольку сигнал *RS* определяет тип передаваемой информации, его часто называют сигналом команда/данные. Сигнал *E* служит для синхронизации работы модуля ЖКИ. С помощью этого сигнала фиксируется элементарное действие выполняемой операции.

Пересылка информации независимо от ее типа (команда или данные) реализуется программной функцией записи в ЖКИ – *ind_wr*, включающей следующую последовательность элементарных действий:

1. на шине DB7-DB0 устанавливается записываемая в ЖКИ информация (команда или данные), а на трехпроводной шине управления - значения сигналов, определяющих реализуемую операцию: на входе $R/\overline{W}$ устанавливается сигнал записи ($R/\overline{W}$=0), на входе RS - значение, соответствующее типу записываемой информации (RS=0 при записи команды и RS=1 при записи данных), на входе E - сигнал E=1. Минимально необходимое время установки интерфейсных сигналов $R/\overline{W}$, *RS* и *E* фиксируется задержкой *delay*, формируемой программно.

2. на выводе *E* формируется строб - сигнал низкого уровня, фиксирующий элементарную операцию записи. Длительность строба *E* определяется задержкой *delay*.

3. возврат системы управления ЖКИ в исходное состояние осуществляется по окончанию строба *E* при выполнении команды установки сигнала *E* в состояние «1».

Программная настройка физического интерфейса на требуемую операцию обмена осуществляется для конкретной схемы подключения модуля ЖКИ к МК. На рис.4.7 показан один из вариантов подключения модуля ЖКИ к МК SAB80C515.

Для схемы подключения модуля ЖКИ к МК SAB80C515 (рис.4.7) в качестве примера приведены алгоритм (рис.4.8) и текст программы *indic.asm* (рис.4.9), осуществляющей вывод на экран ЖКИ заданного текстового сообщения, представленного в виде двух строк по 20 символов в каждой строке. Коды символов сообщения размещены во внешней памяти МК

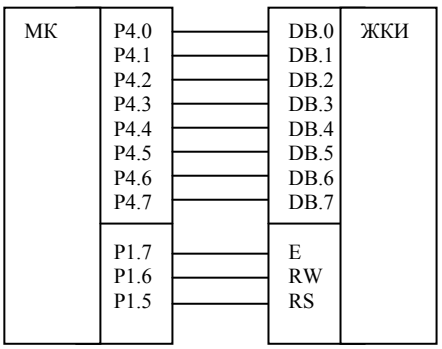


Рис.4.7. Схема подключения модуля ЖКИ к МК SAB80C515

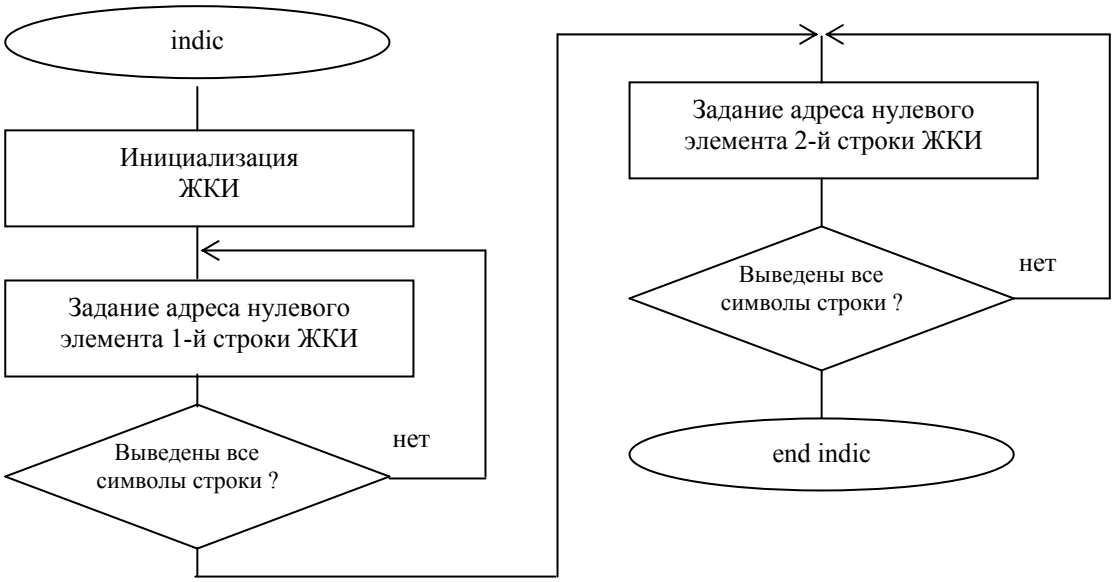


Рис.4.8. Схема программы *indic.asm* вывода текста на экран ЖКИ

В программе *indic.asm* (см. рис. 4.9) использованы следующие назначения:

r4 – регистр временного хранения байта информации, пересылаемого в контроллер ЖКИ. При передаче содержимое *r4* переписывается в порт P4. Выводы порта P1 используются для передачи сигналов управления модулем ЖКИ. Сигнал ES, формируемый на выводе P1.5 определяет тип информационной посылки (0-команда, 1- данные). Сигнал $R/\overline{W}$ на выводе P1.6 определяет выполняемую операцию (0 – запись, 1 – чтение). Сигнал E на выводе P1.7 является стробом управления.

Инициализация подготавливает внутренние схемы модуля ЖКИ для отображения текстовой информации в первой строке дисплея. Инициализация осуществляется при выполнении соответствующих команд управления модулем ЖКИ. Запись команд, как и последующая запись данных, выполняется подпрограммой ind_wr (см. текст программы *indic.asm*), реализующей требуемый протокол управления ЖКИ при записи. В соответствии с рекомендациями производителей время выполнения отдельных процедур записи (время выдержки сигналов $R/\overline{W}$ и *RS* относительно сигнала *E*) должно быть не меньше 40 мкс. Задержка *delay* обеспечивает требуемые временные параметры выполнения процедуры записи

Сигналы аппаратного интерфейса ЖКИ формируются на соответствующих выводах порта P1.

Вывод данных в строку дисплея реализуется в цикле считывания кодов символов из соответствующей строки видеобуфера (внешней памяти МК) и вывода их на экран ЖКИ. Цикл вывода предваряют команда задания адреса видеобуфера и команда настройки модуля для работы с данными (см. текст программы *indic.asm* рис. 4.9).

```
;программный модуль indic.asm, осуществляющий вывод на экран ЖКИ двухстрочного текстового
;сообщения, размещенного по адресам str1 и str2 внешней памяти, задаваемых оператором org addr.

indic:  clr P1.5          ;подготовка к вводу команд (R/S=0)
        mov r4,#38h       ; код команды установки 8-битового режима обмена с выводом
                          ; обеих  строк символов
        lcall ind_wr      ; запись команды в ЖКИ
        mov r4,#0Ch       ; код команды активизации всех знакомест дисплея без индикации  курсора
        lcall ind_wr      ; запись команды в ЖКИ
        mov r4,#80h       ; код команды загрузка в счетчик АС адреса нулевой ячейки 1-ой  строки
        lcall ind_wr      ; запись команды addr_dram 80h в ЖКИ

        mov dptr,#FFD0h   ; задание адреса видеобуфера, содержащего коды  отображаемых  символов
        setb P1.5         ; подготовка к вводу данных (R/S=1)

; цикл считывание символов 1-й строки из видеобуфера и вывод их на экран 1-й строки дисплея ЖКИ
wr_str1: movx a,@dptr     ;считывание символа из внешней памяти
        mov r4,a
        lcall ind_wr      ; запись данных в ЖКИ
        inc dptr          ; формирование следующего адреса видеобуфера
        mov a,dpl
        cjne a,#0E4h,wr_str1  ; проверка окончания вывода всех символов 1-ой строки
; E4h – адрес ячейки видеобуфера, соответствующей  последнему символу 1-ой строки   E4h = D0h + 20

; настройка ЖКИ для вывода данных во 2-ю строку дисплея
        clr P1.5  ; подготовка к вводу команд (R/S=0)
        mov r4,#C0h       ; код команды загрузки в счетчик АС адреса нулевой ячейки 2-й  строки
        lcall ind_wr      ; запись команды addr_dram C0h в ЖКИ

        setb P1.5         ;подготовка к вводу данных (R/S=1)
;цикл считывание символов 2-й строки из видебуфера и вывод их на экран 2-й строки
wr_str2: movx a,@dptr     ;считывание символа из внешней памяти
        mov r4,a
        lcall ind_wr      ;запись данных в ЖКИ
        inc dptr          ;формирование следующего адреса видеобуфера
        mov a,dpl
        cjne a,#0F8h,wr_str2  ;проверка окончания вывода всех символов 2-ой строки
; F8h– адрес ячейки видеобуфера, соответствующей  последнему символу 2-ой строки   F8h = E4h + 20
        ret

; функция записи байта информации в ЖКИ
ind_wr: mov E8h, r4       ; загрузка в порт P4, передаваемой информационной посылки
        setb p1.7         ; установка сигнала E
        clr p1.6  ; формирование сигнала R/W=0 (запись)
        lcall delay       ; формирование необходимой задержки
        clr p1.7  ; сброс сигнала E
        lcall delay
        setb p1.7         ; установка сигнала E
        ret

delay:  mov r3, #7        ; подпрограмма формирования задержки delay
m1:     djnz r3, m1
        ret

        org FFD0h         ; резервирование внешней памяти для видеобуфера.

str1: db  'SPbSPU_ ICC_ CSPT_ 201_'
str2: db 4Dh, B8h, BAh, 70h, 6Fh, BAh, 6Fh, BDh, BFh, 70h, 6Fh, BBh, BBh, 65h, 70h, C3h,32h, 30h, 31h, 34h
```

Рис.4.9. Текст программы *indic.asm* вывода буквенно-цифровой информации на экран дисплея ЖКИ

Для наглядности видеобуфер представлен в виде двух строк HEX-кодов отображаемых символов. Поскольку ASCII-коды латинских букв и десятичных цифр совпадают с HEX-кодами одноименных символов ЖКИ модуля, для отображения текста из латинских букв и десятичных цифр его достаточно заключить в апострофы. Для отображения букв русского алфавита необходимо явно указывать HEX-коды одноименных символов.

С помощью директивы *include <имя файла>* модуль индикации может подключаться к другим программам. Поскольку в подключаемых модулях рекомендуется избегать необязательного использования абсолютных адресов, задаваемых оператором *org address*, в программе *indic.asm* оператор *org FFD0h* желательно исключить. Это можно сделать, если разместить массив отображаемых символов с именем str1 в основной программе. При компиляции исходного текста программы ассемблер автоматически зарезервирует необходимое место для данных видеобуфера. Текст модифицированной программы несколько изменится. Для отображения двухстрочного текста в модифицированную программу необходимо добавить команды определения и запоминания адресов последних ячеек каждой строки видеобуфера и использовать вычисленные адреса при сравнении текущего адреса видеобуфера с адресом последнего символа строки в командах проверки окончания цикла вывода символов строки.

```
        mov dptr, #str1+20
        mov 50h,dpl
        mov dptr, #str1+40
        mov 51h,dpl
              :
        cjne a,50h,wr_str1
              :
str1: db  'SPbSPU_ ICC_ CSPT_ 201_'
str2: db 4Dh, B8h, BAh, 70h, 6Fh, BAh, 6Fh, BDh, BFh, 70h, 6Fh, BBh, BBh, 65h, 70h, C3h,32h, 30h, 31h, 34h
```

Альтернативным способом организации цикла вывода символов строки на экран является способ, который не требует предварительного вычисления и запоминания адреса последней ячейки строки видеобуфера. В нем после выполнения каждой процедуры цикла декрементируется счетчик числа повторений и осуществляется сравнение оставшегося значения с «нулем». Счетчик числа повторений предварительно необходимо инициализировать требуемым числом повторений. Ниже представлен фрагмент программы такого способа организации цикла.

```
; инициализация цикла вывода строки символов
mov dptr,#str1   ; задание адреса видеобуфера, содержащего коды отображаемых
                  ;символов
setb P1.5        ; подготовка к вводу данных (R/S=1)
mov r0, #20      ; задание числа символов в строке
```

```
wr_str: movx a,@dptr   ;считывание символа из внешней памяти
        mov r4,a
        lcall ind_wr     ;запись данных в ЖКИ
        inc dptr                 ;формирование следующего адреса видеобуфера
        djnz r0, ,wr_str ;проверка окончания вывода всех 20-ти символов строки
```

5. Программа работ лабораторного практикума

5.1.Программа первого цикла работ: «Изучение вычислительных возможностей МК»

Цель работы:

- знакомство с программно-аппаратным комплексом поддержки проектирования микроконтроллерных систем на базе МК SAB80C515;
- изучение системы команд МК семейства MCS51 на примере выполнения простейших программ.

Программа работы:

Изучите состав и назначение основных устройств и блоков программно-аппаратного комплекса поддержки проектирования микроконтроллерных систем на базе МК семейства MCS51 (среды проектирования Shell51).

Выполните полный цикл создания прикладного программного обеспечения на примере выполнения тестовой программы proba.asm, которая осуществляет обнуление ячеек заданной области внутренней памяти данных МК.

```
        ; proba.asm
        org 8400h
        mov a,#0h
        mov r0,#50h
m1:     mov @r0,a
        inc r0
        cjne r0,#60h,m1
        ret
```

Для уяснения общего назначения программы и сути каждого оператора (директив ассемблера и ассемблерных команд) используйте справочную информацию, представленную в разделе программирование МК. Дополните текст программы смысловыми комментариями к выполняемым командам. В отчете приведите ответы на следующие вопросы:

- какие способы адресации операндов используются в командах программы?
- какую функцию реализует директива org 8400h?
- с помощью какой команды в программе реализуется цикл? Поясните действие этой команды.

При загрузке, трансляции и чтении результатов выполнения программы пользуясь правилами-рекомендациями, приведенными в информационном окне рабочего поля экрана инструментальной ВМ, изучите основные функции среды Shell51 и способы их применения.

Программа proba.asm, как и другие программы вводного занятия, не предполагает использование устройств ввода/вывода микроконтроллерной

системы. Проверка работоспособности такой программы (ее отладка) может быть выполнена как с использованием симулятора (программной модели МК), так и при непосредственном запуске программы на МК и контроле результатов выполнения на инструментальной ВМ (при загрузке данных из МК в инструментальную ВМ с помощью программных средств оболочки Shell51 – опции «Передать в ПК»). Проверьте работу программы обоими способами, сравните результаты.

Для заполнения ячеек заданной области внутренней памяти данных МК линейно возрастающими (линейно уменьшающимися) значениями **модифицируйте программу proba.asm и выполните** ее. Диапазон области адресуемой памяти и значения, записываемые в конкретные ячейки памяти, выберите самостоятельно.

Знакомство с системой команд МК семейства MCS51 на примере выполнения простейших программ:

Разработайте и выполните программу вычисления арифметического выражения заданного вида. Исходные данные – байтовые переменные. Операнды и результат вычисления разместите в ячейках внутренней памяти данных. При выполнении данного и последующих заданий номер варианта задания совпадает с номером рабочего места

Варианты задания исходных данных программы «вычисление арифметического выражения заданного вида»:

1. F = A * [0,5 * (B + C) - 2B / C], где B <=127 0,5 * (B + C) > 2B / C;
2. F = 1/2 (A + B) * (C - A/B), где C > A/B;
3. F = 1/4 (A + 2B) * (B - B/A), где B<=127;
4. F = (AB - B)/A * (2A + B), где A <> 0;
5. F = 2A * [(B - C) + B/2C], где A <=127; B > C;
6. F = [(A - B)/ (C + 1) + 12h] * C, где A > B; C + 1 <= 127;
7. F = A + [(B + C) + A * C]
8. F = A * [(B / C) + B * C] / 2, где C # 0;
9. F =[(A * C) + B - C] / A, где A # 0;
10. F =[(A - C) * B – 10h] / A, где A > C, (A - C) * B <= 255;

Разработайте и выполните программу вычисления логического выражения заданного вида с использованием команд булевого процессора. Исходные данные – битовые переменные. Операнды и результат вычисления разместите в ячейках внутренней памяти данных битового процессора (в диапазоне адресов 20h –2Fh).

Варианты задания исходных данных программы «вычисление логического выражения заданного вида»:

1. Y = (a AND (NOT b)) XOR (c OR d)
2. Y = aXOR (b AND c)) OR d
3. Y = (a AND (NOT b)) XOR (c AND d)
4. Y = ((NOT a) AND c) XOR (b AND d)
5. Y = (a AND (NOT b)) XOR (c AND d)
6. Y = (a XOR b) AND (c AND d)
7. Y = (a AND (NOT b)) OR (c XOR d)
8. Y = (a OR (NOT b)) XOR (c OR d)
9. Y = (a AND (NOT c)) XOR (a AND d)
10. Y = ((NOT a) AND (NOT b)) OR (c XOR d)

Здесь a, b, c, d - битовые переменные.

Поскольку в системе команд битового процессора отсутствует команда XOR, при реализации операции XOR пользуйтесь выражением: $a \text{ XOR } b = \overline{a} * b + a * \overline{b}$.

Разработайте и выполните программу, которая осуществляет заполнение последовательных ячеек **внешней памяти** значениями, линейно изменяющимися в заданных диапазонах.

Варианты задания исходных данных программы «заполнение последовательных ячеек внешней памяти значениями, линейно изменяющимися в заданном диапазоне»:

1. 0 – 10h – 0
2. 10h – 0 – 10h.
3. 20h – 30h –20h
4. 30h – 20h – 30h
5. 10h – 18h – 08h –10h
6. F8h – FFh – F0h – F8h
7. 70h – 60h – 70h
8. 30h – 28h – 38h – 30h
9. 30h – 38h – 28h – 30h
10. 20h – 18h – 28h –20h

Разработайте и выполните программу функциональной обработки данных. Если по заданию необходимо работать с массивами, они должны быть расположены во внешней памяти данных по адресам, задаваемым в программе.

Варианты задания программа «функциональной обработки данных»:

1. Поиск минимального, максимального и среднего арифметического в массиве.
2. Определение общих для двух массивов элементов, с занесением их в новый массив.
3. Выбор из массива элементов, делящихся нацело на К, с занесением их в новый массив.

4. Подсчет числа вхождений заданной константы в массив.
5. Определение наибольшего общего делителя двух чисел.
6. Определение наименьшего общего кратного двух чисел.
7. Вычисление k-го элемента ряда Фибоначчи.
8. Целочисленное вычисление квадратного корня числа X. При выполнении задания используйте представление квадрата числа суммой k членов ряда $k^2=1+3+... + (2k-1)$.

5.2. Программа второго цикла работ: «Работа с портами МК»

Цель работы:

- знакомство с организацией взаимодействия МК с внешними устройствами (цифровыми и аналоговыми), подключаемыми с помощью портов МК;
- овладение приемами программирования периферийных устройств, входящих в состав лабораторного стенда.

При выполнении работ данного цикла необходимо разработать структуру информационных связей МК с подключаемыми внешними устройствами (клавиатурой, модулем ЖКИ, датчиками аналоговых сигналов) и реализовать ее, соединив разъемы портов МК с разъемами подключаемых устройств. Существует сравнительно большое число вариантов подключения внешних периферийных устройств к МК. При выполнении данного и следующих циклов работ структура информационных связей МК с подключаемыми внешними устройствами определяется вариантом задания, представленным в табл.13. Номер варианта совпадает с номером Вашего рабочего места.

В качестве примера на рис.5.1 приведена структура информационных связей МК с подключаемыми внешними устройствами, реализованная в соответствии с заданием варианта 11.

Программа работы:

Познакомьтесь с организацией и принципом работы модуля ЖКИ (п.4.2.2).

Разработайте и выполните программу, реализующую вывод на экран ЖКИ двухстрочного текста (см. рисунок), в котором используются различные алфавиты (латиница и кириллица)

‘Microcontrollers20_ _ Фамилия студента(ов) в русской транскрипции

При разработке программы используйте модуль *indic.asm* (п.4.2.2). Результат выполнения представьте в отчете.

Варианты назначения портов ввода-вывода МК,
используемых при подключении внешних устройств

Таблица 13

N	Выходной порт МК (вход клавиатуры)	Порт ввода МК (выход клавиатуры)	Шина DB ЖКИ	Шина управления ЖКИ RS R/W E	Входы подключения аналоговых сигналов Потенц-р	Интегр-р	Линия порта, на которой формируется меандр (ШИМ-сигнал) (точка подключения осциллографа)
1.	P1.7 – P1.4	P5.7 – P5.4	P4	P5.0 P5.2 P5.3	ach0	ach7	P1.3
2.	P4.3 – P4.0	P1.7 – P1.4	P5	P4.4 P4.6 P4.7	ach1	ach6	P1.2
3.	P1.7 – P1.4	P5.7 – P5.4	P4	P5.0 P5.2 P5.3	ach2	ach5	P1.1
4.	P4.3 – P4.0	P4.7 – P4.4	P5	P1.**4** P1.6 P1.7	ach3	ach4	P1.1
5.	P5.7 – P5.4	P5.3 – P5.0	P4	P1.4 P1.6 P1.7	ach4	ach7	P1.2
6.	P5.3 – P5.0	P1.7 – P1.4	P4	P5.4 P5.6 P5.7	ach5	ach6	P1.3
7.	P4.7 – P4.4	P4.3 – P4.0	P5	P1.**4** P1.6 P1.7	ach6	ach0	P1.3
8.	P1.7 – P1.4	P5.7 – P5.4	P4	P5.0 P5.2 P5.3	ach7	ach1	P1.2
9.	P5.7 – P5.4	P1.7 – P1.4	P4	P5.0 P5.2 P5.**3**	ach3	ach6	P1.1
10.	P1.7 – P1.4	P4.7 – P4.4	P5	P4.0 P4.**2** P4.**3**	ach5	ach2	P1.2
11.	P4.7 – P4.4	P4.3 – P4.0	P5	P1.**4** P1.6 P1.7	ach0	ach1	P1.3

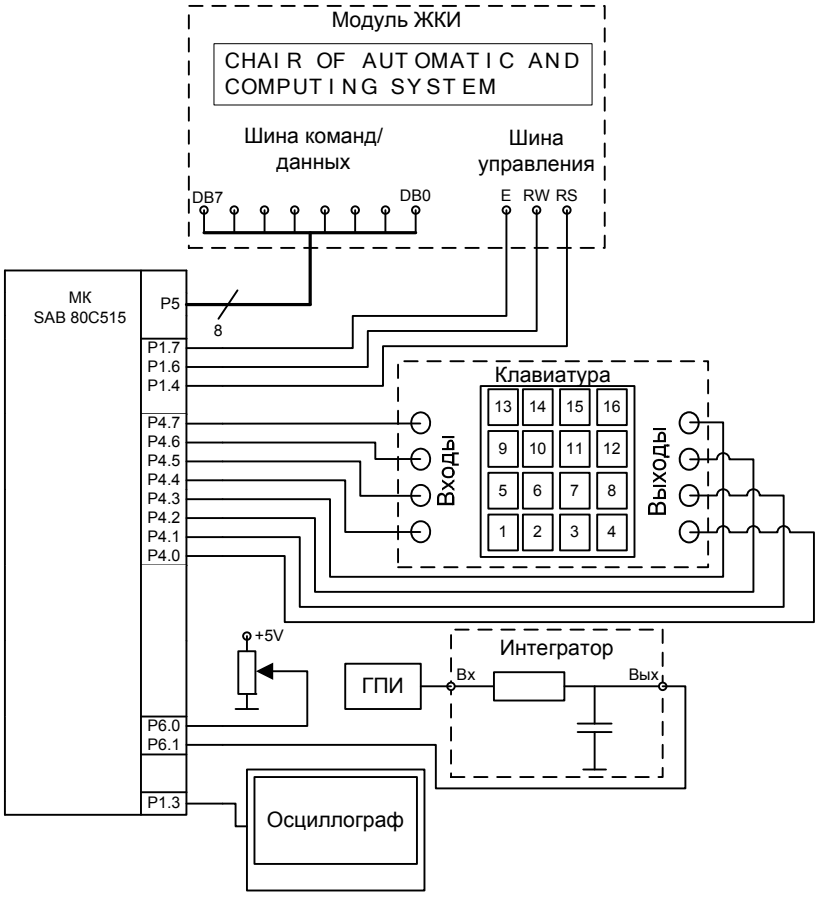


Рис.5.1. Структура информационных связей МК с подключенными внешними устройствами

Разработайте программу *klav.asm*, реализующую опрос клавиатуры и запись кодов состояния клавиш в ячейки памяти с адресами 30h . . . 33h.

Результат выполнения программы (содержимое ячеек 30h . . . 33h карты памяти клавиатуры) представьте в виде таблицы, отображающей состояние кодов клавиш при их нажатии, строки которой содержат следующую информацию:

Номер клавиши	Входное воздействие Выходы порта P_i $P_{i.j+3}$ $P_{i.j+2}$ $P_{i.j+1}$ $P_{i.j}$	Состояние карты памяти при нажатии клавиши, соответствующей строки (содержимое ячеек) 30h 31h 32h 33h

Модифицируйте программу «опрос клавиатуры», дополнив ее модулем, формирующим номер нажатой клавиши. Присвойте ей имя *klav1.asm*.

Если нажата одна клавиша в выбранную ячейку внутренней памяти МК, например 34h, записывайте 16-ричный или десятичный код номера нажатой клавиши. При отсутствии нажатых клавиш фиксируйте в ячейке 34h код 00, а при нажатии нескольких клавиш (более одной) – код FFh. Результат выполнения программы представьте в виде копии экрана вкладки «Окна памяти».

При разработке модифицированной программы *klav1.asm* целесообразно использовать директиву *include asms\klav.asm*, с помощью которой программный модуль *klav.asm* подключается к разрабатываемой программе.

Напомним, что директива *include <имя файла>* указывает ассемблеру на необходимость трансляции текста, расположенного в указываемом файле, начиная с адреса, следующего за адресом последней команды текущего файла. При использовании директивы include <имя файла> часто фиксируются ошибки, связанные с наложением переменных и участков программ (при любых вызовах по общим адресам формируемое обращение будет производиться к участку кода последнего оттранслированного модуля). Поэтому при использовании директивы include <имя файла> необходимо выполнять следующие требования:

- имя файла, указываемого в директиве *include <имя файла>* не должно превышать 8 символов;
- следует избегать абсолютных адресов в подключаемых файлах;
- нельзя использовать одинаковые метки в выполняемой программе и подключаемых файлах.

Разработайте программу, отображающую на экране модуля ЖКИ номер нажатой клавиши, используя программы indic.asm (п.4.2.2) и *klav1.asm*, подключив их к разрабатываемой программе директивой *include <имя файла>*.

При выводе текста «Number of button Nx» номер нажатой клавиши *N*x с помощью команды пересылки с прямой адресацией приемника предварительно должен быть помещен в видеобуфер: str1: ‘Number of button Nx’.

Изучение особенностей работы МК при подключении датчиков с аналоговым выходом.

В качестве источников аналоговых сигналов в стенде используются источник регулируемого напряжения, представленный сигналом с выхода потенциометра, и сигнал с выхода интегратора, на вход которого подается сигнал с выхода генератора прямоугольных импульсов (ГПИ) с регулируемой частотой и скважностью. Формирование цифровых кодов входных аналоговых сигналов выполняется с помощью многоканального аналого-цифрового преобразователя в составе МК. Структура АЦП и особенности программирования режимов его работы рассмотрены в подразд.2.5.

Разработайте и выполните программу аналого-цифрового преобразования и вывода на ЖКИ цифровых эквивалентов аналоговых сигналов от двух источников.

В соответствии с заданием для вашего «рабочего места» **разработайте** схему подключения указанных источников аналоговых сигналов к МК и **выполните** необходимые соединения: выход потенциометра соедините с заданным входом многоканального АЦП, а выход интегратора – с другим входом многоканального АЦП. Вход интегратора необходимо соедините с выходом генератора прямоугольных импульсов.

Интегратор выполняет функции фильтра низких частот. При невысокой скорости изменения параметров колебаний ГПИ он преобразует выходной сигнал генератора в напряжение. Диапазон изменения напряжения на выходе интегратора определяется частотой и скважностью прямоугольных импульсов. **Определите этот диапазон** с помощью осциллографа.

Диапазон изменения сигнала на выходе потенциометра 0 – 5 В.

Вывод на ЖКИ цифровых эквивалентов аналоговых сигналов от двух источников целесообразно реализовать в виде циклической процедуры, в каждом цикле которой выполняются следующие действия:

- преобразование напряжения с выхода потенциометра в цифровой код; запоминание результата в ячейке внутренней памяти $N_{\text{пот}}$;
- преобразование напряжения с выхода интегратора в цифровой код; запоминание результата в ячейке памяти $N_{\text{инт}}$;
- вывод на экран ЖКИ цифрового кода преобразованного напряжения конкретного источника сигнала.

Перед отображением на экране ЖКИ цифровой эквивалент преобразуемого сигнала необходимо преобразовать в трехразрядное

десятичное представление (сотни, десятки, единицы) в ASCII-кодах. Такое преобразование можно выполнить последовательным делением двоичного кода преобразуемого напряжения на 10, последующим формированием ASCII-кодов разрядов десятичного представления и их записью в соответствующие позиции внешнего видеобуфера ЖКИ.

При разработке программы вывода на ЖКИ цифровых эквивалентов аналоговых сигналов от двух источников используйте программы indic.asm (п.4.2.2) и adc_read (п.4.3), подключив их к разрабатываемой программе директивой include <имя файла>.

Для повышения точности преобразования аналоговых сигналов каждый канал АЦП может настраиваться с учетом диапазона изменения преобразуемого сигнала. Настройка осуществляется путем программирования регистра DAPR.

Для каждого значения сигнала на выходе интегратора выполните два преобразования: первое - для диапазона опорных напряжений (0-5В) и второе – для более узкого диапазона, выбранного в соответствии с диапазоном изменения напряжения на выходе интегратора.

Цифровые коды сигналов считывают с экрана ЖКИ, а для наблюдения динамики изменения сигналов к соответствующему входу порта P6 (канала АЦП) необходимо подключить осциллограф. Сравните результат измерений аналогового сигнала осциллографом с цифровым эквивалентом этого сигнала на выходе АЦП. Результаты представьте в виде таблицы содержащей 5 столбцов (см. пример представления результатов измерения/преобразования в табл.14). Для каждого диапазона изменения аналогового сигнала необходимо измерить не менее 5 значений входного сигнала.

Таблица 14

Диапазон опорных напряжений, δ	Напряжение датчика, измеренное осциллографом	Цифровой эквивалент, измеренного значения	Результат преобразования (цифровой код ЖКИ, зафиксированный на выходе АЦП)	Погрешность измерения Δ
0 – 5 В (DAPR = 0) 20 мВ	4,3 В	215	219	4*20 = 80 мВ
0 – 5 В (DAPR = 0) 20 мВ	1,5 В	74	76	2*20 =40мВ
1,25-3,4375 В (DAPR =B4), 8,5 мВ	1,5 В	176	178	2*8,5=17 мВ

δ – дискрета преобразования: $\delta =$ Диапазон / 256

Δ – погрешность измерения: $\Delta = | N_{осц} - N_{ацп} | \times \delta$

Динамику изменения во времени сигнала аналогового датчика можно наблюдать в поле «Выход» вкладки «Окна управления» оболочки Shell51.

Для этого необходимо в окне «Адрес выхода» вкладки «Окна управления» указать адрес ячейки памяти с кодом преобразованного аналогового сигнала выбранного датчика и модифицировать выполненную программу аналого-цифрового преобразования, вставив в ее циклический участок команду lcall 128h.

Модифицируйте программу аналого-цифрового преобразования сигналов с выхода аналоговых датчиков. Модифицированная программа обеспечивает совместную работу пользовательской программы с программой вкладки «Окна управления» оболочки Shell51. Связь с инструментальной ЭВМ (оболочкой Shell51) осуществляется с помощью подпрограммы монитора, вызываемой командой lcall 128h.

При выполнении задания, вращая ручку потенциометра, изменяйте выходной аналоговый сигнал датчика. Цифровые коды аналогового сигнала с выхода АЦП ($N_{пот}$ или $N_{инт}$), отображаемые на экране дисплея ЖКИ, одновременно с помощью подпрограммы монитора lcall 128h передаются в программный модуль «Окна управления». Для отображения динамики изменения сигнала датчика на экране инструментальной ЭВМ в окне «Выход» вкладки «Окна управления» нажмите кнопку «Пуск» вкладки.

Осциллограммы изменяющихся во времени аналоговых сигналов с выходов обоих датчиков для различных диапазонов опорных напряжений, задаваемых при программировании регистра DAPR, **приведите** в отчете.

5.3. Программа третьего цикла работ: «Изучение таймеров и системы прерываний»

Цель работы:

- приобретение практических навыков программирования таймеров
- изучение принципов программного деления частоты;
- знакомство с организацией и использование многоуровневой системы прерываний.

При выполнении работ данного цикла необходимо разработать структуру информационных связей МК с подключаемыми внешними устройствами (клавиатурой, модулем ЖКИ, осциллографом) и реализовать их, соединив разъемы портов МК с разъемами подключаемых устройств. Вариант задания (табл.13) совпадает с номером рабочего места.

Перед выполнением работ данного цикла познакомьтесь со структурой и основными режимами работы счетчиков/таймеров МК SAB 80C515, а также с принципом организации системы прерываний этого МК (см. п.п.2.6, 2.7).

Разработайте и выполните программу, генерирующую на заданном разряде порта МК импульсный сигнал прямоугольной формы (меандр) с частотой, определяемой номером $N_{клав}$ нажатой клавиши блока клавиатуры

(F = $N_{клав}$ * 100 Гц). Формируемые колебания наблюдайте с помощью осциллографа. **Результаты выполнения** программы представьте в виде:

$N_{клав}$ Номер нажатой клавиши	Расчетное значение частоты	Значение частоты, измеренное осциллографом

При разработке программы используйте программные модули, рассмотренные в п.п.4.2.1 и 2.6.1. Двухбайтные константы перезагрузки таймера, определяющие частоту генерируемого меандра, целесообразно разместить в строке внешней памяти. При нажатии клавиши ее номер используйте в качестве индекса соответствующей константы в строке.

Упрощенный алгоритм программы генерирования меандра управляемой частоты представлен на рис.5.2.

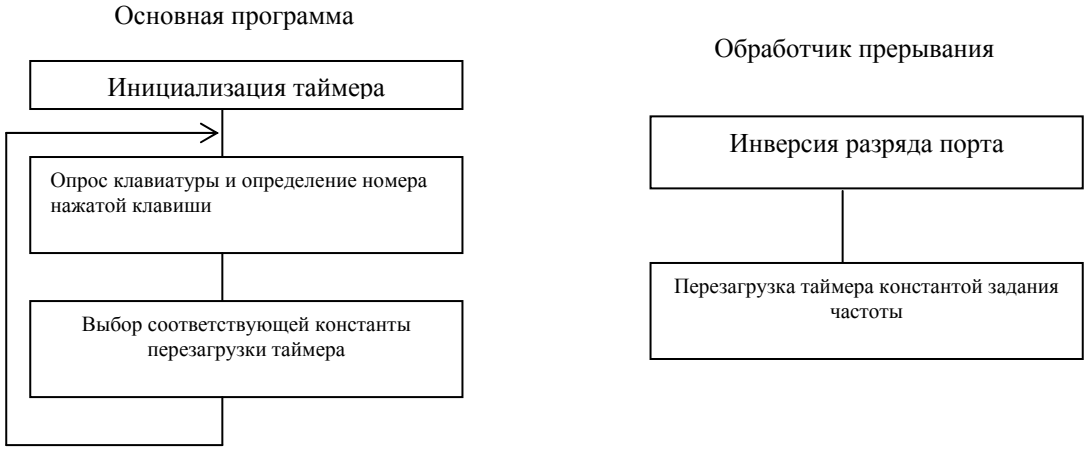


Рис.5.2 Алгоритм программы генерирования меандра требуемой частоты

Разработайте и выполните программу формирования ШИМ-сигнала заданной частоты (F= N*100 Гц, где N - номер Вашего рабочего места). При разработке программ целесообразно использовать программные модули, рассмотренные в п.п. 2.5 и 2.6.3.

Формирование ШИМ-сигнала с фиксированной скважностью рассмотрено в п.2.6.3. Там же приведен алгоритм формирования ШИМ-сигнала с управляемой скважностью. Для управления скважностью используйте цифровой код АЦП, преобразующего аналоговый сигнал с выхода потенциометра. Для запоминания кода управления скважностью используйте ячейку памяти pwm_var. Поскольку на выходе АЦП формируется 8-разрядный код, а для управления скважностью используется 16-разрядный код, содержимое ячейки pwm_var необходимо масштабировать (см. п. 2.6.3). Работу ШИМ-генератора контролируйте с помощью осциллографа.

Результат выполнения программы представьте в виде зависимости измеренных значений τ_{pwm} как функция напряжения на выходе потенциометра $\tau_{pwm} = f(U_{потенц})$. Для измерения указанных параметров используйте осциллограф.

Модифицируйте программу предыдущего задания, используя для управления скважностью ШИМ-сигнала цифровые коды управляющего воздействия, формируемого инструментальной ЭВМ при работе с вкладкой «Окна управления» (Окно «Вход»).

Для связи с инструментальной ЭВМ в циклический участок модифицированной программы вставьте команду lcall 128h – вызова однократного сеанса связи, а в окне «адрес входа» вкладки «Окна управления» установите адрес ячейки памяти pwm_var, в которую передается управляющий код.

Коды управляющего воздействия, формируемые при нажатии кнопки «Пуск» вкладки «Окна управления», запоминаются в приемнике pwm_var. Эти коды используются для управления скважностью ШИМ-сигнала.

График управляющего воздействия, отображаемый в окне «Вход» вкладки, строится пользователем или выбирается из ранее построенных с помощью кнопки «Открыть» вкладки (см. описание вкладки). При выполнении программы используйте оба варианта формирования управляющего воздействия.

Работу ШИМ-генератора контролируйте с помощью осциллографа.

Разработайте программу «Электронные часы» с отображением на экране ЖКИ текущего времени с точностью 0,1 с. В качестве счетных импульсов временных «тиков» используйте прерывания таймера, следующие с частотой 2 кГц. Программный счетчик «тиков» должен быть реализован в фоновой циклической программе. При разработке программы используйте описание электронных часов, рассмотренное в п. 2.6.

Разработайте и выполните программу «Электронный секундомер», которая определяет интервал времени между внешними прерываниями int0 и int1, генерируемыми при нажатии двух клавиш разных столбцов клавиатуры стенда. При разработке программы используйте программный модуль «Электронные часы».

Источником прерываний int0 и int1 является перепад на соответствующих входах микроконтроллера. На стенде входы внешних прерываний int0 и int1 соединены с одноименными выводами. Для подключения прерывания int0 и int1 необходимо соединить два выхода клавиатуры со входами указанных прерываний. На вход клавиатуры следует подать постоянный 0. Этот сигнал должен быть сформирован аппаратно или программно.

Программа иллюстрирует взаимодействие трех обработчиков прерываний (одного внутреннего и двух внешних). При поступлении запроса int0 его обработчик осуществляет запуск таймера, обнуляет счетчик времени, запрещает собственные прерывания int0 и разрешает прерывания int1. Обработчик прерывания int1 должен остановить таймер, блокировать собственные прерывания int1 и разрешить прерывания int0. Значение интервала времени между внешними прерываниями int0 и int1 необходимо отображать на экране ЖКИ.

Модифицируйте программу «Электронные часы», дополнив ее обработчиком прерывания от приемопередатчика последовательного порта, который содержит команду lcall 128h вызова однократного сеанса связи с инструментальной ЭВМ. С помощью этой команды обеспечивается взаимодействие программы оболочки Shell51 и МК во время выполнения программы «Электронные часы».

Исследуйте работу системы (электронных часов) при наличии нескольких источников прерываний.

Цель исследования – анализ влияния приоритетов различных источников прерываний на точность работы часов.

В электронных часах точность работы критична к стабильности средней частоты аппаратных «тиков», формируемых обработчиком прерывания от таймера, и в общем случае зависит от числа источников прерываний в системе, установленных приоритетов их обслуживания и длины самих обработчиков. В системе с одним источником прерываний от таймера ничто не влияет на среднюю частоту аппаратных «тиков», и часы работают правильно. В модифицированной программе присутствует второй (дополнительный) источник прерывания - флаг RI последовательного порта, устанавливаемый при поступлении старт-импульса последовательной посылки, формируемой программой оболочки Shell51. Старт-импульс формируется при нажатии кнопки «Пуск» вкладки «Окна управления» оболочки. При наличии нескольких источников прерываний запросы могут поступать асинхронно, в том числе, когда уже обрабатывается один из запросов. Очевидно, что при неправильном выборе приоритетов обслуживания обработчиков работа программы подсчета времени может нарушаться.

Используя опцию «выход» вкладки «Окна управления», проконтролируйте работу счетчика долей секунд. Содержимое счетчика передается для отображения в окне вкладки в сеансе связи с ПК, вызываемом командой lcall 128h в обработчике последовательного порта.

Отметим некоторые особенности работы модифицированной программы «Электронные часы». Контролируемым параметром в программе является содержимое счетчика долей секунд. Для отображения

скорости работы этого счетчика используется «цифровой осциллограф» - окно «Выход» вкладки «Окна управления» (адрес счетчика долей секунд перед запуском «осциллографа необходимо установить в окне «Адрес выхода» вкладки). Запуск отображения контролируемого параметра осуществляется при нажатии кнопки «Пуск» вкладки. При запуске программа оболочки Shell51 формирует последовательность периодически повторяющихся информационных посылок, которые по последовательному каналу поступают на вход исследуемого МК (стенда). При приеме посылки последовательным портом МК формируется запрос прерывания RI, вызывающий при разрешенных прерываниях обработчик, содержащий команду lcall 128h вызова сеанса связи с ПК. В сеансе содержимое счетчика долей секунд отображается в окне «Выход».

Подпрограмму инициализации электронных часов в модифицированной программе необходимо дополнить командой разрешения прерывания от последовательного порта (setb es) и командами задания приоритетов прерываний (*mov A9h,#const* и *mov B9h,#const*, где #const – код устанавливаемых уровней приоритета задействованных прерываний). Вариант программы обработчика приведен на рис.5.3.

```
        org 8023h     ;Обработчик прерывания последовательного порта
        ljmp usart
usart:  jnb ri,skip   ;проверка наличия запроса RI необходима, поскольку обработчик запускается
                      ; как от RI, так и от TI. При RI=0 осуществляется выход из обработчика
        push a        ;поскольку программа монитора использует некоторые ресурсы МК,
        push 0        ;содержимое регистров a, r0 и psw сохраняется в стеке
        push psw
        lcall 128h    ;вызов однократного сеанса связи с инструментальной ЭВМ
        clr ri
        pop psw
        pop 0
        pop a
skip:   reti
```

Рис.5.3. Текст программы обработчика

С помощью средств среды Shell51 зафиксируйте осциллограммы изменения во времени содержимого счетчика долей секунд для различных комбинаций задаваемых приоритетов источников используемых прерываний.

Определите влияние задаваемых уровней приоритетов прерывания на точность формирования реального времени в данной системе. Осциллограммы **изменения во времени содержимого счетчика долей секунд** приведите в отчете**.**

Разработайте программу управления простейшей многозадачной операционной системой с разделением времени. Система должна выполнять диспетчеризацию трех циклически выполняемых программ:

определение номера нажатой клавиши и преобразование его в ASCII-код (задача 1), отображение на экране ЖКИ номера нажатой клавиши (задача 2) и индикация времени, формируемого программой «Электронные часы» (задача 3).

Пояснения и рекомендации по выполнению задания приведены в прил.3

Формат дескрипторов задач (контекст задачи), включающий адрес точки входа, квант времени выполнения, буфер обмена и т.п., выберите самостоятельно.

Программы задач дополните командой-инвертором, например cpl P1.n, где n – номер задачи, с помощью которой удобно контролировать работу задачи: в процессе работы системы в течение кванта времени, выделяемого для выполнения задачи, на выходе P1.n формируются прямоугольные импульсы с периодом, равным удвоенному времени цикла выполнения задачи. Убедитесь в этом, контролируя сигналы на выходах P1.n с помощью осциллографа. **Определите время выполнения каждой задачи**. **Осциллограммы** работы задач и результаты измерения времени выполнения задач **приведите в отчете**.

5.4. Программа четвертого цикла работ: «Организация и исследование межпроцессорного обмена»

Цель работы:

- ознакомление с принципами организации обменов по последовательному каналу;
- приобретение навыков создания коммуникационных протоколов последовательной связи;
- знакомство с организацией межпроцессорных обменов.

Организация и исследование обменов по последовательному каналу связи изучаются на примерах взаимодействия лабораторного стенда с инструментальной ЭВМ и другим стендом. Последовательные порты SP должны иметь одинаковые параметры коммуникационного протокола. Настройка порта SP компьютера выполняется с помощью вкладки «Терминал» оболочки Shell51. Настройка порта SP МК реализуется программно.

«Поле настроек» вкладки «Терминал» позволяет задавать следующие параметры коммуникационного протокола:

- скорость передачи данных, бит/с. Большая скорость обычно обеспечивает меньшую надежность передачи, поскольку возрастают требования к форме импульсов синхронизации приемника и передатчика;
- длину слова информационной посылки, бит. Этот параметр (обычно 8 бит) используется при передачах символов, представленных различными

кодами, например, при 4-битной кодировке BCD, 5-битной BAUDOT, 6-битной EBCD или 7-битной ASCII;

- количество стоповых бит (1 или 2). Два стоповых бита облегчают синхронизацию, но замедляют скорость обмена;

- контроль передачи на четность/нечетность. При использовании контроля информационная посылка дополняется битом четности/ нечетности.

В качестве примера приведем задаваемый протокол обмена со следующими параметрами приемопередатчика: скорость передачи данных - 1200 бит/с; длина слова передаваемых данных - 8 бит; количество стоповых бит – 2, контроль передачи на четность

Программа работы.

Изучите текст программы send_rec.asm., реализующей прием-передачу данных по последовательному каналу между ПК и МК. Алгоритм и текст программы send_rec.asm представлены на рис.5.4 и 5.5.

Программа send_rec объединяет три подпрограммы: init, receive и send.

Подпрограмма init обеспечивает настройку последовательного порта SP МК и таймера T/C1 на заданный режим работы. Подпрограмма receive реализует прием последовательных данных в МК, а подпрограмма send - передачу последовательных данных из МК.

Выполните программу send_rec.asm, реализующую обмен данными между инструментальной ЭВМ (вкладкой «Терминал» оболочки Shell51) и лабораторным стендом.

Перед выполнением программы с помощью «Поля настроек» вкладки «Терминал» запрограммируйте последовательный порт компьютера для работы по выбранному протоколу обмена.

Для проверки работы программы в окне «Передача кода» вкладки «Терминал» наберите 16-ричный код однобайтной посылки и нажмите кнопку «Послать код». Формируемый при этом старт-бит инициирует работу приемника порта SP микроконтроллера. По завершению приема информационной посылки принятый код переписывается в регистр SBUF приемника и устанавливается флаг прерывания приемника RI. Подпрограмма receive обрабатывает принятую информацию (в данном случае переписывает данные из SBUF в аккумулятор). Поскольку флаг TI установлен, запускается подпрограмма send, реализующая передачу данных из аккумулятора в SP МК. Последний формирует последовательную посылку и пересылает ее в порт SP ЭВМ. При нажатии кнопки «Принять однократно» вкладки принятый по последовательному каналу код отображается на экране ЭВМ в поле «Принятый код».

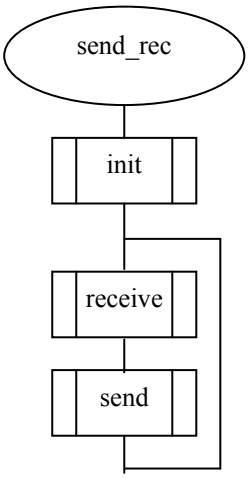


Рис.5.4. Схема алгоритма программы send_rec

```
        ;программа send_rec.asm
           org 8100h

           lcall init
work:      lcall receive
           lcall send
           sjmp work

init:      clr tr1                    ;останов таймера T/C1

      ;задание режима работы SP  ;9-битный асинхронный режим
           mov scon,#11010010b        с переменной скоростью и
                                      ;установленным флагом TI
      ;задание скорости работы SP (программирование таймера T/C1)
           anl tmod,#0Fh              ;выбор режима работы таймера
           orl tmod,#00100000b
; Значения констант перезагрузки T/C1 для произвольных скоростей
; обмена вычисляется по формуле (см.п.2.4.2). Для стандартных
; скоростей обмена ;эти значения выбираются из табл.2 (п.2.4.2),
; при этом также необходимо программировать бит BD в регистре
; ADCON и  бит SMOD в регистре PCON, влияющие на скорость обмена
           anl D8h,#7Fh               ;сброс бита BD в регистре ADCON
           anl 87h,# 7Fh              ;сброс бита SMOD в регистре PCON
           mov th1,#e6h               ;скорость обмена 1200 бит/с
           setb tr1                   ;разрешение работы таймера T/C1
           ret
; подпрограмма receive приема  последовательных данных в МК
receive: jnb ri,receive               ;ожидание завершения приема
           mov a,sbuf
           clr ri
           ret

; подпрограмма receive передачи  последовательных данных из МК
; для ;правильной работы передатчика флаг TI при инициализации
; должен быть установлен.
send:      jnb ti,send                ;ожидание завершения передачи
           mov sbuf,a
           clr ti
           ret
```

Рис.5.5. Текст программы send_rec

Проверьте работу программы при других параметрах коммуникационного протокола обмена.

Модифицируйте программу send_rec, дополнив ее командой cpl a, выполняемой после команды lcall receive (перед командой lcall send). Приведите пример выполнения модифицированной программы.

Разработайте и выполните программу rec_lcd, реализующую вывод на экран ЖКИ текста, передаваемого с инструментальной ЭВМ (вкладки «Терминал» среды Shell51). Вариант схемы соединений МК с инструментальной ЭВМ и блоком ЖКИ показан на рис.5.6.

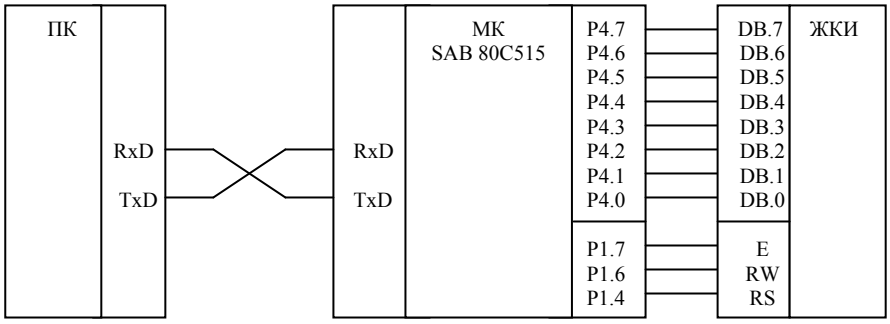


Рис.5.6. Схема информационных связей МК с инструментальной ЭВМ блоком ЖКИ

Передаваемый текст (строка символов) вводится в окно «Передача текстовых сообщений». Строка символов должна заканчиваться символом конца посылки, например ‘.’. При нажатии кнопки «Послать текст» подготовленное сообщение по последовательному каналу должно передаваться в МК и записываться в видеобуфер (внешнюю память МК), при этом необходимо контролировать символ конца посылки и число отображенных символов. По окончанию приема посылки содержимое принятой посылки должно отображаться на экране ЖКИ, начиная с первого знакоместа экрана. Если число символов в посылке ≤ 40, все символы выводятся на экран ЖКИ, при этом незаполненная часть экрана (дополнение до 40) должна отображаться пробелами. При длине принятой посылки более 40 символов из видеобуфера на экран выводятся только первые 40 символов. При завершении вывода на экран ЖКИ управление должно передаваться подпрограмме приема последовательных данных, которая ожидает поступления следующей посылки.

Нарушение протокола обмена (отсутствие символа конца посылки) необходимо контролировать с помощью таймера T/C0. Таймер должен запускаться при поступлении первого импульса посылки. Сброс таймера (его обнуление и запрет счета) должны осуществляться при обнаружении символа конца посылки. При нарушении протокола обмена через заданное время, например, 1с таймер формирует запрос прерывания. Обработчик прерывания должен выводить на экран сообщение типа «Ошибка протокола» и «сбрасывать» таймер. В данной программе таймер T/C0 выполняет функции сторожевого таймера WDT.

Результаты тестирования программы отразите в отчете, в котором также должна быть представлена разработанная структура связей МК с подключаемыми устройствами (инструментальной ЭВМ и модулем ЖКИ).

Разработайте и выполните сервисную программу статистической обработки данных.

Используя поле «Передача текстовых сообщений» вкладки “Терминал” среды Shell51, с инструментальной ЭВМ в МК передается

последовательность чисел произвольного формата, содержащая набор цифровых данных и код инструкции обработки. (Протокол передачи сформируйте самостоятельно). МК принимает эту последовательность и в зависимости от кода инструкции обрабатывает ее. Результат обработки (максимальное, минимальное или среднее арифметическое из чисел) передается в ЭВМ и дублируется на экране ЖКИ.

При разработке программы задания, прежде всего, необходимо определить формат посылки. Например, можно условиться, что первый байт посылки является кодом операции обработки, при этом предварительно необходимо назначить три символа, кодирующих тип обработки. Последующие байты посылки должны быть цифровыми данными. Отдельные разряды чисел передаются ASCII-кодами. Друг от друга числа отделяются пробелами. Поскольку обработка выполняется 8-разрядным микропроцессором, значения чисел и количество чисел ограничим 255. Последовательность чисел завершается символом ‘.’.

Разработайте и выполните программу обмена информацией между соседними микроконтроллерными стендами (2n-1) и 2n, где n =1,2,3,4,…

Соединение двух стендов по последовательному каналу связи осуществляется при нажатии специальной кнопки на задней панели стенда. В нажатом состоянии этой кнопки разрывается связь стенда с инструментальной ЭВМ лабораторного комплекса и реализуется подключение к каналу связи с соседним стендом.

Для иллюстрации межконтроллерного взаимодействия результат выполнения программы на одном из стендов отображается на табло ЖКИ собственного стенда и передается по последовательному каналу для отображения на табло ЖКИ соседнего стенда. На каждом из стендов выполняется циклическая (фоновая) программа определения номера нажатой клавиши. При нажатии клавиши на стенде ее номер отображается в верхней строке табло ЖКИ этого стенда и поступает в буфер передатчика последовательного порта. Передатчик пересылает информацию о номере нажатой клавиши на соседний стенд. Приемник этого стенда принимает посылку, преобразует ее в ASCII-код и отображает номер нажатой клавиши другого стенда в нижней строке табло ЖКИ. Для сокращения объема пересылок между стендами передатчики стендов должны передавать только обновленную информацию.

Перед выполнением задания необходимо разработать структуру информационных связей обоих стендов (МК с подключаемым модулем ЖКИ). Далее выполняется трансляция, загрузка и запуск программы определения номера нажатой клавиши на обоих стендах. После этого при фиксации специальных кнопок-переключателей на задней панели реализуется соединение стендов по последовательному каналу связи.

Модифицируйте программу предыдущего задания для реализации межконтроллерного обмена информацией по последовательному каналу связи в режиме «Master-Slave».

Режим «Master-Slave» используется для организации взаимодействия ведущего МК с ведомыми, которые соединены с ведущим радиальными каналами связи. Для реализации данного режима необходимо задействовать специально предназначенные для этого ресурсы МК, а именно биты SM2, RB8 и TB8 регистра SCON.

Структура информационных связей обоих стендов, выполняемые программы, трансляция, загрузка и запуск программ определения номера нажатой клавиши на обоих стендах соответствуют заданию предыдущей программы.

Программа, иллюстрирующая межконтроллерный обмен, предполагает использование нескольких кнопок, имеющих специальное назначение.

Будем считать, что кнопка 15 предназначена для перевода МК в режим «Master». При ее нажатии МК присваивается статус «Master» с отображением символа «M» на табло ЖКИ, при этом код кнопки передается в канал связи. Ведомые МК (в данном случае МК соседнего стенда) принимают посылку и при декодировании кода 15 переводятся в режим «Slave»: в них передача от «Slave» к «Master» запрещается и в регистре SCON устанавливается бит SM2 (SM2=1). В режиме «Master-Slave» изменяется протокол обмена: ведущий МК сначала передает адресную информацию, после чего разрешается передача данных. Код адресной информации имеет установленный бит TB8 (TB8=1), а код данных – сброшенный бит TB8 (TB8=0). При SM2=1 ведомый МК может принять только адресную информацию, т.е. посылку с установленным битом RB8.

Кнопки 13 и 14 кодируют адреса ведомых МК, при этом код кнопки 14 соответствует адресу соседнего стенда. При нажатии кнопки 14 (или 13) на стенде ведущего МК последний формирует и отправляет адресную посылку с установленным битом TB8, на экране ЖКИ стенда высвечивается информация «addrN» (N=14 или 13). Завершающим этапом посылки адреса является запрет установки бита TB8 в последующих посылках, которые являются посылками данных.

Все ведомые МК, имеющие в исходном состоянии установленный бит SM2, получают адресную посылку, поскольку в ней бит RB8=1, и анализируют ее. Если адрес 14, ведомый МК идентифицирует себя как приемник. Он перепрограммирует свой последовательный порт (сбрасывает бит SM2, подготавливаясь к приему данных) и отображает на экране ЖКИ номер «addr14». Если адрес не равен 14, ведомый МК указанные действия не выполняет.

После отправки адреса последующие посылки, принимаемые ведомым МК, являются посылками данных.

Кнопка 12 кодирует команду «конец посылки данных». При декодировании кода 12 оба МК переводятся в режим межконтроллерного взаимодействия (см.предыдущий пункт программы): в обоих МК сбрасывается бит SM2 и в любом МК разрешены прием/передача данных. В этом режиме при нажатии кнопки 15 любой из МК может быть может быть переведен в режим «Master».

Кнопки 0-11 – это обычные информационные кнопки. При их нажатии на передающей стороне осуществляется процедура определения номера нажатой клавиши и отображения этого номера на экране ЖКИ стенда. Код номера нажатой клавиши передается ведомому. Значение бита TB8 безразлично. На приемной стороне (при SM2=0) данные принимаются и отображаются, поскольку при SM2=0 прием не зависит от значения бита RB8 в посылке.

Выполните программу обмена в режиме «Master-Slave», задавая адрес ведомого МК кнопками 13 и 14. Результаты выполнения отразите в отчете.

5.5. Разработка программ индивидуальных заданий

По заданию преподавателя **разработайте и выполните программу индивидуального задания.** Варианты индивидуальных заданий представлены ниже.

1. Многофункциональный генератор

Формирует в зависимости от комбинации нажатых клавиш 6 видов периодических сигналов: синусоидальный, прямоугольный, треугольный, пилообразный, колоколообразный, трапециидальный; для любого сигнала возможно формирование нескольких (не менее 5) значений периода. Выход генератора – ШИМ-сигнал, моделирующий заданный периодический сигнал.

2. Модель динамического объекта первого порядка:

- инерционно-интегрирующее звено;
- инерционно-дифференцирующее звено.

Требуется программно реализовать разностные уравнения, описывающие работу указанных звеньев. Входные сигналы поступают с инструментальной ЭВМ. Результаты моделирования отображаются на экране ЭВМ. Тип звена и его параметры должны задаваться с клавиатуры стенда.

3. Статистическая обработка сигналов.

Программа должна реализовать накопление цифровых эквивалентов входного аналогового сигнала, формируемых на выходе АЦП, выполнить вычисления, необходимые для построения гистограммы, и передать результаты статистической обработки на инструментальную ЭВМ для их отображения в соответствующем окне оболочки Shell51.

4. Калькулятор.

Программа должна реализовать арифметико-логическую обработку цифровых данных, задаваемых с клавиатуры стенда, отображение на ЖКИ значений входных данных и результатов вычислений.

Формат представления чисел - 16-ричные со знаком с фиксированной точкой. Дополнительные функции: выполняемые операции – сложение, вычитание, умножение, извлечение квадратного корня, взаимное преобразование кодов в 2-, 8-, 10-, 16-ричной системах исчисления, округление результатов, вычисление логических функций.

5. Электронная записная книжка.

Программа должна поддерживать следующие режимы:

- часы-календарь с установкой даты и времени (с возможностью отображения на ЖКИ);
- телефонная база данных формата: «телефон – текстовая строка». Режимы работы базы данных – просмотр, поиск, ввод новых данных, удаление.
- переключение режимов «Время» / «Заметки»;
- формирование специального сигнала (например, мигания текстовой строки) при наступлении указанного пользователем момента времени с выводом на ЖКИ однострочной информационной записи (на русском языке).

6. Электронный экзаменатор.

Программа предлагает пользователю ответить на ряд вопросов с вариантами ответа, отображаемыми на ЖКИ. Проводится подсчет времени каждого ответа, количество правильных и неправильных ответов. Результаты теста отображаются на экране ЖКИ. По окончанию теста возможен просмотр результатов тестирования на экране инструментальной ЭВМ.

7. Игровой тренажер внимания.

Программа реализует следующий алгоритм работы:

По экрану ЖКИ слева направо перемещаются символы, соответствующие определенным клавишам. Задача обучаемого – отреагировать на очередной символ нажатием соответствующей ему клавиши. Правильные и неправильные действия фиксируются в

специальных счетчиках. Запуск и останов программы осуществляются при нажатии выбранных пользователем одноименных клавиш. В процессе выполнения программы темп перемещения предъявляемых символов ускоряется. Статистка выполнения программы должна отображаться на экране ЖКИ после завершения программы.

8. Частотомер

На базе лабораторного стенда реализовать частотомер, обеспечивающий измерение частоты входного периодического сигнала на входе АЦП. Выбор типа входного сигнала: гармонический, прямоугольный, треугольный обеспечивается с помощью клавиатурного блока платы МК

Предусмотреть возможность измерения высокочастотного и низкочастотного сигналов наиболее точными методами.

Режимы работы и результат измерения частоты необходимо отображать на экране модуля ЖКИ. Дополнительно с использованием вкладки «Окна управления» оболочки Shell51 организовать просмотр входного сигнала на инструментальной ЭВМ.

Список литературы

1. В. Ф. Мелехин. Вычислительные системы и сети: Учебник для студ. учреждений высш. проф. образования / В. Ф. Мелехин, Е. Г. Павловский. — М.: Издательский центр «Академия», 2013.

2. В. Ф. Мелехин. Вычислительные машины: Учебник для студ. учреждений высш. проф. образования / В. Ф. Мелехин, Е. Г. Павловский. — М.: Издательский центр «Академия», 2013.

3. Микропроцессорные системы. Учебное пособие для ВУЗов. Под общей редакцией Д.В.Пузанкова. – Политехника, 2002.

4. Боборыкин А.В., Липовецкий Г.П., Литвинский Г.В. и др. Однокристальные микроЭВМ. М.: МИКАП, 1994.

5. Васильев А.Е. Микроконтроллеры. Разработка встраиваемых приложений. Учеб.пособие. СПб.: БХВ-Петербург, 2008.

Приложение 1

Система команд микроконтроллеров семейства MCS-51

Условные обозначения:

Rn – регистры R0 – R7 текущего (рабочего) банка регистров;

bit – прямо адресуемый бит, находящийся в RRAM или регистре специальных функций;

#data или #d – 8-битная константа, входящая в состав команды;

#data16 или #d16 – 16-битная константа, входящая в состав команды;

direct – 8-битный адрес младших 128 байт RRAM или адрес регистра блока SFR;

@Ri – обозначение косвенной адресации через регистр Ri (i = 0, 1);

@DPTR – обозначение косвенной адресации через регистр DPTR;

addr16 – 16-битный адрес безусловной передачи управления;

addr11 – 11-битный адрес безусловной передачи (только в командах передачи управления);

rel – относительный адрес (8-битное значение со знаком, суммируемое с содержимым PC для формирования адреса перехода), в командах передачи управления (условного и короткого безусловного переходов);

/bit – слэш перед операндом указывает, что используется инвертированное значение бита-источника.

Группа команд пересылок

Команды пересылок с мнемоникой MOV

Мнемокод	КОП	Байты	Циклы	Функция
MOV A, Rn	1110 1nnn	1	1	(A) ← (Rn)
MOV A, direct	1110 0101	2	1	(A) ← (direct)
MOV A, @Ri	1110 011i	1	1	(A) ← ((Ri))
MOV A, #data	1110 0100	2	1	(A) ← #data
MOV Rn, A	1111 1nnn	1	1	(Rn) ← (A)
MOV Rn, direct	1010 1nnn	2	2	(Rn) ← (direct)
MOV Rn, #data	0111 1nnn	2	1	(Rn) ← #data
MOV direct, A	1111 0101	2	1	(direct) ← (A)
MOV direct, Rn	1000 1nnn	2	2	(direct) ← (Rn)
MOV direct, direct	1000 0101	3	2	(direct) ← (direct)
MOV direct, @Ri	1000 011i	2	2	(direct) ← ((Ri))
MOV direct, #data	0111 0101	3	2	(direct) ← #data
MOV @Ri, A	1111 011i	1	1	((Ri)) ← (A)
MOV @Ri, direct	1010 011i	2	1	((Ri)) ← (direct)
MOV @Ri, #data	0111 011i	2	1	((Ri)) ← #data

Команда загрузки 16-битных данных

MOV DPTR, #data16	1001 0000	3	2	((DPTR)) ← #data16

Команды пересылок в/из внешней памяти данных

MOVX A, @Ri	1110 001i	1	2	(A) ← ((Ri))
MOVX @Ri, A	1111 001i	1	2	((Ri)) ← (A)
MOVX A, @DPTR	1110 0000	1	2	(A) ← ((DPTR))
MOVX @DPTR, A	1111 0000	1	2	((DPTR)) ← (A)

Команды пересылок из внешней программной памяти

MOVC A, @A+DPTR	1001	0011	1	2	(A) ← ((A)+(DPTR))
MOVC A, @A+PC	1000	0011	1	2	(A) ← ((A)+(PC))

Команды загрузки/извлечения из стека

PUSH direct	1100	0000	2	2	(SP) ← (SP)+1 ((SP)) ← (direct)
POP direct	1101	0000	2	2	(direct) ← ((SP)) (SP) ← (SP)-1

Команды обмена

XCH A, Rn	1100	1nnn	1	1	(A) ↔ (Rn)
XCH A, direct	1100	0101	2	1	(A) ↔ (direct)
XCH A, @Ri	1100	011i	1	1	(A) ↔ ((Ri))

Команды обмена тетрадами

XCHD A, @Ri	1101	011i	1	1	$(A_{3-0}) \leftrightarrow ((Ri)_{3-0})$
SWAP A	1100	0100	1	1	Обмен тетрад аккум-ра

Команды арифметических операций

Мнемокод	КОП		Байты	Циклы	Функция
ADD A, Rn	0010	1nnn	1	1	(A) ← (A) + (Rn)
ADD A, direct	0010	0101	2	1	(A) ← (A) + (direct)
ADD A, @Ri	0010	011i	1	1	(A) ← (A) + ((Ri))
ADD A, #data	0010	0100	2	1	(A) ← (A) + #data
ADDC A, Rn	0011	1nnn	1	1	(A) ← (A) + (C) + (Rn)
ADDC A, direct	0011	0101	2	1	(A) ← (A) + (C) + (direct)
ADDC A, @Ri	0011	011i	1	1	(A) ← (A) + (C) + ((Ri))
ADDC A, #data	0011	0100	2	1	(A) ← (A) + (C) + #data
SUBB A, Rn	1001	1nnn	1	1	(A) ← (A) – (C) – (Rn)
SUBB A, direct	1001	0101	2	1	(A) ← (A) – (C) – (direct)
SUBB A, @Ri	1001	011i	1	1	(A) ← (A) – (C) – ((Ri))
SUBB A, #data	1001	0100	2	1	(A) ← (A) – (C) – #data
DA A	1101	0100	1	1	Десятичная коррекция содержимого аккумулятора
INC A	0000	0100	1	1	(A) ← (A) + 1
INC Rn	0000	1nnn	1	1	(Rn) ← (Rn) + 1
INC direct	0000	0101	2	1	(direct) ← (direct) + 1
INC @Ri	0000	011i	1	1	((Ri)) ← ((Ri)) + 1
INC DPTR	1010	0011	1	2	(DPTR) ← (DPTR) + 1
DEC A	0001	0100	1	1	(A) ← (A) – 1
DEC Rn	0001	1nnn	1	1	(Rn) ← (Rn) – 1
DEC direct	0001	0101	2	1	(direct) ← (direct) – 1
DEC @Ri	0001	011i	1	1	((Ri)) ← ((Ri)) – 1
MUL AB	1010	0100	1	4	$(B)_{15-8}$, $(A)_{7-0}$ ← (A)*(B)
DIV AB	1000	0100	1	4	$(A)_{цел}$, $(B)_{ост}$ ← (A)/(B)

Команды логических операций

Мнемокод	КОП	Байты	Циклы	Функция
ANL A, Rn	0101 1nnn	1	1	(A) ← (A) ∧ (Rn)
ANL A, direct	0101 0101	2	1	(A) ← (A) ∧ (direct)
ANL A, #data	0101 0100	2	1	(A) ← (A) ∧ #data
ANL A, @Ri	0101 011i	1	1	(A) ← (A) ∧ ((Ri))
ANL direct, A	0101 0010	2	1	(direct) ← (direct) ∧ (A)
ANL direct, #data	0101 0011	3	2	(direct) ← (direct) ∧ #data
ORL A, Rn	0100 1nnn	1	1	(A) ← (A) ∨ (Rn)
ORL A, direct	0100 0101	2	1	(A) ← (A) ∨ (direct)
ORL A, @Ri	0100 011i	1	1	(A) ← (A) ∨ ((Ri))
ORL A, #data	0100 0100	2	1	(A) ← (A) ∨ #data
ORL direct, A	0100 0010	2	1	(direct) ← (direct) ∨ (A)
ORL direct, #data	0100 0011	3	2	(direct) ← (direct) ∨ #data
XRL A, Rn	0110 1nnn	1	1	(A) ← (A) ⊕ (Rn)
XRL A, direct	0110 0101	2	1	(A) ← (A) ⊕ (direct)
XRL A, @Ri	0110 011i	1	1	(A) ← (A) ⊕ ((Ri))
XRL A, #data	0110 0100	2	1	(A) ← (A) ⊕ #data
XRL direct, A	0110 0010	2	1	(direct) ← (direct) ⊕ (A)
XRL direct, #data	0110 0011	3	2	(direct) ← (direct) ⊕ #data
CLR A	1110 0100	1	1	Обнуление аккум-ра (A) ← 0
CPL A	1111 0100	1	1	Инверсия аккум-ра (A) ← $(\overline{A})$

Команды сдвиговых операций

RL A	0010	0011	1	1	Сдвиг ACC циклический влево
RLC A	0011	0011	1	1	Сдвиг ACC влево через перенос
RR A	0000	0011	1	1	Сдвиг ACC циклический вправо
RRC A	0001	0011	1	1	Сдвиг ACC вправо через перенос
SWAP A	1100	0100	1	1	Циклический сдвиг ACC на 4 разряда $(A_{3-0}) \leftrightarrow (A_{7-4})$

Команды, влияющие на установку флагов

Команда	Флаг			Команда	Флаг		
	C	OV	AC		C	OV	AC
ADD	+	+	+	ANL C, bit	+	–	–
ADDC	+	+	+	ANL C, /bit	+	–	–
SUBB	+	+	+	ORL C, bit	+	–	–
MUL	0	+	–	ORL C, bit	+	–	–
DIV	0	+	–	MOV C, bit	+	–	–
DA	+	–	–	CLR C	0	–	–
RRC	+	–	–	CPL C	+	–	–
RLC	+	–	–	CJNE	+	–	–
SETB C	1	–	–	POP PSW	+	+	+

Команды битового процессора

Команды пересылки бита

Мнемокод	КОП	Байты	Циклы	Функции
MOV C, bit	1010 0010	2	1	(C) ← bit
MOV bit, C	1001 0010	2	2	(bit) ← (C)

Команды установки и сброса бит

SETB C	1101 0011	1	1	(C) ← 1
SETB bit	1101 0010	2	1	(bit) ← 1
CLR C	1100 0011	1	1	(C) ← 0
CLR bit	1100 0010	2	1	(bit) ← 0

Команды инверсии бит

CPL C	1011 0011	1	1	(C) ← $(\overline{C})$
CPL bit	1011 0010	2	1	(bit) ← $(\overline{bit})$

Команды логических операций для битовых переменных

ANL C, bit	1000	0010	2	2	(C) ← (C) ∧ (bit)
ANL C, /bit	1011	0000	2	2	(C) ← (C) ∧ $(\overline{bit})$
ORL C, bit	0111	0000	2	2	(C) ← (C) ∨ (bit)
ORL C, /bit	1010	0000	2	2	(C) ← (C) ∨ $(\overline{bit})$

Команды условных переходов битового процессора

JB bit, rel	0010	0000	3	2	(PC) ← (PC) + 3 if (bit) = 1 then (PC) ← (PC) + rel
JBC bit, rel	0001	0000	3	2	(PC) ← (PC) + 3 if (bit) = 1 then (bit) ← 0 и (PC) ← (PC) + rel
JNB bit, rel	0011	0000	3	2	(PC) ← (PC) + 3 if (bit) = 0 then (PC) ← (PC) + rel
JC rel	0100	0000	2	2	(PC) ← (PC) + 2 if (C) = 1 then (PC) ← (PC) + rel
JNC rel	0101	0000	2	2	(PC) ← (PC) + 2 if (C) = 0 then (PC) ← (PC) + rel

Команды передачи управления

Команды безусловной передачи управления

Мнемокод	КОП	Байты	Цикл	Функции
LJMP addr16	0000 0010	3	2	(PC) ← $addr_{15-0}$
AJMP addr11	a_{10}-$a_8$0 0001	2	2	(PC) ← (PC) + 2, $(PC)_{10-0}$ ← $addr_{10-0}$
SJMP rel	1000 0000	2	2	(PC) ← (PC) + 2, (PC) ← (PC) + rel
JMP @A + DPTR	0111 0011	1	2	Косвенный переход (PC) ← (A) + (DPTR)
LCALL addr16	0001 0010	3	2	(PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC_{7-0}) (SP) ← (SP) + 1 ((SP)) ← (PC_{15-8}) (PC) ← $addr_{15-0}$
ACALL addr11	a_{10}-$a_8$1 0001	2	2	(PC) ← (PC) + 2 (SP) ← (SP) + 1 ((SP)) ← (PC_{7-0}) (SP) ← (SP) + 1 ((SP)) ← (PC_{15-8}) (PC_{10-0}) ← $addr_{10-0}$
RET	0010 0010	1	2	(PC_{15-8}) ← ((SP)) (SP) ← (SP) − 1 (PC_{7-0}) ← ((SP)) (SP) ← (SP) − 1
RETI	0011 0010	1	2	(PC_{15-8}) ← ((SP)) (SP) ← (SP) − 1 (PC_{7-0}) ← ((SP)) (SP) ← (SP) − 1

Команды условных переходов

Мнемокод	КОП	Байты	Циклы	Функции
JZ rel	0110 0000	2	2	(PC) ← (PC) + 2 if (A) = 0 then (PC) ← (PC) + rel
JNZ rel	0111 0000	2	2	(PC) ← (PC) + 2 if (A) ≠ 0 then (PC) ← (PC) + rel
JC rel	0100 0000	2	2	(PC) ← (PC) + 2 if (C) = 1 then (PC) ← (PC) + rel
JNC rel	0101 0000	2	2	(PC) ← (PC) + 2 if (C) = 0 then (PC) ← (PC) + rel
DJNZ Rn, rel	1101 1nnn	2	2	(PC) ← (PC) + 2, (Rn) ← (Rn) − 1 if (Rn) ≠ 0 then (PC) ← (PC) + rel
DJNZ direct, rel	1101 0101	2	2	(PC) ← (PC) + 2, (direct) ← (direct) −1 if (direct) ≠ 0 then (PC) ← (PC) + rel
CJNE A, direct, rel	1011 0101	3	2	(PC) ← (PC) + 3 if (A) ≠ (direct) then (PC) ←(PC)+ rel при этом if (A) < (direct) then (C) ← 1 else (C) ← 0
CJNE A, #data, rel	1011 0100	3	2	(PC) ← (PC) + 3 if (A) ≠ #data then (PC) ← (PC) + rel при этом if (A) < #data then (C) ← 1 else (C) ← 0
CJNE Rn, #data, rel	1011 1nnn	3	2	(PC) ← (PC) + 3 If (Rn) ≠ #data then (PC) ← (PC) + rel при этом if (Rn) < #data then (C) ← 1 else (C) ← 0
CJNE @Ri, #data, rel	1011 011i	3	2	(PC) ← (PC) + 3 If ((Ri)) ≠ #data then (PC)←(PC) + rel при этом if ((Ri)) < #data then (C) ← 1 else (C) ← 0

Приложение 2

Регистры специальных функций

Таблица

№ п/п	Обозначение Имя регистра SFR	Имена управляющих бит								Адрес Регистра
		7	6	5	4	3	2	1	0	
1*	P0 Порт 0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	80h
2	SP (Указатель стека)									81h
3	DPL (Младший байт указателя данных)									82h
4	DPH (Старший байт указателя данных)									83h
5	PCON (Регистр управления питанием)	SMOD	PDS	IDLS	-	GF1	GF0	PDE	IDLE	87h
6*	TCON (Регистр управления Таймерами T/C0 и T/C1	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88h
7	TMOD (Регистр режимов таймеров)	GATE1	C/T1	M1	M0	GATE0	C/T0	M1	M0	89h
8	TL0 (Младший байт Регистра таймера 0)									8Ah
9	TL1 (Младший байт Регистра таймера 1)									8Bh
10	TH0 (Старший байт регистра таймера 0)									8Ch
11	TH1 (Старший байт регистр таймера 1)									8Dh
12*	P1 Порт 1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	90h
		T2	CLKout	T2EX	INT2	INT6	INT5	INT4	INT3	
						CC3	CC2	CC1	CC0	
13*	SCON (Регистр управления) последовательным портом	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	98h
14	SBUF (Буфер последовательного порта)									99h
15*	P2 Порт 2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	A0h
16*	IEN0 Регистр разрешения Прерываний 0	EAL	WDT	ET2	ES	ET1	EX1	ET0	EX0	A8h
17	IP0 Регистр приоритетов 0	-	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0	A9h
18*	P3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	B0h
	Порт 3	RD	WR	T1	T0	INT1	INT0	TxD	RxD	
19*	IEN1 Регистр разрешения прерываний 1	EXEN2	SWDT	EX6	EX5	EX4	EX3	EX2	EADC	B8h
20	IP1 Регистр приоритетов 1	-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0	B9h
21*	IRCON Регистр управления флагами прерываний	EXF2	TF2	IEX6	IEX5	IEX4	IEX3	IEX2	IADC	C0h
22	CCEN Регистр разрешения захвата/привязки)	CC3h	CC3l	CC2h	CC2l	CC1h	CC1l	CC0h	CC0l	C1h
23	CCL1 (мл. байт) Регистр захвата/сравнения 1									C2h
24	CCH1 (ст. байт) Регистр захвата/сравнения 1									C3h

Регистры специальных функций Продолжение таблицы

№ п/п	Обозначение Имя регистра SFR	Имена управляющих бит								Адрес Регистра
		7	6	5	4	3	2	1	0	
25	CCL2 Регистр захвата/сравнения 2 младший байт									C4h
26	CCH2 Регистр захвата/сравнения 2 старший байт									C5h
27	CCL3 Регистр захвата/сравнения 3 младший байт									C6h
28	CCH3 Регистр захвата/сравнения 3 старший байт									C7h
29*	T2CON Регистр управления таймером 2	T2PS	I3FR	I2FR	T2R1	T2R0	T2CM	T2I1	T2I0	C8h
30	CRCL Регистр загрузки, захвата/сравнения Младший байт									CAh
31	CRCH Регистр загрузки, захвата/сравнения старший байт									CBh
32	TL2 Регистр таймера 2 младший байт									CCh
33	TH2 Регистр таймера 2 старший байт									CDh
34*	PSW Слово состояния программы	C	C'	F0	RS1	RS0	OV	F1	P	D0h
35*	ADCON Регистр управления АЦП	BD	CLK	-	BSY	ADM	MX2	MX1	MX0	D8h
36	ADDAT Регистр данных АЦП									D9h
37	DAPR Регистр программирования источника внутренних опорных напряжений									Dah
38*	ACC Аккумулятор	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	E0h
39*	P4 Порт 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0	E8h
40*	B Регистр B	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	F0h
41*	P5 Порт 5	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0	F8h
42	P6 Порт 6 только в версии ACMOS									DBh

* Регистры, к которым возможно побитовое обращение.

ОПИСАНИЕ СРЕДЫ ПРОЕКТИРОВАНИЯ SHELL51 Приложение 3

Система Shell51. Общие сведения

Одним из направлений применения среды проектирования Shell51 является ее использование для практического ознакомления со структурной организацией МК семейства MCS-51, обучения проектированию прикладного ПО и приобретения опыта построения микропроцессорных систем на основе МК этого семейства.

Система Shell51 является программной составляющей программно-аппаратного комплекса поддержки проектирования прикладного ПО микропроцессорных систем. Комплекс включает инструментальную ЭВМ, лабораторный стенд, реализованный в виде микропроцессорной системы с МК SAB80C535, и собственно систему проектирования ПО Shell51.

Начало работы с системой. Основные требования

Вызов оболочки (вход в среду) Shell51 осуществляется при запуске пакета программ Shell51 (двойным щелчком левой кнопки манипулятора «мышь» на пиктограмме с названием «Система Shell51»), при этом на экране компьютера отображается основное рабочее поле среды (рис. П3.1),

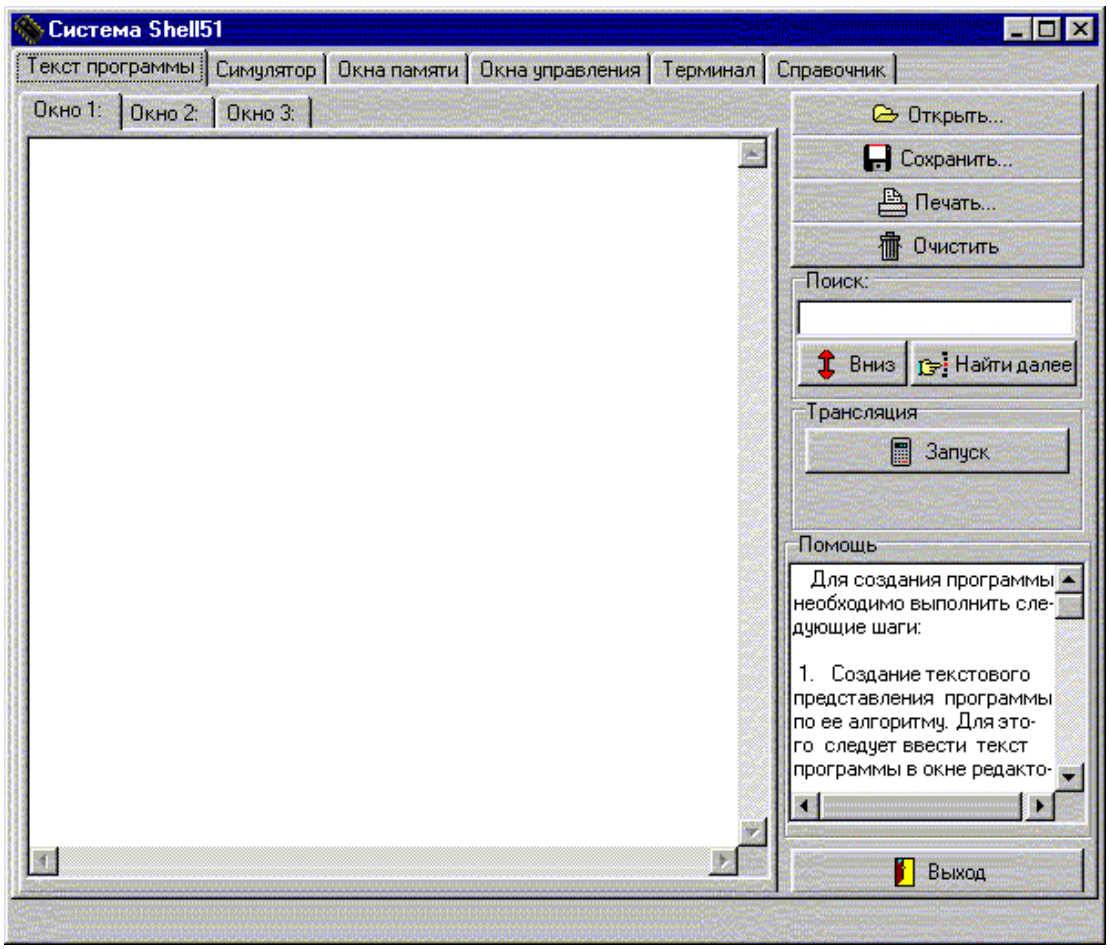


Рис. П3.1. Окно среды Shell51

представляющее собой набор вкладок-опций, каждая из которых ориентирована на реализацию одного из типовых действий программиста при проектировании ПО микроконтроллерных систем. Программы оболочки Shell51 исполняются на инструментальной ЭВМ в среде Windows.

Перед началом работы в среде Shell51 пользователь, используя вкладку «Текст программы», должен создать свой подкаталог в каталоге с номером его группы C:\Shell51\43501/x\Name. Всю работу в среде следует производить только с программными файлами собственного каталога. В этот каталог помещаются файлы с тестовыми программами и разрабатываемыми в процессе работы пользовательскими программами. Модифицировать, копировать, удалять любые файлы в каталоге C:\Shell51 и его подкаталогах, за исключением файлов в собственном каталоге, запрещено.

Лабораторный стенд подключают к сети 220 В с помощью сетевого адаптера при нажатии левой кнопки на задней панели стенда.

По последовательному каналу связи стенд соединяется либо с инструментальной ЭВМ (ПК), либо с соседним стендом. Управление последовательным каналом, обеспечивающим связь стенда с другими устройствами, осуществляется с помощью кнопки (переключателя) - второй слева на задней панели стенда. При ненажатой кнопке МК по последовательному каналу соединяется с ПК. При нажатии кнопки реализуется соединение двух соседних стендов i-го и i+1-го (i – номер рабочего места).

Инициализация подпрограммы связи ПК со стендом осуществляется при нажатии кнопки «Сброс», размещенной на лицевой панели стенда под кнопками клавиатуры.

Для поддержания корректной связи ПК со стендом (МК) по завершению пользовательских программ последние должны завершаться оператором ret, который возвращает управление программе «Монитор» оболочки Shell51.

Вкладка «Справочник»

Вкладка «Справочник» дает общее представление о возможностях системы Shell51 и содержит сведения, необходимые для изучения аппаратуры и программирования микроконтроллерных систем. Вкладка организована в виде гипертекстового документа (рис. П3.2). Выделенные цветом и подчеркиванием элементы документа являются ссылками на соответствующие разделы пособия, переход на которые осуществляется щелчком левой кнопки манипулятора «мышь». Разделы в свою очередь состоят из подразделов с аналогичным принципом переходов. Для возврата на предыдущий уровень используются ссылки «Назад» или «Возврат», присутствующие во всех документах.

Вкладка «Текст программы»

Вкладка «Текст программы» позволяет работать с исходным представлением программы в виде текста на языке ассемблера. Окно вкладки совпадает с окном среды Shell51 (см. рис. П3.1). Вкладка содержит три независимых окна текста (слева), поле справки по использованию вкладки (справа внизу), набор кнопок-панелей, реализующих функции вкладки, а также кнопку-панель «Выход».

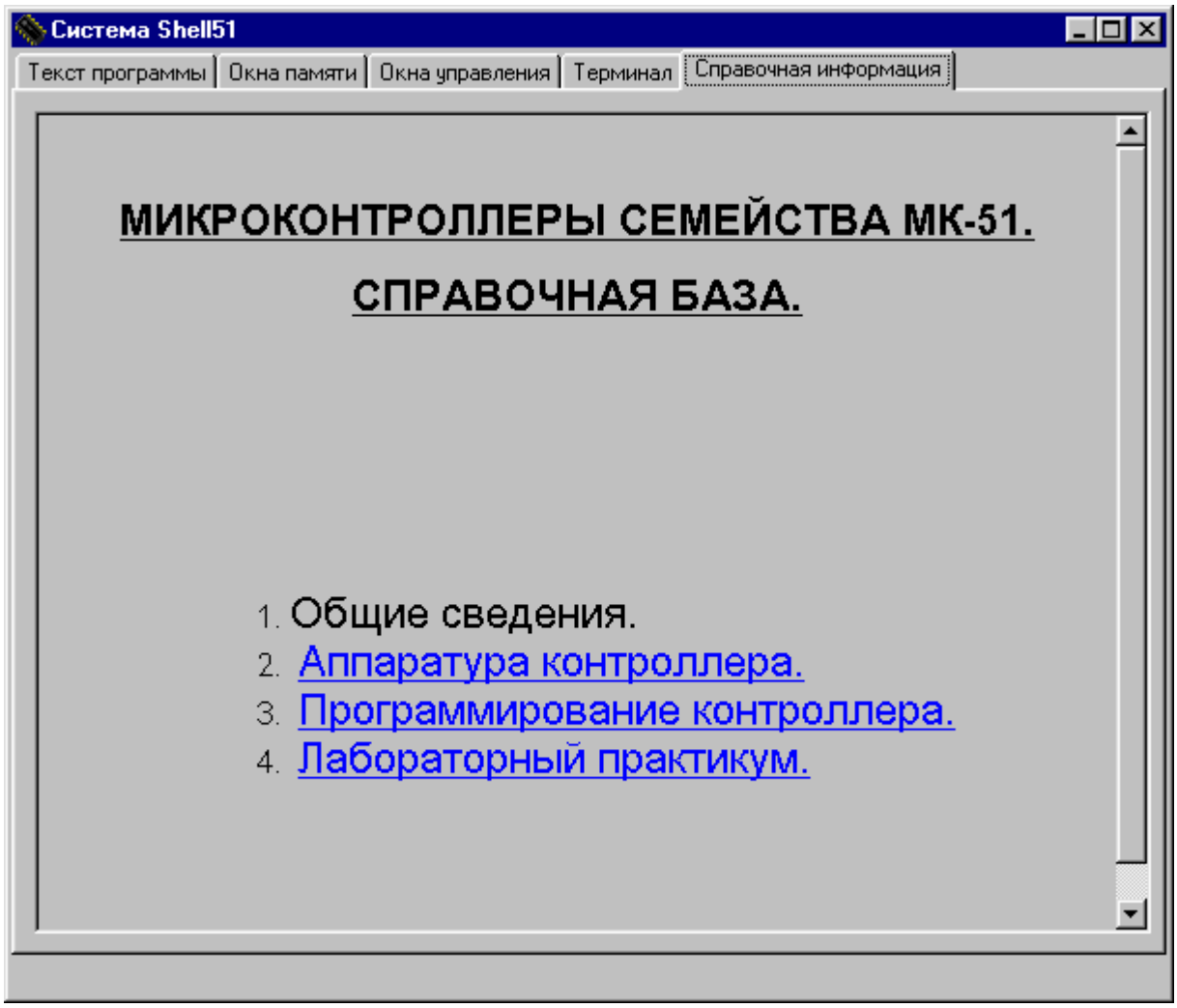


Рис. П3.2. Вкладка «Справочная информация»

Активизация функции, назначенной кнопке-панели, производится путем нажатия левой кнопки манипулятора «мышь» над изображением этой панели.

Кнопка-панель «Открыть» активизирует диалог выбора необходимого пользователю файла с текстом программы, сохраненного на дисковых накопителях инструментальной ЭВМ (рис. П3.3), с возможностью копирования его содержимого в выбранное окно текста. Для открытия требуемого файла следует выделить его (нажатием левой

кнопки манипулятора «мышь»), а затем нажать **кнопку-панель «Открыть»** в правой части диалогового окна (рис. П3.3). В результате текст программы выбранного файла отображается в активном окне и обеспечивается возможность его просмотра и коррекции.

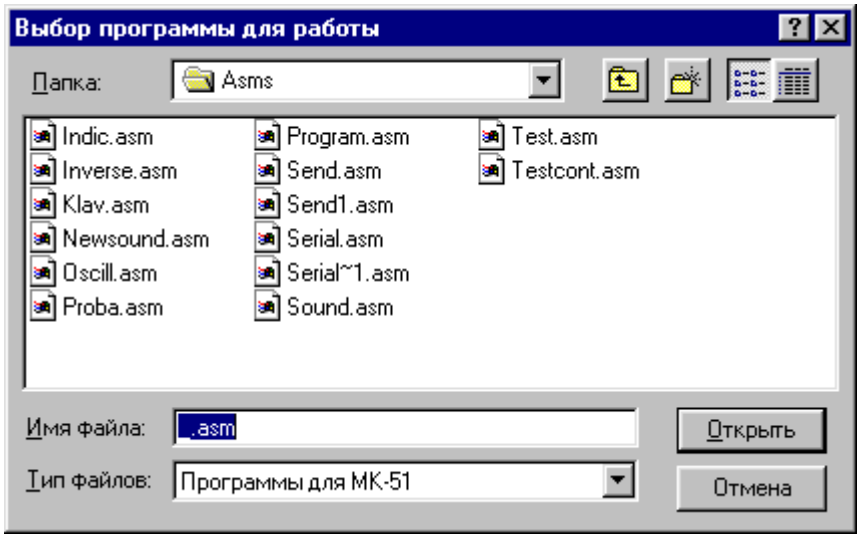


Рис. П3.3. Диалог открытия файла

Для создания новой программы достаточно ввести ее текст в поле текста активного окна. Удаление текста из активного окна осуществляется с помощью **кнопки-панели «Очистить»** вкладки «Текст программы».

Кнопка-панель «Сохранить» вкладки «Текст программы» активизирует диалог записи текста активного окна на диск инструментальной ЭВМ с возможностью модификации имени файла сохраняемой программы (рис. П3.4).

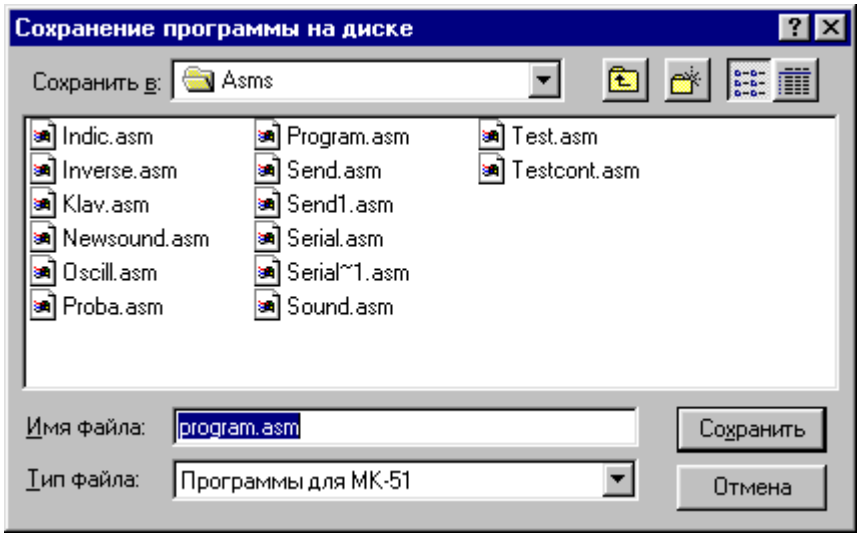


Рис. П3.4. Диалог сохранения файла

При наличии на диске файла с именем, идентичным имени сохраняемого файла, высвечивается окно «Сохранение программы на диске», уточняющее выполняемое действие: с заменой существующего файла или отмены операции записи.

Опция «Поиск» вкладки «Текст программы» используется для облегчения работы с текстами большого объема, в том числе для поиска строк, указанных транслятором как содержащие ошибки.

Текстовую информацию, подлежащую поиску, следует поместить в поле ввода, с помощью кнопки-панели «Вверх» («Вниз») задать направление поиска (от начала документа к концу или наоборот). По нажатию кнопки-панели «Найти далее» в активном окне текста будет выделена строка, содержащая указанный текст. Каждое следующее нажатие этой кнопки-панели продолжает поиск в выбранном направлении (рис. П3. 5). Отсутствие выделенной строки означает отсутствие введенного текста в данной программе.

Кнопка-панель «Печать» вкладки «Текст программы» активизирует диалог назначения параметров печати информации, находящейся в поле активного окна текста программы. При выполнении лабораторных работ вывод текста программы на печатающее устройство не предусмотрен.

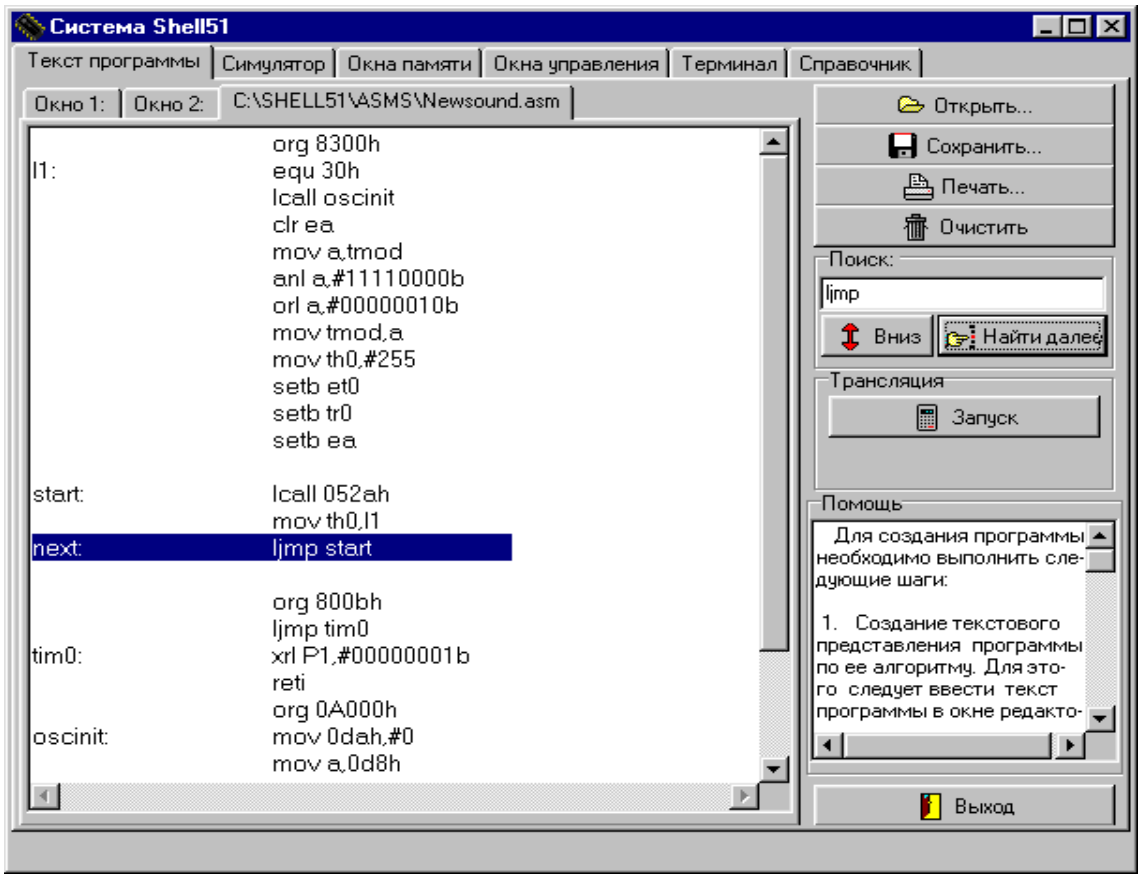


Рис. П3.5. Поиск в тексте программы

Опция «Трансляция» вкладки «Текст программы» обеспечивает трансляцию исходного текста программы в активном окне в представление, пригодное для загрузки в микроконтроллер: ассемблерный текст программы преобразуется в объектный (машинный) код, после чего происходит преобразование объектного кода в шестнадцатеричный исполняемый код.

Процесс трансляции запускается при нажатии **кнопки-панели «Запуск»** в поле «Трансляция». Перед запуском трансляции новая или модифицированная программа должна быть сохранена. Результат трансляции отображается в виде двух диагностических сообщений: «ОК» - при отсутствии ошибок в программе и предупреждения «Внимание!» со стороны транслятора и компоновщика - в противном случае.

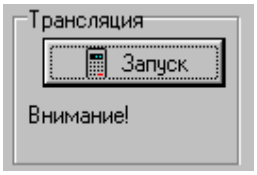


Рис. П3.6. Текст с сообщением о результате трансляции

Детальная информация о результатах трансляции и компоновки расположена на вкладке «Листинги».

Вкладка «Листинги» предназначена для анализа результатов выполнения этапа трансляции и компоновки текста исходной программы. Для вызова опции «Листинги» необходимо щелкнуть манипулятором «мышь» на строке-сообщении о результате трансляции «Внимание» (рис. П3.6). Название опции «Листинги» появляется во вкладке «Текст программы» на месте опции «Симулятор».

При активизации опции «Листинги» щелчком манипулятора «мышь» на имени опции открывается окно-вкладка «Листинг трансляции» (рис. П3.7). Вкладка содержит поле вывода листинга результатов трансляции (слева вверху), поле вывода листинга результатов компоновки (слева внизу), поле перечня ошибок, обнаруживаемых ассемблером (справа вверху), поле перечня ошибок, обнаруживаемых компоновщиком (по центру внизу), поле помощи по использованию данной вкладки (справа внизу), поле ввода строки контекстного поиска, а также три управляющих кнопки-панели.

В полях листинга отображаются результаты прохождения пользовательской программой этапов ассемблирования (преобразования в объектный код) и компоновки (преобразования в шестнадцатеричный исполняемый код). Наличие информации в окнах «Ошибки ассемблирования» или «Ошибки компоновки» свидетельствует о присутствии ошибок в программе пользователя, ошибках объектных библиотек либо присутствии предупреждений со стороны транслятора и/или компоновщика. При наличии сообщений об ошибках необходим анализ текста программы, выявление недопустимых элементов пользовательской программы и их исключение.

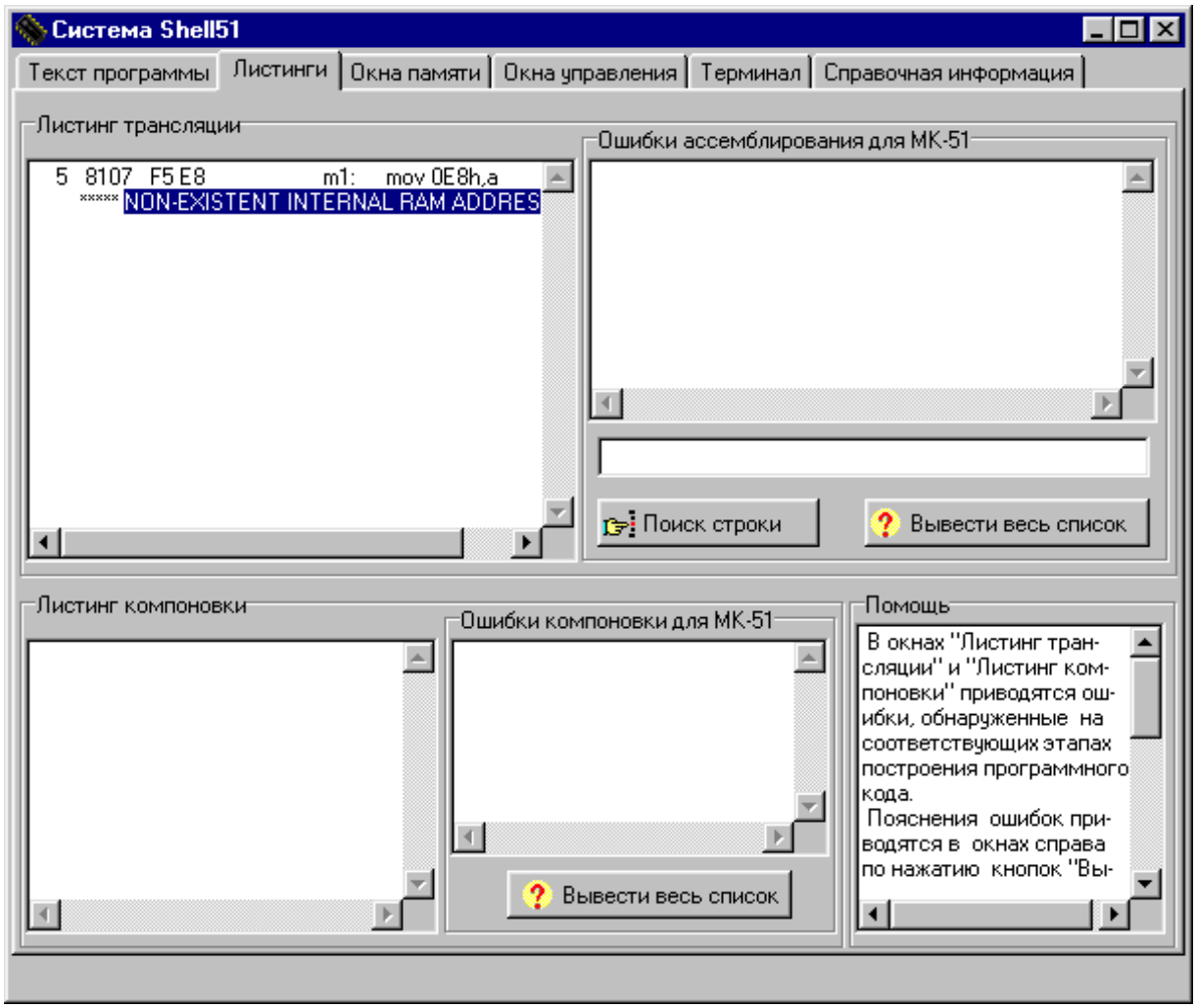


Рис. П3.7. Окно-вкладка листинга трансляции компоновки

С помощью кнопок-панелей «Вывести весь список» система Shell51 позволяет просмотреть полный перечень ошибок этапов ассемблирования и компоновки. Дополнительно имеется возможность выполнить контекстный поиск информации по конкретной ошибке трансляции с языка ассемблера. Для проведения контекстного поиска в поле «Поиск строки» необходимо вставить строку с сообщением об ошибке и активизировать одноименную кнопку-панель «Поиск строки». Сообщение об ошибке может быть введено в поле «Поиск строки» путем набора сообщения с помощью клавиатуры. Альтернативным способом ввода сообщения является его загрузка из буфера обмена (при нажатии комбинации клавиш Ctrl+V), в который сообщение предварительно необходимо скопировать из списка ошибок, указанных в листинге трансляции.

При выполнении указанных операций в поле «Ошибки ассемблирования» появится детальное описание данной ошибки и рекомендации по ее устранению (рис. П3.8). Пустое поле свидетельствует о неточном вводе строки либо отсутствии описания данной ошибки.

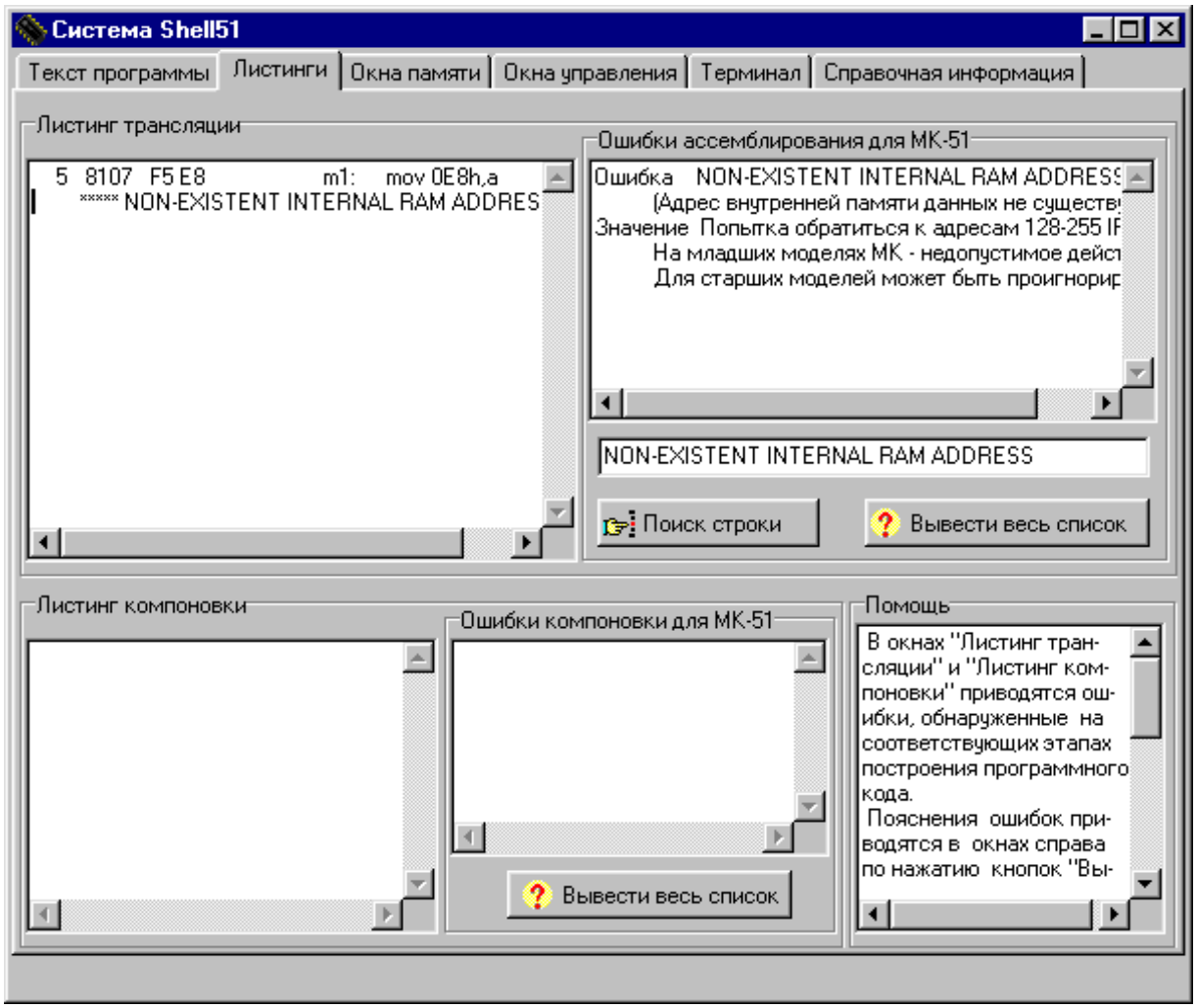


Рис. П3.8. Проведение контекстного поиска

При наличии ошибок результирующий исполняемый файл для микроконтроллера создан не будет.

Неправомочное игнорирование пользователем предупреждений может повлечь за собой работу с исполняемым файлом, не соответствующим исходной программе.

Вкладка «Симулятор»

Симулятор является программным средством отладки программ. Симулятор содержит в своем составе программную модель МК и отладчик. Содержимое внутренней памяти данных МК и состояние периферийных устройств МК с помощью программы-отладчика отображается на экране инструментальной ЭВМ.

Окно-вкладка «Симулятор» (рис. П3.9) открывается при активизации одноименной опции «Симулятор» вкладки «Текст программы».

Вкладка «Симулятор» содержит окно памяти данных программной модели МК (симулятора), окно памяти программ симулятора (справа), окно помощи (слева внизу) и органы управления симулятором (справа внизу).

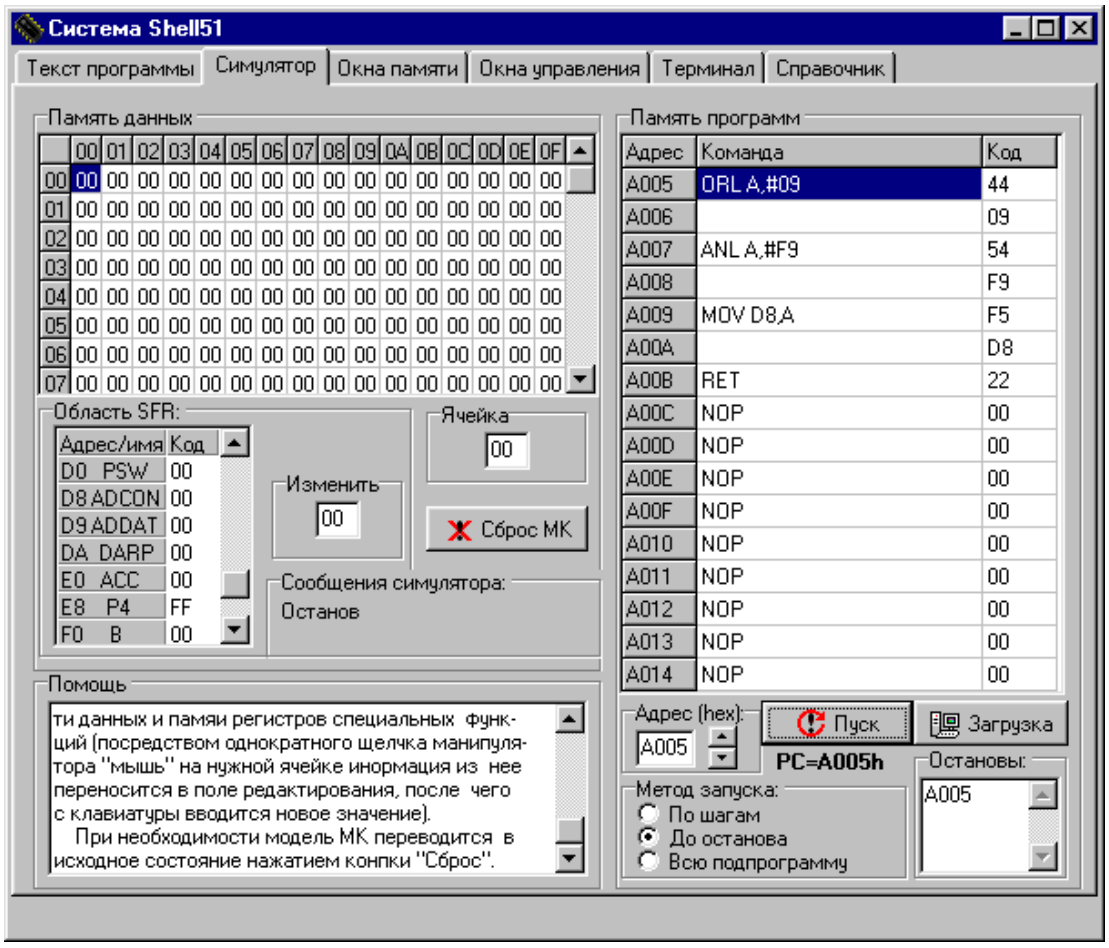


Рис. П3.9. Вкладка "Симулятор".

Симулятор позволяет загрузить оттранслированную программу в память симулятора и отслеживать ее выполнение либо по шагам, либо до установленного пользователем адреса, либо целиком.

Загрузка оттранслированной программы производится при нажатии кнопки-панели «Загрузка». Способ выполнения программы задается с помощью во вкладке «Метод запуска» (рис. П3.10). Выполнение команд программы реализуется при нажатии кнопки «Пуск».

При выборе способа исполнения «По шагам» останов симулятора производится после выполнения каждой команды.

Способ исполнения «До останова» обеспечивает выполнение команд программы до точки останова, соответствующей команде, адрес которой указан в окне «Остановы». Точки останова задаются/удаляются в окне «Остановы» однократным щелчком манипулятора «мышь» на строке команды в памяти программ (рис. П3.11).

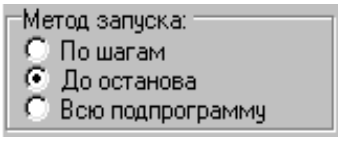


Рис. П3.10. Метод запуска на выполнение

Рис. П3.11. Список Остановов

Способ исполнения «Всю подпрограмму» обеспечивает выполнение всей программы.

Установка счетчика команд (PC) на задаваемый адрес исполняемой команды реализуется двойным щелчком манипулятора «мышь» на строке памяти программ, содержащей необходимую команду. По ходу выполнения программы указатель PC отображает адрес очередной команды из памяти программ симулятора.

Просмотр содержимого памяти программ производится с помощью окна «Адрес». Допускается как непосредственный ввод шестнадцатеричного адреса участка памяти, подлежащего просмотру, так и использование бегунка (рис. П3.12).

Рис. П3.12. Ввод адреса просмотра программы

Память данных симулятора представлена в виде двумерного массива ячеек, адрес которых определяется как $У_{16}*10_{16}+Х_{16}$, где Х и У – шестнадцатеричные значения номеров столбца и строки, которые содержат искомую ячейку. Память регистров специальных функций представлена линейным списком с указанием символических имен регистров SFR.

Пользователь имеет возможность корректировать информацию в памяти данных и памяти регистров SFR. Для модификации данных в указанных областях памяти необходимо выделить нужную ячейку. Эта операция выполняется при подведении курсора к требуемой ячейке и однократном щелчке манипулятором «мышь». При выделении ячейки информация из нее переносится в поле редактирования содержимого ячейки памяти данных симулятора (рис. П3.13) или регистра SFR (рис. П3.14). После этого с помощью клавиатуры в выбранную ячейку можно вводить новое значение.

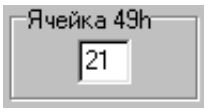


Рис. П3.13. Изменение значения ячейки памяти данных симулятора

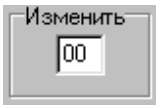


Рис. П3.14. Изменение значения регистра SFR

При нажатии кнопки-панели «Сброс» модель МК переводится в исходное состояние.

Симулятор позволяет быстро проверить логику выполняемой программы, но, к сожалению, он не учитывают в полном объеме реальных характеристик МК и подключаемых к нему внешних устройств.

Комплексная проверка работы микроконтроллерной системы с подключенными внешними устройствами выполняется на лабораторном стенде, подключенном к сети 220 В, с использованием вкладки «Окна памяти» системы Shell51.

Вкладка «Окна памяти»

Вкладка «Окна памяти» (рис. П3.15) предназначена для работы с содержимым памяти программ и данных МК, загрузки в МК пользовательских программ и их запуске на исполнение.

Вкладка содержит окно внутренней памяти данных микроконтроллера (слева вверху), окно текущей ячейки внутренней памяти данных (в центре), окно внешней совмещенной памяти программ-данных (справа), окно справки по использованию данной опции (слева внизу), поле состояния связи и ряд кнопок-панелей.

Внутренняя память данных МК представлена в виде двумерного массива ячеек, адрес которых определяется как $У_{16}*10_{16}+Х_{16}$, где Х и У – шестнадцатеричные значения номеров столбца и строки, которые содержат искомую ячейку.

Любая ячейка внутренней памяти данных допускает модификацию. Для изменения содержимого ячейки памяти ее необходимо выделить

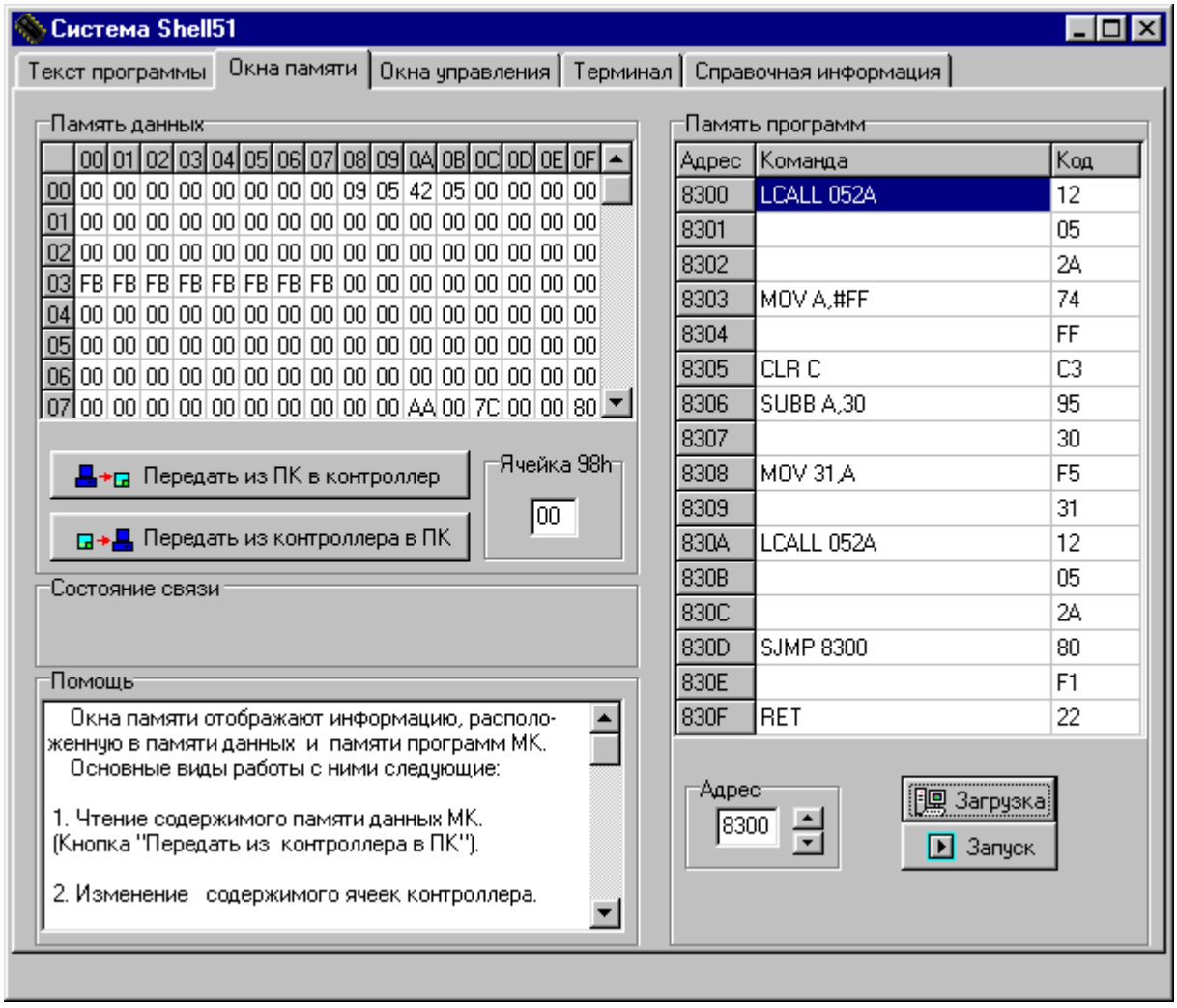


Рис. П3.15. Вкладка "Окна памяти" системы

(выполняется однократным щелчком левой кнопкой манипулятора «мышь» на ячейке), при этом содержимое выделенной ячейки переносится в поле редактирования ячейки памяти данных микроконтроллера (рис. П3.16). Затем с помощью клавиатуры требуемое шестнадцатеричное значение данных вводится в окно редактирования содержимого выбранной ячейки и из него переписывается в окно памяти данных отладчика (среды Shell51). Измененное значение необходимо передать во внутреннюю память данных МК.

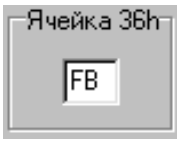


Рис. П3.16.
Ввод значения
выбранной ячейки

Обмены информацией между окном внутренней памяти данных среды Shell51, установленной на ПК, и внутренней памятью данных МК, выполняемые по последовательному каналу, синхронизированы. Обмены выполняются в сеансах связи, инициируемых при нажатии кнопок-панелей «Передать из ПК в контроллер» и «Передать из контроллера в ПК». Для записи новых значений в память данных контроллера используется кнопка-панель «Передать из ПК в контроллер». Чтение (индикация) содержимого внутренней памяти данных контроллера выполняется при нажатии кнопки-панели «Передать из контроллера в ПК».

В ходе обмена информацией поле «Состояние связи» отображает статус линий связи. Информационное сообщение «Обмен с МК» свидетельствует о корректности выполнения очередного сеанса обмена. Сообщение «Нет связи» информирует о невозможности выполнения обмена и блокировке дальнейших сеансов обмена. Это сообщение формируется при отсутствии соединения стенда с ПК (необходимо проверить наличие «питания» стенда и состояние кнопки управления последовательным каналом – она должна быть не нажата). Более распространенной причиной нарушения синхронизации обмена по последовательному каналу являются ошибки в пользовательской программе. Вызов сеанса связи разрешен, если пользовательская программа завершена, т.е. выполнена команда *ret*, передающая управление оболочке Shell51. Если команда *ret* не выполняется (она отсутствует в тексте программы или пользовательская программа содержит некорректные (незавершающиеся) циклы), вызов сеанса связи невозможен. Пользователь должен выявить причину нарушения синхронизации обмена и устранить ее, а затем снять блокировку двойным щелчком манипулятора «мышь» по сообщению.

Перед работой с памятью МК (чтение, загрузка) рекомендуется проверить наличие связи «ПК-МК» путем передачи информации из памяти данных МК в ПК. Эта проверка выполняется при нажатии кнопки-панели «Передать из контроллера в ПК».

Перед выполнением пользовательская программа в исполняемом формате, полученном на этапе трансляции, загружается в память программ контроллера. Эта операция выполняется при нажатии кнопки-панели «Загрузить». При загрузке программы в окне памяти программ, начиная с адреса, заданного директивой ассемблера «org <адрес>» в исходном тексте, отображаются коды первых 16 байт пользовательской программы и их дизассемблированный текст.

Собственно выполнение пользовательской программы активизируется при нажатии кнопки-панели «Запуск». Исполнение программы, осуществляется с адреса, указанного в окне «Адрес». При необходимости значение адреса может быть скорректировано пользователем (рис.П3.17), при этом в окне памяти программ отобразится иной 16-байтный участок памяти программ, начинающийся с заданного адреса.

Рис. П3.17.
Поле ввода адреса
памяти программ

Вкладка «Окна управления»

Вкладка «Окна управления» предназначена для интерактивного исследования систем управления и отладки пользовательских программ. Вкладка содержит: поле формируемого управляющего воздействия на контроллер (слева), поле реакции контроллера на управления (справа), поле справки (слева внизу), и поле настроек подсистемы (справа внизу) (рис.П3.18) .

В окне поля управляющего воздействия, называемом «Вход», пользователь имеет возможность сформировать или выбрать тип управляющего воздействия в виде дискретной функции $\Phi(n)$, где n - номер выборки функции Φ.

Выбор ранее созданной и сохраненной на диске ПК функции управляющего воздействия Φ осуществляется при нажатии кнопки-панели «Открыть».

График управляющего воздействия $\Phi(n)$ строится с помощью манипулятора «мышь» в виде кусочно-линейной функции. Двойной щелчок означает начало/окончание режима ввода, однократный - фиксацию текущего отрезка графика. Для удобства построения графика $\Phi(n)$ в поле «Позиция» отображаются текущие координаты манипулятора. График $\Phi(n)$ можно запомнить на диске ПК, нажав кнопку-панель «Сохранить».

В окне «Выход» отображается динамика изменения контролируемого параметра пользовательской программы в процессе ее выполнения. По выполняемой функции окно «Выход» фактически является «цифровым

осциллографом». При запуске «осциллографа» на экране ПК отображается график функции $M(n)$, являющейся результатом функционирования пользовательской программы на контроллере. Формируемый в окне «Выход» график $M(n)$ можно запомнить на диске ПК, нажав кнопку-панель «Сохранить».

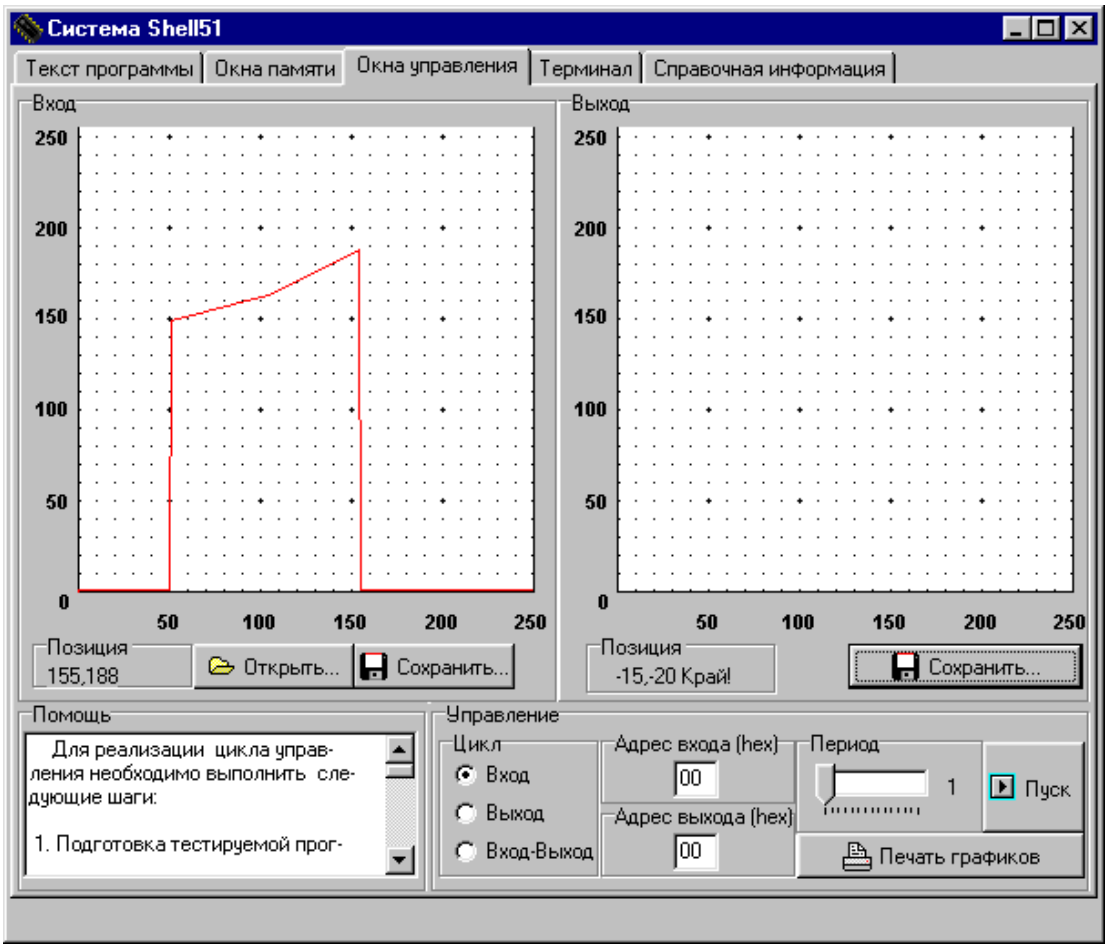


Рис. П 3.18. Вид окна вкладки «Окна управления»

Функции $\Phi(n)$ и $M(n)$ представляют собой развернутые во времени значения задаваемых пользователем ячеек внутренней памяти данных, в связи с чем обе функции нужно адресовать, указав необходимые значения в полях «Адрес входа и «Адрес выхода».

Длительность развертывания наблюдаемых процессов в условных временных единицах задается изменением положения движка «Период» посредством манипулятора «мышь».

Начало сеанса интерактивного управления/регистрации инициируется при нажатии кнопки-панели «Пуск».

Возможны три варианта использования вкладки «Окна управления»:

- формирование управляющего воздействия $\Phi(n)$ на МК в окне «Вход»;

- отображение (регистрация) состояния контроллера - функция $M(n)$ в окне «Выход»;
- совместное применение функций $\Phi(n)$ и $M(n)$ при формировании управляющего воздействия $\Phi(n)$ на МК в окне «Вход» с получением резу результирующего отклика в окне «Выход».

Выбор необходимого варианта осуществляется установкой переключателя «Цикл» в требуемое положение.

Пользовательская программа, работающая совместно с программами вкладки «Окна управления», должна содержать команду вызова подпрограммы однократного сеанса связи МК с ПК *lcall 128h*. В сеансе связи компьютер выдает очередное значение управляющей функции $\Phi(n)$ и/или принимает значение отображаемого параметра с выхода МК $M(n)$. Поскольку программное управление обычно организуется в виде цикла, команду *lcall 128h* следует поместить внутрь цикла.

Другим вариантом организации взаимодействия оболочки Shell51 (компьютера) с МК (стендом) является взаимодействие по прерываниям от приемопередатчика последовательного порта. При данном способе организации взаимодействия ПК с МК команда *lcall 128h* помещается в обработчик прерывания последовательного порта МК.

Вкладка «Терминал»

Вкладка «Терминал» предназначена для исследования последовательного интерфейса связи МК с ЭВМ без участия связных компонент программы-монитора, и построения пользовательских связных компонент.

Окно вкладки (рис. П3.19) содержит: поле настроек приемопередатчика последовательного порта инструментальной ЭВМ (слева вверху), окно справки (слева внизу), поле окна передачи (справа вверху), поле окна приема (справа внизу) и поле состояния связи (справа в центре).

Основой правильной работы последовательного интерфейса является синхронность работы приемника и передатчика, достигаемая заданием одинаковых значений скорости передачи и длины слова передаваемых бит для приемопередатчиков ПК и МК. Требуемый режим работы приемопередатчика ПК задается с помощью переключателей поля настроек приемопередатчика (рис. П3.19). Аналогичный режим приемопередатчика МК программируется пользователем.

При исследовании системы связи компьютера с МК пользователю доступны следующие варианты организации работы подсистемы:

1. Посылка информации в порт:
   - если посылка однобайтная, используется окно передачи шестнадцатеричных кодов и панель-кнопка «Послать код», активизирующая выдачу значения в последовательный порт на МК;

- если посылка многобайтная, целесообразно использовать окно передачи текста, предварительно введенного с клавиатуры ЭВМ, и панель-кнопку «Послать текст», при нажатии которой производится выдача кодов символов, составляющих текст.

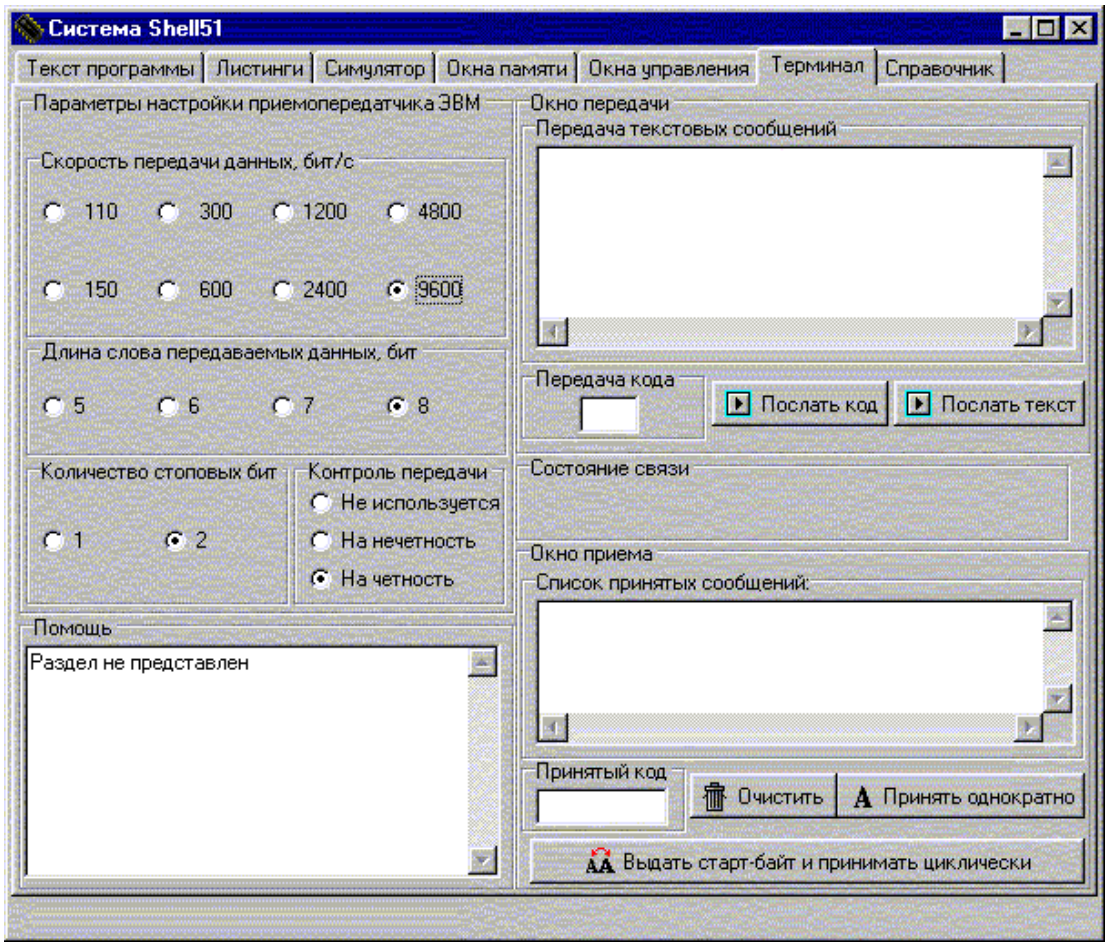


Рис. П3.19. Вид окна вкладки «Терминал»

2. Прием информации из порта:

- при активизации кнопки-панели «Принять однократно» код, содержащийся в буфере приемника ЭВМ, отображается в окне «Принятый код», а также очередным эквивалентным символом в окне «Принятый текст». Кнопка-панель «Очистить» позволяет освободить окна приема от содержащейся в них информации;
- при активизации кнопки-панели «Выдать старт-байт и принимать циклически» система посылает в последовательный порт код $A5_{16}$, а затем переходит к циклу ожидания поступления входной информации из последовательного порта. Старт-байт может быть использован пользователем как уведомление о необходимости передавать результаты в ЭВМ.

3. Комбинация вышеуказанных вариантов.

При наличии ошибок в логике взаимодействия пользователя и программы МК возможно нарушение обмена, обозначаемое сообщением «Нет связи» в поле статуса линии. Необходимо выявить и устранить причину ошибки, после чего разблокировать приемопередатчик путем двойного щелчка манипулятором «мышь» на указанном сообщении.

(см. комментарии на с. 175)

Приложение 4

ЧАСТО ИСПОЛЬЗУЕМЫЕ ДИРЕКТИВЫ АССЕМБЛЕРА X8051

Введение в программирование на ассемблере x8051. В настоящее время при проектировании микропроцессорных систем широко применяются программные среды, в которых программирование выполняется на языках высокого уровня. Такой подход позволяет значительно упростить и ускорить процедуру создания проекта, поскольку он не требует знаний тонкостей архитектуры и структуры используемых технических средств. Некоторым недостатком способа программирования на языках высокого уровня является то, что такой способ не всегда обеспечивает оптимальное использование аппаратных ресурсов процессора и не позволяет минимизировать объем программного кода и некоторые другие параметры системы. При повышенных требованиях к быстродействию, массогабаритным характеристикам, пониженному энергопотреблению применяют машинно-зависимые языки- ассемблеры.

Принципиальное отличие языка ассемблера от других языков программирования состоит в том, что язык ассемблера оперирует командами, которые непосредственно распознает и выполняет процессор. По исходному тексту программы, написанной на ассемблере, всегда можно определить время исполнения программы и требуемый объем программной памяти. Ассемблер позволяет программисту наилучшим образом использовать аппаратные ресурсы процессора, обеспечивая доступ к любому из встроенных периферийных устройств наиболее быстрым и рациональным способом. Именно поэтому, несмотря на некоторую специфику использования ассемблера, связанную с необходимостью хорошего знания аппаратной части процессора, ассемблер является основным языком программирования встроенных микропроцессорных систем управления.

Ассемблеры разрабатываются для конкретного процессора или серии процессоров, имеющих единую базовую систему команд. Главное достоинство языка ассемблера заключается в том, что он допускает представление всех элементов программы в символической (буквенно-

цифровой) форме, отражающих их содержательный смысл. Преобразование символических наименований в двоичные коды машинного языка возлагается на специальную программу, называемую ассемблирующей программой или транслятором с ассемблера. Процесс трансляции символических имен в машинный объектный код называется ассемблированием или компиляцией.

Любая программа на ассемблере, кроме машинных команд, исполняемых процессором, содержит неисполняемые команды - директивы ассемблера.

Машинные команды - это команды из системы команд микроконтроллера. При трансляции исходного текста программы мнемоники команд процессора в конечном итоге преобразуются в исполняемый объектный код.

Директивы ассемблера - это специальные команды транслятору о необходимости выполнения определенных действий в процессе ассемблирования. Фактически они являются командами управления процессом ассемблирования. Директивы ассемблера не преобразуются в двоичные коды, а потому не могут быть выполнены процессором. Они являются псевдокомандами и применяются для задания структуры программы, определения переменных, резервирования памяти и ряда других целей.

Директивы можно разделить на несколько групп:

- директивы определения программного модуля и организации межмодульных связей: *MODULE*, *PUBLIC*, *EXTERN*, *GLOBAL*, *END*;
- директивы управления размещением данных в отдельных сегментах: *USING, SEG, DEFSEG*:
- директивы определения символических имен переменных и задания атрибутов переменных (значения данных, их типа): *EQU, SET*;
- директивы резервирования места переменных в памяти: *DB, DW, DS*;
- директивы условного ассемблирования: *IF, ENDIF*;
- специальные операторы для использования в макросах : *IFB, IFNB, IFEQ, IFNE*.

Для удобства представления и чтения программы, написанные на ассемблере, представляют в виде последовательности строк (операторов) стандартного формата: каждый оператор, т.е. любая машинная команда или директива, записываются в одной строке, имеющей четыре поля:

Метка Код Операнд Комментарий

Поля ассемблерной строки имеют следующее назначение:

Поле «Метка» может содержать необязательное символическое имя. Метки ассоциируются с адресами памяти. Метки, как и имена операндов, нельзя дублировать. Метка имеет длину от одного до шести символов,

первым из которых должна быть буква. За последним символом метки должен указываться специальный разделитель - двоеточие. Метка может предшествовать машинным командам, макровызовам, директивам резервирования памяти, пустым операторам. Часто метки используются в качестве адресов команд передачи управления. Они освобождают программиста от необходимости оперировать абсолютными адресами. Для некоторых директив ассемблера, например, директив MACRO, EQU, SET имя, указываемое в поле «Метка» является обязательным атрибутом. В директивах, в которых используется символическое имя, двоеточие после имени не ставится.

Поле мнемоники (поле «Код») является единственным обязательным полем команды. Оно содержит символическое описание машинной команды, директивы ассемблера или имя пользовательской макрокоманды.

Большинство мнемоник представляют собой аббревиатуры предложений, характеризующих основные функции машинных команд и директив. Если содержимое этого поля не входит в множество допустимых мнемоник, ассемблер выдает сообщение о недействительной команде.

Поле «Операнд» в большинстве случаев содержит имена операндов, участвующих в операции. Содержимое и синтаксис этого поля зависят от мнемокода команды, указываемой в поле «Код». В качестве операндов могут фигурировать адреса памяти, внутренние регистры МК, адреса портов ввода-вывода, числовые и символьные константы. В большинстве команд символическое имя операнда-приемника в поле «Операнд» указывается первым и отделяется от символического имени операнда-источника запятой. В некоторых машинных командах и директивах поле «Операнд» не используется.

Константы бывают двух типов: численные и адресные. Численные константы в командах обозначаются как непосредственные операнды *#data*, а адресные не имеют специальных обозначений. Символы могут быть заключены в апострофы (‘’): *mov a,#‘C’*. Константы могут быть представлены явно (конкретными значениями) и неявно (выражениями, вычисляемыми компилятором). При явном задании констант с помощью спецификаторов указывается система счисления. Спецификаторы могут указываться как перед константой, так и после нее. Обозначение спецификатора зависит от его типа и системы счисления:

Основание	Начальный спецификатор	Конечный спецификатор
2	%	B
8	@	O or Q
16	$	H

Примеры: mov a,#254; mov a,#%11111110; mov a,# 11111110b;
mov a,#@200; mov a,#376Q; mov a,$FE; mov a,#FEh;
mov dptr,# A5F2h

Поле комментария используется для обозначения действий, которые выполняет команда в контексте конкретной прикладной программы. Поле комментария должно начинаться с некоторого разделителя (в большинстве случаев с точки запятой). Комментарий не ассемблируется, поэтому в нем можно помещать любой текст. В отношении содержания комментария желательно руководствоваться следующими рекомендациями. Во-первых, в комментарии акцентируется не общая функция команды, которая определяется мнемоникой, а действие данной команды в контексте данной программы. Во-вторых, не следует сокращать объем комментария в ущерб смыслу. Четкие и содержательные комментарии облегчают понимание и использование программы.

Ассемблер допускает обращение к переменным по символическим адресам, а к данным (константам) по предварительно определенным символическим именам. Поддержка символических адресов исключает необходимость работы с абсолютными адресами, при этом возможность задания значимых имен адресам переменных и константам делает программу более понятной.

Имена, определяемые пользователем, являются идентификаторами и могут содержать до 31 символов. Символические имена должны начинаться с алфавитного символа (или специального символа ? или _) и могут включать любую комбинацию букв, цифр, знака вопроса и подчеркиваний. Регистр символов внутри идентификатора ассемблером игнорируется (допускается произвольное смешивание строчных и прописных букв; все они воспринимаются как прописные). В качестве символических имен не допускается использование зарезервированных слов. Зарезервированные слова - это имена, которые имеют специфическое значение а ассемблере и недоступны для определения пользователем. Примером зарезервированных слов являются мнемоники машинных команд, директивы ассемблера, макродирективы, идентификатор STACK. Особое значение в ассемблере имеет символ $, называемый счетчиком размещения. С помощью символа $ отмечается текущая ячейка внутри активного сегмента программы. Счетчик размещения обычно используется для поддержки режимов относительной адресации операндов.

Для справки приведем описание часто употребляемых директив ассемблера x8051

Директивы управления адресами и файлами программ:

Команды размещаются в программном сегменте по последовательным адресам. Адрес команды определяется содержимым счетчика команд PC. При компиляции значение счетчика PC в соответствии с программным

кодом автоматически модифицируется. Для задания начального значения счетчика PC (начального адреса программы) применяется директива *ORG VALUE*, имеющая следующий формат:

Метка	Код	Операнд
[Label:]	ORG	<адресное выражение>

До новой директивы ORG команды и данные размещаются в смежных ячейках памяти. Если в самом начале программы директива ORG отсутствует, то по умолчанию подразумевается наличие директивы ORG с нулевым операндом.

Использование меток для обозначения адресов команд позволяет реализовать ветвления по командам условного (JC, JNC, JZ, JB, JNB, JBC, DJNZ) и безусловного (JMP) переходов или команды CALL.

Команды *JMP* и *CALL* конкретизируются компилятором в соответствующие команды *LJMP, AJMP, SJMP, LCALL, SCALL, JMP@+dptr* с учетом адресного интервала при переходе.

В качестве метки команды можно использовать символ $, который, как отмечено выше, отображает текущее значение счетчика команд PC. Например, команда

```
      djnz   R0,$    эквивалентна команде перехода с меткой
m1:   djnz   R0, m1
```

После сегмента CODE в ассемблерном тексте рекомендуется размещать сегмент DATE, который предназначен для хранения переменных и результатов вычислений. В сегменте DATE можно использовать только директивы резервирования переменных DB, DW, DS.

Директива END VALUE отмечает конец программы или подключенного файла. Выражение, следующее за END, дополнительное и, если существует, указывает стартовый адрес программы.

Директива INCLUDE <имя файла> позволяет подключать в тело программы фрагменты кода файла, задаваемого в поле «имя файла». Директива INCLUDE указывает ассемблеру на необходимость трансляции текста, расположенного в подключаемом файле, начиная с адреса, следующего за адресом последней команды, расположенной в текущем файле.

При использовании директивы *include <имя файла>* часто фиксируются ошибки, связанные с наложением переменных и участков программ (при любых вызовах по общим адресам формируемое обращение будет производиться к участку кода последнего оттранслированного модуля). Поэтому при использовании директивы *include <имя файла>* необходимо выполнять следующие требования:

- имя файла, указываемого в директиве *include <имя файла>* не должно превышать 8 символов;

- следует избегать абсолютных адресов в подключаемых файлах;
- нельзя использовать одинаковые метки в выполняемой программе и подключаемых файлах;
- при указании имени подключаемого файла необходимо указывать полный адрес этого файла: asm\4081_3\…\имя_файла.asm.

Директивы определения символических имен

Для определения символических имен в ассемблере предназначены *директивы EQU и SET*, которые имеют следующий формат:

Метка	Код	Операнд	Комментарий
<имя>	EQU	<выражение>	
<имя>	SET	<выражение>	

Директива EQU (EQUAL - приравнять, присвоить) присваивает абсолютное значение, псевдоимя или текстовое символьное имя имени, указанному в поле «Метка». Константу, определенную директивой EQU, нельзя переопределять. При ассемблировании вместо имени, определенного этой директивой, подставляется присвоенное ему значение.

Директива SET (установить) имеет такой же формат и выполняет те же функции, что и директива EQU. Однако в отличие от директивы EQU, значение символического имени в директиве SET можно изменять с помощью новой директивы SET. Имя директивы SET можно рассматривать как переменную, значение которой зависит от этапа компиляции и может быть использовано в условной компиляции.

Директивы резервирования памяти

Директивы DB, DW, DS резервируют память под элементы памяти соответствующего типа и имеют следующий формат

Метка	Код	Операнд	Комментарий
[Label :]	DB	<список>	;формат директивы DB
[Label :]	DW	<список>	;формат директивы DW
[Label :]	DS	<выражение>	;формат директивы DS

Операнд директивы DB может быть последовательностью 8-битных значений, разделяемых запятыми, либо цепочкой символов, заключенных в апострофы. Примеры использования директивы DB:

Метка	Код	Операнд	Комментарий
ARRAY:	DB	3, 7, 15, 31	;запоминаются четыре значения

```
        DB      'HELLO'   ;запоминаются пять символов
COMPL:  DB      - 63      ;запоминается дополнительный код числа  - 63.
```

Директивы DB и *DW* не только резервируют память, но и осуществляют инициализацию данных. Операндом директивы DW является последовательность 16-битных значений.

Примеры использования директивы DW:

```
Метка   Код     Операнд     Комментарий
ADDR:   DW      0FF00h      ;(ADDR) = 00h, (ADDR + 1) = FFh
DATA:   DW      100h, 200h  ;инициализируются четыре ячейки памяти
```

Директива DS (STRING – строка) используется только для резервирования памяти. Операндом директивы DS является выражение, определяющее число ячеек (байт) памяти, резервируемых для размещения данных. Инициализация ячеек памяти не выполняется. Примеры использования директивы DS

```
Метка       Код     Операнд         Комментарий
ARRAY:      DS      32          ;резервируются  32 байта в памяти
TABLE:      DS      64          ;резервируются  64 байта в памяти
```

Директива BIT связывает адрес бита с именем булевой переменной. Она имеет следующий формат:

```
Метка       Код     Операнд
[Name]      BIT     <address bit>
Например:
  X1        BIT       P3.2
   S        BIT       ACC.0
```

Приложение 5
Пояснения и рекомендации по выполнению некоторых заданий

П5.1. Многозадачная операционная система

Многозадачность - это способ одновременного выполнения нескольких задач однопроцессорной системой. Поскольку в однопроцессорной системе различные задачи могут выполняться физическим процессором только поочередно, суть многозадачности заключается в организации работы процессора при выполнении нескольких задач (взаимодействующих между собой или независимых).

Один из способов реализации многозадачности, называемый *разделением времени*, заключается в предоставлении каждой задаче некоторого интервала времени (кванта обслуживания), в течение которого процессор выполняет команды соответствующей программы. Задачи обслуживаются последовательно.

Квант времени выполнения задач обычно задается таймером. Если в течение выделенного программе кванта времени ее обработка не заканчивается, программа прерывается, и она становится в очередь программ, ожидающих обработку. Чтобы имитировать одновременное, а не поочередное выполнение процессов, квант времени выполнения задач не должен быть слишком большим.

С целью исключения взаимного влияния задач (процессов) друг на друга информация о состоянии некоторых ресурсов процессора на момент переключения сохраняется. Эта информация называется контекстом или дескриптором процесса. В состав контекста обязательно входит адрес возврата (адрес команды, с которой будет возобновлено выполнение программы задачи при следующем вызове), а также может входить содержимое регистров общего назначения и регистров специальных функций (для МК SAB 80C515 - это регистр *B*, аккумулятор *A*, слово состояния *psw*, указатель *dptr*, содержимое стека задачи). Поскольку контекст задачи в общем случае имеет достаточно большой объем, контексты задач целесообразно сохранять во внешней памяти МК. В последующем изложении определение «контекст задачи» будем использовать для обозначения области внутренней памяти МК, выделяемой для размещения информации о состоянии некоторых ресурсов процессора при выполнении текущей задачи. Определением «дескриптор задачи» будем обозначать структуру данных с контекстом задачи во внешней памяти.

В многозадачной системе переключение задач (прерывание текущей и вызов следующей) осуществляет диспетчер системы – программный модуль, реализующий необходимые действия при переключении. Поскольку разрешенное прерывание может прервать любую программу в

любой момент времени, диспетчер системы может быть реализован в виде обработчика прерывания по переполнению таймера. На обработчик прерывания возлагаются следующие функции:

- задание кванта времени выполнения задачи (реализуется путем перезагрузки счетчика таймера T/C0 расчетной константой). При равных паритетах задач значение константы для всех задач одинаковое, например, соответствует 5 мс. В системе с разными паритетами задач каждой из них может быть задан собственный квант времени выполнения. Константы перезагрузки в этом случае выбираются из контекста вызываемой задачи, который должен опрашиваться до перезагрузки таймера новым значением;
- определение адреса дескриптора текущей задачи (адрес дескриптора текущей задачи определяется по номеру выполняемой задачи);
- сохранение контекста прерываемой задачи. Запоминание контекста текущей задачи (заполнение дескриптора) является существенной функцией диспетчера. Запоминание упрощается, если контекст задачи предварительно разместить в непрерывной области внутренней памяти. В этом случае для запоминания контекста можно использовать циклическую процедуру, обеспечивающую заполнение дескриптора соответствующими командами пересылки во внешнюю память;
- выбор очередной задачи на выполнение (определение номера вызываемой задачи и адреса ее дескриптора);
- восстановление контекста вызываемой задачи;
- передача управления новой задаче по завершении программы обработчика. Передача управления осуществляется автоматически посредством замены содержимого счетчика PC адресом возврата, выбираемым из восстановленного стека вызываемой задачи (ее контекста) при выполнении последней команды обработчика *reti*.

Формат дескриптора задачи определяется объемом контекста, который, в свою очередь, зависит от числа локальных переменных задачи и их размещении в памяти МК. Запоминание контекста задачи и его восстановление упрощается, если он занимает непрерывную область внутренней памяти МК. Учитывая минимально необходимый объем контекста (он рассмотрен выше) будем считать, что объем контекста переключаемых задач не превышает 32 байт и для размещения контекста задачи используется начальная область внутренней памяти МК с адресами 00h – 1Fh. Область внутренней памяти с адресами большими 20h является общим ресурсом для всех задач. Она не перезаписывается при смене задач и может содержать данные, являющиеся для одних задач входными параметрами и выходными для других.

В МК семейства MCS51 указатель стека SP при включении питания инициализируется значением 07h и стек растет вверх (в область старших адресов). Нетрудно заметить, что содержимое регистров R0 – R7 и стека, если указатель SP не инициализировался в программе выполняемой

задачи, на момент переключения задач уже являются частью контекста, при этом в вершине находится адрес команды, которая должна выполняться при следующем вызове прерываемой задачи. Этот адрес является адресом возврата $PC_H : PC_L$ в эту задачу. Формируя контекст содержимое регистров SFR, входящих в контекст (*dptr*, *psw*, *a*, *b*), поместим в стек, используя для этого команды *push direct*. В стек также поместим содержимое регистров R0 и R1. Эти регистры будут использоваться самим диспетчером (обработчиком прерывания) при выполнении процедур сохранения/восстановления контекста в дескрипторе задачи. После выполнения команд *push dph, push dpl, push psw, push b, push a, push 0, push 1* контекст задачи принимает вид (рис.П5.1).

Имя	R0	R1	R2	R3	R4	R5	R6	R7	…	…	PC_L	PC_H	dph	dpl	psw	b	a	R0	R1	…	…
Адрес	00	01	02	03	04	05	06	07	Стек задачи				Продолжение области контекста								1Fh

Рис.П5.1 Контекст задачи

Нетрудно заметить, что при выбранной структуре контекста его объем определяется значением указателя SP, который адресует вершину (последнюю ячейку) стека. При такой структуре контекста собственно стек задачи не должен иметь глубину более (32-17)=13 байт. Здесь 17 – минимально необходимое число параметров контекста задачи.

Сохранение контекста во внешней памяти и его возврат во внутреннюю память реализуются с помощью циклических процедур, которые могут быть реализованы в виде следующих фрагментов программы (рис.П5.2).

```
Циклическая процедура
сохранения контекста
            . . . . . .
            mov r0,sp
            inc r0
            mov dptr, # d_tskn
            mov r1,#0
m1:         mov a,@r1
            movx @dptr,a
            inc r1
            inc dptr
            djnz r0,m1
```

```
Циклическая процедура
восстановления контекста
            . . . . . .
            . . . . . .
            . . . . . .
            mov dptr, # d_tskn
            mov r1,#0
m2:         movx a,@dptr,
            mov @r1,a
            inc r1
            inc dptr
            djnz r0,m2
```

Рис.П5.2. Фрагменты программ сохранения/восстановления

Для задания параметров циклов сохранения/восстановления контекста используются регистры R0 и R1. Регистр R0 содержит число сохраняемых

(извлекаемых) параметров контекста. Это число равняется значению указателя стека SP плюс 1. В процедуре сохранения контекста регистр R0 инициализируется вычисленным значением числа параметров. В процедуре восстановления контекста в регистр R0 заносится число извлекаемых параметров контекста из дескриптора вызываемой программы. Регистр R1 используется в качестве указателя адреса внутренней памяти МК с контекстом задачи.

В процедуре сохранения контекста содержимое контекста переписывается в дескриптор задачи *d_tskn*, размещаемый во внешней памяти по адресу *#d_tskn*. По завершению процедуры сохранения значение R0, равное числу сохраненных параметров sp+1, запоминается в первом байте дескриптора.

В циклической процедуре восстановления контекста вызываемой задачи содержимое дескриптора прерывающей (вызываемой) задачи переписывается в область контекста процессора, при этом регистр R0 модифицируется автоматически значением числа извлекаемых параметров из дескриптора вызываемой задачи. Поскольку объем контекста задачи определяется значением указателя SP, значение числа извлекаемых параметров R0 можно использовать для модификации указателя стека SP вызываемой задачи. Эту операцию можно выполнить командами *dec R0* и *mov SP,R0*.

Содержимое регистров SFR вызываемой задачи восстанавливается из стека задачи командами *pop 1, pop 0, pop a, pop b, pop psw, pop dpl, pop dph*. После выполнения указанных команд адрес возврата в вызываемую задачу оказывается в вершине стека. Передача управления этой задаче осуществляется по завершению программы обработчика командой *reti*.

Адрес дескриптора задачи определяется номером прерываемой задачи. Этот номер хранится в переменной *num_tsk*. Преобразование номера задачи в ее адрес и адрес дескриптора реализуется в зависимости от способа задания этих адресов. Адреса дескрипторов задач *d_tskn* (*d_tsk0*, *d_tsk1*, *d_tsk2*) и адреса самих задач (*tsk0*, *tsk* и *tsk0*) могут быть заданы оператором *org addr* или определены в процессе компиляции программы (см. ниже). Для хранения дескрипторов во внешней памяти должны быть зарезервированы области данных, размером равным длине дескриптора. По завершению процедуры сохранения значение R0, равное числу сохраненных параметров sp+1, запоминается в первом байте дескриптора.

Выбор очередной задачи на исполнение

Порядок выполнения задач в системе определяется последовательностью 0, 1, 2, 0, 1, 2 и т.д. В соответствии с этой последовательностью должен определяться и номер *num_tsk* (идентификатор) задачи, которая будет выполняться следующей. Алгоритм определения номера следующей задачи для указанной дисциплины обслуживания достаточно очевиден. Если номер выполняемой задачи

(значение num_tsk) меньше 2, то значение num_tsk увеличивается на 1, в противном случае номеру присваивается значение 0 (*num_tsk*:=0). На рис.П5.4 показан один из возможных вариантов реализаций этого алгоритма. Номер (идентификатор) задачи используется для определения адреса ее дескриптора *d_tskn*.

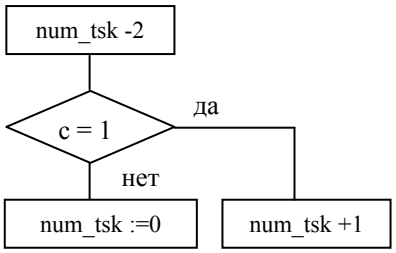


Рис.П5.4. Алгоритм определения номера следующей задачи

Дескрипторы имеют размер кратный 2^n (как правило, не меньше 16 байт и не больше 64 байт) и обычно занимают смежные области памяти. Определение адреса дескриптора по номеру задачи рассмотрим на примере. Пусть для дескрипторов всех задач зарезервирована область памяти, задаваемая оператором *org E000h*, и размер дескриптора равен 32 байт. Адреса дескрипторов: для первой задачи *d_tsk0 = E000h*, для второй задачи *d_tsk1 = E020h*, для третьей задачи *d_tsk2 = E040h*. Нетрудно заметить, что адреса дескрипторов задач отличаются только значением младшего байта. Для определения этого значения достаточно сдвинуть содержимое ячейки *num_tsk* с номером задачи на 3 разряда вправо (или 5 разрядов влево). После определения адреса дескриптора задачи содержимое дескриптора можно модифицировать значениями сохраняемого контекста для прерываемой задачи или использовать для восстановления контекста вызываемой задачи.

При инициализации программы, имитирующей работу многозадачной операционной системы, дескрипторы задач *tskn_d* (рис.П5.1) должны содержать контекст, обеспечивающий правильную работу программы при ее запуске. Обязательной информацией в исходном контексте задачи является длина дескриптора и адрес возврата. В рассматриваемом примере длина дескриптора определяется числом сохраненных параметров и равна 11h (рис.П5.1). Это число необходимо указать в нулевой ячейке дескриптора. Поскольку при первом вызове задачи адресом возврата является адрес самой задачи, то этот адрес должен быть указан в девятой (PC_L) и десятой (PC_H) ячейках дескриптора. В первой ячейке дескриптора должно быть задано значение 1, обеспечивающее корректное выполнение процедуры восстановления контекста. Остальные параметры контекста задачи безразличны, поскольку они инициализируются соответствующей подпрограммой *init* самой задачи.

Пример реализации определения дескрипторов операторами org:

```
        org 9000h
task1:   sjmp task1

        org 9200h
task2:  sjmp task2

        org 9400h
task3:  sjmp task3
        org E000h


tsk1_d:  db 11h,1,0,0,0,0,0,0,00,90h, 0,0,0,0,0,0,0,0

        org E020h
tsk2_d:  db 11h,1,0,0,0,0,0,0,00h,92h, 0,0,0,0,0,0,0,0

        org E040h
tsk2_d:  db 11h,1,0,0,0,0,0,0, 00h,94h, 0,0,0,0,0,0,0,0
```

Пример инициализации дескрипторов, адреса которых определяются в процессе компиляции:

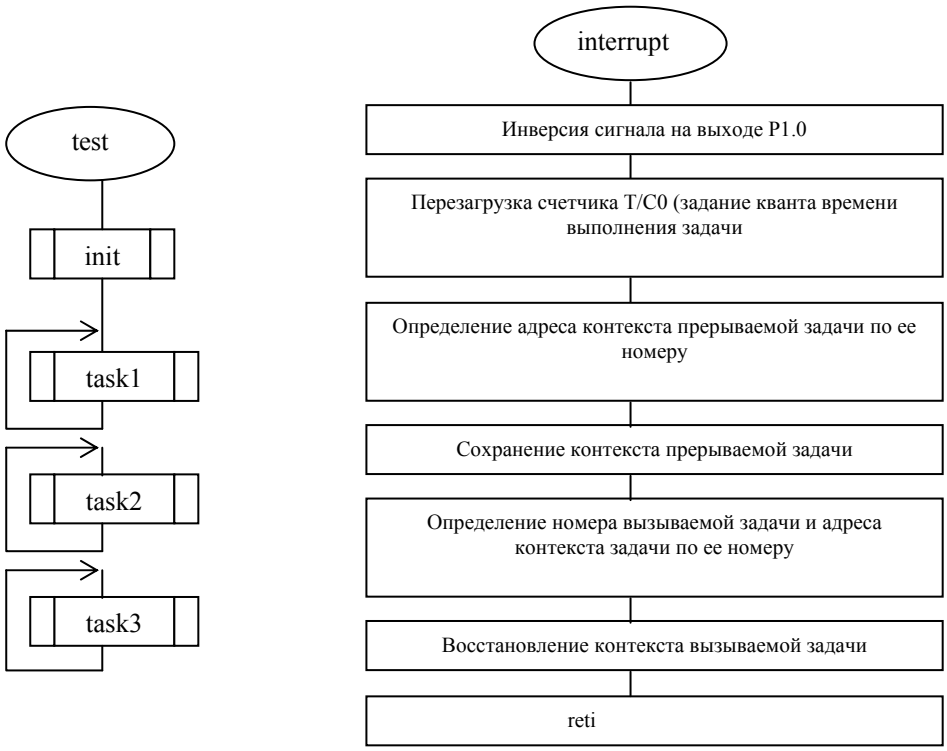
```
    ;инициализации дескриптора d_tsk2 второй задачи

        mov dptr,#task2          ; загрузка адреса второй задачи
        mov r0,dpl               ;  запоминание адреса второй задачи в регистрах r0 (младший байт адреса) и
        mov r1,dph               ; r1 (старший байт адреса)
        mov dptr,#tsk2_d; загрузка адреса дескриптора второй задачи
        mov a,dpl                ; формирование адреса ячейки дескриптора,
        add a,#8                 ; в которую заносится
        mov dpl,a                ; адрес второй задачи (адрес возврата)
        mov a,r0          ; загрузка адреса
        movx @dptr,a             ; второй задачи
        inc dptr                 ; в дескриптор
        mov a,r1
        movx @dptr,a

task1:  sjmp task1

task2:  sjmp task2

task3:  sjmp task3

tsk1_d:  db 11h,1,0,0,0,0,0,0, 00,00, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

tsk2_d:  db 11h,1,0,0,0,0,0,0, 00,00, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

tsk2_d:  db 11h,1,0,0,0,0,0,0, 00,00, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

    ; инициализации дескриптора d_tsk3 третьей  задачи выполняется аналогично
```

Из представленных пояснений видно, что сохранение и восстановление контекста является сравнительно сложной процедурой. Для контроля правильности ее выполнения целесообразно предельно упростить переключаемые задачи, сосредоточив основное внимание на обработчике

прерывания таймера, реализующем функции диспетчера при переключении задач. Разработайте и выполните тестовую программу, управляющую циклическим выполнением 3-х простейших задач, каждая из которых представлена только одной циклически выполняемой командой *cpl P1.n,* где *n* – номер разряда порта P1, совпадающий с номером задачи.

Тестовая программа позволяет проконтролировать последовательность переключения выполняемых задач и правильность сохранения/ восстановления контекстов задач при их переключении. Алгоритм тестовой программы, иллюстрирующий работу многозадачной ОС и алгоритм обработчика прерывания по переполнению T/C0 (программа диспетчера) показаны на рис.П5.3.

В подпрограмме *init* таймер T/C0 настраивается для циклического формирования кванта времени выполнения отдельной задачи, например, 2-5 мс (см. подразд. 2.6.1). Кроме того, в подпрограмме *init* выполняется инициализация дескрипторов второй и третьей задач. Примеры инициализации дескрипторов рассмотрены выше.



Алгоритм программы обработчика прерывания по переполнению T/C0 (алгоритм диспетчера)

Рис.П5.3. Алгоритм тестовой программы, иллюстрирующей работу многозадачной ОС

Для контроля выполнения переключаемых программ в исходные коды программ необходимо добавить команды инвертирования соответствующих разрядов порта P0.

Работу многозадачной системы контролируйте с помощью осциллографа. На выходе P1.0 можно наблюдать меандр, длительность полупериода которого соответствует формируемому кванту времени, предоставляемому для выполнения задач. На выходах P1.n в течение кванта времени, выделяемого задаче, должны формироваться меандры с периодом, равным удвоенному времени цикла выполнения задачи.

После верификации тестовой программы разработка основной программы задания п.7 не должна представлять труда. Задачи (программы реальных процессов) подключаются к ядру ОС операторами include.

П5.2. Статистическая обработка данных

Используя поле «Передача текстовых сообщений» вкладки “Терминал” среды Shell51, с инструментальной ЭВМ в МК передается последовательность чисел произвольного формата, содержащая набор цифровых данных и код инструкции обработки. (Протокол передачи сформируйте самостоятельно). МК принимает эту последовательность и в зависимости от кода инструкции обрабатывает ее. Результат обработки (максимальное, минимальное или среднее арифметическое из чисел) передается в ЭВМ и дублируется на экране ЖКИ.

При разработке программы задания, прежде всего, необходимо определить формат посылки. Например, можно условиться, что первый байт посылки является кодом операции обработки, при этом предварительно необходимо назначить три символа, кодирующих тип обработки. Последующие байты посылки должны быть цифровыми данными. Отдельные разряды чисел передаются ASCII-кодами. Друг от друга числа отделяются пробелами. Поскольку обработка выполняется 8-разрядным микропроцессором, значения чисел и количество чисел ограничим 255. Последовательность чисел завершается символом ‘.’.

Выберем следующий формат посылки: сначала посылается код операции обработки («+» - поиск максимального, «-» - поиск минимального, «=» поиск среднего арифметического). После кода операции следует последовательность данных, представленная ASCII-кодами десятичных разрядов передаваемых чисел. Формат чисел в последовательности – произвольный, Значение отдельного числа ≤ 255.Числа отделяются друг от друга пробелами. Конец последовательности отмечает символ ’.’.

Пример последовательности:

-5 126 15 2 48 247.

П5.3. Игровой тренажер внимания

Тренажеры внимания широко применяют для тестирования скорости реакции испытуемых: пилотов, водителей, военных, представителей других профессий, для которых важна реакция на факт наступления определенного события. Не менее существенным является использование тренажеров внимания при обучении «слепой печати».

Одним из вариантов реализации алгоритма работы тренажера может быть следующий:

Символы перемещаются слева направо в верхней (нижней) строке модуля ЖКИ. Скорость перемещения символов может изменяться в процессе выполнения тестирования: при высоком проценте «правильной» реакции (≥70 %) скорость предъявления символов возрастает, при низком проценте «правильной» реакции (≤ 40 %) скорость снижается. Скорость предъявления символов должна отображаться в нижней (верхней) строке и может изменяться дискретно с шагом 0,1 (или 0,2) сек. Например, нормальное значение скорости – 1 символ в 1 сек, максимальная скорость – 1 символ в 0,1 сек. Минимальная скорость не регламентируется. Для простоты реализации ее можно ограничить величиной 1 символ в 2 сек..

Вариантом усложненной реализации является тестирование в условиях возникновения помех. Помехи представляют собой символы, отличные от контролируемых. Они появляются в строке с информационным символом в случайные моменты времени и в случайных позициях экрана со скоростью, более высокой по сравнению со скоростью перемещения символа.

Контролируемые символы, как и помехи, генерируются генератором случайных чисел. Количество контролируемых символов ограничено количеством кнопок клавиатуры, выделенных для идентификации различных символов.

Принцип работы тренажера.

При запуске программы генератор случайных чисел формирует контролируемый символ, который высвечивается в начальной области информационной строки (например, в позиции 0 … 8) и с заданной (начальной) скоростью начинает перемещаться по последовательным позициям строки экрана.

Скорость перемещения предъявляемых символов задается с помощью таймера. При переполнении таймера положение символа изменяется на одно знакоместо вправо (при этом на «старом» знакоместе символ стирается и помещается на «новое» знакоместо). Поскольку процедура обновления содержимого экрана является достаточно продолжительной, ее целесообразно выполнять в основной (фоновой) программе. В относительно «коротком» обработчике прерывания по переполнению таймера выполняются только перезагрузка таймера константой,

определяющей скорость перемещения символов, и изменяется координата очередного отображаемого символа.

В основном цикле программы выполняется сканирование клавиатуры. Если определяется нажатая клавиша, соответствующая предъявляемому символу

При работе программы тестирования пользователь с помощью соответствующей идентификационной клавиши реагирует на предъявляемую информационную строку. Сканирование клавиатуры (определение нажатой клавиши) выполняется в основном цикле программы. При «правильной» реакции счетчик правильных ответов инкрементируется, «старый» символ исчезает с экрана и предъявляется «новый». При «неправильной» реакции возможны варианты продолжения программы. Например, инкрементируется счетчик неправильных ответов, «старый» символ исчезает с экрана и предъявляется «новый». Другим вариантом является только фиксация неправильного ответа с возможностью исправления: символ продолжается перемещаться по экрану. При исправлении (нажатии «правильной» клавиши) тренажер продолжает работу.

Если за время перемещения символа по экрану «правильная» клавиша не нажата, по достижению границы строки контролируемый символ исчезает с экрана и предъявляется новый символ, при этом в счетчик неправильных ответов добавляется штрафная сумма, например, 5 (не отвлекайся !!!).

Интерфейс программы.

При вызове программы тренажера на экране ЖКИ может высвечиваться какое-либо приветственное сообщение с приглашением начать работы, например, VNIMANIE. PRESS ANY KEY

Далее при нажатии определенной (или любой) клавиши на экране высвечивается меню, с помощью которого обеспечивается настройка тренажера на заданный режим работы (выбор уровня сложности, скорости перемещения символов, включение помех и пр.).

При нажатии кнопки «Пуск» начинается тестирование.

При нажатии кнопки «Стоп» тестирование заканчивается, при этом на экран ЖКИ выводится сообщение с результатами тестирования:

- время затраченное на тестирование $X_{мин}$ $Y_{сек}$;
- количество правильных и неправильных ответов.

Далее при нажатии любой клавиши высвечивается стартовое сообщение и может быть начато новое тестирование.

Принцип работы генератора случайных чисел (ГСЧ) – генератора псевдослучайных 8-битных слов.

Имеются две 8-битных константы A и B. Первое число формируемой последовательности случайных чисел X_1 является младшим байтом произведения A и B. Все последующие числа определяются в соответствии с формулой:

$$X_{n+1} = A \cdot X_n + B, \quad \text{здесь } X_n \text{ – текущее число.}$$

Для получения последовательности с «хорошими» свойствами при подборе констант (коэффициентов) A и B следует руководствоваться следующими правилами: числа A и B должны быть взаимно простыми и старший бит этих констант должен быть «1» (т.е. A и $B \geq 128$). В качестве результата (очередного числа последовательности) всегда берется младший байт произведения, обладающий большей «случайностью» по сравнению со старшим байтом.

При удачном выборе коэффициентов A и B можно получить псевдослучайную последовательность с хорошими свойствами (равномерность распределения длины периода и т.п.). Важным преимуществом рассмотренного ГСЧ является его простота: для поиска следующего числа последовательности необходимо знать только предыдущее число (фактически отсутствуют затраты памяти). Недостатком рассмотренного ГСЧ является низкая защищенность генерации – легко подобрать формулу, с помощью которой можно предсказать следующее число последовательности.

Существуют ГСЧ, при построении которых используются более сложные математические зависимости. Они обладают лучшими качествами (линейный конгруэнтный, вихрь Мерсенна и др.).

Некоторые рекомендации:

Важной функцией программы «Тренажер внимания» является перемещение символа по экрану. Эта функция реализуется в обработчике прерывания по переполнению таймера T_0. Обработчик не только осуществляет перезагрузку таймера константой, определяющей требуемую задержку (скорость перемещения отображаемого символа), но и изменяет координату символа в строке – переменную *pos*.

Само перемещение реализуется в основной (фоновой) программе: если при сравнении текущей координаты символа $pos_т$ с переменной *pos* выявляется несовпадение, необходимо выполнить стирание отображаемого символа в текущей позиции и перезаписать его в новую позицию.

Вариант реализации:

```
                mov dptr,#str1
                .
                .
m1:             mov a, posт                 ; стирание символа в текущей позиции
                cjne  a, pos, m1
                mov a, # ‘ ‘
                movx @dptr, a

                mov a, pos
                mov posт , a

                inc dptr                ; запись символа в новую позицию
                mov a, # smvl
                movx @dptr, a
```

П5.4. Электронный экзаменатор

Электронный экзаменатор – устройство, отображающее (предъявляющее) на экране модуля ЖКИ контрольные вопросы теста и варианты ответа на них. Контрольные вопросы отображаются в верхней (1-й) строке дисплея ЖКИ, варианты ответа – в нижней строке.

При вызове программы «Электронный экзаменатор» на экране модуля ЖКИ должно высвечиваться приветствие - приглашения начать новое тестирование, например, «Press any key to start». После нажатия любой клавиши запускается тест: в верхнюю строку экрана ЖКИ выводится 1-й вопрос, а в нижнюю – 1-й вариант ответа. Одновременно запускается таймер, подсчитывающий время тестирования.

Для просмотра вариантов ответа обучаемый (тестируемый) может использовать специально выделенные кнопки клавиатуры, например, кнопки одной из строк клавиатуры (1-я кнопка – 1-й вариант ответа, 2-я кнопка – 2-й вариант ответа и т.д.) или использовать специальные кнопки прокрутки вариантов ответа (вверх и вниз по списку вариантов ответа).

Ответ на вопрос вводится при нажатии кнопки с номером выбранного варианта ответа в клавишной строке ответов, при этом выбор варианта ответа может выполняться, независимо от того все ли варианты ответа просмотрены. После ответа на очередной вопрос (автоматически) появляется следующий вопрос и 1-й вариант ответа на него. Количество правильных и неправильных ответов фиксируется.

После завершения тестирования (ответа на последний вопрос) результат тестирования отображается в текстовом сообщении на экране ЖКИ: в первой строке отображается число правильных и неправильных ответов, во второй – время выполнения теста. При нажатии на кнопку «Сброс» осуществляется возврат в исходное состояние программы с отображением приветствия.

Число контрольных вопросов теста и число вариантов ответа на каждый вопрос определяется заданием.

Программа может быть дополнена блоком оценки качества тестирования. Оценка тестирования зависит от числа правильных ответов. Например, «отлично» - при числе правильных ответов больше 80 %, «хорошо» - 60 %, «удовлетворительно» - 50 %, «неудовлетворительно» - меньше 50 % правильных ответов или тест не выполняется в контрольное время.

Программа может быть расширена более подробным анализом результатов тестирования с выводом номеров вопросов, на которые был выбран неправильный ответ.

П5.5. Многофункциональный генератор инфранизких частот

Многофункциональный генератор инфранизких частот должен формировать аналоговые сигналы заданной формы (синусоидальный, колоколообразный, треугольный, пилообразный, трапециидальный прямоугольный) в заданном диапазоне частот.

Для воспроизведения сигналов сложной формы (требуемой функции времени) широко используются функциональные генераторы, среди которых в настоящее время наиболее высокими метрологическими и эксплуатационными характеристиками обладают функциональные генераторы, построенные на принципе преобразования кода в аналоговый сигнал, известные в литературе как цифро-аналоговые генераторы.

Основным методом приближенного представления сигналов заданной формы является метод аппроксимации. Сигналы заданной формы сравнительно просто аппроксимируются ступенчатой кривой. Генераторы аппроксимационного типа имеют структуру, легко перестраиваемую с одного вида функциональной зависимости на другой, что позволяет реализовывать на их основе многофункциональные генераторы.

Функциональные генераторы аппроксимационного типа воспроизводят на своем выходе аппроксимирующую функцию $f(t,A)$, зависящую в общем случае от фиксированного числа параметров $A = \{\alpha_1, \alpha_2, \ldots, \alpha_N\}$, которую получают в результате решения задачи приближения $f(t,A)$ к $f(t)$.

В основе решения задачи приближения лежит возможность представления заданной функции $f(t)$ со сколь угодно большой точностью в смысле равномерной или среднеквадратичной нормы ошибки в виде обобщенного полинома $f(t,A)$ по системе базисных функций $\{\xi_i(t)\}_{i=0}$

$$f(t,A = \sum \alpha_i \cdot \xi_i(t),$$

где α_i – весовые коэффициенты при базисных функциях.

Основной структурой функционального генератора аппроксимационного типа является цифро-аналоговый генератор, в

котором для реализации математических алгоритмов представления функциональных зависимостей используется метод весового суммирования.

Наиболее сложным блоком функционального генератора аппроксимационного типа является цифро-аналоговый преобразователь, преобразующий выходной цифровой код генератора в выходной аналоговый сигнал.

При формировании инфранизких частот для преобразования кода в напряжения можно использовать ШИМ-генератор с выходным фильтром низких частот. ЦАП, реализованный на базе ШИМ-генератора с выходным ФНЧ, обладает высокими точностными характеристиками, но имеет низкое быстродействие. На выходе ФНЧ (в простейшем случае интегратора) с некоторой задержкой, связанной с инерционностью RC-цепочки, формируется напряжение требуемого уровня. Максимальная частота ШИМ при 8-разрядном управлении равна $f_0/256$. Для f_0 = 1 МГц частота ШИМ равна ≈ 4 КГц (Δt = 256 мкс). Частота ШИМ является важным параметром генератора аппроксимационного типа, поскольку она также определяет период дискретизации формируемого сигнала заданной формы и частоту формируемых сигналов ($T_{сигн\ мин} = n \cdot \Delta t$), где n – число точек аппроксимации, Δt – период импульсов дискретизации. (Число точек аппроксимации в n раз уменьшает частоту формируемого сигнала). При увеличении частоты формируемого сигнала (уменьшении периода дискретизации и значения частоты ШИМ) на выходе ФНЧ формируется сигнал с нежелательными искажениями формы. Степень искажения определяется видом формируемого сигнала (ФНЧ не пропускает фронты сигнала и наиболее эффективен при формировании «гладких» сигналов).

При числе точек аппроксимации n = 20 и опорной частоте ШИМ-генератора 1 МГц максимальная частота воспроизводимых сигналов заданной формы не превышает 200 Гц. Нижняя частота воспроизводимых сигналов ограничена максимально возможным значением периода импульсов дискретизации Δt. При больших значениях параметра Δt пульсации формируемых сигналов на выходе фильтра нижних частот могут оказаться выше допустимого значения, например, 50 мВ.

Оценим величину пульсаций на выходе интегратора (ФНЧ)

$$U_x = [U_c(0) - U_c(\infty)] \cdot e^{-t/\tau} + U_c(\infty)$$

$$U_B = [U_A - 5] \cdot e^{-t/\tau} + 5$$

$$U_A = [U_B - 0] \cdot e^{-t/\tau} + 0$$

$$U_B - U_A = 5 \cdot (1 - e^{-t/\tau}) / (1 + e^{-t/\tau})$$

Условие работоспособности: $\tau = RC \geq \Delta t$ (при выполнении этого условия напряжение U_c за время между импульсами ШИМ изменится незначительно).

$$\Delta U = U \cdot t / RC$$

Наибольшие пульсации имеют место при формировании среднего уровня выходного сигнала, когда $t_p = t/2$

Оценим пульсации для диапазона частот дискретизации (частот ШИМ) 4КГц – 400 Гц $t = 256$ μc $t_p = 128$ μc

$t = 2{,}56$ mc $t_p = 1{,}28$ mc

Пусть $\Delta U = 100$ мВ

$100 \cdot 10^3 = 2{,}5$ В $\cdot\ 1{,}28\ 10^{-3}$ с / $1 \cdot 10^3$ ом $\cdot\ 100 \cdot 10^{-3}$ В $= 3{,}3\ 10^{-5}$ Ф $= 0{,}33$ мкФ

$\tau = RC = 1 \cdot 10^3$ ом $\cdot\ 0{,}33$ мкФ $= 33$ мс

8-разрядный код управления скважностью $N_{упр}$, определяющий точность задания выборки) рассчитывается для максимальной частоты дискретизации (максимальной частоты ШИМ $f_{макс} = 4$ КГц). При уменьшении частоты дискретизации (увеличении Δt) значения кодов $N_{упр}$ должны быть пересчитаны (промасштабированы)

$N_{упр}$ - 256 мкс

$$N_x = N_{упр} \cdot \Delta t / 256$$

N_x - Δt

Для качественного восстановления сигнала на основе его выборки, каждая из выборок должна удерживаться в течение нескольких периодов Δt (время удержания должно быть достаточным для точного интегрирования уровня выборки).

Приложение 6

Порядок подготовки и проведения лабораторных НИР

Лабораторные НИР по курсу «Микропроцессорные системы» помогают закрепить теоретические знания соответствующей учебной дисциплины. Выполнение НИР обеспечивает практическое освоение методов построения микроконтроллерных систем, способствует приобретению навыков решения прикладных задач с использованием устройств ввода-вывода разных типов, знакомит с программированием на языке Ассемблера.

Отдельные исследования по темам лабораторного практикума выполняются студентами индивидуально в течение четырехчасовых посещений в лаборатории.

Подготовка к проведению НИР. Выполнению НИР должна предшествовать обязательная домашняя подготовка. В процессе подготовки к НИР по материалам разделов 1–3 данного пособия изучаются принципы организации и особенности функционирования исследуемого микроконтроллера. Итогом домашней подготовки является макет отчета по НИР, содержащий краткое описание работы исследуемых подсистем микроконтроллера, структуру информационных связей МК с подключаемыми внешними устройствами (клавиатурой, модулем ЖКИ, датчиками аналоговых сигналов), алгоритмы и программы тестовых и индивидуальных заданий, протокол исследований с таблицами для записи результатов. Наличие макета отчета является основанием для допуска к выполнению исследований в лаборатории.

Проведение НИР. При выполнении работ для реализации разработанной структуры информационных связей МК с подключаемыми внешними устройствами (клавиатурой, модулем ЖКИ, датчиками аналоговых сигналов) необходимо соединить разъемы портов МК с разъемами подключаемых устройств. Результаты исследований и данные по отладке программ заносятся в протокол отчета. Записи в отчете должны быть четкими и ясными. Выполняемые программы должны быть снабжены поясняющими комментариями.

Исследования отдельных циклов работ предполагают их выполнение в течение 2-х - 3-х лабораторных занятий. Работа считается выполненной после подписи преподавателем протокола отчета. Единый (окончательно оформленный) отчет по законченному исследованию представляется к проверке и защите не позднее начала работ следующего цикла.

Рекомендуемое содержание отчета.

Отчет должен содержать титульный лист и собственно отчет.

На титульном листе необходимо указать тему НИР, дату ее проведения, фамилию, имя, отчество студента и номер его академической группы. (Образец представлен ниже).

Собственно отчет должен включать следующие разделы:

– цель исследования;

– программу исследований;

– краткое описание работы исследуемых подсистем микроконтроллера;

– схемы информационных связей МК с подключаемыми внешними устройствами (клавиатурой, модулем ЖКИ, датчиками аналоговых сигналов);

– схемы алгоритмов и программы работы по каждому пункту заданий;

– протокол исследований с таблицами результатов и комментариями по каждому пункту программы исследований (графики, полученные средствами оболочки Shell51, осциллограммы зафиксированных процессов);

– анализ полученных результатов на каждом этапе исследования и общие выводы по результатам работы в целом.

Приложение 7. Образец титульного листа отчета

Санкт-Петербургский государственный политехнический университет
Институт информационных технологий и управления
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе
по курсу «Микропроцессорные системы»
(Название отчета соответствует теме выполненной работы), например,
«Изучение вычислительных возможностей МК SAB 80C515»

Работу выполнил студент группы № ФИО

Работу принял преподаватель ФИО

Санкт-Петербург

201_

ПАВЛОВСКИЙ Евгений Григорьевич
ЖВАРИКОВ Владимир Анатольевич
КУЗЬМИН Александр Александрович

ОСНОВЫ ОРГАНИЗАЦИИ МИКРОКОНТРОЛЛЕРОВ

Учебное пособие и методические указания к лабораторному практикуму

Подписано в печать 15.03.2014??. Формат 60x84
Усл. печ. л. Уч.-изд. л. Тираж 100 экз.

Отпечатано с оригинал-макета авторов в центре оперативной полиграфии
факультета технической кибернетики СПбГПУ
195251 Санкт-Петербург, Политехническая ул.,,21