Федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Курс лекций

дисциплины базовой части математического и естественнонаучного цикла Б.1

«Микропроцессорные системы ч.2»

(7 семестр)

Направление подготовки 09.03.01 Информатика и вычислительная техника

Профили: Автоматизированные системы обработки информации и управления,

Вычислительные машины, комплексы, системы и сети

Авторский коллектив:

Доцент кафедры ВМСС, к.т.н. _____ С.Н. Михалин

«01» сентября 2017 г.

МИКРОКОНТРОЛЛЕРЫ

Микроконтроллер (МК) можно определить как самостоятельную микропроцессорную систему (МПС), которая содержит ядро, вспомогательные схемы и устройства ввода-вывода, размещенные в одном корпусе и предназначенная для управления различными электронными устройствами. Как следует из определения, основным назначением микроконтроллеров является программное управление объектами и связь с внешними устройствами (это преобразование интерфейса, функции обработки аналоговых сигналов, мониторинг показателей и т.п.). На рис.1 представлена обобщенная структура МК. Можно видеть, что МК содержат те же узлы, что и микропроцессорные системы.

Рис. 1 – Общая структура микроконтроллера

«Ядро» – базовое устройство внутренней вычислительной системы. Ядро определяет систему команд, шинный интерфейс, архитектуру памяти, т.е. главные отличия вычислительных систем друг от друга. Ядро МК может быть одинаковым, а изготовители – разными.

«Семейство» – группа микросхем, имеющих одно ядро, у которых примерно одинаковый набор программных и периферийных функций.

«Серия» или «линейка» – это фирменный бренд (реклама), например, серия «*Classic*», серия «*tinyAVR*», линейка «*MegaPIC*». Встречаются и обобщенные названия по типу «линейка 16-битных МК общего назначения».

«Модель» – несколько микросхем одного семейства, различающиеся между собой второстепенными цифрами (буквами) в названии, что определяет разный температурный диапазон, тактовую частоту, вариант корпуса, диапазон питающего напряжения и т.п.

Как и микропроцессоры (МП) развитие архитектуры и изменение состава периферии различных МК происходит быстро. Например, только одна компания *Microchip* за 1997 год выпустила свыше 40 видов МК, а клонов распространенного *Intel* 8051 на сегодняшний день существует более 200 модификаций, выпускаемых двумя десятками компаний. Практическая потребность в МК (множество задач, решаемых с помощью МК) побуждает многих производителей выпускать целые серии различных МК. Среди таких компаний можно назвать *Atmel*, *Dallas Semiconductor*, *Intel*, *Analog Devices*, *Infineon Technologies*, *Hitachi Semiconductor*, *Microchip Technology Inc.*, *Mitsubishi Electronics*, *National Semoconductor*, *Philips Semiconductors*, *STMicroelectronics*, *Texas Instruments*, *Toshiba*, *Zilog* и др. Отметим, что и в СССР в 1979 году была разработана однокристальная 16-разрядная ЭВМ К1801ВЕ1, микроархитектура которой называлась «Электроника НЦ». В связи с этими обстоятельствами нерационально изучать конкретный вид МК, но имеет смысл в первую очередь проанализировать общие подходы к их построению и применению, а затем рассмотреть наиболее популярные и перспективные серии.

Все МК можно разделить по назначению на три большие во многом пересекающиеся группы:

- встраиваемые 8-разрядные МК (*embedded*) – дешевые, простые в применении, содержащие минимальное количество компонентов для реализации самых простых задач (требующих малого потребления, невысокой производительности и небольших объемов памяти);
- 16- и 32-разрядные МК – устройства, на кристалле которых интегрированы специфические блоки: такие как монитор отладки, аналоговый ввод-вывод, развитая система прерываний, наличие внешней полноценной шины для подключения внешней памяти, процессорных устройств и т.п. (для управления несколькими устройствами, простейшей обработки НЧ сигналов);
- цифровые процессоры *DSP* – 32-х разрядные и более МК с высокой производительностью и специализацией системы команд под стандартные процессы обработки данных (фильтрация, БПФ, сверточное кодирование, преобразование кодов и т.п.), могут содержать несколько вычислительных модулей и средства поддержки вычислений с плавающей точкой.

В свою очередь МК каждой этой группы можно разделить по принципу построения:

- МК с Гарвардской архитектурой (разделены память программ и данных; наличие однородных по назначению регистров общего назначения);
- МК с Принстонской (Фон Неймана) архитектурой (память программ и данных едина, регистры имеют предписанное назначение, например: аккумулятор);
- МК с *RISC* ядром (сокращенная система команд);
- МК с *CISC* ядром (полная система команд).

Разные архитектуры каждого МК, построенные с тем или иным ядром, обладают как преимуществами, так и недостатками. Однако есть МК, в которых разработчики сделали попытку соединить достоинства разных архитектур. Поэтому решение одной и той же задачи, реализуемой на разных МК, должно быть оптимизировано под конкретную архитектуру и систему команд конкретного МК.

С учетом сказанного рассмотрим обобщенные структурные схемы основных узлов применяемых в МК (таймеры, подсистемы прерываний, параллельный и последовательный интерфейсы, организация памяти, принципы построения МК и т.п.). Затем рассмотрим конкретные реализации этих идей в семействах МК *Intel* 8051 (как классический *CISC* МК с гарвардской архитектурой), *Motorola 68HC*05*xx* (*CISC* МК с принстонской архитектурой), *Atmel AVR AT*90*S*xxxx (как классический и простейший *RISC* МК с гарвардской архитектурой).

АППАРАТНЫЕ СРЕДСТВА МИКРОКОНТРОЛЛЕРОВ

Запуск (сброс в начальное состояние) МК осуществляется подачей питания или активным уровнем сигнала на входе *Reset*. После включения питания МК должен начать работу после окончания переходных процессов, связанных с установлением нормального уровня питающего напряжения и нормальной частоты и формы тактирующих импульсов. Для обеспечения задержки между появлением питания и началом работы применяют *RC*-цепочку, включенную к входу *Reset*. Часто такая схема задержки уже встроена в МК.

Тактирование системы осуществляется от внешнего или встроенного генератора – рис.2. Для работы последнего обычно требуется включение внешнего элемента – кварцевого резонатора (*Q*1). Однако если при работе МК не требуется измерений или отсчета интервалов точного времени, то вместо кварца допускается включение *RC*-цепи.

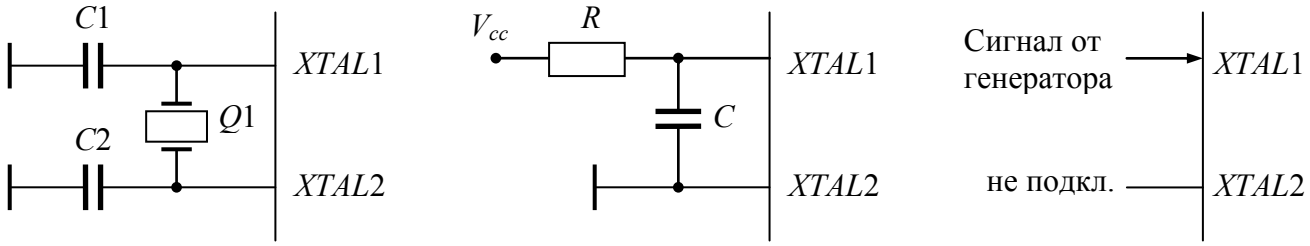


Рис. 2 – Схемы тактирования МК

Командный цикл МК состоит из нескольких тактов синхронизации, которые необходимы для выполнения команд. Некоторые команды могут требовать нескольких командных циклов, что свойственно *CISC* архитектурам.

Арифметико-логическое устройство (АЛУ) представляет собой аппаратный блок, реализующий операции сложения, вычитания (умножения, деления), логических операций (*and*, *or*, *xor*, *not*) и сдвигов влево и вправо. АЛУ не используется при чтении или записи данных или команд, оно служит только для обработки данных. АЛУ обрабатывает два операнда и сохраняет результат в третьем. Но физическая реализация (источники и приемник данных) зависит от конкретной модели МК.

Сторожевой таймер – это аппаратная схема, позволяющая исключать из работы МК или процессора ситуации его зависания, блокировки и т.п. Устройство вызывает сброс системы, если через фиксированный момент времени (обычно от нескольких десятков миллисекунд до нескольких секунд) содержимое определенного регистра не будет обновлено. Проще говоря, запускается аппаратный таймер, считающий к нулю, происходит проверка управляющего слова и, если записи в управляющее слово за время счета таймера не произошло, то генерируется сигнал сброса МК, иначе счетчик таймера перезагружается и начинает счет сначала.

Прерывание – это аппаратная реализация отклика синхронной системы на асинхронное событие. Любой МК, получив запрос на прерывание, может реагировать на него одним из трех способов:

а) игнорировать запрос (маскирование прерывания) и при необходимости реализацией поллинга выполнение прерывания позже – например, после завершения текущей задачи;
б) быстро среагировать на запрос, сообщив внешнему устройству (которое выставило прерывание), что МК занят и процедура обработки поступившего прерывания будет выполнена позже – например, после завершения текущей передачи данных;
в) быстро среагировать и обслужить запрос полностью, а затем вернуться к прерванной задаче.

Обработчик прерываний обычно выполняет следующую последовательность действий: сохранение контекста регистров, посылка управляющей комбинации подсистеме прерываний и внешнему устройству, обработка данных, восстановление контекста регистров, возврат к прерванной программе. В МК передача управления обработчику и возврат управления прерванной программе реализуются аппаратно, но в любом случае для этих целей применяется стек, глубина которого ограничивает количество вложенных прерываний.

Таймеры – аппаратный счетчик со схемой предварительного деления частоты, с регистрами управления, статуса, начальной загрузки и защелки (позволяющей читать значение счетчика «на лету»), а также со схемами коммутации входного и выходного сигналов. Обычно один МК содержит несколько таймеров (необязательно одинаковых). На рис.3 представлена обобщенная схема таймера.

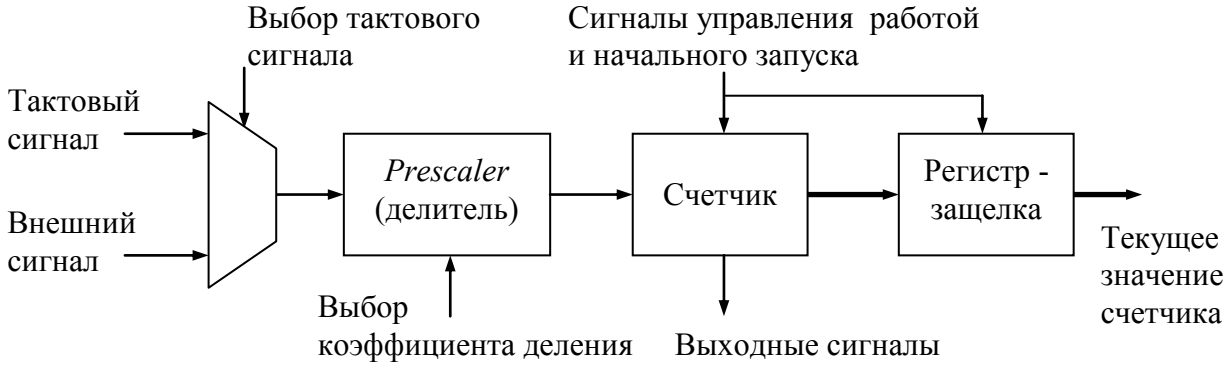


Рис. 3 – Обобщенная структурная схема таймера

Часто в МК таймеры используются для вывода сигналов с широтно-импульсной модуляцией (ШИМ или *Pulse Width Modulated*). Для реализации режима ШИМ к структуре рис.3 необходимо добавить два регистра и два компаратора – рис.4.

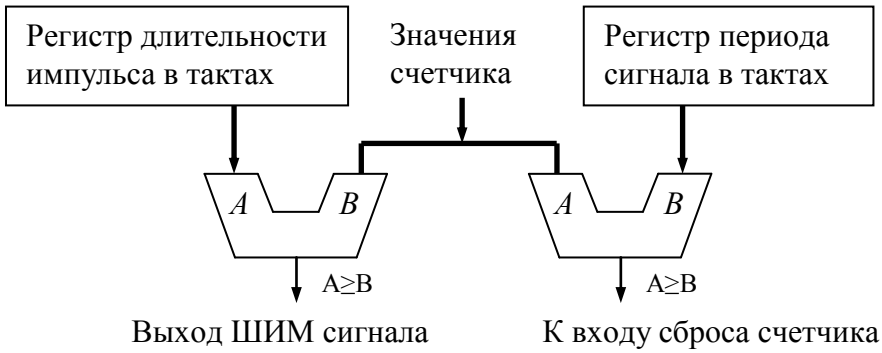


Рис. 4 – Дополнение структурной схемы таймера

Параллельный ввод-вывод данных является основным интерфейсом МК – порт. Во многих МК для сокращения внешних выводов микросхемы порты ввода/вывода сочетаются с другими функциями, а выбор назначения такого контакта осуществляется программным способом. Типовая схема организации линии (одного бита) порта приведена на рис.5.

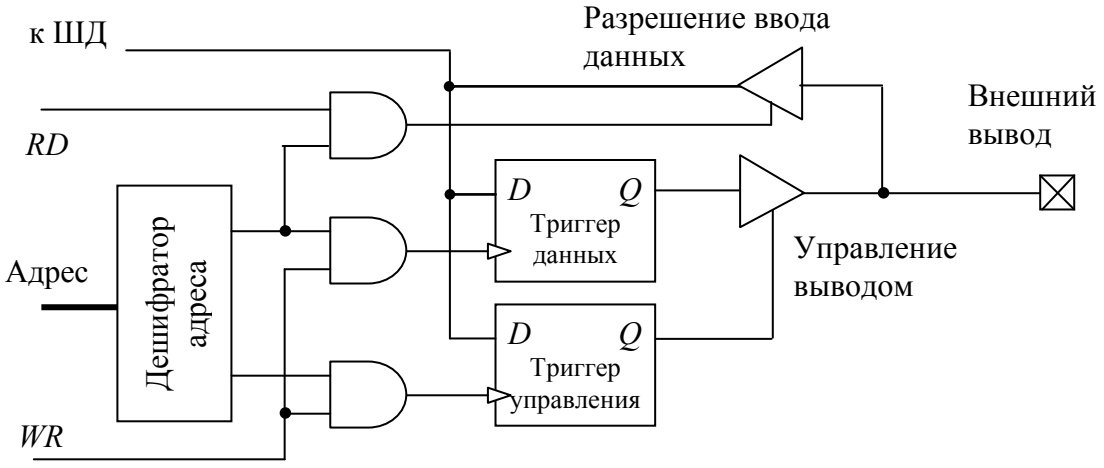


Рис. 5 – Типовая схема линии порта ввода-вывода

Последовательный ввод-вывод – наиболее распространенный вид связи для реализации обмена между несколькими МК или между МК и различными устройствами. Существует множество протоколов передачи данных последовательным кодом, которые имеют различное назначение (обмен с другими контроллерами, управление чем-либо, обмен с датчиками): *Microwire*, *SPI*, *CAN*, *I2C* и т.п.

Асинхронный обмен *SCI* (*Serial Communication Interface*) предполагает пакетную передачу данных, т.е. аналогичен интерфейсу *RS*-232 (*COM* порт *x*86). Часто для увеличения помехоустойчивости передачи применяют кодирование как сообщения сигнала так и логических уровней (например: лог. 1 это перепад с высокого напряжения на низкое, а ноль – наоборот или логические уровни кодируются серией перепадов). Примеры: локальная вычислительная сеть – манчестерский код, *USB* – дифференциальная передача с кодированием лог. «0» как перепад напряжения и самосинхронизацией.

Синхронный обмен предполагает непрерывную передачу данных, характерную потоковому или конвейерному методу обработки данных.

В частности, для подключения *Flash*-памяти, медленных АЦП применяют: *SPI*, а для обмена между несколькими МК: *I2C* и *CAN*.

Краткие сведения о *SPI* и *Microwire*

Последовательный периферийный интерфейс *SPI* (*Serial Peripheral Interface*) обеспечивает высокоскоростной синхронный дуплексный обмен данными между МК и периферийными устройствами или между несколькими МК (по дальности: в рамках платы, блока и т.п.). В общем случае, один из многих МК или устройств должен быть ведущим (*Master*), остальные – ведомыми (*Slave*). Для обеспечения обмена данными между устройствами используются четыре линии: *SCK* (*Serial clock*), *MOSI* (*Master Output data*, *Slave Input*), *MISO* (*Master Input data, Slave Output*), *SS* (*Slave Select*). Сигнал *SS* используется ведущим для выбора одного из ведомых устройств (при обмене типа "точка-многоточка"). Тактовые сигналы по линии *SCK* всегда генерирует ведущий МК. Данные от ведущего передаются по линии *MOSI*, прием данных осуществляется по линии *MISO*. Если ведомым устройством является МК, то его вывод, соответствующий линии *SS*, должен быть настроен как вход. Протокол *SPI* обеспечивает:

- скорость передачи достигает 3 Мбит/сек.;
- разрядность данных в пакете составляет 8 бит;
- передатчик может приостанавливать передачу данных;
- данные могут передаваться блоками (страницами), которые объединяют множество байтов.

Формат пакета *SPI* представляет собой байт, информативное наполнение которого определяется разработчиком системы и, например, серия байтов может включать: 8-разрядную команду, необязательный 16-разрядный адрес и 8-разрядные данные.

Тактовые сигналы представляют собой меандр. Выходные данные выдаются на линию не позже чем за 30 нс до переднего фронта тактового сигнала, а считывание происходит за 30 нс до заднего фронта. На рис.6 показана временная диаграмма обмена данными по интерфейсу *SPI* для случая передачи байта (восемь периодов сигнала *SCK*), *CPOL*=0 – управляющий бит определяет полярность сигналов *SCK*, *CPHA*=0 – фаза синхронизации: по переднему фронту в цикле синхронизации будет выполняться выборка данных, а по заднему фронту – установка данных, в цикле обмена *SS*=0.

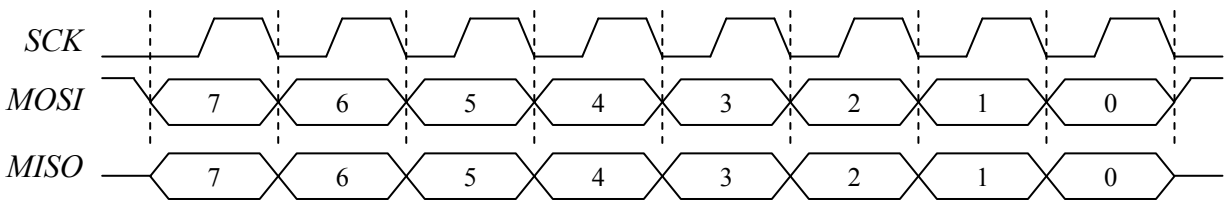


Рис. 6 – Временная диаграмма обмена данными по интерфейсу *SPI*

На рис.7 представлены схемы соединения устройств (*slave*) с МК (*master*) по шине *SPI* (вариант кольцевого соединения поддерживается не всеми устройствами).

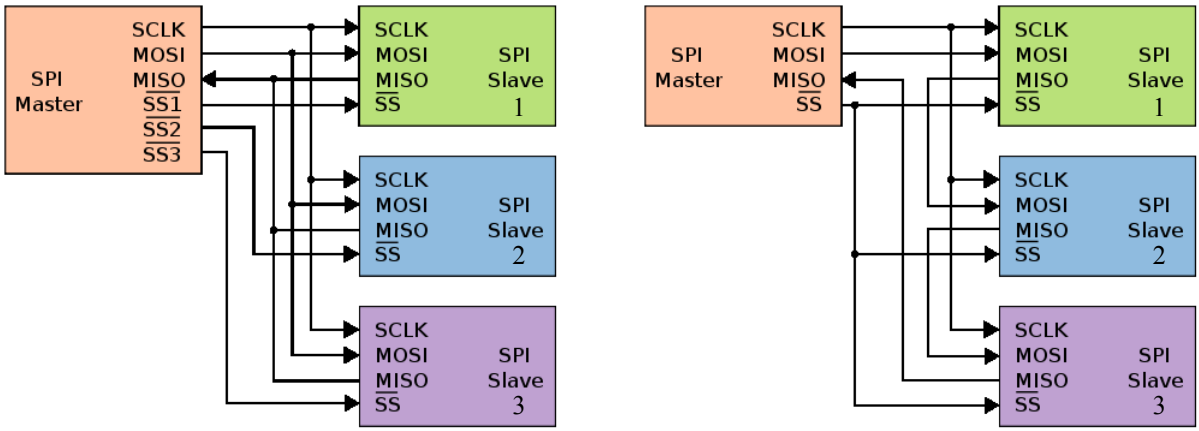


Рис.7 – Схемы соединения по шине *SPI.* Слева – радиальная; справа – кольцевая.

Протокол *Microwire* является клоном *SPI*, в котором более строго оговариваются правила "общения" устройств, однако он почти не отличается от *SPI*, обеспечивая передачу данных со скоростью до 1 Мбит/сек. Структура пакета состоит из 8-разрядной команды, необязательного 16-разрядного адреса и 16-разрядных данных. Передаваемые данные должны выдаваться на линию за 100 нс до поступления переднего фронта тактового сигнала. Чтение данных должно происходить за 100 нс до заднего фронта тактового сигнала.

Основными недостатками интерфейсов типа *SPI* является:

- нет подтверждения приема данных со стороны ведомого устройства;
- нет определенного стандартом механизма обнаружения ошибок;
- ведомое устройство не может управлять потоком данных.

Краткие сведения о *I2C*

Шина *I2C* (*Inter-Integrated Circuit*) была разработана компанией *Philips* в конце 70-х и позволяет разделять ресурсы между несколькими устройствами (до 112 шт., 16 адресов зарезервированы). Это двунаправленная (полудуплексная) синхронная шина с последовательной передачей данных (стандартизована в 1992 г.).

Шина содержит две сигнальные линии: *SCL*-синхросигналы (*serial clock*) и *SDA*-данные (*serial data*), выполненные по схеме с открытым коллектором. Поэтому обе линии подтягивают к высокому уровню (через внешние резисторы к цепи питания) и такое состоянии линий является пассивным (*idle* состояние). Обмен информацией осуществляется байтами. Каждый байт передается в течение 9 тактов – периодов сигнала синхронизации на линии *SCL*.

Чтобы инициировать передачу данных ведущее устройство устанавливает низкий уровень сначала на линии данных *SDA*, а затем на линии *SCL* (стартовое состояние – рис. 8). В процессе пересылки данных такое состояние шины является нерабочим, т.к. прием данных осуществляется при высоком уровне синхроимпульсов на линии *SCL*. Данные передаются синхронно начиная со старшего бита. После передачи данных ведущее устройство переводит линию в плавающее состояние, ожидая подтверждения приема данных от ведомого устройства (рис.8). Таким подтверждением (*ACK*) является установка низкого уровня на линии *SDA*. Затем пересылается следующий байт или шина переводится в пассивное состояние (как показано на рис.8). Высокий уровень на линии *SDA* в девятом периоде свидетельствует о возникшей ошибке в приемнике (отсутствие подтверждения: *NAK*). Следует отметить, что передатчиком может быть как ведущее, так и ведомое устройства и, следовательно, получатель информации всегда должен формировать подтверждение *ACK*.

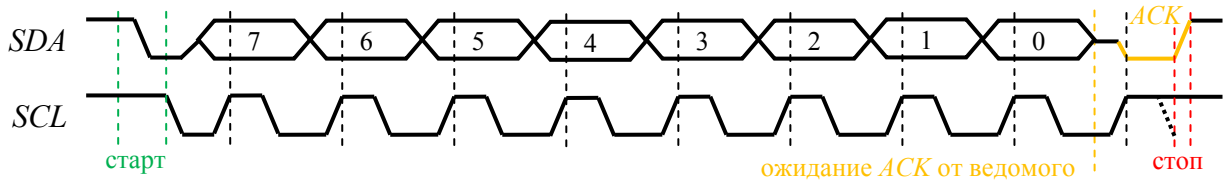


Рис. 8 – Временная диаграмма обмена данными по шине *I2C*

Максимальная скорость обмена по этому протоколу составляет 400 Кбит/сек (стандартно – до 100 Кбит/сек) при длине линии связи до 2 м (суммарная емкость не более 400 пФ), существуют «модификации» увеличивающие скорость передачи до 3.4 Мбит/сек.

Пакет состоит из адреса приемника (7 бит – 128 устройств), бита *R*/*W* указывающего направление передачи данных, бита подтверждения, собственно передаваемых или принимаемых данных (байт) и еще одного бита подтверждения. На рис. 9 показан цикл обмена.

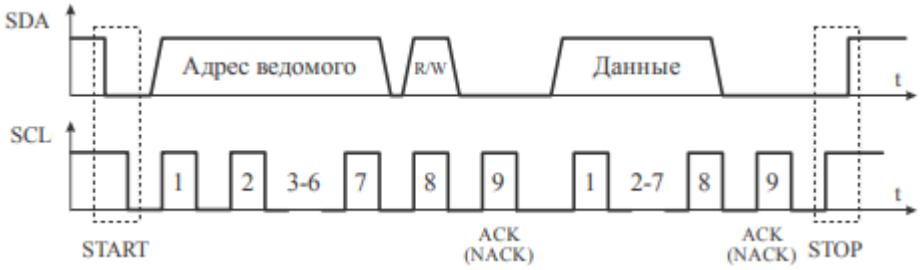


Рис. 9 – Временная диаграмма цикла обмена на шине *I2C*

Все устройства шины делятся на два типа: ведущее (*Master*) и ведомое (*Slave*). Тактовый сигнал *SCL* генерирует только ведущее устройство. Оно может самостоятельно выходить на шину и обращаться по адресу к любому ведомому устройству. При обнаружении собственного адреса ведомые устройства выполняют предписываемую операцию. Возможен режим работы шины с несколькими ведущими устройствами. На рис. 10 представлена схема соединения по шине *I2C*, резисторы *R* подтягивают потенциалы линий к напряжению питания.

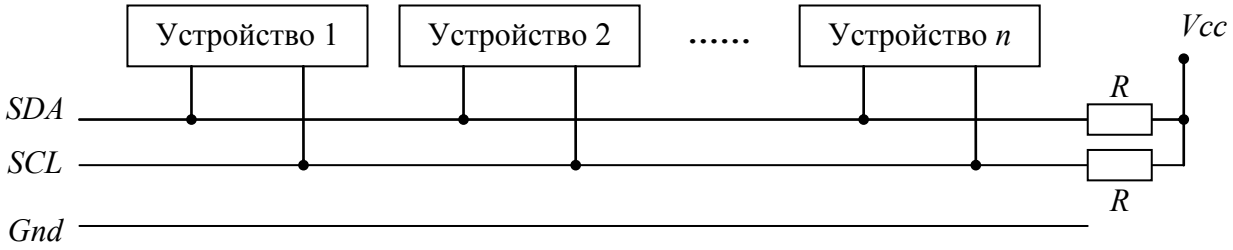


Рис. 10 – Схема соединения по шине *I2C*

Краткие сведения о *CAN*

Последовательный интерфейс *CAN* (*Controller Area Network*) был разработан компанией *Bosch* в 1980-х для связи компьютерных систем (главным образом в автомобилестроении) и удовлетворяет следующим требованиям:

- полудуплексный обмен до 1 Мбит/сек (на расстоянии до 30-40 м);
- нечувствительность к электромагнитным помехам;
- простота применения и небольшое количество разъемных контактов.

Физически протокол *CAN* основан на драйверах *RS*-485, обеспечивающих дифференциальную передачу сигнала, и реализуется с использованием операции «монтажное И» – рис.11.

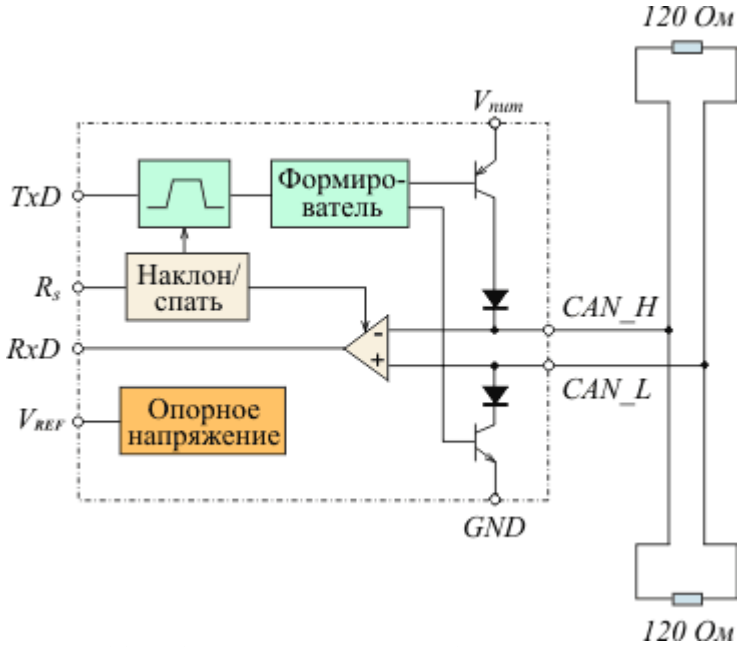


Рис. 11 – Структурная схема *CAN* трансивера

Если базироваться на семиуровневой модели сети *OSI*, то *CAN* описывает передачу данных между узлами на двух нижних уровнях – физическом и канальном. Битовый поток кодируется по методу *NRZ* (*Non Return to Zero* – кодирование без возврата к нулю – простое потенциальное кодирование – рис. 12). Для повышения устойчивости синхронизации используется вставка нулевого бита в случае следования подряд пяти единиц или вставка единицы в случае следования подряд пяти нулей (этот процесс называется *Bit Stuffing*).

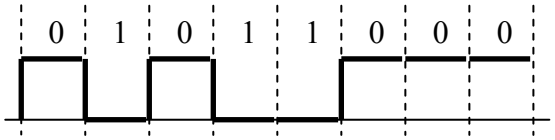


Рис. 12 – Пример *NRZ* кодирования

Кадр (фрейм) протокола включает:

- старт-бит (*SOF - start of frame*) – низкий уровень сигнала;
- идентификатор (11 бит - версия *A* или 29 бит - версия *B*);
- бит, определяющий направление передачи;
- два нулевых бита (резерв);
- длина передаваемого сообщения в байтах (4 бита);
- данные (0-8 байт);
- 15-битный *CRC* кода (формируется циклическим кодом и служит для обнаружения ошибок, возникающих при передаче кадра, вероятность необнаруженной ошибки: 4.7×10^{-11});
- один единичный бит (разделитель);
- 2-битное поле подтверждения готовности: бит подтверждения (*ACK*) и единичный бит;
- семь единичных бит окончания кадра (*EOF - end of frame*).

Отметим, что версия *B*, часто именуемая *FullCAN*, постепенно вытесняет версию *A*, которую называют также *BasicCAN*. Шина стандартизована *ISO* (*ISO* 11898) и *SAE* (*Society of Automotive Engineers*).

Между кадрами имеется промежуток *IFS* (*Inter Frame Spacing*), состоящий из не менее трех битов.

Рис. 13 – Кадр протокола *CAN*

Краткие сведения о 1-*wire*

Фирмой *Dallas Semiconductor* разработан последовательный асинхронный полудуплексный интерфейс, позволяющий обеспечивать низкоскоростной обмен данными по одной линии (вторая линия – общий провод). Сеть устройств, соединенных этим интерфейсом, известна под названием *MicroLAN*. К однопроводной шине могут быть подключены несколько устройств, но только одно из них является ведущим, а все остальные ведомыми. Допускается питание устройств по линии данных.

Все устройства шины самотактируемые, т.е. логика работы основана на измерении и формировании относительных временных интервалов (импульсов различной длительности). Выходной каскад ведущего строится на схеме с открытым стоком. Синхронизация обмена на шине происходит по падающему фронту (переключение с высокого на низкий уровень напряжения), поскольку в схемах с открытым стоком именно падающий фронт менее зависит от емкости нагрузки.

Активная часть временного интервала шины составляет 60 мкс. Для надежности приема данных, чтение осуществляется примерно в середине интервала (фактически выборка может осуществляться спустя 15-60 мкс после синхронизирующего фронта). После окончания активной части временного интервала требуется освобождение линии не менее чем на 1 мкс (в линии должен быть установлен высокий уровень). Логические уровни передаваемых данных кодируются разной длительностью импульса низкого уровня в линии – рис. 14:

- логическая "1" – длительность импульса <15 мкс;
- логический "0" – длительность импульса ≥60 мкс и <120 мкс.

Рис. 14 – Состояние линии при передаче логической "1" слева и "0" справа

Цикл обмена данными на шине начинается с формирования ведущим устройством состояния сброса. Оно заключается в передаче сигнала низкого уровня длительностью 480 мкс (8 активных периодов шины) и ожидания подтверждения от ведомого устройства в течение еще 480 мкс – рис.15 (пунктирной линией обозначен уровень напряжения, обеспечиваемый подтягивающим резистором). В ответ на импульс сброса каждое устройство, подключенное к линии, производит сброс своих внутренних цепей и через 15-60 мкс (после перевода ведомым линии в высокое состояние) выдает низкий уровень сигнала подтверждения в течение 60-240 мкс. Обнаружив этот импульс, ведущий узел передает адрес нужного ему устройства. Все устройства в 1-*wire* сети обладают идентификатором (адресом), который записан в ПЗУ микросхемы *MicroLAN* (это уникальный ее номер, производитель гарантирует невозможность выпуска двух микросхем с одним номером). Все устройства, адрес которых не совпал с

переданным, логически отключаются от шины до следующего импульса сброса. Выбранному устройству передается код операции обмена данными, который определяет направление передачи. По окончании операции ведущий узел генерирует новый импульс сброса и начинается новый цикл обмена.

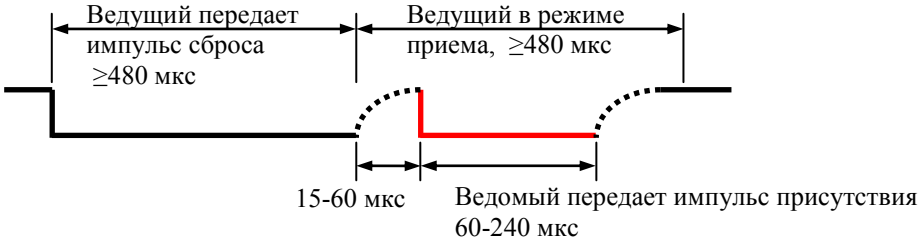


Рис. 15 – Инициализация на шине 1-*wire*

Передача данных в ведомое устройство начинается с установки ведущим устройством на линии низкого уровня длительностью не менее 15 мкс, затем устанавливается уровень передаваемого бита.

Приём данных из ведомого устройства также начинается с установки ведущим устройством в течение 15 мкс на линии низкого уровня, а затем ведущее устройство устанавливает на линии высокий уровень и не более чем через 15 мкс ведомое устройство должно выдать уровень нуля или единицы.

1. Общие сведения о МК *Intel* 8051

В 1980 году *Intel* выпустила МК 8051. Исторически это не первый МК компании, однако, удачный набор периферийных устройств, возможность выбора внешней или внутренней программной памяти и приемлемая цена обеспечили (в свое время) этому МК успех. С точки зрения технологии 8051 являлся для своего времени сложным изделием – кристалл содержит 128 тыс. транзисторов, что в 4 раза превышало количество транзисторов в 16-разрядном МП i8086. На сегодняшний день существует более 10 различных версий классического МК 8051 (87*C*51 – «7» означает наличие *EPROM*), которые совместимы между собой, но отличаются в основном внутренней архитектурой, позволяющей увеличить производительность.

Основными производителями клонов этого семейства МК являются: *Atmel*, *Philips*, *Siemens*, *Dallas*, *AMD*, *Winbond* и другие. В СССР производство его аналогов осуществлялось в Киеве, Воронеже, Минске и Новосибирске (1816ВЕ31/51, 1830ВЕ31/51, 1834ВЕ31, 1850ВЕ31).

МК 8051 имеет гарвардскую архитектуру с возможностью обращения к внешней памяти. На рис.1.1 представлена упрощенная структура МК, в его состав входят:

- внутренний генератор тактовой частоты *G*;
- арифметико-логическое устройство *ALU*;
- внутренняя память программ *ROM*;
- внутренняя память данных *RAM*;
- четыре восьмиразрядных порта ввода/вывода *P*0 – *P*3;
- интерфейсные устройства *IU* (реализуют специальные функции).

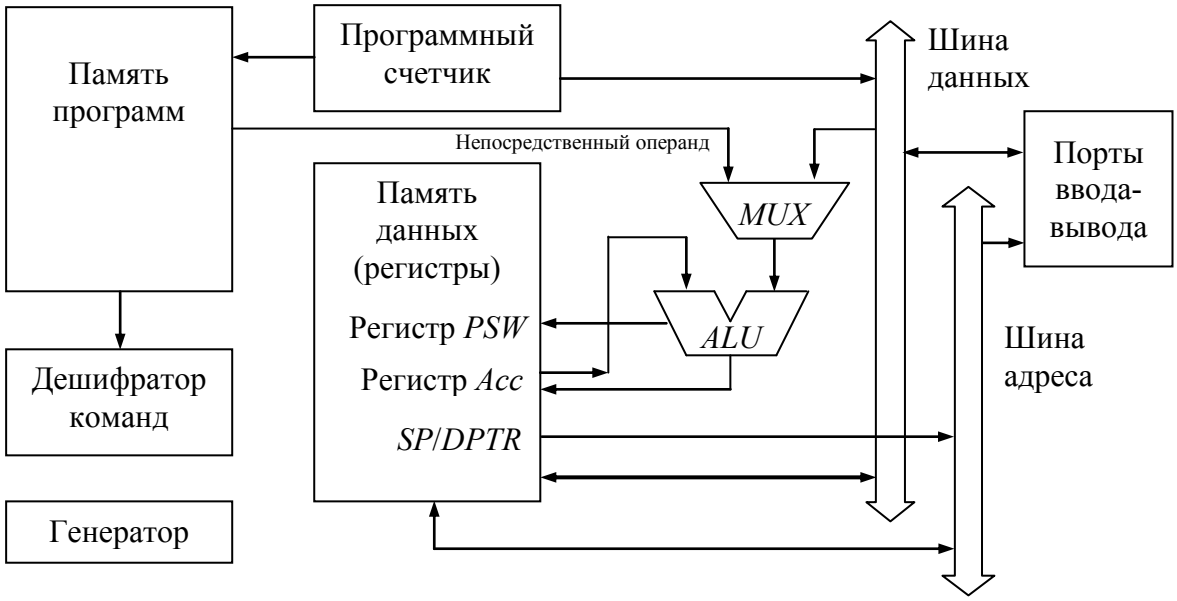


Рис. 1.1 – Упрощенная структурная схема МК 8051

Основные характеристики *CISC* МК 8051:

- восьмиразрядное ядро, оптимизированное для реализации функций управления;
- адресное пространство памяти программ – 64 Кбайт;
- адресное пространство памяти данных – 64 Кбайт;
- внутренняя память программ – 4 Кбайт;
- внутренняя память данных – 128 байт;
- дополнительные возможности по выполнению булевых операций (побитовые операции);
- 32 двунаправленные и индивидуально адресуемые линии ввода-вывода (4 порта);
- два 16-разрядных многофункциональных таймера/счетчика;
- полнодуплексный асинхронный приемопередатчик;
- векторная система прерываний с двумя уровнями приоритета.

Особенностью семейства этих МК является организация памяти – рис.1.2 (в адресном пространстве 080h-0FFh показаны только основные специальные функции).

<table>
<tr><th>Адрес</th><th colspan="2">Прямая адресация</th><th>Косвенная адресация</th></tr>
<tr><td>0FFh</td><td colspan="2"></td><td rowspan="32">Доступная память
(в некоторых моделях может отсутствовать в принципе)</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0F0h</td><td colspan="2">Регистр «B»</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0E0h</td><td colspan="2">Аккумулятор (Acc)</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0D0h</td><td colspan="2">PSW – слово состояния</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0B8h</td><td colspan="2">IP – регистр приоритетов</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0B0h</td><td colspan="2">Порт 3</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0A8h</td><td colspan="2">IE - регистр разрешения IRQ</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0A0h</td><td colspan="2">Порт 2</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>099h</td><td colspan="2">SBUF – буферный регистр приемопередатчика UART</td></tr>
<tr><td>098h</td><td colspan="2">SCON – регистр управления приемопередатчика UART</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>090h</td><td colspan="2">Порт 1</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>08Dh</td><td>TH1</td><td>Старший байт таймера/счётчика 1</td></tr>
<tr><td>08Ch</td><td>TH0</td><td>Старший байт таймера/счётчика 0</td></tr>
<tr><td>08Bh</td><td>TL1</td><td>Младший байт таймера/счётчика 1</td></tr>
<tr><td>08Ah</td><td>TL0</td><td>Младший байт таймера/счётчика 0</td></tr>
<tr><td>089h</td><td colspan="2">TMOD – регистр режимов таймеров</td></tr>
<tr><td>088h</td><td colspan="2">TCON – регистр управления таймерами</td></tr>
<tr><td>087h</td><td colspan="2">PCON – регистр «мощности»</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>083h</td><td>DPH</td><td rowspan="2">DPTR – регистр косвенной адресации</td></tr>
<tr><td>082h</td><td>DPL</td></tr>
<tr><td>081h</td><td colspan="2">SP – указатель стека</td></tr>
<tr><td>080h</td><td colspan="2">Порт 0</td></tr>
<tr><td>07Fh
030h</td><td colspan="3">ОЗУ (80 байт) – доступная память (обычно здесь размещают стек и переменные)</td></tr>
<tr><td>02Fh
020h</td><td colspan="3">Побитно доступная область (128 бит)</td></tr>
<tr><td>01Fh
000h</td><td colspan="3">4 банка по 8 байт каждый (регистровый файл)</td></tr>
</table>

Рис. 1.2 – Карта памяти МК 8051 (256 первых байт)

Для доступа к первым 256 байтам применяется прямая адресация при этом ячейки памяти можно интерпретировать как отдельные регистры. Для доступа к памяти за пределами 256 байт необходимо применять косвенную адресацию: посредством регистра *DPTR* (*DPH*+*DPL*). При этом служебные регистры с адресами 080h-0FFh не могут адресоваться с помощью косвенной адресации. Эти адреса занимают регистры устройств (таймеры, *UART*, АЦП и т.п.), которые называются регистрами специальных функций (*SFR*). При этом для регистров, адреса которых кратны восьми, возможен побитный доступ (это порты, аккумуляторы, регистры управления).

Регистры-аккумуляторы *Acc* и *B* используются для хранения промежуточных результатов арифметических операций. Регистр *B* используется при умножении и делении.

Указатель стека *SP* может адресовать только 256 байт памяти. После включения МК содержимое *SP* равно 07h. При этом загрузка в стек (*push*) производит инкремент регистра *SP*, тогда как в большинстве других МК указатель стека при этом декрементируется.

Регистр *DPTR* разрядностью 16 бит состоит из двух 8-разрядных регистров *DPH* и *DPL* и может адресовать до 64 Кбайт памяти при косвенной (индексной) адресации.

Слово состояния программы *PSW* содержит флаги результата арифметических операций, флаги управления ходом вычислительного процесса. На рис.1.3 представлен формат регистра состояния и приведено назначение его битов (обратите внимание – нет флага нуля).

CY	*AC*	*F*0	*RS*1	*RS*0	*OV*	*F*1	*P*

CY – флаг переноса, устанавливается когда возникает перенос в старший разряд или заем из него;

AC – флаг полупереноса, устанавливается когда при сложении или вычитании результат операции над младшим полубайтом (тетрадой) влияет на результат операции над старшим полубайтом;

*RS*1, *RS*0 – биты, выбирающие текущий 8-байтовый банк регистров (внутри банка регистрам присвоены имена: *R*0-*R*7), используемых для реализации однобайтных команд, которые содержат меньше байтов и выполняются быстрее (при включении МК или сбросе – устанавливается активным 0-й банк);

*RS*1	*RS*0	банк	адрес
0	0	0	00h - 07h
0	1	1	08h - 0Fh
1	0	2	10h - 17h
1	1	3	18h - 1Fh

OV – флаг арифметического переполнения;

P – флаг четности, устанавливается если результат содержит четное количество единиц;

*F*0, *F*1 – резерв.

Рис. 1.3 – Состав регистра состояния (*PSW*) и назначение его битов

Двунаправленный порт *P*0 может служить для передачи информации по шине данных, связывающей МК с внешней памятью или другими устройствами МПС. Через этот же порт передаётся младший байт 16-разрядного адреса (при работе с внешней памятью). Разделение данных от адреса производится с помощью сигнала *ALE*. Старший байт адреса выводится через двунаправленный порт *P*2.

Двунаправленный порт *P*1 применяется как обычный порт ввода-вывода данных (в более поздних модификациях также может быть многофункциональным).

Двунаправленный порт *P*3 многофункциональный. Его линии используются для ввода-вывода следующих управляющих сигналов:

- бит 0 - последовательный ввод *UART* (приёмник *RxD*);
- бит 1 - последовательный вывод *UART* (передатчик *TxD*);
- бит 2 - вход внешнего прерывания от источника 0 (*INT*0);
- бит 3 - вход внешнего прерывания от источника 1 (*INT*1);
- бит 4 - вход таймера/счётчика0 (*T*0);
- бит 5 - вход таймера/счётчика1 (*T*1);
- бит 6 - сигнал записи данных *WR* во внешнюю память данных;
- бит 7 - сигнал чтения данных *RD* из внешней памяти данных.

1.1 ВИДЫ АДРЕСАЦИИ

Непосредственная адресация не требует обращения к регистрам или памяти данных, т.к. операнд содержится непосредственно в команде (т.е. поступает из памяти программ):
add A,#77; добавить к аккумулятору число (константу) 77.

Адресация к регистровым банкам обеспечивает доступ к одному из 8 байтов, размещенных в текущем банке. Регистры банка называются *R*0-*R*7 (рис.1.2), выбор банка осуществляется битами *RS* слова состояния *PSW* (рис.1.3).

Прямая адресация отличается от регистровой тем, что можно адресовать 256 ячеек. При этом на ячейки памяти с адресами 080h-0FFh отображаются регистры МК (регистры *Special Function Register*). Например: команда mov A,88h загружает в аккумулятор содержимое регистра управления таймером (*TCON*), а не содержимое ячейки памяти с таким адресом.

Косвенно-регистровая адресация осуществляется с помощью регистров *R*0 или *R*1 текущего банка, т.е. значение одного из этих регистров интерпретируется как 8-разрядный адрес для обращения к первым 256 ячейкам памяти. При этом такая адресация не позволяет обращаться к регистрам по адресам 080h-0FFh (соответствующее адресное пространство заполнено ячейками ОЗУ) – говорят: нет отображения регистров. Пример: orl A,@R0 – логическое сложение аккумулятора с байтом по адресу, расположенному в регистре *R*0 (т.е. если *R*0=88h, то обращение будет к ячейке ОЗУ, а не к регистру *TCON* как при прямой адресации).

Косвенно-регистровая адресация со смещением образуется при помощи 16-разрядного индексного регистра *DPTR*. Этот вид адресации удобен для доступа к структурированным переменным образующих массив (отображения регистров нет):
mov A,@(DPTR+const).

Для доступа к таблицам (константам) хранящимся в памяти программ можно использовать команды: movc A,@A+DPTR или movc A,@A+PC, которые загружают в аккумулятор значение из ячейки с адресом образуемым содержимым аккумулятора и регистра *DPTR* или *PC* соответственно. Последняя адресует в памяти программ байт, расположенный со смещением относительно адреса текущей команды. Значение смещения содержится в аккумуляторе и вычисляется заранее.

Концептуально, компания *Intel* реализовала три способа адресации, которые позволяют учесть приоритет доступа к определенным массивам данных. В небольшой программе все переменные можно разделить на две группы: те к которым обращение происходит часто и остальные. Поэтому для часто требуемых данных были определены однобайтные команды. Это группа команд работающих в пределах текущего банка регистров (банк это 8 байт). Имеются также двух байтовые команды, обеспечивающие доступ к памяти за пределами текущего банка, но в пределах 256 байт. И существуют трех байтовые команды (всего их 17), обеспечивающие доступ к памяти за пределами 256 байт.

Большинство команд (64 из 111) выполняются за 12 тактов (1 цикл), 45 команд – за 24 такта (2 цикла) и 2 команды – за 48 тактов (4 цикла).

Поэтому при написании программ есть возможность оптимизировать размещение переменных, уменьшая при этом время выполнения программы.

1.2 ПРЕРЫВАНИЯ

Организация прерываний у этого МК достаточно специфична. Рассмотрим карту памяти программ – рис.1.4.

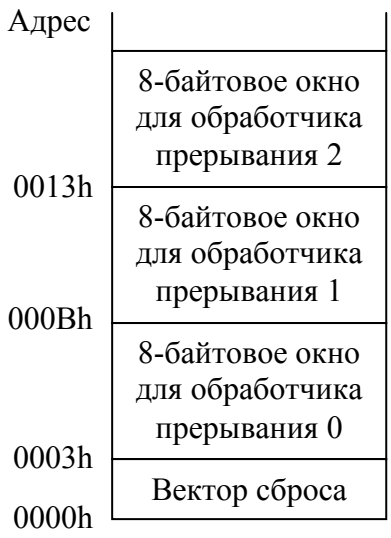


Рис. 1.4 – Карта нижней области памяти программ

По адресу 0 расположен адрес точки программы, с которого процессор начинает выполнение команд при поступлении сигнала сброса или при переполнении сторожевого таймера. При поступлении прерывания с номером 0 ядро МК передает управление на ячейку с адресом 0003h, а при поступлении прерывания 1 – на ячейку с адресом 000Bh и т.д. Таким образом, в МК 8051 обработчик может размещаться непосредственно в «таблице прерываний». Конечно восемь байт это немного, но достаточно, например, для размещения команды передачи управления, либо вызова функции или сброса прерывания, инкрементирования метки реального времени и т.п.

Следует отметить, что при поступлении прерывания МК не сохраняет в стеке регистр флагов. Это связано с тем, что регистр флагов не содержит флага нуля, а код обработчика из арифметических операций обычно содержит только команды инкремента/декремента или установки/сброса битов, которые не влияют на флаги.

В МК могут быть обработаны сигналы от пяти источников прерываний (первые 43 байта памяти программы заняты под «таблицу векторов прерываний»):

- два по переполнению встроенных таймеров/счетчиков;
- два от внешних источников прерывания;
- один от последовательного порта (*UART*).

Прерывания от каждого из указанных источников могут быть независимо друг от друга разрешены или запрещены (замаскированы), причем каждому источнику может быть присвоен приоритет (высокий или низкий). Источник с более высоким приоритетом может прервать программу обслуживания прерывания источника с более низким приоритетом.

Управление системой прерывания обеспечивается с помощью регистров *IE* (адрес 0A8h) и *IP* (адрес 0B8h), которые доступны программно. Назначение бит регистров приведены в нижеследующей таблице.

№ бита	7	6	5	4	3	2	1	0
Регистр *IE*	*EA*	-	-	*ES*	*ET*1	*EX*1	*ET*0	*EX*0
Регистр *IP*	-	-	-	*PS*	*PT*1	*PX*1	*PT*0	*PX*0

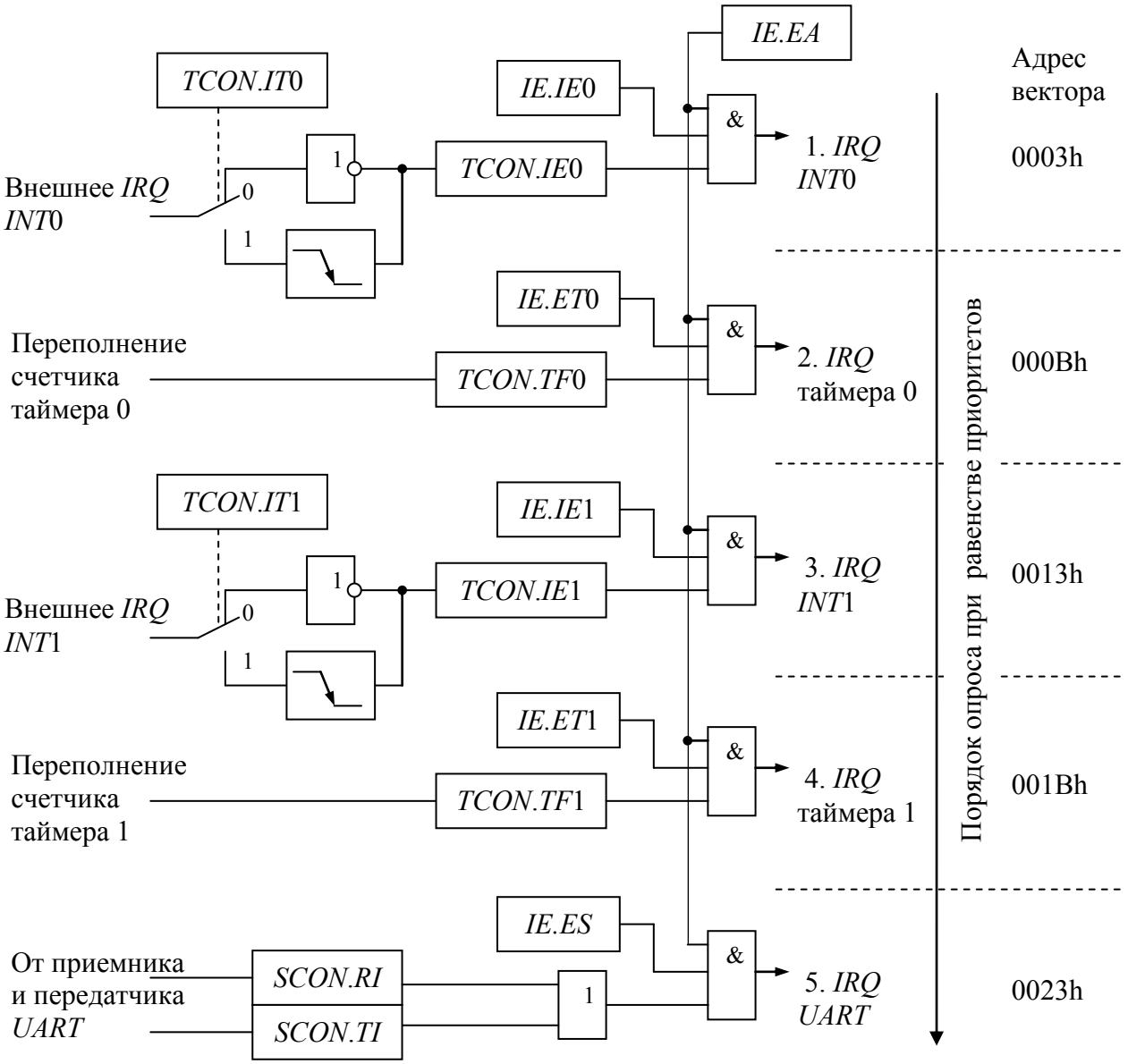
*EX*0, *EX*1 – биты разрешения (маски) внешних прерываний *INT*0 и *INT*1;
*ET*0, *ET*1 – биты разрешения (маски) прерываний от таймеров;
ES – бит разрешения (маски) *UART* (маскируемый сигнал является дизъюнкцией сигналов прерывания от приемника и передатчика);
EA – бит разрешения (маски) всех прерываний независимо от битов 0-4 (=0 – запрет);
PS, *PT*1, *PT*0, *PX*1, *PX*0 – биты управления приоритетами соответствующих прерываний (0 – низкий; 1 – высокий приоритеты).

Обработка прерываний от внешних источников осуществляется при поступлении сигналов на входы *INT*1 (*INT*0), причем формирование флага прерывания *IE*1 (*IE*0) в регистре *TCON* (088h) производится либо по уровню внешнего сигнала, либо по спадающему фронту внешнего сигнала. Соответствующий выбор осуществляется с помощью разрядов *IT*1 (*IT*0) регистра *TCON* (1 – обеспечивается прерывание по срезу сигнала, поступающего на вход порта *P*3, иначе – по уровню сигнала). Сброс этих флагов выполняется аппаратно только в том случае,

если прерывание вызвано срезом внешнего сигнала. При потенциальном способе фиксации сигнала прерывания сброс флагов должна осуществлять процедура обслуживания прерывания, воздействуя на источник прерывания с целью снятия им запроса. Разрешение обработки прерывания от внешних источников осуществляется установкой в единицу разрядов *EX*1 (*EX*0) регистра *IE*. Если прерывания разрешены, то обеспечивается переход к программе обслуживания прерывания, адрес которой определяется в соответствии с жестко зафиксированными векторами прерываний – рис.1.5.

Обработка прерываний по переполнению таймера/счетчика производится в том случае, когда разряды *TF*1 (*TF*0) регистра *TCON* (088h) установлены в единицу и прерывания разрешены (разряды *ET*1 (*ET*0) регистра *IE* установлены в единицу). Значения разрядов *TF*1 (*TF*0) автоматически сбрасываются в нуль при входе в подпрограмму обслуживания прерывания.

Формирование флагов *RI* и *TI* осуществляется по готовности приемника или передатчика последовательного порта (*UART*). Разрешение прерывания от этого источника обеспечивается установкой в единицу разряда *ES* регистра *IE*. Сброс флага осуществляется программно.



IE (0A8h) – регистр разрешения прерываний;
TCON (088h) – регистр управления таймерами.

Рис. 1.5 – Схема формирования запросов прерываний в МК 8051

Механизм приоритетов прерываний предназначен для выбора одного из источников прерываний при одновременном приходе нескольких запросов, а также для решения вопроса о прерывании текущей программы обслуживания прерывания вновь поступившем запросом. Все источники прерываний проверяются на наличие запроса во время фазы *S*5*P*2 каждого машинного цикла (рис.1.6, кроме команды *RETI* и команд обращение к регистрам *IE*, *IP*). В течение следующего машинного цикла анализируются биты регистра приоритетов *IP* и выполняется внутренний выбор (поллинг) более приоритетного запроса.

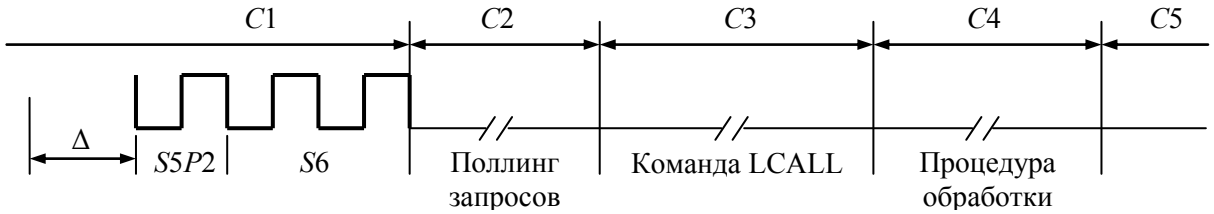


Рис. 1.6 – Циклы вызова процедуры обслуживания прерывания (наиболее быстрый вариант, когда *C*1 является последним машинным циклом команды)

Обработка запроса прерывания начинается после выполнения текущей команды прерываемой программы до конца и сохранения содержимого счетчика команд (адреса возврата) в стеке. Это происходит посредством аппаратно генерируемой команды *LCALL*. В результате в счетчик команд загружается адрес соответствующего обработчика прерываний (рис.1.5). Поскольку восьми байт выделенных под обработчик прерывания обычно мало, то в ячейке памяти по этому адресу размещают команду безусловного перехода к начальному адресу фактической подпрограммы обслуживания прерывания. Такая подпрограмма должна обеспечить сохранность используемых ею регистров и слова состояния (обычно они сохраняются в стеке). При переходе на подпрограмму обработки прерывания автоматически (независимо от регистра *IE*) запрещаются все прерывания с низким (бит регистра *IP* которых сброшен) и равным уровнем приоритета. Таким образом, прерывания с низким приоритетом могут прерываться запросами с высоким уровнем (бит регистра *IP* которых установлен), но обработка высокоприоритетного прерывания не может быть прервана. В конце обработчика прерывания обязательно должна быть записана команда возврата из подпрограммы *RETI*, которая загружает из стека в счетчик команд адрес возврата и разблокирует прерывания.

Минимальное время, необходимое на переход к обработчику прерывания, составляет 38 тактов. Максимальное время, необходимое на переход к обработчику прерывания для случая завершения текущей команды умножения или деления, составляет 86 тактов.

1.3 ПАРАЛЛЕЛЬНЫЙ ВВОД-ВЫВОД

Схемотехническая организация портов 8051 различна. На рис.1.7 показана обобщенная схема вывода параллельного порта 8051. Видно, например, что чтение данных возможно только при записи в этот порт логической единицы. Такая схема не позволяет получить значительных выходных токов, что часто усложняет проектирование устройств на основе этого МК.

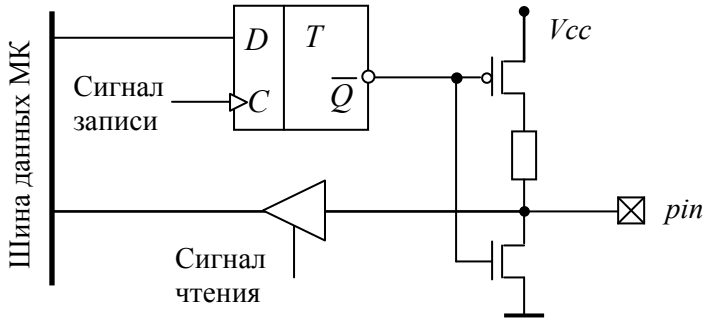


Рис. 1.7 – Обобщенная схема вывода параллельного порта

Помимо основной функции порта (прием/передача данных в параллельном коде) схемотехника портов всех МК (в т.ч. и 8051) предусматривает реализацию специальных функций (входы/выходы таймеров, компараторов, АЦП, ЦАП и служебных сигналов). Таким образом, физическая линия микросхемы (*pin*) разделяется во времени (одновременно используется) несколькими функциональными узлами МК.

Каждый из четырех портов МК 8051 содержит защелку – *D*-триггер, выходной драйвер (представляющий собой последовательное соединение транзисторов, управляемых парафазным сигналом) и входную цепь (типа повторитель с тремя состояниями). По сигналу сброса во все защелки портов записываются единицы – режим ввода данных. Каждая линия любого порта может быть индивидуально настроена как вход или выход. Если к МК подключена внешняя память, то порты *P*0 и *P*2 не могут использоваться, т.к. реализуют функцию мультиплексированной шины адреса/данных.

Выходные драйверы (каскады) портов *P*1-*P*3 выполнены на полевых транзисторах с внутренней нагрузкой, а драйвер порта *P*0 – на транзисторах с открытым стоком.

На рис.1.8 представлена схема порта *P*0.

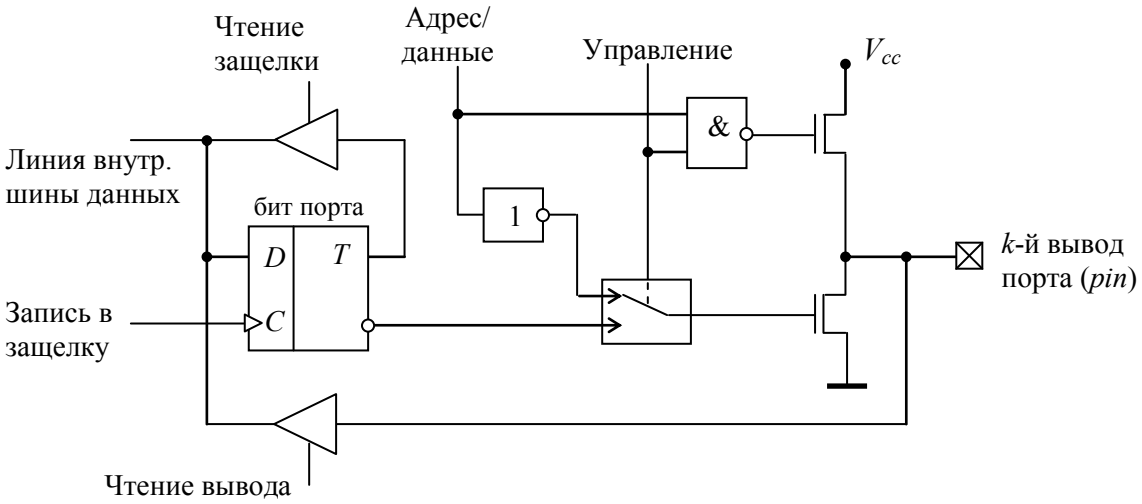


Рис. 1.8 – Схема вывода параллельного порта *P*0

Высокий уровень сигнала "управление" позволяет перевести линии порта в режим, когда к МК подключена внешняя память, при этом сигнал "адрес/данные" выполняет роль линии мультиплексированной шины адреса/данных (младшие восемь бит). При низком уровне сигнала "управление" схема работает в режиме порта (*k*-й бит регистра), при этом транзистор верхнего плеча драйвера закрыт и получается схема открытого стока (при использовании могут

понадобиться внешние подтягивающие резисторы). Для чтения вывода порта необходимо в защелку записать "1", это закроет транзистор нижнего плеча драйвера и обеспечит третье состояния вывода (состояние высокого импеданса).

На рис.1.9 представлена схема порта *P*2, которая отличается от схемы порта *P*0 (рис.1.8) наличием встроенного подтягивающего резистора *R*, поэтому этот порт не может обеспечить состояние высокого импеданса. Высокий уровень сигнала "управление" позволяет перевести линии порта в режим, когда к МК подключена внешняя память, при этом сигнал "адрес/данные" выполняет роль линии мультиплексированной шины адреса/данных (старшие восемь бит). При низком уровне сигнала "управление" схема работает в режиме порта (*k*-й бит регистра). Для чтения вывода порта необходимо в защелку записать "1", это закроет транзистор нижнего плеча драйвера и обеспечит возможность прохождения сигнала с вывода (*pin*) к внутренней шине МК с подтягиванием к уровню напряжения питания.

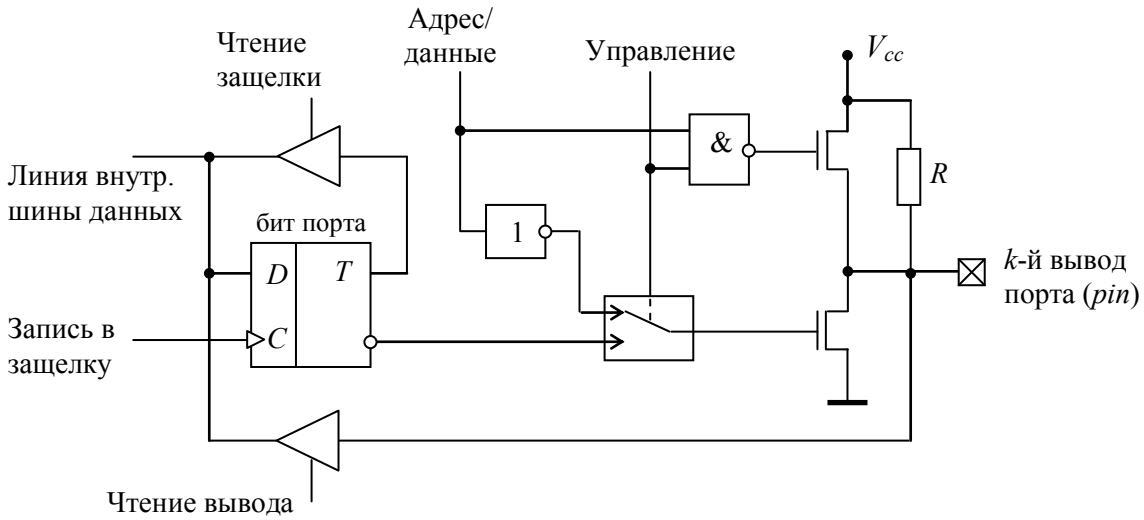


Рис. 1.9 – Схема вывода параллельного порта *P*2

На рис.1.10 представлена схема порта *P*1. Для чтения вывода порта необходимо в защелку записать "1", это закроет транзистор нижнего плеча драйвера и обеспечит возможность прохождения сигнала с вывода (*pin*) к внутренней шине МК с подтягиванием к уровню напряжения питания.

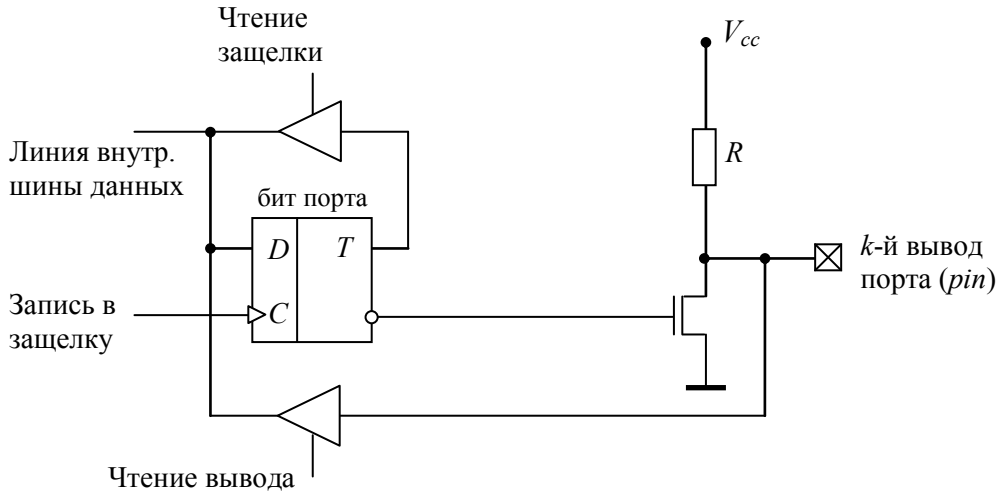


Рис. 1.10 – Схема вывода параллельного порта *P*1

На рис.1.11 представлена схема порта *P*3. Каждая линия порта реализует специальную функцию (альтернативное назначение). Для включения альтернативной функции линии порта необходимо записать в защелку "1", а для его чтения еще необходимо, чтобы сигнал "альтернативный сигнал выхода" был высокого уровня.

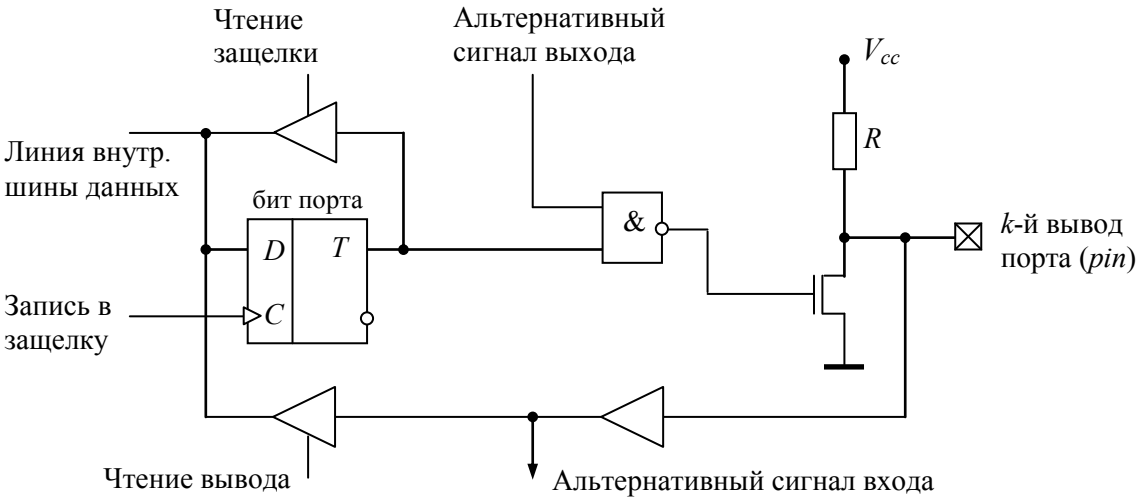


Рис. 1.11 – Схема вывода параллельного порта *P*3

Обращение к портам происходит с помощью команд, оперирующих с байтом, отдельным битом или произвольной группой бит. При этом в одной команде порт может быть и источником и приемником данных, в этом случае устройство управления МК автоматически включает режим "чтение-модификация-запись". Этот режим предполагает чтение защелки (а не линии порта) изменение значения и запись результата обратно в защелку в цикле одной команды (отметим, что содержимое защелки и состояние внешней линии порта могут не совпадать).

1.4 ТАЙМЕРЫ

В базовых моделях МК 8051 имеются два программируемых 16-разрядных таймера/счетчика. На рис.1.12 представлена упрощенная структура таймера/счетчика.

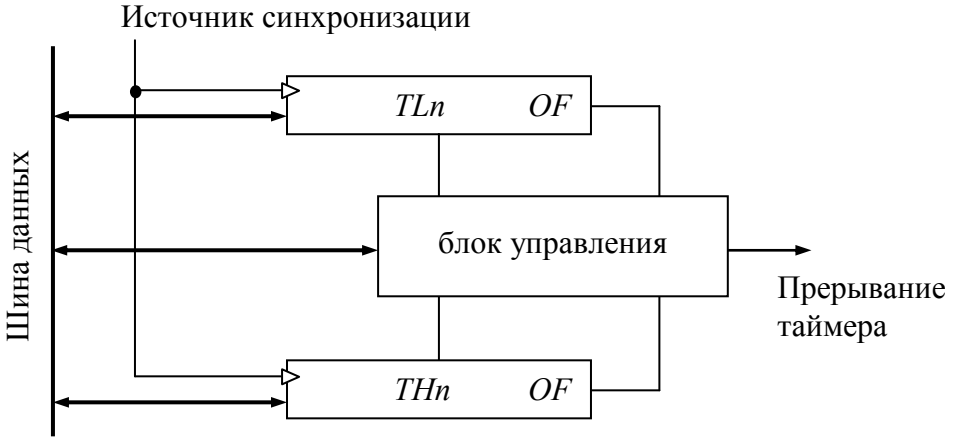


Рис. 1.12 – Упрощенная структура таймера МК 8051

В этой структуре *TLn* и *THn* являются 8-разрядными счетчиками-регистрами и представляют младшие и старшие разряды таймера, где *n* – номер таймера (0, 1 и более – для старших моделей). В качестве источника синхронизации служат синхроимпульсы МК, поступающие с *prescaler* (входного делителя). Кроме того, таймер может синхронизироваться от внешнего источника (входы *Tn*). Опрос внешнего входного сигнала *Tn* осуществляется в фазе *S*5*P*2 каждого машинного цикла (аналогично опросу входов внешних прерываний – рис.10). Инкремент счетчика происходит при фиксации изменения входного сигнала с высокого уровня на низкий (т.е. в предыдущем цикле была зафиксирована "1", а в следующем – "0"). Новое значение счетчика (инкрементированное) будет сформировано в середине цикла следующего за циклом, в котором обнаружен переход с "1" на "0" входа *Tn*. Так как для распознавания периода требуется два машинных цикла (12 тактов каждый), то максимальная частота переключения входного сигнала *Tn* ограничена величиной 1/24 от тактовой частоты генератора МК. Длительность входных сигналов *Tn* должна быть не меньше длительности одного цикла МК. Например, при тактовой частоте МК F_{clk}=12 МГц диапазон изменения интервала времени таймера составляет 1-65536 мкс, а частота переключений на входе *Tn* – не более 500 кГц.

Такая организация таймера позволяет реализовывать часто востребованные функции:

- реализация меток реального времени;
- измерение числа событий за определенный промежуток времени;
- измерение длительности импульса;
- измерение частоты сигнала;
- реализация (программно-аппаратная) ШИМ сигнала.

В МК 8051 существует четыре режима работы таймера:

1) Режим 0 – таймер конфигурируется как 13-разрядный счетчик.
2) Режим 1 – таймер конфигурируется как 16-разрядный счетчик.
3) Режим 2 – применяется для задания скорости последовательного порта. Когда содержимое счетчика *TLn* переполняется, в него загружается содержимое счетчика *THn*, который работает как регистр (не изменяется в ходе счета, но доступен программно). И счет возобновляется, а сигнал переполнения используется для задания скорости обмена последовательного порта (*UART*).
4) Режим 3 – *TLn* и *THn* работают как два независимых 8-разрядных счетчика (этот режим обеспечивается только таймером 0). Счетчик *TL*0 переключается внешним сигналом, а *TH*0 – внутренними тактовыми сигналами (реализуется тахометр).

Для управления таймерами/счетчиками в МК предусмотрены два регистра специальных функций: *TMOD* (089h) и *TCON* (088h).

Назначение бит регистра *TMOD* приводится в нижеследующей таблице.

Номер бита	7	6	5	4	3	2	1	0
Наименование	*Gate*	*C/T*	*M*1	*M*0	*Gate*	*C/T*	*M*1	*M*0
	Таймер/счетчик 1				Таймер/счетчик 0			

Gate – управление блокировкой	=0 – таймер разрешен, если бит *TCON.TRn* установлен =1 – таймер разрешен, если на входе внешнего прерывания *INTn* высокий уровень и бит *TCON.TRn* установлен
C/T – управление тактированием	=0 – таймер работает от внутреннего генератора с делителем на 12 =1 – таймер работает от сигнала на внешнем входе *Tn*
*M*1, *M*0 – выбор режима работы	0 0 – режим 0 (рис.1.13А), 13-бит счетчик 0 1 – режим 1 (рис.1.13Б), 16-бит счетчик 1 0 – режим 2 (рис.1.14), 8-бит счетчик с перезагрузкой 1 1 – режим 3 (рис.1.15), два независимых 8-бит счетчика (таймер 0)

Назначение бит регистра *TCON* приводится в нижеследующей таблице.

Номер бита	7	6	5	4	3	2	1	0
Наименование	*TF*1	*TR*1	*TF*0	*TR*0	*IE*1	*IT*1	*IE*0	*IT*0
	Таймер/счетчик 1		Таймер/счетчик 0		см. раздел 1.2, внешние прерывания			

TFn – флаг переполнения счетчика
TRn – сигнал разрешения работы счетчика (=0 – не считает, запрещен)

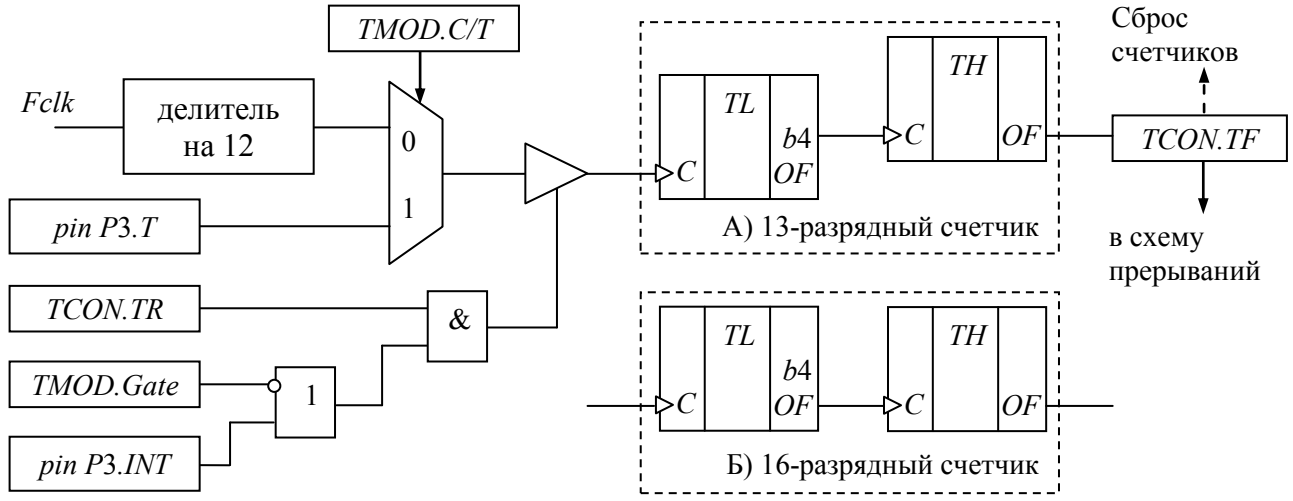


Рис. 1.13 – Схема таймера/счетчика в режимах 0 и 1 (варианты А и Б соответственно)

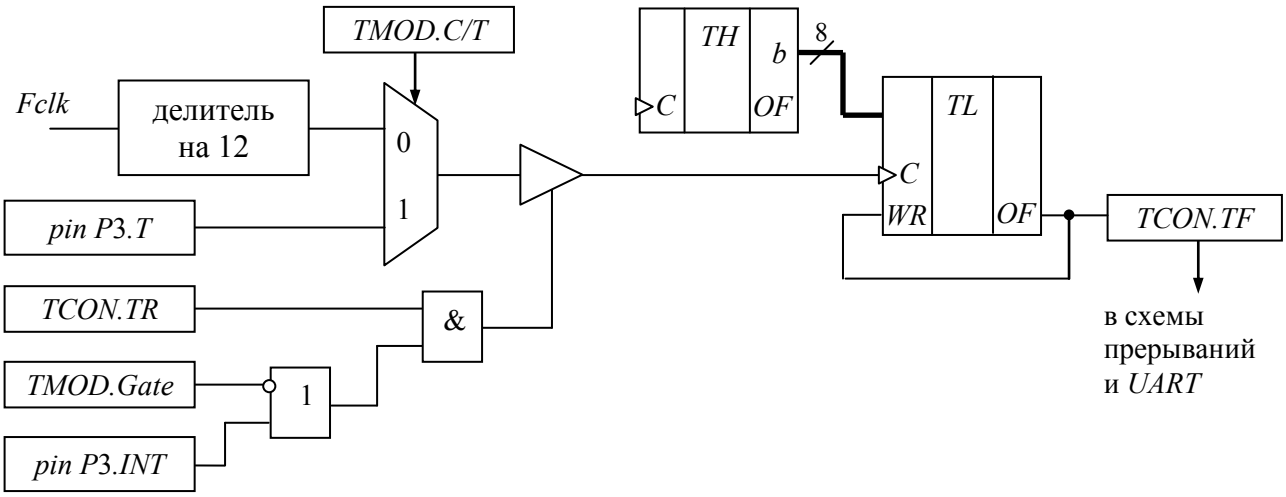


Рис. 1.14 – Схема таймера/счетчика в режиме 2

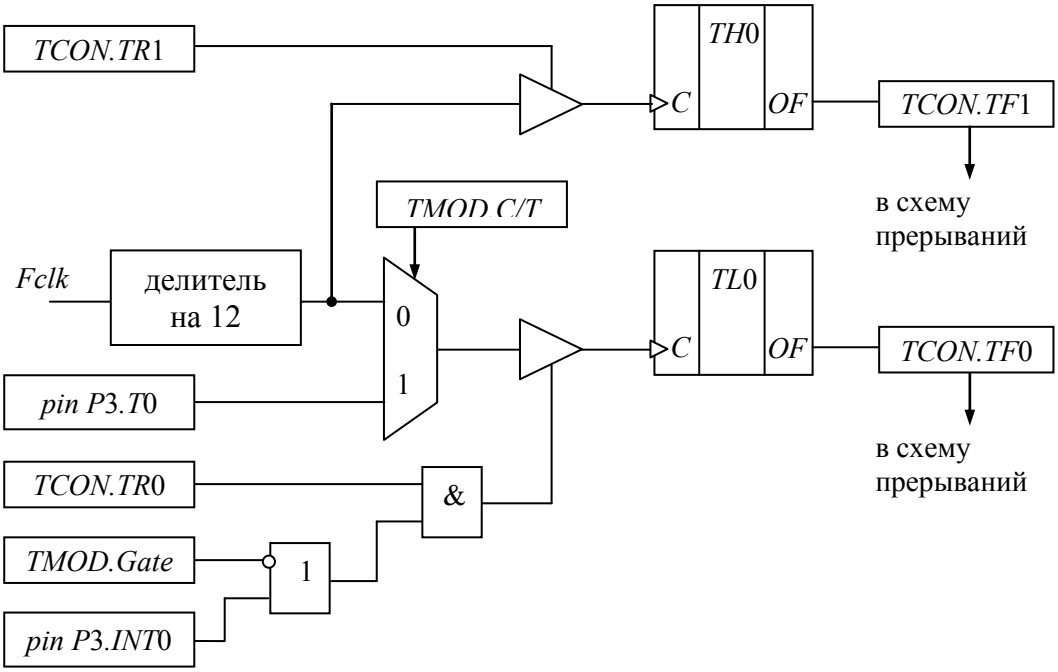


Рис. 1.15 – Схема таймера/счетчика 0 в режиме 3

(*) При включении у таймера/счетчика0 режима 3 таймер/счетчик1 лишается своего бита управления *TCON.TR*1 и флага прерывания *TCON.TF*1. Однако сохраняет свою работоспособность без прерываний. Поэтому в режимах 0, 1, 2 при *TMOD.Gate*1=0 всегда включен и при переполнении в режимах 0 и 1 обнуляется, а в режиме 2 перезагружается, не устанавливая флага прерываний *TCON.TF*1. Управление от входов *P*3.*INT*1, *P*3.*T*1 и применение битов управления *TMOD.C/T*1, *TMOD.Gate*1 для таймера/счетчика1 не зависят от режима таймера/счетчика 0.

1.5 ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД-ВЫВОД

Последовательный порт (универсальный приемопередатчик – *UART*) 8051 способен функционировать как в асинхронном режиме, так и в синхронном. При этом передача данных происходит младшими битами вперед. Упрощенная структурная схема последовательного порта представлена на рис.1.16.

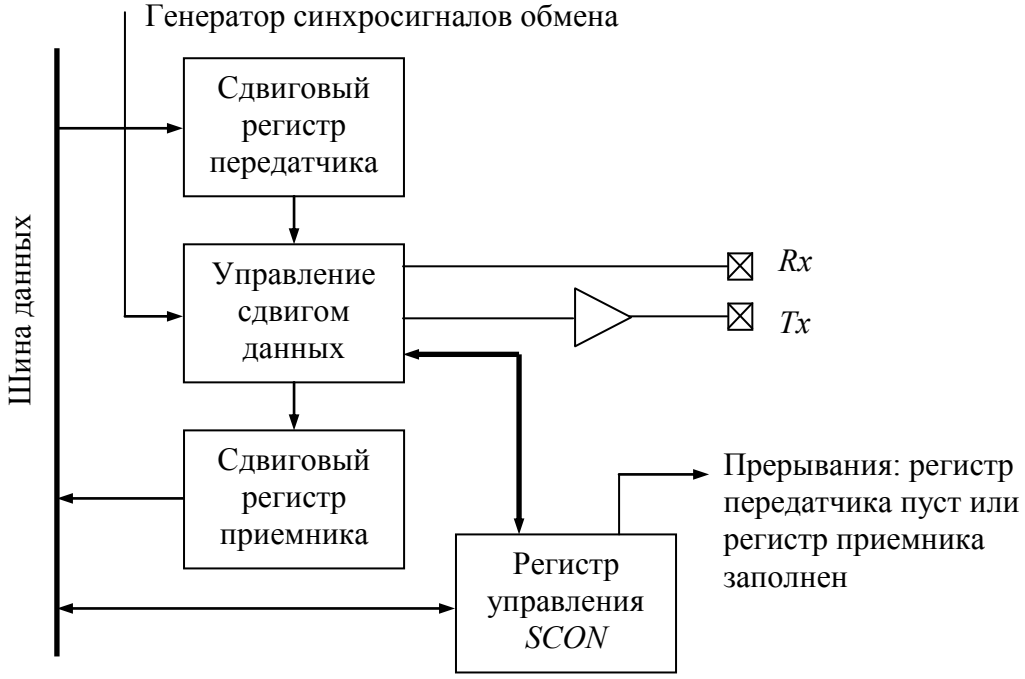


Рис. 1.16 – Структурная схема последовательного порта

Для управления работой интерфейса в МК предусмотрен регистр *SCON* (098h), назначение бит которого приводится в нижеследующей таблице.

Бит		Назначение
0	*RI*	Флаг прерывания приемника (активен, если принят байт данных)
1	*TI*	Флаг прерывания передатчика (активен, если байт передан в линию)
2	*RB*8	Прием 9-го бита для режимов 2 и 3 или стоп-бита для режима 1
3	*TB*8	Передача 9-го бита для режимов 2 и 3
4	*REN*	Разрешение/запрет приема данных
5	SM2	Устанавливается программно для запрета приема сообщений, в которых девятый бит имеет значение 0 (режимы 1, 2 и 3)
6	*SM*1	Режим работы *UART* (*SM*1 *SM*0): 0 0 - режим 0 (синхронный обмен) 0 1 - режим 1 (асинхронный обмен, 8 бит, изменяемая скорость передачи) 1 0 - режим 2 (асинхронный обмен, 9 бит, фиксированная скорость передачи) 1 1 - режим 3 (асинхронный обмен, 9 бит, изменяемая скорость передачи)
7	*SM*0	

Режим 0 (синхронный обмен). Данные посылаются и принимаются по линии *Rx* (полудуплексный режим), а синхросигналы выдаются по линии *Tx*. В этом режиме МК может быть только ведущим устройством. Передача данных инициируется записью слова в регистр данных порта и ведется по переднему фронту тактового сигнала. После приема слова может быть реализовано аппаратное прерывание. Максимальная скорость передачи бит данных определяется частотой тактирования МК деленной на 12.

Режим 1 (асинхронный обмен). Данные выдаются через линию *Tx*, а принимаются через *Rx* (полнодуплексный режим). Пакет передачи представляет собой: старт-бит, 8 бит данных,

стоп-бит. Причем при приеме данных стоп-бит заносится в разряд *RB*8 регистра *SCON*. Скорость работы порта задается с помощью таймера/счетчика1.

Режим 2 (асинхронный обмен). Данные выдаются через линию *Tx*, а принимаются через *Rx* (полнодуплексный режим). Пакет передачи представляет собой: старт-бит, 8 бит данных, программируемый бит, стоп-бит. При передаче и приеме программируемый бит связан с битами *TB*8 и *RB*8 регистра *SCON* соответственно. Его можно интерпретировать как 9-й бит данных или бит четности. Стоповый бит в отличие от режима 1 теряется. Скорость передачи в этом режиме равна либо 1/32, либо 1/64 – определяется управляющим битом *SMOD* регистра *PCON* ("0" – 1/64; "1" – 1/32).

Режим 3 – аналогичен режиму 2, но скорость определяется таймером/счетчиком1.

Во всех режимах передача данных инициируется записью байта в регистр передатчика *SBUF*. Это означает, что для достижения максимальной скорости обмена МК должен отслеживать момент окончания передачи байта для загрузки следующего байта. Этот момент можно контролировать с помощью прерывания.

Во всех режимах флаг прерывания передатчика *SCON.TI* устанавливается аппаратно в конце передачи стоп-бита.

Флаг приемника *SCON.RI* устанавливается аппаратно в конце приема 8-го бита данных в режиме 0 и в середине приема стоп-бита в режимах 1,2,3.

Подпрограмма обработки прерывания *UART* должна сбрасывать соответствующие флаги *RI* и *TI* в регистре *SCON*.

В режимах (1 и 3), где источником тактирования является таймер 1, бит *SMOD* регистра *PCON* также влияет на частоту работы последовательного порта – делит частоту сигнала переполнения таймера/счетчика1 на 32 (*SMOD*=1) или на 64 (*SMOD*=0).

На рис. 1.17а-в представлены для ознакомления временные диаграмма работы узла в всех режимах.

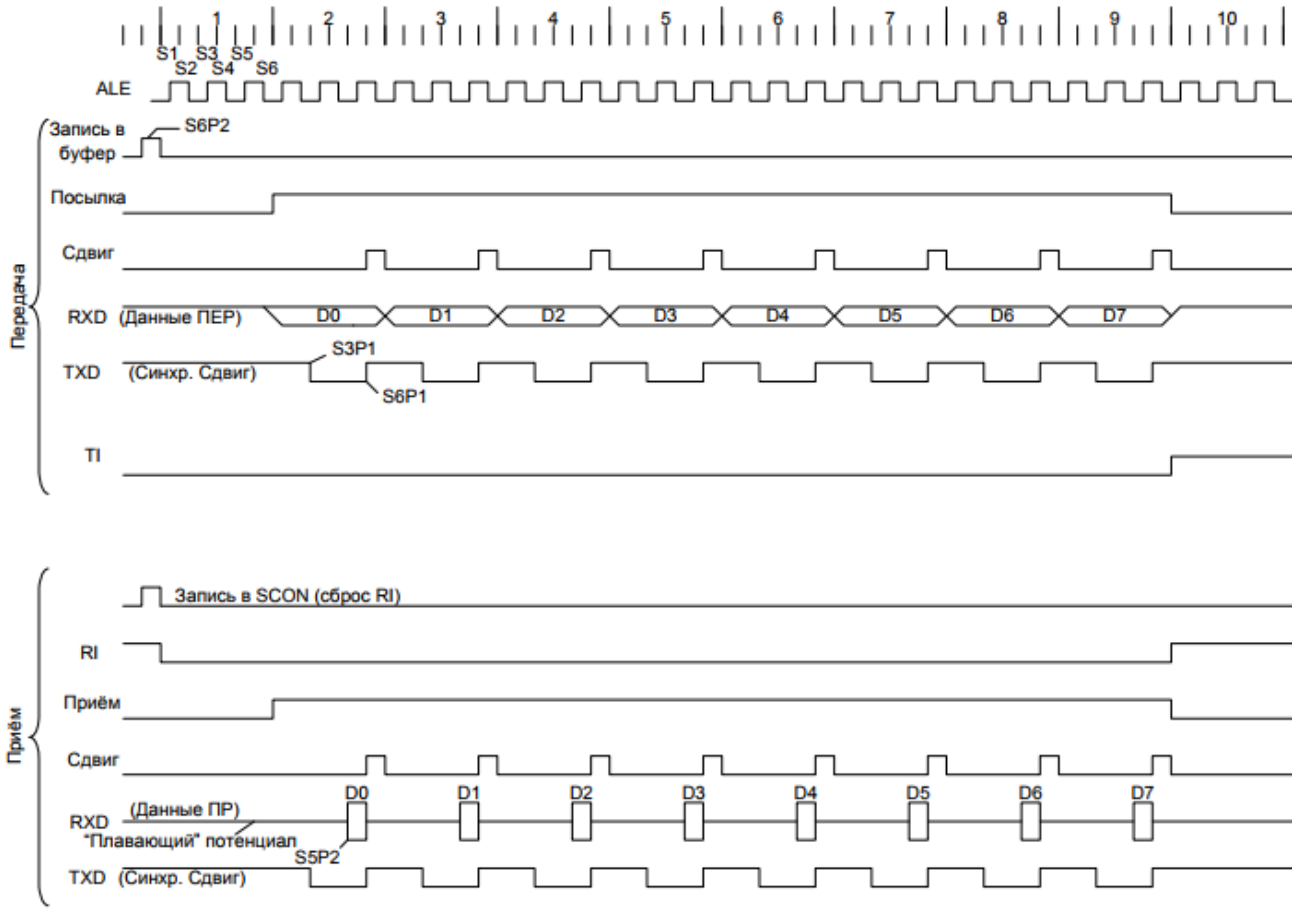


Рис. 1.17а - Временная диаграмма работы *UART* в режиме 0

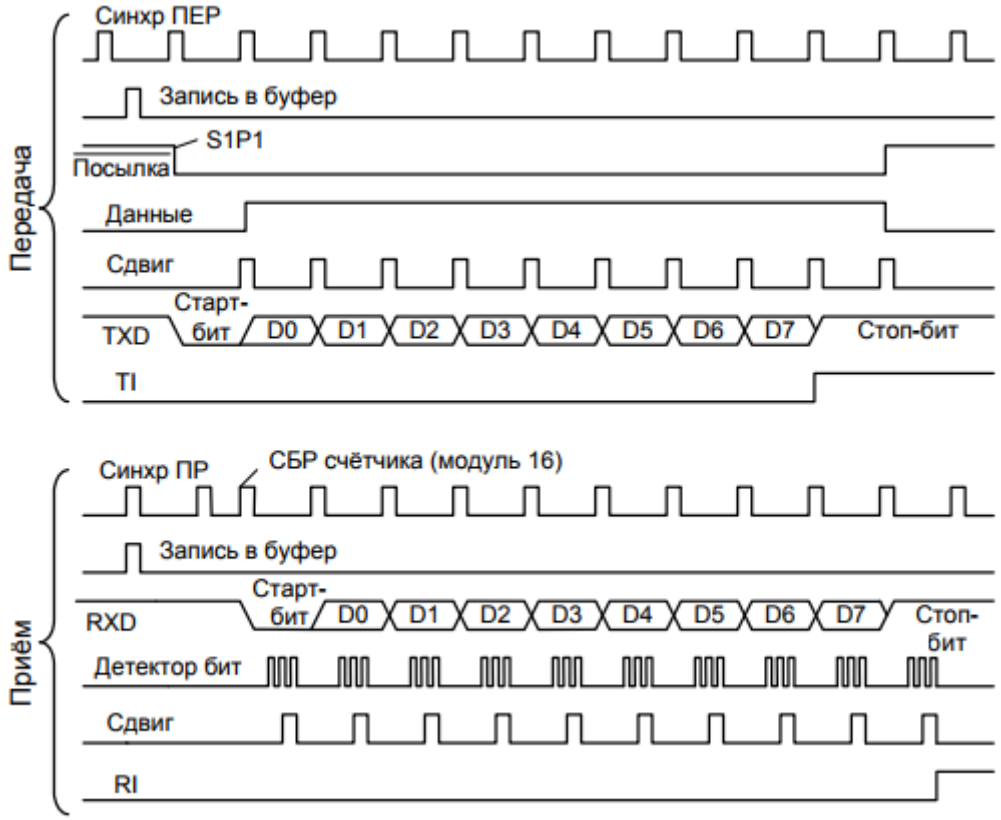


Рис. 1.17б - Временная диаграмма работы *UART* в режиме 1

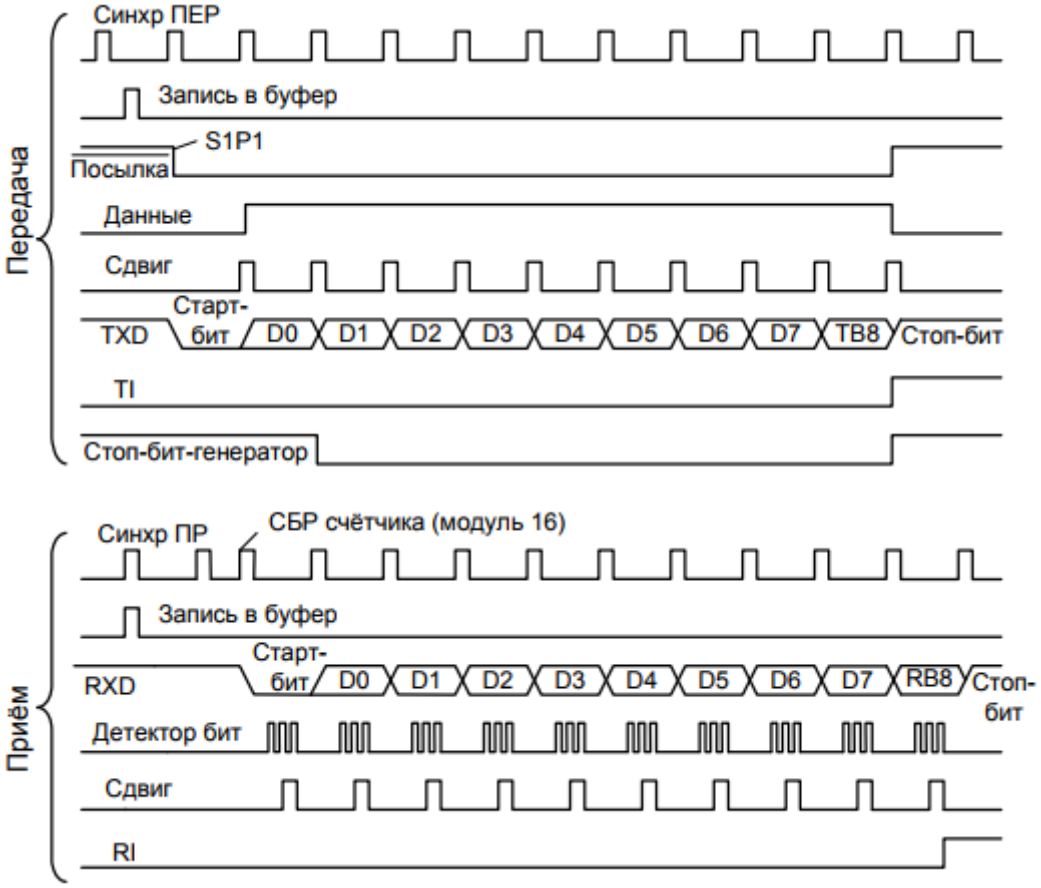


Рис. 1.17в - Временная диаграмма работы *UART* в режимах 2 и 3

1.6 ЭНЕРГОПОТРЕБЛЕНИЕ МК

Одной из ключевых возможностей любого МК является способность снижать свое энергопотребление, отключая часть внутренних узлов и/или понижать частоту тактовых импульсов. Не исключение и МК i8051. Он предоставляет два режима пониженного энергопотребления.

1. Режим холостого хода. В этом режиме блоки прерываний, таймеров и последовательного порта продолжают тактироваться, все регистры сохраняют свое значение, все линии портов ввода-вывода сохраняют свое активное состояние (удерживают логический уровень, который был на момент перехода в режим XX). Однако ядро отключается от тактового генератора. На выводах *ALE*, *PSEN* устанавливается высокий уровень. Режим активируется при установке бита *IDL* в регистре *PCON*. Выход из режима осуществляется по сигналу сброса или при поступлении любого сигнала прерывания. В последнем случае после выполнения процедуры обработки прерывания выполнение программы продолжается с команды, следующей за командой установившей бит *PCON.IDL*.
2. Режим выключенного питания (*Power Down Mode*). В этом режиме не тактируются: ядро, таймеры, *UART*, система прерываний. Однако состояние внутренней памяти (в т.ч. регистры) и линий портов ввода-вывода остаются неизменными. На выводах *ALE*, *PSEN* устанавливается низкий уровень. Режим включается установкой бита *PD* регистра *PCON*. При этом напряжение питание с вывода V_{cc} (*pin* 40) можно снизить или отключить, но необходимо к выводу *Reset* подключить резервное питание, обеспечивающее сохранность памяти и регистров. Выход из режима осуществляется только по сигналу сброса.

(*) При одновременной установке битов *IDL* и *PD* регистра *PCON* последний имеет приоритет – будет включен режим *PDM*.

Регистр *PCON* (87h) и назначение его разрядов приводится в нижеследующей таблице.

Бит		Назначение
0	*IDL*	Бит холостого хода. Установка в «1» переводит МК в режим пониженного энергопотребления – режим холостого хода.
1	*PD*	Бит пониженной мощности. При установке в «1» МК переходит в *power down mode*.
2	*GF*0	Флаги общего назначения – могут использоваться по усмотрению пользователя.
3	*GF*1	
4	-	Резерв. Не рекомендуется записывать в эти биты «1».
5	-	
6	-	
7	*SMOD*	Удвоенная скорость передачи. Установка в «1» удваивает скорость *UART* – см. раздел 1.5.

При сбросе *PCON*=0xxx0000. Биты 3-0 регистра поддерживаются не всеми моделями i8051.

1.7 СТОРОЖЕВОЙ ТАЙМЕР

На рис.1.18 представлена структурная схема сторожевого таймера МК 8051.

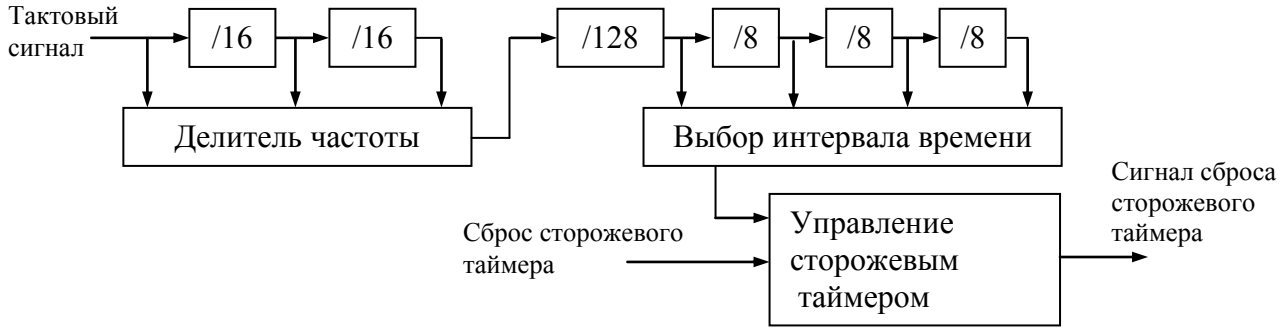


Рис. 1.18 – Структурная схема сторожевого таймера 8051

Управляемый программно делитель и блок выбора интервала времени позволяет запрограммировать сторожевой таймер на отсчет интервалов длительностью от 10 мс до 24 мин при тактовой частоте 12 МГц.

Для сброса сторожевого таймера, т.е. для предотвращения сброса МК необходимо выполнить следующую последовательность команд:

```
mov 0C7, #0AAh;       послать контрольные символы
mov 0C7, #055h
mov WDCON, #002h;     сбросить сторожевой таймер
```

1.8 НАЗНАЧЕНИЕ ВЫВОДОВ МК 8051

Классический i8051 (*MCS*-51) и отечественный аналог КМ1816ВЕ51 выполнены на основе *n*-МОП технологии и выпускались в 40-выводном корпусе – рис.1.19.

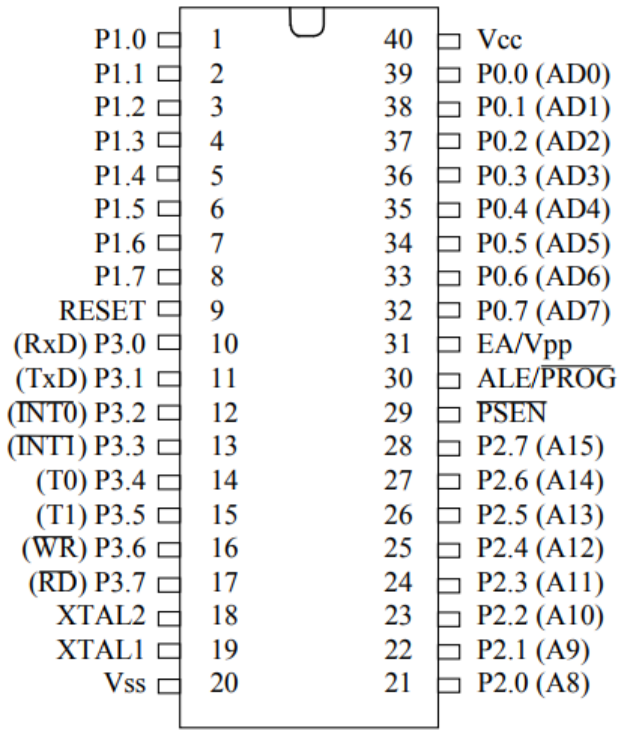


Рис. 1.19 – Цоколевка микросхемы *MSC*-51

Назначение выводов.

Vss - потенциал общего провода "земля"

Vcc - напряжение питания +5 В

*XTAL*1, *XTAL*2 - выводы подключения кварцевого резонатора

Reset (*RST*) - вход общего сброса МК (не влияет на ячейки памяти данных)

PSEN - строб внешней памяти программ, выдается только при обращении к внешнему ПЗУ

ALE - строб адреса внешней памяти

EA - отключение внутренней памяти программ, низкий уровень заставляет МК выполнять программу из внешнего ПЗУ игнорируя внутреннее

*P*0 - 8-разрядный порт ввода-вывода, при работе с внешними ОЗУ и ПЗУ эти линии мультиплексированы с адресом (младшие линии адреса)

*P*1 - 8-разрядный порт ввода-вывода, каждый разряд порта может быть независимо запрограммирован на вход или выход

*P*2 - 8-разрядный порт ввода-вывода, аналогичный P0, при работе с внешними ОЗУ и ПЗУ эти линии мультиплексированы с адресом (старшие линии адреса)

*P*3 - 8-разрядный порт ввода-вывода, мультиплексирован с узлами специальных функций (*UART*, таймеры, *IRQ*, стробы внешней памяти).

На рис.1.20 представлена структурная схема контроллера *MCS*-51.

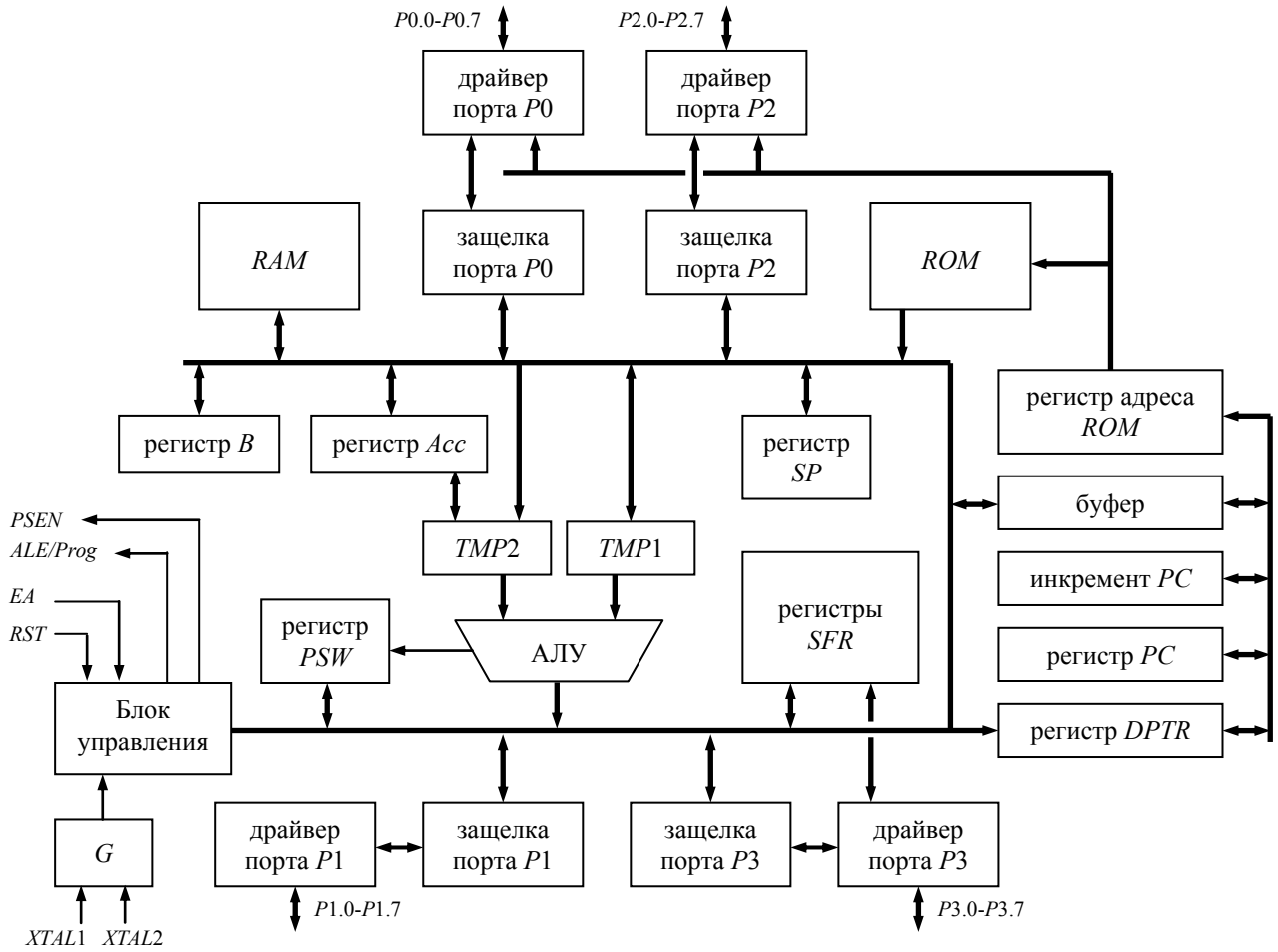


Рис. 1.20 – Структурная схема *MSC*-51

1.9 СИСТЕМА КОМАНД 8051

Система команд МК 8051 содержит 111 базовых команд, которые по функциональному признаку могут быть разделены на пять групп:

1) Команды передачи данных;
2) Арифметические операции;
3) Логические операции;
4) Операции с битами;
5) Команды передачи управления.

Все команды с точки зрения кодирования представляются либо однобайтовыми, либо двухбайтовыми, либо трехбайтовыми – в соответствии с этим признаком определяют 13 типов команд – рис. 1.21, где

КОП – код операции;
#d – 8-разрядная численная константа (хранится в памяти программ);
#d16 – 16-разрядная численная константа (хранится в памяти программ);
ad (ad1, ad2) – 8-разрядный адрес (хранится в памяти программ);
ad16 – 16-разрядный адрес (хранится в памяти программ);
bit – номер бита (хранится в памяти программ);
rel – относительное смещение (хранится в памяти программ).

	1-й байт $d_7 \dots d_0$	2-й байт $d_7 \dots d_0$	3-й байт $d_7 \dots d_0$
1	КОП		
2	КОП	#d	
3	КОП	ad	
4	КОП	bit	
5	КОП	rel	
6	$a_{10}a_9a_8$ КОП	$a_7 \dots a_0$	
7	КОП	ad	#d
8	КОП	ad	rel
9	КОП	ad1	ad2
10	КОП	#d	rel
11	КОП	bit	rel
12	КОП	ad16(high)	ad16(low)
13	КОП	#d16(high)	#d16(low)

Рис. 1.21 – Типы команд МК 8051

Определение и ассемблерная мнемоника команд, их тип в соответствии с рис. 1.21 (Т - тип команды: от 1 до 13), число байтов в командах (Б), а также продолжительность исполнения команд в циклах (Ц) даны в нижеследующей таблице.

Команды передачи данных	Т Б Ц	Операция

Пересылка аккумулятор - регистр (*n*=0÷7)
MOV A, Rn
MOV Rn, A
	1 1 1	
		(A)←(Rn)
		(Rn)←(A)

Пересылка аккумулятор - прямоадресуемый байт
MOV A, ad
MOV ad, A
	3 2 1	
		(A)←(ad)
		(ad)←(A)

Пересылка аккумулятор - байт из ПД (*i*=0,1)
MOV A, @Ri
MOV @Ri, A
	1 1 1	
		(A)←((Ri))
		((Ri))←(A)

Загрузка в аккумулятор константы MOV A, #d	2 2 1	 (A)←#d
Загрузка в регистр константы MOV Rn, #d	2 2 1	 (Rn)←#d

Пересылка регистр - прямоадресуемый байт
MOV Rn, ad
MOV ad, Rn
	3 2 2	
		(Rn)←(ad)
		(ad)←(Rn)

Пересылка байта ПД по прямому адресу
MOV ad, @Ri
MOV @Ri, ad
	3 2 2	
		(ad)←((Ri))
		((Ri))←(ad)

Пересылка прямоадресумая "память - память" MOV add, ads	9 3 2	 (add)←(ads)
Пересылка по прямому адресу константы MOV ad, #d	7 3 2	 (ad)←#d
Пересылка в ПД константы MOV @Ri, #d	2 2 1	 ((Ri))←#d
Загрузка указателя данных MOV DPTR, #d16	13 3 2	 (DPTR)←#d16

Пересылка в аккумулятор байта из ПП
MOVC A, @A+DPTR
MOVC A, @A+PC
	1 1 2	
		(A)←((A) + (DPTR))
		(PC)←(PC)+1, (A)←((A)+(PC))

Пересылка аккумулятор - байт из внешней ПД
MOVX A, @Ri
MOVX @Ri, A
MOVX A, @DPTR
MOVX @DPTR, A
	1 1 2	
		(A)←((Ri))
		((Ri))←(A)
		(A)←((DPTR))
		((DPTR))←(A)

Стек: загрузка/выгрузка PUSH ad POP ad	3 2 2	 (SP)←(SP)+1, ((SP))←(ad) (ad)←(SP), (SP)←(SP)–1
Обмен аккумулятора с регистром XCH A, Rn	1 1 1	 (A)↔(Rn)
Обмен аккумулятора с прямоадресуемым байтом XCH A, ad	3 2 1	 (A)↔(ad)
Обмен аккумулятора с байтом из ПД XCH A, @Ri	1 1 1	 (A)↔((Ri))
Обмен младших тетрад аккумулятора и байта ПД XCHD A, @Ri	1 1 1	 $(A_{0...3})\leftrightarrow((Ri)_{0...3})$

Арифметические команды	Т Б Ц	Операция

Сложение аккумулятора с константой
ADD A, #d
ADDC A, #d
	2 2 1	
		(A)←(A)+#d
		(A)←(A)+#d+(C)

Сложение аккумулятора с регистром (*n*=0÷7)
ADD A, Rn
ADDC A, Rn
	1 1 1	
		(A)←(A)+(Rn)
		(A)←(A)+(Rn)+(C)

Сложение аккумул. с прямоадресуемым байтом
ADD A, ad
ADDC A, ad
	3 2 1	
		(A)←(A)+(ad)
		(A)←(A)+(ad)+(C)

Сложение аккумулятора с байтом из ПД (*i*=0,1)
ADD A, @Ri
ADDC A, @Ri
	1 1 1	
		(A)←(A)+((Ri))
		(A)←(A)+((Ri))+(C)

Десятичная коррекция аккумулятора DA A	1 1 1	Если ($A_{0...3}$)>9 или ((AC)=1), то ($A_{0...3}$)←($A_{0...3}$) + 6, затем если ($A_{4...7}$) >9 или ((C)=1), то ($A_{4...7}$)←($A_{4...7}$) + 6
Вычитание из аккумулятора регистра и заёма SUBB A, Rn	1 1 1	 (A)←(A)–(C)–(Rn)
Вычитание из аккумулятора прямоадресуемого байта и заема SUBB A, ad	3 2 1	 (A)←(A)–(C)–((ad))
Вычитание из аккумулятора байта ПД и заема SUBB A, @Ri	1 1 1	 (A)←(A)–(C)–((Ri))
Вычитание из аккумулятора константы и заема SUBB A, #d	2 2 1	 (A)←(A)–(C)–#d
Инкремент INC A INC Rn INC ad INC @Ri INC DPTR	 1 1 1 1 1 1 3 2 1 1 1 1 1 1 2	 (A)←(A)+1 (Rn)←(Rn)+1 (ad)←(ad)+1 ((Ri))←((Ri))+1 (DPTR)←(DPTR)+1
Декремент DEC A DEC Rn DEC ad DEC @Ri	 1 1 1 1 1 1 3 2 1 1 1 1	 (A)←(A)–1 (Rn)←(Rn)–1 (ad)←(ad)–1 ((Ri))←((Ri))–1
Умножение/деление аккумулятора на регистр *B* MUL AB DIV AB	1 1 4	 (B)(A)←(A)*(B) (B).(A)←(A)/(B), без знака

Логические команды	Т Б Ц	Операция
Логическое умножение		
ANL A, #d	2 2 1	(A)←(A) AND #d
ANL ad, #d	7 3 2	(ad)←(ad) AND #d
ANL A, Rn	1 1 1	(A)←(A) AND (Rn)
ANL A, ad	3 2 1	(A)←(A) AND (ad)
ANL ad, A	3 2 1	(ad)←(ad) AND (A)
ANL A, @Ri	1 1 1	(A)←(A) AND ((Ri))
Логическое сложение		
ORL A, #d	2 2 1	(A)←(A) OR #d
ORL ad, #d	7 3 2	(ad)←(ad) OR #d
ORL A, Rn	1 1 1	(A)←(A) OR (Rn)
ORL A, ad	3 2 1	(A)←(A) OR (ad)
ORL ad, A	3 2 1	(ad)←(ad) OR (A)
ORL A, @Ri	1 1 1	(A)←(A) OR ((Ri))
Логическое исключающее сложение		
XRL A, #d	2 2 1	(A)←(A) XOR #d
XRL ad, #d	7 3 2	(ad)←(ad) XOR #d
XRL A, Rn	1 1 1	(A)←(A) XOR (Rn)
XRL A, ad	3 2 1	(A)←(A) XOR (ad)
XRL ad, A	3 2 1	(ad)←(ad) XOR (A)
XRL A, @Ri	1 1 1	(A)←(A) XOR ((Ri))
Сброс аккумулятора CLR A	1 1 1	(A)←0
Инверсия аккумулятора CPL A	1 1 1	(A)←NOT(A)
Сдвиг аккумулятора влево циклический	1 1 1	
RL A		$(A_{n+1})\leftarrow(A_n)$, n=0÷6, $(A_0)\leftarrow(A_7)$
RLC A		$(A_{n+1})\leftarrow(A_n)$, n=0÷6, $(A_0)\leftarrow$(C), (C)$\leftarrow(A_7)$
Сдвиг аккумулятора вправо циклический	1 1 1	
RR A		$(A_n)\leftarrow(A_{n+1})$, n=0÷6, $(A_7)\leftarrow(A_0)$
RRC A		$(A_n)\leftarrow(A_{n+1})$, n=0÷6, $(A_7)\leftarrow$(C), (C)$\leftarrow(A_0)$
Обмен местами тетрад в аккумуляторе	1 1 1	
SWAP A		$(A_{0\ldots3})\leftrightarrow(A_{4\ldots7})$

Команды операций с битами	Т Б Ц	Операция
Операции с битом переноса	1 1 1	
CLR C		(C) ← 0
SETB C		(C) ← 1
CPL C		(C) ← NOT(C)
Сброс/установка бита с номером bit	4 2 1	
CLR bit		(b) ← 0
SETB bit		(b) ← 1
CPL bit		(b) ← NOT(b)
Логическое умножение переноса и бита	4 2 2	
ANL C, bit		(C) ← (C) AND (b)
ANL C, /bit		(C) ← (C) AND (NOT(b))
Логическое сложение переноса и бита	4 2 2	
ORL C, bit		(C) ← (C) OR (b)
ORL C, /bit		(C) ← (C) OR (NOT(b))
Пересылка бита		
MOV C, bit	4 2 1	(C) ← (b)
MOV bit, C	4 2 2	(b) ← (C)

Команды передачи управления	Т Б Ц	Операция
Длинный переход LJMP ad16	12 3 2	 (PC)←ad16
Абсолютный переход в границах 2 кБ AJMP ad11	6 2 2	 (PC)←(PC)+2, ($PC_{0\text{-}10}$)←ad11
Короткий относит. переход в границах 256 байт SJMP rel	5 2 2	 (PC)←(PC)+2, (PC)←(PC)+rel
Косвенный относительный переход JMP @A+DPTR	1 1 2	 (PC)←(A)+(DPTR)
Переход, если аккумулятор =0, ≠0 JZ rel JNZ rel	5 2 2	 (PC)←(PC)+2, если (A)=0,то (PC)←(PC)+rel (PC)←(PC)+2, если (A)≠0,то (PC)←(PC)+rel
Переход, если перенос =1, ≠1 JC rel JNC rel	5 2 2	 (PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel (PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel
Переход, если бит =1, =0 JB bit, rel JNB bit, rel	11 3 2	 (PC)←(PC)+3, если (b)=l, то (PC)←(PC)+rel (PC)←(PC)+3, если (b)=0, то (PC)←(PC)+rel
Переход, если бит установлен, с последующим сбросом бита JBC bit, rel	11 3 2	 (PC)←(PC)+3, если (bit)=1, то (bit)←0 и (PC)←(PC)+rel
Декремент операнда и переход, если ≠0 DJNZ Rn, rel DJNZ ad, rel	 8 2 2 8 3 2	 (PC)←(PC) +2 (3), (op)←(op)–1, если (op) ≠ 0, то (PC)←(PC)+rel
Сравнение операнда1 с операндом2 и переход, если не равно CJNE A, ad, rel CJNE A, #d, rel CJNE @Ri, #d, rel	 8 3 2 10 3 2 10 3 2	 (PC)←(PC)+3, если (op1)≠(op2), то (PC)←(PC)+rel, если (op1)<(op2), то (C)←1, иначе (C)←0
Длинный вызов подпрограммы LCALL adl6	12 3 2	(PC)←(PC) + 3, (SP)←(SP)+1, ((SP))←($PC_{0\ldots7}$), (SP)←(SP)+1, ((SP))←($PC_{8\ldots15}$), (PC)←ad16
Абсолютный вызов подпрограммы в пределах 2 кБ ACALL ad11	6 2 2	(PC)←(PC)+2, (SP)←(SP)+1, ((SP))←($PC_{0\ldots7}$), (SP)←(SP)+1, ((SP))←($PC_{8\ldots15}$), ($PC_{0\text{-}10}$)←ad11
Возврат из подпрограммы RET	1 1 2	($PC_{8\ldots15}$)←((SP)), (SP)←(SP)–1, ($PC_{0\ldots7}$) ←((SP)), (SP)←(SP)–1
Возврат из прерывания RETI	1 1 2	($PC_{8\ldots15}$)←((SP)), (SP)←(SP)–1, ($PC_{0\ldots7}$)←((SP)), (SP)←(SP)–1
Пустая операция NOP	1 1 1	 (PC)←(PC)+1

Примечание. Ассемблер допускает использование обобщенного имени команд *JMP* и *CALL*, которые в процессе трансляции заменяются оптимальными по формату командами перехода (*AJMP, SJMP, LJMP*) или вызова (*ACALL, LCALL*).

Команды, влияющие на флаги результата представлены в нижеследующей таблице.

Мнемоника	Флаги
ADD A, <байт источника> ADDC A, <байт источника> SUBB A, <байт источника>	AC, C, OV
CLR C CPL C SETB C MOV C, <бит источника> ANL C, <бит источника> ANL C, </бит источника> ORL C, <бит источника> ORL C, </бит источника> RLC A RRC A	C
CJNE <байт назначения>, <байт источника>, <смещение>	C
DA A	AC, C
MUL AB DIV AB	C=0, OV

2. Общие сведения о МК *Motorola* 68*HC*05

Семейство МК 68*HC*xx имеют принстонскую архитектуру (Фон-Неймана), а ядром МК является *CISC*-процессор. Существует более 180 моделей 65*HC*05, специализированных для различных областей применения.

На рис.2.1 представлена упрощенная структурная схема МК.

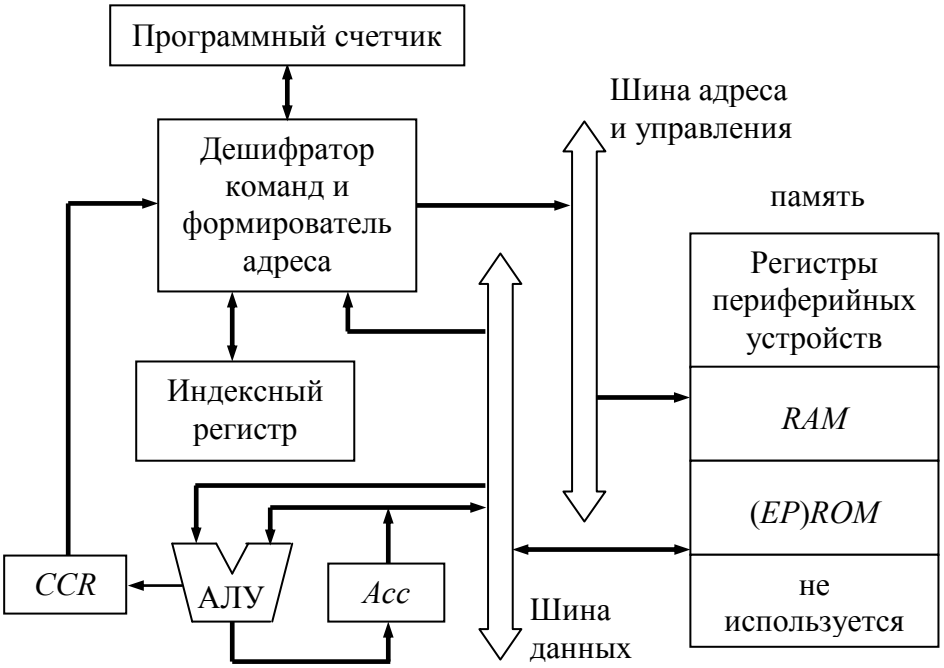


Рис. 2.1 – Структурная схема МК семейства 68*HC*05

CCR – регистр условий. На рис.2.2 представлена его структура и назначение битов.

Бит	7	6	5	4	3	2	1	0
Флаг	1	1	1	*A*	*I*	*N*	*Z*	*C*

Перенос между тетрадами – устанавливается при переносе (заеме) в (из) старшую тетраду при сложении (вычитании)

Маскирование прерываний – установка данного бита разрешает обслуживание прерываний

Отрицательный результат – устанавливается при отрицательном результате

Нулевой результат – устанавливается при нулевом результате операции

Перенос/заем – устанавливается при выходе результата за границы 8-разрядной сетки

Рис. 2.2 – Регистр условий (*CCR*), состав и назначение битов

После включения питания МК прежде чем начать работу вводит задержку длительностью 4064 такта, необходимую для стабилизации работы генератора и окончания переходных процессов, вызванных включением питания. После этого, а также после снятия активного сигнала *Reset* процессор загружает вектор начального состояния – адрес первой выполняемой команды, расположенной в конце адресного пространства *EPROM*.

Большинство МК семейства 86*HC*05 содержит в своем составе масочное ПЗУ, в котором имеется несколько программ. Эти программы предназначены для программирования МК через последовательный или параллельный интерфейс. Активация режима программирования происходит путем подачи на входы МК определенной комбинации сигналов при включении МК.

Особенностями МК данного семейства является:

1. Указатель стека в этом МК является 16-разрядным индексным регистром, значение которого всегда используется со смещением $0C0 (биты 6 и 7 всегда установлены). Начальным состоянием указателя стека является $0FF, т.е. стек при помещении в него слов растет вниз. При вызове подпрограммы содержимое программного счетчика сохраняется в стеке (первым сохраняется младший байт).
2. Реализация прерываний похожа на процесс начального запуска. При поступлении запроса на прерывание при разрешенных прерываниях МК выбирает из конца *EPROM* адрес обработчика прерывания. При этом в стеке сохраняется контекст регистров в следующем порядке: старший байт программного счетчика, младший байт программного счетчика, индексный регистр, аккумулятор, регистр условий. Эта процедура занимает 10 машинных тактов с учетом загрузки вектора прерывания.
3. Доступ к периферийным регистрам, которые занимают первые 32 байта в адресном пространстве, осуществляется непосредственно путем записи/чтения по соответствующим адресам.
4. Программный код может быть загружен с помощью специальной программы хранящейся в ПЗУ в ОЗУ, упрощая тем самым процесс отладки программы.

2.1 СПОСОБЫ АДРЕСАЦИИ ДАННЫХ

МК семейства 68*HC*xx при хранении данных в памяти существенно отличаются от других МК, которые при сохранении 16-разрядного числа в памяти используют правило: младший байт по меньшему адресу. В МК *Motorola* все наоборот.

В МК 68*HC*05 реализуются следующие виды адресации:

- Регистровая (регистр, с которым работает команда, содержится в ее теле): INCA – инкремент аккумулятора.
- Непосредственная (операнд содержится в команде): ADD #2 – добавить 2 к содержимому аккумулятора.
- Прямая (в команде содержится адрес операнда): LDA $37 – загрузить в аккумулятор содержимое ячейки памяти с адресом 37h (адресует память в первых 256 байтах).
- Расширенная (16-разрядный адрес ячейки памяти содержится в команде): goto $1234 – загрузить в программный счетчик значение 1234h.
- Индексная: 8-разрядный адрес операнда находится в индексном регистре
- Индексная со смещением (8-разрядное смещение содержится в команде): адрес операнда вычисляется суммированием смещения и значением индексного регистра.
- Индексная с 16-разрядным смещением: значение индексного регистра суммируется с 16-разрядным смещением, содержащимся в команде.

2.2 ПАРАЛЛЕЛЬНЫЙ ВВОД-ВЫВОД ДАННЫХ

Порты ввода-вывода этих МК мало отличаются от портов других МК. На рис.2.3 представлена структура порта ввода-вывода МК данного семейства.

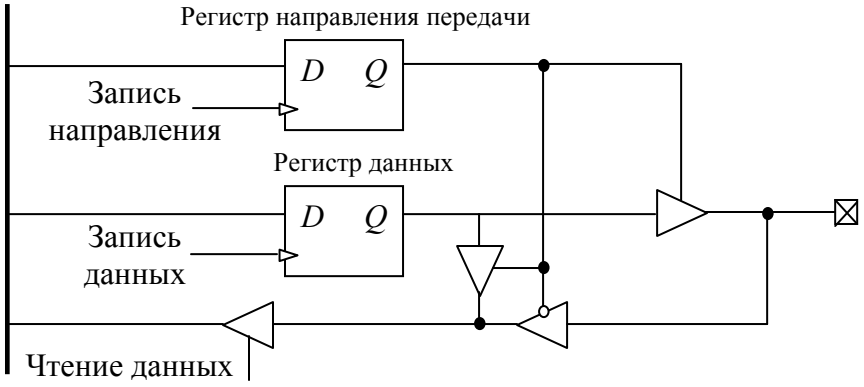


Рис. 2.3 – Структурная схема порта ввода-вывода семейства МК 68*HC*05

2.3 ПРОГРАММИРУЕМЫЙ ТАЙМЕР

Таймеры МК этого семейства существенно отличаются от таймеров других МК. Вместо того чтобы формировать временные интервалы, заданные пользователем, таймеры 68*HC*05 работают непрерывно, позволяя программе следить за их состоянием. Кроме того, структура таймера изменяется в пределах семейства. Рассмотрим два таймера МК простого: 68*HC*05*J*1*A* и более сложного: 68*HC*05*C*8.

Таймер МК 68*HC*05*J*1*A* работает по внутреннему тактовому сигналу или от внешнего синхросигнала деленного на два. При переполнении таймера может быть вызвано прерывание. При работе с таймером надо учитывать особенности:

1) для реализации задержки необходимо производить циклический опрос таймера;
2) сигнал переполнения счетчика таймера заведен на линейку делителей;
3) сигнал с последней ступени линейки делителей служит синхроимпульсами для сторожевого таймера (*COP*).

На рис.2.4 показана структура таймера МК 68*HC*05*J*1*A*.

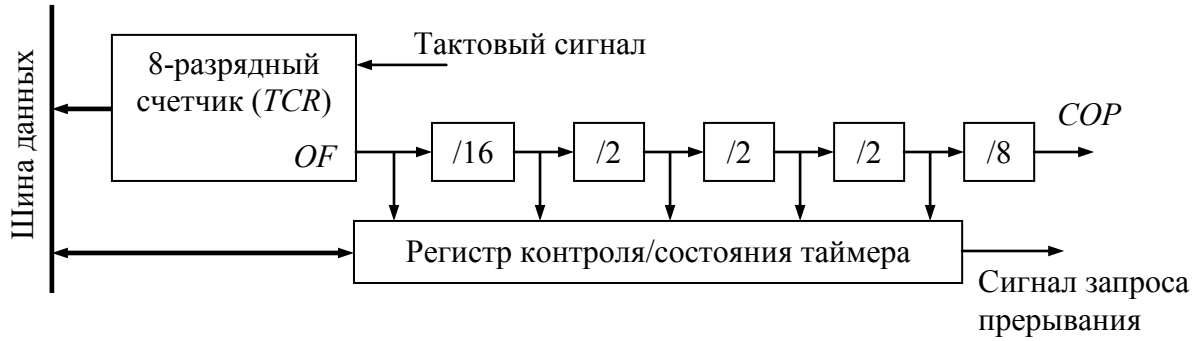


Рис. 2.4 – Структурная схема таймера МК 68*HC*05*J*1*A*

На рис.2.5 показана структура таймера МК 68*HC*05*C*8. Такая конфигурация более гибкая, но и более сложная. Здесь таймер может реагировать на различный входной сигнал (по уровням или перепадам), есть возможность аппаратного отсчета временного интервала, доступны режимы: сравнения и захвата.

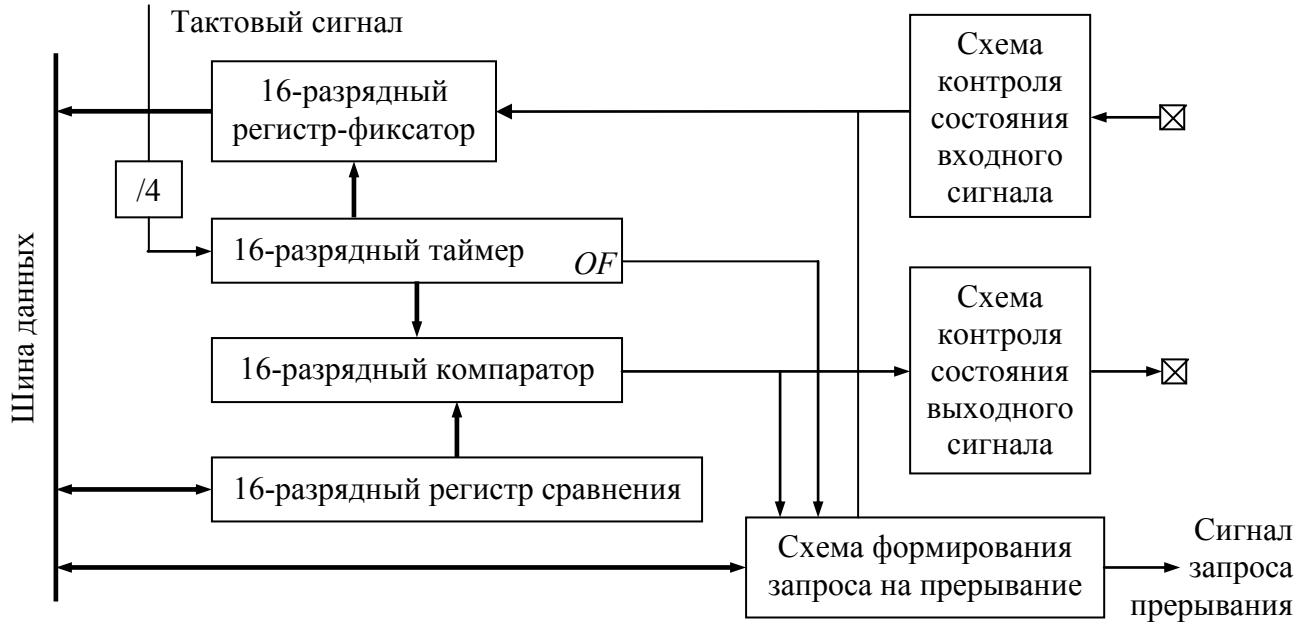


Рис. 2.5 – Структура таймера МК 68*HC*05*C*8

2.4 ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД-ВЫВОД

Как и в других МК последовательный порт этого семейства МК может работать в синхронном и асинхронном режимах.

Последовательный порт асинхронного приемопередатчика является полным дуплексным портом. Этот режим (*SCI*) обеспечивает обмен данными в формате *RS*-232. На рис.2.6 представлены структурные схемы передатчика, приемника и генератора синхросигналов модуля *SCI* в 68*HC*05.

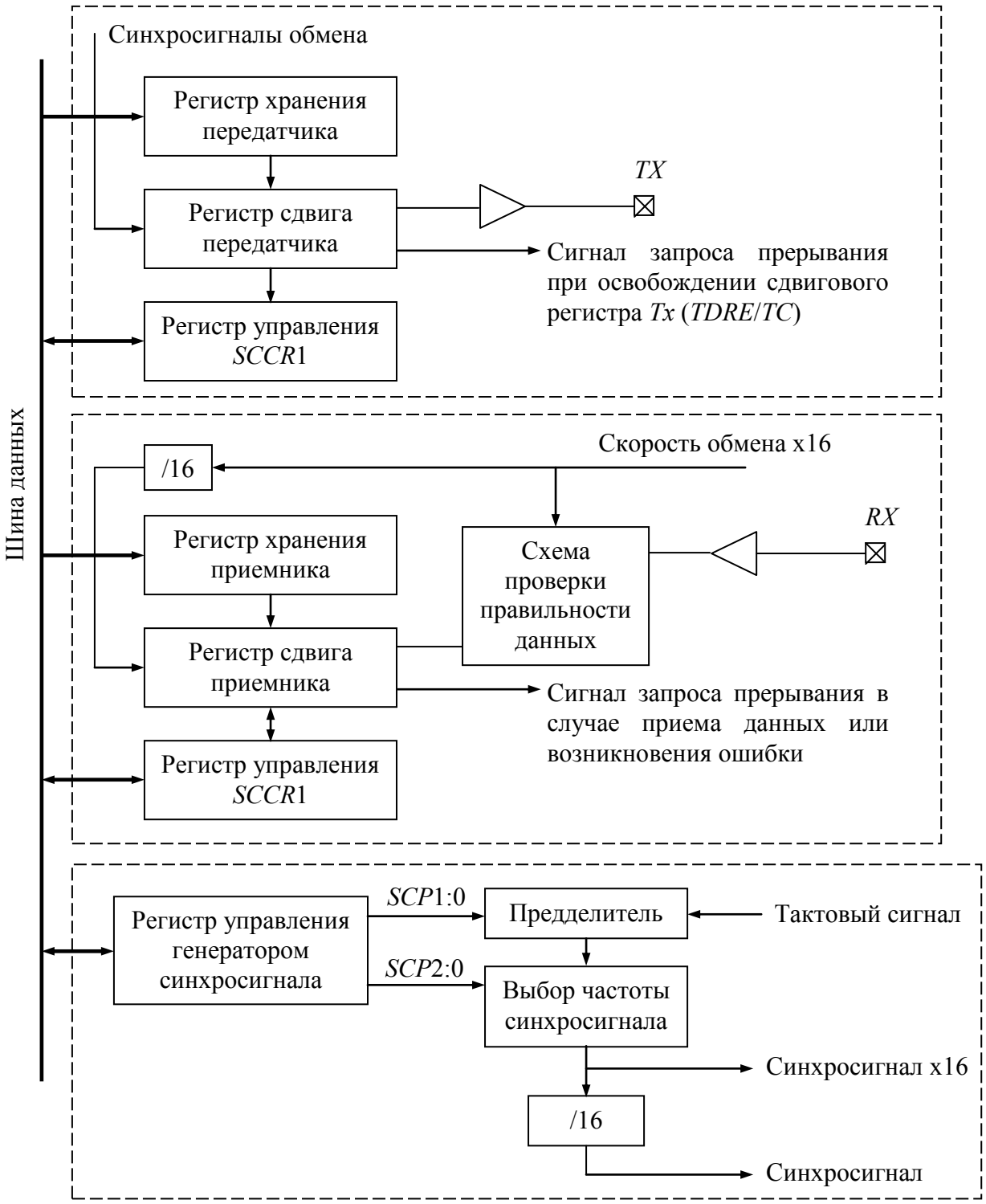


Рис. 2.6 – Структурные схемы передатчика, приемника и генератора синхросигналов модуля *SCI* в 68*HC*05 (асинхронный режим)

Синхронный модуль последовательного интерфейса (*SPI*) разрабатывался для взаимодействия с преобразователями последовательного кода в параллельный. Поэтому синхронный режим последовательного порта отличается от аналогичного режима других МК. Структурная схема последовательного порта в синхронном режиме представлена на рис.2.7. Передача данных инициируется записью байта в сдвиговый регистр и происходит по переднему фронту синхроимпульса. При этом первым передается старший бит.

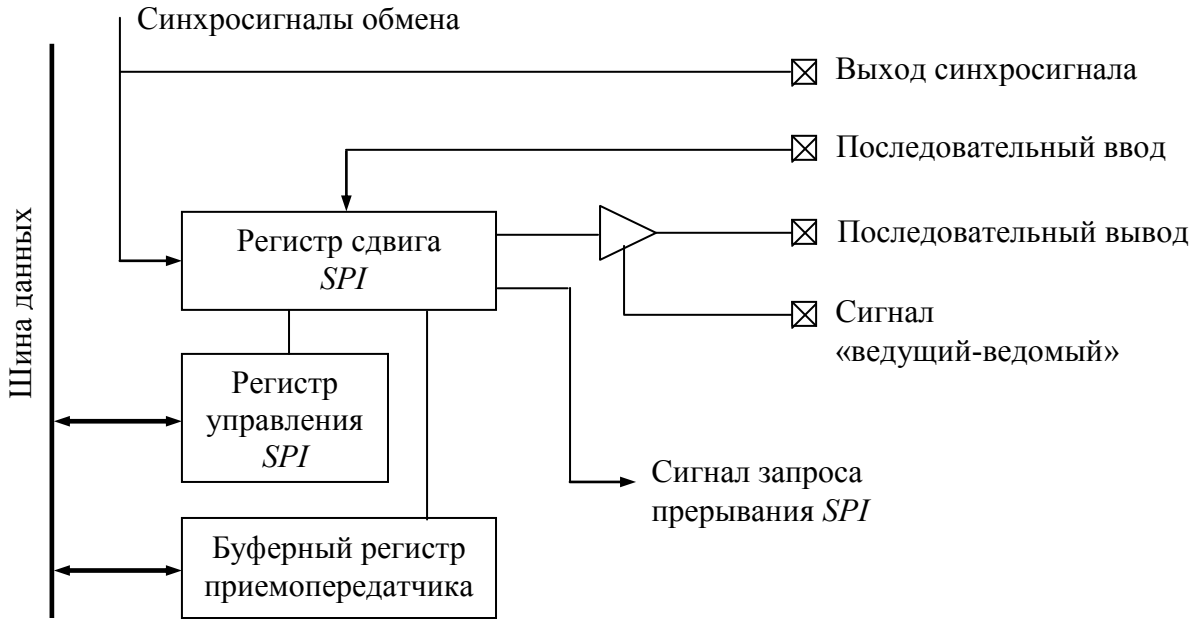


Рис. 2.7 – Структурная схема модуля *SPI* в МК 68*HC*05

3. *AVR* микроконтроллеры.

Компания *Atmel* на сегодняшний день, как и другие компании, производят широкую гамму микроконтроллеров, подходящих для решения различных задач. При этом, например серия МК *AT*89*Sxxxx* повторяет по возможностям и системе команд серию МК *Intel* 87*C*51. Также *Atmel* производит ряд собственных МК, основные характеристики некоторых из них приведены в табл.3.1.

Таблица 3.1 – Некоторые МК семейств 8 бит

МК	*Flash КБ*	*SRAM* (*EEPROM*) байт	F_{max} (МГц)	Макс *I/O pin*	Прерывания (внеш)	*SPI*	АЦП каналов	*UART*	ШИМ каналов	*Vcc* (В)	Корпус
AT90S1200	1	0 (64)	12	15	3 (1)					2.7-6.0	DIP20, SO20, SSOP20
AT90S2313	2	128 (128)	10	15	10 (2)			1	1	2.7-6.0	DIP20, SO20
ATmega128	128	4096 (4096)	16	53	34 (8)	1	8	2	8	4.5 - 5.5	QFP64
ATmega162	16	1024 (512)	16	35	28 (3)	1		2	4	4.5 - 5.5	DIP40, QFP44
ATmega168	16	1024 (512)	20	23	26 (26)	1	8	*USART*	3	1.8 - 5.5	DIP28, QFP32, MLF32
ATmega32	32	2048 (1024)	16	32	19 (3)	1	8	1	4	4.0 - 5.5	DIP40, QFP44, MLF44
ATmega324P	32	2048 (1024)	20	32	31 (3)	1	8	1	3	4.5 - 5.5	DIP40, QFP44 MLF44
ATmega48	4	512 (256)	24	23	26 (26)	1	8	*USART*	3	1.8 - 5.5	DIP28, QFP32, MLF32
ATmega64	64	4096 (2048)	16	53	34 (8)	1	8	2	8	4.5 - 5.5	QFP64
ATmega644P	64	4096 (2048)	20	32	31 (3)	1	8	1	3	4.5 - 5.5	DIP40, QFP44 MLF44
ATmega8	8	1024 (512)	16	23	18 (2)	1	8	1	3	4.5 - 5.5	DIP28, QFP32, MLF32
ATtiny11	1		6	6	4 (1)					4 - 5.5	DIP8, SO8
ATtiny12	1	0 (64)	8	6	5 (1)					4 - 5.5	DIP8, SO8
ATtiny2313	2	128 (128)	20	18	10 (2)	*USI*		*USART*	4	1.8 - 5.5	DIP20, SO20
ATtiny26	2	128 (128)	16	16	11 (1)	*USI*	11		2	4.5 - 5.5	DIP20, SO20, MLF32

- *AT*90*S*1200, *AT*90*S*2313, *ATtiny*11 – не имеют детектора питания
- *AT*90*S*1200, *AT*90*S*2313, все *ATtiny* – не имеют аппаратного умножителя
- Разрядность АЦП – 10 бит.
- *USART* (*Universal Synchronous Asynchronous Receiver-Transmitter*) отличается от *UART* (*Universal Asynchronous Receiver-Transmitter*) более высокой максимальной скоростью (до 4 Мбит/сек) и большими возможностями.
- Интерфейс *USI* присутствует во многих МК и представляет собой заготовку из сдвигового регистра, 4х-битного счетчика и набора регистров инициализации с прерываниями. Может работать как в 3х-проводном режиме (как *SPI*) так и в 2х-проводном режиме (*TWI* или *I2C*), также у него есть нестандартные применения для реализации *UART*.

Необходимо отметить, что все МК одного семейства совместимы по исходным кодам, обеспечены программным обеспечением, а также аппаратными средствами, позволяющими отлаживать приложения с учетом реального масштаба времени. МК семейства *AVR* делятся на три группы (подсемейства): *Classic*, *Tiny* и *Mega*. МК семейства *Classic* (*AT*90*S*xxxx) в

настоящее время уже не выпускаются, однако все «классические» *AVR* имеют полные аналоги в семействах *Tiny* и *Mega*.

Архитектура всех МК является гарвардской с элементами принстонской архитектуры, что обеспечивает эффективность МК при решении широкого круга задач. Так архитектура включает в себя 32 регистра общего назначения (РОН), выполняют большинство команд за один цикл – свойственно *RISC* системам; память программ и данных разделены – свойственно гарвардской архитектуре; имеются предопределенные регистры – свойственно принстонской архитектуре; имеются сложные команды, требующие нескольких циклов – свойственно *CISC* системам и т.п. В общем, если попытаться сформулировать отличительные черты МК этих серий, то получится примерно следующее:

1. Производительность порядка 1 *MIPS*/МГц, вычислительное ядро *AVR* на ряде задач по производительности превосходит 16-разрядный процессор *i*80286.
2. Усовершенствованная *RISC* архитектура, концепция которой предполагает наличие набора команд, состоящего из минимума компактных и быстро выполняющихся инструкций. Это упрощает устройство ядра и ускоряет его работу: типовая инструкция выполняется за один такт (кроме команд ветвления программы, обращения к памяти и некоторых других, оперирующих с данными большей длины). В *AVR* имеется двухступенчатый конвейер, когда в одном такте совмещаются операции выполнения одной команды и загрузки другой.
3. Раздельные шины памяти команд и данных. Реализована гарвардская архитектура, т.е. данные и команды могут выбираться одновременно.
4. 32 регистра общего назначения (РОН). *Atmel* была первой компанией, ушедшей далеко от классической модели вычислительного ядра, в которой выполнение команд предусматривает постоянный обмен данными между АЛУ и аккумулятором. Введение РОН в таком количестве в ряде случаев позволяет полностью отказаться от расположения локальных и глобальных переменных в ОЗУ и от использования стека. Правда, это привело к некоторому усложнению системы команд пересылок данных.
5. *Flash*-память программ (10 000 циклов стирание/запись) с возможностью внутрисистемного перепрограммирования и загрузки через последовательный канал прямо в готовой схеме.
6. Наличие отдельной области энергонезависимой памяти (*EEPROM*, 100 000 циклов стирание/запись) для хранения данных, с возможностью записи программным путем, или внешней загрузки через *SPI*-интерфейс.
7. Встроенные устройства для обработки аналоговых сигналов: аналоговый компаратор и многоканальный АЦП.
8. Сторожевой таймер, позволяющий осуществлять автоматическую перезагрузку контроллера через определенные промежутки времени.
9. Последовательные интерфейсы *SPI*, *TWI* (*I2C*) и *UART* (*USART*), позволяющие осуществлять обмен данными с большинством стандартных датчиков и других внешних устройств.
10. Таймеры/счетчики с предустановкой и возможностью выбора источника счетных импульсов: как правило, один или несколько 8-разрядных и как минимум один 16-разрядный, в т.ч. способные работать в режиме широтно-импульсной модуляции (ШИМ, англ.: *PWM*).
11. Возможность работы при тактовой частоте от 0 Гц до 20 МГц.
12. Диапазон напряжений питаний от 2.7 до 5.5 В (в некоторых случаях от 1.8 до 6.0В).
13. Низкое потребление и многочисленные режимы энергосбережения, различающиеся числом узлов, остающимися включенными. Выход из «спящих» режимов осуществляется либо по сторожевому таймеру, либо по внешним прерываниям.
14. Встроенный монитор питания – детектор падения напряжения.

Из широкой номенклатуры данного семейства МК рассмотрим *AT*90*S*2313:

- 120 команд, большинство которых выполняются за один цикл;
- 2 Кбайт *Flash* ПЗУ программ с возможностью внутрисистемного перепрограммирования (1000 циклов стирания/записи) и загрузки через *SPI*;
- 128 байт электрически стираемого и программируемого ПЗУ (10000 циклов стирания/ записи) с возможностью внутрисистемной загрузки через *SPI*;
- 15 программируемых линий ввода/вывода;
- полностью статический прибор (работает от 0 до 10 МГц);
- напряжение питания от 2.7 В до 6 В;
- 8-разрядный и 16-разрядный с режимами сравнения и захвата таймеры/счетчики с общим прескаляром;
- функция ШИМ с 8, 9, 10 битным разрешением;
- полный дуплексный *UART*;
- два внешних и восемь внутренних источника сигналов прерывания;
- программируемый сторожевой таймер с собственным генератором;
- встроенный аналоговый компаратор;
- режимы энергосбережения: пассивный (*idle*) и стоповый (*power down*);
- возможность работы без внешних компонентов (встроенный *RC* генератор);
- 20-контактный корпус - рис. 3.1.

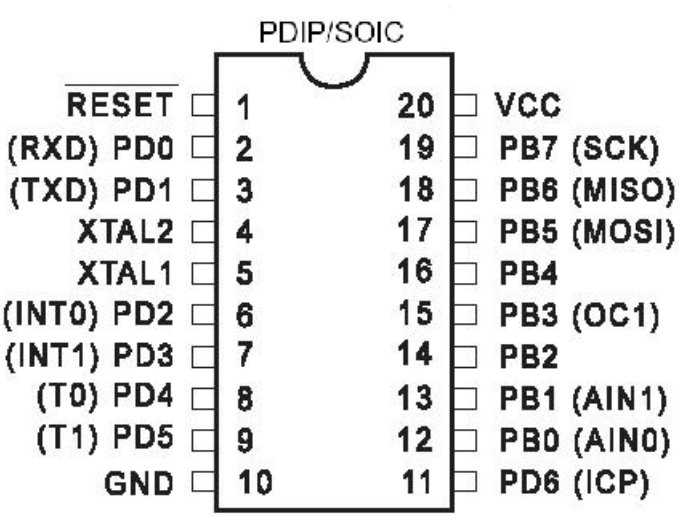


Рис. 3.1 – Цоколевка *AT*90*S*2313

3.1 АРХИТЕКТУРА

На рис.3.2 представлена архитектура МК *AT*90*S*2313.

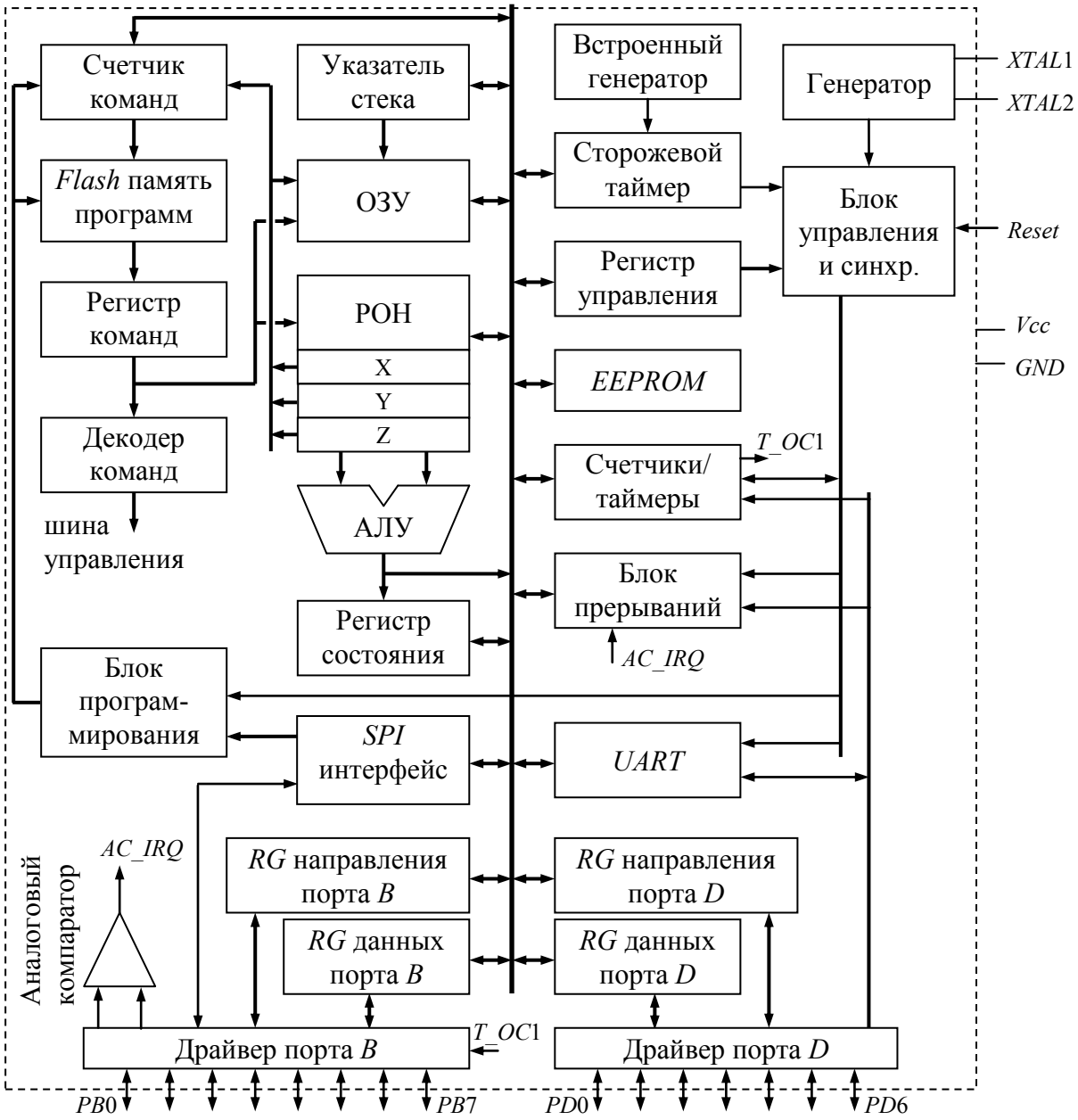


Рис. 3.2 – Блок схема МК *AT*90*S*2313 и расположение выводов

Можно видеть, что использование сразу всех блоков невозможно (например: компаратор и *SPI* разделяют выводы порта *B*, число линий порта *D* – 7 (не 8) и они разделяются с *UART* и таймерами, внешними прерываниями и т.п.). Такая ситуация свойственна всем МК.

Прежде чем начать рассмотрение блоков структурной схемы и соответственно возможностей и попутно системы команд рассмотрим распределение памяти МК и систему адресации. Существует два способа обращения (адресации) к ресурсам МК: регистрам (РОН), портам ввода-вывода и памяти. Первый способ – это прямое обращение к требуемой области. Второй – обращение к областям как к единому пространству адресов. На рис.3.3 представлено распределение памяти МК. Соответственно для обращения к каждой области памяти существуют соответствующие команды:

- к регистрам ввода-вывода: IN/OUT – для обмена данными между РОН и портами;
- к памяти: LD – загрузить из памяти (*load*) и ST – записать в память (*store*) – используются для пересылки данных между ОЗУ и РОН, между ПЗУ и РОН (только чтение);
- к РОН: MOV – для пересылки данных между РОНами.

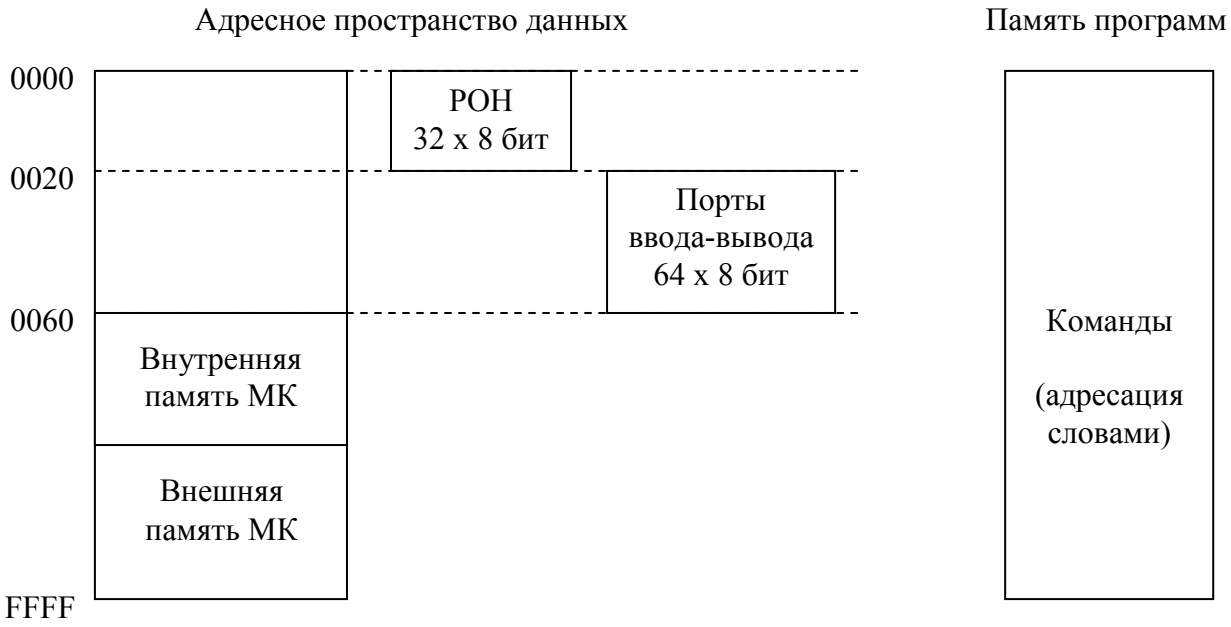


Рис. 3.3 – Распределение памяти семейства МК *AVR AT*90*S*xxxx

3.1.1 СПОСОБЫ АДРЕСАЦИИ ДАННЫХ

В МК *AVR* по признаку адресации можно все команды можно разбить на две группы: непосредственная адресация и индексная адресация. К первой группе относятся команды, работающие с РОН и непосредственными операндами, указанными в самой команде – рис.3.4 – для команд с одним, двумя операндами и с непосредственным операндом. Ко второй группе команд относятся команды с косвенной адресацией операнда, при этом адрес формируется согласно структуре – рис.3.5 (возможна автоматическая инкрементация/декрементация индексных регистров до выполнения команды или после нее).

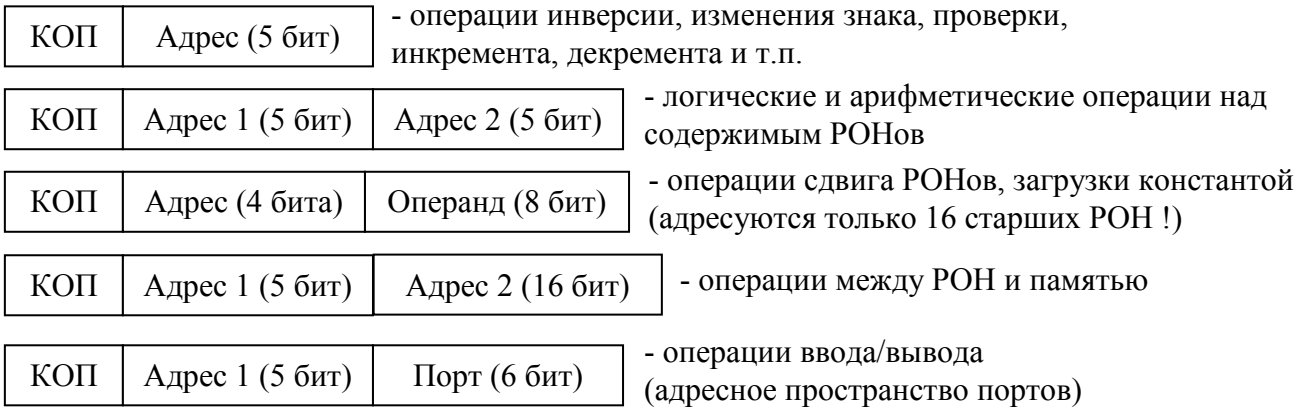


Рис. 3.4 – Структура команд с непосредственной адресацией операндов

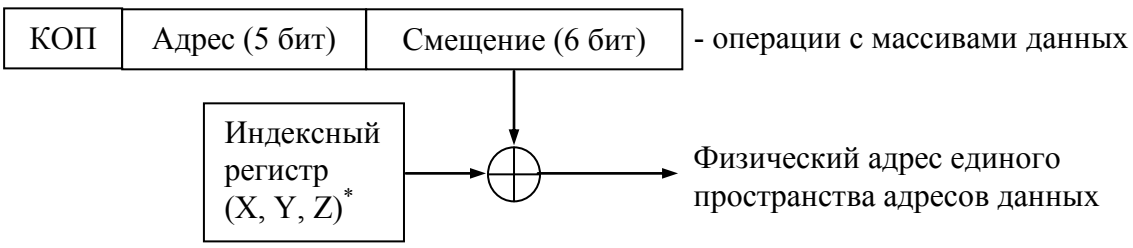


Рис. 3.5 – Формирование адреса операнда команд с косвенной адресацией

3.1.2 РОНы и АЛУ

АЛУ выполняет 91 из 120 команд МК. Все 32 регистра подключены к АЛУ. Последние шесть 8-разрядных регистров попарно объединены и образуют три 16-разрядных индексных регистра (*X*, *Y*, *Z*). При этом индексный регистр *Z* может применяться для чтения данных из памяти программ (младший бит адреса указывает читаемый байт в 16-разрядном слове из ПЗУ).

АЛУ выполняет три категории команд: арифметические, логические и битовые, результат операций отражается на флагах регистра состояния. На рис.3.6 представлен состав и назначение бит регистра состояния *SREG*.

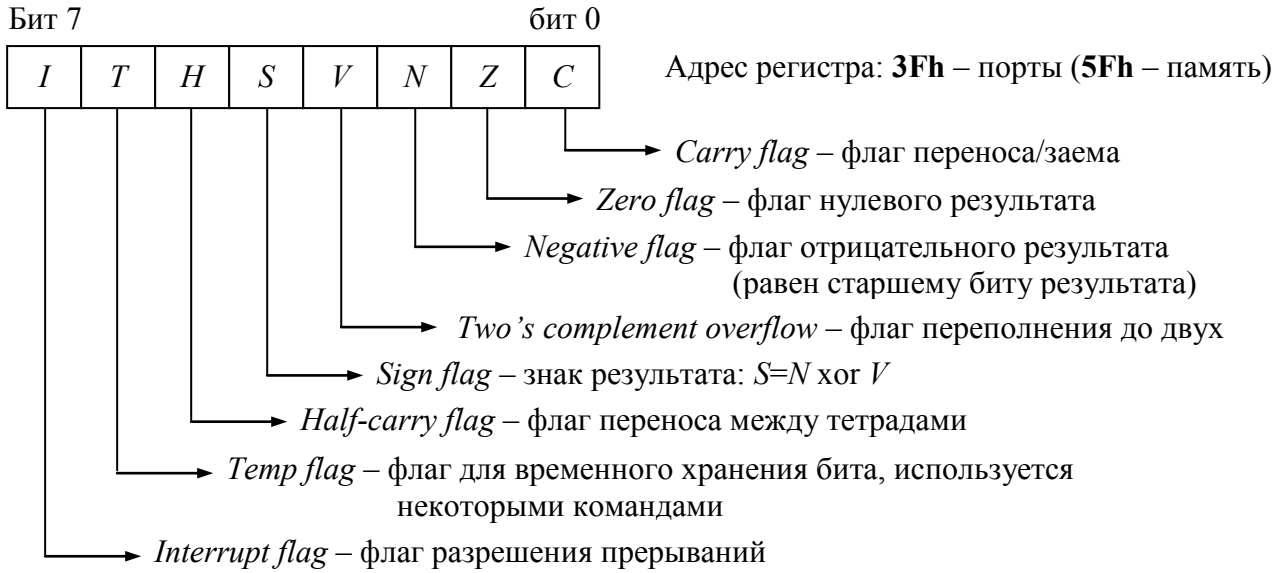


Рис. 3.6 – Регистр состояния МК (*SREG*)

3.1.3 ПРЕРЫВАНИЯ

Данный МК поддерживает 10 прерываний. Каждому прерыванию присвоен вектор прерывания, которые расположены в начале памяти программ. Каждый вектор прерывания содержит команду относительного перехода на заданный адрес обработчика прерывания (*IRQ handler*) – 2 байта (1 слово). Назначение векторов прерываний следующее:

Адрес	Вектор	Назначение
0000	1	– вектор сброса (*Reset*)
0001	2	– внешний запрос (*INT*0)
0002	3	– внешний запрос (*INT*1)
0003	4	– таймер 1, событие захвата (*Timer Capt* 1)
0004	5	– таймер 1, событие совпадения (*Timer Comp* 1)
0005	6	– таймер 1, переполнение (*Timer Ovf* 1)
0006	7	– таймер 0, переполнение (*Timer Ovf* 0)
0007	8	– *UART*, прием завершен (*UART RX*)
0008	9	– *UART*, регистр данных пуст (*UART UDRE*)
0009	10	– *UART*, передача завершена (*UART TX*)
000A	11	– Аналоговый компаратор (*ANA_COMP*)

Следовательно, любая программа, использующая прерывания, должна начинаться с адреса выше 000Bh (адресация в ПЗУ словами).

Особенности обработки внешних прерываний:

- активным сигналом может быть *rise, falling, low level* – см. *MCU Control RG* (биты *ISC*);
- входные сигналы *IRQ* фиксируются триггерами (флаги) – *GIFR* (*General Interrupt Flag RG*, биты *INTF*1, *INTF*0), лог. "1" в битах говорит о наличии прерывания

	7	6	5	4	3	2	1	0
GIFR $3A ($5A)	*INTF*1	*INTF*0	-	-	-	-	-	-
	R/W	*R/W*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

- есть регистр маски внешних прерываний *GIMSK* (*General Interrupt Mask RG*), лог. "1" в битах этого регистра разрешает прерывания

	7	6	5	4	3	2	1	0
GIMSK $3B ($5B)	*INT*1	*INT*0	-	-	-	-	-	-
	R/W	*R/W*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

- если прерывание разрешено битами *GIMSK* и флагом *I* (*SREG*), то запрос по входам *INT* может быть выполнен, при этом соответствующие биты *GIFR* автоматически сбрасываются при выполнении обработчика прерываний (для программной очистки этих битов необходима запись «1»);
- фиксация активного сигнала по входам *INT* происходит всегда, даже если соответствующие выводы сконфигурированы как выход (порт), это позволяет эмулировать программное прерывание.

По аналогии обрабатывается прерывания от таймера, отличия состоят в других наименованиях регистров:

- входные сигналы фиксируются триггерами – *TIFR* (*Timer/Counter Interrupt Flag*)

	7	6	5	4	3	2	1	0
TIFR $38 ($58)	*TOV*1	*OCF*1*A*	-	-	*ICF*1	-	*TOV*0	-
	R/W	*R/W*	*R*	*R*	*R/W*	*R*	*R/W*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

TOV1 – (*Timer/Counter Overflow Flag*) устанавливается в «1» когда таймер 1 переполняется, бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

OCF1A – (*Output Compare Flag* 1*A*) устанавливается в «1» когда значение таймера 1 совпадает со значением регистра *OCR*1 (*Output Compare RG*), бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

ICF1 – (*Input Capture Flag*) устанавливается в «1» когда происходит событие захвата, т.е. показывает, что текущее значение таймера 1 скопировано в регистр *ICR*1 (*Input Capture RG*), бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

TOV0 – (*Timer/Counter Overflow Flag*) устанавливается в «1» когда таймер 0 переполняется, бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

- регистр маски прерываний таймеров *TIMSK* (*Timer/Counter Interrupt Mask RG*)

	7	6	5	4	3	2	1	0
TIMSK $39 ($59)	*TOIE*1	*OCIE*1*A*	-	-	*TICIE*1	-	*TOIE*0	-
	R/W	*R/W*	*R*	*R*	*R/W*	*R*	*R/W*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

TOIE1 – (*Timer/Counter*1 *Overflow Interrupt Enable*) установка бита в «1» разрешает прерывание таймера 1 по переполнению;

OCIE1A – (*Timer/Counter*1 *Output Compare Match Interrupt Enable*) установка бита в «1» разрешает прерывание таймера 1 по сравнению;

TICIE1 – (*Timer/Counter*1 *Input Capture Enable*) установка бита в «1» разрешает прерывание таймера 1 по захвату;

TOIE0 – (*Timer/Counter*0 *Overflow Interrupt Enable*) установка бита в «1» разрешает прерывание таймера 0 по переполнению;

- если прерывание разрешено битами *TIMSK* и флагом *I* (*SREG*), то запрос от таймера может быть обслужен.

Общий порядок обработки прерываний следующий:

1) фиксация сигнала прерывания входными схемами;
2) проверка масок прерываний;
3) выбор разрешенного прерывания с наивысшим приоритетом (по номеру: 0 – наивысший);
4) проверка флага *I* (*SREG*), если равен «1», то начало обработки прерывания;
5) сброс соответствующего бита во флагах фиксирующих сигнал прерывания (*TIFR*, *GIFR*);
6) сохранение в стеке значения программного счетчика (*PC*), уменьшение значения указателя стека на 2;
7) передача управления процедуре обработки прерывания и сброс бита *I* (*SREG*).

При возврате из прерывания:

1) из стека загружается значение *PC,* значение указателя вершины стека увеличивается на 2;
2) бит *I* (*SREG*) устанавливается в «1»;
3) передается управление прерванной программе.

Время отклика МК на прерывание определяется 4 тактами. Однако если прерывание пришло во время выполнения сложной инструкции, то запрос ожидает ее окончания. Возврат из прерывания также занимает 4 такта.

Управляющий регистр *MCUCR* (*MCU Control RG*)

	7	6	5	4	3	2	1	0
MCUCR \$35 (\$55)	-	-	*SE*	*SM*	*ISC*11	*ISC*10	*ISC*01	*ISC*00
	R	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

SE – (*Sleep Enable*) установка в «1» разрешает спящий режим;

SM – (*Sleep Mode*) управляет типом спящего режима:

"0" – *idle* режим – остановка *CPU*, работа таймеров разрешена, *Watchdog* и система прерываний функционируют. Выход из режима осуществляется поступлением прерывания. Выполнение программы при этом начинается немедленно.

"1" – *power down* режим – все выключается, работают только *Watchdog*, внешние прерывания и схема сброса. Только эти виды прерываний могут «разбудить» МК.

ISC11, ISC10 – (*Interrupt Sense Control* 1) выбор типа активного сигнала по входу *INT*1:

00 – низкий уровень;
01 – зарезервировано;
10 – задний фронт сигнала (*falling edge*);
11 – передний фронт сигнала (*rise edge*).

ISC01, ISC00 – (*Interrupt Sense Control* 0) выбор типа активного сигнала по входу *INT*0:

00 – низкий уровень;
01 – зарезервировано;
10 – задний фронт сигнала (*falling edge*);
11 – передний фронт сигнала (*rise edge*).

Для входа в спящий режим (*Sleep mode*) необходимо определить его тип (бит *SM*) и разрешить спящий режим (бит *SE*), а затем инструкция *SLEEP* переведет МК в соответствующий режим с пониженным энергопотреблением. Если во время «сна» происходит прерывание МК «пробуждается», выполняет процедуру обработки прерывания и начинает выполнять программу с команды, следующей за командой *SLEEP*. Если во время «сна» произошел сброс МК, то МК «пробуждается» и начинает работу с начала программы по вектору *Reset*.

3.1.4 СБРОС МК

МК имеет три источника сигнала сброса:

1) при включении питания (после достижения напряжения уровня V_{POT} включается МК);
2) при поступлении сигнала сброса с внешнего вывода МК (длительность сигнала должна превышать 50 нс);
3) от *Watchdog*.

Во всех случаях выполнение программы начинается спустя некоторое время после сброса, что необходимо для окончания различных переходных процессов. Длительность этой задержки по умолчанию составляет 16 мс (16k циклов *Watchdog*). Однако можно программно уменьшить ее до 0.28 мс (256 циклов *Watchdog*), этим управляет *Fuse* бит *FSTRT* (при этом необходимо применение внешнего кварцевого резонатора). На рис.3.7 представлена структурная схема цепей сброса.

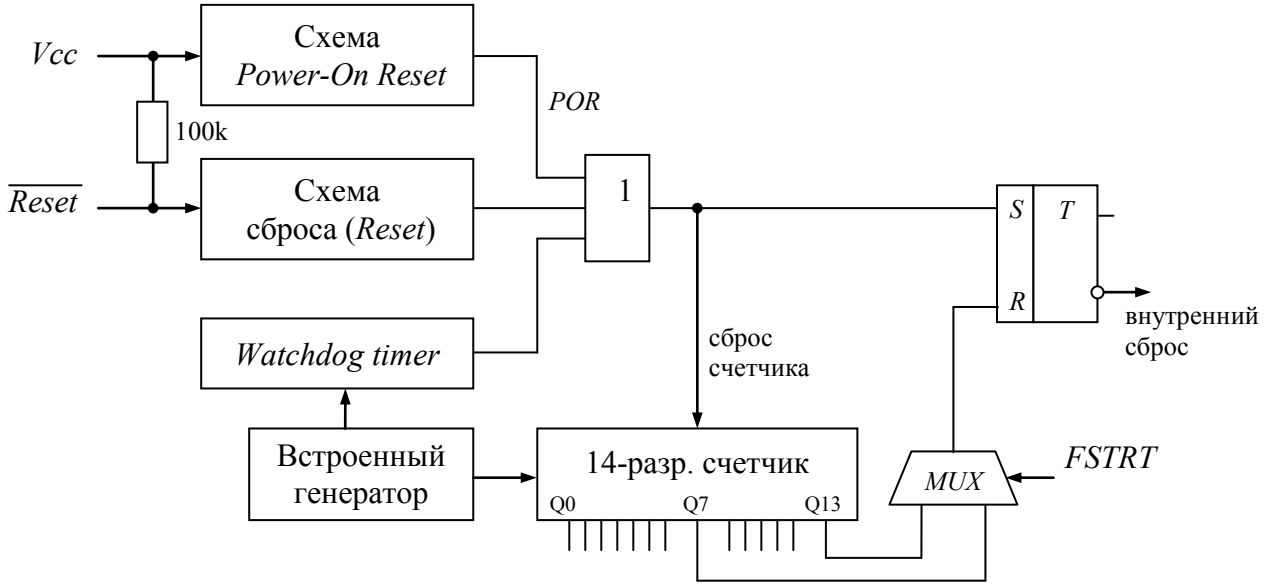


Рис. 3.7 – Структура цепей сброса

3.2 АССЕМБЛЕР

Все команды можно разделить на 4 большие группы:

- команды пересылки данных;
- команды вычислительного процесса;
- команды передачи управления;
- команды управления ходом работы МК.

Команды пересылки данных.

Эту группу команд можно разделить на три вида: обмен типа RG-RG, RG-mem, RG-порт.

MOV Rd,Rs – Rd=Rs
(mov R1, R20)

LDI Rd,const – Rd=const, в качестве Rd могут быть только 16 старших РОН
(ldi R22, $A5)

IN Rd,port – читает указанный порт (00-$3F) и помещает значение в Rd
(in R0, $10)

OUT port, Rs – записывает содержимое Rs в указанный порт (00-$3F)
(out $11, R4)

LDS Rd,addr – читает байт из памяти (addr - слово) и помещает в Rd
(lds R16, $0000 - аналогично mov R16, R0
lds R2, $0020 - аналогично in R2, $00
lds R2, $0080 - обращение к памяти)

STS Addr,Rs – сохраняет байт из Rs в памяти (addr - слово)
(sts $0055, R2)

LD Rd,index – загружает Rd байтом из памяти с адресом из индексного регистра X (R26:R27), Y (R28:R29) или Z (R30:R31) или R24:R25
(ld R1, X)

Команда LD Rd,index± – загрузка байта из памяти и инкрем./декрем. индексного RG
(ld R12,Y+ или ld R17,X-)

Команда LD Rd, ±index – инкрем./декрем. индексного RG и загрузка байта из памяти
(ld R12,+Y или ld R17,-X)

Команда LDD Rd,index+Ofs – загрузка байта в Rd из памяти по адресу определяемому суммой содержимого индексного регистра (Y или Z) и указанного смещения (0<=Ofs<=63)
(ldd R3,Y+6)

Команды ST index,Rd; ST index±,Rd; ST ±index,Rd; STD index+Ofs,Rd – аналогичны командам загрузки LD, но направление передачи – наоборот

Команда PUSH Rs – сохранить содержимое Rs в стеке ([sp]=Rs; sp=sp-1)

Команда POP Rd – извлечь из стека байт в Rd (sp=sp+1: Rd=[sp])

Команда LPM – чтение памяти программ: R0=[Z]

Резюме: команды типа LDS; STS; LPM выполняются за 3 цикла
команды типа LD; ST; LDD; STD; PUSH; POP выполняются за 2 цикла
остальные команды пересылки выполняются за 1 цикл

Команды арифметических, логических и битовых операций.

ADD Rd,Rs - Rd=Rd+Rs

ADC Rd,Rs - сложение значений регистров с учетом флага переноса

ADIW Rdd,const - сложение 16-разрядного регистра (R24,R26,R28,R30) с константой (0-63)
(adiw R27:R26,3 - X=X+3 как 16-разрядный регистр)

INC Rd - инкрементация значения регистра (не влияет на флаг переноса)

SUB Rd,Rs - Rd=Rd-Rs

SBC Rd,Rs - вычитание с учетом заема (флага переноса)

SUBI Rd,const - Rd=Rd-const (подобной команды сложения нет -> const<0 это решение)

SBCI Rd,const - вычитание из значения регистра константы с учетом заема

SBIW Rdd,const - вычитание из 16-разрядного регистра (R24,R26,R28,R30) константы (0-63)
DEC Rd - декрементация значения регистра (не влияет на флаг переноса)
NEG Rd - изменяет знак числа хранимого в Rd (Rd=0-Rd)
COM Rd - Rd=$FF-Rd

Команды сравнения CP Rd,Rs; CPC Rd,Rs и CPI Rd,const выполняют сравнение между двумя регистрами, регистрами с учетом переноса, а также регистром и константой соответственно. CPI работает только с 16-ю старших РОН, константа – 8 разрядов.

TST Rd – проверка битов: Z=((Rd or 0)==0); N=(Rd>>7); V=0; S=N
MUL Rd,Rs – R1:R0=Rd*Rs; C=((Rd*Rs)>>15)

Команды логического умножения: AND Rd,Rs; ANDI Rd,const; CBR Rd,const выполняют соответственно Rd=Rd&Rs; Rd=Rd&const; Rd=Rd&(not const). ANDI и CBR работают только с 16-ю старших РОН, константа 8 бит.

Команды логического сложения OR Rd,Rs; ORI Rd,const; SBR Rd,const выполняют соответственно Rd=Rd|Rs; Rd=Rd|const; Rd=Rd|(const). ORI и SBR работают только с 16-ю старших РОН, константа 8 бит.

Команда «исключающего или» EOR Rd,Rs - Rd=Rd xor Rs

Команды CLR Rd; SER Rd – устанавливают все биты регистра соответственно в нули и единицы.

SWAP Rd - меняет местами старшую и младшую тетрады регистра Rd
BST Rd, bit - передает указанный по номеру бит из регистра Rd в бит *T* регистра *SREG*
BLD Rd, bit - передает указанный по номеру бит из бита *T* регистра *SREG* в регистр Rd
BCLR bit - сбрасывает указанный по номеру бит в регистре *SREG* (см табл.3.2)
BSET bit - устанавливает указанный по номеру бит в регистре *SREG* (см табл.3.2)

Таблица 3.2 – Соответствие номера бита *SREG* битовым командам

bit	Бит регистра *SREG*	Синоним	
		сброс бита	уст-ка бита
000	Флаг переноса	clc	sec
001	Флаг нуля	clz	sez
010	Флаг отрицательного результата	cln	sen
011	Флаг переполнения	clv	sev
100	Флаг знака	cls	ses
101	Флаг промежуточного переноса	clh	seh
110	Временный бит	clt	set
111	Флаг разрешения прерываний	cli	sei

CBI ioreg, bit – сбрасывает указанный по номеру бит порта ioreg (только младшие 32 порта ввода-вывода)
SBI ioreg, bit – устанавливает указанный по номеру бит порта ioreg (только младшие 32 порта ввода-вывода)

Команды LSL Rd и LSR Rd – логический сдвиг регистра на один разряд влево и вправо соответственно, заполнение нулями, выдвигаемый бит передается во флаг переноса.

Команды ROL Rd и ROR Rd – циклический сдвиг регистра на один разряд влево и вправо соответственно с учетом флага переноса как 9-го разряда.

ASR Rd – арифметический сдвиг регистра вправо на один разряд – то же что и LSR, но заполнение знаковым разрядом.

Резюме: Все команды выполняются за 1 цикл, исключая: MUL, CBI, SBI – за 2 цикла.

Команды ветвления

rjmp Label – безусловный относительный переход на метку Label (±2 кб) //PC=PC+1+offs (2 байт)
rcall Label – вызов подпрограммы Label (±2 кб) //push PCL; push PCH; PC=PC+1+offs (2 байта)
jmp Label – безусловный переход на метку Label (4 Мб) //PC=offs (4 байта)
call Label – вызов подпрограммы Label (4 Мб) //push PCL; push PCH; PC=offs (4 байта)

ijmp – безусловный переход по значению индексного регистра Z (±2 кб) //PC=Z (2 байта)
icall – вызов подпрограммы по значению регистра Z //push PCL; push PCH; PC=Z (2 байта)
ret – возврат из подпрограммы //pop PCH; pop PCL (2 байта)
reti – возврат из подпрограммы прерывания //pop PCH; pop PCL; SREG.I=1 (2 байта)

Условные переходы (±63 байта)

brbc bit, Label – переход на метку Label если *SREG*.bit=0 (2 байта) //см табл. 3.3
brbs bit, Label – переход на метку Label если *SREG*.bit=1 (2 байта) //см табл. 3.3

Таблица 3.3 – Соответствие номера бита *SREG* команде ветвления

Бит состояния регистра SREG	Номер бита	Синоним команды	
		бит=0	бит=1
Флаг переноса	000	brcc, brsh	brcs, brlo
Флаг нуля	001	brne	breq
Флаг отрицательного результата	010	brpl	brmi
Флаг переполнения	011	brvc	brvs
Флаг знака	100	brge	brlt
Флаг промежуточного переноса	101	brhc	brhs
Временный бит	110	brtc	brts
Флаг разрешения прерываний	111	brid	brie

Команды пропуска

sbic RegIO, bit – пропустить следующую команду если RegIO.bit=0
sbis RegIO, bit – пропустить следующую команду если RegIO.bit=1
(*) sbic, sbis могут работать только с младшими 32 портами
sbrc RG, bit – пропустить следующую команду если RG.bit=0
sbrs RG, bit – пропустить следующую команду если RG.bit=1
cpse Rd,Rs – пропустить следующую команду если (Rd-Rs)=0

Резюме: Команды rjmp, rcall, jmp, icall выполняются за 3 цикла
call, ret, reti – 4 цикла; ijmp – 2 цикла; cpse – 1 цикл (не изменяет флагов !)
Остальные команды выполняются за 1-2 цикла (условие не выполняется – 1 цикл, условие выполняется и след. команда не jmp, call – 2 цикла, иначе – 3 цикла).

Прочие команды

NOP – ничего не делать в течении 1 цикла
WDR – сброс сторожевого таймера (1 цикл)
SLEEP – ожидать прихода прерывания (1 цикл)

Примеры написания программ

1. Реализация целочисленной КИХ фильтрации
   Пусть дан оцифрованный сигнал (4 бита на точку со знаком) в памяти по адресу $60 и длиной 50 байт. Коэффициенты КИХ фильтра длиной 7 хранятся в ПЗУ и представляют собой числа со знаком квантованные в 4 бита. Выполнить фильтрацию сигнала с округлением результата до 8 бит со знаком и разместить его на месте входного сигнала.
   Общие положения:
   - регистр Z указывает на коэффициенты фильтра;
   - регистр X указывает на исходные данные и используется для сохранения результата;
   - для представления результата свертки необходимо (3 бита исх. данные)+(3 бита коэф.)+(1 бит знака)+(3 бита на сумму свертки)=10 бит – округление результата на 2 бита;
   - R16 зарезервируем для организации внешнего цикла;
   - пару R18:R17 отведем для вычисления свертки;
   - R19 зарезервируем для организации внутреннего цикла.

……
LDI R26, $60; загружаем указатель на исх. данные – X

```
LDI R27, 0;  или EOR R27,R27 или CLR R27
LDI R16, $60+50-7; адрес следующей ячейки за последним элементом массива исх. данных
L_FIR_ext:
    LDI R30, coef_FIR&$ff; загружаем указатель на коэффициенты фильтра – Z
    LDI R31, coef_FIR>>8
    LDI R17, 0; мл. часть свертки
    LDI R18, 0; ст. часть свертки
    LDI R19, 7; длина фильтра
    PUSH R26; сохранили значение указателя X (старший байт – всегда ноль, т.к. ОЗУ 128 байт)
    L_FIR_int:
        LPM; R0=коэф. фильтра
        INC R30; Z=Z+1 – к следующему коэффициенту
        LD R1,X+; загрузка отсчета сигнала и инкремент указателя
        MUL R0,R1; R1:R0=R0*R1
        ADD R17,R0
        ADC R18,R1
        DEC R19
        BRBC 1, L_FIR_int; переход если SREG.Z=0 (аналог: brne L_FIR_int)
    POP R26; восстановили значение указателя X
    ASR R18; сдвиг вправо ст. части свертки, выдвинутый бит – во флаге переноса
    ROR R17; сдвиг вправо мл. части свертки с заполнением из флага переноса
    ASR R18
    ROR R17; округленный результата свертки в R17
    ST R17,X+; сохр. результат на место исх. данных и инкремент указателя
    CPSE R28, R16; пропустить след. команду если X>=$60+50-7
                        (аналог CPI R28,const и brne label – дальность 63 байта против 2 кб)
    RJMP L_FIR_ext
……
coef_FIR:
    db 0,1,2,3,4,5,6; коэффициенты фильтра
……
```

2. Поиск максимального элемента массива данных.

Пусть имеется массив данных. Найти максимальный элемент массива.

Общие положения:

- массив расположен в памяти МК по адресу $60 и имеет длину 30 байт;
- регистр X указывает на текущий элемент массива;
- регистр R16 хранит максимальное значение массива.

```
……
LDI R26, $60; загрузили указатель на данные
LDI R27, 0
LD R16, X+; загружаем первый элемент, считая его максимальным, и инкрементируем
указатель
LDI R18, $60+30; адрес следующего за последним элементом массива
L_loop:
    LD R17, X+; загружаем очередной элемент массива и инкрементируем указатель
    CP R17, R16
    BRBS 4, L_next; переход если знак результата – отрицательный
        MOV R16,R17
    L_next:
    CPSE R26, R18
    RJMP L_loop
; R16 содержит максимальный элемент массива….
```

3.3 Аппаратные возможности МК

3.3.1 ТАКТОВЫЙ ГЕНЕРАТОР

Тактовый генератор представляет собой устройство, которое обеспечивает работу МК как самостоятельного устройства без навесных элементов, либо от внешнего генератора (сигнал подается на вход *XTAL*1), либо от внутреннего с подключением внешнего кварцевого резонатора к выводам *XTAL*1 и *XTAL*2.

3.3.2 СТОРОЖЕВОЙ ТАЙМЕР

Сторожевой таймер – устройство, предназначенное для перезапуска МК в случае его зависания. Структурная схема устройства представлена на рис.3.8. Таким образом, имеется возможность выключения *Watchdog*, задания восьми различных длительностей, программного перезапуска счетчика.

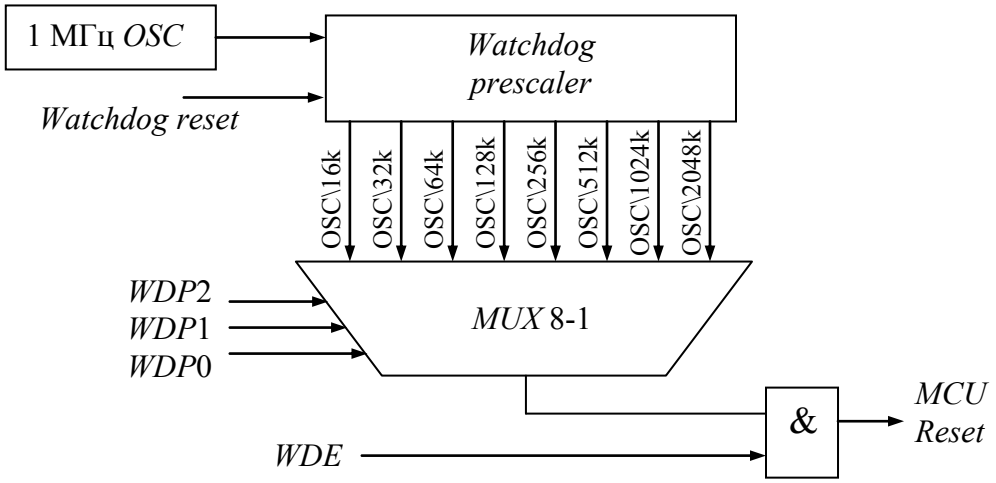


Рис. 3.8 – Структура *Watchdog*

Все эти функции реализуются с помощью управляющего регистра *WDTCR*. Перед включении *Watchdog* таймера необходимо его сбросить!!!

	7	6	5	4	3	2	1	0
WDTCR $21 ($41)	-	-	-	*WDTOE*	*WDE*	*WDP*2	*WDP*1	*WDP*0
	R	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

WDE – (*Watchdog Enable*) установка в «1» разрешает работу *Watchdog*, установка «0» запрещает. Однако для выключения *Watchdog* необходимо установить в «1» бит разрешения *WDTOE*, значение которого автоматически обнуляется через 4 такта.

WDP2-WDP0 – (*Watchdog prescaler*) биты управляющие коэффициентом деления для задания различных интервалов сброса МК – см. табл.3.4.

Таблица 3.4 – Значения интервалов до сброса МК

*WDP*2	*WDP*1	*WDP*0	Делитель, циклов	Период сброса при V_{cc}=3 В	Период сброса при V_{cc}=5 В
0	0	0	16k	47 мс	15 мс
0	0	1	32k	94 мс	30 мс
0	1	0	64k	0,19 с	60 мс
0	1	1	128k	0,38 с	0,12 с
1	0	0	256k	0,75 с	0,24 с
1	0	1	512k	1,5 с	0,49 с
1	1	0	1024k	3,0 с	0,97 с
1	1	1	2048k	6,0 с	1,9 с

3.3.3 ТАЙМЕРЫ

Как уже отмечалось, тактирование таймеров может происходить разными сигналами. На рис.3.9 представлена структурная схема, из которой видны пути сигналов тактирующих счетчики таймеров.

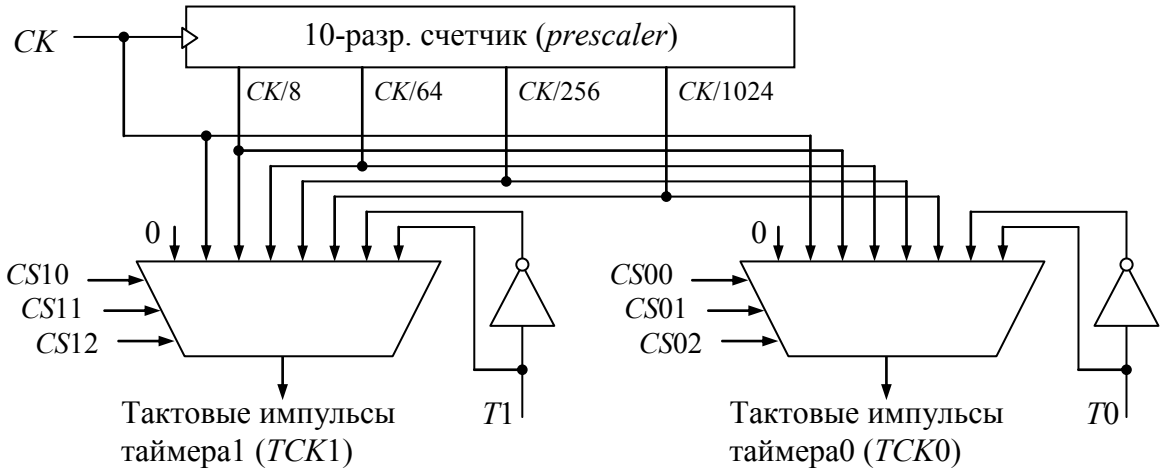


Рис. 3.9 – Структурная схема цепей тактирования таймеров МК

В данной модели МК реализованы два таймера на основе 8-разрядного счетчика и 16-разрядного счетчика. Оба счетчика могут работать только на одинаковой частоте (делитель один). Однако могут тактироваться разными внешними синхросигналами.

8-разрядный таймер, структурная схема которого показана на рис.3.10, с точки зрения программного управления имеет:

- регистр управления (*TCCR*0);
- флаг переполнения (*TIFR.TOV*0);
- флаг управления прерыванием при переполнении счетчика (*TIMSK.TOIE*0).

При тактировании счетчика внешним сигналом активным является передний фронт.

	7	6	5	4	3	2	1	0
TCCR0 $33 ($53)	-	-	-	-	-	*CS*02	*CS*01	*CS*00
	R	*R*	*R*	*R*	*R*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

CS02-CS00 – выбор источника тактирования счетчика – табл.3.5 (также см. рис.3.9)

Таблица 3.5 – Коэффициент деления частоты таймера 0

*CS*02	*CS*01	*CS*00	Описание
0	0	0	стоп (запрет счета)
0	0	1	*CK*
0	1	0	*CK*/8
0	1	1	*CK*/64
1	0	0	*CK*/256
1	0	1	*CK*/1024
1	1	0	внешний вывод *T*0, задний (*falling*) фронт
1	1	1	внешний вывод *T*0, передний (*rising*) фронт

(*) счетчик будет считать от внешнего сигнала, даже если соответствующий *pin* сконфигурирован как выход.

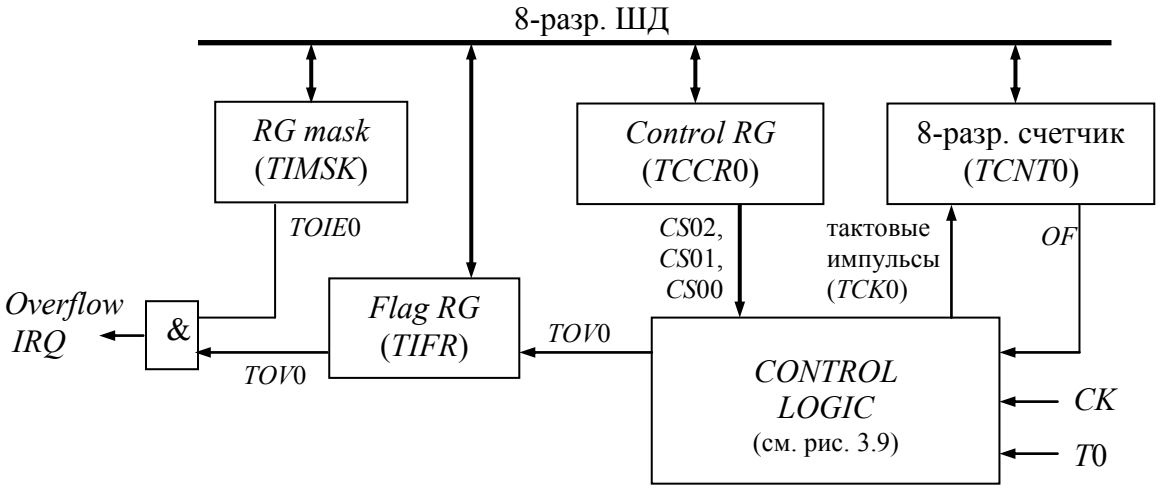


Рис. 3.10 – Структурная схема таймера 0

16-разрядный таймер, структурная схема которого показана на рис.3.11, с точки зрения программного управления имеет:

- регистры управления (*TCCR*1*A*, *TCCR*1*B*);
- флаг переполнения и его маска прерывания (*TIFR.TOV*1, *TIMSK.TOIE*1);
- регистр входного захвата (*ICR*1);
- флаг захвата и его маска прерывания (*TIFR.ICF*1, *TIMSK.TICIE*1);
- регистр сравнения (*OCR*1*A*);
- флаг сравнения и его маска прерывания (*TIFR.OCF*1*A*, *TIMSK.OCIE*1*A*).

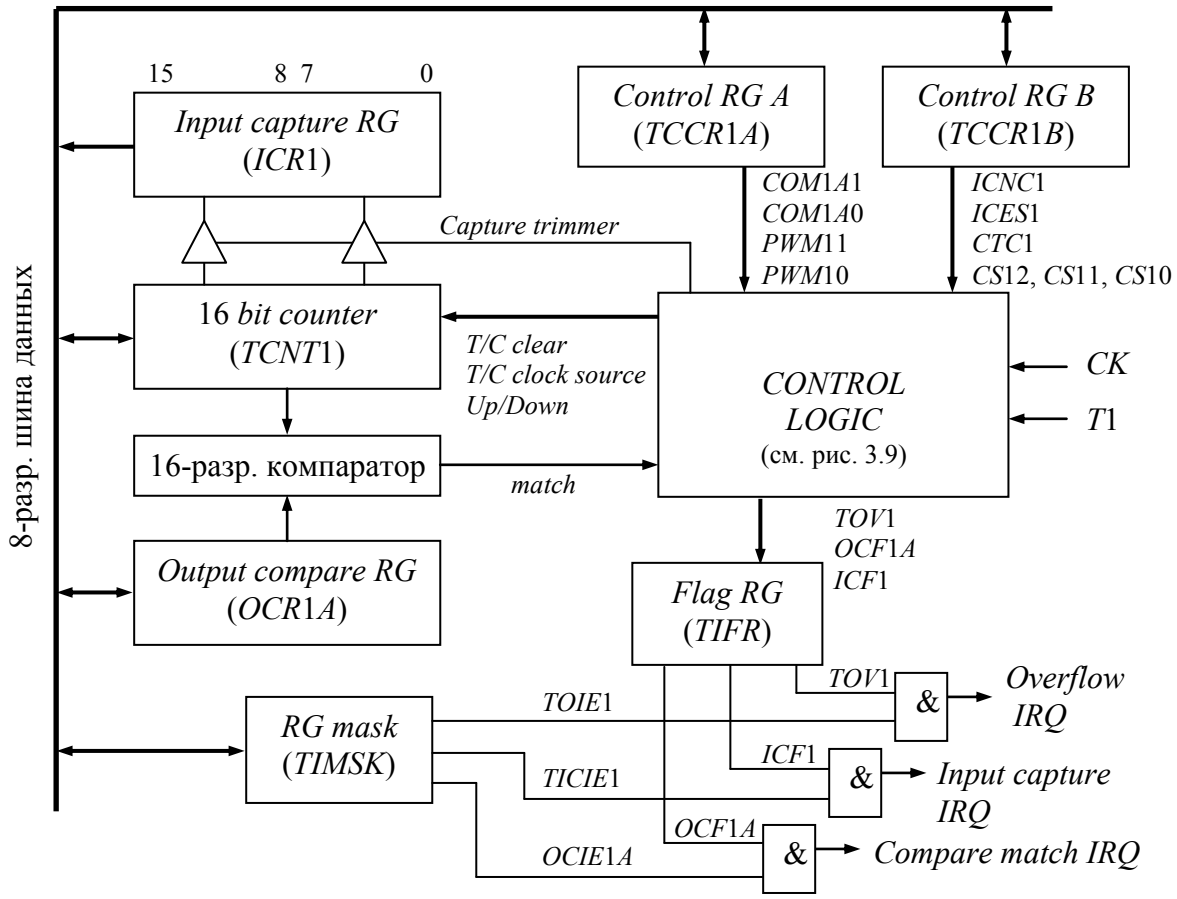


Рис. 3.11 – Структурная схема таймера 1

Возможности таймера 1 значительно шире по сравнению с таймером 0. В частности, с помощью встроенных регистра сравнения и компаратора возможно, чтобы счетчик работал как делитель на любой коэффициент (т.е. досчитывал от начального значения до значения регистра сравнения, сбрасывался, вызывая прерывание по необходимости, и начинал сначала). Таким образом, имеется возможность реализации генератора обеспечивающего широтно-импульсную модуляцию (8, 9, 10 бит).

Также имеется возможность аппаратного измерения длительностей внешних процессов, т.е. запустив счетчик на счет, схемы захвата могут зафиксировать значение счетчика в момент прихода внешнего сигнала. Причем для выделения активного сигнала (фронт или уровень) в МК встроена специальная схема, показанная на рис.3.12.

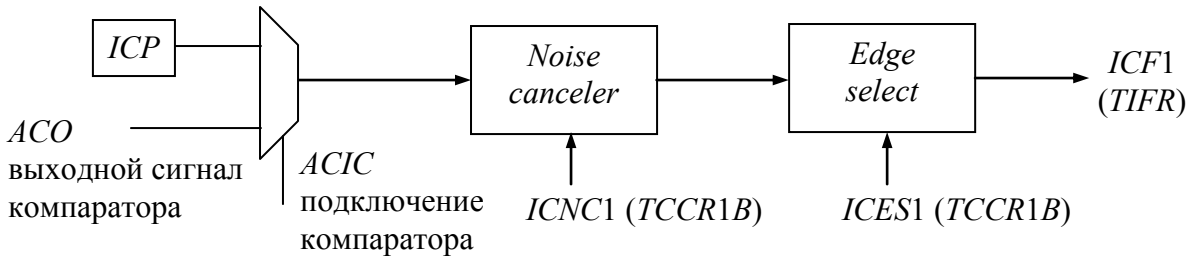


Рис. 3.12 – Структурная схема цепей захвата

Рассмотрим назначение бит управляющего регистра таймера 1.

	7	6	5	4	3	2	1	0
TCCR1A $2F ($4F)	*COM*1*A*1	*COM*1*A*0	-	-	-	-	*PWM*11	*PWM*10
	R/W	*R/W*	*R*	*R*	*R*	*R*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

COM1A1, **COM1A0** – (*Compare Output Mode* 1) определяют состояние внешнего вывода *OC*1 (*pin* 15) в момент совпадения значения счетчика и регистра сравнения:

0 0 - таймер отключен от вывода *OC*1;
0 1 - состояние линии *OC*1 изменяется на противоположное;
1 0 - на выходе *OC*1 устанавливается низкий уровень (лог. 0);
1 1 - на выходе *OC*1 устанавливается высокий уровень (лог. 1);
(*) Отметим, что в режиме ШИМ эти разряды имеют другой смысл.

PWM11, **PWM10** – (*Pulse Width Modulator*) выбирают режим ШИМ:

0 0 - ШИМ выключен;
0 1 - ШИМ 8 бит;
1 0 - ШИМ 9 бит;
1 1 - ШИМ 10 бит.

	7	6	5	4	3	2	1	0
TCCR1B $2E ($4E)	*ICNC*1	*ICES*1	-	-	*CTC*1	*CS*12	*CS*11	*CS*10
	R/W	*R/W*	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

ICNC1 – (*Input Capture*1 *Noise Canceler*) если установлен в «0», то функция подавления шумов отключена (первый же активный уровень сигнала приведет к срабатыванию триггера); если «1» - функция включена (для фиксации триггером входного сигнала необходимо чтобы активный его уровень был зафиксирован на протяжении 4 тактов);

ICES1 – (*Input Capture*1 *Edge Select*) определяет активный фронт сигнала захвата; 0 – задний (падающий) фронт; 1 – передний (нарастающий);

CTC1 – (*Clear Timer/Counter*1 *on Compare match*) если установлен в «1», то при равенстве значений счетчика и регистра сравнения *OCR*1*A* значение счетчика обнуляется (сравнение выполняется за 1 цикл независимо от коэффициента деления) кроме ШИМ режима;
CS12, CS11, CS10 – (*Clock Select*) определяют коэффициент деления частоты, а также выбирают источник тактирования в соответствии с табл.3.5 (заменяя индексы *CS*0*x* на *CS*1*x* и *T*0 на *T*1).

Назначение разрядов текущего значения счетчика таймера очевидно, однако поскольку последний 16-разрядный, а шина 8-разрядная, то есть определенные условия, позволяющие одновременно прочитать/записать значение счетчика. Для этих целей в МК встроен дополнительный регистр, обеспечивающий временное хранение одного байта при доступе к 16-разрядному значению регистра таймера:

- для записи слова (ядро МК в таймер) необходимо записывать сначала старшую половину слова (т.е. в регистр *TCNT*1*H*), значение которого будет записано во временный регистр, а затем записывать младшую половину слова (т.е. в регистр *TCNT*1*L*), значение которого, объединяясь со значением временного регистра, одновременно записывается в счетчик;
- для чтения слова (таймер в ядро МК) необходимо сначала читать младший регистр *TCNT*1*L* при этом значение старшей половины будет автоматически одновременно с младшей размещено во временном регистре, а следующее чтение из старшей половины приведет к передаче значения из временного регистра.

	7	6	5	4	3	2	1	0
TCNT1H $2D ($4D)	b15	b14	b13	12	b11	b10	b9	b8
TCNT1L $2C ($4C)	b7	b6	b5	b4	b3	b2	b1	b0
Доступ	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Аналогичным образом происходит обращение к регистру сравнения (*OCR*1) и регистру захвата (*ICR*1 только чтение). При этом используется один и тот же временный регистр!

	7	6	5	4	3	2	1	0
OCR1AH $2B ($4B)	b15	b14	b13	12	b11	b10	b9	b8
OCR1AL $2A ($4A)	b7	b6	b5	b4	b3	b2	b1	b0
Доступ	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
ICR1H $25 ($45)	b15	b14	b13	12	b11	b10	b9	b8
ICR1L $24 ($44)	b7	b6	b5	b4	b3	b2	b1	b0
Доступ	*R*	*R*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

Далее, рассмотрим таймер 1 в режиме ШИМ. Когда включен режим ШИМ регистр *OCR*1 конфигурируется как 8-, 9- или 10-разрядный регистр, счетчик свободно считает от нуля до максимального значения и обратно к нулю при этом частота сигнала определяется в соответствии с табл. 3.6.

Таблица 3.6 – Выбор частоты ШИМ сигнала

PWM разрешение	*TOP* значение таймера	Частота
8 бит	255 (0FFh)	$f_{TC1}/510$
9 бит	511 (01FFh)	$f_{TC1}/1022$
10 бит	1023 (03FFh)	$f_{TC1}/2046$

При совпадении значения счетчика со значением регистра *OCR*1 выход ШИМ устройства *OC*1 (*PB*3) изменяет свое состояние в соответствии с табл.3.7 (битами регистра *TCCR*1).

Таблица 3.7 – Функциональность сигнала *OC*1

*COM*1*A*1	*COM*1*A*0	Действия
0	0	Выход *OC*1 не подключен
0	1	
1	0	*OC*1 сбрасывается в «0» при совпадении и счете вверх, устанавливается в «1» при совпадении и счете вниз
1	1	Наоборот (инвертированный ШИМ)

(*) Начальное состояние выхода *OC*1 неопределено.

(*) Запись *OCR*1*A* происходит через временный регистр – необходимо для синхронизации, полноту формирования импульса.

Пример работы с таймером – тахометр.

Пусть события (изменение сигнала) поступает на вход *INT*0 (вход *pin*6), таймер 0 осуществляет отсчет интервалов времени. Глобальная переменная R16 считает число событий, глобальная переменная *R*17 считает число полных циклов таймера (до 16). Отношение значения *R*16 к интервалу времени *T*=(*n**256**T*0) равно числу событий в единицу времени. Причем, выбирая делитель частоты таймера и параметр *n* так, чтобы величина *T* была близка к 1 сек., *R*16 содержит измеряемую величину (*n*=16, делитель 1024).

```
rjmp main;  вектор сброса
rjmp couse; вектор обработчика прерывания INT0
nop
nop
nop
rjmp full; вектор обработчика прерывания переполнения таймера 0
nop
nop
nop
nop
nop
main:
ldi R16, 0; иниц. глобальных переменных
ldi R17, 16
; настраиваем стек в конец памяти
ldi R20, $60+63
out $3d, R20; SPL=$60+63
; инициализируем таймер 0, разрешаем прерывания
bset 7; SREG.7=1 – разрешить IRQ (аналог sei)
; разрешаем прерывания таймера 0 по переполнению: TIMSK.1=1
in R20, $39; читаем TIMSK
ori R20, 2; установили бит 1
out $39, R20
; разрешаем прерывания по INT0: GIMSK.6=1
```

```
in R20, $3b; читаем GIMSK
ori R20, 64; установили бит 6
out $3b, R20
; выбираем тип активного сигнала на линии INT0 – низкий уровень
in R20, $35; читаем MCUCR
andi R20, 255-3; сбросили два мл. бита
out $3b, R20
; Устанавливаем делитель таймера 0, запуская счет
ldi R20, 5; установили биты CS02, CS01, CS00 – коэф. деления 1024
out $33, R20; в TCCR0
loop:
    brbc 1, loop; переход если SREG.Z=0 (R17>0)
; R16 содержит число событий в единицу времени
……
; например, вывод на индикатор (экран)
……
; реинициализация глобальных переменных
ldi R17,16
ldi R16,0
rjmp main
; обработчики прерываний
cause:
    inc R16; зарегистрировали событие
    reti
full:
    dec R17
    reti
……
```

3.3.4 ПОРТЫ ВВОДА/ВЫВОДА И ИХ АЛЬТЕРНАТИВНЫЕ ФУНКЦИИ

Порт *B* является 8-разрядным с независимым программированием каждого вывода порта. Физически порт обеспечивается тремя адресами:

Data RG – **PORTB**

	7	6	5	4	3	2	1	0
PORTB $18 ($38)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Direction RG – **DDRB**

	7	6	5	4	3	2	1	0
DDRB $17 ($37)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Input pins RG – **PINB**

	7	6	5	4	3	2	1	0
PINB $16 ($36)	b7	b6	b5	b4	b3	b2	b1	b0
	R	*R*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Ниже в таблице представлено определение каждого *n* бита порта.

DDRBn	*PortBn*	*I/O*	*Pull-up*	Описание
0	0	*input*	-	Высокоомный вход
0	1	*input*	+	Подтянутый к *Vcc* вход
1	0	*output*		Выход – лог. 0
1	1	*output*		Выход – лог. 1

Альтернативные функции порта

Бит порта	Альтернативная функция
*PB*0	*AIN*0 - неинверсный вход аналогового компаратора
*PB*1	*AIN*1 - инверсный вход аналогового компаратора
*PB*3	*OC*1 - выход таймера/счетчика1 по совпадению (*compare match output*)
*PB*5	*MOSI* - выход данных *SPI* интерфейса
*PB*6	*MISO* - вход данных *SPI* интерфейса
*PB*7	*SCK* - линия тактирования *SPI* интерфейса

Также интерфейс *SPI* применяется при загрузке *Flash* памяти (программирование МК)

Для лучшего понимания функций порта рассмотрим его функциональную схему. На рис.3.13-3.18 показаны схемы драйверов выводов порта *B*, где
RD, *WD* – стробы чтения и записи регистра направления (*DDRB*);
RL, *WP* – стробы чтения и записи регистра порта (*PORTB*);
RP – строб чтения вывода порта;
Reset – внутренний сигнал сброса (установки нуля).

Нагрузочная способность портов 20 мА.

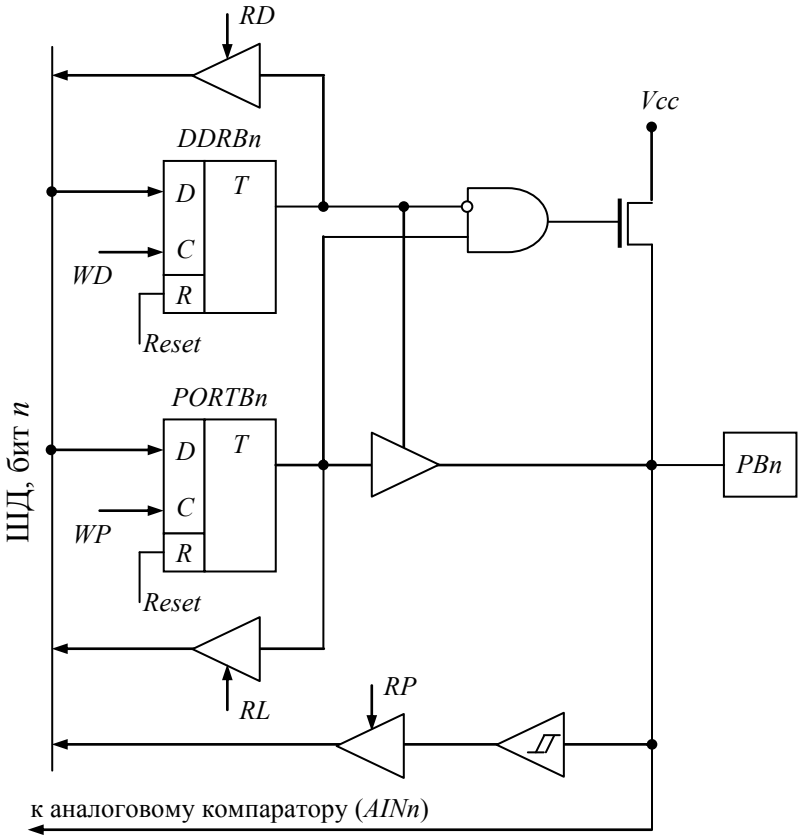


Рис. 3.13 – Схема линий порта *PBn* (*n*=0, 1)

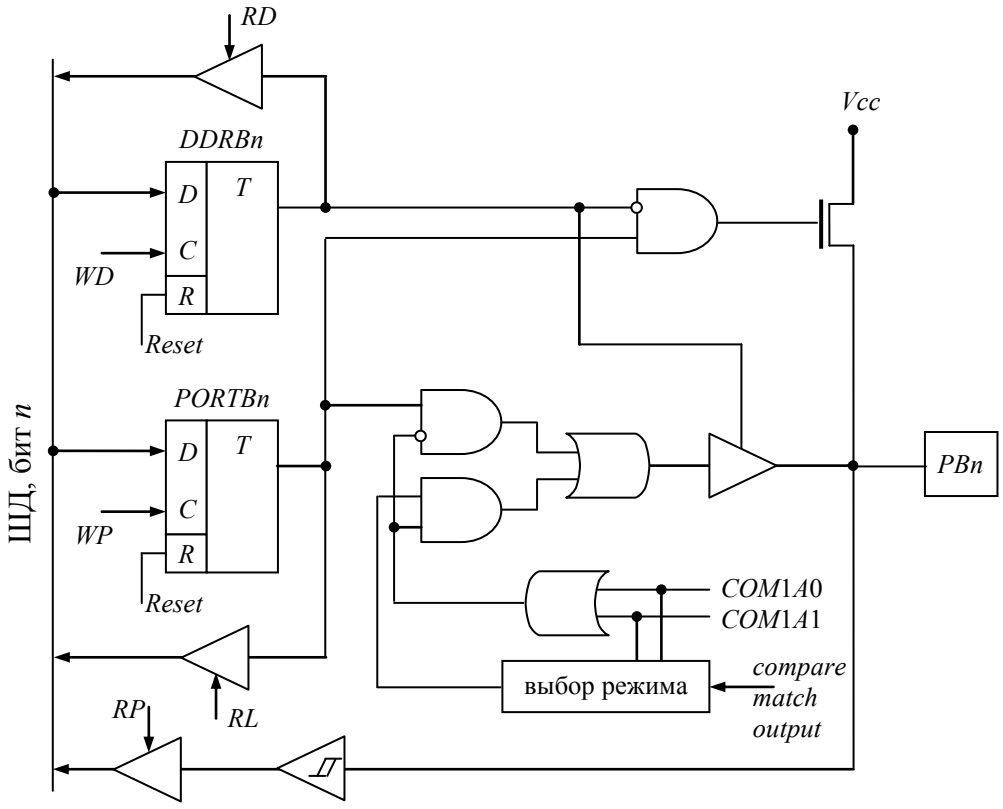


Рис. 3.14 – Схема линий порта *PBn* (*n*=3)

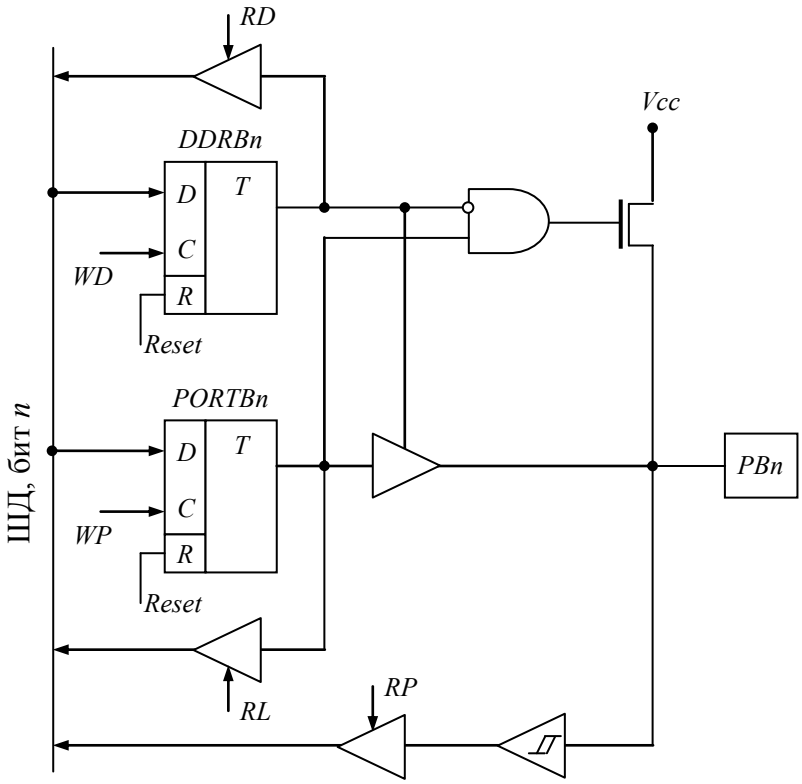


Рис. 3.15 – Схема линий порта *PBn* (*n*=2, 4)

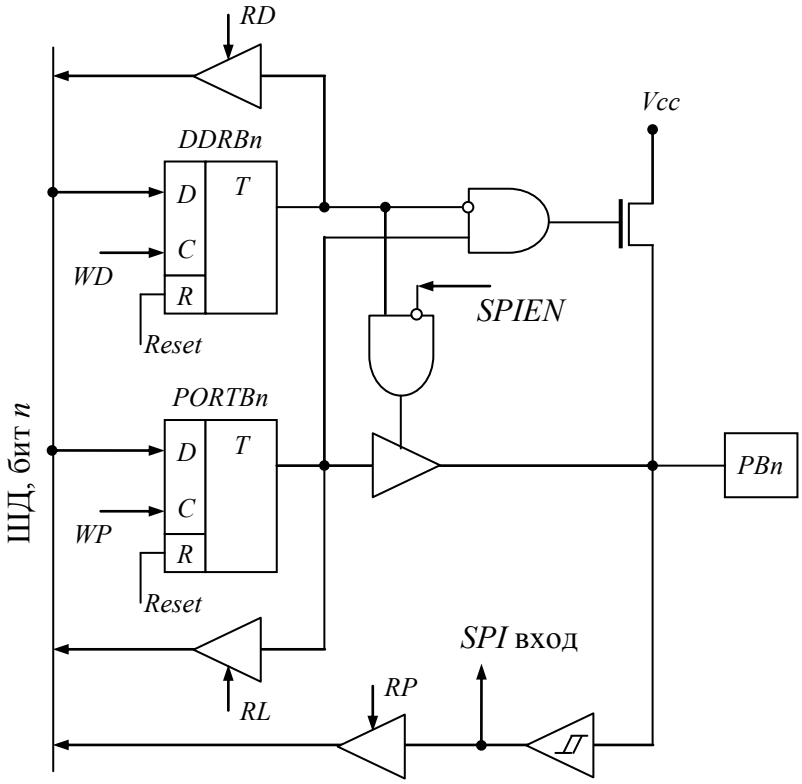


Рис. 3.16 – Схема линий порта *PBn* (*n*=5)

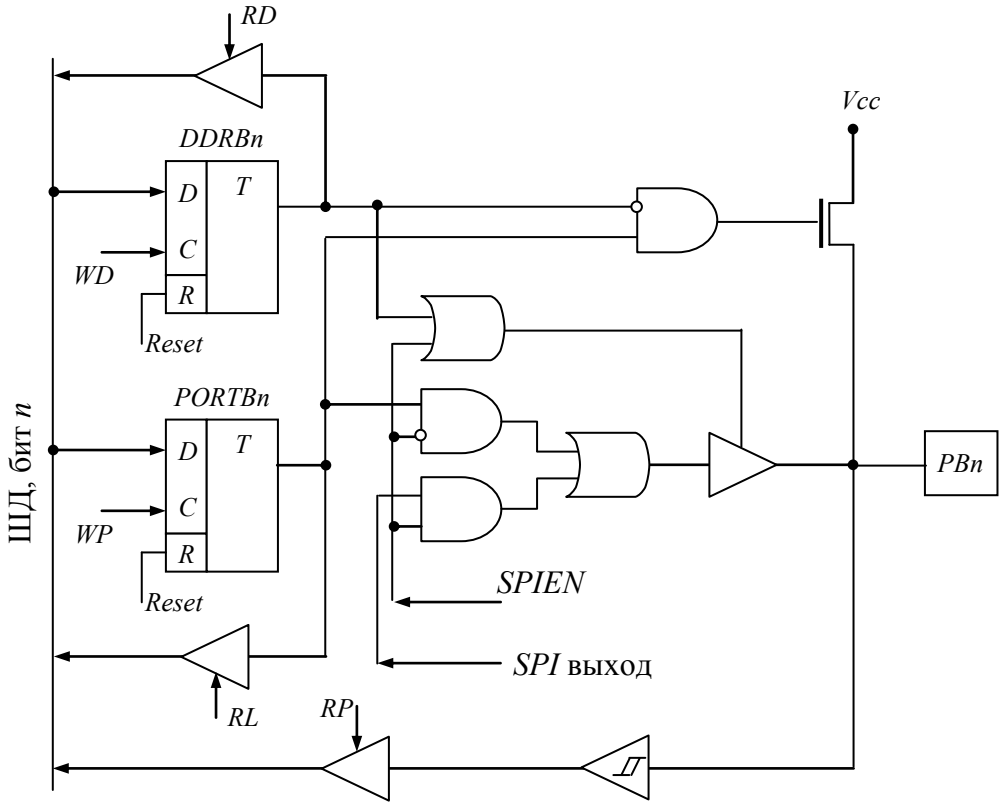


Рис. 3.17 – Схема линий порта *PBn* (*n*=6)

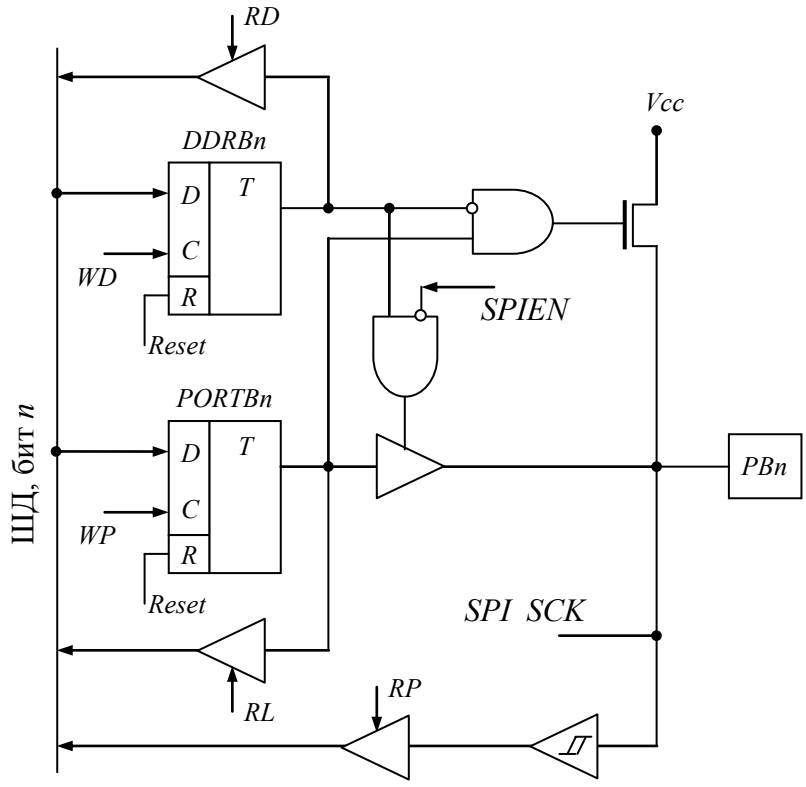


Рис. 3.18 – Схема линий порта *PBn* (*n*=7)

Порт *D* является 7-разрядным с независимым программированием каждого вывода порта. Физически порт обеспечивается тремя адресами:

Data RG – **PORTD**

	7	6	5	4	3	2	1	0
PORTD $12 ($32)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Direction RG – **DDRD**

	7	6	5	4	3	2	1	0
DDRD $11 ($31)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Input pins RG – **PIND**

	7	6	5	4	3	2	1	0
PIND $10 ($30)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Ниже в таблице представлено определение каждого *n* бита порта.

DDRDn	*PortDn*	*I/O*	*Pull-up*	Описание
0	0	*input*	-	Высокоомный вход
0	1	*input*	+	Подтянутый к *Vcc* вход
1	0	*output*		Выход – лог. 0
1	1	*output*		Выход – лог. 1

Альтернативные функции порта

Бит порта	Альтернативная функция
*PD*0	*RxD* - вход приемника *UART*
*PD*1	*TxD* - выход передатчика *UART*
*PD*2	*INT*0 - вход внешнего прерывания
*PD*3	*INT*1 - вход внешнего прерывания
*PD*4	*T*0 - вход таймера/счетчика0
*PD*5	*T*1 - вход таймера/счетчика1
*PD*6	*ICP* - вход схемы захвата таймера/счетчика1

Для лучшего понимания функций порта рассмотрим его функциональную схему. На рис.3.19-3.23 показаны схемы драйверов выводов порта *D*, где
RD, *WD* – стробы чтения и записи регистра направления (*DDRD*);
RL, *WP* – стробы чтения и записи регистра порта (*PORTD*);
RP – строб чтения вывода порта;
Reset – внутренний сигнал сброса (установки нуля).

Нагрузочная способность портов 20 мА.

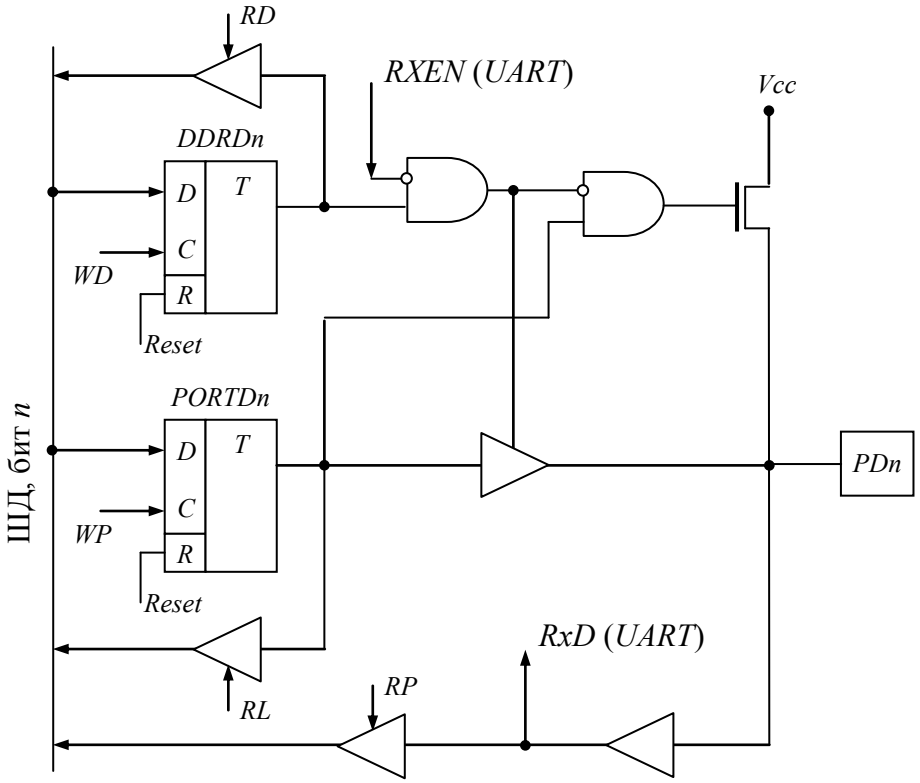


Рис. 3.19 – Схема линий порта *PDn* (*n*=0)

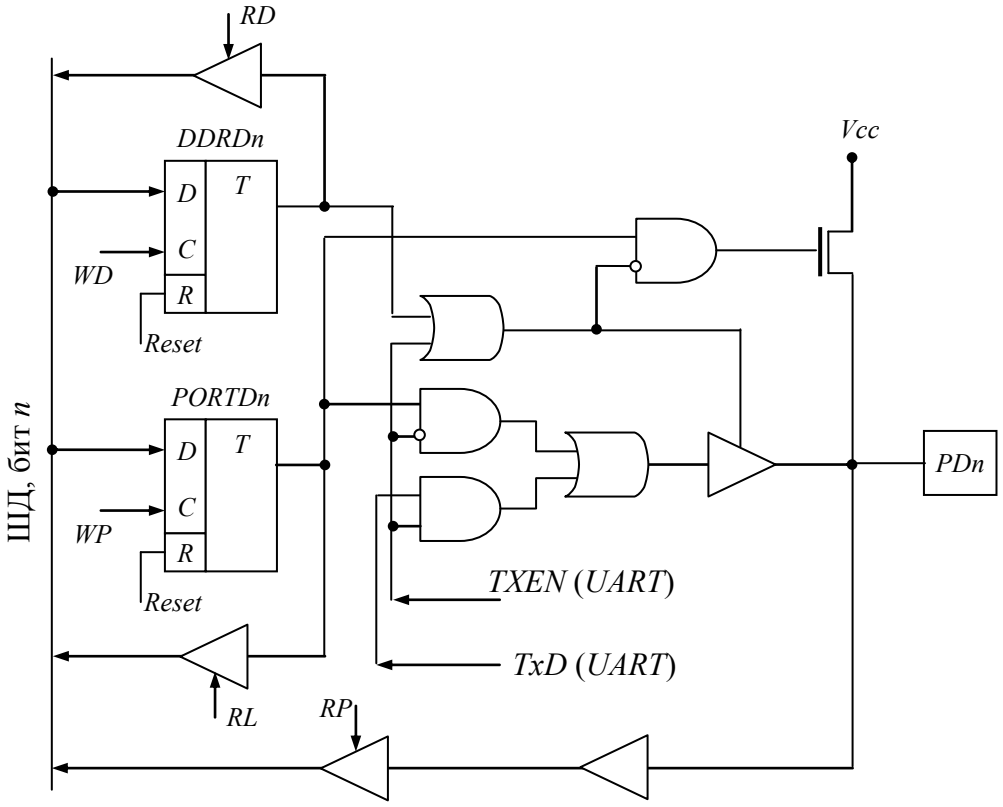


Рис. 3.20 – Схема линий порта *PDn* (*n*=1)

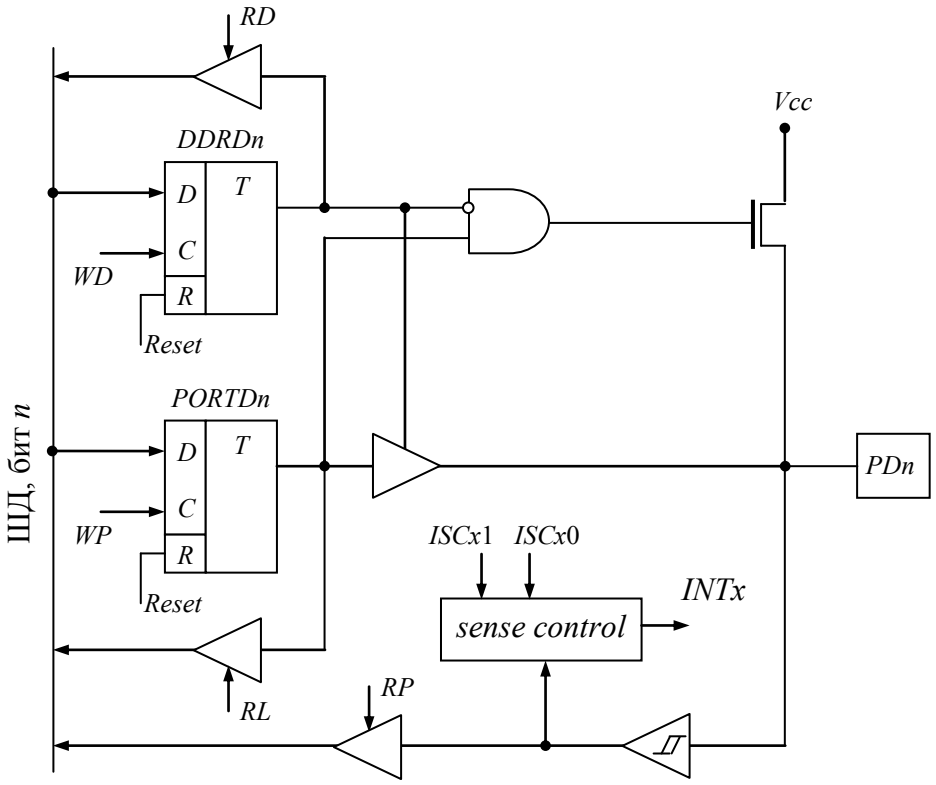


Рис. 3.21 – Схема линий порта *PDn* (*n*=2, 3)

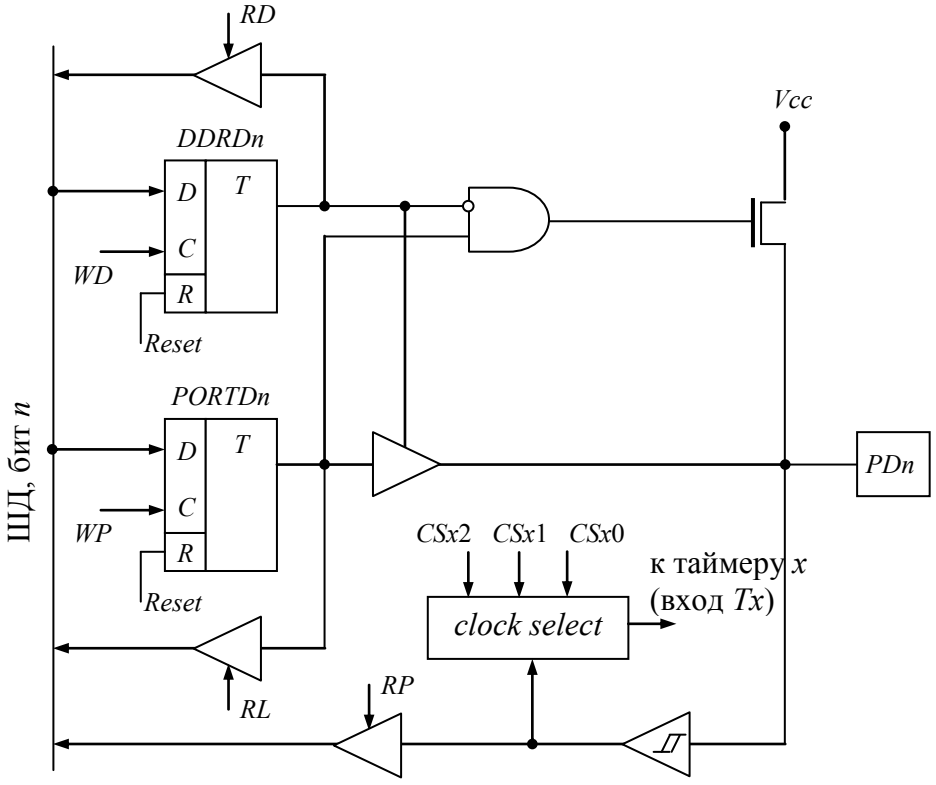


Рис. 3.22 – Схема линий порта *PDn* (*n*=4, 5)

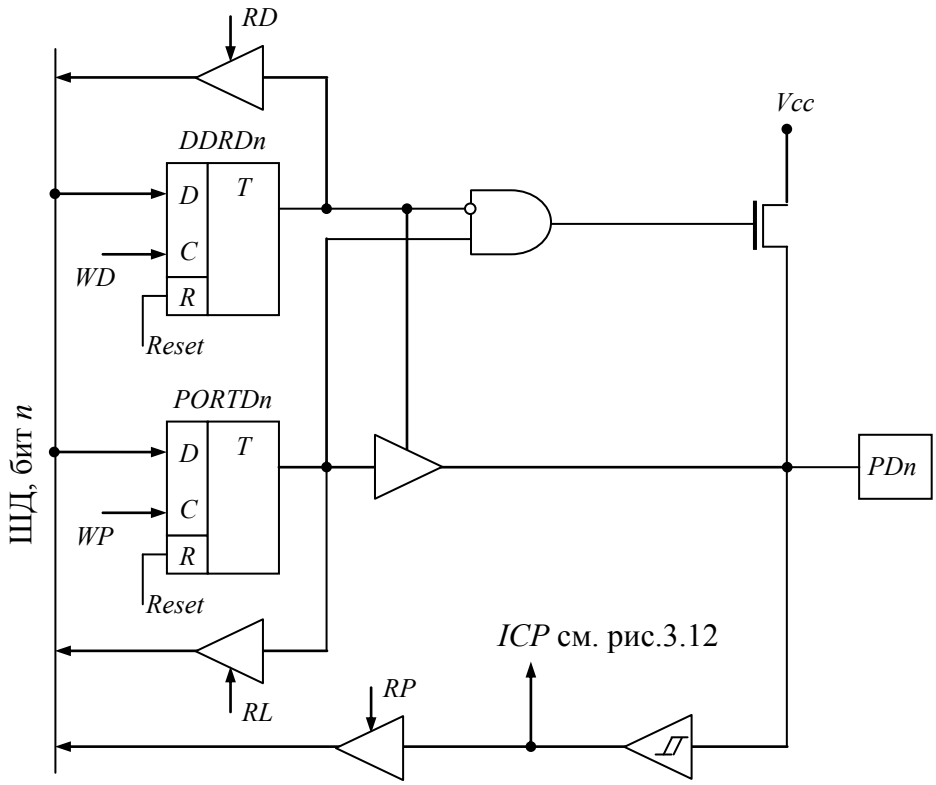


Рис. 3.23 – Схема линий порта *PDn* (*n*=6)

3.3.5 АНАЛОГОВЫЙ КОМПАРАТОР

Структурная схема компаратора представлена на рис.3.24.

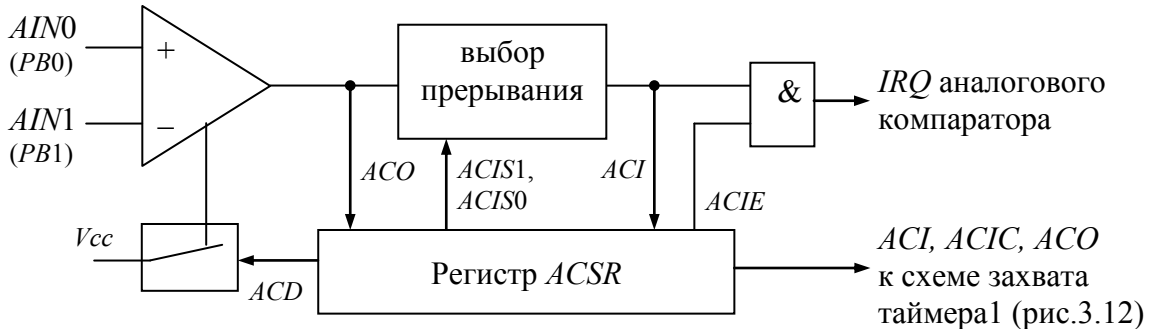


Рис. 3.24 – Структурная схема компаратора

Управление компаратором осуществляется с помощью одного регистра *ACSR* (*Analog Comparator Control and Status RG*).

	7	6	5	4	3	2	1	0
ACSR $08 ($28)	*ACD*	-	*ACO*	*ACI*	*ACIE*	*ACIC*	*ACIS*1	*ACIS*0
	R/W	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

ACD (*Analog Comparator Disable*) – если установлен в «1», то питание компаратора выключено, это снижает потребление МК.
ACO (*Analog Comparator Output*) – прямой выход ОУ.
ACI (*Analog Comparator Interrupt Flag*) – устанавливается при возникновении прерывания от компаратора, автоматически сбрасывается при выполнении процедуры обслуживания запроса, для программного сброса бита необходимо записать лог. «1».
ACIE (*Analog Comparator Interrupt Enable*) – разрешение прерываний от компаратора.
ACIC (*Analog Comparator Input Enable*) – если установлен в «1», то выход компаратора коммутируется на вход схемы захвата таймера 1 (иначе выход компаратора и таймер 1 не соединены).
ACIS1, ACIS0 (*Analog Comparator Interrupt Mode Select*) – определяют тип срабатывания компаратора в соответствии с табл. 3.8.

Таблица 3.8 – Выбор типа прерываний аналогового компаратора

*ACIS*1	*ACIS*0	Вид прерывания
0	0	Прерывание по переключению выхода (изменению *ACO*)
0	1	Зарезервировано
1	0	Прерывание по спадающему фронту *ACO* (из "1" в "0")
1	1	Прерывание по нарастающему фронту *ACO* (из "0" в "1")

Пример – мигание светодиода по таймеру (подключен к выводу *PB*7 и земле через *R*).

```
rjmp main
nop
nop
nop
nop
nop
rjmp switch
nop
nop
nop
nop
switch:
    sbis $18, 7;  если PB7=1 пропустить след. команду – светодиод включен, надо выкл.
      rjmp L01; светодиод выключен, надо включить
    cbi $18, 7; выключить светодиод
    reti
    L01:
    sbi $18, 7; включить светодиод
    reti
main:
ldi R16, $60+63
out $3d, R16; SPL=$60+63
ldi R16, $80
out $17, R16; DDRB.7=1 – выход, остальные – входы
clr R16
out $18, R16; PortB=0, начальное состояние – выключенный светодиод
ldi R16, 5
out $33, R16; делитель таймера 0  =1024
ldi R16, 2
out $39, R16; размаскировали IRQ таймера
sei; SREG.7=1
loop:
rjmp loop
```

3.3.6 ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД/ВЫВОД (*UART*)

*AT*90*S*2313 обеспечивает полный дуплекс при последовательном обмене данными. Основные возможности *UART* (*Universal Asynchronous Receiver and Transmitter*):

- данные 8 или 9 бит;
- встроенная фильтрация шумов;
- обнаружение перегрузки;
- обнаружение ошибки кадра;
- обнаружение ложного старт бита;
- независимые источники прерываний: передача закончена, регистр передачи пуст и прием окончен.

Структурная схема передатчика приведена на рис.3.25. Функционирование передатчика начинается после записи в регистр передачи *UDR* (*UART I/O Data RG*) собственно данных подлежащих передаче. Далее эти данные передаются в сдвиговый регистр:

- немедленно, если символ был записан после выдвижение в линию стоп бита предыдущего символа;
- после окончания передачи предыдущего символа (после передачи его стоп бита), если символ был записан во время передачи предыдущего.

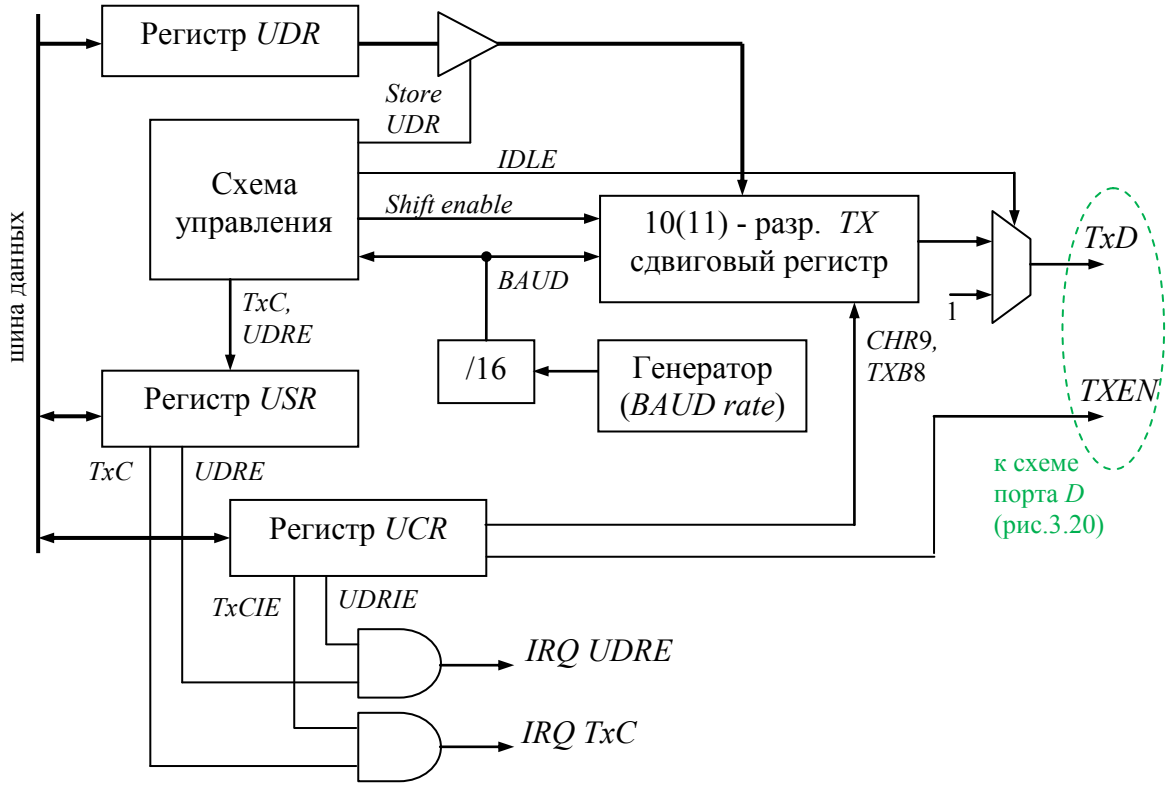


Рис. 3.25 – Структурная схема передатчика *UART*

Если сдвиговый регистр пуст, то после передачи данных из *UDR* в регистр сдвига бит *UDRE* (*UART Data RG Empty*) регистра статуса *USR* (*UART Status RG*) устанавливается в «1», что означает готовность передатчика к приему следующего символа для передачи в линию.

После размещения данных в регистре сдвига нулевой бит сбрасывается (старт бит), 9 или 10 бит устанавливается (стоп бит). Если выбрана длина символа в 9 бит (бит *CHR*9 регистра управления *UCR* установлен), то бит 0 регистра *UCR* (обозначен *TXB*8) предается в бит 9 регистра сдвига.

При передаче первым выдвигается старт бит, за ним – младший бит символа и т.д. После выдвижения стоп бита регистр сдвига заполняется новым символом из регистра *UDR*. Если этот регистр пуст, то бит *TXC* (*TX Complete Flag*) регистра *USR* устанавливается в «1».

Включением/выключением передатчика управляет бит *TXEN* регистра *UCR*. Если передатчик включен (*TXEN*=1) то вывод МК *PD*1 используется как выход передатчика *UART* независимо от настроек порта. В противном случае этот *pin* работает как порт.

Структурная схема приемника приведена на рис.3.26.

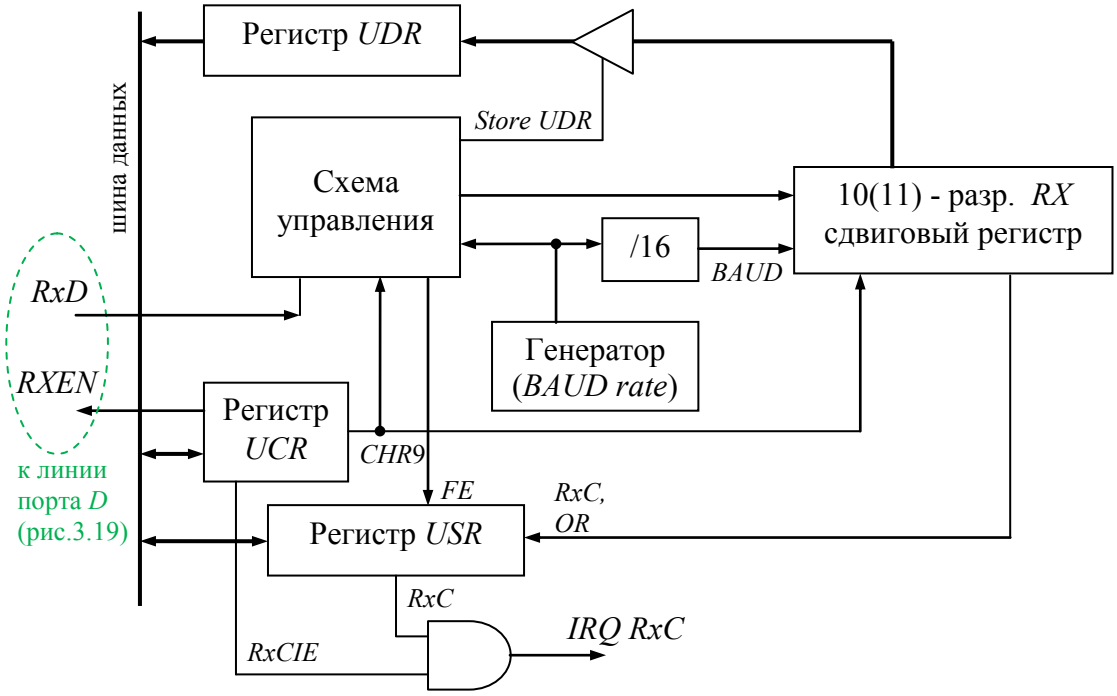


Рис. 3.26 – Структурная схема приемника *UART*

Входные цепи приемника непрерывно следят за состоянием линии с частотой дискретизации в 16 раз большей частоты передачи данных. Это позволяет отследить фронт с лог. «1» в лог. «0», т.е. обнаружить старт бит. Однако для начала приема этого недостаточно. Например, пусть 1 отсчет оказался низким уровнем, тогда приемник, продолжая наблюдать сигнал, считывает 8, 9 и 10 отсчеты. Если любые два или более этих отсчетов будут содержать лог. «1», то принятый фронт – помеха и приемник перейдет в режим поиска очередного спадающего фронта. Если после обнаружения фронта хотя бы два отсчета из трех (8, 9, 10 отсчеты) содержат низкие уровни, то принимаемая последовательность – данные, которые будут приниматься по тому же правилу: два из трех. Это обеспечивает дополнительную помехоустойчивость приема данных – рис.3.27, где "черточками" показаны моменты чтения линии *RxD*, длинные "черточки" соответствуют 8, 9, 10 отсчетам (нумерация с единицы).

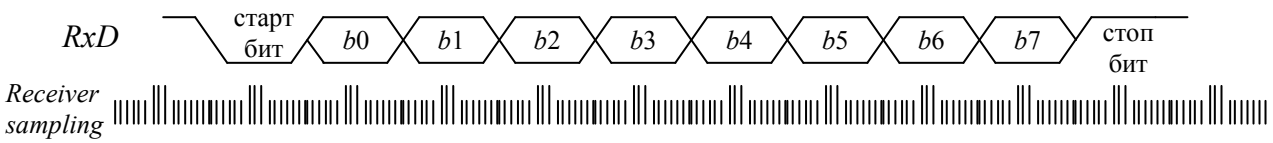


Рис. 3.27 – Мажоритарный принцип приема сигнала из цифрового канала

При окончании приема ожидается получение отсчетов содержащих лог. «1» (стоп бит). Если при этом из трех отсчетов хотя бы два содержат низкий уровень, то фиксируется ошибка кадра *FE* (*Framing Error*) и соответствующий бит регистра статуса *USR* устанавливается в «1».

В независимости от того был ли принят стоп бит или произошла ошибка при его приеме данные из сдвигового регистра приемника передаются в регистр *UDR*, а бит окончания приема *RXC* в регистре статуса *USR* устанавливается в «1».

Если при приеме установлен режим 9-разрядных данных (бит *CHR*9=1 регистра *UCR*), то бит *RXB*8 регистра *UCR* принимает значение бита 9 регистра сдвига.

Если при приеме очередного символа оказывается, что регистр *UDR* не был прочитан фиксируется ошибка перегрузки *OR* (*OverRun*) и соответствующий бит регистра *USR* устанавливается в «1». Это означает, что принятый символ не может быть передан из сдвигового регистра в *UDR* и его значение будет потеряно первым же обнаруженным фронтом старт бита. Значение бита *OR* буферизировано, т.е. обновляется при чтении регистра *UDR* (проверять ошибку перегрузки надо после чтения принятого символа).

Включением/выключением приемника управляет бит *RXEN* регистра *UCR*. Если приемник включен (*RXEN*=1) то вывод МК *PD*0 используется как вход приемника *UART* независимо от настроек порта. В противном случае этот *pin* работает как порт.

UART обслуживается четырьмя регистрами: *RG* данных приема/передачи, *RG* управления, *RG* статуса и *RG* управления скоростью обмена.

UDR (*UART I/O Data RG*) – регистр данных устройства.

	7	6	5	4	3	2	1	0
UDR $0C ($2C)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Под одним адресом физически скрыты два регистра – приемника по чтению и передатчика по записи.

USR (*UART Status RG*) – статусный регистр устройства.

	7	6	5	4	3	2	1	0
USR $0B ($2B)	*RxC*	*TxC*	*UDRE*	*FE*	*OR*	-	-	-
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R*	*R*	*R*
Начал. знач.	0	0	1	0	0	0	0	0

RxC (*UART Receive Complete*) – устанавливается в «1» когда принимаемый символ передан из сдвигового регистра в регистр данных *UDR*, установка бита происходит независимо от наличия ошибки кадра. Бит сбрасывается при чтении регистра. Если в регистре управления разрешены прерывания, то установка этого бита вызывает соответствующее прерывание.
TxC (*UART Transmitte Complete*) – устанавливается в «1» когда последний бит текущего символа, включая служебные биты, был выдвинут из сдвигового регистра в линию и регистр данных *UDR* не содержит нового символа для передачи. Если в управляющем регистре разрешены прерывания, то установка бита вызывает соответствующее прерывание. Бит сбрасывается автоматически при начале выполнении обработчика прерывания. Бит можно сбросить программно записью в него лог. «1».
UDRE (*UART Data RG Empty*) – устанавливается когда символ из регистра *UDR* передан в освободившейся сдвиговый регистр и означает готовность передатчика к приему очередного символа для передачи. Если в управляющем регистре разрешены прерывания, то установка этого бита вызывает соответствующее прерывание до тех пор пока он не будет сброшен. Сброс бита осуществляется записью в регистр *UDR*. При сбросе МК этот бит установлен, показывая готовность передатчика.
FE (*Framing Error*) – устанавливается когда на месте стоп бита принимаемого символа оказывается ноль.
OR (*Overrun*) – устанавливается когда в регистре *UDR* имеется не прочитанный символ и в сдвиговом регистре принят очередной символ. Бит буферизирован, что означает, что его обновление происходит при чтении регистра *UDR*, бит сбрасывается при перезаписи принятых данных в *UDR*.

UCR (*UART Control RG*) – регистр управления устройством.

	7	6	5	4	3	2	1	0
UCR $0A ($2A)	*RxCIE*	*TxCIE*	*UDRIE*	*RXEN*	*TXEN*	*CHR*9	*RXB*8	*TXB*8
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R*	*W*
Начал. знач.	0	0	0	0	0	0	1	0

RxCIE (*RX Complete Interrupt Enable*) – установка в «1» разрешает прерывание при окончании приема (по установке бита *RXC* статусного регистра *USR*).
TxCIE (*TX Complete Interrupt Enable*) – установка в «1» разрешает прерывание при окончании передачи (по установке бита *TXC* статусного регистра *USR*).
UDRIE (*UART Data RG Empty Interrupt Enable*) – установка в «1» разрешает прерывание при окончании передачи и пустоте регистра данных (по установке бита *UDRE* статусного регистра *USR*).
RXEN (*RX Enable*) – установка в «1» включает приемник. При сброшенном бите установка битов *RXC*, *OR*, *FE* невозможна. При сбросе этого бита влияние на эти разряды не оказывается.
TXEN (*TX Enable*) – установка в «1» включает передатчик. Если сброс бита произошел во время передачи символа, то передатчик не выключится до тех пор пока не будет передан текущий символ из сдвигового регистра и плюс все символы записанные в *UDR*.
CHR9 (9 *Bit Characters*) – установка в «1» устанавливает длину символа 9 бит плюс старт бит и стоп бит. Девятый символ принимается в разряде *RXB*8, а передается через разряд *TXB*8 управляющего регистра. Данный бит позволяет эмулировать дополнительный разряд данных или дополнительный стоп бит или бит четности.

UBRR (*UART Band Rate RG*) – регистр управления частотой приемопередатчика.

	7	6	5	4	3	2	1	0
UBRR $09 ($29)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Частота приемопередатчика определяется на основании формулы и нижеследующей таблицы.
$BAUD=F_{CK}/(16(UBRR+1))$, где F_{CK} – частота кварца.

Baud Rate	1 MHz		%Error	1.8432 MHz		%Error	2 MHz		%Error	2.4576 MHz		%Error
2400	UBRR=	25	0.2	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	63	0.0
4800	UBRR=	12	0.2	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	31	0.0
9600	UBRR=	6	7.5	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	15	0.0
14400	UBRR=	3	7.8	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	10	3.1
19200	UBRR=	2	7.8	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	0.0
28800	UBRR=	1	7.8	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	6.3
38400	UBRR=	1	22.9	UBRR=	2	0.0	UBRR=	2	7.8	UBRR=	3	0.0
57600	UBRR=	0	7.8	UBRR=	1	0.0	UBRR=	1	7.8	UBRR=	2	12.5
76800	UBRR=	0	22.9	UBRR=	1	33.3	UBRR=	1	22.9	UBRR=	1	0.0
115200	UBRR=	0	84.3	UBRR=	0	0.0	UBRR=	0	7.8	UBRR=	0	25.0

Baud Rate	3.2768 MHz		%Error	3.6864 MHz		%Error	4 MHz		%Error	4.608 MHz		%Error
2400	UBRR=	84	0.4	UBRR=	95	0.0	UBRR=	103	0.2	UBRR=	119	0.0
4800	UBRR=	42	0.8	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	59	0.0
9600	UBRR=	20	1.6	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	29	0.0
14400	UBRR=	13	1.6	UBRR=	15	0.0	UBRR=	16	2.1	UBRR=	19	0.0
19200	UBRR=	10	3.1	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	14	0.0
28800	UBRR=	6	1.6	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	9	0.0
38400	UBRR=	4	6.3	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	6.7
57600	UBRR=	3	12.5	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	0.0
76800	UBRR=	2	12.5	UBRR=	2	0.0	UBRR=	2	7.8	UBRR=	3	6.7
115200	UBRR=	1	12.5	UBRR=	1	0.0	UBRR=	1	7.8	UBRR=	2	20.0

Baud Rate	7.3728 MHz		%Error	8 MHz		%Error	9.216 MHz		%Error	11.059 MHz		%Error
2400	UBRR=	191	0.0	UBRR=	207	0.2	UBRR=	239	0.0	UBRR=	287	-
4800	UBRR=	95	0.0	UBRR=	103	0.2	UBRR=	119	0.0	UBRR=	143	0.0
9600	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	59	0.0	UBRR=	71	0.0
14400	UBRR=	31	0.0	UBRR=	34	0.8	UBRR=	39	0.0	UBRR=	47	0.0
19200	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	29	0.0	UBRR=	35	0.0
28800	UBRR=	15	0.0	UBRR=	16	2.1	UBRR=	19	0.0	UBRR=	23	0.0
38400	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	14	0.0	UBRR=	17	0.0
57600	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	9	0.0	UBRR=	11	0.0
76800	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	6.7	UBRR=	8	0.0
115200	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	0.0	UBRR=	5	0.0

3.3.7 *EEPROM* – энергонезависимая память данных

Разработчики МК часто встраивают в свои изделия энергонезависимую память *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*). Это позволяет сохранить данные на период пропадания питания, что характерно устройствам с батарейным или комбинированным питанием. Также общей чертой обращения к *EEPROM* является то, что процедуры записи и чтения требуют посылки последовательности определенных команд, а сам процесс обращения занимает несколько миллисекунд, требуя полной остановки выполнения программы на этот период. Это обеспечивает сохранность данных от случайной записи, стирания при различных сбоях – отказ генератора, снижение напряжения питания и т.д.

Программно *EEPROM* доступно с помощью трех адресов: регистр адреса (*EEPROM Address RG*), регистр данных (*EEPROM Data RG*) и регистр управления (*EEPROM Control RG*).

	7	6	5	4	3	2	1	0
EEAR $1E ($3E)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
EEDR $1D ($3D)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
EECR $1C ($3C)	-	-	-	-	-	*EEMWE*	*EEWE*	*EERE*
	R	*R*	*R*	*R*	*R*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

EEMWE (*EEPROM Master Write Enable*) – разрешает запись в память, если этот бит равен нулю, то запись не возможна (после программной установки этого бита, МК автоматически его сбросит после четырех тактов МК).
EEWE (*EEPROM Write Enable*) – назначение это бита отождествлено со стробом записи *EEPROM*, который посылается после корректной установки адреса и данных, при этом бит *EEMWE* должен быть установлен в «1».
EERE (*EEPROM Read Enable*) - назначение это бита отождествлено со стробом чтения *EEPROM*, который посылается после корректной установки адреса и данных; при этом процессор останавливается на 4 такта перед выполнением следующей команды; в конце операции бит автоматически сбрасывается; попытка чтения во время записи приведет к прекращению цикла записи и неопределенности данных при чтении.

Таким образом, процедура записи в энергонезависимую память следующая:

- дождаться когда бит *EEMWE* управляющего регистра *EECR* сбросится в ноль;
- записать адрес в регистр *EEAR* (если нужно), записать данные в регистр *EEDR* (если нужно);
- установить в «1» бит *EEMWE* управляющего регистра *EECR*, при этом бит *EEWE* должен быть сброшен;
- в течение 4 тактов после установки *EEMWE* послать команду записи установкой в «1» бита *EEWE*.

Доступ при записи в память занимает примерно 2.5 мс при *Vcc*=5 В и 4 мс при *Vcc*=2.7 В. После успешной записи бит *EEWE* автоматически установится в ноль. После установки бита *EEWE* процессор останавливается на два такта перед выполнением следующей инструкции. Особое внимание при записи в *EEPROM* следует уделять прерываниям, которые могут прервать цикл записи и привести к потере данных.

Сводная таблица портов

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	Reserved								
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	Reserved								
$3B ($5B)	GIMSK	INT1	INT0	–	–	–	–	–	–
$3A ($5A)	GIFR	INTF1	INTF0						
$39 ($59)	TIMSK	TOIE1	OCIE1A	–	–	TICIE1	–	TOIE0	–
$38 ($58)	TIFR	TOV1	OCF1A	–	–	ICF1	–	TOV0	–
$37 ($57)	Reserved								
$36 ($56)	Reserved								
$35 ($55)	MCUCR	–	–	SE	SM	ISC11	ISC10	ISC01	ISC00
$34 ($54)	Reserved								
$33 ($53)	TCCR0	–	–	–	–	–	CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)							
$31 ($51)	Reserved								
$30 ($50)	Reserved								
$2F ($4F)	TCCR1A	COM1A1	COM1A0	–	–	–	–	PWM11	PWM10
$2E ($4E)	TCCR1B	ICNC1	ICES1	.	–	CTC1	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 – Compare Register High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 – Compare Register Low Byte							
$29 ($49)	Reserved								
$28 ($48)	Reserved								
$27 ($47)	Reserved								
$26 ($46)	Reserved								
$25 ($45)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
$24 ($44)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
$23 ($43)	Reserved								
$22 ($42)	Reserved								
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0
$20 ($40)	Reserved								
$1F ($3F)	Reserved								
$1E ($3E)	EEAR	–	EEPROM Address Register						
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	–	–	–	–	–	EEMWE	EEWE	EERE
$1B ($3B)	Reserved								
$1A ($3A)	Reserved								
$19 ($39)	Reserved								
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	Reserved								
$14 ($34)	Reserved								
$13 ($33)	Reserved								
$12 ($32)	PORTD	–	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	–	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	–	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
…	Reserved								
$0C ($2C)	UDR	UART I/O Data Register							
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR	–	–	–
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
$09 ($29)	UBRR	UART Baud Rate Register							
$08 ($28)	ACSR	ACD	–	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
…	Reserved								
$00 ($20)	Reserved								

3.4 ПРОГРАММИРОВАНИЕ МК И ЕГО ИДЕНТИФИКАЦИЯ

В настоящее время практически все МК имеют в своем составе управляющие биты конфигурации, которые называются обычно *Lock*, *Fuse* битами. Назначением этих бит является конфигурирование МК, например, разрешение/запрещение программирования памяти и т.п.

В серии МК *AT*90 *Lock* биты предназначены для управления возможностью записи во *Flash* и *EEPROM*, а также возможностью чтения программы из вне. При этом восстановление значений этих битов в исходное состояние (все разрешено) возможно только операцией *erase chip* – полным удалением программы и данных.

МК *AT*90*S*2313 имеются два *Fuse* бита: *SPIEN*, *FSTRT*. Когда значение первого равно нулю, программирование по последовательному интерфейсу и загрузка данных разрешены (по умолчанию *SPIEN*=0). Когда *FSTRT*=0 включается режим ускоренного старта – за счет сокращения времени ожидания окончания переходных процессов при включении генератора (значение по умолчанию *FSTRT*=1) – см. рис.3.7.

Также *AT*90*S*2313 как и другие содержит нестираемые биты опознавания: *signature bits*, чтение которых позволяет идентифицировать марку МК, его производителя, определить объем *Flash* памяти и получить некоторые другие сведения. Отметим, что с помощью *Lock* битов чтение этих бит может быть запрещено.

Программирование *Flash* возможно двумя путями: по параллельному или последовательному интерфейсу. В исходном состоянии вся память готова к записи (каждая ячейка содержит 0FFh). При этом параллельное программирование требует наличия +12 В, которое используется только для перевода чипа в режим параллельного программирования. В документации на каждый МК имеются подробные алгоритмы программирования и схемы включения.

4. Разработка устройств на основе МК.

Разработка программно-аппаратных устройств это процесс проектирования и создания документации. Проектирование – комплекс мероприятий, обеспечивающих поиск технических решений, удовлетворяющих заданным требованиям, их оптимизацию и реализацию в виде комплекта конструкторских документов и опытного образца (образцов), подвергаемого циклу испытаний на соответствие требованиям технического задания (ТЗ). Здесь следует отметить, что развитость средств математического моделирования иногда позволяет заменить часть экспериментально-физических испытаний на математический эксперимент.

В самом общем случае техническая разработка и внедрение чего-либо (в т.ч. программно-аппаратных систем) подразумевает выполнение следующих стадий (стадии 3, 4, 5 и 6 могут отсутствовать по разным причинам, например: устройство построено на существующей элементной базе с использованием существующих технологий).

1. Научно-исследовательская разработка (НИР). На этой стадии проходят проверку новые идеи и изобретения, в т.ч. оценивается выполнимость поставленных в ТЗ задач (часто на этом этапе ТЗ корректируется или создается заново). Теоретические предпосылки решения научных проблем проверяются в ходе опытно-экспериментальных работ или математическим моделированием (если это не противоречит фундаментальным положениям теорий и ТЗ). Порядок выполнения НИР регламентируется ГОСТ Р 15.101.98 "Порядок выполнения научно-исследовательских работ". Стандарт устанавливает общие требования к организации и выполнению НИР (порядок выполнения и приемки, этапы выполнения, правила их выполнения и приемки, порядок разработки, согласования и утверждения документов в процессе организации и выполнения, порядок реализации результатов).
2. Опытно-конструкторская разработка (ОКР). На этой стадии идеи и решения, возникающие в процессе НИР, реализуются в опытных образцах и технической документации. Таким образом, ОКР является процессом инженерного воплощения теоретических результатов, полученных на этапе НИР, в схему и конструкцию изделия. Основная задача ОКР – создание комплекта конструкторской документации для производства изделия (если имеющиеся в распоряжении разработчика научно-технические материалы не требуют проведения НИР, основанием для выполнения ОКР является ТЗ, утвержденное заказчиком). На этапе ОКР на первый план выступают экономические задачи, так как именно здесь формируются основные параметры изделия, влияющие на его себестоимость и на длительность и стоимость его разработки. ОКР заканчивается выпуском комплекта технической документации, изготовлением и испытанием его опытного образца (или опытной партии).
3. Конструкторская подготовка производства. Осуществляется проектирование нового изделия, разрабатываются рабочие чертежи и техническая документация.
4. Технологическая подготовка производства. Разрабатываются и проверяются новые технологические процессы, проектируется и изготавливается технологическая оснастка для производства изделия.
5. Организационная подготовка производства. На этой стадии выбираются методы перехода на выпуск новой продукции, проводятся расчеты потребности в материалах и комплектующих изделиях, определяются продолжительность производственного цикла изготовления изделия, размеры партий, и пр.
6. Отработка изделия в опытном производстве. Осваивается выпуск опытного образца (опытной партии), проводится отладка новых технологических процессов.

Современная электронная промышленность выпускает огромное количество МК, которые можно применять для решения различных задач управления и обработки информации. Поэтому процесс проектирования устройств с помощью МК сильно упрощается (не надо проводить указанные выше стадии разработки по отношению к самому МК, остается дополнить его в случае необходимости также стандартными изделиями и применить стандартные технологии пайки, монтажа элементов и даже поместить результат в типовой корпус). Поэтому

специфика разработки программно-аппаратных комплексов диктует похожие стадии разработки, но они "ориентированы" на решаемую задачу, а не на средства решения.

Безусловно, решение конкретной задачи из области "измерения, контроля, управления" начинается с анализа способов решения на фундаментальном уровне (это своеобразная стадия НИР, требующая описания постановки задачи и ее решения в терминах математики, физики, информатики и т.п.). Оптимальный выбор решения сложной задачи часто можно сделать благодаря личному опыту разработчика и здесь большую роль играет математическое моделирование, которое также берет на себя часть задач стадии ОКР и позволяет снизить затраты на последнюю. В любом случае итогом стадии НИР является конкретный способ решения, который представляется в виде структурной и функциональной схем и алгоритма программной части будущего устройства с оглядкой на возможности (состав и характеристики) современных МК.

Структурная схема – это схема, определяющая основные функциональные части устройства, их назначение и основные взаимосвязи. В более общем смысле "структура" – это совокупность частей системы, на которые она может быть разделена по определенному признаку, а также основные пути передачи воздействия между ними.

В качестве примера перехода от математической модели к структурной схеме рассмотрим задачу сложения числа и единицы: $x=x+1$. Назовем устройство "Инкрементор". С точки зрения структуры нам потребуется: сумматор (СУМ) и, очевидно, устройства для хранения данных (УХД): операнда (УХО), константы (К1) и результата (УХР), а также схема управления (СУ). Тогда структурная схема такого устройства может иметь вид – рис. 4.1 (пунктиром выделены два варианта, чем лаконичней схема – тем лучше, без потери ясности конечно):

- блоки обозначаются прямоугольниками с названиями внутри (допускается обозначения буквами/надписями на русском и английском языках);
- основные связи обозначаются линиями (допускается указание направления передачи сигналов/данных по смыслу);
- блоки могут не иметь связей (как СУ), но это не означает, что он "бесполезный" – просто он связан со всеми блоками и, если нарисовать эти связи, то схема будет "захламлена" – снижается ее читаемость, информативность (чем больше блоков – тем будет хуже, именно поэтому говорят, что на структурной схеме показывают только основные связи).

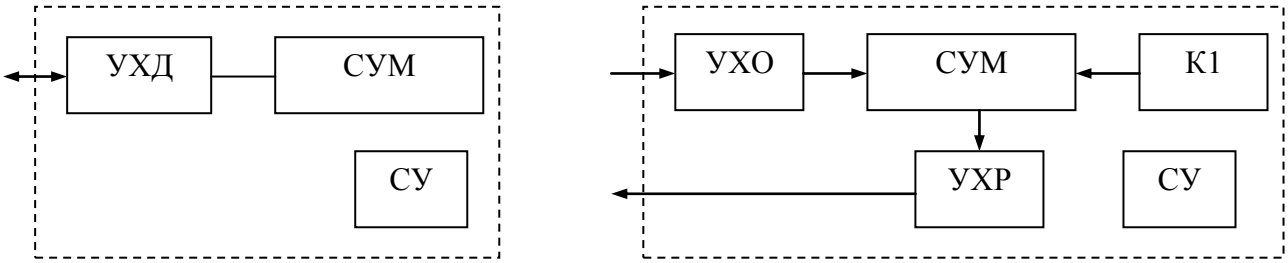


Рис. 4.1 – Варианты структурной схемы инкрементора

Функциональная схема – документ, отражающий процессы, протекающие в устройстве или в его отдельных функциональных узлах. Функциональная схема поясняет процессы (как правило, основные), протекающие в функциональных блоках и цепях устройства. Применительно к микропроцессорным устройствам, по сути, это развернутая структурная схема с пояснениями, учитывающими их реализацию (в частности, блоки группируются по месту их реализации, привязываются к смысловому назначению: индикация, панель ввода данных, интерфейсный блок и т.п.).

На примере "Инкрементора" функциональная схема обязана указать способ реализации устройств хранения данных и сумматора, а вот схема управления в данном случае – не основное. УХД можно реализовать регистрами, а СУМ в виде сумматора или на основе АЛУ (выпускается готовый функциональный блок). Тогда функциональная схема устройства может

иметь вид – рис. 4.2 (обозначения УХД и СУМ – наследуются из структурной схемы). Обратите внимание, что параметры (скорость работы, питание) регистров и сумматора на этой схеме не имеют значения, а вот разрядность может указываться (т.к. разрядность входных данных может быть разной). Логика работы устройства состоит в записи входного операнда в регистр для "*X*", сложения его значения с единицей, хранимой в регистре для "1", и выдаче результата в регистр "результата" (т.е. по сути это "указания" для схемы управления).

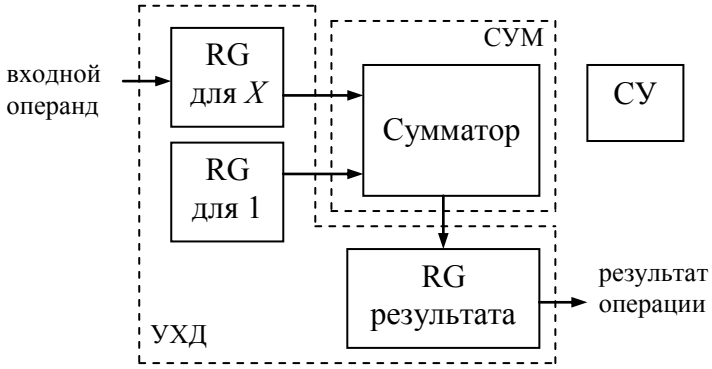


Рис. 4.2 – Функциональная схема "Инкрементора"

После разработки функциональной схемы устройства (временно отложив вопрос создания алгоритма устройства) приступают к выбору элементной базы, которая будет учитывать не только функциональный смысл, но и требования по быстродействию, уровню питания и т.п. После выбора элементной базы рисуют принципиальную электрическую схему, которая содержит промышленно выпускаемые элементы и учитывает законы электротехники (в части нагрузочной способности элементов, соответствия уровней питания и его качества, а также временных характеристик элементов и блоков). Если необходимо схему разбивают на фрагменты – в соответствии с их реализацией на печатных платах. На рис. 4.3 представлена схема принципиальная электрическая "Инкрементора" без указания цепей питания (для наглядности примера), а в таблице 4.1 приводятся перечень элементов и их назначение. Отметим, что на реализации УХР в виде отдельного регистра можно сэкономить, т.к. сумматор сохраняет результат при сохранении исходных данных.

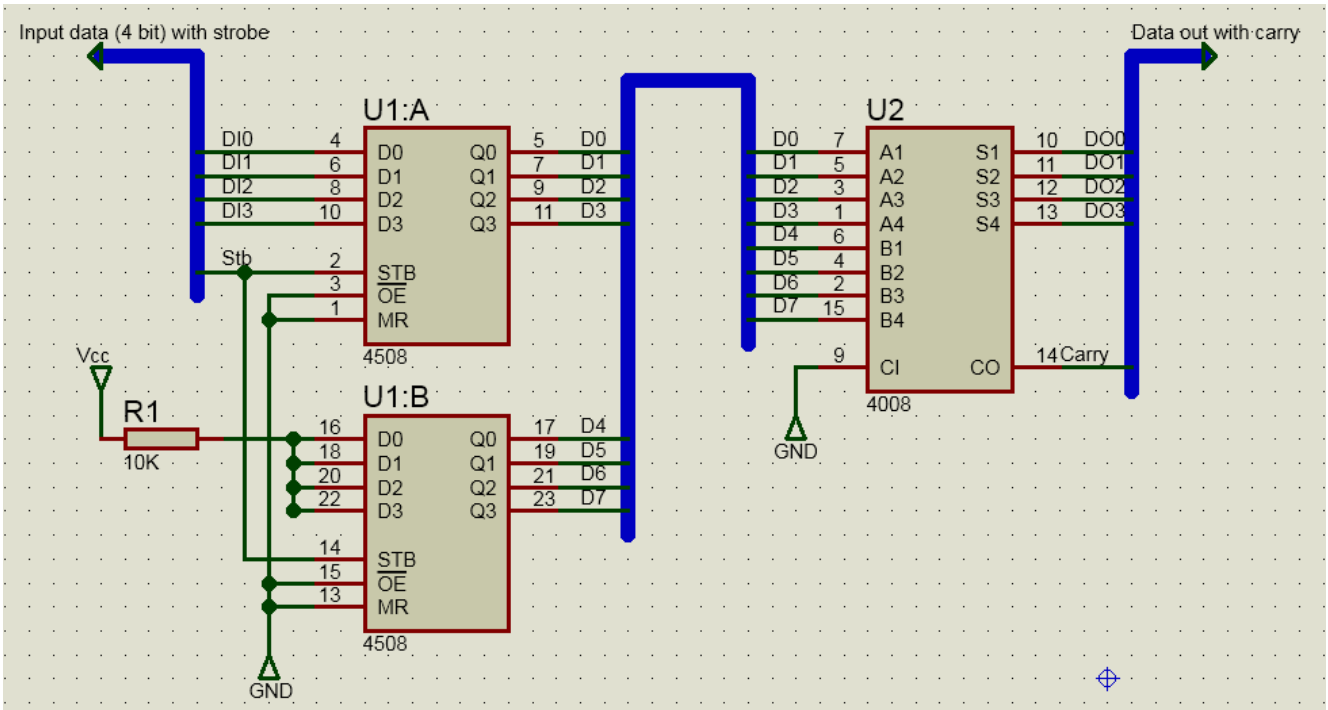


Рис. 4.3 – Принципиальная электрическая схема "Инкрементора" (цепи питания не показаны)

Приведенная схема неоптимальна, например можно было бы вместо *U*1:*B* подтягивающий резистор включить ко входам сумматора *B*1-*B*4, а освободившийся элемент использовать в других целях проекта или найти не сдвоенный регистр, чтобы уменьшить стоимость к примеру.

Таблица 4.1. Перечень элементной базы

Обозн.	Маркировка	Назначение
*U*1	*CD*4508 Dual 4 bit latch	Сдвоенный 4-разрядный регистр (УХД) - хранение операндов
*U*2	*CD*4008 4 bit binary full adder with fast carry	Сумматор 4-разрядный с переносом (СУМ) - вычисление сумму и хранение результата

Далее прорабатываются временные диаграммы работы блоков (элементов), позволяющие распределить общие ресурсы во времени и, например определить время работы устройства до появления результата. На рис. 4.4 представлены временные диаграммы работы "Инкрементора". Видно, что результат готов через максимум 3200 нс после записи входных данных (нарастающего фронта строба *Stb*), данные по задержкам работы самих элементов схемы берутся из документации производителя.

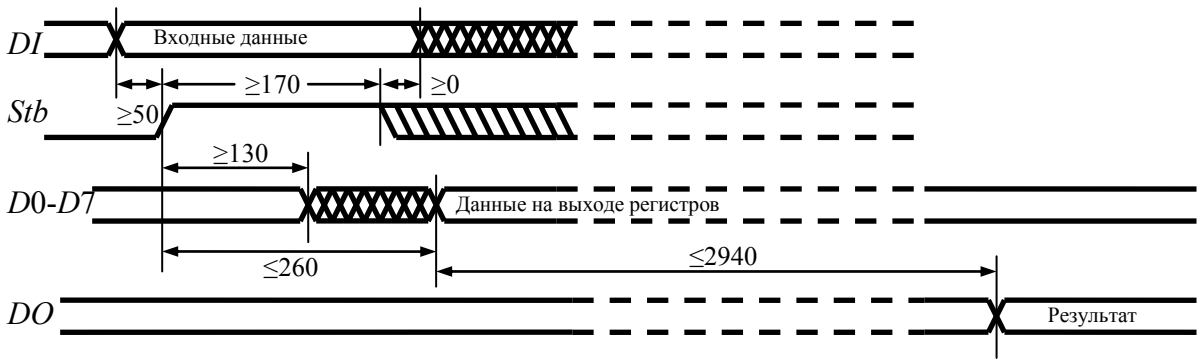


Рис. 4.4 – Временные диаграммы работы "Инкрементора"

На этом разработка аппаратной части заканчивается. А после выполнения математического моделирования, подтверждающего заявленные характеристики, оформления пакета документации и производства опытного образца стадия ОКР аппаратной части считается завершенной.

Следующим этапом является разработка программной части, которая начинается с разработки алгоритма программы, учитывающего специфику выбранной элементной базы (исходными данными является функциональная и принципиальная схемы, а также ТЗ). В примере с "Инкрементором" программная составляющая отсутствует (не предполагается, т.к. нет программируемых элементов). В общем случае, алгоритмы должны выделять основные процедуры (функции) и формировать исходные и выходные структуры данных в соответствии с аппаратной частью и ТЗ. По разработанному алгоритму пишется программный код на каком-либо подходящем языке программирования (в случае с МК в идеале – ассемблере).

Наконец, если на какой-то стадии или каком-то этапе становится очевидным невозможность достижения параметров/показателей ТЗ, то процесс проектирования повторяют с того этапа, который может обеспечить исправление ситуации (или обосновывают невозможность решения такой задачи и согласуют изменение требований в ТЗ).

Разрабатываемое устройство обязано сопровождаться документацией (чертежи, схемы, алгоритмы, программы и т.п.), оформление которой может происходить параллельно с разработкой или по ее окончанию (в этом случае говорят: "рабочие материалы" превращаются в "документацию"). Для их единообразного оформления приняты стандарты, описывающие состав документации, порядок ее оформления и правила поддержания, внесения изменений и т.п.: ЕСКД – единая система конструкторской документации, ЕСТД – единая система технологической документации (это касается производственных процессов и жизненных циклов изделий – выходит за рамки курса) и ЕСПД – единая система программной

документации. Со стандартами можно ознакомиться в библиотеке и в электронных базах, например: https://docs.cntd.ru

4.1. Стандарты аппаратной части (ЕСКД)

В настоящее время разработка устройств выполняется в рамках специализированных *CAD* систем, что обеспечивает выполнения многочисленных стандартов ЕСКД. С другой стороны, ограничиваясь необходимостью выполнения курсового проекта, обозначим основные из них.

- ГОСТ 2.102-2013 "Виды и комплектность конструкторских документов"
- ГОСТ 2.701-2008 "Схемы. Виды и типы. Общие требования к выполнению"
- ГОСТ 2.702-2011 "Правила выполнения электрических схем"
- ГОСТ 2.417-91 "Платы печатные. Правила выполнения чертежей"
- ГОСТ Р 53736-2009 "Изделия электронной техники. Порядок создания и постановки на производство. Основные положения"

4.2. Стандарты программной части (ЕСПД)

- ГОСТ 19.001-77 Общие положения.
- ГОСТ 19.101-77 Виды программ и программных документов (переиздан в ноябре 1987 г. с изменениями).
- ГОСТ 19.102-77 Стадии разработки.
- ГОСТ 19.103-77 Обозначение программ и программных документов.
- ГОСТ 19.104-78 Основные надписи.
- ГОСТ 19.105-78 Общие требования к программным документам.
- ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом.
- ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению.
- ГОСТ 19.202-78 Спецификация. Требования к содержанию и оформлению.
- ГОСТ 19.301-79 Программа и методика испытаний.
- ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению.
- ГОСТ 19.402-78 Описание программы.
- ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению.
- ГОСТ 19.501-78 Формуляр. Требования к содержанию и оформлению.
- ГОСТ 19.502-78 Описание применения. Требования к содержанию и оформлению.
- ГОСТ 19.503-79 Руководство системного программиста. Требования к содержанию и оформлению.
- ГОСТ 19.504-79 Руководство программиста.
- ГОСТ 19.505-79 Руководство оператора.
- ГОСТ 19.506-79 Описание языка.
- ГОСТ 19.508-79 Руководство по техническому обслуживанию. Требования к содержанию и оформлению.
- ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполняемые печатным способом.
- ГОСТ 19.701-90 (ИСО 5807-85) Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
- ГОСТ 19.781-90 Обеспечение систем обработки информации программное.

Стоит упомянуть стандарт ГОСТ Р 51189-98 "Средства программные систем вооружения". Некоторые заказчики не военного профиля требуют выполнения этого стандарта, т.к. он учитывает вопросы несанкционированного доступа, а также затрагивает проблематику качества программных средств.

Также существует ГОСТ Р 51904-2002 "Программное обеспечение встроенных систем", стандарт подготовлен в развитие ГОСТ Р ИСО/МЭК 12207-99 "Информационная технология.

Процессы жизненного цикла программных средств" с целью учета специфики разработки и документирования программного обеспечения встроенных систем реального времени.

Помимо отечественных стандартов (которые надо признать устарели) существуют международные стандарты для написания документации, ниже приводится список основных:

- *IEEE Std* 1063-2001 «*IEEE Standard for Software User Documentation*» – стандарт для написания руководства пользователя;
- *IEEE Std* 1016-1998 «*IEEE Recommended Practice for Software Design Descriptions*» – стандарт для написания технического описания программы;
- *ISO/IEC FDIS* 18019:2004 «*Guidelines for the design and preparation of user documentation for application software*» – ещё один стандарт для написания руководства пользователя, который включает большое количество примеров;
- *ISO/IEC* 26514:2008 «*Requirements for designers and developers of user documentation*» – стандарт для дизайнеров и разработчиков документации пользователя.

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Национальный исследовательский университет «МЭИ»

КОНСПЕКТ ЛЕКЦИЙ

дисциплины базовой части профессионального цикла Б3.14

«Микропроцессорные системы»

Направление подготовки 230100 Информатика и вычислительная техника

Профили Вычислительные машины, комплексы, системы и сети,
Семестр – 7

Авторский коллектив:

Доцент кафедры ВМСиС А.В. Иванов

Москва 2012 НИУ «МЭИ»

Содержание

Лекция 1....4
1. Однокристальные микро-ЭВМ (микроконтроллеры). Основные характеристики микроконтроллеров....4
Лекция 2....6
2. Микроконтроллеры (МК) семейства MCS – 51....6
2.1. Структурная схема микроконтроллера семейства MCS51....6
2.2. Организация памяти программ....8
2.3.Организация памяти данных....9
2.4.Регистры специальных функций (SFR)....11
Лекция 3....13
3. Программирование на языке Ассемблера для микроконтроллеров семейства MCS – 51....13
3.1.Способы адресации....13
3.2. Система команд....13
3.2.1. Команды передачи данных....13
Лекция 4....15
3.2.2 Команды обработки данных....15
3.2.3. Команды управления....17
3.2.4 Команды для выполнения операций с отдельными битами....19
3.3. Средства отладки программ....20
Лекция 5....23
4. Построение микропроцессорной системы на базе микроконтроллера семейства MCS51....23
4.1. Порты ввода/вывода....23
4.1.1. Особенности выполнения команд при обращении к портам ввода/вывода....24
4.2. Подключение внешних БИС памяти программ и данных....26
4.2.1. Временные диаграммы выполнения машинного цикла в микроконтроллере....26
4.2.2. Схема подключение внешних БИС памяти программ и данных....29
4.2.3. Стирание и программирование РПЗУ....31
Лекция 6....32
4.3. Таймер/счетчик....32
Лекция 7....38
4.4. Организация прерываний....38
Лекция 8....42
5.1.2. Последовательный интерфейс RS232....47
Лекция 9....51
5.2.1. Организация последовательного интерфейса I^2C....51
5.2.2. Последовательный периферийный интерфейс SPI....53
Лекция 10....55
5.3. Организация последовательного интерфейса CAN....55
Лекция 11....57
5.4. Последовательный однопроводный интерфейс One Wire....57
5.5. Универсальная последовательная шина....60
Лекция 12....67
6. Массив программируемых счетчиков....67
6..1. Назначение и состав массива программируемых счетчиков....67
6.2. Режимы работы общего таймера/счетчика....69
6.3. Режим захвата события....71
6.4. Режим 16-разрядного программируемого таймера, высокоскоростного вывода и сторожевого таймера....73

6.5. Режим генератора импульсов с заданной скважностью 75
6.6. Обработка прерываний от источников PCA 77
Лекция 13 79
7. Микроконтроллеры с RISC архитектурой 79
7.1. Микроконтроллеры AVR 79
7.1.1. Структурная схема микроконтроллера AVR 79
7.1.2. Организация памяти 80
7.1.3. Система команд 82
7.1.4. Порты ввода/вывода 87
7.1.5. Таймеры счетчики 90
7.1.6. Сторожевой таймер 90
7.1.8. Аналого-цифровой преобразователь и компаратор 92
7.1.9. Организация прерываний 93
7.1.10. Сброс микроконтроллера 95
Лекция 14 95
7.2.Микроконтроллеры PIC 95
7.2.1. Семейства микроконтроллеров PIC 95
7.2.2. Особенности архитектуры PIC16F74 97
7.2.3. Принцип "чтение-модификация-запись" 101
7.2.4. Особенности языка ассемблера и системы команд 103
7.2.5. Особенности ассемблера MPASM 107
Лекция 15 110
8. Цифровая обработка сигналов 110
8.1. Общие сведения о цифровой обработке сигналов 110
Лекция 16 114
8.2. Сигнальные микропроцессоры 114
8.3. Сигнальные процессоры фирмы Texas Instruments 116
Лекция 17 119
9. Комплексная отладка МПС 119

Лекция 1

1. Однокристальные микро-ЭВМ (микроконтроллеры). Основные характеристики микроконтроллеров

В настоящее время разработаны и выпускаются микропроцессорные БИС, содержащие на одном кристалле все необходимые узлы микро-ЭВМ (ОЭВМ – однокристальные микро-ЭВМ). Поскольку ОЭВМ получили широкое применение в управляющих устройствах, системах передачи данных и системах управления технологическими процессами, то их также называют микроконтроллерами. Из них фирма Intel разработала микроконтроллер 8051, который лёг в основу семейства микроконтроллеров MSC51.

Разработка систем управления и контроля с использованием однокристальных микроконтроллеров в настоящее время переживает настоящий бум. Системы на базе микроконтроллеров используются практически во всех сферах жизнедеятельности человека, и каждый день появляются все новые и новые области применения этих устройств. В последнее время в связи с бурным развитием электроники и схемотехники расширились возможности и самих микроконтроллеров, позволяющие выполнять многие задачи, ранее недоступные для реализации, такие, например, как обработка аналоговых сигналов. Одним из наиболее ранних микроконтроллеров, появившихся на рынке, является микроконтроллер 8051, разработанный фирмой Intel более двадцати лет назад. Несмотря на столь приличный возраст, классический 8051 и его клоны в настоящее время остаются одними из наиболее популярных при разработке систем управления и контроля. Хорошо продуманная архитектура и интуитивно понятная система команд оказывают решающее влияние на выбор многих разработчиков аппаратно-программных систем.

Да и сами микроконтроллеры линейки 8051 постоянно развиваются, предлагая разработчику все новые и новые возможности. На основе базового кристалла 8051 созданы и успешно применяются устройства с развитой периферией и большими объемами памяти.

Программирование микроконтроллеров в настоящее время значительно упростилось благодаря инструментальным средствам высокого уровня, разработанным ведущими фирмами. Сегодня микроконтроллеры кроме использования языка Ассемблера можно программировать на языках C, Pascal и др., что во многом облегчает жизнь программистам, не знакомым с аппаратной частью этих устройств.

Основными производителями клонов 51-го семейства в мире являются фирмы Philips, Siemens, Intel, Atmel, Dallas, Temic, Oki, AMD, MHS, Gold Star, Winbond, Silicon Systems и ряд других.

Все микроконтроллеры из семейства MCS-51 имеют общую систему команд. Наличие дополнительного оборудования влияет только на количество регистров специального назначения.

В СССР производство микроконтроллера 8051 осуществлялось в Киеве, Воронеже (1816ВЕ31/51, 1830ВЕ31/51), Минске (1834ВЕ31) и Новосибирске (1850ВЕ31). Микроконтроллеры данного семейства выпускаются в PLCC, DIP и QFP корпусах и могут работать в следующих температурных диапазонах:
• коммерческий (0°С — +70°С);
• расширенный (-40°С — +85°С):
• для военного использования (-55°С — +125°С).

Базовой моделью семейства микроконтроллеров MCS-51 и основой для всех последующих модификаций является восьмиразрядный микроконтроллер 8051. Он построен с использованием Гарвардской архитектуры. Обобщённая структурная схема микроконтроллера показана на рис.1. В состав микроконтроллера входят:

– внутренний генератор тактовой частоты *G*;
– арифметико-логическое устройство *ALU*;
– внутренняя память программ *ROM*;
– внутренняя память данных *RAM*;
– четыре восьмиразрядных порта ввода/вывода *PORT0 – PORT3*;
– интерфейсные устройства *IU*.

Его основные характеристики следующие:
• восьмиразрядный ЦП, оптимизированный для реализации функций управления;
• встроенный тактовый генератор;
• адресное пространство памяти программ - 64 К;
• адресное пространство памяти данных - 64 К;
• внутренняя память программ - 4 К;
• внутренняя память данных - 128 байт;
• дополнительные возможности по выполнению операций булевой алгебры (побитовые
операции);
• 32 двунаправленные и индивидуально адресуемые линии ввода/вывода;
• 2 шестнадцатиразрядных многофункциональных таймера/счетчика;
• полнодуплексный асинхронный приемопередатчик;
• векторная система прерываний с двумя уровнями приоритета и шестью источниками
событий [2-4,7,8].

Лекция 2

2. Микроконтроллеры (МК) семейства MCS – 51

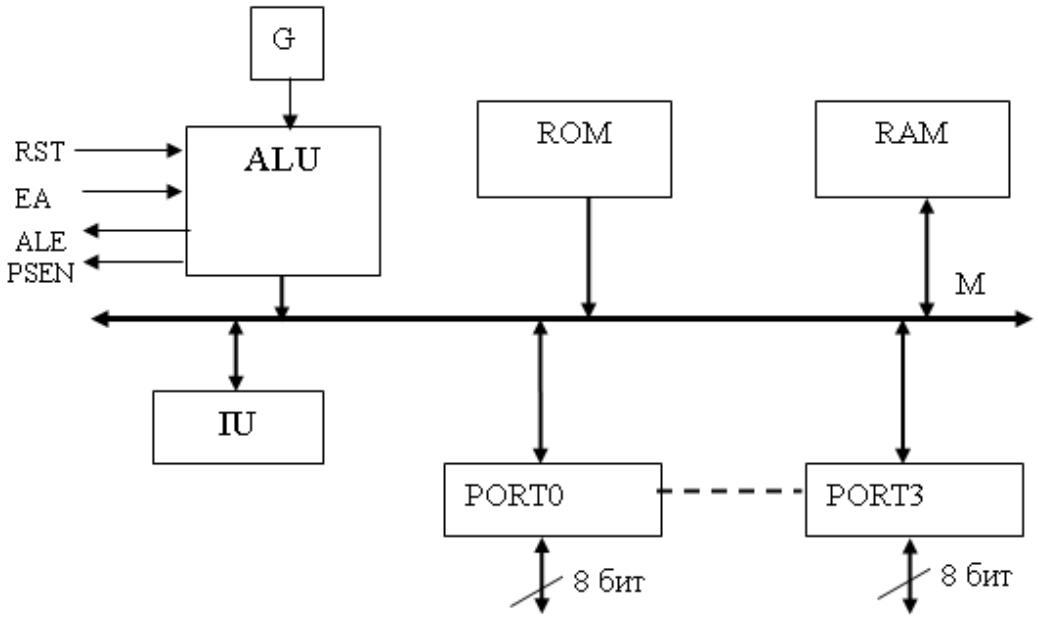


Рис.1. Обобщённая структурная схема микроконтроллера

2.1. Структурная схема микроконтроллера семейства MCS51

Структурная схема микроконтроллера (рис.2.) условно может быть разделена на четыре узла:

- операционный узел ;
- узел внутренней памяти;
- узел управления и синхронизации;
- узел сопряжения с внешними устройствами, к которому относятся порты ввода/вывода.

В состав операционного узла входят арифметико-логическое устройство *ALU*, регистр аккумулятор *ACC*, буферные регистры временного хранения операндов *TMP1* и *TMP2*, регистр слова состояния программы (флагов) *PSW*, регистр специального назначения *B*, участвующий в операциях умножения и деления.

8-разрядное *ALU* выполняет арифметические операции сложения, вычитания, умножения и деления; логические операции И, ИЛИ, инвертирования, операции сдвига влево и вправо.

Важной особенностью *ALU* является его способность оперировать не только байтами, но и битами. Возможность оперировать битами определяется тем,

что в микроконтроллере имеется специальный механизм, который в некоторых описаниях называется логическим процессором.

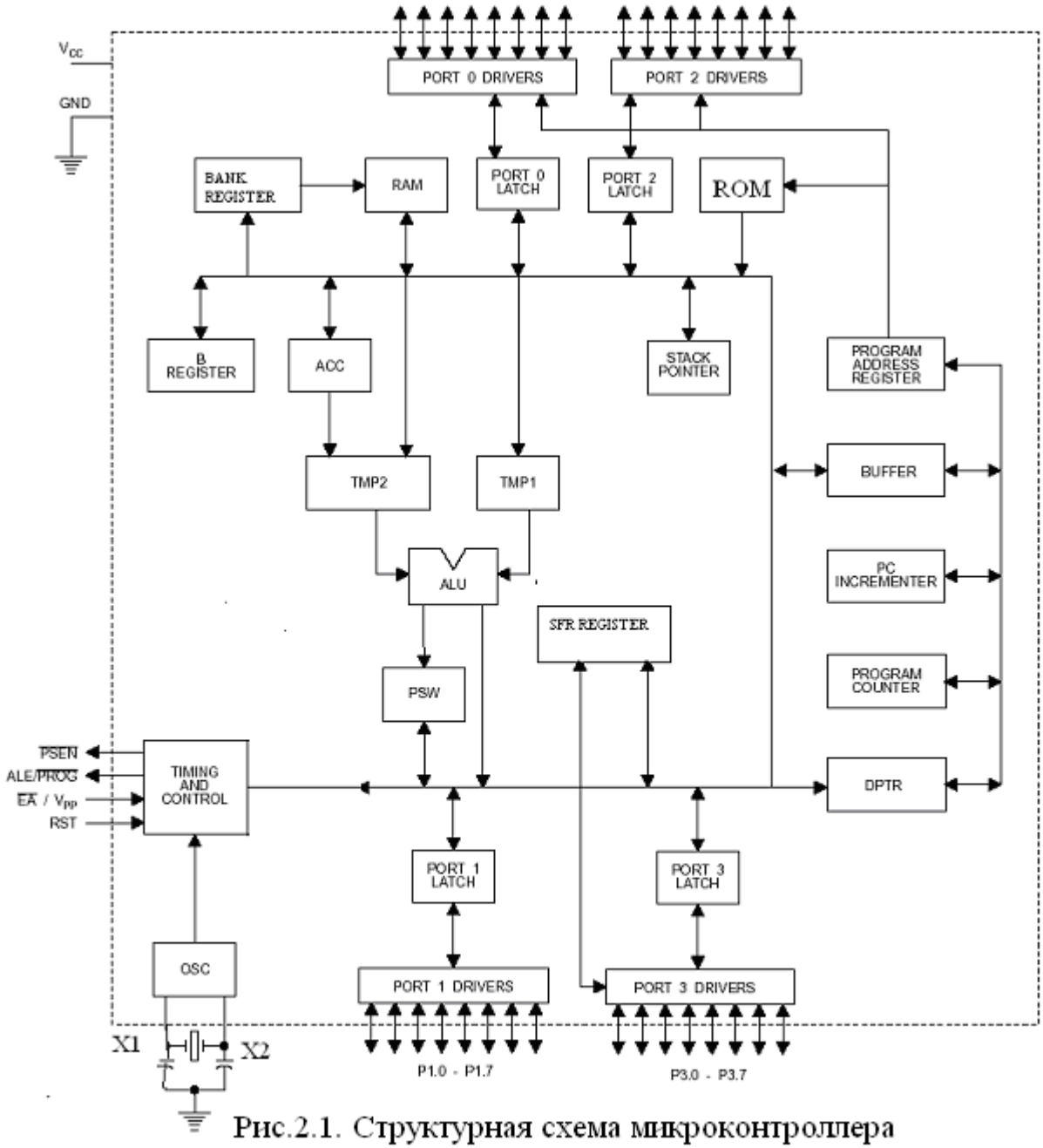


Рис.2.1. Структурная схема микроконтроллера

Формирование признаков (флагов) производится при выполнении команд, в которых участвует аккумулятор, а также при выполнении некоторых команд побитной обработки. В табл.2.1 приведён формат регистра *PSW* .

Таблица 2.1

Формат регистра *PSW*

7	6	5	4	3	2	1	0
C	AC	F0	RS1	RS0	OV	1	P

Старший седьмой разряд предназначен для хранения флага *C* переноса. В шестом разряде записывается флаг *AC* промежуточного переноса (из младшей тэтрады в старшую). Пятый разряд резервируется для записи бита по усмотрению пользователя. Разряды 4 и 3 служат для выбора банков регистров общего назначения. В микроконтроллере, начиная с нулевого адреса внутренней памяти данных, 32 ячейки памяти функционально могут быть использованы как регистры общего назначения РОН. Эти ячейки разбиваются на четыре банка по восемь регистров в каждом банке. Внутри каждого банка регистрам РОН присвоены номера от R0 до R7. При включении питания или сбросе по умолчанию устанавливается нулевой банк. Установка номера банка может быть изменено программно в соответствии с табл.2.2.

Таблица 2.2

Выбор номера банка регистров РОН

RS1	RS0	Номер банка	Имя регистра	Адрес регистров
0	0	0	R0 – R7	00H – 07H
0	1	1	R0 – R7	08H – 0Fh
1	0	2	R0 – R7	10H – 17H
1	1	3	R0 – R7	18H – 1FH

Во втором разряде записывается флаг арифметического переполнения *OV* .Первый разряд не используется и в нём содержится единица. Нулевой разряд используется для записи флага *P* чётности единиц в байте результата. Нулевой результат относится к чётному.

В узел внутренней памяти входят: встроенная память программ *ROM*, встроенная память данных *RAM*: в память данных входят регистры РОН, адреса которых соответствуют адресам, указанных в табл.1.2 (*BANK RAM*) и регистры специальных функций *SFR*.

2.2. Организация памяти программ

На рис. 2.3 показана организация памяти программ (для случая внутренней программной памяти ёмкостью 4Кх8). Число разрядов адреса равно 16. Для формирования адреса служит 16-разрядный счётчик команд *PC (PROGRAM COUNTER***).** Программы пользователя записываются в память, обычно начиная с последнего адреса области векторов прерывания. При включении питания и формирования сигнала *RST* (сброс) устанавливается адрес *0000H* По этому адресу должна быть записана команда безусловного перехода (вектор сброса) для обхода зарезервированной области векторов прерываний.

Если в системе предусмотрено расширение памяти программ с помощью подключения внешних БИС памяти, то переход к выборке команд из внешней памяти программ осуществляется автоматически при значении

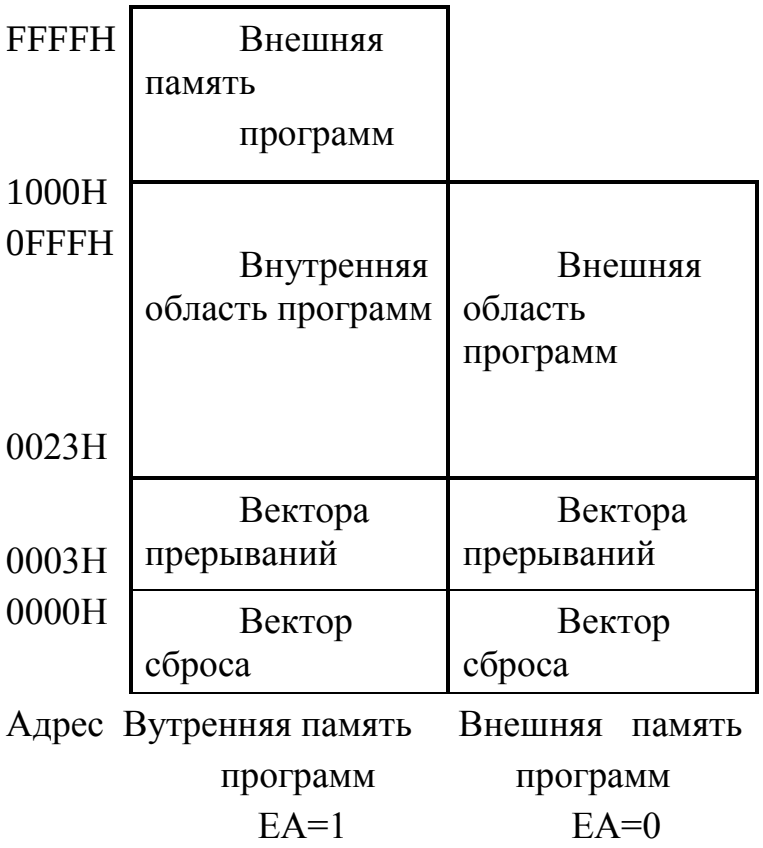


Рис.2.3. Организация памяти программ

адресов больше величины *0FFFH*. В том случае, когда вся программа размещается в пределах внутреннего ПЗУ и внешняя память отсутствует, по адресам из диапазона *1000H – FFFFH* будет считываться неопределённая информация. Режим работы с внутренней памятью обеспечивается значением сигнала *EA=1* на соответствующем вывода микроконтроллера.

Подача на вывод *EA* сигнала низкого уровня переводит к отключению внутренней памяти программ выборке кодов команд только из внешней памяти программ по всем адресам из диапазона 0*000H – FFFFH*. Время выборки команд из внешней и внутренней памяти одинаково.

2.3. Организация памяти данных

На рис.2.4 показана структура внутренней памяти данных. Пространство адресов внутренней памяти данных ёмкостью 128 байтов складывается из адресов внутреннего ОЗУ *00H – 7FH* и адресов регистров специального назначения *SFR*, размещённых в диапазоне *80H – FFH*. В последних модификациях микроконтроллеров семейства MCS51 внутренняя

память данных может иметь ёмкость 256 байтов и тогда её диапазон адресов возрастает до значения *FFH*.

Ячейки памяти ОЗУ из диапазона *20H – 2FH* (16 байтов) допускают адресацию к каждому биту. В этом диапазоне 128 битов имеют свой уникальный адрес от 0 до 127 (*00H – 7FH*). Например, в ячейке памяти *2FH* по адресу 120 (*78H*) в нулевой бит может быть записана/прочитана единица или ноль.

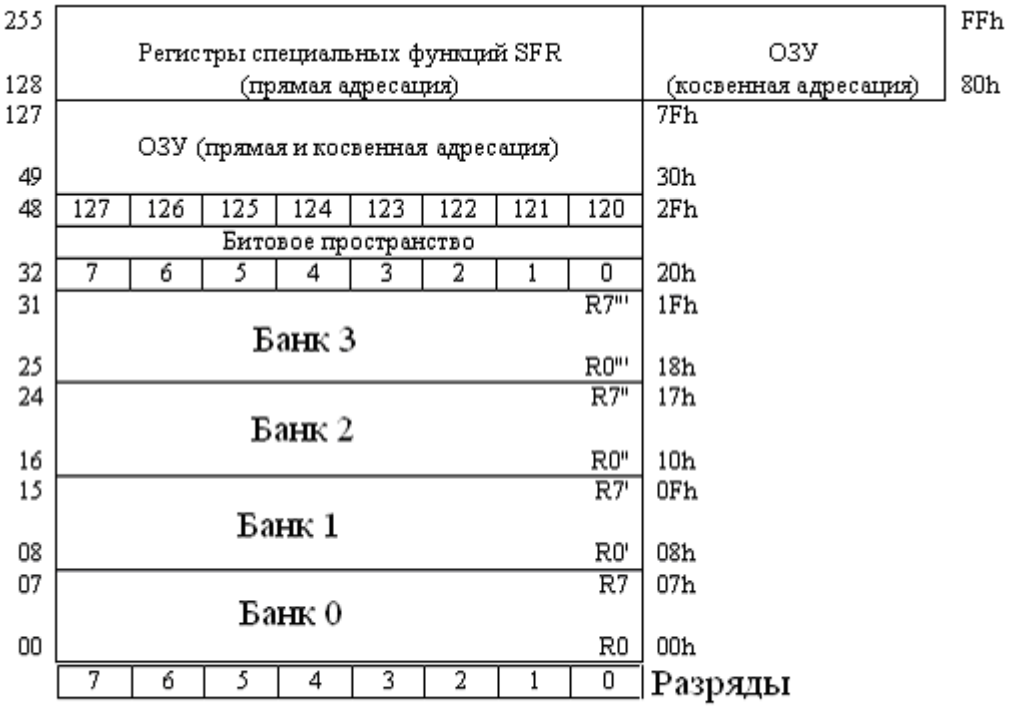


Рис.2.4. Внутренняя память данных

Внутренняя память данных используется также для организации стека. 8-разрядный указатель стека *SP (STACK POINTER)* служит для указания адреса последнего байта, записанного в стек. При заполнении стека адрес увеличивается, а при считывании адрес уменьшается. Начальное значение указателя стека *SP* после сброса соответствует величине 07H и может достигать максимального значения ёмкости внутренней памяти данных. Программное изменение содержимого указателя стека *SP* даёт возможность перемещения стека в любую область адресного пространства внутренней памяти данных.

Если в системе предусмотрено расширение памяти данных с помощью подключения внешних БИС, то выбор данных производится из диапазона адресов 0*000H – FFFFH*. Но обращение к внешней памяти данных производится только командами типа *MOVX*, при выполнении которых формируются сигналы чтения *RD* в цикле приёма и *WR* в цикле записи данных во внешнюю память данных. Указанные сигналы формируются на выходах порта *P3* и не вырабатываются при обращении к памяти программ.

2.4.Регистры специальных функций (SFR)

Регистры специальных функций *SFR* имеют фиксированные адреса и обращение к ним обеспечивается как к ячейкам внутренней памяти данных, но с использованием только прямой адресации. Функционально регистры *SFR* могут быть разделены на:

– арифметические регистры;
– регистры указатели;
– регистры управления для организации прерывания;
– таймеры/счётчики и регистры управления ими;
– порты ввода/вывода;
– регистры управления последовательным вводом/выводjм (UART).

В табл.2.3 перечислены регистры специальных функций.

В узел управления и синхронизации входит внутренний генератор *OSC*, К внешним выводам *X1* и *X2* микроконтроллера подключается кварцевый резонатор для стабилизации тактовой частоты сигналов синхронизации. Устройство управления *TIMUNG AND CONTROL* на основе сигналов синхронизации формирует машинный цикл фиксированной длительности, равный 12 периодам тактовой частоты внутреннего генератора. В зависимости от кода операции на выходе устройства управления формируется сигнал выборки внешней памяти программ *PSEN* и сигнал стробирования адреса *ALE,* фиксирующий момент выдачи адреса в порт *P0*. На устройство управления подаются также сигнал *EA*, назначение которого описано выше и сигнал сброса *RST*. Сигнал *RST* служит для установки микроконтроллера в исходное состояние. На вход *RST* необходимо подавать высокий уровень в течение двух машинных циклов (не менее 24 тактов синхронизации). Сигнал сброса настраивает порты *P0 – P3* на ввод, регистр указателя стека в состояние *07H*, а остальные регистры специальных функций в – нуль.

Таблица 2.3

Регистры специальных функций

Функциональ-ное назначение	Обозначение	Назначение регистра	Адрес
Арифметические регистры	ACC	Аккумулятор	E0H
	B	Регистр B участвует в операциях умножения и деления	F0H
	PSW		D0H

Регистры указатели	SP DPTR DPH DPL	Указатель стека 16-разрядный указатель памяти Старший байт DPTR Младший байт DPTR	81H 83H 82H
Регистры управления прерываниями	IP IE	Регистр приоритетов Регистр разрешения прерываний	B8H A8P
Таймеры/счётчики и регистры управления ими	TH0 TL0 TH1 TL1 TCON TMOD	Старший байт таймера/счётчика 0 Младший байт таймера/счётчика 0 Старший байт таймера/счётчика 1 Младший байт таймера/счётчика 1 Регистр управления таймерами Регистр режимов таймеров	8CH 8Ah 8Dh 8BH 88H 89H
Порты ввода/вывода	P0 P1 P2 P3	Порт 0 Порт 1 Порт 2 Порт 3	80H 90H A0H B0H
Регистры управления UART	SBUF SCON	Буферный регистр приемопередатчика Регистр управления приемопередатчика	99H 98H

Примечание к табл. 2.3. Для регистров, адреса которых кратны восьми, разрешён побитный доступ.

Узел сопряжения с внешними устройствами содержит четыре восьмиразрядных порта ввода/вывода или 32 линии ввода/вывода. Каждую линию можно рассматривать как независимую друг от друга. Поэтому в каждом порте часть линий может быть использована для ввода, а другая часть – для вывода. Все порты кроме их прямого назначения для обмена данными с внешними устройствами могут иметь и другое функциональное назначение.

Двунаправленный порт *P0* может служить для перелачи информации по шине данных, связывающей микроконтроллер с внешней памятью или другими устройствами микропроцессорной системы (МПС). Через этот же порт передаётся младший байт 16-разрядного адреса. Разделение данных от адреса производится с помощью сигнала *ALE*. Старший байт адреса выводтся через квазидвунаправленный порт *P2*.

Квазидвунаправленный порт *P3* многофункциональный. Его линии используются для ввода/вывода следующих управляющих сигналов:

– P3.0 последовательный ввод (приёмник *RXD*);
– P3.1 последовательный вывод (передатчик *TXD*);
– P3.2 вход внешнего прерывания от источника 0 (*INT0*);
– P3.3 вход внешнего прерывания от источника 1 (*INT1*);
– P3.4 вход таймера/счётчика *T0*;
– P3.5 вход таймера/счётчика *T1*;
– P3.6 сигнал записи данных WR во внешнюю память данных;
– P3.7 сигнал чтения данных RD из внешней памяти данных.

Лекция 3

3. Программирование на языке Ассемблера для микроконтроллеров семейства MCS – 51

3.1.Способы адресации

Система команд состоит из 111 базовых команд. Их машинные коды могут быть записаны с помощью одного, двух или трех байтов. При описании команд использованы следующие обозначения: Rr – один из восьми регистров общего назначения текущего банка, номер которого задан в регистре PSW; ad – адрес ячейки памяти внутреннего ОЗУ или регистра специального назначения; bit – адрес бита, rel – смещение, которое указывается в командах управления для вычисления адреса перехода.

Для обозначения косвенной адресации используется знак @ коммерческое «эт» (далее этот знак в тексте имеет вид - *a*), а для непосредственной – знак #.

Все команды могут быть разделены на четыре группы: команды передачи данных, команды обработки данных, команды управления, команды выполнения операций с отдельными битами.

3.2. Система команд

3.2.1. Команды передачи данных

В группу команд передачи данных входит команда типа MOV, которая применяется для пересылки операндов и имеет пять модификаций. При выполнении команды содержимое источника сохраняется.

Команды типа MOV A, Rr и MOV Rr, A служат для пересылки операнда из регистра Rr в ячейку внутренней памяти ОЗУ (или регистр специального назначения), адрес которой указан в команде, и наоборот из памяти или регистра специального назначения – в РОН. Вместо регистра Rr может быть использован ACC.

Для этого случая в составе команд имеются две команды со следующей мнемоникой MOV ad, A или MOV A, ad.

Команда с прямой адресацией MOV ad, ad типа память-память, служит для пересылки содержимого одной ячейки внутренней памяти ОЗУ или регистра специального назначения в другую ячейку внутренней памяти или регистр специального назначения.

Команды с косвенной адресацией MOV A, *a* R0 или MOV A, *a* R1 служат для пересылки в аккумулятор содержимого ячейки внутренней памяти данных (или регистра специального назначения), адрес которой указан в регистре R0 или R1 и наоборот, команды с косвенной адресацией MOV *a* R0, A и MOV *a* R1, A используются для пересылки содержимого аккумулятора в ячейку памяти внутреннего ОЗУ (или регистра специального назначения), адрес которой указан в регистре R0 или R1. К этому же типу относятся команды MOV ad, *a* R0; MOV ad, *a* R1 и MOV *a* R0, ad; MOV *a* R1, ad.

Команды: MOV A, #data; MOV Rr, #data; MOV ad, #data; - служат для пересылки 8-разрядного операнда, который содержится в команде, в аккумулятор или РОН или в ячейку внутренней памяти данных (регистр специального назначения). При выполнении команд MOV *a* R0, #data и MOV *a* R1, #data 8-разрядный операнд, содержащийся в команде, пересылается в ячейку внутренней памяти (или регистр специального назначения), адрес которой указан в регистре R0 или R1. Команда MOV DPTR, #data пересылает в указатель данных DPTR 16-разрядный операнд, указанный в команде. Причем в регистр DPL засылается младший байт операнда, а в регистр DPH – старший байт операнда.

Команды типа MOVX служат для обмена операндами между аккумулятором и ячейками внешней памяти данных.

Команды: MOV, *a* DPTR; MOVX A, *a* R0; MOVX A, *a* R1 – применяются для пересылки операнда из внешней памяти данных в аккумулятор. В этих командах используется косвенная адресация, причем, если указаны регистры R0 или R1, то операнд выбирается из внешней памяти данных, емкость которой не должна превышать 256 байт, а если в команде приводится указатель данных DPTR, то емкость внешней памяти может достигать 64 Кбайт.

Команды: MOVX a DPTR, A; MOVX a R0, A; MOVX a R1, A – наоборот используются для пересылки содержимого аккумулятора в ячейки внешней памяти данных.

Команды MOVC A, *a* A+DPTR и MOVC A, *a* A+PC с косвенной адресацией по сумме базового и индексного регистров используются для просмотра таблиц, записанных в памяти программ. В этих командах в качестве базового регистра служит аккумулятор, а в качестве индексного – указатель данных DPTR или счетчик команд PC. Содержимое ячейки памяти программ, адрес которой определяется суммой содержимого аккумулятора и DPTR (или PC), пересылается в аккумулятор.

Для работы со стеком служат две команды PUSH и POP. С помощью команды PUSH ad в стек записывается содержимое ячейки внутренней памяти данных (или регистра специального назначения), адрес которой указан в команде. С помощью команды POP ad из стека выбирается операнд, который засылается по адресу, указанному в команде.

К группе команд передачи данных могут быть также отнесены команда XCH обмена операндами между аккумулятором и внутренней памятью данных.

Команда XCH A, Rr служит для обмена содержимым аккумулятора и РОН. Команда XCH A, ad используется для обмена содержимым аккумулятора и приема адресуемой ячейкой внутренней памяти данных (или регистром специального назначения). Команда XCH A, *a* R0 и XCH A, *a* R1 аналогична предыдущей, но с косвенной адресацией.

Команды с косвенной адресацией: XCH DA, *a* R0 или XCH D A, *a* R1 – служат для обмена младшими тетрадами между аккумулятором и внутренней памятью данных.

Команда SWAP A служит для перестановки тетрад содержимого аккумулятора (младшая тетрада записывается на место старшей, а старшая – на место младшей).

Лекция 4

3.2.2. Команды обработки данных

Команды обработки данных можно разбить на команды арифметических и логических операций. При выполнении команд арифметических и логических операций формируются признаки, которые записываются в регистр PSW.

К арифметическим командам относятся команды сложения, вычитания, умножения и деления.

При выполнении команд сложения участвуют два операнда, один из которых обязательно содержится в аккумуляторе. Результат сложения также записывается в аккумулятор. Для записи команд сложения используется

мнемоника ADDC для случая учета признака C и мнемоника ADD для случая, когда признак переноса в сложении не участвует. Каждая из указанных команд имеет четыре модификации: сложение содержимого аккумулятора с содержимым регистра общего назначения Rr; сложение содержимого аккумулятора с содержимым ячейки внутренней памяти (регистра специального назначения), адрес которой прямо указан в команде; то же, но с использованием косвенной адресации; сложение содержимого аккумулятора с операндом, содержащимся в команде (непосредственная адресация). Запись команд сложения приведена в табл. П1.3.

В командах вычитания так же, как и в командах сложения участвуют два операнда, один из которых содержится в аккумуляторе. Результат вычитания записывается в аккумулятор. Вычитание производится с учетом заема (содержимого разряда переноса), и команда имеет мнемонику SUBB. В зависимости от использования того или иного способа адресации так же, как и для команд сложения, могут быть использованы четыре модификации (табл. П 1.3).

При выполнении команды умножения MUL AB перемножаются 8-разрядные двоичные числа без знака, содержащиеся в аккумуляторе ACC и в регистре специального назначения B. Младший байт двухбайтового результата помещается в аккумулятор ACC, а старший байт результата – в регистр B. Признак OV устанавливается в единицу, если произведение больше 255 (FFH), иначе признак OV равен нулю.

Признак переполнения C сбрасывается в нуль в любом случае.

При выполнении команды DIV AB 8-разрядное целое двоичное число без знака, содержащееся в аккумуляторе ACC, делится на 8-разрядное целое число без знака, содержащееся в регистре B. После выполнения операции деления в аккумулятор заносится целая часть частного от деления, а в регистр B – остаток от деления. Признаки OV и C сбрасываются в нуль. Если перед выполнением операции деления в регистре B содержится значение 00H (деление на нуль), то результат деления будет неопределенным и признак OV установится с единицу. Признак C сбрасывается в нуль в любом случае.

Команда DA A десятичной коррекции применяется для коррекции результата сложения двух операндов, представленных в двоично-десятичном коде.

Команда инкремента INC служит для увеличения на единицу содержимого аккумулятора регистров общего и специального назначения, содержимого ячеек внутренней памяти данных, а также указателя данных DPTR (табл. П 1.3).

Команда декремента DEC служит для уменьшения на единицу содержимого тех же регистров и ячеек внутренней памяти данных, перечисленных при описании команд INC, кроме указателя данных DPTR (табл. П 1.3).

К командам логических операций относятся команды ANL (поразрядного И), ORL (поразрядного ИЛИ), XRL (поразрядного

исключения ИЛИ) и команды сдвига вправо или влево содержимого аккумулятора ACC.

Команды ANL, ORL и XRL имеют одинаковые модификации и выполняют поразрядно соответствующую логическую операцию над операндами, один из которых содержится в аккумуляторе, а другой либо в РОН, либо в ячейках внутренней памяти данных (в регистрах специального назначения) (табл. П 1.3).

Команда RL A служит для циклического сдвига влево содержимого аккумулятора.

Команда RLC A служит для циклического сдвига влево 9-разрядного значения, которое определяется восемью разрядами содержимого аккумулятора и разрядом признака переноса C (сдвиг через разряд признака переноса).

Такого же типа команды используются для сдвига вправо: RR A – циклический сдвиг, вправо содержимого аккумулятора; RRC A – циклический сдвиг вправо содержимого аккумулятора через разряд переноса.

При выполнении команды CLP A значение всех разрядов аккумулятора сбрасывается в нуль, а при выполнении команды CPL A все разряды аккумулятора инвертируются.

3.2.3. Команды управления

К командам управления относятся команды условного и безусловного переходов, а так же команды вызова подпрограмм и возвращения из подпрограмм.

Команда безусловного перехода LJMP addr 16 используется для организации безусловного перехода по 16-разрядному адресу, указанному в команде. Адрес перехода может быть любым адресом из 64 Кбайт памяти программ.

Команда безусловного перехода AJMP addr 11 служит для организации безусловного перехода по 11 разрядному адресу, причем три старших разряда адреса (A10 – A8) указываются в старших разрядах (7 – 5) первого байта команды, а оставшиеся разряды адреса (A7 – A0) указываются во втором байте команды.

Команда SJMP rel выполняет безусловный переход по адресу, который вычисляется путем сложения смещения rel, указанного в команде, со значением PC+2, где содержимое счетчика команд PC соответствует значению адреса первого байта команды SJMP. Смещение rel обеспечивает изменение адреса относительно адреса команды SJMP на величину -128 – +127 байтов. Если переход осуществляется в сторону увеличения адреса, то значение смещения rel записывается в прямом коде. Если переход осуществляется в сторону уменьшения адресов, то значение смещения rel записывается в дополнительном коде. Ниже показаны фрагменты программы с использованием команды SJMP.

Переход к началу программы Переход в конец программы

```
0010 7F25M1: MOV R7, #25H   0010 7F25M1: MOV R7, #25H
0012 FD      MOV R5, A      0012 FD MOV R5, A
0013 80FB    SJMP M1        0013 8001 SJMP M3
                            0015 ED M2:  MOV A, R5
                            0016 EE M3: MOV A, R6
```

С помощью команд JMP *a* A+DPTR осуществляется косвенный переход по адресу, определяемому суммой содержимого аккумулятора A и 16-разрядного указателя данных DPTR. Сложение выполняется по модулю 2^{16}, причем содержимое A и DPTR не изменяется. Например, пусть в A содержится 05H, а в DPTR – 001CH. Тогда после выполнения команды JMP *a* A+DPTR осуществится переход к адресу 0021H.

Для организации перехода по нулевому значению результата, получаемого после выполнения арифметических и логических операций служит команда JZ rel, а по ненулевому значению результата – команда JNZ rel. Значение смещения выполняется так же, как и в команде SJMP rel.

Обычно при организации циклов применяется следующая последовательность команд: сначала производится загрузка счетчика циклов, затем осуществляется декремент и анализ его содержимого на нуль. Если содержимое счетчика не равно нулю, то производится переход к началу цикла.

В системе команд ОМЭВМ51 декремент и анализ содержимого счетчика выполняется с помощью одной двухбайтной команды DJNZ Rr, rel, где Rr – один из РОН, а rel – смещение, которое вычисляется так же как и в команде SJMP.

При выполнении трехбайтной команды DJNZ ad, rel уменьшается на единицу содержимое прямоадресуемой ячейки внутренней памяти данных (или регистра специального назначения) и, если оно не равно нулю, то осуществляется переход по адресу (PC)+3+rel, где (PC) – значение адреса первого байта команды DJNZ. Эта команда аналогична команде DJNZ Rr, rel.

Для организации сравнения двух операндов служат команды типа CJNE (табл. 3.2.1).

В указанных командах один операнд содержится в аккумуляторе, место другого определяется тем или иным способом адресации. Если сравниваемые операнды не равны. От осуществляется переход по смещению rel. При этом, если содержимое аккумулятора меньше второго операнда, то признак переноса C=1, иначе C=0. Оба операнда, участвующих в сравнении, не изменяются. Если оба операнда равны, то осуществляется переход к команде, следующей за командой CJNE.

Для вызова подпрограмм служит команда LCALL addr 16 по 16-разрядному адресу и команда ACALL addr 11 по 11-разрядному адресу. В последней команде восемь младших разрядов адреса записываются во втором байте команды, а три старших – в трех старших разрядах первого байта команды.

Таблица 3.2.1

Команды сравнения

Запись команды	Число байтов	Вычисление адреса перехода	Примечание
CJNE A, ad, rel	3	(PC)+3+rel	Второй операнд выбирается прямо из адресуемой ячейки внутренней памяти данных (или регистра специального назначения).
CJNE A, #data, rel	3	(PC)+3+rel	Второй операнд содержится в команде.
CJNE Rr, #data, rel	3	(PC)+3+rel	Первый операнд содержится в регистре, второй – в команде.
CJNE *@*R0, #data, rel CJNE *@*R1, #data, rel	3	(PC)+3+rel	Первый операнд выбирается по адресу, указанному в регистре R0(R1), второй операнд содержится в команде.

Для возврата из подпрограммы служит команда RET. В случае, когда вызов подпрограмм производится по прерыванию, возврат из подпрограммы осуществляется с помощью команды RETI.

3.2.4 Команды для выполнения операций с отдельными битами

При выполнении операций с отдельными битами в командах прямо указывается адрес бита. Адрес бита признак переноса C указывается неявно. Все команды побитовой обработки также можно разделить на команды передачи, логических операций и управления.

К командам передачи относятся MOV C, bit и MOV C, bit, где C – разряд переноса, а в поле bit записывается соответствующий адрес бита. С помощью указанных команд пересылается значение бита в разряд переноса или наоборот.

К командам логических операций относятся команда ANL – логическое «И» и команда ORL – логическое «ИЛИ».

С помощью команды ANL C, bit осуществляется логическое «И» признака переноса и значение прямоадресуемого бита, а с помощью команды ANL C,/bit выполняется логическое И инвертированного значения прямоадресуемого бита со значением разряда признака переноса.

Аналогично выполняются команды логического ИЛИ ORL C, bit и ORL C,/bit.

К командам управления относятся команды установки в единицу, сброса, инвертирования соответствующих битов, а также команды для

организации условного перехода к значению бита, адрес которого указан в команде.

Для установки в единицу используются команды SETB C и SETB bit, для сброса в нуль используются команды CLR C и CLR bit, а инвертирования – CPL C и CPL bit.

При выполнении команды JC rel осуществляется переход по значению признака переноса C=1. Адрес перехода вычисляется сложением содержимого счетчика команд и смещения rel. Аналогично вычисляется команда JNC rel, но переход осуществляется по значению C=0.

При выполнении команды JB bit, rel осуществляется переход, если значения прямоадресуемого бита равно единице. Адрес перехода вычисляется сложением содержимого счетчика команд и смещением rel. Аналогично выполняется команда JNB bit, rel, но переход осуществляется по значению прямоадресуемого бита, равного нулю. Значение бита при выполнении указанных команд не изменяется. Команда JBC bit, rel, но значение бита, участвующего в операции, сбрасывается в нуль.

3.3. Средства отладки программ

В настоящее время для отладки программ микроконтроллеров семейства MCS-51 имеются следующие средства:

– интегрированная отладочная среда mVision2;

– FLIP – гибкий внутрисистемный программатор (Flexible In-system Programmer) микроконтроллеров ATMEL с флэш памятью;

– дизассемблеры для MCS-51;

– компиляторы с языка ассемблера для микроконтроллеров семейства MCS-51.

Интегрированная отладочная среда mVision2mVision2 - новая отладочная среда фирмы Keil Software для микроконтроллеров семейства MCS-51. Она включает средства управления проектами, мощный текстовый редактор и многофункциональный отладчик в удобной программной оболочке. В комплект входит подробное руководство, в котором есть справочная информация по всем вопросам и раздел для быстрого освоения программы.

Поддерживаются микроконтроллеры фирм: Analog Devices, AMD, Atmel, Dallas Semiconductor, Philips.

FLIP (гибкий внутрисистемный программатор) – программа для Windows 9x/Me/NT/2000/XP или Linux. FLIP поддерживает внутрисистемное программирование флэш- микроконтроллеров C51 через интерфейс RS232 (Windows и Linux), порты USB или CAN (Windows). USB-драйвер для Windows WinDriver поставляется Jungo. Отличительные особенности и преимущества:

– Поддерживает внутрисистемное программирование (ISP) флэш-–микроконтроллеров семейства C51 фирмы Atmel через интерфейсы RS232, USB и CAN;

– Работает под платформами Windows и Linux;

– Не требуются дополнительные аппаратные средства;

– Мощный набор динамически загружаемых библиотек (DLL);

– Доступна версия под DOS (BatchISP);

– Вводит ISP-режим без ручных установок на целевой плате (AutoISP);

– Поддерживает основные CAN-интерфейсы (PEAK, IXXAT® и Vector);

– Отладочный режим для визуализации и проверки трафика между FLIP и целевым аппаратным обеспечением;

– Расширенная оперативная помощь;

– Свободно распространяемое инструментальное средство.

FLIP – свободно распространяемое программное инструментальное средство.

FLIP – мощный набор инструментальных средств, который дает возможность пользователю легко внедрить библиотеки функций внутрисистемного программирования в его приложение без необходимости вникать в особенности протокола программирования. FLIP позволяет увеличить гибкость, мощность и простоту внутрисистемного программирования флэш-микроконтроллеров Atmel семейства C51.

Дизассемблер предназначен для преобразования исполняемого кода микроконтроллеров MCS-51 в текст программы на языке ассемблера.

Форматы входных данных: HEX, OBJ, BIN.

Компиляторы с языка ассемблера для микроконтроллеров семейства MCS-51 позволяют преобразовывать исходный текст программы на языке ассемблера в объектный код и код микроконтроллера для ПЗУ. Программы для разных модификаций микроконтроллеров следует писать с учетом особенностей конкретного кристалла и его периферийных модулей.

Рассмотрим примеры разработки программ, написанных на языке Ассемблера, в интегрированной отладочной среде mVision2

Микроконтроллеры семейства MCS51 имеют специфические области памяти: память программ (внутренняя и внешняя), внутренняя и внешняя память данных.

Памяти программ соответствует сегмент кода CSEG, причем он может быть объявлен для любой доступной области внутренней или внешней памяти программ с помощью директивы AT. Например

cseg at 200h

Если директива AT не указана, то по умолчанию сегмент кода начинается с нулевого адреса.

Код может содержать область констант (например, строку символов). Тогда для удобства чтения программы код может быть разбит на несколько сегментов: сегменты кода и сегменты констант. В этом случае сначала присваиваются имена сегментам с помощью директивы SEGMENT, а затем указывается их место в программе с помощью директивы RSEG. Например,

```
prog segment code ; prog - имя сегмента
const segment code ; const - имя сегмента
cseg at 0
jmp start
org 40h
rseg prog
;текст программы
rseg const
string: db ‘ТЕМПЕРАТУРА’
end
```

Внутренняя память данных может содержать сегменты данных и сегмент области памяти с побитным доступом. Сегментам данных объявляется с помощью директивы *DSEG* для доступа к данным с любым способом адресации и *ISEG* для доступа к данным, размещенным по адресу, превышающим значение 7Fh(127), с косвенным способом адресации. Резервирование ячеек внутренней памяти данных осуществляется с помощью использования директивы *ds*. Например, с помощью следующей записи

```
dseg at 30h
var: ds 1
mass: ds 20
```

во внутренней памяти данных для переменной с именем var резервируется один байт, начиная
с адреса 30h, и для массива с именем mass резервируется 20 байтов, начиная с адреса 31h. Сегмент данных с побитным доступом обозначается как BSEG.
Стек размещается в любом месте внутренней памяти данных. Наиболее распространенным способом стек задается с помощью использования имени *stack* и резервирования для него необходимого числа байтов (глубины стека). Адрес дна незаполненного стека записывается в указатель стека SP командой mov в начале сегмента кода. Например,

```
dseg at 60h
stack: ds 30
cseg
jmp start
org 40h
start: mov sp,#stack-1
end
```

Сегмент внешней памяти данных обозначается как *XSEG*.

Лекция 5

4. Построение микропроцессорной системы на базе микроконтроллера семейства MCS51

4.1. Порты ввода/вывода

Квазидвунаправленный порт *P1* применяется как обычный порт ввода/вывода данных (в более поздних модификациях также может быть многофункциональным).

Порт P0 (рис.4.1) содержит фиксатор *LATCH* (защёлку) на триггерах *T*, два буфера 1 и 2,мультиплексор *MUX* и два выходных транзистора.

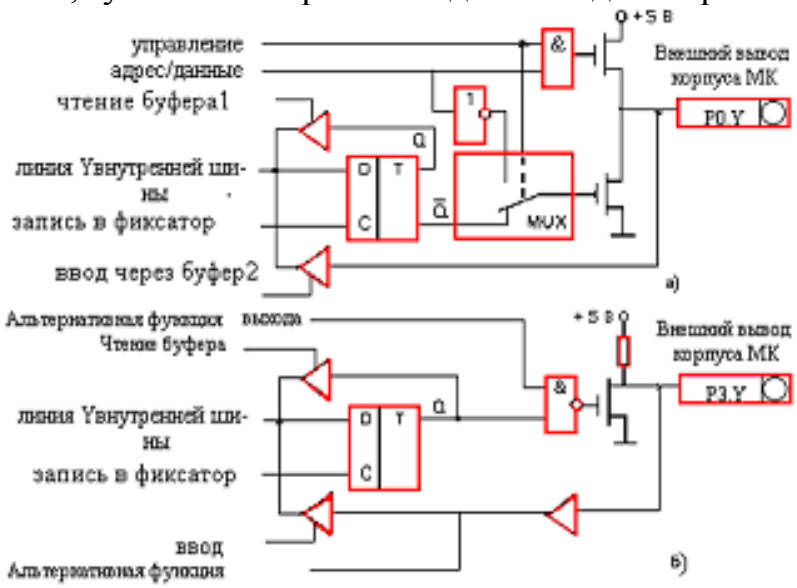


Рис.4.1. Схема для одного разряда портов микроконтроллера: а - порт0; б - порт3

При работе с внешней памятью (или другими устройствами, подключёнными к шинам адреса и данных) адрес или данные преобразуются в парафазный код, сигналы которого управляют затворами транзисторов. Если передаётся единица, то верхний транзистор открыт, а нижний закрыт. Когда передаётся нуль, то наоборот верхний транзистор открыт, а нижний закрыт. Адрес или данные сопровождаются разрешающим сигналом «управление». Этот сигнал управляет мультиплексором и верхним транзистором. Чтение данных с внешнего вывода производится через буфер 2. Оба транзистора в этом случае закрыты.

Если порт *P0* применяется как обычный порт ввод/вывода, то данные записываются в фиксатор с внутренней шины микроконтроллера. Инверсный выход Q триггера через мультиплексор подключён к нижнему транзистору, верхний транзистор всегда закрыт. Следовательно, состояние нижнего транзистора полностью определяется состоянием фиксатора. Ввод данных в этом случае возможен только при закрытом нижнем транзисторе и

осуществляется через буфер 2. Запись единицы в фиксатор обеспечивает закрытие нижнего транзистора. Состояние фиксатора может быть опрошено через буфер 1.

Поскольку верхний транзистор в режиме использования порта P0 как обычного порта ввода/вывода всегда закрыт, то при выводе единицы оба транзистора закрыты, что эквивалентно третьему состоянию, т.е. выходной ток высокого уровня равен нулю. Чтобы освободится от третьего состояния в этом случае, необходимо каждый вывод порта через внешний резистор подключать к источнику электропитания (рис. 4.2).

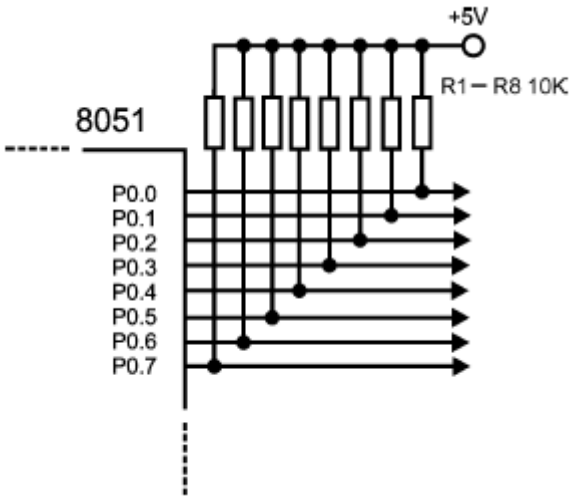


Рис. 4.2. Подключение внешних резисторов к порту P0

Как уже упоминалось, при обычном вводе через порт P0 необходимо следить за состоянием фиксатора, который должен находиться в состоянии единицы. В режиме работы с внешней памятью следить за состоянием фиксатора нет необходимости. Поэтому порт P0 называется двунаправленным.

В квазидвунаправленном порту (рис. 4.1. б) имеется фиксатор и один транзистор на выходе с встроенным резистором нагрузки. (В современных модификациях роль резистора выполняет управляемый транзистор). Работа квазидвунаправленного порта аналогична работе порта P0 в обычном режиме ввода/вывода, в котором необходимо следить за состоянием фиксатора при вводе информации. Поэтому порты P1, P2, P3 называются квазидвунаправленными.

Сигнал сброса RST устанавливает фиксаторы всех портов в состояние единицы.

4.1.1. Особенности выполнения команд при обращении к портам ввода/вывода

Работа с портами имеют некоторые особенности. Они касаются команд ввода данных с порта, в которых порт является одновременно операндом и местом размещения результата операции. В этих случаях реализуется специальный режим, который называется «чтение-модификация-запись». В этом режиме считывание содержимого порта осуществляется не с внешних

выводов, а из фиксатора (защёлки). Рассмотрим следующий пример. Пусть имеется следующая схема, показанная на рис.4.1.1 и выполняется следующая последовательность команд:

MOV A,#10101010B
MOV P!,A
ANL P!,#00101010B

Рассмотрим два случая:

1. Ключ Sw разомкнут, тогда после выполнения команды *ANL* состояние фиксатора и внешних выводов Pins совпадают.

Разряды	7	6	5	4	3	2	1	0
Фиксатор P1	0	0	1	0	1	0	1	0
Pins	0	0	1	0	1	0	1	0

2. Ключ Sw замкнут, тогда состояние внешних выводов отличается от состояния фиксатора.

Разряды	7	6	5	4	3	2	1	0
Фиксатор P1	0	0	1	0	1	0	1	0
Pins	0	0	0	0	1	0	1	0

В режиме «чтение-модификация-запись» ввод данных считается правильным не с внешних выводов, а из фиксатора.

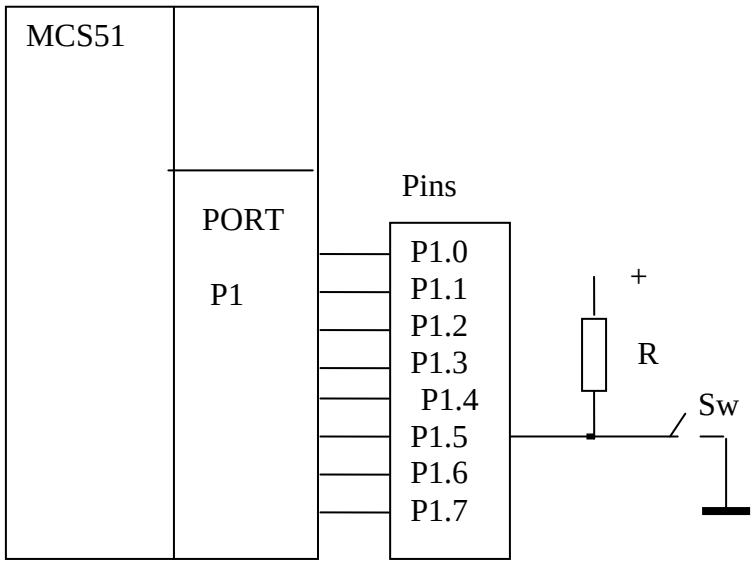


Рис. 4.1.1. Выполнение альтернативной команды ANL P1,#data

Ниже приведены команды, работающие в режиме «чтение-модификация-запись»:
– ANL поразрядное И;
– ORL поразрядное ИЛИ4
– XRL исключающее ИЛИ;

– JBC переход на метку, если бит равен 1;
– CPL инверсия бита;
– INC инкремент содержимого порта;
– DEC декремент содержимого порта;
– DJNZ декремент и переход на метку, если содержимое порта не ноль;
– MOV PX,Y пересылка бита переноса в бит Y порта X;
– CLR PX,Y очистка бита Y порта X;
– SETB PX,Y установка бита Y порта X.

4.2. Подключение внешних БИС памяти программ и данных

4.2.1. Временные диаграммы выполнения машинного цикла в микроконтроллере

Устройство управления микроконтроллера формирует машинный цикл, длительность которого равна 12 периодам тактовой частоты внутреннего генератора. Машинный цикл делится на шесть состояний *S1 – S6* (рис. 4.2..1), причём каждое состояние состоит из двух тактов *P1* и P2.

Команды выполняются либо за один машинный цикл, либо за два машинных цикла. Только команды умножения и деления выполняются каждая за четыре машинных цикла. Рассмотрим выполнение команд с помощью четырёх примеров: выполнение однобайтовой одноцикловой команды; выполнение духбайтовой одноцикловой команды; выполнение однобайтовой двухцикловой команды; выполнение команды обращения к внешней памяти.

Выполнение любой команды начинается с чтения кода операции. С этой целью микроконтроллер вырабатывает в состояниях *S1* и *S2* сигнал *ALE* независимо от того во внутренней или внешней памяти содержится команда. Если команда содержится во внешей памяти программ, то вырабатывается также сигнал *PSEN* (рис. 4.2.2), который отсутствует при чтении команд из внутренней памяти программ.

Если выполняется однобайтовая одноцикловая команда, то её чтение и выполнение осуществляется в течение трёх состояний *S1 – S3*. Во время состояний *S4 – S6* производится холостое чтение кода следующей команды (игнорируется), которая в микроконтроллере не выполняется.

Если во время состояния *S1 – S3* был принят код операции двух байтовой команды, то второй байт команды читается во время действия состояний *S4 – S6*. Таким образом, двухбайтовая команда выполняется за один машинный цикл.

Некоторые команды, например, команда инкремента указателя данных *INC DPTR*, являются однобайтовыми, но выполняются за два машинных цикла. В этом случае в первом машинном цикле во время действия состояний *S1 – S3* читается код операции, а в остальное время производится

инкремент указателя *DPTR*. Поскольку во время состояний *S4 – S6* первого машинного цикла и во время состояний *S1 – S3* и *S4 – S6* второго машинного цикла вырабатывается сигнал *ALE*, то происходит холостое считывание кода операции команды ,следующей за командой *INC DPTR.*

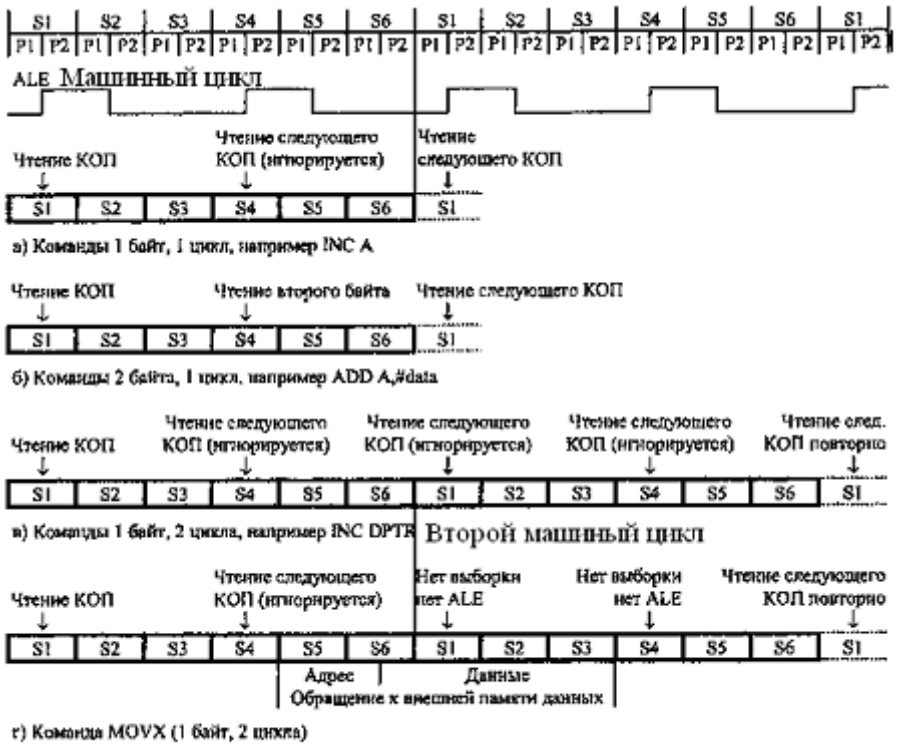
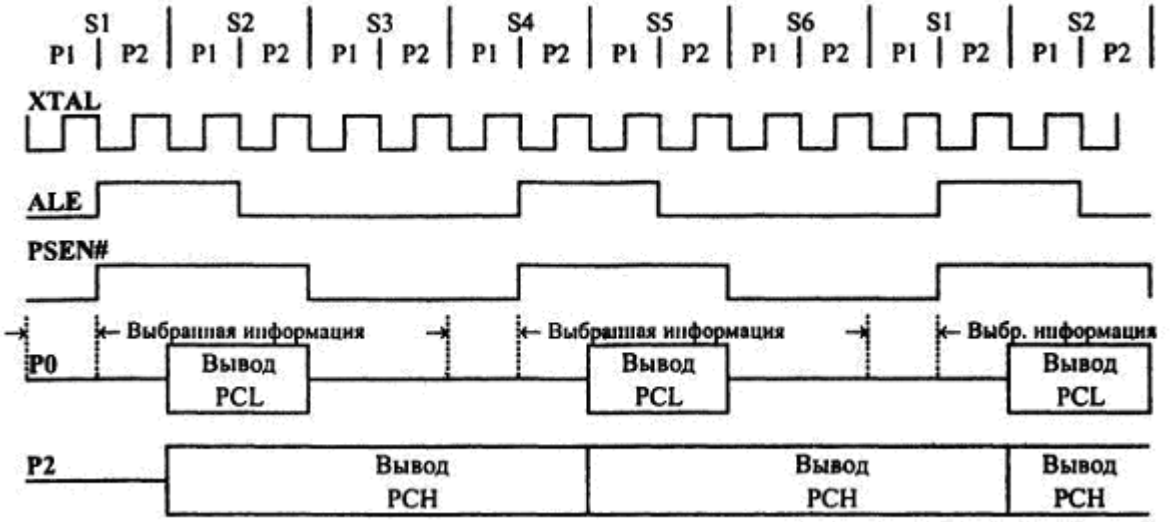


Рис. 4.2.1. Примеры выполнения команд в микроконтроллере

При выполнении команды выборки данных из внешней памяти данных (рис. 4.2.2 – 4.2.3) с помощью команды *MOVX A, @DPTR* или команды *MOVX @DPTR,A* в первой половине первого машинного цикла осуществляется чтение кода операции команды *MOVX*. Во второй половине первого машинного цикла производится холостое считывание кода операции следующей команда. А в первой половине второго машинного цикла в режиме (цикле) чтения по сигналу *RD* считываются данные из внешней памяти или в режиме записи (цикле) по сигналу *WR* данные записываются во внешнюю память данных. Во втором машинном цикле во время действия сигнала *RD* или *WR* сигнал *ALE* не вырабатывается.

Следует помнить, что если не используется внешняя память данных, то сигнал *ALE* генерируется во всех машинных циклах дважды и поэтому может быть использован как сигнал синхронизации для устройств, входящих в микропроцессорную систему.



Выборка команды из внешней памяти программ

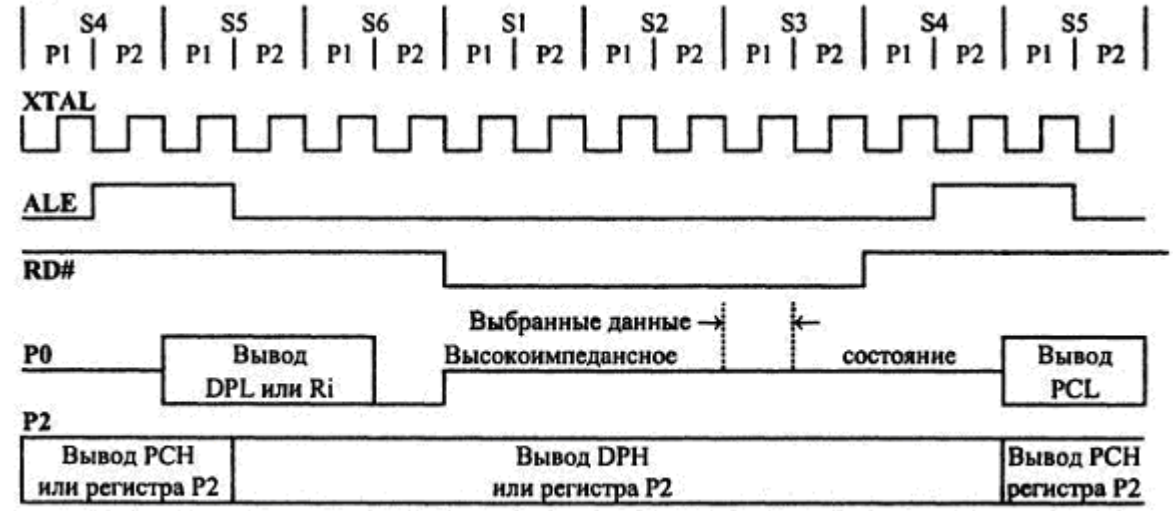


Рис. 4.2.2. Чтение из внешнепамяти данных

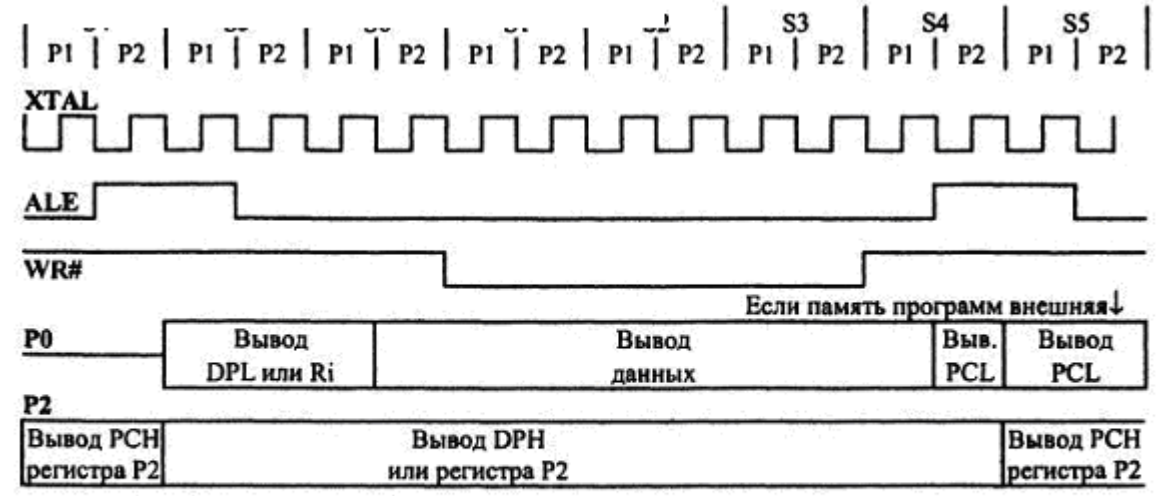


Рис.4.2.3. Запись во внешнюю память данных

4.2.2. Схема подключение внешних БИС памяти программ и данных

В микропроцессорных системах, построенных на базе микроконтроллера семейства *MCS51*, доступ к внешней памяти программ осуществляется с помощью управляющего сигнала *PSEN*, выполняющего функцию стробирующего сигнала для выбора внешней микросхемы ПЗУ (рис. 4.2.4). При обращении к внешней памяти устанавливается 16-разрядный адрес. Напомним, что старший байт адреса выводится с порта *P2*, а младший байт адреса с порта *P0*, причем вслед за младшим байтом адреса следует код команды. Разделение адреса от кода команды производится по времени с использованием сигнала ALE, во время действия которого младший бай адреса защёлкивается во внешнем регистре адреса *RGA*. Старший байт адреса запоминается и сохраняется неизменным в портеР2 в течение цикла обращения к памяти программ.

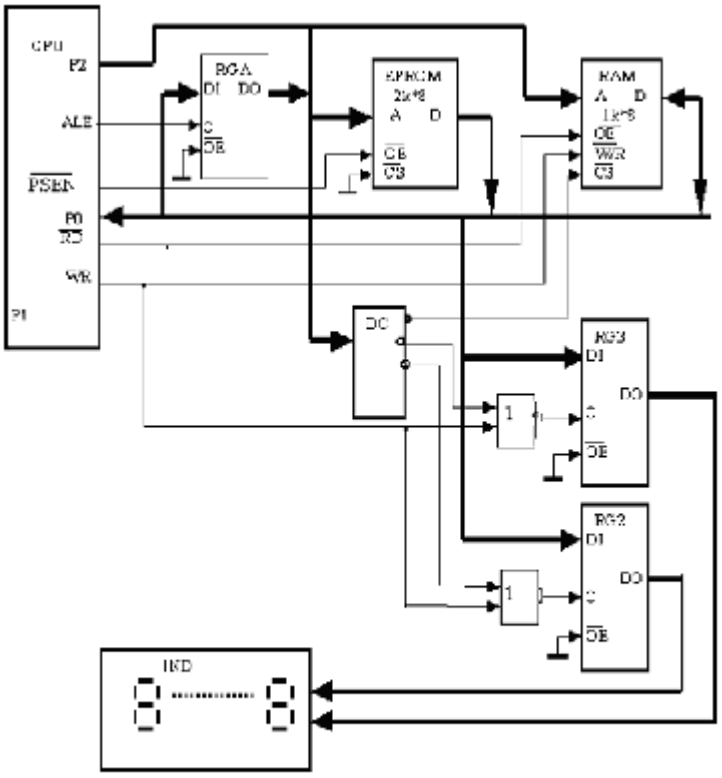


Рис.4.2.4. Структурная схема МПС

Доступ к внешней памяти данных возможен с использованием как 16-разрядного адреса (команды *MOVX A, @DPTR, MOVX @DPTR,A*), так и 8-разрядного адреса (команды *MOVX A,@R0, MOVX @R0,A*). При использовании 18-разрядного адреса старший бай устанавливается в порте *P2,* а младший байта адреса защёлкивается во внешнем регистре адреса *RGA*. При использовании 8-разрядного адреса порт *P2* может быть использован как обычный порт ввода/вывода. Если на вход W/R БИС внешней памяти данных поступает сигнал низкого уровня, то осуществляется циел записи, иначе – цикл чтения. Формирование циклов записи или чтения производится с помощью команд типа *MOVX*, во время выполнения которых с выводов порта P3 снимаются сигнал записи WR или сигнал чтения RD. Поскольку

сигналы записи и чтения взаимоисключающие то во время цикла чтения на выводе P3.6 (вывод WR) формируется сигнал высокого уровня, и поэтому на вход W/R БИС внешней памяти данных поступает сигнал высокого уровня, который служит для формирования режима чтения ОЗУ.
В тех случаях, когда необходимо подключать другие микросхемы, например, внешние порты, реализованных на регистрах, то используется дешифратор адреса DC, с помощью которого могут быть выбраны различные микросхемы, входящие в состав микропроцессорной системы (МПС).
В командах, которые изменяют значения в защёлках, портов P0 – P3, новые данные фиксируются в фазе S6P2 последнего машинного цикла. На выводе эти данные появляются в фазе S1P1 следующего цикла (рис. 4.2.5).

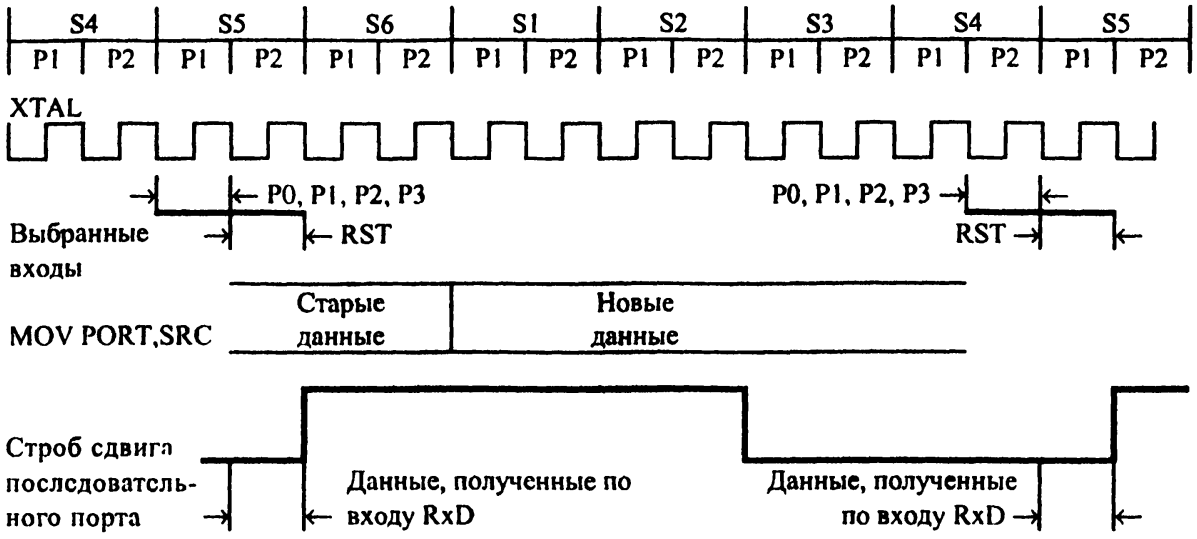


Рис. 4.2.5. Данные на выводах порта

В некоторых МПС имеет смысл совместить адресное пространство внешней памяти программ и памяти данных. В этом случае в качестве внешней памяти используется только микросхема ОЗУ, в одной части которой записана программа, а в другой данные, тогда с помощью схемы (рис.4.2.6) объединения сигналов чтения RD и *PSEN* из микросхемы ОЗУ могут быть прочитаны как коды команд так и данные.

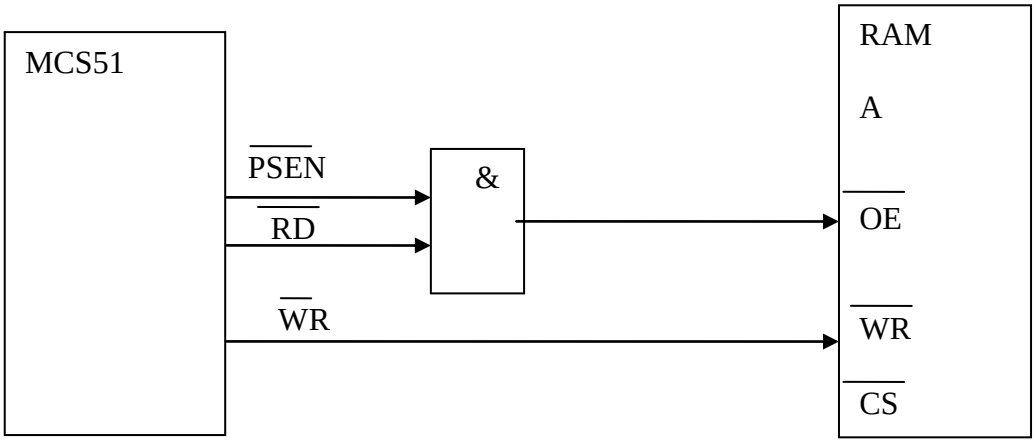


Рис. 4.2.6. Формирование сигнала чтения совмещённой внешней памяти

4.2.3. Стирание и программирование РПЗУ

В ранних модификациях память программ микроконтроллеров изготавливалась электрически программируемой со стиранием ультрафиолетовым излучением. Поэтому микроконтроллер необходимо оберегать от солнечного и комнатного освещения. При солнечном освещении содержимое РПЗУ стирается через 1 неделю, а при комнатном – через 3 года. Если необходимо стереть содержимое РПЗУ, то ОМЭВМ выдерживается под ультрафиолетовой лампой с излучением мощностью 12000 мкВт/см^2 в течение 20 – 30 минут на расстоянии примерно 2,5 см.

В режиме программирования РПЗУ тактовый генератор должен работать с частотой 4 – 6 МГц. Коды адреса ячеек РПЗУ (рис. 4.2.7) подаются на выводы порта P1 и выводы P2.0 – P2.3 порта P2 (рис. 4). Коды данных должны поступать на входы порта P0. На выводы P2.4 – P2.6 и вывод $\overline{PSEN}$ подается низкий уровень, а на вывод P2.7, $\overline{EA}$, ALE высокий уровень. На вывод RST подается напряжение +2.5 В.

Программирование выполняется в следующей последовательности. На вывод EA подается напряжение 21 В. Затем на вывод ALE, который имел высокий уровень, подается низкий уровень длительностью 50 мс. После на выводе $\overline{EA}$ восстанавливают высокий ТТЛ-уровень. Следует помнить, что на выводе $\overline{EA}$ недопустимо даже мгновенно превысить напряжение выше значения 21,5 В. Короткие выбросы напряжения выше указанного уровня могут привести к необратимым разрушениям БИС ОМЭВМ. Поэтому необходимо предпринимать дополнительные меры к стабилизации источника питания 21 В, чтобы защитить БИС ОМЭВМ от выбросов напряжения во время программирования.

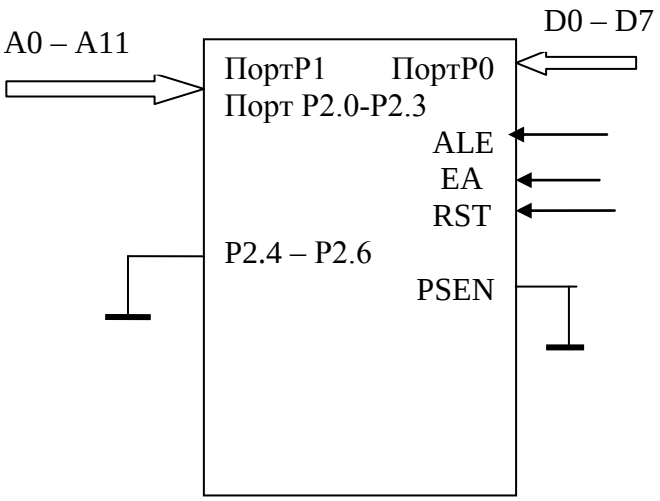


Рис. 4.2.7. Схема программирования внутренней памяти

После программирования необходимо проверить правильность кодов команд в РПЗУ. Адрес подается на выводы порта Р1 и Р2.0 – Р2.3 порта Р2. На выводах Р2.4 – Р2.6 и $\overline{PSEN}$ поддерживается низкий уровень, а на выводах ALE и $\overline{EA}$ – высокий. На вывод RST тоже должен быть подан уровень +2,5 В. С выводов порта Р0 считываются коды содержимого РПЗУ. Вывод Р2.7 используется для управления. Если на вход Р2.7 подать высокий уровень, то считывается неопределенная информация. Если на вход Р2.7 подать низкий уровень, то с выводов порта Р0 считывается содержимое ячейки памяти, адрес которой подается на входы портов Р0 и Р2. В режиме проверки РПЗУ к выводам порта Р0 требуется подключать внешние нагрузочные резисторы 10 кОм.

Лекция 6

4.3. Таймер/счетчик

В состав ОМЭВМ51 входят два 16-разрядных многорежимных таймера/счетчика. Регистры таймера/счетчика 0 обозначаются TH0 и TL0, а регистры таймера/счетчика 1 – TH1 и TL1. Каждая пара регистров независимо одна от другой может работать либо как таймер, либо как счетчик. Режим таймера используется для задания интервалов времени, а режим счетчика – для счета внешних событий (счет входных импульсов, поступающих на входы порта Р3 от внешнего источника).

С помощью таймера могут быть заданы интервалы времени в диапазоне от длительности выполнения одного машинного цикла до длительности выполнения 65536 циклов. При тактовой частоте внутреннего

генератора 12 МГц диапазон изменения интервала времени таймера составляет 1 – 65536 мкс.

Максимальная частота входных сигналов для их счета в режиме счетчика составляет1/24 тактовой частоты внутреннего генератора, что составляет – 500 кГц, если ft=12 МГц.

Выбор режимов осуществляется с помощью регистра TMOD (адрес 89H), а управление таймером/счетчиком – с помощью регистра TCON (адрес 88H). В табл. 4.3.1 приведен формат слова TMOD для задания режимов.

Таблица 4.3.1

Формат слова TMOD

Номер разряда	7 6 5 4	3 2 1 0
Обозначение разряда	GATE $\bar{T}$/C M1 M0	GATE $\bar{T}$/C M1 M0
Номер таймера	Таймер/счетчик 1	Таймер/счетчик 0

В табл. 4.3.2 приведен формат слова TCON для выбора значений разрядов управления таймером/счетчиком. Назначение младших четырех разрядов рассматривается в параграфе, в котором описывается организация прерывания.

Таблица 4.3.2

Формат слова TCON

Номер разряда	7 6 5 4	1 2 3 4
Обозначение разряда	TF1 TR1 TF0 TR1	IE1 IT1 IE0 IT0
		Используются для организации прерываний

Выбор режимов работы таймера/счетчика 0 осуществляется с помощью установки в нуль или единицу разрядов 0 – 3 регистра TMOD, а таймера/счетчика 1 – с помощью разрядов 4 – 7.

Значение GATE (TMOD 7 или TMOD 3) служит для разрешения запуска таймера/счетчика. Если разряд GATE установлен в единицу, то таймер/счетчик запускается в том случае, когда на входах P3. 3 для (таймера/счетчика 1) и P3. 2 (для таймера/счетчика 0) порта P3 поступает сигнал высокого уровня и значение разряда TR1 (TR0) регистра TCON равно единице. Если разряд GATE установлен в нуль, то таймер/счетчик запускается в том случае, когда значение разряда TR1 (TR0) установлено в единицу, а значение уровня сигнала на входе P 3. 3 (P3. 2) может быть любым.

Значение $\bar{T}$/C (TMOD . 6 TMOD . 2) определяет работу в режиме таймера/счетчика. Если разряд $\bar{T}$/C установлен в единицу, то выбирается режим счетчика и внешние сигналы, предназначенные для счета, должны быть подключены к входам P 3. 5 (P 3. 4) порта P3. Если разряд $\bar{T}$/C в нуль,

то выбирается режим таймера и сигналы на вход таймера поступают от внутренних схем синхронизации.

С помощью разрядов M1 и M0 задаются четыре варианта загрузки регистров TH и TL и управления ими. Каждый из вариантов соответственно называется режимом 0, 1, 2 или 3.

Режим 0 (13-разрядный таймер/счётчик) устанавливается при значении разрядов M1=0 и M0=0. В этом режиме (рис. 4.3.1) в регистре TL1 (TL0) 3 значения трёх старших битов неопределенны и значащими являются только 5 младших битов. В регистр TH1 (TH0) загружается 8-разрядное значение. При переполнении регистра TH1(TH0) в единицу устанавливается разряд TF1 (TF0) регистра управления TCON. Признак TF1 (TF0) может быть опрошен программно либо использован как источник прерывания от таймера/счетчика. Таким образом могут быть заданы интервалы времени в диапазоне [8192 – (начальное значение)] · Тмц, где Тмц – длительность машинного цикла или произведен счет внешних сигналов в диапазоне от 1 до 2^{13}. Следует заметить, что режим 0 совпадает с режимом работы таймера/счетчика ОМЭВМ К1816ВЕ48.

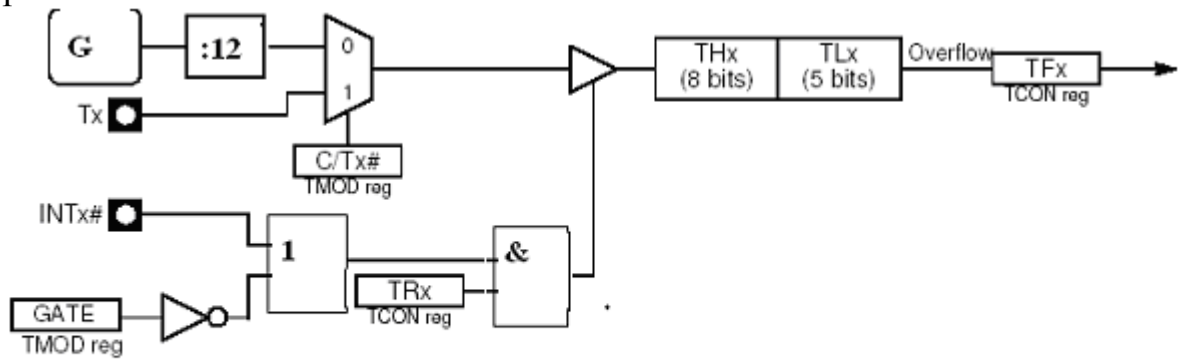


Рис.4.3.1. **Схема таймера/счётчика в режиме 0**

Режим 1 устанавливается при значении разрядов M1=0 и M0=1. В режиме 1 (рис.4.3.2) таймер/счетчик работает как 16-разрядное устройство. Старший байт начального 16-разрядного значения заносится в регистр TH1 (TH0), а младший байт – в регистр TL1 (TL0). Разряд TF1 (TF0) устанавливается в единицу после переполнения 16-разрядного регистра, составленного из регистров TH и TL.

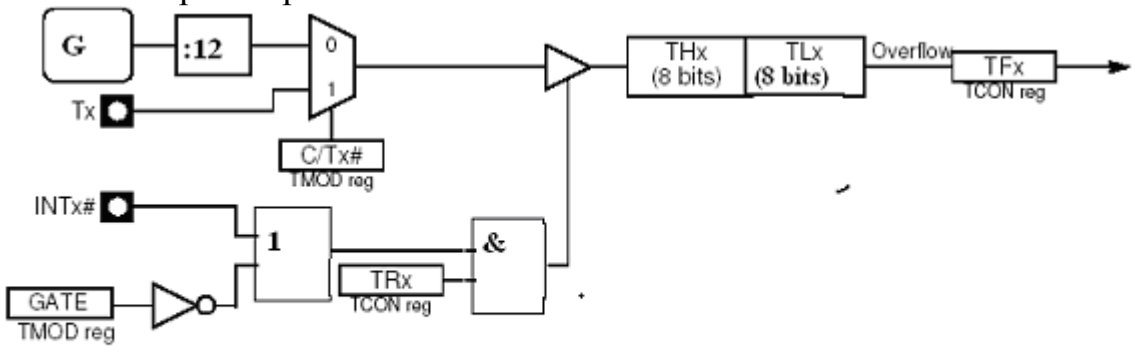


Рис. 4.3.2. **Схема таймера /счётчика в режиме 1**

Режим 2 (рис.4.3.3) устанавливается при значении разрядов M1=1 и M0=0. В режиме 2 таймер/счетчик является 8-разрядным и автоматически перезагружается при каждом переполнении. В этом режиме начальное значение загружается в регистр TH1 (TH0), которое переписывается в TL1 (TL0) при каждом переполнении. Перезагрузка не меняет содержимое TH1 (TH0).

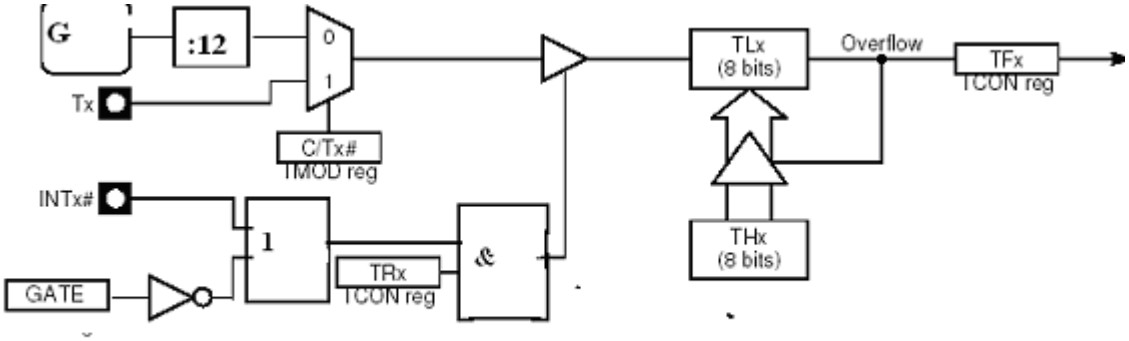


Рис.4.3.3. **Схема таймера/счётчика в режиме 2**

Режим 3 устанавливается при значении разрядов M1=1 и M0=1. В режиме 3 (рис.4.3.4) таймер/счетчик 1 и таймер/счетчик 0 работают по-разному. В этом случае таймер/счетчик 0 работает как 8-разрядный таймер с использованием регистра TH0 и разряда и разряда переполнения TF1 и как 8-разрядный счетчик с использованием регистра TL0 и разряда переполнения TF0. Запуск таймера 0 производится при установке в единицу разряда TR1. Запуск счетчика 0 производится сигналами TR0, GATE и $\overline{INT0}$.

Если таймер/счетчик 0 настроен на режим 3, то таймер/счетчик 1 может быть использован в любом из режимов, но без анализа разряда переполнения. Запуск таймера/счетчика 1 в этом случае производится автоматически после загрузки слова в регистр TMOD. Когда таймер/счетчик 0 работает в режиме 3, таймер/счетчик 1 может быть использован, например, как генератор, задающий скорость передачи последовательного порта.

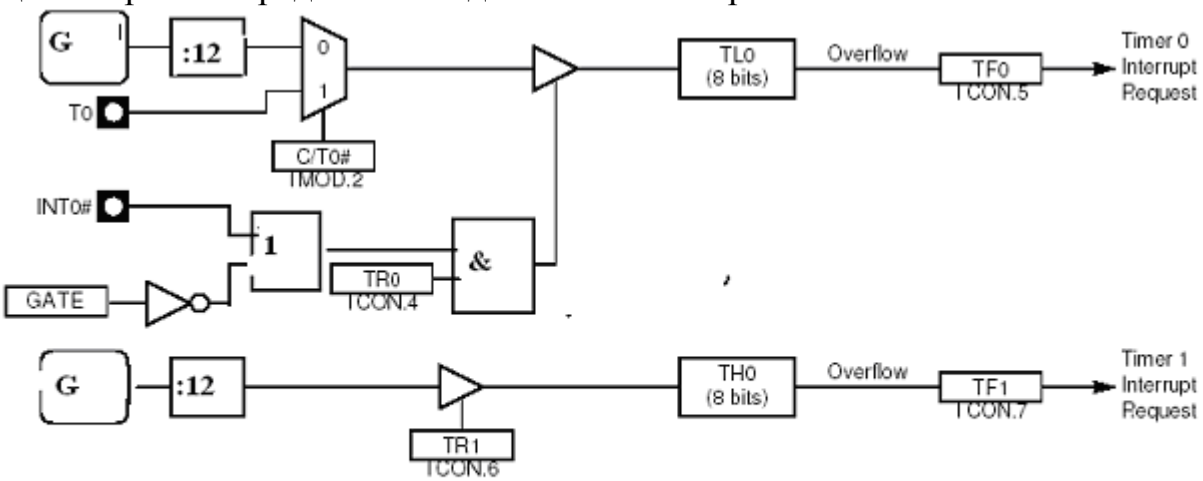


Рис.4.3.4. **Схема таймера/счётчика в режиме 3**

В качестве примера рассмотрим использование таймера 0 в режиме 2 для формирования импульса на выводе P1.0 порта P1.

Программа имеет следующий вид:

```
MOV TMOD, #02; установка режима 2 таймера/счетчика 0
MOV TH0, #0FCH; загрузка начального значения таймера
M2: SETB TCON . 4; запуск таймера
        CLR P1 . 0; сброс в «0» разряда 0 порта P1
M1: JNB TCON . 5, M1; проверка переполнения регистра TH0
        SETB P1 . 0; установка в «1» разряда 0 порта P1
CLR TCON . 5; сбор признака переполнения
JNB P1 . 1, M2; проверка состояния разряда 1 порта P1.
```

Первая команда программы загружает в регистр TMOD слово, соответствующее режиму 2 таймера/счетчика (M1=1, M0=0). Разряд 2 регистра TMOD устанавливается в нуль, поскольку выбирается режим таймера. Разряд 3 также устанавливается в нуль, чтобы осуществлять запуск таймера программно, не используя управление по сигналу $\overline{INT0}$. Таймер/счетчик 1 не используется и поэтому остальные разряды регистра TM0 не определяются и в данном примере установлены в нули.

Вторая команда устанавливает регистр TH0 в состоянии 0FCH, и, следовательно, при запуске таймера его переполнение наступит через 4 машинных цикла или через 4 мкс при длительности машинного цикла 1мкс.

Третья команда запускает таймер и с этого момента начинается отсчет длительности интервала, заданной в регистре TH0.

С помощью четвертой команды CLR P1 . 0 разряд P1 . 0 порта P1 устанавливается в нуль. При условии, что предварительно этот разряд был установлен в единицу. Например, после начальной установки при включении питания системы или после подачи сигнала СБРОС (RST).

При выполнении пятой команды осуществляется программная проверка признака переполнения TF0. Как только переполнение наступает, то осуществляется переход к следующей команде SETB P1 . 0, которая установит разряд P1 . 0 порта P1 в единицу. Таким образом, на выводе P1 . 0 сформируется импульс низкого уровня длительностью в 4 машинных цикла. Если потребуется повторить формирование импульса, то это может быть выполнено, если использовать проверку состояния разряда P1 . 1 порта P1. Если внешнее устройство (например, объект управления) сформирует на указанном выводе порта P1 сигнал низкого уровня, то с помощью команды JNB P1 . 1, M2 можно осуществить возврат на метку M2 и формирование импульса на выводе P1 . 0 повторится. В этом случае необходимо сбрасывать признак переполнения TF0 командой CLR TCON . 5. Если на выводе P1 . 1 окажется сигнал высокого уровня, то осуществится выход из цикла.

Указанная программа имеет один недостаток, заключающийся в том, что в течение всего времени формирования импульса необходимо проверять состояние признака TF0 и процессор не может выполнять другие операции. Этот недостаток может быть преодолен, если использовать выход из цикла

по прерыванию, возникающему при переполнении таймера. Организация прерываний рассматривается в следующем параграфе.

Укажем, что установка разряда GATE в единицу регистра TMOD позволяет организовать запуск таймера аппаратно по наличию высокого уровня на входе INT порта P3. В этом случае с помощью таймера может быть измерена длительность сигнала, поступающего на вход INT.

Рассмотрим случай формирования интервала времени более, чем 65535*$T_{МЦ}$, где $T_{МЦ}$ длительность машинного цикла.Например, пусть требуется сформировать интервал времени длительностью 1сек. Разобьём интервал 1сек на 20 участков, длительностью 50000мкс. Полученная длительность одного участка может быть сформирована с помощью таймера в режиме 1. С этой целью необходимо в регистры TH и TL загрузить начальное значение (65536-N), где N=50000/ $T_{МЦ}$. число импульсов, которые необходимо подать на вход таймера, чтобы получить его переполнение через 50000мкс. Время машинного цикла равно

$$T_{МЦ} = 12/ Ft,$$

где Ft – частота внутреннего тактового генератора. Тогда для Ft=12Мгц время машинного цикла равно $T_{МЦ}$ =1мкс и N=50000 (для Ft=11,0592Мгц N=46083). Чтобы получить время 1сек необходимо повторить загрузку таймера 20 раз и каждое переполнение таймера запоминать в памяти данных. Ниже приведён пример текста программы формирования интервала времени 1сек.программы

```
        repit equ 20
        N    equ 50000
        N_sec equ 60
        dseg at 30h
        sec: ds 1
        cseg
        jmp begin
        org 0040h
begin:
        mov r2,#0
        mov r3,#0
        mov tmod,#00000001b    ; режим 1 таймера 0
        mov th0,#high(65536-N);загрузка старшего байта начального
                               ;значения
        mov tl0,#low(65536-N) );загрузка младшего байта начального
                               ;значения
        setb tr0               ;запуск таймера
clock:  jnb tf0,clock          ; программная обработка переполнения
        clr tf0
        mov th0,#high(65536-N)
        mov tl0,#low(65536-N)
        inc r2                 ;накопление двадцати повторений по 50000мкс
        cjne r2,repit,end_clock
```

```
        mov r2,#0
        inc r3              ;накопление секунд
        cjne r3,#n_sec, end_clock
        mov sec,r3
        mov r3,#0
        jmp exit    ;выход через n секунд (в данном примере через
                    ;минуту)
end_cloc:   jmp clock
; обработка интервала времени, заданного числом n_sec
exit:
```

Лекция 7

4.4. Организация прерываний

В микроконтроллере могут быть обработаны сигналы от пяти источников прерываний: два по переполнению встроенных таймеров/счетчиков, два от внешних источников прерывания и один по состоянию последовательного порта. Прерывания от каждого из указанных источников могут быть независимо друг от друга разрешены или запрещены, причем каждому источнику может быть присвоен соответствующий приоритет. Источник с более высоким приоритетом может прервать программу обслуживания прерывания источника с более низким приоритетом.

Для контроля состояния признаков прерываний служат два регистра TCON (адрес 88H) и SCON (адрес 98H). Форматы слов, загружаемых в регистры приведены в табл. 4.4.3. Управление системой прерывания обеспечивается с помощью регистров IE (адрес A8H) и IP (адрес 0B8H). Форматы слов, которые записываются в регистры, приведены в табл. 3.3. Содержимое всех регистров может быть опрошено программно.

Обработка прерываний от внешних источников осуществляется при поступлении сигналов на входы INT1 (INT0), причем формирование признака прерывания IE1 (IE0) в регистре TCON производится либо по уровню внешнего сигнала, либо по спадающему фронту внешнего сигнала. Выбор уровня или фронта обеспечивается соответствующей установкой разрядов IT1 (IT0) регистр TCON. Если разряды IT1 (IT0) установлены в единицу, то обеспечивается формирование признака прерывания по спадающему фронту сигнала, поступающего на вход порта P3, иначе – по уровню сигнала.

Таблица 4.4.3

Обозначение разрядов, необходимых для организации прерываний

Номер разряда	7	6	5	4	3	2	1	0
Регистр TCON					IE1	IT1	IE0	IT0

Регистр SCON							T1	R1
Регистр IE	$\overline{EA}$	-	-	ES	ET1	EX1	ET0	EX0
Регистр IP	-	-	-	PS	PT1	PX1	PT0	PX0

Разрешение обработки прерывания от внешних источников осуществляется установкой в единицу разрядов EX1 (EX0) регистра IE. Если прерывания разрешены, то обеспечивается переход к программе обслуживания прерывания, адрес которой определяется в соответствии с табл. 4.4.4.

Обработка прерываний по переполнению таймер/счетчик производится в том случае, когда разряды TF1 (TF0) устанавливаются в единицу и прерывания разрешены (разряды ET (ET0) регистра IE установлены в единицу). Значения разрядов TF1 (TF0) автоматически сбрасываются в нуль при выполнении первой команды подпрограммы обслуживания прерывания.

Формирование признаков RI и TI осуществляется по готовности приемника или передатчика последовательного порта. Разрешение прерывания от этого источника обеспечивается установкой в единицу разряда ES регистра IE.

Таблица 4.4.4

Адрес подпрограммы обслуживания прерываний

Источник запроса прерывания	Признак	Позиция в регистре	Адрес подпрограммы
Внешний источник прерывания 0	IE0	TCON.1	0003H
Внешний источник прерывания 1	IE1	TCON.3	0013H
Встроенный таймер/счетчик 0	TF0	TCON.5	000BH
Встроенный таймер/счетчик 1	TF1	TCON.7	001BH
Приемник последовательного порта	R1	SCON.0	0023H
Приемник параллельного порта	TI	SCON.1	0023H

На рис. 4.4.1 показаны источники прерываний и их вектора.

Запрещение обработки Прерываний сразу от всех источников выполняется с помощью установки в нуль разряда EA регистра IE. Установка признака EA в единицу разрешает обработку прерываний от любого из пяти источников в зависимости от состояния разрядов 0 – 4 регистра IE.

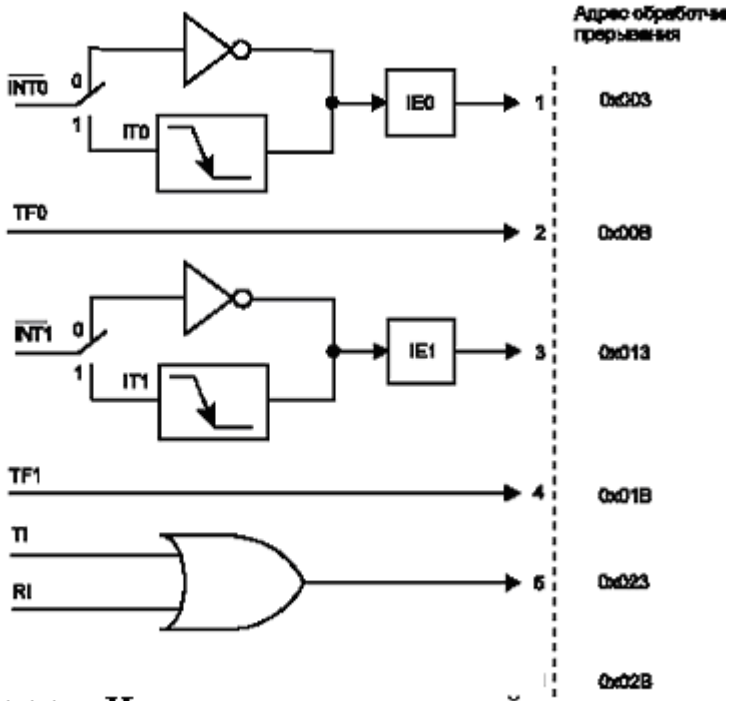


Рис.4.4.1. **Источники прерываний**

Приоритеты источников прерываний указывается с помощью установки в соответствующее состояние разрядов 0 – 4 регистра IP (табл. 4.4.5).

Таблица 4.4.5

Уровни приоритета прерываний

Обозначение	Позиция в регистре	Источник запроса прерываний	Приоритет
PX0	IP.0	Внешний источник 0	0 (высший)
PT0	IP.1	Таймер/счетчик 0	1
PX1	IP.2	Внешний источник 1	2
PT1	IP.3	Таймер/счетчик 1	3
PS	IP.4	Последовательный порт	4 (низший)
	IP.5 – IP.7	Резервные	

Действие механизма приоритетов прерываний заключается в выборе одного из источников при одновременном приходе нескольких запросов, а также в принятии решения о прерывании текущей программы обслуживания прерывания вновь поступившем запросом. Все источники прерываний проверяются на наличие запроса во время фазы S5P2 каждого машинного цикла (рис. 4.4.2). В течение следующего машинного цикла анализируются биты регистра приоритетов IP и выполняется внутренний выбор (поллинг) более приоритетного запроса.

Наиболее быстрый вариант, когда C2 является последним машинным циклом команды (кроме RETI, а также команд обращения к IE и IP).

Рис.4.4.2. **Вызов процедуры обслуживания прерывания**

Обработка прерываний начинается после выполнения текущей команды прерываемой программы до конца и запоминание содержимого счетчика команд (адреса возврата) в стеке. В счетчик команд загружается один из адресов, указанных в табл. 4.4.4. В ячейке памяти по этому адресу должна быть записана команда типа JMP безусловного перехода к начальному адресу подпрограммы обслуживания прерываний. Подпрограмма обслуживания прерываний должна начинаться командами записи в стек слова состояния программы PSW, аккумулятора, указателя данных и всех регистров, значение которых может быть изменено в процессе выполнения подпрограммы, и заканчиваться командами POP восстановления из стека. В конце подпрограммы обслуживания прерывания обязательно должна быть записана команда RETI возврата из подпрограммы, после выполнения которой в счетчик команд загружается из стека адрес возврата.

Минимальное время, необходимое на переход к обработчику прерывания, составляет 38 тактов.

МЦ	МЦ	МЦ
S5P2 S6	Поллинг	Переход к обработчику
Запрос T T	запросов	(CALL или JMP)
2T	12T	24T

Максимальное время, необходимое на переход к обработчику прерывания для случая завершения текущей команды умножения или деления, составляет 86 тактов.

Следует учесть, что в случае обработки внешнего запроса прерывания по спадающему фронту , бит IE в регистре TCON очистится после выполнения первой команды обработчика. Если используется обработка внешнего запроса прерывания по уровню, то его следует программно сбросить до завершения программы обработчика, чтобы не было повторного вызова обработчика.

Пример составления программы для обработки прерывания.

```
cseg
jmp begin
org 0003h
call prog1
reti
```

```
        org  000bh
        call prog2
        reti
          .
          .
          .
bgin:   setb IE.x          ;разрешение одного из источников прерываний
        orl IE,#10000000b; разрешение прерываний
```

Лекция 8

5. Организация последовательного ввода/вывода

5.1. Организация последовательного ввода-вывода в микроконтроллерах семейства MCS – 51 (UART)

Порт последовательного ввода/вывода используется в асинхронном режиме для связи со стандартными периферийными устройствами (дисплеем, телетайпом, модемом и др.), а также для объединения нескольких микроконтроллеров. Для организации последовательного ввода/вывода используется регистр специального назначения SBUF (адрес 99H), который обеспечивает обмен данными между регистром SBUF и сдвигающими регистрами приема и передачи информации. Запись байта в регистр SBUF приводит к автоматической переписи байта в сдвигающий регистр передатчика и последовательную передачу байта на внешнее устройство через вывод P3.1 порта P3. Использование регистра SBUF при приеме позволяет совмещать операцию чтения ранее принятого байта с приемом очередного байта. Если к моменту окончания приема байта предыдущий байт не был считан из регистра SBUF, то он будет потерян.

Программно могут быть заданы четыре режима работы последовательного порта ввода/вывода. Для управления последовательным обменом и задания режимов служит регистр SCON (адрес 98H). В табл. 5.1.1 показан формат слова, которое необходимо загрузить в регистр SCON для организации последовательного ввода/вывода.

Таблица 5.1.1

Формат слова SCON

Номер разряда	7	6	5	4	3	2	1	0
Обозначение разрядов	SM0	SM1	SM2	REN	TB8	RB8	T1	R1

С помощью разрядов SM0 и SM1 устанавливаются режимы, которые имеют наименование 0, 1, 2, или 3.

Режим 0 устанавливается при значении SM0=0 и SM1=0. Этот режим используется для ввода и вывода последовательного 8-разрядного кода данных через разряд P3.0 (RXD) порта P3. Через вывод P3.1 (TXD) порта P3 выдаются импульсы синхронизации, которые сопровождают каждый переданный или принятый бит. Частота синхронизации равна 1/12 частоты внутреннего генератора ОМЭВМ, т.е. за один машинный цикл последовательный порт принимает или передает один бит данных.

Передача в режиме 0 начинается в следующем машинном цикле после выполнения команды, по которой в регистр SBUF записывается байт данных. Затем выполняется преобразование параллельного8-разрядного кода данных в последовательный и на десятом машинном цикле, начиная с цикла записи данных в регистр SBUF, признак T1 регистра SCON устанавливается в единицу.

Прием в режиме 0 обеспечивается при значении разрядов REN=1 и RI=0 регистра SCON. После установки указанных разрядов со следующего машинного цикла начинается формирование синхросигналов, которые выдаются через вывод P3.1 (TXD) порта P3. В момент выдачи каждого синхроимпульса вводится один бит данных и на десятом машинном цикле, начиная с установки значений REN и RI в регистре SCON, признак RI перебрасывается в состояние единицы.

Режим 0 обычно применяется для организации синхронного обмена информацией. В этом случае используются внешние микросхемы для реализации логических функций управления синхронным обменом.

Режим 1 устанавливается при значении разрядов SM0=0, SM1=1 (рис.5.1.1).

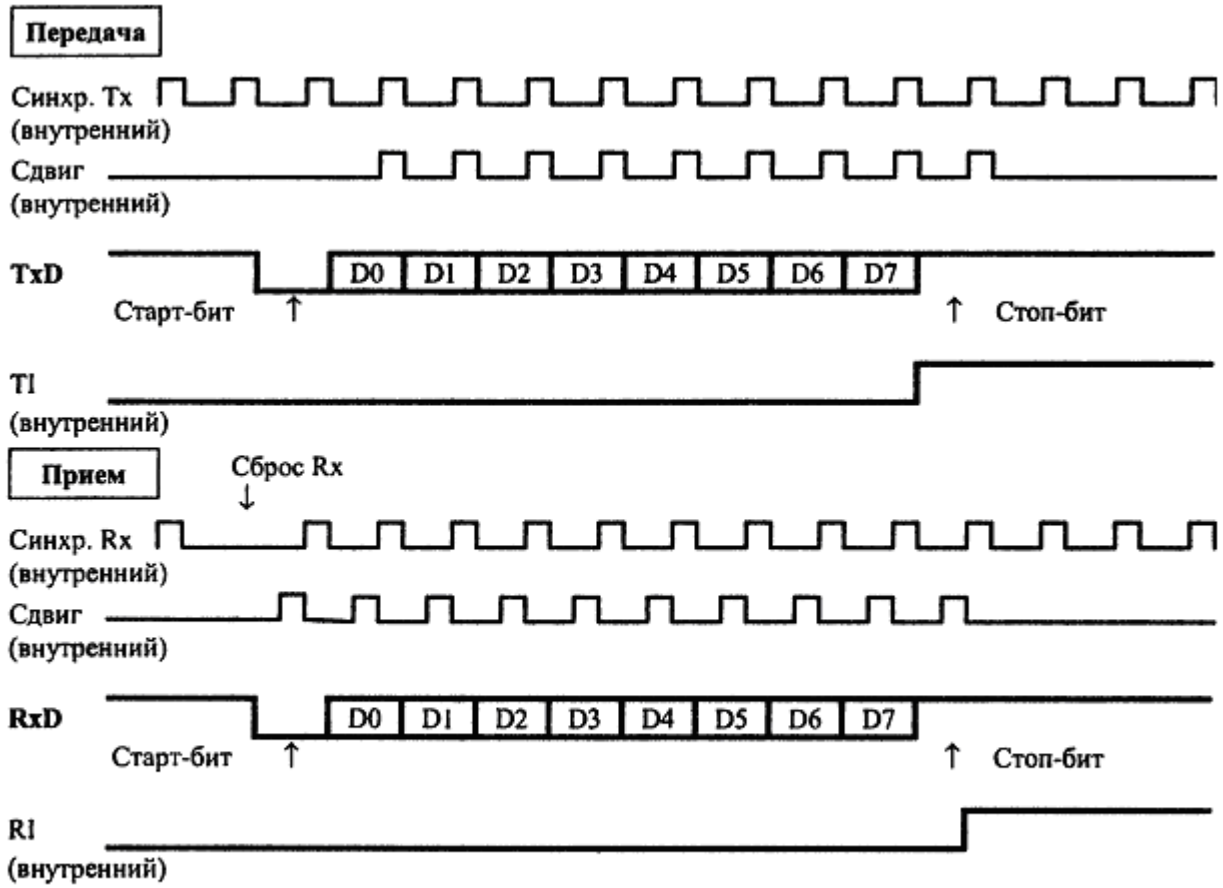


Рис.5.1.1. **Работа последовательного порта в режиме 1**

В этом режиме осуществляется асинхронная старт/стоповая передача через вывод TXD или прием с вывода RXD десяти бит информации: стартового бита, восьми разрядов данных (причем первым передается, принимается младший разряд данных) стопового бита. Скорость передачи или приема переменная и задается таймером 1, причем предварительно осуществляется деление на 32 частоты внутреннего генератора.

Режим 2 (рис.5.1.2) устанавливается при значении разрядов SM0=1, SM1=0. В этом режиме осуществляется асинхронная старт/стоповая передача через вывод TXD или прием с вывода RXD одиннадцати бит информации. Отличие от режима 1 состоит в том, что к восьми разрядам данных добавляется разряд контроля четности. Скорость передачи/приема составляет 1/32 и 1/64 частоты внутреннего генератора или – в нуль для скорости 1/64 частоты внутреннего генератора.

Режим 3 устанавливается при значении разрядов SM0=1, SM1=1. Этот режим полностью совпадает с режимом 2 за исключением правила выбора скорости передачи/приема, которая устанавливается так же, как и в режиме 1.

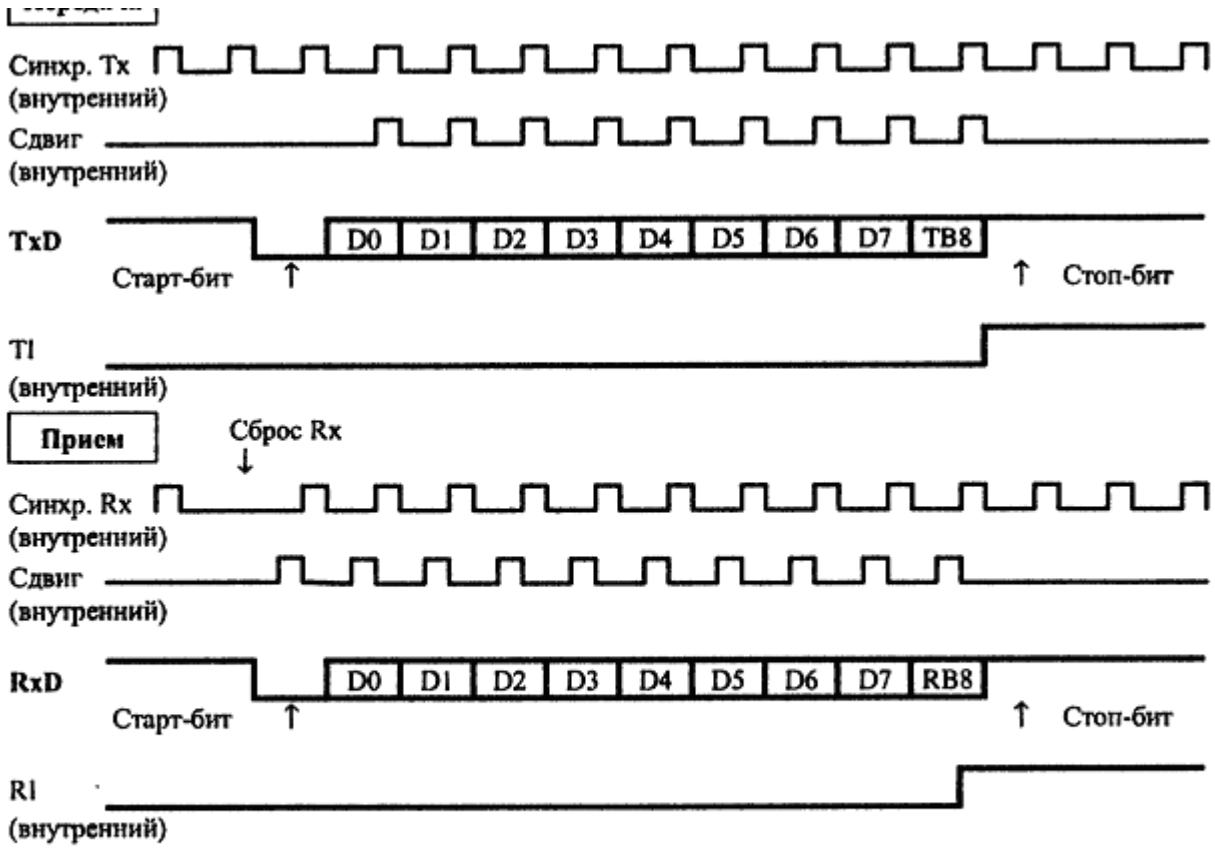


Рис.5.1.2. **Работа последовательного порта в режимах 2 и 3**

Рассмотрим пример программирования таймера 1 для задания скорости передачи 2400 бод в режиме 1 или 3 при тактовой частоте внутреннего генератора 12МГц. Если f_T=12МГц, то таймер работает с частотой счета 1 МГц и для получения скорости 2400 необходимо частоту 1 МГц разделить на величину 32·2400=76800. В этом случае для таймера получим коэффициент деления, равный 13 (точнее 13,02). Поскольку таймер обнуляется при переполнении, то для обеспечения деления на 13 (0DH) необходимо найти величину начального значения таймера 1. Для его работы выбирается режим 2 перезагружаемого таймера. В этом случае в регистр TH1 загружается значение F3H. Программа выбора скорости последовательной передачи/приема имеет следующий вид:

```
MOV TCON,#0010xxxB; выбор режима 2 таймера 1, значения разрядов ;
                  ;таймера/счетчика 0 не определены
MOV TH1, #0F3H    ; загрузка в TH1 коэффициента деления
SETB TR1          ; запуск таймера 1
```

С помощью разряда SM2 регистра SCON может быть заблокирован прием байта данных, в котором девятый бит имеет значение, равное нулю. Содержимое разряда SM2 устанавливается/сбрасывается программно.

Разряд REN служит для управления приемом данных. Значение разряда REN устанавливается/сбрасывается программно и разрешает (REN=1) или запрещает (REN=0) последовательный прием данных.

Разряд TB8 служит для заполнения признака четности при передаче, а разряд RB8 – при приеме. Добавление разряда четности при передаче может быть осуществлено с помощью выполнения следующей последовательности команд:

```
ADD A, 0        ; содержимое А не меняется, но формируется признак Р
MOV C, P        ; признак Р пересылается в разряд переноса С
MOV TB8, C      ; признак С помещается в разряд 9 буфера передачи
MOV SBUF, A     ; загрузка буфера передачи байтом данных и начало
                ;последовательной передачи.
```

Разряд T1 используется как признак готовности передатчика передавать следующий байт данных. Признак T1 устанавливается в единицу после окончания передачи байта данных и может быть опрошен программно или служить запросом на прерывание. В любом случае бит T1 сбрасывается программно.

Для иллюстрации составления программы последовательной передачи и приема данных рассмотрим два примера.

Программа передачи массива *mass* имеет следующий вид:

```
xseg
MASS:    ds 128
MASS2: ds 128
ORL P3,#02              ; установка вывода TXD в «1»
MOV SCON, #0100xx1xB    ; режим 1, разряд T1 установлен в «1»
MOV TMOD, #0010xxxxB    ; установка таймера 1 в режим 2
MOV TH1, #0F3H          ; установка скорости передачи 2400 бод
SETB TR1                ; запуск таймера
MOV DPTR,#MASS          ; загрузка в DPTR адреса массива MASS
MOV R3,#128             ; загрузка в R3 числа пересылаемых байтов
                        ; (например 128 байтов)
M1: MOVX A, a DPTR      ; загрузка в аккумулятор байта из массива
                        ;MASS
M2: JNB T1, M2          ;проверка состояния готовности передатчика
CLR TI                  ; сброс TI
MOV SBUF,A              ; передача байта данных
INC DPTR                ; инкремент DPTR
DJNZ R3,M1              ; возврат к передаче пока R3 не обнулится
```

Программа приема данных и запись его в массив *MASS2* имеет следующий вид:

```
ORL P3,#01          ; установка вывода RXD в «1»
MOV SCON,#50H       ; установка режима 1 для последовательного ввода
MOV TMOD,#20H           ; установка таймера 1 в режим 2
MOV TH1,#0F3H       ; установка скорости приема 2400 бод
SETB TR1            ; запуск таймера
MOV DPTR, #MASS2    ; загрузка в DPTR адреса массива MASS2
```

```
MOV R3,#128          ; загрузка в P3 число пересылаемых байтов
M3:  JNB RI, M3      ; проверка готовности приемника
     CLR RI          ; сброс признака готовности
     MOV A,SBUF      ; пересылка в А содержимого буфера приемника
     MOVX a DPTR, A  ; пересылка содержимого А в массив В
     INC DPTR        ; инкремент DPTR
     DJNZ R3, M3          ; возврат к приему пока R3 не обнулится
```

Работа программ можжет быть проверена для двух микроконтроллеров по следующей схеме (рис. 5.1.3).

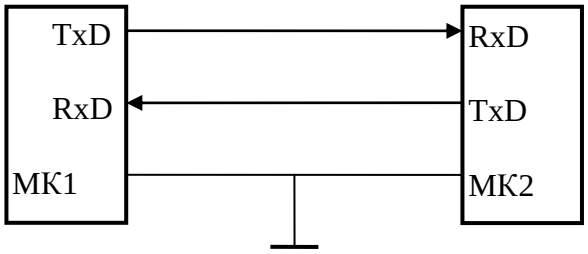


Рис. 5.1.3. Последовательная передача и приём данных

Для выполнения указанных программ необходимо сначала запустить программу приёма микроконтроллера МК2, который переходит в состояние ожидания, поскольку разряд RI регистра SCON установлен в нуль, только после того, как из МК1 поступит байт данных со стартовым битом в начале, разряд RI установится в единицу и осуществится преобразование последовательного кода в параллельный.

5.1.2. Последовательный интерфейс RS232

Используя микроконтроллеры семейства MCS51, легко реализовать интерфейс *RS232* - популярный протокол, применяемый для связи компьютеров с модемами и другими периферийными устройствами. Могут быть использованы режимы синхронной и асинхронной передачи данных, определяемые стандартом *EIA RS*-232-*C* и рекомендациями *V.24 CCITT*. Изначально создавался для связи компьютера с терминалом. В настоящее время используется в самых различных применениях. Интерфейс *RS-232*-C соединяет два устройства. Интерфейс *RS-232C* предназначен для соединения аппаратуры, передающей или принимающей данные (ООД – оконечное оборудование данных или АПД – аппаратура передачи данных, иначе *DTE* –

Data Terminal Equipment). К АПД можно отнести компьютер и другое периферийное оборудование. Тогда связь между ними может быть обозначена как «*DTE* – интерфейс *RS-232C* – *DTE*». Если требуется соединить устройства АПД через линию связи, то АПД подключаются к оконечной аппаратуре каналов данных (АКД, иначе *DCE* – *Data Communication Equipment*) В качестве АКД может быть использован модем. В этом случае связь может быть обозначена как «*DTE* – интерфейс *RS-232C* – *DCE* – линия связи – *DCE*– интерфейс *RS-232C* – *DTE*».

Стандарт интерфейса *RS-232C* описывает управляющие сигналы, пересылку данных, электрическое соединение и типы разъёмов. В персональном компьютере (*PC*) интерфейс *RS-232C* реализован с помощью *COM*-порта. На входе приёмника логической единице соответствует сигнал напряжением в диапазоне –12В…..–3В, логическому нолю – +3В…+12В. Для формирования указанных сигналов передатчиком и преобразования их к уровням ТТЛ в приёмнике выпускаются специальные микросхемы (например, *ADM202, MAX202*). На аппаратуре АПД (в том числе на выходах *COM*-порта) принято устанавливать вилки (*male* – папа), а на аппаратуре АКД (модемах) – розетки(*femal*e – мама). Разъёмы имеют 25 или 9 контактов. Все 9 сигналов интерфейса задействуются только при соединении компьютера с модемом.

На рис.5.1.4 показано соединение типа *DTE* – *DTE* с помощью минимального варианта нуль-модемного кабеля. В качестве устройства *DTE* может быть подключена микропроцессорная система, построенная на основе микроконтроллера.

Назначение выводов разъема *COM* следующие:

— ***TD*** - данные, передаваемые компьютером в последовательном коде (передатчик);

— ***RD*** - данные, принимаемые компьютером в последовательном коде (приемник);

— ***DTR*** - готовность выходных данных;

— ***DSR*** - готовность данных. Используется для задания режима модема;

— ***RTS*** - сигнал запроса передачи. Активен во все время передачи;

— ***CTS*** - сигнал сброса (очистки) для передачи. Активен во все время передачи; —***DCD*** - обнаружение несущей данных (детектирование принимаемого сигнала);

— ***RI*** - индикатор вызова. Приеме модемом сигнала вызова по телефонной сети;

— ***SG*** - сигнальное заземление, нулевой провод.

Преобразование параллельного кода в последовательный для передачи данных и обратное преобразование при приеме осуществляется с помощью

Обозначение вывода	№ контакта	
	разъем 9	разъем 25
TD	3	2
RD	2	3
DTR	4	20
DSR	6	6
RTS	7	4
CTS	8	5
DCD	1	8
RI	9	22
SG	5	7

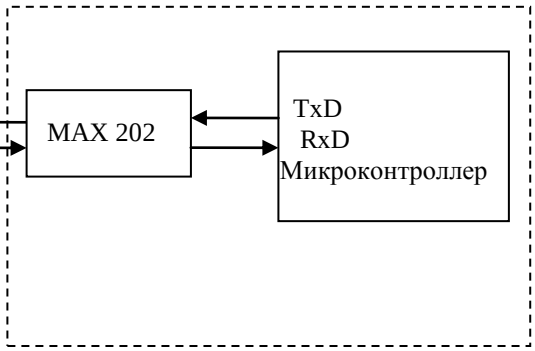


Рис. 5.1.4. Соединение типа *DTE – DTE*

устройства универсального синхронного асинхронного приемника передатчика УСАПП (*UART*) Для реализации устройства *UART* выпускаются специализированные микросхемы. В микроконтроллерах почти всегда присутствует последовательный универсальный асинхронный приемопередатчик (*UART*). Его выход и вход можно соединить с аналогичным входом и выходом другого микроконтроллера или персонального компьютера и передавать данные (соблюдая согласование сигналов по уровню и полярности).

Длина кабеля влияет на максимальную скорость передачи информации. Более длинный кабель имеет большую емкость и соответственно для обеспечения надежной передачи более низкую скорость. Большая емкость приводит к тому, что изменение напряжения одного сигнального провода может передаться на другой смежный сигнальный провод. Максимальным расстоянием обычно считается равным 15 м, но это не установлено в стандарте. К основным недостаткам интерфейса *RS232* относятся низкая скорость передачи данных и низкая помехоустойчивость сигнала и соединение двух устройств (двухточечное соединение).

Для улучшения характеристик последовательного интерфейса *RS232* существует ряд родственных стандартов: *RS423*, RS422, *RS485*. Лучшими характеристиками обладает интерфейс *RS485*, в котором для передачи сигнала используется дифференциальный метод.

Дифференциальный метод передачи обладает меньшей чувствительностью к общим помехам, чем простая однопроводная схема. Дифференциальный метод передачи использует двухпроводную схему соединения с формированием перепадов инверсией тока или напряжения в отличие от однопроводной простой схемы передачи информации. Достоинством дифференциального метода является то, что шумы наводящиеся на двухпроводной линии симметричны и не нарушают дифференциального

сигнала к которому чувствителен приёмник. Дифференциальный метод так же обладает меньшей чувствительностью к искажениям сигнала от внешних магнитных полей. Для формирования дифференциального сигнала для последующего передачи его в двухпроводную линию многие фирмы мира выпускают микросхемы интерфейса *RS485*. Лидером в разработке и выпуске новых микросхем интерфейса *RS485* является известная фирма *MAXIM* Интерфейс *RS485* многоточечный и наиболее часто используется при создании современных локальных сетей различного назначения в промышленных изделиях. Основными преимуществами интерфейса являются использование всего трех проводов (третий, общий, не всегда является обязательным) и повышенную нагрузочную способность. Если раннее большинство микросхем было насчитано на работу с 32 станциями, то современные модели обеспечивают нормальное функционирование до 256 станций.

В настоящее время выпускаются микросхемы с высокой предельной скоростью передачи. Это позволяет создавать высокоскоростные сети, и снижает количество ошибок в сети за счет улучшения формы передаваемого сигнала. На рис.5.1.5 показана структура сети с использованием микросхемы *MAX3443E*, назначение выводов которой следующие:

– *RO — Receiver Output* — Выход приемника. Если на линии A сигнал больше сигнала на линии *B* на 200*mV*, то RO=1, иначе *RO=0*;

– *RE — Receiver Output Enable* — Разрешение выхода приемника при *RE=0*. При *RE=1* выход *RO* находится в высокоимпедансном состоянии;

– *DE — Driver Output Enable* — Разрешение выходов передатчика. Если *DE=1* выходы активны, в противном случае они находятся в высокоимпедансном состоянии;

– *DI — Driver Input* — Вход передатчика;

– *GND — Ground* — Общий провод питания;

–*A — Noninverting Receiver Input* and *Driver Output* — Линия для неинвертирующего входа/выхода;

– *B — Inverting Receiver Input and Driver Output* — Линия для инвертирующего входа/выхода.

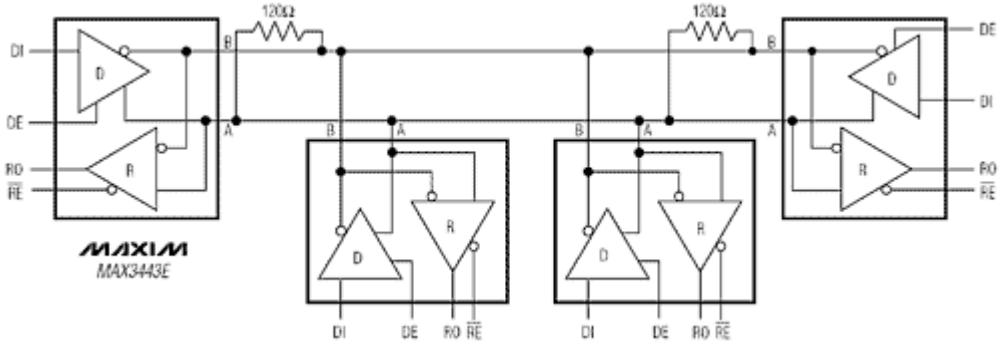


Рис. 5.1.5. Структура сети на базе интерфейса RS485

Развитие дифференциального метода получил в способе передачи информации дифференциальными сигналами малых напряжений *LVDS* (*Low Voltage Differential Signaling*). Это направление передачи данных использует очень малые перепады дифференциального напряжения на двух линиях печатной платы или сбалансированного кабеля со скоростью до сотен и даже нескольких тысяч мегабит в секунду.

Лекция 9

5.2..Организация последовательных интерфейсов I^2C, SPI

5.2.1. Организация последовательного интерфейса I^2C

Разработанная фирмой *Philips* шина I^2C (Inter-Integrated Circuit), - это двунаправленная асинхронная шина с последовательной передачей данных и возможностью адресации до 128 устройств. Физически шина I^2C содержит две сигнальные линии, одна из которых (*SCL*) предназначена для передачи тактового сигнала, вторая (*SDA*) для обмена данными. Для управления линиями применяются выходные каскады с открытым коллектором, поэтому линии шины должны быть подтянуты к источнику питания +5В через резисторы сопротивлением 1─10 кОм, в зависимости от физической длины линий и скорости передачи данных. Длина соединительных линий в стандартном режиме может достигать 2-х метров, скорость передачи до 400 кбит/с. Суммарная емкость линий должна быть не больше 400 пФ, входная емкость на каждую ИС должна быть в пределах 5...10 пФ.

Все абоненты шины делятся на два типа устройств: ведущее (*Master*) и ведомое (*Slave*) устройства. Тактовый сигнал *SCL* генерирует только ведущее устройство. Оно может самостоятельно выходить на шину и обращаться по адресу к любому ведомому устройству При обнаружении собственного адреса и, распознав его, ведомые устройства выполняют предписываемую операцию. Кроме того, возможен так называемый *"Multi Master"* – режим, когда на шине установлено несколько ведущих абонентов, которые либо совместно разделяют общие ведомые устройства, либо попеременно являются то ведущим устройством, то ведомым. Режим *"Multi Master"* требует арбитража и распознавания конфликтов. Естественно, он сложнее для создания программного обеспечения и, как следствие, реже используется в реальных изделиях.

В начальный момент времени (в режиме ожидания) обе линии *SC*L и *SDA* находятся в состоянии логической единицы (рис.5.2.1). Затем формируется старт условие, с которого начинается передача пакета информации, состоящего из одгого или нескольких байтов. Передача пакета заканчивается формированием стоп условия. Старт условие образуется при отрицательном перепаде линии *SDA*, когда линия *SCL* находится в единичном состоянии, и наоборот, стоп условие образуется при положительном перепаде линии *SDA* при единичном состоянии линии *SCL*. Изменение бита данных на линии *SDA*

производится при нулевом состоянии линии *SCL*. Прежде чем передать следующий бит ведущее устройства всегда должно проверять состояние линии *SCL*, если ведомое устройство установит линию *SCL* в низкий уровень, то ведущее устройство должно дождаться момента, когда на линии *SCL* установится высокий уровень.

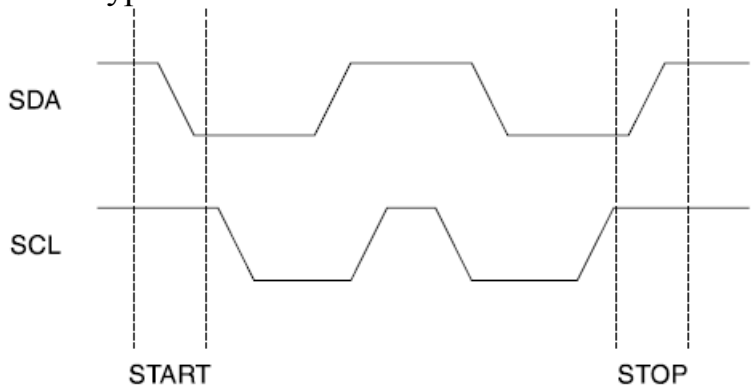


Рис. 5.2.1. Формирование старт и стоп условий на шине I^2C

Обмен информацией осуществляется байтами. Каждый байт передается в течении 9 тактовых периодов сигнала синхронизации на линии SCL. В девятом такте устройство, передавшее байт (8 периодов сигнала SCL) должно по линии SDA получить подтверждение ACK – низкий уровень на линии SDA (рис.5.2.2).

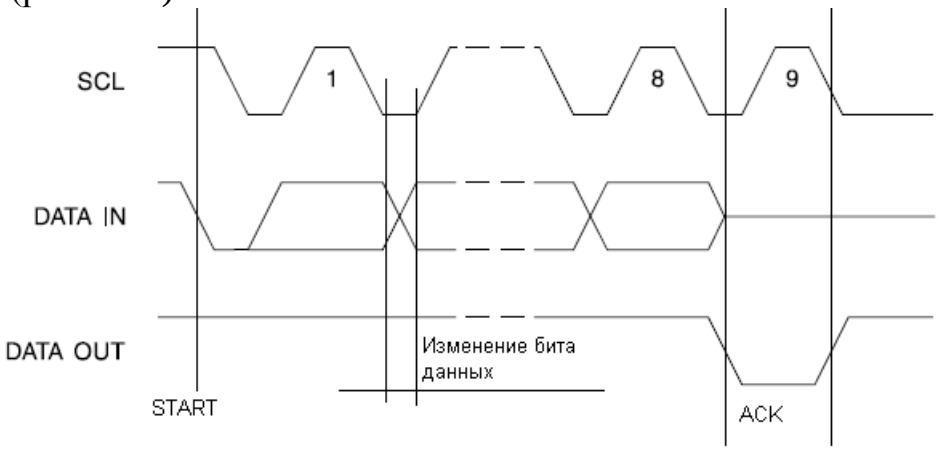


Рис. 5.2.2. Прием байта и формирование подтверждения *ACK*

Если в девятом периоде на линии *SDA* установился высокий уровень, то это свидетельствует о возникшей ошибке в приемнике (отсутствие подтверждения *NAK*). Следует отметить, что передатчиком может быть как ведущее, так и ведомое устройства и, следовательно, получатель информации всегда должен формировать подтверждение *ACK*.

Протокол передачи данных покажем на примере использования микросхемы памяти *EEPROM AT24C16* фирмы Atmel с произвольным доступом, емкостью 2к х 8. В режиме чтения обмен может быть осуществлен со скорость до 400 кбит/с. В режиме записи после каждых 16 байтов требуется время 10 мс для записи байтов во внутренние ячейки памяти

микросхемы. Если от микросхемы во время передачи данных не поступает подтверждение *ACK*, то ведущее устройства должно подождать, пока данные не запишутся в память. Память микросхемы разбита на восемь блоков по 256 байтов. Микросхеме присвоен 7-разрядный адрес, который состоит из 4-разрядной последовательности 1010 и 3х-разрядного номера блока.

Чтобы начать запись данных в микросхему памяти *AT24C16* (рис.5.2.3) ведущее устройство должно в линию *SDA* выдать *START* условие, за которым следует семиразрядный адрес ведомого устройства (*DEVICE ADDRES*).

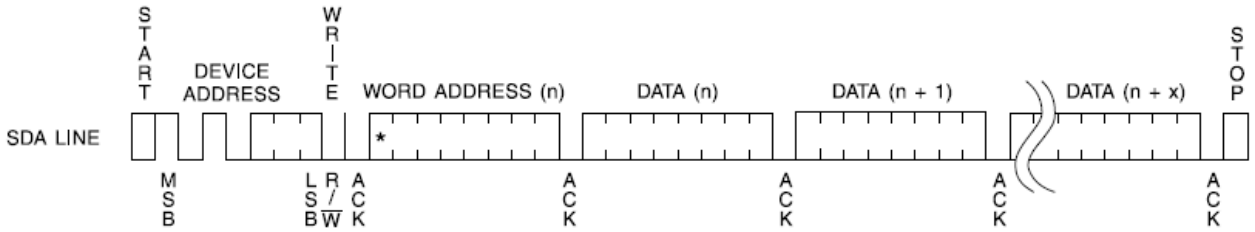


Рис. 5.2.3. Запись данных в микросхему памяти AT24C16

Восьмой бит в байте адреса предназначен для указания последующей операции записи или чтения. Низкий уровень ─запись, высокий уровень ─чтение. Далее в нашем примере следует запись данных и, следовательно, устанавливается низкий уровень. В расширенной версии протокола I^2C адрес может быть 10-разрядным и, следовательно, состоять из двух байтов. За байтом адреса следует адрес ячейки памяти внутри блока. Далее передаются байты данных, предназначенных для хранения в ячейках памяти, адрес которых после записи каждого байта увеличивается на единицу. После приема каждого байта микросхема *AT24C16* должна ответить подтверждением *ACK*.

5.2.2. Последовательный периферийный интерфейс SPI

Последовательный периферийный интерфейс *SPI* (*Serial Peripheral Interface*) обеспечивает высокоскоростной синхронный обмен данными между микроконтроллерами и периферийными устройствами или между несколькими микроконтроллерами (до 1,5 Мбит/с). Один из микроконтроллеров или устройств должен быть ведущим (*Master*) другие ведомыми (*Slave*). Для обеспечения обмена данными между устройствами используются четыре линии (рис.5.2.4).

Тактовые сигналы по линии *SCK* всегда генерирует ведущий

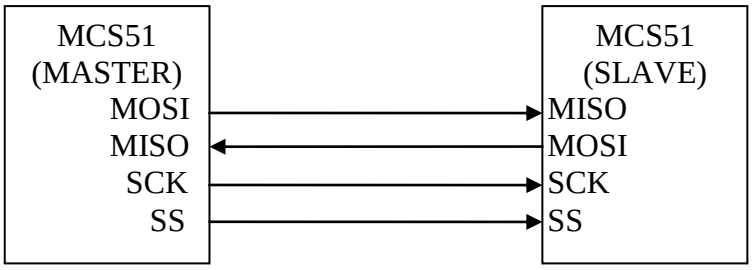


Рис. 5.2.4. Соединение устройств по интерфейсу *SPI*

микроконтроллер или ведущее устройство *MOSI* (*Master clock output, Slave clock input*). Данные от ведущего передаются по линии *MOSI* (*Master data output, Slave input*), прием данных осуществляется по линии *MISO* (*Master data input, Slave output*). Линия *SS* используется для ведомого устройства.

Если ведомым устройством является микроконтроллер, то его вывод, к которому подключена линия *SS*, должен быть настроен как вход.

На рис.5.2.5 показана временная диаграмма обмена данными по интерфейсу *SPI* для случая передачи байта (восемь циклов сигнала *SCK*) старшими битами *MSB* (*Most Significant Bit*) вперед. В последнем восьмом цикле передается младший бит *LSB* (*Least Significant Bit*). Полярность *CPO*L сигналов *SCK* определяется параметрами конкретного ведомого устройства.

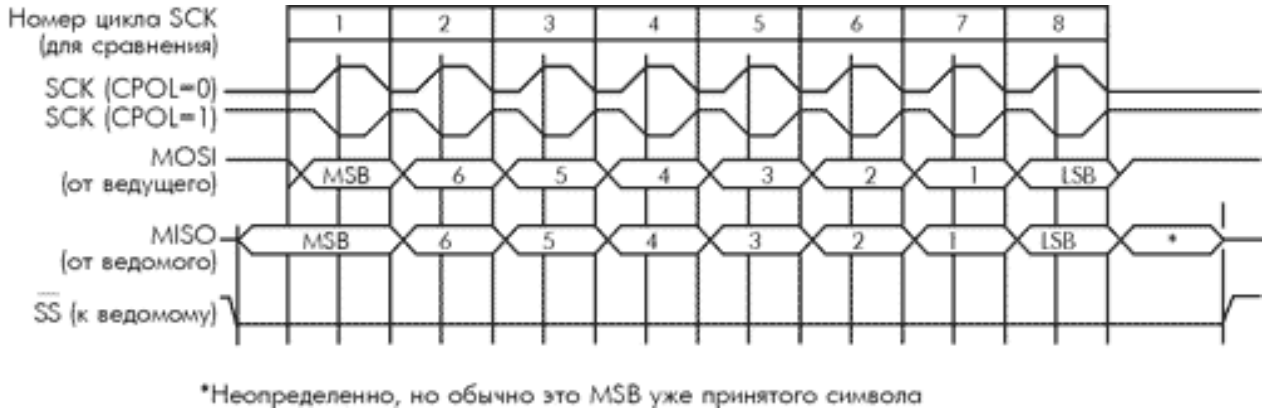


Рис. 5.2.5. Временная диаграмма обмена данными по интерфейсу *SPI*

В настоящее время во многих микроконтроллерах формирование тактовых сигналов *SCK*, преобразование байта данных в последовательный код *MOSI* при передачи и преобразование принятого последовательного кода *MISO* в параллельный осуществляется аппаратным способом. Для обеспечения указанных преобразований в микроконтроллерах имеются регистры режимов и управления и регистры-буферы данных.

Например, фирма *Atmel* выпускает микроконтроллеры, в которых для установки режимов и управления используется регистр *SPCR* (*SPI Control Register*) и регистр данных *SPDR* (*SPI Data Register*).

Лекция 10

5.3. Организация последовательного интерфейса CAN

Внедрение микропроцессоров и микроконтроллеров в самые различные распределенные системы управления потребовало построение сетей, объединяющих многообразные электронные управляющие устройства. С этой целью Робертом Бошем (*Robert Bosch*) в 80-х годах для автомобильной промышленности была разработана сеть и соответствующий ей протокол *CAN* (*Controller Area Network*). Сегодня большинство европейских автомобильных гигантов (например, *Audi, BMW, Renault, Saab, Volvo, Volkswagen*) используют сеть *CAN* в системах управления двигателем, безопасности и обеспечения комфорта. В настоящее время протокол *CAN* широко применяется в промышленности, энергетике и на транспорте. С его помощью могут быть построены как высокоскоростные сети, так и системы с дешевыми мультиплексными каналами.

Обмен информацией в сети *CAN* осуществляется по последовательной шине дифференциальными сигналами по витой паре проводов (рис.5.3.1). При скорости передачи 1 Мбит/с длина шины может достигать 30 м. При меньших скоростях ее можно увеличить до километра. Если требуется большая длина, то ставятся мосты или повторители. Теоретически число подсоединяемых к шине устройств не ограничено, практически — до 64-х. В узлах для поддержания протокола *CAN* могут быть использованы как микроконтроллеры с встроенным интерфейсом, так и внешние по отношению к узлу контроллеры *CAN*.

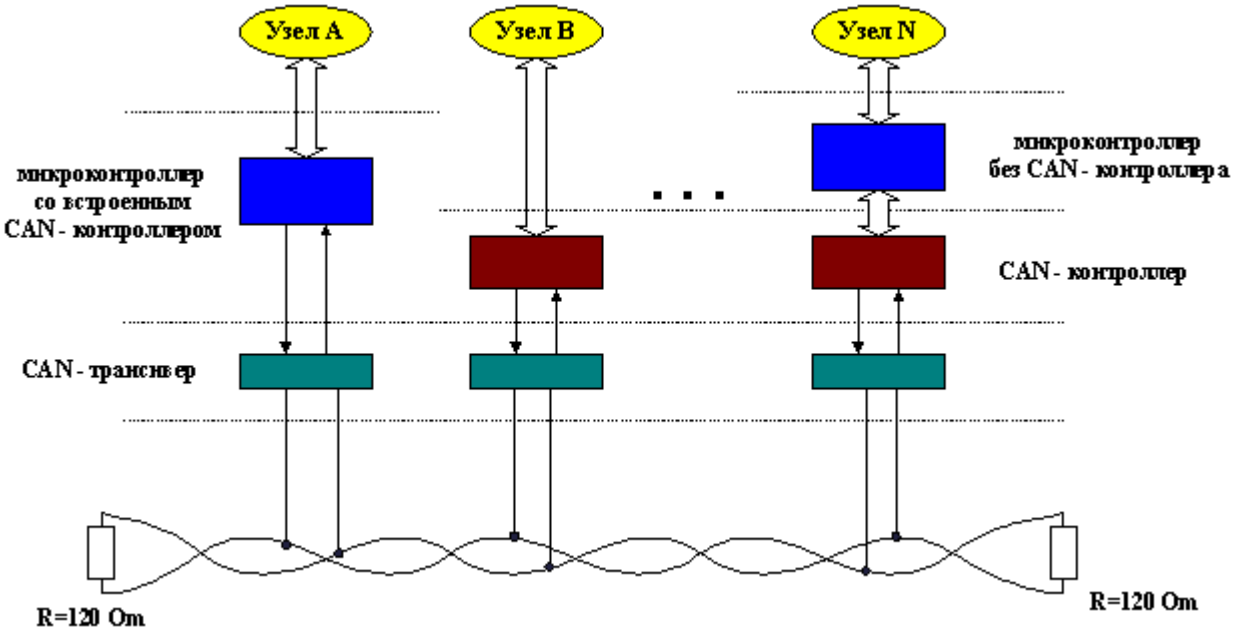


Рис. 5.3.1 Соединение устройств по интерфейсу *CAN*

Если базироваться на семиуровневой модели сети *OSI*, то *CAN* описывает передачу данных между узлами на двух нижних уровнях — физическом и канальном. Битовый поток кодируется по методу *NRZI* (без возвращения к нулю), что позволяет работать на меньших частотах, чем, например, при других видах кодирования. Для повышения устойчивости синхронизации используется вставка нулевого бита в случае следования подряд шести единиц (*Bit Stuffing*).

Для протокола *CAN* существуют две версии: версия *A* задает 11-битную идентификацию сообщений (т. е. в системе может быть 2048 сообщений), версия *B* — 29-битную (536 млн. сообщений). Отметим, что версия B, часто именуемая *FullCAN*, все больше вытесняет версию A, которую называют также *BasicCAN*. стандартизована *ISO* (*ISO* 11898) и *SAE* (*Society of Automotive Engineers*).

На рис. 5.3.2 приведен формат кадра для версии *B* (*CAN 2.0B*). Начало

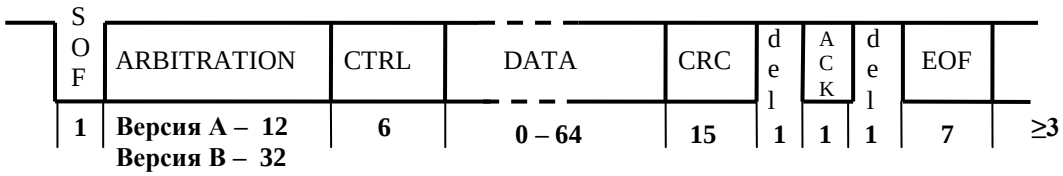


Рис. 5.3.2. Формат кадра протокола *CAN*

кадра *SOF* отмечает начало кадра данных или кадра удаленного запроса данных. Это поле состоит из одного разряда, равного нолю.

Поле арбитража *ARBITRATION* (*Arbitration field*) для версии A состоит из 11 разрядного идентификатора *ID* и *RTR*-бита. Для версии B идентификатор *ID*

состоит из 29 разрядов. Коме того для версии *B* в поле арбитража добавлены два бита *SRR* и *IDE*. Для различения версий служит бит *IDE*. Бит *SRR* указывает на то, что за 11разрядами идентификатора будут следовать 18 разрядов его расширения. Назначение бита *RTR* для обеих версий одинаково. Он определяет тип кадра (кадр данных или кадр удалённого запроса).

Поле управления *CTRL* (*Control field*) состоит из двух резервных битов и 4-х разрядов *DLC* (*Data length code*), которые определяют число байтов в поле *DATA*. В версии *A* один из резервных битов используется как бит *IDE*.

Пятнадцать разрядов поля CRC формируются циклическим кодом и служат для обнаружения ошибок, возникающих при передачи кадра. Вычисление контрольных разрядов поля *CRC* осуществляется с по*мощью полинома*

$$X^{15}+X^{14}+X^{10}+X^{8}+X^{7}+X^{4}+X^{3}+1.$$

За полем *CRC* следует однобитный разделитель *del*. Узлы, вычислившие последовательность *CRC*, совпадающую с переданной (ошибки отсутствуют) сообщают об этом передатчику путем замены в поле подтверждения *ACK* бита с единицы на ноль. Общая вероятность необнаруженной ошибки $P_{ош}$ равна $4.7x10^{-11}$.

За полем *ACK* следует разделитель del.

Поле конца кадра *EOF* (*End of frame*) состоит из семи единиц. Между кадрами имеется промежуток *IFS* (*Inter Frame Spacing*), состоящий из не менее трех битов.

Протокол *CAN* использует оригинальную систему адресации сообщений. Каждое сообщение снабжается идентификатором, который определяет назначение передаваемых данных, но не адрес приемника. Любой приемник может реагировать как на один идентификатор, так и на несколько. На один идентификатор могут реагировать несколько приемников.

Лекция 11

5.4. Последовательный однопроводный интерфейс One Wire

Фирмой *Dallas Semiconductor* разработан последовательный интерфейс, позволяющий обеспечивать обмен данными по одной линии (вторая линия – общий провод). Этот интерфейс известен под названием *MicroLAN*. В настоящее время устройства обменивающиеся данными по однопроводной линии (шине) получили название *1-Wire* интерфейс. К однопроводной шине могут быть подключены несколько устройств, но только одно из них является ведущим, а все остальные ведомыми. Обмен данными осуществляется в полудуплексном режиме. Допускается питание устройств по линии данных. В этом случае ведомые устройства к ведущему подключаются по двум проводам: линия данных и земля.

Цикл обмена данными начинается с формирования ведущим устройством

состояния сброса (рис. 5.4.1). Оно заключается в передачи сигнала низкого

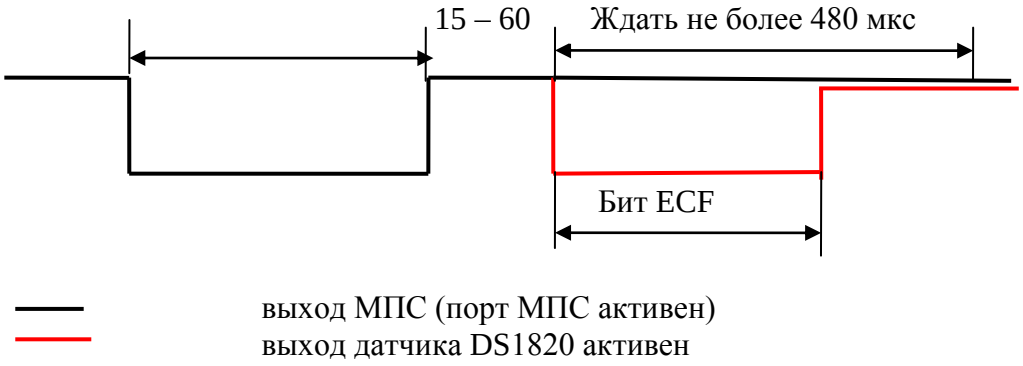


Рис. 5.4.1. Состояние сброса и подтверждения

уровня длительностью не мене 480мкс и ожидания подтверждения от ведомого устройства .

В ответ каждое устройство, подключенное к линии данных, производит сброс своих внутренних цепей и через 15-60 мкс выдает низкий уровень сигнала подтверждения (*presence pulse*) в течение 60-240 мкс. Обнаружив этот импульс, ведущий узел передает 8-битный код из списка команд выбираемого устройства. Все устройства, адрес которых не совпал с переданным, логически отключаются от шины. Выбранному устройству передается код операции обмена данными. По окончании операции ведущий узел генерирует новый импульс сброса и начинается новый цикл обмена.

На рис. 5.4.2 приведена временная диаграмма записи/чтения одного бита в процессе обмена данными между ведущим и ведомым устройствами.

Передача данных в ведомое устройство начинается с установки ведущим устройством на линии низкого уровня длительностью не менее 15мкс, затем устанавливается уровень передаваемого бита. Приём данных из ведомого устройства также начинается с установки ведущим устройством в течении 15мкс на линии низкого уровня, а затем ведущее устройство устанавливает на линии высокий уровень и не более чем через 15мкс ведомое устройство должно выдать уровень нуля или единицы.

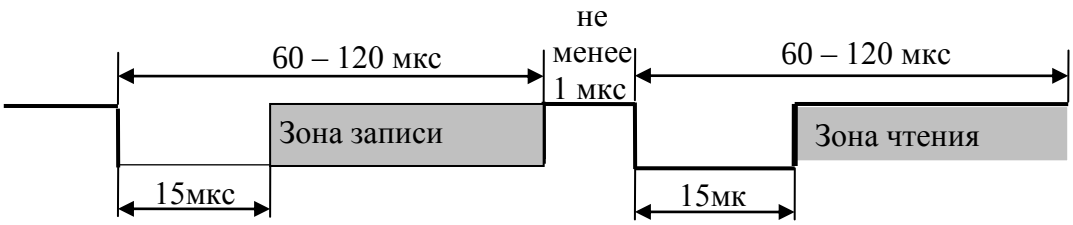


Рис. 5.4.2. Запись/чтение бита в термодатчик *DS1820*

Рассмотрим обмен данными на примере применения датчика *DS1820* для измерения температуры.

Датчик температуры формирует два байта, в которых записан 9-разрядный двоичный код из диапазона –55°С - +125°*С* с дискретностью 0,5°*С*. отрицательные температуры кодируются дополнительным кодом.

Протокол обмена данными между МПС и датчиком *DS1820* может быть задан последовательностью команд, указанной в табл.5.4.1.

Таблица 5.4.1

Последовательность команд для обмена данными между МПС и датчиком

Состояние МПС	Команда/данные	Коментарий
Передача	Состояние сброса	
Прием	Состояние подтверждения	
Передача	Команда *ССН*	Обращение к памяти данных
Передача	Команда 44Н	Запуск датчика
Прием данных	Прием одногобайта	Проверка готовности, прием бит за битом пока все 8 не будут равны 1
Передача	Состояние сброса	
Прием	Состояние подтверждения	
Передача	Команда ССН	Обращение к памяти данных
Передача	Команда *ВЕН*	Чтение памяти данных
Прием	Прием 2- 9 байтов данных	2 байта – данные измерений. 9 – байтов включают содержимое регистров TH и TL, резерв и CRC
Передача	Состояние сброса	
Прием	Состояние подтверждения	

5.5. Универсальная последовательная шина

Универсальная последовательная шина *USB* была разработана сравнительно недавно — в 1996 г. Она обеспечила разработчикам относительно дешевый, высокоскоростной (до 12 и до 400 Мбит/с для стандарта 2.0) и удобный в использовании интерфейс. Удобство применения шины *USB* определяется следующими характеристиками:

1. Простая реализация расширения периферии персонального компьютера.
2. Высокая скорость от 1,5 Мбит/с до 480 Мбит/с (*USB* 2.0).
3. Простота кабельных подключений и дешевизна реализации.
4. Поддержка одновременной работы со многими устройствами (127 на шине).
5. Надёжность. (Обнаружение ошибок, идентификация неисправных и неправильно подключены устройств).
6. Возможность простого обновления.

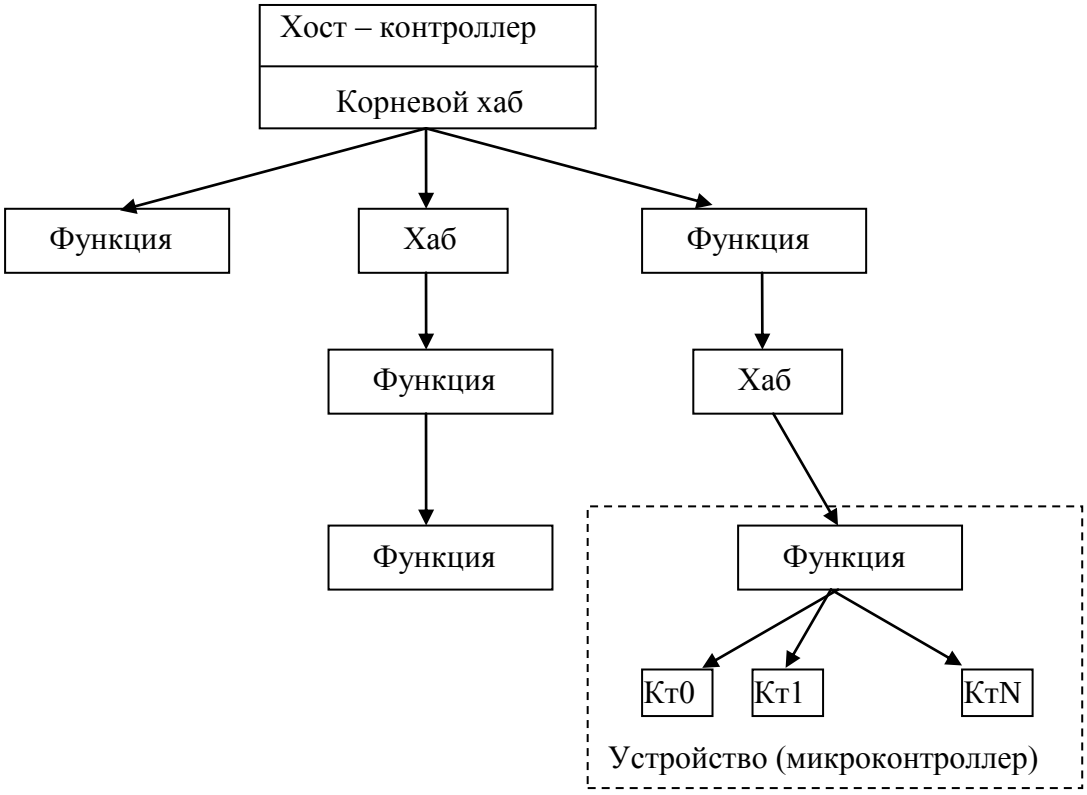


Рис. 5.5.1. Состав шины *USB*

На рис. 5.5.1 показан состав шины *USB* в случае соединения персонального компьютера (ПК) с периферийными устройствами.

Периферийные устройства **(**функции**)** подключаются к хост-контроллеру, причем соединение может быть по топологии многоярусной

звезды, но вершиной звезды должен быть корневой хаб**.** На шине *USB* допускается только один хост.

Точкой подключения устройства является порт. Для обеспечения подключения нескольких устройств к шине служит хаб (концентратор). Хаб распознает подключение и отключение устройств к портам и может управлять подачей питания на устройства.

Функция – периферийное устройство (*device*). В состав функции . входят конечные точки, каждая из которых должна быть настроена на приём или передачу данных. Любое *USB* устройство имеет конечную точку с нулевым номером Кт0 (*Endpoint Zero*). Нулевая конечная точка предназначена для инициализации и конфигурирования устройства и доступна хосту сразу после подключения к шине *USB.*

Шина *USB* может быть построена и без ПК. В этом случае устройства получили название *OTG* (устройства – On-The-Go). Например, подключение фотоаппарата к принтеру.

Любое устройство должно поддерживать:

1. отзыв на присвоенный ему уникальный адрес, причём при каждом новом подключении адрес может быть присвоен другой.
2. конфигурирование.
3. настройку на тип передачи и приёма данных.
4. управление энергопотреблением.
5. приостановку (снижение тока потребления).
6. удалённое прбуждениеэ

Максимальная скорость передачи данных определена номером версии:

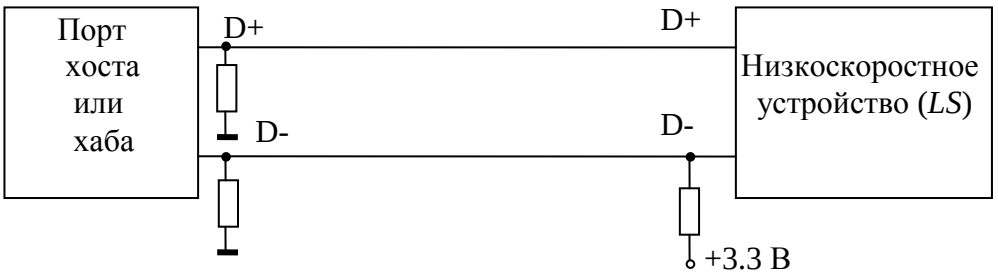
– *High Speed* (*HS*) – 480 *Mbits/s* (*USB* 2.0);

– *Full Speed* (*FS*) – 12 *Mbits/s* (*USB* 1.1);

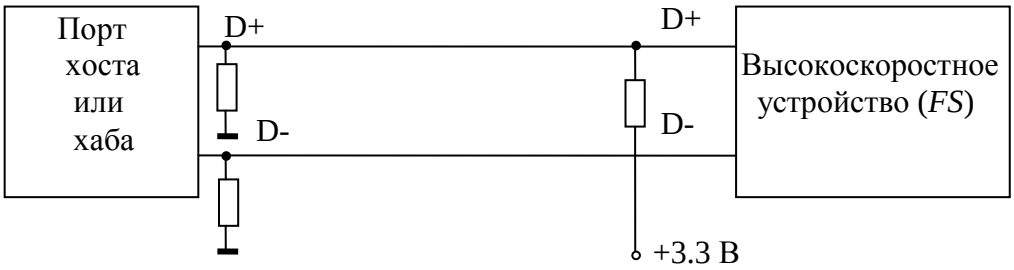
– *Low Speed* (*LS*) – 1,5 *Mbits/s* (*USB* 1.0).

Для передачи данных по шине *USB* используется дифференциальный метод передачи сигналов (рис. 5.5.2). Последовательность двоичных разрядов, передаваемая по линии дифференциальными сигналами, кодируется кодом *NRZI*. Хост или хаб определяет тип устройства по тому, как подключен к линиям *D*+ и *D*- подтягивающий резистор. Если через подтягивающий резистор напряжение 3.3 В подключено к *D*+, то определяется высокоскоростное устройство, если напряжение 3.3 В подключено к *D*-, то – низкоскоростное устройство.

Передача данных осуществляется кадрами и асинхронно. Кадр *USB* или фрейм (рис.5.5.3) передаётся за фиксированный интервал времени (1мс). Логическая связь между хост-контроллером и конечной точкой образует логический канал. Его пропускная способность определяется скоростью. Каждый кадр начинается с посылки маркера *SOF* (*Start Of Frame*), который является синхронизирующим сигналом для всех устройств, включая хабы. В конце каждого кадра выделяется интервал времени *EOF* (*End Of Frame*), на время которого хабы запрещают передачу по направлению к контроллеру. В режиме *HS* маркеры *SOF* передаются в начале каждого микрокадра (период 125±0,0625 мкс).



а) подключение низкоскоростного устройства



б) подключение высокоскоростного устройства

Рис.5.5.2. Подключение к порту хаба:
а) низкоскоростного устройства, б) высокоскоростного устройства

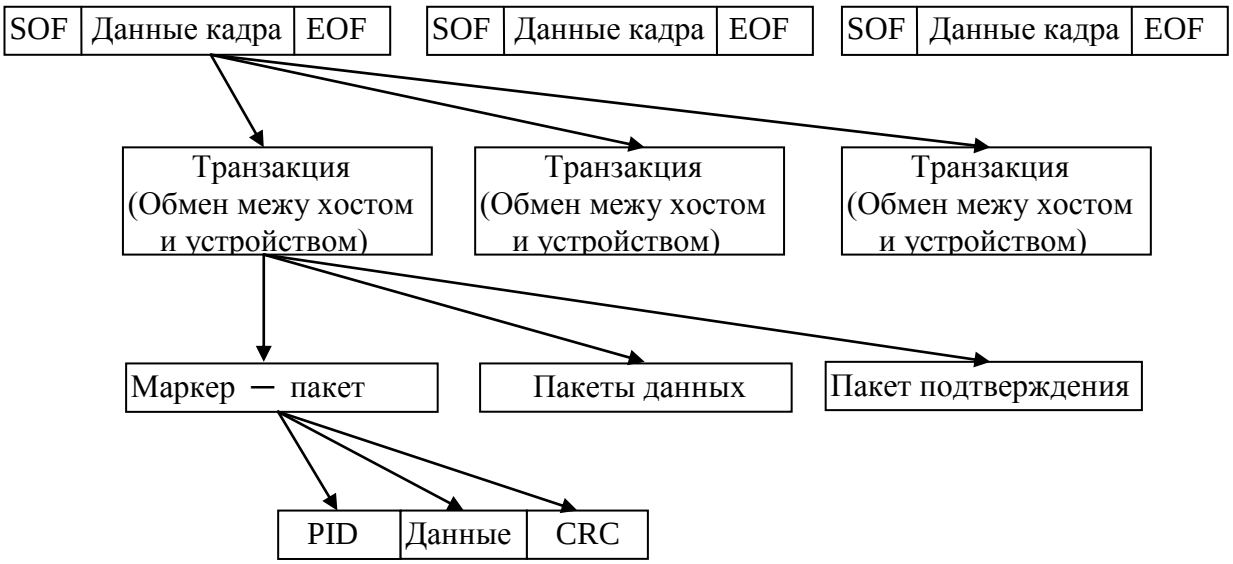


Рис. 5.5.3. Структура кадра шины *USB*

Хост планирует загрузку кадров так, чтобы в них всегда находилось место для обмена (транзакций) информацией, предназначенной для управления и прерываний. Свободное время кадров может заполняться передачами массивов (*bulk transfers*). В каждом (микро)кадре может быть

выполнено несколько транзакций, их допустимое число зависит от длины поля данных в каждой из них. Транзакция состоит из
пакетов, причем сначала передается маркер – пакет, в котором указывается идентификатор *PID* (назначение) пакета и контроль ошибок *CRC* с использованием циклического кода.

Протокол *USB*, допускает четыре базовых типа передачи данных:

─ управляющие посылки (*control transfers*) используются для конфигурирования устройств во время их подключения и для управления устройствами в процессе работы. Протокол обеспечивает гарантированную доставку данных.Длина 64 байта для *FS* и 8 байтов для *LS.* Гарантированно выделение 10% внутри кадра;

─ передачи массивов данных (*bulk data transfers*) - это передачи без каких-либо обязательств по задержке доставки и скорости передачи. Передачи массивов могут занимать весь кадр, свободный от передач других типов. Приоритет этих передач самый низкий, они могут приостанавливаться при большой загрузке шины. Доставка гарантированная - при случайной ошибке выполняется повтор. Передачи массивов уместны для обмена данными с принтерами, сканерами, устройствами хранения и т. п. Длина поля данных 8,16,32,64 байтов.

─ прерывания (*interrupt*) ─ короткие передачи, которые имеют случайный характер и должны обслуживаться не медленнее, чем того требует устройство. Прерывания используются, например, при вводе символов с клавиатуры или для передачи сообщения о перемещении мыши.

─ изохронные передачи (*isochronous transfers*) - непрерывные передачи в реальном времени, занимающие предварительно согласованную часть кадра с гарантированным временем задержки доставки. В случае обнаружения ошибки изохронные данные не повторяются. Пакеты с ошибками игнорируются.

Все обмены (транзакции) с устройствами *USB* состоят из двух-трех пакетов (рис 5.5.4). Каждая транзакция начинается по инициативе хост –

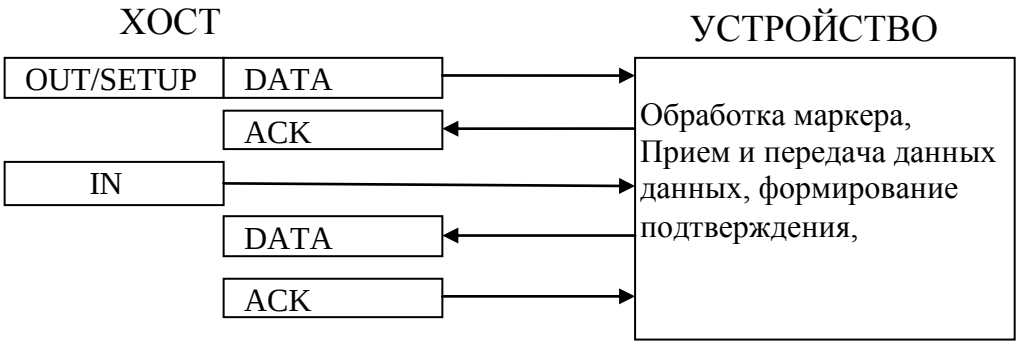


Рис. 5.5.4. Обмен транзакциями между хостом и устройством

контроллера, который посылает пакет-маркер (*token packet*). Он описывает тип и направление передачи, адрес устройства *USB* и номер конечной точки. В каждой транзакции возможен обмен только между адресуемым

устройством (его конечной точкой) и хостом. Адресуемое маркером устройство распознает свой адрес и готовится к обмену.

Источник данных (определенный маркером) передает пакет данных (или уведомление об отсутствии данных, предназначенных для передачи). После успешного приема пакета приемник данных посылает пакет пдтверждения (*handshake packet*).

Каждый пакет начинается с поля синхронизации *SYNC*, за которым следует четыре бита идентификатора *PID* и его побитной инверсии *CHECK*. Идентификаторы *PID* делятся на четыре группы:

1 ─ маркеры пакетов (*OUT, IN, SOF, SETUP*);
2 ─ идентификаторы данных(*DATA0, DATA1, DATA2*);
3 ─ идентификаторы подтверждений (*ACK, NAK, STALL*);
4 ─ идентификаторы специальных пакетов (*ERR* и др.).

В поле данных первой группы *OUT*, *IN* и *SETUP* указывается адрес функции *FUNC*(до 127 адресов), адрес конечной точки КТ (*EndP*) (до 16 конечных точек) и контрольная последовательность *CRC*, вычисленная циклическим кодом с полиномом пятой степени. Для маркера *SOF* в поле данных указывается номер кадра и контрольная последовательность.

В поле данных пакетов *DATA* может содержаться от 0до 1023байтов данных и контрольная последовательность *CRC*, вычисленная циклическим кодом с полиномом 16 степени.

Поле данных пакета подтверждения *ACK* пустое.

Работа *USB* устройства начинается с его конфигурирования. Хост контроллер посылает стандартные запросы (дескрипторы хоста), в которых должны быть переданы тип запроса, код запроса, параметр запроса, индекс и число байт для передачи. На рис. 5.5.5 показана последовательность запросов от хоста. В список дескрипторов устройства входят дескрипторы описания *USB* устройства, параметры конфигурации и интерфейса, параметры конечной точки и т.д. В данном примере хост запрашивает дескриптор устройства (код 01).

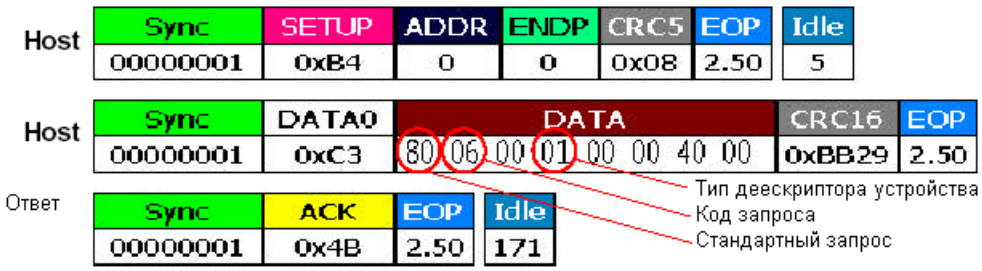


Рис. 5.5.5. Стандартный запрос дескриптора от устройства

После получения подтверждения в ответ на стандартный запрос хоста устройство отвечает дескриптором устройства (рис.5.5.6). В ответе содержится число байтов дескриптора (12*H*=18 байтов), тип дескриптора 01, версия *USB* 1.1(код 1001*H*), в поле данных для нулевой точки далее обмен будет осуществляться не более чем по 8 батов.

Рис. 5.5.6. Ответ на стандартный запрос

В список стандартных дескрипторов устройства должны также входить дескриптор конфигурации (код 02), дескриптор интерфейса (код 03) и дескриптор конечной точки (код 04). После конфигурирования может быть начат обмен данными.

В настоящее время выпускается большой набор микросхем, позволяющих заменить некоторые выше описанные интерфейсы более прогрессивным последовательным интерфейсом *USB*. Микросхемы делятся на следующие группы:

– преобразователи последовательного интерфейса *COM* в *USB* (Фирмы *FTDI*, *Maxim*, *Philips*);
– преобразование параллельного в *USB*;
– микроконтроллеры с *USB* интерфейсом;
– микросхемы хабов
– микросхемы *OTG*.

В заключении рассмотрим структурную схему устройства *USB*, встроенного в микроконтроллер *AT89C5131* фирмы *ATMEL* (рис. 5.5.7). Устройство может работать как с версией *USB* 1.1, так и с версией *USB* 2.0 *FS* и включает 7 конечных точек. Управляющая коечная точка КТ0 имеет 32-х разрядный буфер *FIFO*. Остальные конечные точки обеспечивают как прием так и передачу данных в режимах передачи массивов данных (*bulk data transfers*), прерывания (*interrupt*) и изохронной передачи (*isochronous transfers*). Конечные точки КТ 1, 2, 3 имеют буфер *FIFO* 32 байта, КТ4, 5 имеют два буфера *FIFO* по 64 байта с возможностью их попеременной загрузки, КТ6 тоже имеют два буфера *FIFO*, но емкостью 512 байтов.

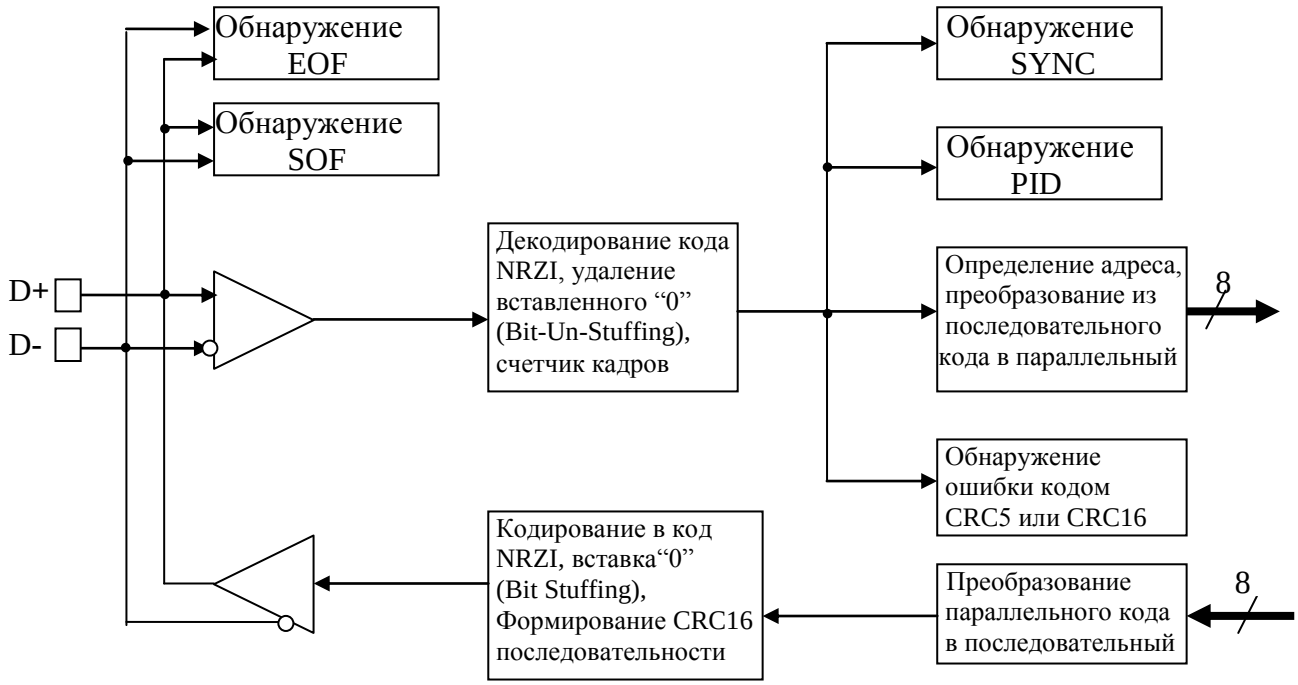


Рис. 5.5.7. Структурная схема встроенного контроллера USB

Для передачи данных по шине USB используется дифференциальный способ передачи сигналов по двухпроводным проводам D+ и D-. Для обеспечения низкой скорости передачи данных линия D- подключается (подтягивается) через резистор 1,5 кОм к напряжению 3,3 В, а для высокой скорости передачи данных напряжение 3,3 В через резистор 1,5 кОм подключается к линии D+.

Тактовые сигналы для всех узлов устройства *USB* 12 Мгц формируются делением опорной частоты 48 Мгц, которая вырабатывается микроконтроллером. В регистр *DPLL* заносится коэффициент деления, который определяется в зависимости от тактовой частоты внутреннего генератора микроконтроллера. Тактовые сигналы 12 Мгц синхронизируются с принятыми по шине *USB* дифференциальными сигналами.

Все операции по преобразованию кодов при передачи и приеме кадров осуществляется аппаратно. В микроконтроллере имеется набор регистров специальных функции, с помощью которых устанавливаются режимы работы внутреннего устройства *USB* и в которых сохраняются флаги, сформированные аппаратно в узлах структурной схемы. Анализ флагов может быть осуществлен программно или разрешено соответствующее прерывание.

Лекция 12

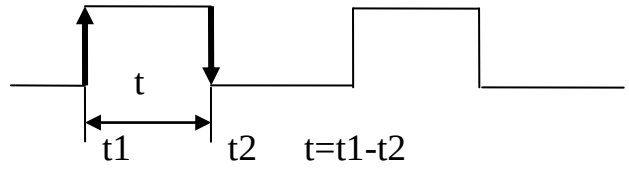
6. Массив программируемых счетчиков

6.1. Назначение и состав массива программируемых счетчиков

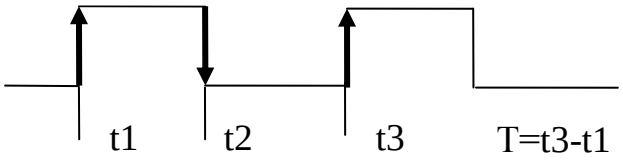
В состав современных микроконтроллеров входят несколько таймеров, позволяющих с высокой точностью формировать интервалы времени и обрабатывать сигналы от различных внешних устройств. Однако, во многих случаях для решения прикладных задач требуется либо усложнять программу либо добавлять внешние микросхемы. Для расширения возможностей микроконтроллера с целью обработки широкого набора сигналов внешнего оборудования и генерирования временных последовательностей для управления внешним оборудованием в настоящее время в микроконтроллеры встраиваются массив программируемых счётчиков *PCA* (*Programmable Counter Array*).

Рассмотрим (рис.6..1) несколько часто встречающихся задач обработки сигналов. На рис. 6..1 приведены примеры измерения длительности импульса, измерения периода последовательности импульсов, измерения скважности и измерение фазового сдвига.

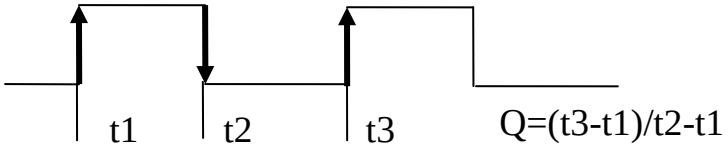
Из рис.6.1 видно, что в перечисленных случаях необходимо фиксировать времена формирования начала и конца сигналов, поступающих от внешних устройств. Переход напряжений от низкого уровня к высокому и наоборот от высокого к низкому называется событием. Для обнаружения событий используются детекторы захвата событий. Если зафиксировать время захвата события, то могут быть получены все необходимые данные для вычисления длительности импульса, периодичности поступления сигналов,



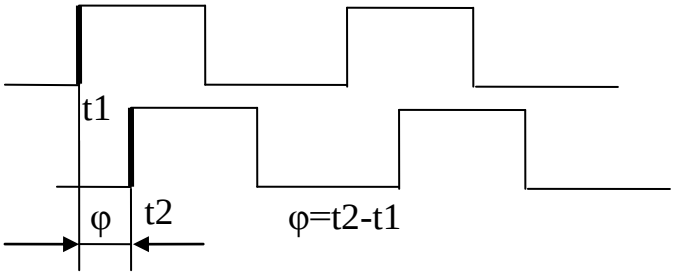
а) измерение длительности импульса t



б) измерение периода T



в) измерение скважности Q



г) измерение сдвига фазφ

Рис. 6.1. Измерение длительности импульса (а), периода (б), скважности (в), сдвига фаз (г)

скважности, сдвига фаз и т.п. времени. Перечисленные случаи могут быть реализованы с помощью встроенного в микроконтроллер массива программируемых счётчиков *PCA* (рис.6.2).

В состав *PCA* входят пять модулей, двунаправленные выводы которых могут быть использованы как входы внешних сигналов и как выходы сигналов управления внешними устройствами. В микроконтроллерах семейства *MSC51* выводы модулей подключены к порту *P1*. Каждый модуль может работать в одном из следующих режимов:

– захвата события по фронту или спаду внешнего сигнала;
– использование модуля как программируемого таймера;
– скоростного вывода;
– генератора импульсов с заданной скважностью (генератор ШИМ);

Модуль 4 может быть использован как сторожевой таймер в тех модификациях микроконтроллеров, в которых отсутствует его аппаратная реализация.

Для формирования отсчетов времени используется общий для всех модулей таймер/счетчик.

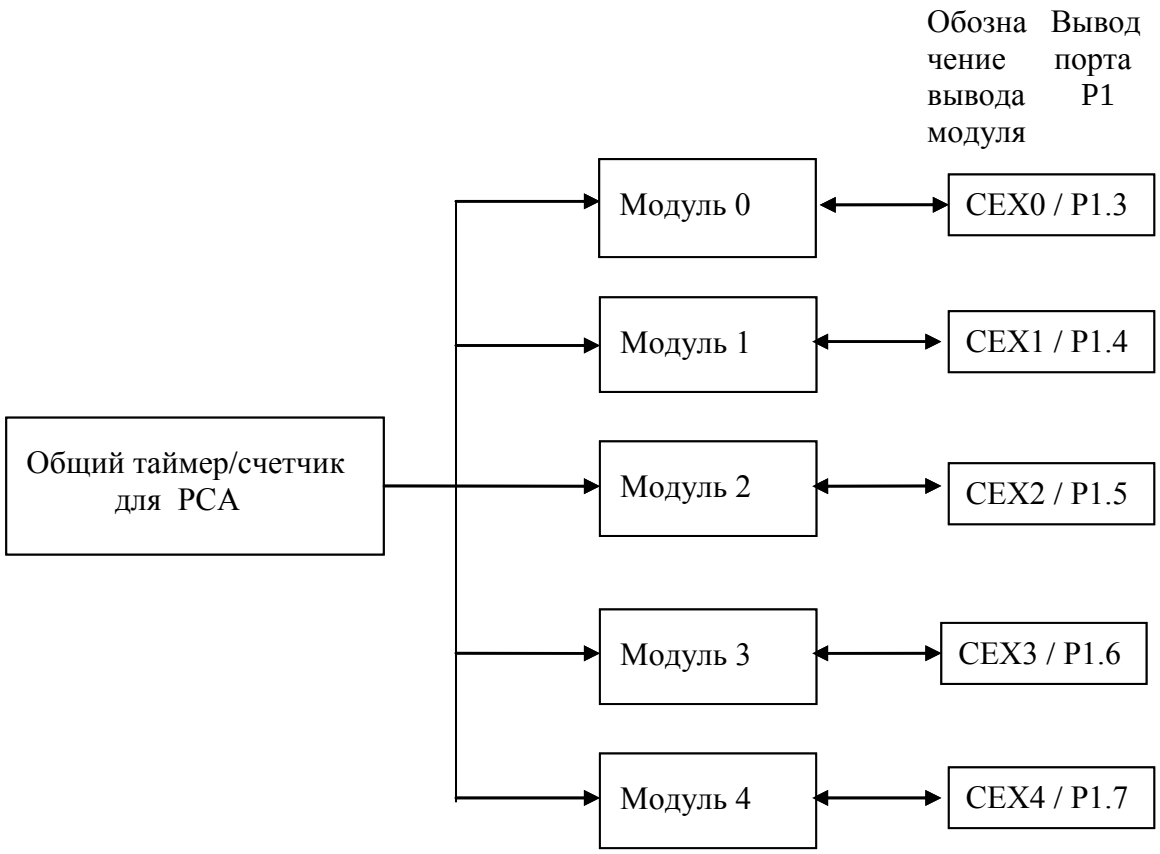


Рис. 6.2. Состав массива программируемых счетчиков (PCA)

6.2. Режимы работы общего таймера/счетчика

На рис.6.3 показана структурная схема общего *PCA* таймера/счетчика.

Шестнадцатиразрядный таймер/счетчик накапливающего типа состоит из двух 8-разрядных регистров *CH* и *CL*. Счет тактовых сигналов может осуществляться как от внутреннего генератора тактовой частоты, так и от внешнего источника тактовой частоты. Выбор источника тактовой частоты производится мультиплексором *MUX*. Если выбран внутренний источник тактовой частоты, то используется режим таймера, если – внешний, то режим счетчика.

В режиме таймера может быть использован один из трёх вариантов:

1. На вход таймера поступают сигналы тактовой частоты от внутреннего генератора, делённые на 12 ($F_T/12$);
2. На вход таймера поступают сигналы тактовой частоты от внутреннего генератора, делённые на 4 ($F_T/4$);
3. На вход таймера поступают сигналы при каждом переполнении таймера 0;
4. На вход счётчика поступают внешние сигналы с входа *P1.2*. Максимальная

частота в этом случае не должна превышать значения $F_T/8$.

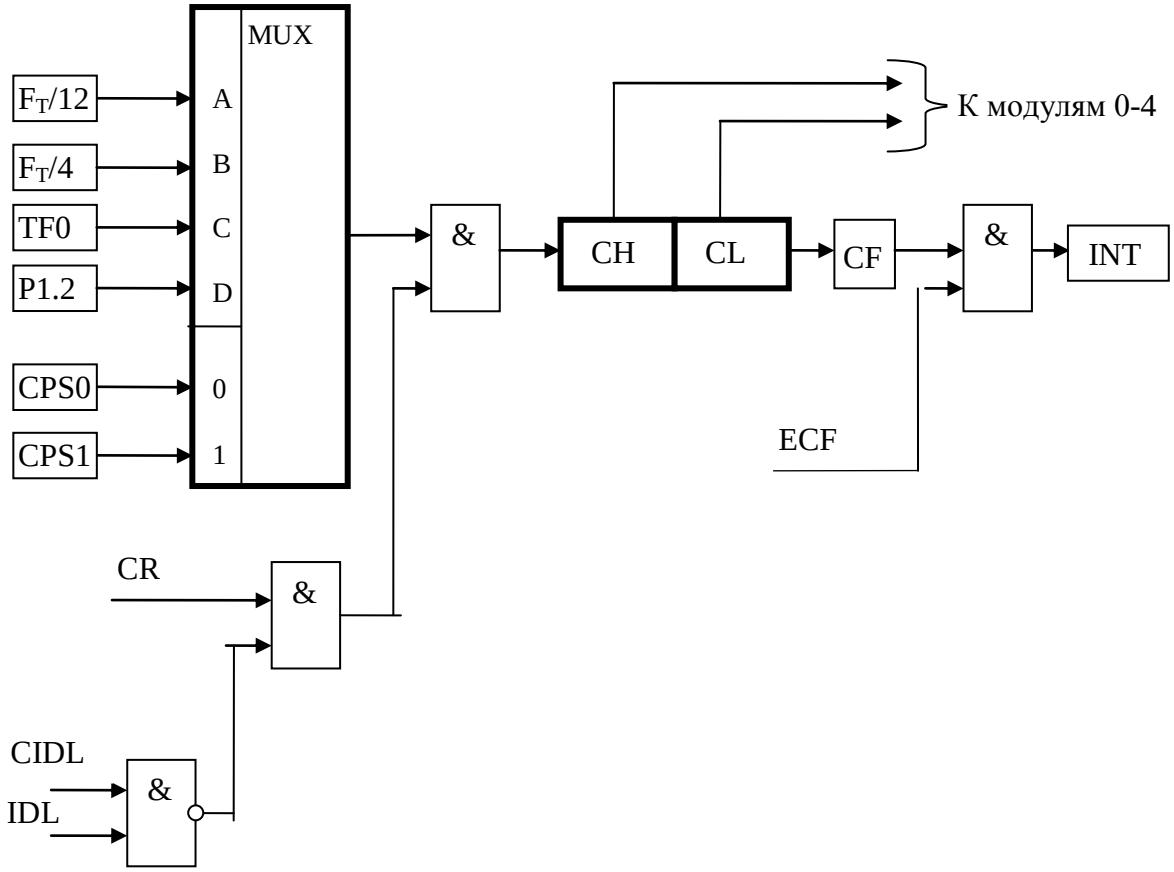


Рис. 6.3. Структурная схема общего таймера/счетчика PCA

Содержимое регистров *CH* и *CL* поступает на все модули. При переполнении таймера/счетчика формируется флаг *CF*, который может быть обработан либо программно, либо по прерыванию. В последнем случае должен быть установлен бит разрешения прерывания *ECF*.

Таймер/счетчик запускается битом *CR* при условии, что режим *IDLE* отключен. Если режим *IDLE* включен, то при значении бита *CIDL=0* запуск таймера/счетчика разрешен, а при значении бита *CIDL=1* запуск таймера/счетчика запрещен.

Для установки режимов работы таймера/счетчика служит регистр *CMOD*, а для управления работой таймера/счетчика служит регистр *CCON*.

Формат содержимого регистра *CMOD* показан в табл.6.1.

Адрес регистра – *D9H*.

Содержимое регистра после сброса – 0XXXX000B

Таблица 6.1

Регистр *CMOD*

Номер бита	Мнемоническое Обозначение бита	Описание

7	*CIDL*	В режиме *IDLE*, если *CIDL=0* запуск таймера/счетчика разрешен, иначе – запрещен
6	*WDTE*	Бит управления сторожевым таймером модуля 4
5 – 3	Зарезервировано	
2 – 1	*CPS1 – CPS0*	код управления мультиплексором *MUX*: 00 – $F_T/12$ 01 – $F_T/4$ 10 – переполнение таймера 0 11 – вход *P1.2*
0	*ECF*	Разрешение прерывания

Формат содержимого регистра *CMOD* показан в табл. 6.2.

Адрес регистра – *D8H*.

Содержимое регистра после сброса – 00XXXX00B

Таблица 6.2

Регистр *CCON*

Номер бита	Мнемоническое Обозначение бита	Описание
7	*CF*	Флаг переполнения таймера/счетчика. Сбрасывается программно
6	*CR*	Бит запуска таймера/счетчика
5	Зарезервировано	
4 – 0	*CCF4 – CCF0*	Флаги обнаружения захвата событий модулями

6.3. Режим захвата события

На рис. 6.4 показана структурная схема модуля *PCA*, работающего в режиме захвата событий.

Сигнал, фронт или спад, которого следует захватить, поступает на один из входов порта *P1.x* (*CEXn*), где *x* принимает значение от 3 до 7, а *n* номер модуля (рис.6.2).

Определение фронта или спада сигнала осуществляется в детекторах захвата *CAP* (*capture*). Обработка фронта или спада сигнала выбирается битами *CAPPn* (positive) в случае перехода от низкого уровня к высокому и *CAPNn* (negative) в случае перехода от высокого к низкому уровню.

Если фронт или спад обнаружен, то формируется флаг захвата события *CCFn*, который может быть обработан программно или по прерыванию. Для обработки захвата события должен быть установлен бит разрешения прерывания *ECCPn*. Флаг *CCFn* сбрасывается программно.

Для измерения временных параметров сигнала используется общий для

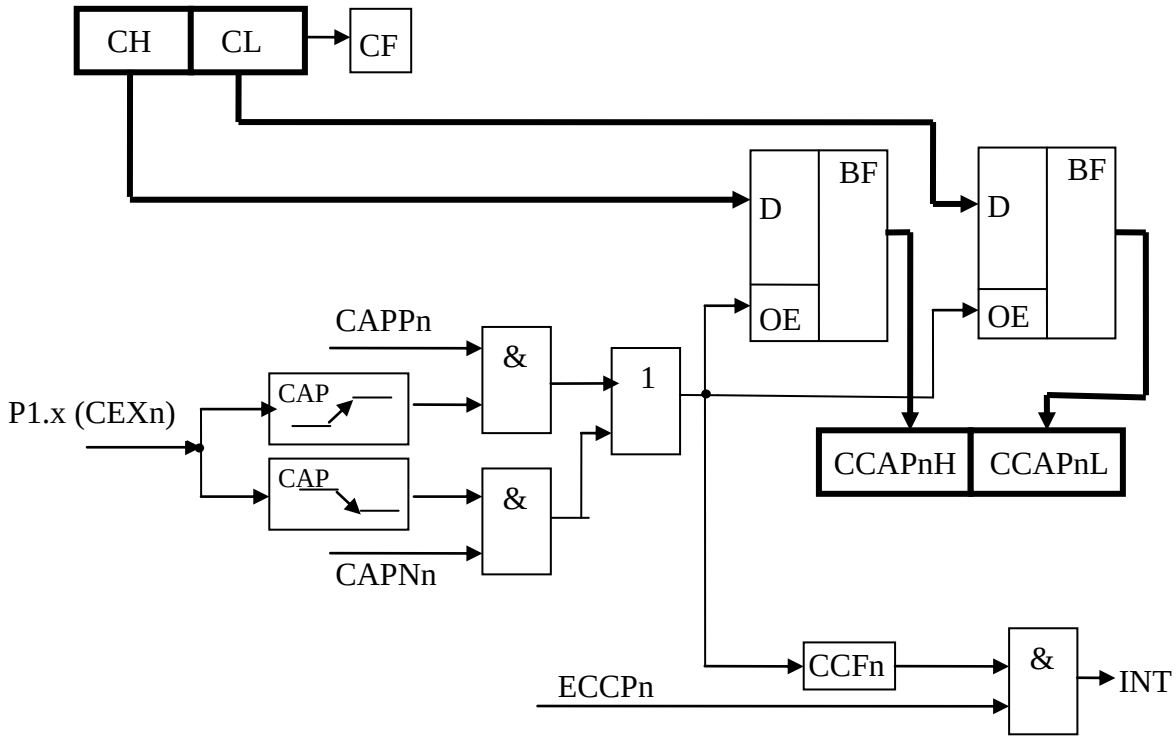


Рис. 6.4. Структурная схема модуля в режиме захвата

всех модулей таймер/счетчик. С этой целью текущее содержимое регистров *CH* и *CL* в момент захвата события защелкивается в регистрах CCAPnH и CCAPnL модуля. Например, в случае измерения длительности импульса, число защелкнутое в регистрах CCAPnH и CCAPnL будет соответствовать началу счета тактовых сигналов таймера/счетчика от момента захвата события (момент t1 на рис.6.6.1). До наступления захвата следующего события содержимое регистров *CCAPnH* и *CCAPLn* должно быть переписано в память микроконтроллера. После захвата следующего события в регистрах *CCAPHn* и CCAPnL. Для измерения временных параметров сигнала используется общий для *CCAPnL* будет защелкнуто число, соответствующее моменту t2 (рис.6.1) Разность двух чисел будет соответствовать длительности измеряемого импульса в тактах таймера/счетчика. Чтобы получить значение длительности импульса в единицах времени, необходимо умножить полученную разность в тактах на время периода тактовой частоты таймера/счетчика.

6.4. Режим 16-разрядного программируемого таймера, высокоскоростного вывода и сторожевого таймера

На рис.6.5 показана структурная схема модуля, работающего в режиме 16-разрядного программируемого таймера, высокоскоростного вывода и сторожевого таймера. Основными узлами модуля в этом случае являются регистры *CCAPHn*, *CCAPLn* и 16-разрядный узел сравнения *COMP*(компаратор).

В режиме 16-разрядного программируемого таймера формирование заданных интервалов времени осуществляется в следующем порядке. Предварительно в регистре управления модулями *CCAPMn* устанавливается бит *MATn* для разрешения формирования флага *CCFn*. В регистры *CCAPHn* и *CCAPLn* загружается 16-разрядное начальное значение и в регистре управления модулями *CCAPMn* устанавливается бит *ECOMn*, который разрешает сравнение содержимого основного таймера/счетчика (регистры *CH* и *CL*) с содержимым регистров модуля (*CCAPHn*, *CCAPLn*). Когда происходит совпадение содержимого основного таймера/счетчика и содержимого регистров *CCAPHn*, *CCAPLn* модуля, формируется флаг *CCFn*, который может быть обработан по прерыванию, если установлен бит *ECCFn*. При каждом сравнении флаг *CCFn* должен быть сброшен программно. В результате в интервале между каждым сравнением получим длительность времени в тактах.

Если не менять начальное значение регистров *CCAPHn*, *CCAPLn*, то получим режим перезагружаемого таймера. Интервалы между каждым сравнением могут быть заданы переменными. В этом случае после сравнения необходимо изменить содержимое регистров *CCAPHn*, *CCAPLn*. При каждой загрузке нового начального значения автоматически сбрасывается бит разрешения сравнения *ECOMn* и тем самым запрещается сравнение на время обновления содержимого регистров *CCAPHn* и *CCAPLn*, что позволяет избежать неверного сравнения.

В режиме работы скоростного вывода *HSO* (*High Speed Output*) в модуле дополнительно к установленным битам *MATn* и *ECOMn* должен быть еще установлен бит *TOGn*. Устанавливая или сбрасывая этот бит, пользователь в случае совпадения содержимого регистров таймера/счетчика и содержимого регистров модуля может разрешить формирование на выводе порта P1.x (*CEXn*) выходного сигнала высокого или низкого уровня.

Режим скоростного вывода является более точным по сравнению вывода сигнала на выход порта с помощью команд *SETB bit* или *CLR bit*, поскольку на их выполнение требуется дополнительное время.

Четвертый модуль может работать в режиме сторожевого таймера (*WatchDog Timer*). Работа в этом режиме возможна, если установлен бит *WDTE* в регистре режимов *CMOD*. При совпадении содержимого регистров таймера/счетчика *CH* и *CL* и содержимого регистров *CCAPHn* и *CCAPLn* модуля сформируется сигнал сброса *RESET* микроконтроллера. Предотвратить сброс можно тремя способами:

1. В программе пользователя, для которой требуется защита от сбоя, необходимо периодически изменять содержимое регистров *CH* и *CL* так чтобы никогда не было совпадений с содержимым регистров *CCAPHn* и *CCAPLn*. Этот способ лучше применять, когда общий таймер/счетчик не используется другими модулями.
2. В программе пользователя, для которой требуется защита от сбоя, необходимо периодически изменять содержимое регистров *CCAPHn* и *CCAPLn* так чтобы никогда не было совпадений с содержимым регистров *CH* и *CL*.
3. В программе пользователя, для которой требуется защита от сбоя, необходимо в нужный момент отключить режим сторожевого таймера сбросом бита *WDTE* прежде, чем произойдет совпадение содержимого регистров *CCAPHn* и *CCAPLn* и содержимого регистров *CH* и *CL* и затем

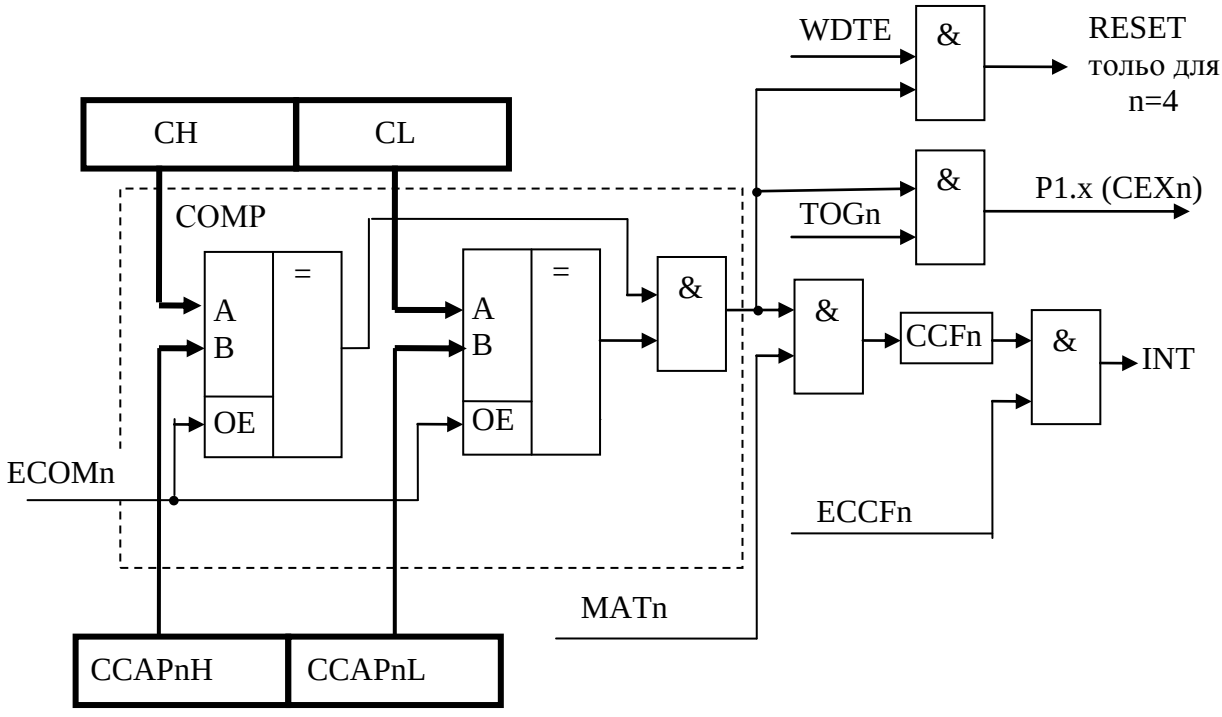


Рис. 6.5. Структурная схема работы модуля в режимах 16-разрядного таймера, высокоскоростного вывода и сторожевого таймера

снова установить бит *WDTE*.

Для управления работой модулей служит регистр *CCAPMn*.

Формат содержимого регистра *CCAPMn* показан в табл.6.3.

Адрес регистра *CCAPM0* для модуля 0 – *DAH*.
Адрес регистра *CCAPM1* для модуля 1 – *DBH*.
Адрес регистра *CCAPM2* для модуля 2 – *DCH*.
Адрес регистра *CCAPM3* для модуля 3 – *DDH*.
Адрес регистра *CCAPM4* для модуля 4 – *DEH*.
Содержимое регистра после сброса – X0000000B

Таблица 6.3

Регистр *CCAPMn*

Номер бита	Мнемоническое Обозначение бита	Описание
7	Зарезервировано	Флаг переполнения таймера/счетчика. Сбрасывается программно
6	*ECOMn*	Бит разрешения работы компаратора
5	*CAPPn*	Для сигнала на входе *P1.x/CEXn* разрешение работы детектора *CAPPn* (positive). 1 – разрешено, 0 – запрещено
4		Для сигнала на входе *P1.x/CEXn* разрешение работы детектора *CAPNn* (negative). 1 – разрешено, 0 – запрещено
3	*MATn*	Разрешение формирования флага *CCFn* 1 – разрешено, 0 – запрещено
2	*TOGn*	Разрешение режима *HSO* 1 – разрешено, 0 – запрещено
1	*PWMn*	Разрешение режима ШИМ 1 – разрешено, 0 – запрещено
0	*ECCFn*	Разрешение прерывания 1 – разрешено, 0 – запрещено

6.5. Режим генератора импульсов с заданной скважностью

Любой из пяти модулей может быть использован как генератор импульсов с заданной скважностью или как широтно импульсный модулятор (ШИМ). Английская аббревиатура – *PWM* (*Pulse Width Modulation*). На (рис.6.6) показана структурная схема модуля, работающего в режиме ШИМ.

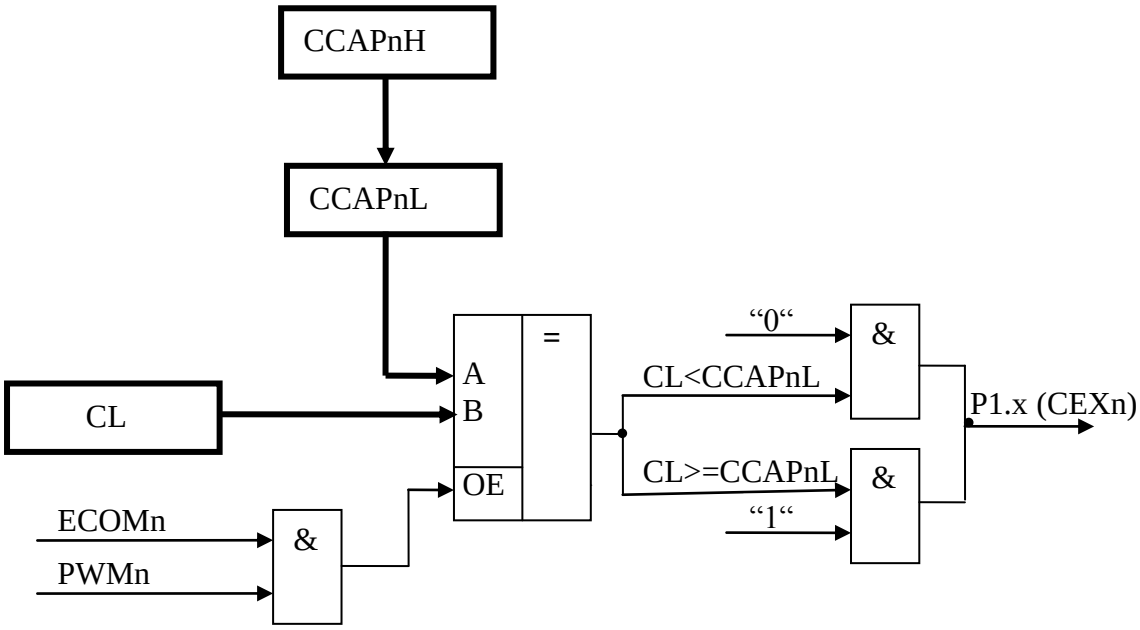


Рис. 6.6. Структурная схема работы модуля в режимте генератора ШИМ

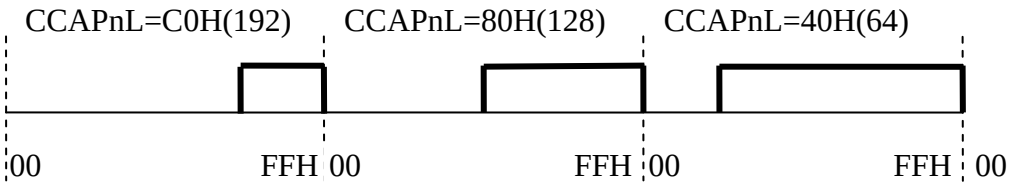


Рис. 6.7. Временная диаграмма формирования импульсов

В этом режиме компаратор модуля работает как 8-разрядное устройство и производится сравнение содержимого регистра *CL* общего таймера/счетчика с содержимым регистра *CCPALn* модуля. Запуск режима ШИМ осуществляется установкой битов *ECOMn* и *PWMn* в регистре *CCAPMn* (табл.6.3).Предварительно в регистр *CCPAHn* должно быть загружено начальное значение длительности импульса ШИМ. После каждого сравнения в регистр *CCPALn* пересылается содержимое регистра *CCPAHn*. Общий таймер счетчик задает частоту (период) генерируемых импульсов.

Формирование импульсов ШИМ иллюстрируется временной диаграммой на рис.6.7. Таймер/счетчик формирует период длительностью 256тактов (*00 – FFH*). Если *CL< CCPALn*, то на выходе модуля *P1.x (CEXn)* формируется сигнал низкого уровня. Если *CL≥ CCPALn*, то на выходе модуля *P1.x (CEXn)* формируется сигнал высокого уровня. Чтобы импульсы ШИМ имели переменную длительность, нужно новое содержимое регистра *CCPAHn* переписывать в регистр *CCPALn* в момент перехода содержимого *CL* из значения *FFH* в *00H*.

6.6. Обработка прерываний от источников PCA

На рис.6.8 показана структурная схема обработки прерываний от различных источников массива программируемых счетчиков *PCA*. В микроконтроллерах семейства *MCS51* имеется регистр разрешения прерываний *IE*, который существует с самых первых модификаций.

В регистре *IE* были зарезервированы два бита, один из которых *IE.6* стал использоваться как бит *EC* для разрешения прерывания от PCA (от таймера/счетчика и всех модулей). В этом же регистре *IE* имеется бит *EA* (*IE.7*) запрещения прерывания от всех устройств микроконтроллера. Назовем его глобальным. Следовательно, для разрешения обработки прерываний от *PCA* необходимо установить оба бита *EA* и *E*C в регистре *IE*. Разрешение прерывания от таймера/счетчика осуществляется установкой бита *ECF* в регистре *CMOD*. Разрешение прерывания от модулей осуществляется установкой битов *ECCFn* в регистрах *CCAPMn*. Если прерывания от источников возникают одновременно, то определение источника осуществляется программно, анализируя содержимое регистра *CCON*.

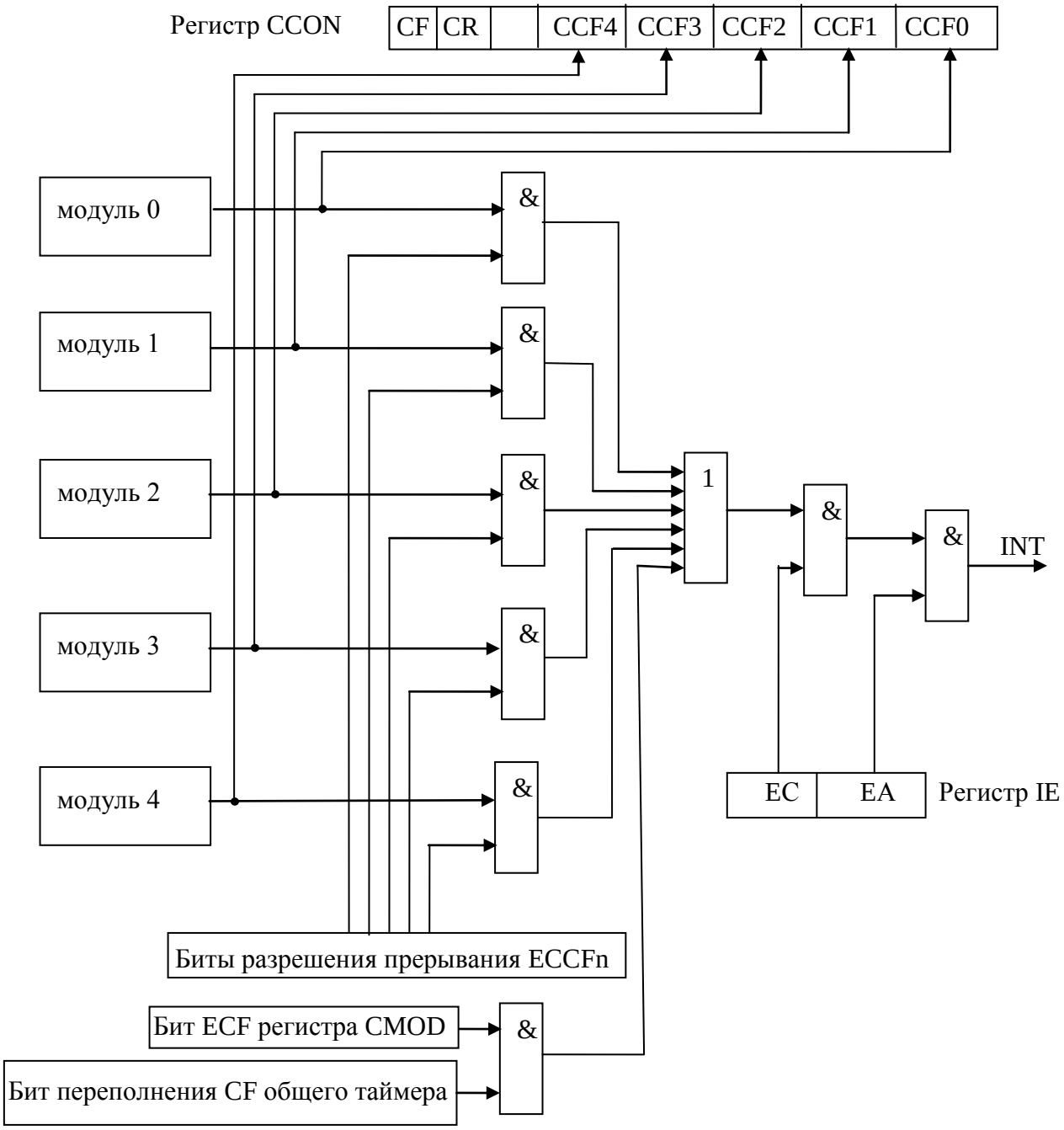


Рис. 6.8. Формирование прерывания от PCA

Лекция 13

7. Микроконтроллеры с RISC архитектурой

7.1. Микроконтроллеры AVR

Компания Atmel выпускает 8-разрядные микроконтроллеры с высокой производительностью и малым потреблением энергии. Для достижения указанных свойств используются достижения микроэлектроники последних лет. Микроконтроллер построен на основе RISC архитектуры, что позволяет применить двухступенчатый конвейер команд, выполнение которых осуществляется за один такт тактового генератора. 32 регистров общего назначения универсальны, так как в них может быть записан результат операции. Внутренняя память программ и данных имеют большой объем и построены на основе Гарвардской архитектуры. Развитое встроенное периферийное оборудование позволяет применять микроконтроллеры AVR во многих областях техники.

7.1.1. Структурная схема микроконтроллера AVR

На рис.7.1.1 показана типовая структурная схема микроконтроллера AVR. Структурная схема микроконтроллера состоит из АЛУ, внутренней памяти программ и памяти данных и внутренних периферийных устройств.

Память Программ | АЛУ и память данных | Внутренние периферийные устройства

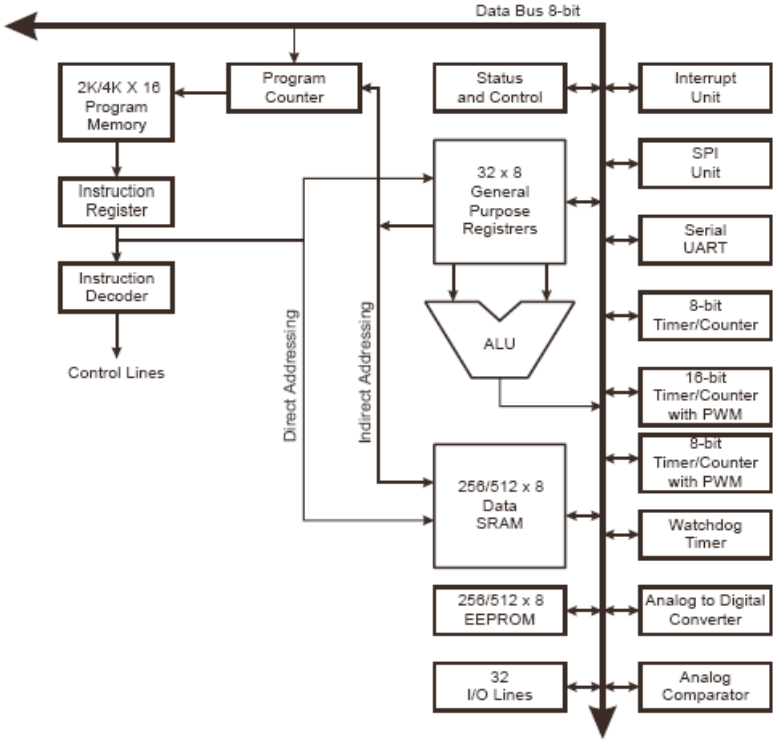


Рис. 7.1.1. Структурная схема микроконтроллера AVR

Внешние устройства могут быть подключены к четырем портам ввода вывода (32 линии I/O).

7.1.2. Организация памяти

В микроконтроллере использована концепция Гарвардской архитектуры, в соответствии с которой внутренняя память состоит их памяти программ Program Memory и памяти данных Data Memory (рис.7.1.2).

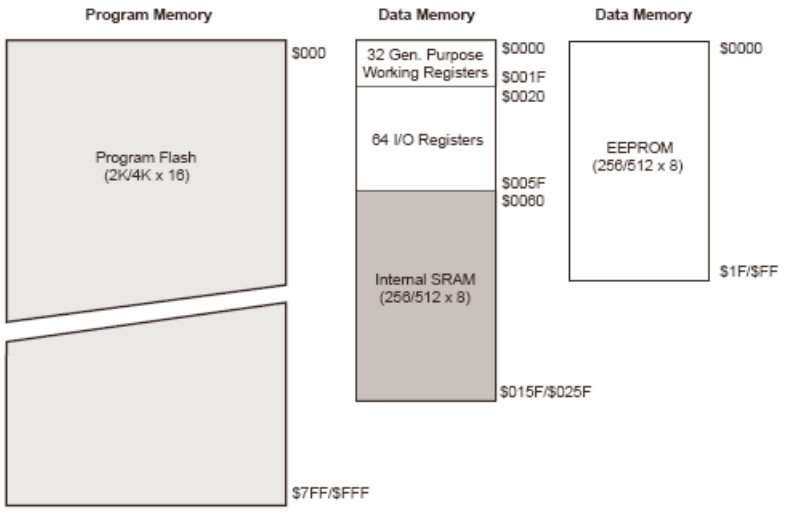


Рис. 7.1.2. Организация памяти

В памяти программ содержатся 16-разрядные команды (инструкции), причем пока команда выполняется в процессоре микроконтроллера, следующая команда выбирается из памяти, в результате одна команда может быть выполнена за один такт. Память программ реализована как флэш память и может быть перепрограммирована (не менее 1000 циклов записи/стирания), ее емкость зависит от конкретной модификации микроконтроллера.

В состав микроконтроллера входят два типа внутренней памяти данных: ОЗУ (RAM) и энергонезависимая память данных (EEPROM не менее 100000 циклов записи/стирания) каждая с собственным адресным пространством.

Память данных ОЗУ состоит из трех областей: универсальных регистров общего назначения (файл из 32 регистров), регистров для обслуживания встроенных интерфейсных устройств (64 регистров I/O) и внутреннего ОЗУ (SRAM).

На рис.7.1.3 показан состав регистров общего назначения. Регистры имеют имена от R0 до R31, и вся область регистров общего назначения поделена на две половины. Регистры R0 – R15 могут быть использованы во всех командах кроме команд *SBCI, SUBI, CPI, ANDI, ORI*, в которых использованы операнды константа и содержимое регистра, и команда *LDI* с

непосредственной адресацией. Перечисленные команды используются с регистрами второй половины регистрового файла.

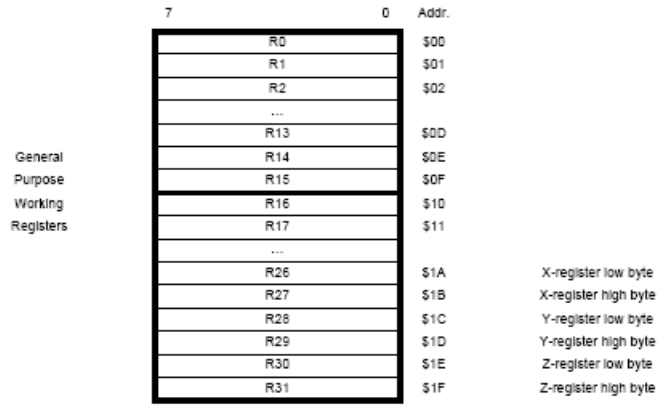


Рис. 7.1.3. Состав регистров общего назначения

Три пары регистров R26,R27 – X, R28,R29 – Y, R30,R31 – Z могут быть использованы в командах с косвенной адресацией, разрядность операндов в которых более восьми. Как показано на рис.3 каждый регистр кроме имени также может быть ассоциирован с адресом внутренней памяти данных и указан прямо по адресу. Далее на рисунках шестнадцатеричное значение обозначается знаком $, стоящим перед шестнадцатеричной цифрой или буквой.

Регистры I/O внутренних интерфейсных устройств пронумерованы от $00 до $3F (0 – 64), но поскольку адресное пространство внутренней памяти данных имеет сквозную нумерацию адресов ячеек памяти, то, как показано на рис.4 адреса регистров I/O начинаются с адреса $0020 вслед за адресами регистров общего назначения.

Соответствие адресов регистров в адресном пространстве внутренней памяти данных показано на рис. 7.4.

Память SRAM начинается с адреса $0060. Стек может быть размещен в любом месте памяти SRAM и его глубина ограничена только общим адресным пространством SRAM. Регистр-указатель стека SP состоит из двух регистров SPH и SPL с адресами $3E ($5E) и $3D ($5D) соответственно.

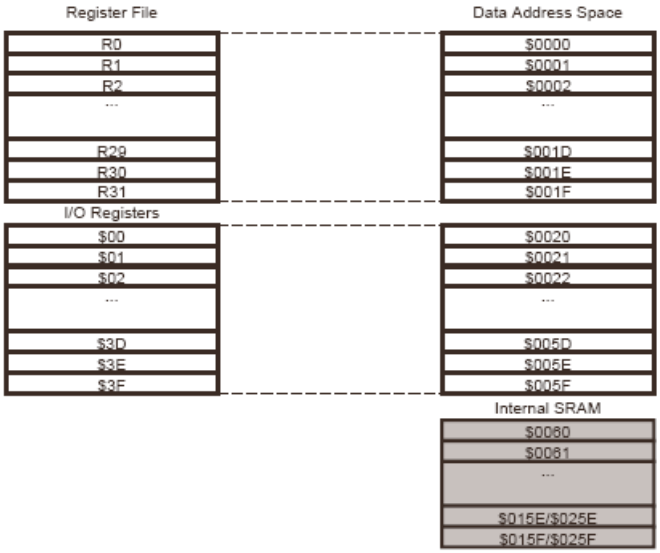


Рис. 7.1.4. Соответствие адресов регистров в адресном пространстве внутренней памяти данных

Инициализация регистров должна быть произведена программистом после сброса.

Для записи или чтения памяти EEPROM в области регистров I/O имеются регистры адреса, регистр данных и регистр управления. В регистре управления программист может установить режим записи или чтения и бит разрешения прерывания по готовности памяти EEPROM. Время записи лежит в пределах 2,5 – 4мс в зависимости от напряжения источника питания.

7.1.3. Система команд

Для выбора операндов в командах микроконтроллера используются прямая, косвенная и непосредственная адресация. Далее при описании команд применяются следующие обозначения:

Rd: R0-R31 or R16-R31 (в зависимости от используемой команды);

Rr: R0-R31;

b: константа для указания адреса или имени бита;

s: флаг в командах условного перехода и в командах побитных операций;

P: адрес или имя регистра I/O;

K: 8-разрядная константа, может быть использовано выражение;

k: константа, разрядность которой зависит от используемой команды;

q: смещение в командах с косвенной адресацией;

При выполнении арифметических и логических команд, в командах битовых операций формируются флаги, которые содержатся в регистре состояния SREG (5Fh). Формат регистра SREG показан на рис.7.1.5.

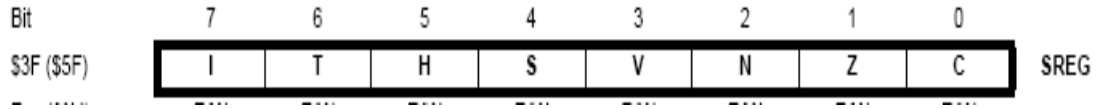


Рис. 7.1.5. Формат регистра состояния SREG

C – флаг переноса;
Z – флаг нуля результата;
N – флаг отрицательного результата;
V – флаг арифметического переполнения;
S – флаг знака, причем S =N⊕V;
H – Промежуточный перенос (из младшей тетрады в старшую);
T – место для записи значения бита в командах побитных операций;
I – если I=0, то запрещены все прерывания, если I=1, то разрешены прерывания только от тех источников, в регистрах управления которых установлен соответствующий бит.

Система команд состоит из 118 базовых команд, объединенных в четыре группы: группу команд арифметических и логических операций (табл. 7.1.1), группу команд пересылок (табл. 7.1.2), группу команд передачи управления (табл. 7.1.3) и группу команд побитных операций.

Таблица 7.1.1

Команды арифметических и логических операций

Мнемо-ника	Опера нды	Описание команды	Операция	Флаги	Такты
ADD	Rd, Rr	Сложение без учёта флага C	Rd ¬ Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Сложение с учётом флага C	Rd ¬ Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rd, k	Сложение с 16-р. константой	Rd+1:Rd ¬ Rd+1:Rd + k	Z,C,N,V	2
SUB	Rd, Rr	Вычитание без учёта заёма C	Rd ¬ Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Вычитание константы из Rd	Rd ¬ Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Вычитании с учётом заёма C	Rd ¬ Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Вычитании K с учётом заёма C	Rd ¬ Rd - K - C	Z,C,N,V,H	1
SBIW	Rd, k	Вычитании 16-р. константы	Rd+1:Rd¬ Rd+1:Rd - k	Z,C,N,V	2
AND	Rd, Rr	Поразрядное логическое И	Rd: Rd&Rr	Z,N,V	1
ANDI	Rd, K	Поразрядное И с константой K	Rd: Rd&K	Z,N,V	1
OR	Rd, Rr	Поразрядное ИЛИ	Rd: Rd v Rr	Z,N,V	1
ORI	Rd, K	Поразрядное ИЛИ	Rd: Rd v K	Z,N,V	1
EOR	Rd, Rr	Исключающее ИЛИ	Rd: Rd A Rr	Z,N,V	1
COM	Rd	Обратный код	Rd: $FF - Rd	Z,C,N,V	1
NEG	Rd	Дополнительный код	Rd: $00 - Rd	Z,C,N,V,H	1
SBR	Rd,K	Установить бит(ы)	Rd: Rd v K	Z,N,V	1
CBR	Rd,K	Обнулить бит(ы)	Rd: Rd&($FF - K)	Z,N,V	1
INC	Rd	Инкремент	Rd ¬ Rd +	Z,N,V	1
DEC	Rd	Декремент	Rd ¬ Rd - 1	Z,N,V	1
TST	Rd	Тест регистра	Rd ¬ Rd&Rd	Z,N,V	1
CLR	Rd	Обнулить (очистить) регистр	Rd ¬ Rd - Rd	Z,N,V	1
SER	Rd	Установить разряды регистра	Rd ¬ $FF	None	1
MUL	Rd,Rr	Умножение без знака	R1, R0 ¬ Rd . Rr	C	2

Таблица 7.1.2

Команды пересылок

Мнемо-ника	Операнды	Описание команды	Операция	Флаги	Такты
MOV	Rd, Rr	Копирование регистра	Rd ¬ Rr	None	1
LDI	Rd, K	Загрузить константу в регистр	Rd ¬ K	None	1
LDS	Rd, k	Переслать содержимое SRAM	Rd ¬ (k)	None	3
LD	Rd, X	Косвенная адресация	Rd ¬ (X)	None	2
LD	Rd, X+	Косвенная адресация с post X+1	Rd ¬ (X), X ¬ X + 1	None	2
LD	Rd, -X	Косвенная адресация с pre X-1	X ¬ X - 1, Rd ¬ (X)	None	2
LD	Rd, Y	Косвенная адресация	Rd ¬ (Y)	None	2
LD	Rd, Y+	Косвенная адресация с post Y+1	Rd ¬ (Y), Y ¬ Y + 1	None	2
LD	Rd, -Y	Косвенная адресация с pre Y-1	Y ¬ Y - 1, Rd ¬ (Y)	None	2
LDD	Rd,Y+q	Косвенная адресация + смещение	Rd ¬ (Y + q)	None	2
LD	Rd, Z	Косвенная адресация	Rd ¬ (Z)	None	2
LD	Rd, Z+	Косвенная адресация с post Z+1	Rd ¬ (Z), Z ¬ Z+1	None	2
LD	Rd, -Z	Косвенная адресация с pre Z-1	Z ¬ Z - 1, Rd ¬ (Z)	None	2
LDD	Rd, Z+q	Косвенная адресация + смещение	Rd ¬ (Z + q)	None	2
STS	k, Rr	Запомнить в SRAM	(k) ¬ Rr	None	3
ST	X, Rr	Косвенная запись в SRAM	(X) ¬ Rr	None	2
ST	X+, Rr	Косвенная запись в SRAM	(X) ¬ Rr, X ¬ X + 1	None	2
ST	-X, Rr	Косвенная запись в SRAM	X ¬ X - 1, (X) ¬ Rr	None	2
ST	Y, Rr	Косвенная запись в SRAM	(Y) ¬ Rr	None	2
ST	Y+, Rr	Косвенная запись в SRAM	(Y) ¬ Rr, Y ¬ Y + 1	None	2
ST	-Y, Rr	Косвенная запись в SRAM	Y ¬ Y - 1, (Y) ¬ Rr	None	2
STD	Y+q,Rr	Косвенная запись в SRAM	(Y + q) ¬ Rr	None	2
ST	Z, Rr	Косвенная запись в SRAM	(Z) ¬ Rr	None	2
ST	Z+, Rr	Косвенная запись в SRAM	(Z)¬ Rr, Z ¬ Z + 1	None	2
ST	-Z, Rr	Косвенная запись в SRAM	Z ¬ Z - 1, (Z) ¬ Rr	None	2
STD	Z+q,Rr	Косвенная запись в SRAM	(Z + q)¬ Rr	None	2
LPM		Переслать из памяти программ	R0 ¬ (Z)	None	3
IN	Rd, P	Переслать из порта (регистр I/O)	Rd ¬ P	None	1
OUT	P, Rr	Переслать в порт (регистр I/O)	P ¬ Rr	None	1
PUSH	Rr	Запомнить в стеке	STACK ¬ Rr	None	2
POP	Rd	Извлечь из стека	Rd ¬ STACK	None	2

Таблица 7.1.3

Команды передачи управления

Мнемо-ника	Операнды	Описание команды	Операция	Флаги	Такты
RJMP	k	Относительный безусловный переход	PC ¬ PC + k + 1	None	2
IJMP		Косвенный безусловный переход	PC ¬ Z		2
JMP	k	Безусловный переход	PC ¬ k	None	3

RCALL	k	Относительный вызов подпрограммы	PC ¬ PC + k + 1	None	3
ICALL		Косвенный вызов подпрограммы	PC ¬ Z	None	3
CALL	k	Вызов подпрограммы	PC ¬ k	None	4
RET		Возврат из подпрограммы	PC ¬ STACK	None	4
RETI		Возврат из прерывания	PC ¬ STACK	I	4
CPSE	Rd,Rr	Сравнить и пропустить команду, если (Rd = Rr)	PC ¬ PC + 2 or 3	None	1, 2, 3
CP	Rd,Rr	Сравнить	Rd - Rr	Z,C,N,V,H	1
CPC	Rd,Rr	Сравнить с учётом флага C	Rd - Rr - C	Z,C,N,V,H	1
CPI	Rd,K	Сравнить с константой	Rd - K	Z,C,N,V,H	1
SBRC	Rr, b	Пропустить команду, если в регистре бит равен 0 (Rr(b)=0)	PC ¬ PC + 2 or 3	None	1, 2, 3
SBRS	Rr, b	Пропустить команду, если в регистре бит равен 1 (Rr(b)=1)	PC ¬ PC + 2 or 3	None	1, 2, 3
SBIC	P, b	Пропустить команду, если в I/O регистре бит равен 0 (I/O(P,b)=0)	PC ¬ PC + 2 or 3	None	1, 2, 3
SBIS	P, b	Пропустить команду, если в I/O регистре бит равен 1 (I/O(P,b)=1)	PC ¬ PC + 2 or 3	None	1, 2, 3
BRBS	s, k	Переход, если флаг в регистре SREG равен 1 (SREG(s) = 1)	PC ¬ PC+k + 1	None	1 / 2
BRBC	s, k	Переход, если флаг в регистре SREG равен 0 (SREG(s) = 0)	PC ¬ PC+k + 1	None	1 / 2
BREQ	k	Переход, если равно (Z = 1)	PC ¬ PC + k + 1	None	1/2
BRNE	k	Переход, если не равно (Z = 0)	PC ¬ PC + k + 1	None	1 / 2
BRCS	k	Переход, если C = 1	PC ¬ PC + k + 1	None	1 / 2
BRCC	k	Переход, если C = 0	PC ¬ PC + k + 1	None	1 / 2
BRSH	k	Переход, если равно или больше (C = 0)	PC ¬ PC + k + 1	None	1 / 2
BRLO	k	Переход, если меньше (C = 1)	PC ¬ PC + k + 1	None	1 / 2
BRMI	k	Переход, если N = 1	PC ¬ PC + k + 1	None	1 / 2
BRPL	k	Переход, если N = 0	PC ¬ PC + k + 1	None	1 / 2
BRGE	k	Переход, если N v V= 0	PC ¬ PC+ k + 1	None	1 / 2
BRLT	k	Переход, если N v V= 1	PC ¬ PC + k + 1	None	1 / 2
BRHS	k	Переход, если H = 1	PC ¬ PC + k + 1	None	1 / 2
BRHC	k	Переход, если H = 0	PC ¬ PC + k + 1	None	1 / 2
BRTS	k	Переход, если T = 1	PC ¬ PC + k + 1	None	1 / 2
BRTC	k	Переход, если T = 0	PC ¬ PC + k + 1	None	1 / 2
BRVS	k	Переход, если V=1	PC ¬ PC + k + 1	None	1 / 2
BRVC	k	Переход, если V = 0	PC ¬ PC + k + 1	None	1 / 2
BRIE	k	Переход, если прерывание разрешено (I = 1)	PC ¬ PC + k + 1	None	1 / 2
BRID	k	Переход, если прерывание запрещено (I = 0)	PC ¬ PC + k + 1	None	1 / 2

Таблица 7.1.4

Команды побитных операций

Мнемо-ника	Операнды	Описание команды	Операция	Флаги	Такты
LSL	Rd	Логический сдвиг	Rd(n+1) ←□Rd(n),Rd(0)	Z,C,N,V,H	1

		влево	←□0, C ←□Rd(7)		
LSR	Rd	Логический сдвиг вправо	Rd(n) ←□Rd(n+1),Rd(7) ←□0, C ←□Rd(0)	Z,C,N,V	1
ROL	Rd	Циклический сдвиг влево через флаг C	Rd(0) ←□C,Rd(n+1) ←□Rd(n), C ←□Rd(7)	Z,C,N,V,H	1
ROR	Rd	Циклический сдвиг вправо через флаг C	Rd(7) ←□C,Rd(n) ←□Rd(n+1), C ←□Rd(0)	Z,C,N,V	1
ASR	Rd	Арифметический сдвиг впрво	Rd(n) ←□Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Поменять местамитетрады	Rd(3..0) ←□Rd(7..4)	None	1
BSET	s	Установить флаг	SREG(s) ←□1 SREG(s)		1
BCLR	s	Сбросить флаг	SREG(s) ←□0 SREG(s)		1
SBI	P, b	Установить бит в регистре I/O	I/O(P, b) ←□1	None	2
CBI	P, b	Сбросить бит в регистре I/O	I/O(P, b) ←□0	None	2
BST	Rr, b	Переслать битна место флага T	T ←□Rr(b)	T	1
BLD	Rd, b	Переслать бит T на место b в регистре Rd	Rd(b) ←□T	None	1
SEC		Установить флаг C	C ←□1	C	1
CLC		Сбросить флаг C	C ←□0	C	1
SEN		Установить флаг N	N ←□1	N	1
CLN		Сбросить флаг N	N ←□0	N	1
SEZ		Установить флаг Z	Z ←□1	Z	1
CLZ		Сбросить флаг Z	Z ←□0	Z	1
SEI		Разрешить выбранные прерывания	I ←□1	I	1
CLI		Запретить все прерывания	I ←□0	I	1
SES		Установить флаг S	S ←□1	S	1
CLS		Сбросить флаг S	S ←□0	S	1
SEV		Установить флаг V	V ←□1	V	1
CLV		Сбросить флаг V	V ←□0	V	1
SET		Установить флаг T	T ←□1	T	1
CLT		Сбросить флаг T	T ←□0	T	1
SEH		Установить флаг H	H ←1	H	1
CLH		Сбросить флаг H	H ←□0	H	1
NOP		Нет операции		None	1
SLEEP		Sleep		None	1
WDR		Сброс сторожевого таймера		None	1

Рассмотрим две задачи как примеры выполнения команд. Задача номер дин: пусть требуется разметить стек в памяти SRAM, начиная с адреса

0060h., Для решения задачи необходимо в указатель стека загрузить значение 0060h. Указатель стека состоит из двух регистров: SPH и SPL, которые находятся в области I/O регистров, а команды с непосредственной адресацией для этой области памяти отсутствуют. Поэтому сначала следует использовать регистры общего назначения для загрузки адреса дна пустого стека, а затем переслать их содержимое в регистры SPH и SPL.

```
LDI R16,0
OUT SPH,R16
LDI R16,60h
OUT SPL,R16
```

Задача номер два: пусть требуется переслать строку символов, содержащуюся в памяти программ как константы, в память данных SRAM,.начиная с адреса 0100h.

```
        LDI R16,6       ; счетчик циклов (в строке символов 6 букв)
        LDI R28,0       ; R28,R29 – указатель на Y
        LDI R29,01
        LDI R30,0       ; R30,R31 – указатель на Z (адрес string)
        LDI R31,04
M1:     LPM             ; (R0)←( Z) содержимое ячейки памяти программ, адрес
                        ; которой содержится в Z, пересылается в R0
        ST Y+,R0        ; содержимое R0 пересылается командой с косвенной
                        ; адресацией в SRAM с пост инкрементом адреса Y
        INC R30
        DEC R16
        BRNE M1

        org 0400h
string: db 'IVANOV'
```

7.1.4. Порты ввода/вывода

К внешним устройствам микроконтроллер подключается через порты ввода вывода. Их в микроконтроллере четыре и обозначаются как порт A, порт B, порт C и порт D.

Выводы порта A двунаправленные и имеют нагрузочную способность выходного тока 20 *mA*, что позволяет непосредственно подключать к выводам порта без всяких согласований светодиодные устройства отображения информации (LED). Выводы порта могут служить аналоговыми входами АЦП.

Выводы порта B имеют такие же характеристики как у порта A. В табл.7.1.5 указаны дополнительные функции порта B.

Таблица 7.1.5

Дополнительные функции выводов порта B

Вывод порта B	Дополнительная функция вывода порта B
PB0	T0 (вход внешних сигналов таймера/счетчика T/C0)
PB1	T1 (вход внешних сигналов таймера/счетчика T/C1)
PB2	AINO (вход аналогового компаратора +)
PB3	AIN1 (вход аналогового компаратора –)
PB4	SS (выбор SPI Slave)
PB5	MOSI (SPI Master Output Slave Input)
PB6	MISO (SPI Master Input Slave Output)
PB7	SCK (SPI синхроимпульсы)

Выводы порта C имеют такие же характеристики как у порта A. Дополнительные функции имеют выводы PC6 и PC7, к которым может быть подключен кварцевый резонатор для формирования внешних тактовых сигналов таймера 2.

Выводы порта D имеют такие же характеристики как у порта A. В табл.7.1.6 указаны дополнительные функции порта D.

Таблица 7.1.6

Дополнительные функции выводов порта D

Вывод порта D	
PD0	RXD (вход приемника UART)
PD1	TXD (выход передатчика UART)
PD2	INT0 (вход сигнала от источника вешнего прерывания 0)
PD3	INT1 (вход сигнала от источника вешнего прерывания 1)
PD4	OC1B (выход канала сравнения B таймера/счетчика T/C1)
PD5	OC1A (выход канала сравнения A таймера/счетчика T/C1)
PD6	ICP (вход захвата события таймера/счетчика T/C1)
PD7	OC2 (выход канала сравнения таймера/счетчика T/C2)

На рис. 7.1.6 показана функциональная схема для одного из восьми разрядов порта A, которая может служить типовой для объяснения работы выводов всех портов кроме объяснения работы дополнительных функций.

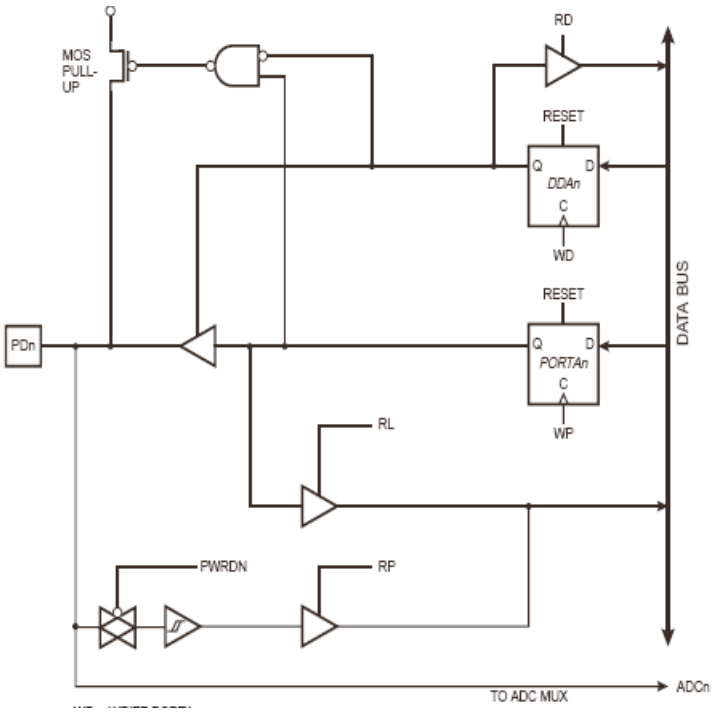


Рис. 7.1.6. Функциональная схема порта A

Все порты имеют три адреса: адрес защелки порта PORTA(B,C,D), адрес регистра для выбора направления передачи DDRA(B,C,D) и адрес выводов в режиме ввода PINA(B,C,D). Выводы PIN не являются регистром, поэтому значения сигналов в режиме ввода требуется запоминать в регистрах общего назначения или в памяти SRAM. Для настройки порта на вывод (output) в соответствующий разряд регистра DDR записывают 1, а в защелку PORT записывают 0 для формирования выходного сигнала низкого уровня или единицу для формирования выходного сигнала высокого уровня (табл. 7.1.7). Для настройки порта на ввод (input) в соответствующий разряд регистра DDR записывают 0. Если в защелке записан 0, то вывод PINn устанавливается в 3-е состояние. Чтобы обеспечить необходимый ток для формирования внешнего сигнала низкого уровня, необходимо избавиться от третьего состояния. С этой целью транзистор MOS используется как резистор и перевод транзистора в режим резистора осуществляется установкой защелки в единицу.

Таблица 7.1.7

Управление вводом выводом

Вывод порта B	Дополнительная функция вывода порта B
PB0	T0 (вход внешних сигналов таймера/счетчика T/C0)
PB1	T1 (вход внешних сигналов таймера/счетчика T/C1)

PB2	AINO (вход аналогового компаратора +)
PB3	AIN1 (вход аналогового компаратора –)
PB4	SS (выбор SPI Slave)
PB5	MOSI (SPI Master Output Slave Input)
PB6	MISO (SPI Master Input Slave Output)
PB7	SCK (SPI синхроимпульсы)

7.1.5. Таймеры счетчики

В состав микроконтроллера входят три таймера/счетчика(T/C0, T/C1, T/C2). Таймеры/счетчики T/C0 и T/C2 – 8-разрядные, а таймер/счетчик T/C1 16-разрядный. Тактовые сигналы на таймеры/счетчики T/C0 и T/C1 в режиме таймера подаются от внутреннего генератора, частота которого с помощью предделителя может быть разделена на 8. 64. 256, 1024. Коэффициент деления выбирается программно. В режиме счетчика сигналы от внешнего источника подаются на входы PB0, PB1 (табл. 7.1.5). Таймер/счетчик T/C2 может работать как от внутреннего тактового генератора с коэффициентами деления 8, 32, 64, 128, 256, 1024 так и от внешнего тактового генератора (входы PC6, PC7). В таймере/счетчике имеется канал сравнения текущего значения T/C2 с заданным. При сравнении формируется выходной сигнал OC2 на выводе PD7 (табл.6). Этот же вывод PD7 таймера/счетчика T/C2 может быть использован как выход 8-разрядного широтно-импульсного модулятора (PWM).

16-разрядный таймер счетчик T/C1 имеет устройство для захвата событий и в этом режиме может быть использован для измерения длительности внешних сигналов (вход ICP табл.6). Таймер/счетчик T/C1 также имеет возможность использования его как 8, 9, 10-разрядного широтно-импульсного модулятора (PWM). В нем имеется два канала сравнения текущего значении с заданным OC1A и OC1B (табл.6).

7.1.6. Сторожевой таймер

Сторожевой таймер (Watchdog Timer) предназначен для формирования внутреннего сигнала сброса микроконтроллера (MCU RESET, (рис. 7.1.7)). Таймер работает от собственного внутреннего генератора тактовых импульсов с частотой 1 Мгц (период 1 мкс). Интервал времени между сбросом в ноль таймера (WATCHDOG RESET) и формированием сигнала MCU RESET может быть программно установлен со значениями 16, 32, 64, 128, 256, 512, 1024, 2048 мс (16000 тактов и т.д.) Бит WDE служит для запрещения или разрешения формирования сигнала MCU RESET, а выбор длительность интервала выбирается битами WDP0, WDP1, WDP2. Команда

WDR должна быть вставлена в текст программы, предназначенной для защиты от сбоев. Если по какой либо причине команда WDR не будет выполнена, то через интервал времени, заданный битами WDP, произойдет общий сброс микроконтроллера и перезапуск системы.

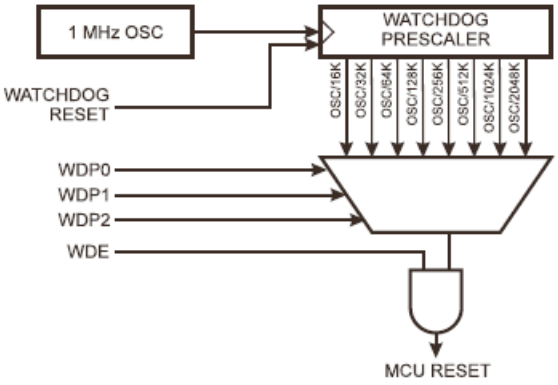


Рис. 7.1.7. Сторожевой таймер

7.1.7. Организация последовательного ввода вывода

В микроконтроллере реализован дуплексный прием/передача данных. С этой целью в UART введен регистр состояния USR, анализ битов которого позволяет во время записывать и считывать данные из буфера UDR. Буфер UDR имеет один и тот же адрес как для приемника так и для передатчика, но физически имеются два буфера данных, один для приемника другой для передатчика.

Последовательный ввод вывод данных осуществляется со скоростью V_{BAUD}, которая может быть задана программно в регистре UBRR.

$$V_{BAUD} = f_{clk} / 16(UBRR+1),$$

где f_{clk} – тактовая частота внутреннего генератора;

UBRR – число, которое должно быть записано в регистр UBRR.

Например, при тактовой частоте $f_{clk}=11,0592$ Мгц для обеспечения скорости 9600 бит/с в регистр UBRR следует записать число 71 ($UBRR+1=11059200/16*9600=72$).

Формат данных может быть задан без проверки на четность (8 бит) или с проверкой на четность 9 (бит). Проверяется потеря стопового бита (Framing Error detection), случай приема нового байта, если предыдущий еще не был прочитан (Overrun detection). В регистре состояний USR формируются биты готовности приемника, готовности передатчика и бит, когда регистр данных передатчика пуст. Указанные биты могут быть обработаны программно или по прерыванию. Кроме того аппаратно фильтруются шумы и осуществляется обнаружение ложного старт-бита.

Интерфейс SPI (Serial Peripheral Interface) предназначен для обмена данными между микроконтроллером и периферийными устройствами в синхронном режиме со скоростью до 1,5 Мбит/с. Подключение микроконтроллера (Master) к периферийному устройству (Slave) показано на рис.7.1.8. В режиме, когда микроконтроллер работает как Master, сигналы

синхронизации (SCK) формируются на выводе PB7 (табл.7.1.5). Если Master передает данные, то после записи байта в регистр записи/чтения данных (SPDR) на выводе SCK формируются сигналы синхронизации, а на выводе MOSI (Master Output Slave Input – PB5) сигналы данных. После передачи байта генерация синхроимпульсов прекращается и в регистре управления SPCR устанавливается флаг конца передачи (преобразования) SPIF, который может быть обработан по прерыванию. Если Master принимает данные, то они поступают на вход MISO (Master Input Slave Output – PB6). Для выбора устройства Slave формируется сигнал ss на выводе PB4.

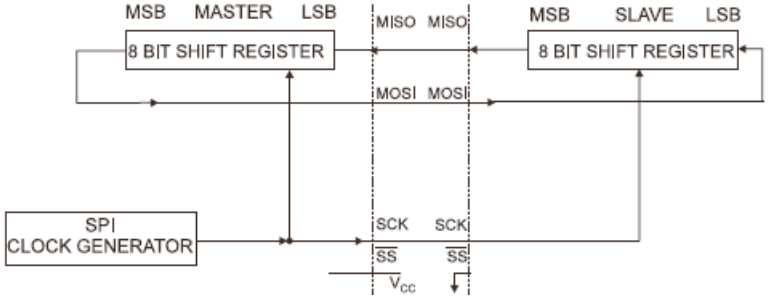


Рис. 7.1.8. Схема соединений между микроконтроллером и периферийным устройством по интерфейсу SPI

На рис 7.1.9 показана временная диаграмма передачи байта от микроконтроллера к периферийному устройству.

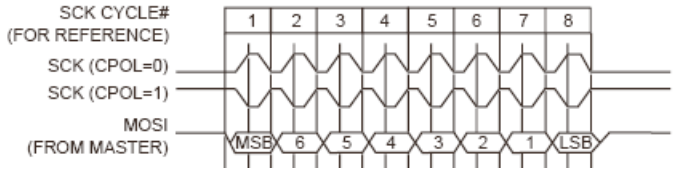


Рис. 7.1.9. Временная диаграмма передачи байта

В регистре управления SPCR имеется возможность устанавливать режимы интерфейса SPI: менять полярность и фазу синхроимпульсов, менять последовательность передачи битов (начинать передачу со старшего бита MSB или с младшего бита LSB), выбирать для микроконтроллера режим Master или Slave,устанавливать скорость передачи (f_{clk} /4, f_{clk}/16, f_{clk}/64, f_{clk}/128).

7.1.8. Аналого-цифровой преобразователь и компаратор

В микроконтроллер встроен 8-канальный 10-разрядный аналого-цифровой преобразователь (АЦП). Работа АЦП обеспечивается генератором тактовой частоты в диапазоне 50 – 200кГц. Преобразование аналоговой величины в дискретную осуществляется за 13 тактов и, следовательно,

минимальное время преобразования равно 65мкс. Частота тактового генератора АЦП может быть задана установкой коэффициентов деления 2, 4, 8, 16, 32, 62, 128 в регистре управления. АЦП работает в двух режимах: режим запуска АЦП программистом с помощью установки бита старта преобразования (ADC Start Conversion) и режим постоянного преобразования. Для АЦП используется внешний источник опорного напряжения V_{REF}, напряжение которого равно 2,7 – 6,0В.

Аналоговый компаратор сравнивает две аналоговые величины (рис.7.1.10), причем если напряжение на входе PB2, чем на входе PB3, то сигнал на выходе компаратора высокого уровня и бит сравнения ACO, равен единице. При сравнении бит ACO сбрасывается в ноль. Бит ACO, может быть использован в режиме захвата таймера/счетчика T/C1. Момент сравнения может быть обработан по прерыванию. Программист может указать три способа формирования бита прерывания ACI; по уровню сигнала на выходе компаратора, по возрастанию сигнала на выходе компаратора или по спаду сигнала на выходе компаратора. Выбор способа формирования бита ACI осуществляется установкой битов ACIS1, ACIS2.

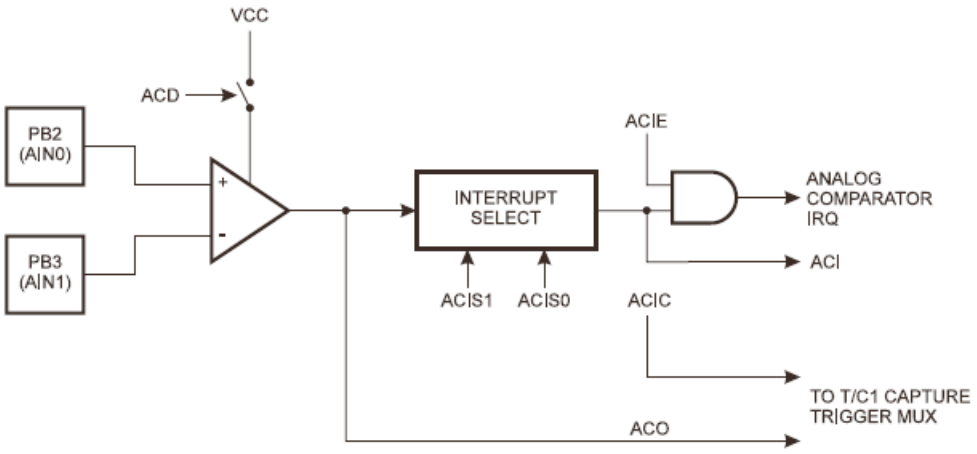


Рис. 7.1.10. Аналоговый компаратор

7.1.9. Организация прерываний

В прерывания в микроконтроллере делятся на внешние и внутренние. К внешним прерываниям относятся прерывания, которые формируются от сигналов, поступающих от внешних источников прерывания. Для организации внешних прерываний предназначены три регистра: глобальный регистр GIFR, регистр разрешения/запрещения прерывания GIMSK, (регистр маски) и регистр управления MCUCR. Сигналы от внешних источников прерываний INT0, INT1 подаются соответственно на входы PD2, PD3 порта D. Обработка сигналов INT0, INT1 может быть осуществлена по низкому уровню сигнала либо по возрастанию или по спаду сигнала. Выбор типа обработки сигнала производится установкой соответствующих битов в

регистре MCUCR. Запоминание запросов прерываний от внешних сигналов INT0, INT1 осуществляется в регистре GIFR. Обработка прерываний от внешних источников производится в том случае, если установлены соответствующие биты в регистре GIMSK и бит I общего разрешения прерываний в регистре SREG.

Внутренние прерывания формируются при переполнении таймеров, при приеме или передаче байта последовательного ввода вывода, при преобразовании аналоговой величины в дискретную, при сравнении аналоговых величин с помощью компаратора. Для организации прерываний от внутренних источников прерываний имеются соответствующие регистры для маскирования прерываний, для запоминания битов запросов прерываний.

В микроконтроллере нет регистра для указания приоритета источника прерывания. Если возникает прерывание, то автоматически сбрасывается флаг I общего разрешения прерывания и программист должен сам выбрать момент, когда его снова установить.

За каждым источником прерывания закреплен вектор прерывания. Ниже показан пример как может быть организован переход к соответствующему обработчику прерываний.

```
Адрес          Команда          Комментарий
0000h          rjmp  on_reset      ;переход к программе после сброса
0001h          rjmp   EXT_INT0 ;переход к обработчику внешнего прерывания
0
0002h          rjmp   EXT_INT1 ;переход к обработчику внешнего прерывании
1
0003h          rjmp   T2_COMP  ;переход к обработчику канала сравнения
T/C2
0004h          rjmp   T2_OVF     ;переход к обработчику по переполнению
T/C2
0005h          rjmp   T1_CAPT  ;переход к обработчику захвата события
0006h          rjmp   T1_COMPA ;переход к обработчику канала сравнения A
0007h          rjmp   T1_COMPB ;переход к обработчику канала сравнения B
0008h          rjmp   T1_OVF     ;переход к обработчику по переполнению
T/C1
0009h          rjmp   T0_OVF     ;переход к обработчику по переполнению
T/C0
000Ah          rjmp   SPI          ;переход к обработчику по готовности SPI
000Bh          rjmp   UART_RxD;переход к обработчику по готовности RxD
000Ch          rjmp   UART_DRE;переход к обработчику, если буфер UART
пуст
000Dh          rjmp   UART_TxD;переход к обработчику по готовности TxD
000Eh          rjmp   ADC          ;переход к обработчику по готовности АЦП
000Fh          rjmp   EE_RDY     ;переход к обработчику по готовности
EEPROM
0010h          rjmp   ANA_COMP;переход к обработчику компаратора;
0011h  on_reset:   <первая команда программы>
```

7.1.10. Сброс микроконтроллера

В микроконтроллере используются три способа сброса:

– сброс при включении питания V_{CC};

– сброс от внешнего источника, когда установлен низкий уровень на входе RESET в течение времени более двух периодов тактовой частоты внутреннего генератора;

– сброс от сторожевого таймера.

Лекция 14

7.2. Микроконтроллеры PIC

7.2.1. Семейства микроконтроллеров PIC

В настоящее время фирмой Microchip выпускаются 8- и 16-битные микроконтроллеры общего назначения семейства PIC, а также цифровые сигнальные процессоры семейства dsPIC, 16- и 32-разрядные. Перечень семейств микроконтроллеров приведен на рис. 7.2.1.

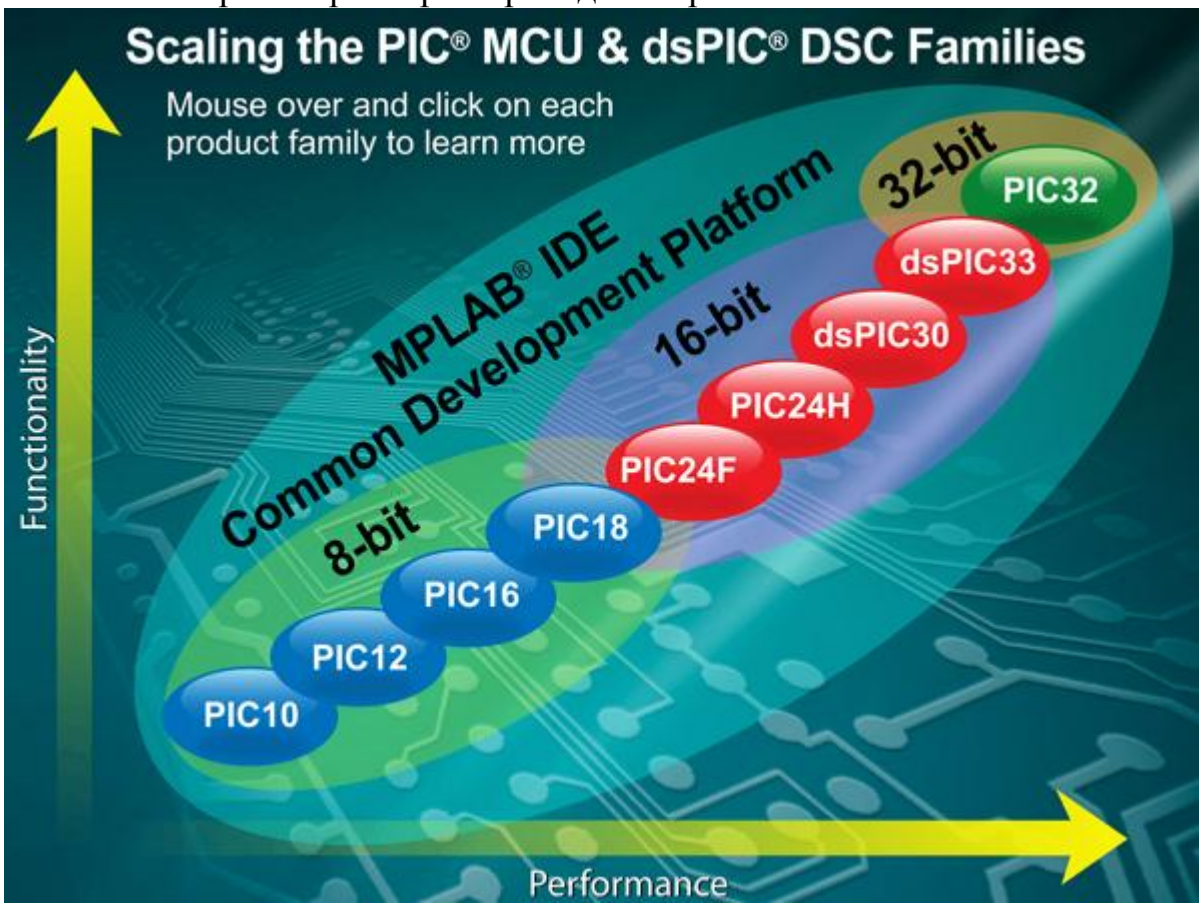


Рис. 7.2.1. Семейства микроконтроллеров PIC

Семейства 8-битных контроллеров PIC10 и PIC12 ориентированы на применение в простых системах, использующих малое количество управляющих сигналов, поэтому контроллеры этого семейства выпускаются в корпусах малого форм-фактора (рис.7.2.2),зачастую всего с 6 или 8 выводами (до 14).

Рис. 7.2.2. Микроконтроллеры PIC в корпусах малого форм-фактора

Семейства PIC16 и PIC18 - это 8-битные микроконтроллеры общего назначения в корпусах среднего форм-фактора с количеством выводов как правило 40 или 44 (от 14 до 44).

Рис.7.2.3. Микроконтроллеры PIC в корпусах среднего форм-фактора

8-битные контроллеры PIC имеют систему команд RISC, разрядность слова в памяти программ 12, 14 или 16 бит. Системы команд схожи, однако для разных семейств не являются совместимыми ни на уровне кодов (разная разрядность памяти программ), ни на уровне операций.

Назначение выводов в корпусах одного форм-фактора для контроллеров PIC разных семейств как правило одинаковое, что позволяет с появлением более современных моделей контроллеров модернизировать существующие схемы на базе этих контроллеров без изменения схем путем простой замены контроллера на более современный. К сожалению, в общем случае это требует переписывания старого программного обеспечения для контроллеров новых моделей. Фирмой Microchip разработаны и предлагаются стандартные методики адаптации программ для контроллеров новых версий, что в данном случае несколько упрощает задачу программистов.

Семейство PIC24 - это 16-разрядные микроконтроллеры.
dsPIC30 и dsPIC33 относятся к классу цифровых сигнальных процессоров за счет специальных расширений системы команд. Все 16-битные контроллеры имеют одинаковую архитектуру, базовую систему команд и назначение

выводов корпусов. Архитектура и система команд этих контроллеров, также относящаяся к классу RISC, оптимизированы для использования компиляторов языка Си.

Для разработки и отладки программного обеспечения для всех семейств выпускаемых микроконтроллеров фирма Microchip выпускает интегрированную среду разработки и отладки MPLab. Бесплатная версия среды включает ассемблер MPASM, программный эмулятор с моделями всех микроконтроллеров PIC, отладчик кодов и интерфейсы к популярным программаторам и внутрисхемным эмуляторам. Дополнительно у пользователя имеется возможность приобрести компилятор языка Си.

В рамках данного курса будут рассмотрены 8-битные контроллеры PIC16F74 как классические представители контроллеров PIC, демонстрирующие особенности, достоинства и недостатки подходов, общих для всех 8-битных контроллеров Microchip.

Следует заметить, что семейство PIC16 считается несколько устаревшим, и на смену ему повсеместно приходят контроллеры семейства PIC18. Основное отличие этих семейств -разрядность слова в памяти программ, составляющая 14 бит и 16 бит соответственно. 14-разрядные слова в памяти программ PIC16 создают для программиста некоторые неудобства в силу отсутствия кратности привычным 8-разрядным байтам. Отчасти, в силу этого для рассмотрения выбрано именно это семейство.

7.2.2. Особенности архитектуры PIC16F74

Архитектура микроконтроллеров семейства PIC16 типична для большинства современных микроконтроллеров – они построены по Гарвардской архитектуре с раздельными адресными пространствами для ПЗУ программ и ОЗУ. На борту микросхемы имеется три таймера, АЦП, два компаратора, синхронный и асинхронный последовательные порты, сторожевой таймер, таймер начальной инициализации, устройство параллельного ведомого порта, пять портов ввода-вывода с мультиплексированными функциями выводов.

Архитектура микроконтроллера представлена на рис. 7.2.4.

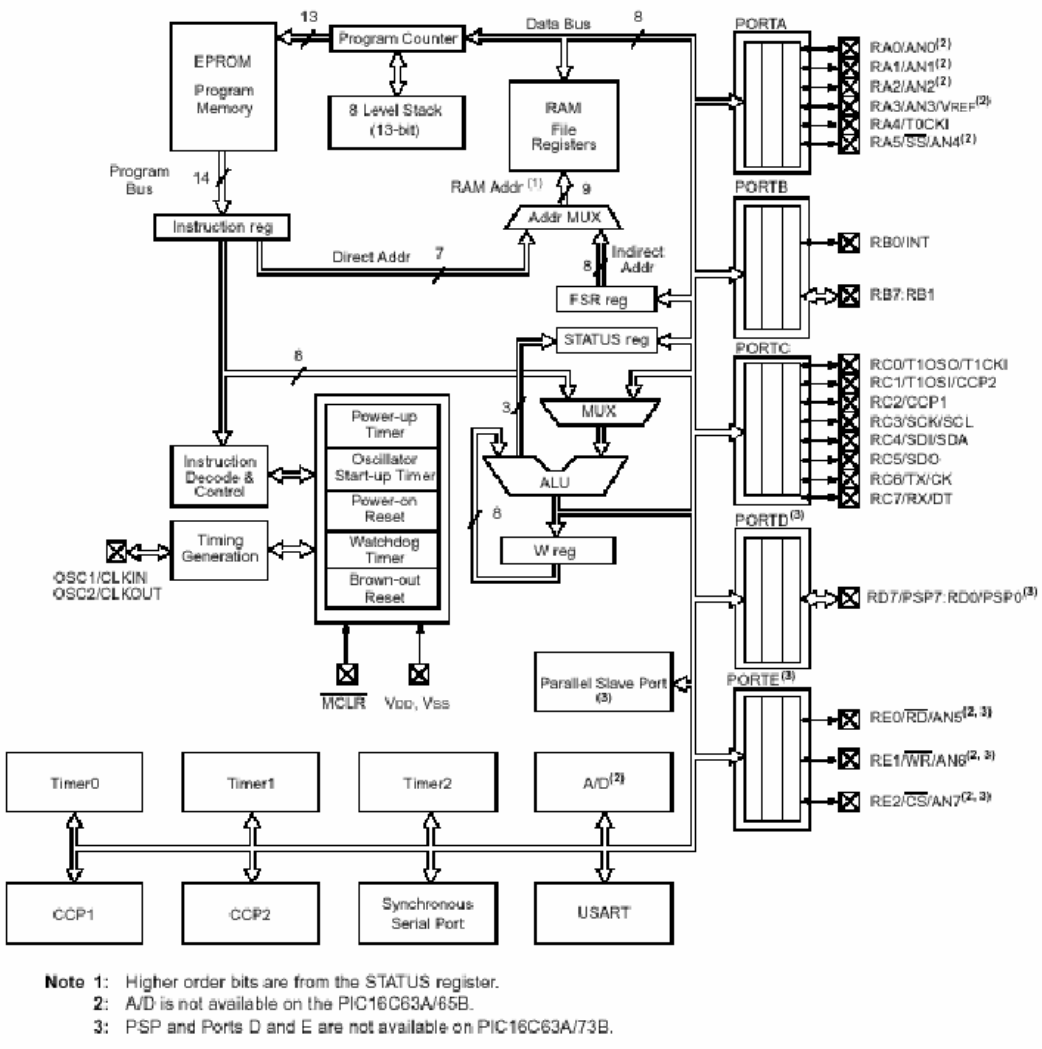


Рис. 4. Архитектура микроконтроллера PIC16F74

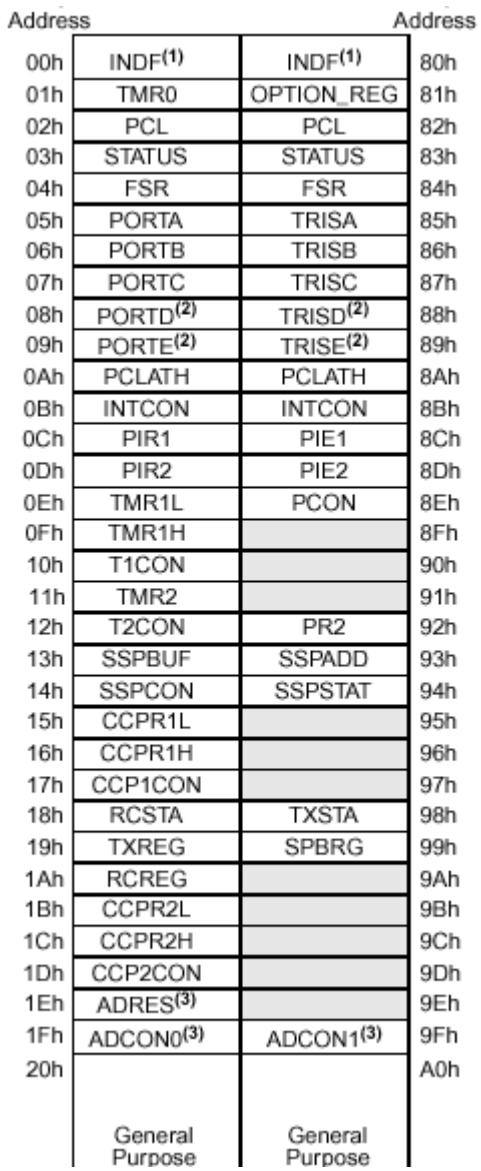
Основными критериями, которыми руководствовались разработчики семейства PIC16, были дешевизна и быстродействие. В жертву этим принципам частично принесена полнота и понятность системы команд: среди команд нет не только деления или умножения, но даже сложения с учетом переноса, а восприятие логики условных переходов или вычитаний констант требует от программиста существенных мысленных усилий – зато такая система команд легко поддается аппаратной интерпретации.

Ниже перечислены основные особенности рассматриваемых микроконтроллеров. Контроллер выполнен по технологии КМОП. Питание 5В. Все внутренние устройства хранения – статические, рабочие частоты – от 0 до 20 МГц. Выводы портов достаточно мощные и позволяют, например, непосредственно управлять светодиодными индикаторами, как в режиме нуля, так и в режиме единицы. Максимально допустимый выходной ток – 20 мА на вывод, при этом суммарный ток на микросхему в целом не должен превышать 200 мА.

Память программ и память данных разделены, их адресные пространства индивидуальны. По внутренним шинам одновременно могут

передаваться 14-разрядные коды команд и 8-разрядные данные. Каждая команда занимает ровно одно слово в памяти команд (14 бит). Память команд адресуется словами. Отсутствуют команды явного чтенияиз памяти программ (например, чтения констант). Память программ встроенная, объемом 4k слов, выполнена по технологии flash-ПЗУ. Память программ разбита на 2 страницы по 2k слов, во избежание проблем с адресацией рекомендуется размещать программу (если это возможно) в младших 2k памяти программ. Ранее выпускались модификации этого микроконтроллера с масочным и однократно программируемым ПЗУ программ, а также отладочные варианты, снабженные ПЗУ сультрафиолетовым стиранием.
Подключение внешней памяти программ или данных по схеме, аналогичной используемой в микроконтроллерах семейства MCS51, в контроллерах семейства PIC не предусмотрено. Для организации вызовов подпрограмм и прерываний имеется аппаратный стек глубиной 8 уровней. Стек программно недоступен, т.е. отсутствуют команды, аналогичные PUSH и POP, а также любая возможность явно обратиться к ячейкам стека. Стек не отображается на адресное пространство памяти данных. Стек имеет кольцевую структуру, то есть при переполнении стека ошибка не возникает, но происходит запись поверх значений, лежащих на дне стека.
При включении питания микроконтроллер стартует с адреса 0000h. Так как по адресу 0004h располагается вектор (единственного) прерывания, то по адресу 0000h обычно размещают команду перехода, а основную программу располагают с адреса 0005h или выше.

Память данных объемом 256 байт именуется "файлом регистров".Часть ячеек имеет общее назначение (192 байта), 64 ячейки представляют собой регистры специальных функций (РСФ). Все ячейки доступны побитно. В системе команд способ адресации к ячейкам общего назначения и РСФ не различается (формально для доступа к ячейкам памятиданных используется регистровая адресация). Память данных разделена на два банка, выбор банка осуществляется установкой определенного бита в управляющем регистре. Карта адресов памяти данных представлена на рис. 7.2.5.



Address			Address
00h	INDF(1)	INDF(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	PORTC	TRISC	87h
08h	PORTD(2)	TRISD(2)	88h
09h	PORTE(2)	TRISE(2)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	PIR1	PIE1	8Ch
0Dh	PIR2	PIE2	8Dh
0Eh	TMR1L	PCON	8Eh
0Fh	TMR1H		8Fh
10h	T1CON		90h
11h	TMR2		91h
12h	T2CON	PR2	92h
13h	SSPBUF	SSPADD	93h
14h	SSPCON	SSPSTAT	94h
15h	CCPR1L		95h
16h	CCPR1H		96h
17h	CCP1CON		97h
18h	RCSTA	TXSTA	98h
19h	TXREG	SPBRG	99h
1Ah	RCREG		9Ah
1Bh	CCPR2L		9Bh
1Ch	CCPR2H		9Ch
1Dh	CCP2CON		9Dh
1Eh	ADRES(3)		9Eh
1Fh	ADCON0(3)	ADCON1(3)	9Fh
20h			A0h
	General Purpose	General Purpose	

Рис. 7.2.5. Адреса регистров специальных функций и общего назначения в файле регистров

В архитектуре контроллера предусмотрен специальный 8-разрядный регистр аккумулятора. Его мнемоническое обозначение несколько необычно – W (working register), а не A или Acc, что является более привычным. Аккумулятор не отображается на адресное пространство памяти данных. Аккумулятор используется при выполнении большинства арифметических действий и операций пересылок. Вместе с тем, например, операции сдвига возможны только над ячейками файла регистров, но не над аккумулятором; также аккумулятор недоступен побитно.

Адресация к файлу регистров – прямая или косвенная. Для организации косвенной адресации к файлу регистров используется прямое обращение к псевдорегистру INDF (по адресу 00). При обращении к INDF производится реальное косвенное обращение по адресу, содержащемуся в регистре FSR.

Контроллер имеет 5 портов ввода-вывода (33 линии):

PORTA – 6-разрядный, 5 из 6 выводов могут использоваться как входы АЦП;
PORTB – 8-разрядный, изменение состояние выводов этого порта может приводить к прерыванию;
PORTC – 8-разрядный, входы со свойствами триггера Шмитта, выводы порта могут использоваться как входы/выходы различных устройств, например – имеющегося на борту контроллера асинхронного приемопередатчика (UART), устройства ШИМ и т.д.;
□□PORTD – 8-разрядный, входы со свойствами триггера Шмитта, может быть
сконфигурирован как параллельный ведомый порт;
□□PORTE – 3-разрядный, входы со свойствами триггера Шмитта, линии порта могут использоваться как входы АЦП или как управляющие для параллельного ведомого порта.

Каждый вывод любого из портов может быть независимо от других сконфигурирован для работы в режиме входа или выхода, для этого служат регистры TRISA – TRISE. В режиме входа сопротивление вывода велико, и его можно рассматривать как вывод, находящийся в третьем (высокоимпедансном) состоянии.
На кристалле имеются следующие периферийные устройства, в качестве выходов иливходов которых могут быть сконфигурированы выводы портов PORTA – PORTE:
– три таймера/счетчика;
□□– два модуля компаратора/ШИМ;
□□– один универсальный синхронно/асинхронный приемник/передатчик;
□□– один синхронный последовательный порт, функционирующий в режиме SPI или I2C;
□□– одно устройство 8-разрядного ведомого параллельного порта;
□□– быстродействующий 8-разрядный АЦП, который может быть скоммутирован с любым из 8 возможных аналоговых входов;
□□– сторожевой таймер с собственным RC-генератором;
□□– устройство сброса при падении напряжения питания (brown-out reset).
ЦАП среди устройств микроконтроллера отсутствует.
В микроконтроллере реализован режим останова, в этом режиме энергопотребление чрезвычайно низко.

7.2.3. Принцип "чтение-модификация-запись"

Типичная функциональная схема порта (в данном случае это PORTD, другие порты могут иметь дополнительные особенности) показана на рис. 7.2.6. Данная схема показывает, что при чтении из регистра порта всегда читается состояние ВЫВОДОВ порта (напряжение на ножках), а при записи в регистр порта записываемое значение попадает в регистр-защелку.

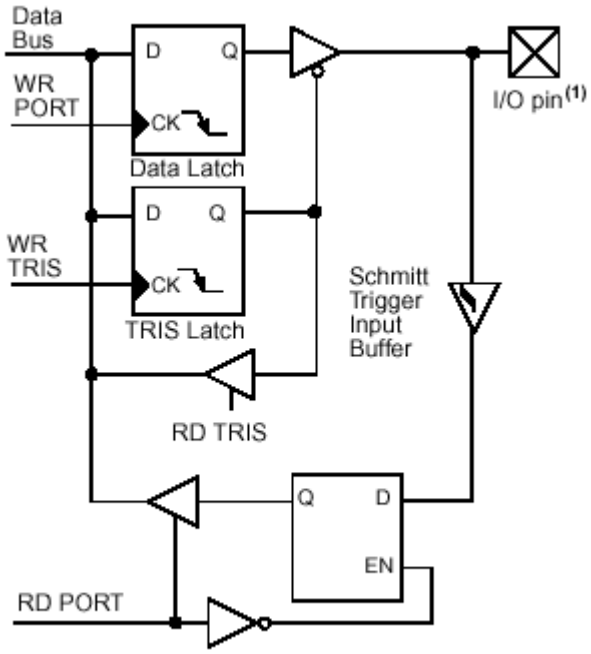


Рис. 7.2.6. Функциональная схема вывода порта PORTD

Команды установки отдельных битов следует применять к портам контроллера с осторожностью именно из-за особенностей реализации подхода "чтение-модификация- запись". При выполнении команды установки бита происходит следующее:

□□– в узел управления считывается состояние ВСЕХ выводов порта;

□□– состояние нужного бита в прочитанном значении изменяется;

□□– полученное значение заносится в регистр-защелку порта, который непосредственно управляет состоянием всех выводов, сконфигурированных как выходы.

Проблема может возникнуть при попытке последовательно изменить два бита, например 1 и 2:

□□– результатом первой команды будет изменение бита 1 в регистре защелке;

□□– если бит 1 сконфигурирован как выход, то время, за которое изменится напряжение на соответствующем выводе порта, зависит от емкостной нагрузки, изменение не происходит мгновенно;

□□– следующая команда, призванная изменить бит 2, произведет чтение напряжений с выводов порта, при этом вывод 1 может оказаться в переходном (предыдущем) состоянии;

□□– в прочитанном значении бит 2 будет модифицирован, но бит 1 окажется таким, каким был до выполнения предыдущей команды;

□□– в результате последовательного выполнения двух этих команд бит 1 может оказаться не модифицированным.

Выход из такой ситуации – при необходимости изменения нескольких битов на выходе порта следует либо проводить такое изменение одной операцией (работая с портом как с байтом), либо в явном виде считывать байт из порта,

модифицировать в нем биты по отдельности и записывать готовый байт обратно в порт также в явном виде (отдельной командой).

7.2.4. Особенности языка ассемблера и системы команд

Микроконтроллер PIC16F74 имеет систему команд с сокращенным набором инструкций (RISC). Все 35 команд микроконтроллера перечислены ниже. Команды либо не имеют операндов, либо работают с прямо адресуемыми регистрами внутренней памяти данных ("файла регистров") или непосредственно указываемыми в теле команды константами. Код любой команды умещается в 14-разрядное слово в памяти программ.
Для организации условных переходов имеется безусловный переход и команды, обеспечивающие пропуск следующей команды по условию.
Мнемоническое обозначение команд в языке ассемблера микроконтроллеров PIC также своеобразно.

Далее применяются следующие обозначения:
k означает литеральную константу (непосредственно адресуемый операнд).
f – адрес ячейки памяти данных (прямо адресуемый операнд).
d – уточняющий флаг направления, принимающий значение W или F и означающий, должен ли результат операции быть помещен в аккумулятор или в указанную в команде ячейку памяти. Данная особенность (возможность помещать результат операции аккумулятор-регистр как в аккумулятор, так и в регистр) обеспечивает системе команд существенную дополнительную гибкость.

Команды пересылки данных

MOVLW k
Заносит в аккумулятор число k.
MOVWF f
Заносит значение из аккумулятора в ячейку f.
MOVF f,d
Заносит содержимое ячейки f в аккумулятор или оставляет его в ячейке f, в зависимости от флага направления. Флаг нуля (Z) выставляется в соответствии с переносимым числом, что нехарактерно для операций пересылки в микроконтроллерах других семейств и может служить источником ошибок программиста. Эта команда позволяет, в частности, просто выполнить проверку нулевого значения в ячейке памяти, указав в качестве приемника данных саму ячейку (например, movf 100,f).

Арифметические команды

ADDWF f,d

Складывает содержимое аккумулятора с указанной ячейкой памяти f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Влияет на флаги C, DC, Z.

SUBWF f,d

Вычитает содержимое аккумулятора из ячейки памяти f (обратите внимание! – ячейка минус аккумулятор), результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Влияет на флаги C, DC, Z.

ADDLW k

Прибавляет к содержимому аккумулятора константу k. Влияет на флаги C, DC, Z.

SUBLW k

Вычитает из константы k содержимое аккумулятора (обратите внимание! – константа минус аккумулятор), результат заносит в аккумулятор. Влияет на флаги C, DC, Z.

ANDWF f,d

Выполняет операцию побитового И над содержимым аккумулятора и указанной ячейкой памяти f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Влияет на флаг Z.

IORWF f,d

Выполняет операцию побитового ИЛИ над содержимым аккумулятора и указанной ячейкой памяти f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Влияет на флаг Z.

XORWF f,d

Выполняет операцию побитового исключающего ИЛИ над содержимым аккумулятора и указанной ячейкой памяти f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Влияет на флаг Z.

ANDLW k

Выполняет операцию побитового И над содержимым аккумулятора и константой k, результат помещает в аккумулятор. Влияет на флаг Z.

IORLW k

Выполняет операцию побитового ИЛИ над содержимым аккумулятора и константой k, результат помещает в аккумулятор. Влияет на флаг Z.

XORLW k

Выполняет операцию побитового исключающего ИЛИ над содержимым аккумулятора и константой k, результат помещает в аккумулятор. Влияет на флаг Z.

INCF f,d

Прибавляет единицу к содержимому ячейки f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Обратите внимание: в системе команд нет команды инкремента аккумулятора. Влияет на флаг Z.

DECF f,d

Вычитает единицу из содержимого ячейки f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Обратите внимание: в системе команд нет команды декремента аккумулятора. Влияет на флаг Z.

SWAPWF f,d

Переставляет местами тетрады в значении, содержащемся в ячейке f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Обратите внимание: в системе команд нет команды перестановки тетрад в аккумуляторе. Влияет на флаг Z.

RLF f,d

Осуществляет сдвиг влево через перенос значения ячейки f (старший бит попадает в во флаг переноса, младший бит берется из флага переноса), результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Обратите внимание: в системе команд нет команд сдвига аккумулятора. Влияет на флаг C.

RRF f,d

Осуществляет сдвиг вправо через перенос значения ячейки f (младший бит попадает в во флаг переноса, старший бит берется из флага переноса), результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Обратите внимание: в системе команд нет команд сдвига аккумулятора. Влияет на флаг C.

CLRF f

Помещает в ячейку f нулевое значение (очищает ячейку). Устанавливает флаг Z.

CLRW

Помещает в аккумулятор нулевое значение (очищает аккумулятор). Устанавливает флаг Z.

COMF f,d

Образует дополнение для значения ячейки f (т.е. вычисляет "минус f", аналог команды NEG в i8086), результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Обратите внимание: в системе команд нет аналогичной команды для аккумулятора. Влияет на флаг Z.

Битовые команды

Обратите внимание: битовые операции над аккумулятором не предусмотрены.

BCF f,b

Сбрасывает бит номер b в ячейке памяти f в значение "0". Результат – в ячейке f. Не влияет на флаги.

BSF f,b

Устанавливает бит номер b в ячейке памяти f в значение "1". Результат – в ячейке f. Не влияет на флаги.

Управляющие команды

NOP

Пустая команда.

GOTO k

Безусловный переход на адрес k (адрес в команде кодируется 11 разрядами). Старшие биты адреса берутся из PCLATH (актуально при переходах межу 2k-словными страницами).
CALL k
Вызов процедуры по адресу k (адрес в команде кодируется 11 разрядами). Старшие биты адреса берутся из PCLATH (актуально при переходах межу 2k-словными страницами). Текущее значение счетчика команд помещается в стек. Стек программно недоступен и имеет кольцевую структуру, 8 ячеек.
RETURN
Возврат из процедуры. Значение счетчика команд восстанавливается из стека.
RETLW k
Возврат из процедуры с возвратом значения. Значение счетчика команд восстанавливается из стека. Дополнительно в аккумулятор заносится значение k. Обратите внимание – с помощью этой команды кодируются массивы констант в памяти программ.
RETFIE
Возврат из прерывания. Аналогично RETURN, но дополнительно разрешает прерывания путем занесения единицы в бит GIE.
SLEEP
Переводит процессор в спящий режим (см. документацию).
CLRWDT
Сбрасывает сторожевой таймер. Если при прошивке программы был установлен бит использования сторожевого таймера, то эту команду необходимо периодически вызывать для предотвращения сигнала сброса (см. документацию).

Команды организации условных переходов (условный пропуск)

BTFSC f,b
Если бит b в ячейке f сброшен (равен нулю), то команда, расположенная непосредственно за данной, не исполняется, а исполняется следующая. Часто следующая команда – GOTO, CALL или RETURN.
Для проверки состояния флагов используется, например, такая конструкция, использующая явное обращение к регистру флагов STATUS:
```
BTFSC STATUS, Z
GOTO metka ; переход, если флаг Z установлен, иначе эта команда
                игнорируется
```
BTFSS f,b
Если бит b в ячейке f установлен (равен единице), то команда, расположенная непосредственно за данной, не исполняется, а исполняется следующая. См. примечание выше.
INCFSZ f,d
Прибавляет единицу к содержимому ячейки f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Не влияет на флаги.

Если в результате операции получается нулевое значение, то команда, расположенная непосредственно за данной, не исполняется, а исполняется следующая. Данная команда иногда применяется для организации циклов со счетчиком.

DECFSZ f,d

Вычитает единицу из содержимого ячейки f, результат помещает в аккумулятор или в ячейку f в зависимости от флага d. Не влияет на флаги. Если в результате операции получается нулевое значение, то команда, расположенная непосредственно за данной, не исполняется, а исполняется следующая. Данная команда часто применяется для организации циклов со счетчиком.

Следующие примеры поясняют, каким образом в системе команд PIC может быть закодирован традиционный условный переход по "больше" или "меньше или равно":

1. Переход, если аккумулятор больше константы X

```
sublw x ; w=x-w
btfss STATUS,C ; перенос сброшен при отрицательном результате!
goto lbl
```

2. Переход, если аккумулятор меньше или равен ячейке по адресу Addr

```
subwf Addr,w ; w=[Addr]-w
btfsc STATUS,C ; перенос сброшен при отрицательном результате!
goto lbl
```

7.2.5. Особенности ассемблера MPASM

Язык ассемблера MPASM имеет ряд особенностей, которые делают его несколько нестандартным среди прочих ассемблерных языков микроконтроллеров. Эти особенности следующие.

В идентификаторах большие и малые буквы являются различимыми. Однако при записи мнемоник **команд и директив ассемблера** допустимо пользоваться и большими, и малыми буквами.

- Двоеточие после метки не ставится. Метка располагается на строке одна, команда в той же строке не пишется.
- В MPASM действует соглашение, согласно которому метки начинаются с начала строки, а мнемоники команд пишутся через пробелы или символы табуляции от начала строки.

Это требование не является жестким требованием синтаксиса (его можно нарушать), однако ассемблер выдает предупреждение, если встречает метку не в начале строки и/или код операции или директиву – в начале строки без ведущих пробелов.

- Комментарии записываются после точки с запятой.
- Десятичные константы записываются после символа точки, например

```
movlw .16 ; занести в аккумулятор число 16
```

□□ Шестнадцатеричные константы записываются с префиксом 0x, например

movlw 0x10 ; в аккумулятор заносится число 16=10h

□□ Двоичные константы записываются в апострофах и с префиксом b, напрмер

movlw b'00010000' ; в аккумулятор заносится число 16=10000b

Очень часто при кодировании алгоритмов управления в программе требуется задать какие-то данные в виде массивов, обращение к элементам которых строится по индексному принципу (по номеру элемента массива). В микроконтроллерах PIC16F74 данная задача сопряжена с существенными трудностями, так как память программ напрямую недоступна для считывания, а кроме того разрядность слова в памяти программ не равна 8 битам. Для решения проблемы используется команда RETLW, производящая возврат из процедуры с занесением в аккумулятор указанной константы, директива ассемблера DT, а также некоторая на первый взгляд нетривиальная программная конструкция.

Директива DT (Define Table), после которой через запятую перечисляются однобайтовые значения, позволяет разместить в последовательных словах памяти программ коды команды RETLW с указанными значениями в качестве аргументов.

Например, в ассемблере микроконтроллера семейства MCS51 программист написал бы конструкцию:

```
table DB 10,20,30,40
```

что привело бы к расположению в памяти программ четырех байтов со значениями 10, 20, 30 и 40, откуда они в дальнейшем могут быть считаны напрямую, например, с помощью конструкции:

```
mov DPTR, #table
movx A,@DPTR
```

В ассемблере PIC вместо этого записывается конструкция

```
Table DT .10,.20,.30,.40
```

которая на самом деле эквивалентна

```
table
RETLW .10
RETLW .20
RETLW .30
RETLW .40
```

Кроме того, для доступа к этой таблице следует написать специальную псевдопроцедуру, разместив ее в пределах одной 256-словной страницы памяти программ с таблицей table:

```
GetTable
ADDLW table
MOVWF PCL
```

Теперь, чтобы наконец прочитать в аккумулятор значение какого-то элемента таблицы, программист должен поместить номер элемента в аккумулятор и вызвать псевдопроцедуру по адресу GetTable:

MOVLW 2
CALL GetTable
В результате приведенной последовательности команд происходит следующее:
– в аккумулятор помещается двойка;
– происходит переход к метке GetTable с сохранением в стеке адреса возврата;
– к аккумулятору прибавляется адрес начала таблицы table, таким образом, в нем теперь содержится адрес команды RETLW .30, элемента номер 2 считая с нуля;
– в младший байт счетчика команд в явном виде заносится адрес команды RETLW .30, и осуществляется переход на нее в пределах 256-байтовой страницы памяти программ;
– команда RETLW .30 помещает в аккумулятор число 30 и возвращает управление в точку, следующую за вызовом процедуры GetTable.
Приведенный код, способный привести в ужас любого сторонника структурного подхода к программированию, иллюстрирует негативные побочные эффекты, способные возникнуть при использовании сокращеннного набора команд в RISC-процессорах.

Ассемблер семейства PIC16 позволяет объявлять константы с помощью традиционной директивы EQU, например
const1 EQU 0x55
Часто возникает необходимость объявить блок констант, числовые значения которых увеличиваются на единицу в порядке перечисления. Самый распространенный случай, когда возникает такая необходимость, это назначение адресов переменным в памяти данных.
Первые 20h адресов в каждой странице памяти данных – это регистры специальных функций. Следовательно, пользовательские данные могут быть размещены по адресам 20h..7Fh. Пусть в программе используется всего три переменные d1,d2 и d3, и пользователь желает разместить их по адресам 20h, 21h и 22h. Тогда это может быть закодировано средствами ассемблера PIC16 двояко:
d1 EQU 0x20
d2 EQU 0x21
d3 EQU 0x22
или
CBLOCK 0x20
d1,d2,d3
ENDC
Следует подчеркнуть, что директива CBLOCK прямого отношения к распределению ячеек в памяти данных не имеет, и может быть использована для назначения последовательных значений именованным константам для любых целей, определяемых кодируемым алгоритмом.

Лекция 15

8. Цифровая обработка сигналов

8.1. Общие сведения о цифровой обработке сигналов

В настоящее время во многих областях техники осуществляется замена аналоговых устройств на цифровые. В этом случае возникает необходимость преобразования амплитуд входных аналоговых величин, поступающих через определенные (чаще равные) временные промежутки, в число. Преобразование аналоговой величины в цифровую производится с помощью АЦП, и затем цифровая величина программным способом обрабатывается в компьютере. В качестве цифровой обработки может быть приведен пример выделения полезного сигнала из помех с помощью фильтра.

На рис.8.1.1 показана реализация аналогового фильтра с использованием операционного усилителя. Входной сигнал X(t), на который воздействуют помехи, поступает на вход фильтра. На выходе фильтра полезный сигнал Y(t) показан жирной линией.

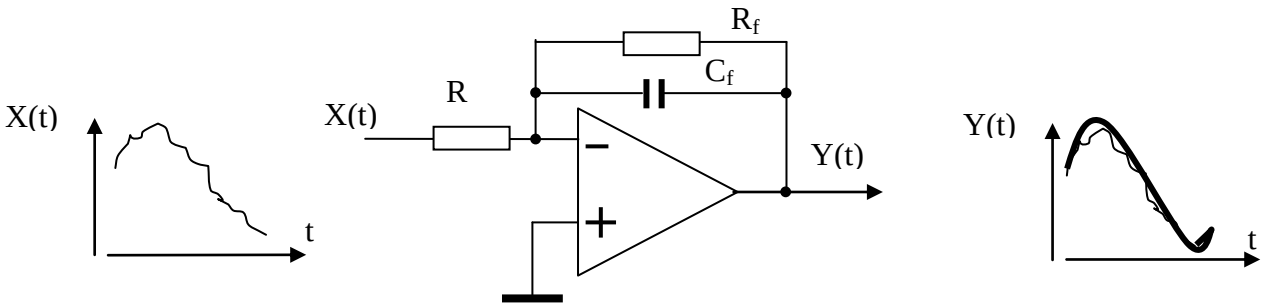


Рис. 8.1.1. Аналоговый фильтр

Амплитудно-частотная характеристика фильтра и его частота среза f_c определяются параметрами цепочки R_fC_f. На рис.8.1.2 показаны амплитудно-частотные характеристики идеального фильтра и реального фильтра, построенного на основе операционного усилителя.

Характеристики аналогового фильтра сильно зависят от внешних факторов, таких как изменение температуры, старения элементов фильтра. Кроме того, возникают дополнительные трудности, если потребуется изменить частотные характеристики фильтра. В этом случае надо установить емкость C_f и резистор R_f с другими параметрами.

Если применить теорему Котельникова, то аналоговая величина X(t) может быть представлена совокупностью дискретных отсчетов X(n), взятых через интервалы Δt.

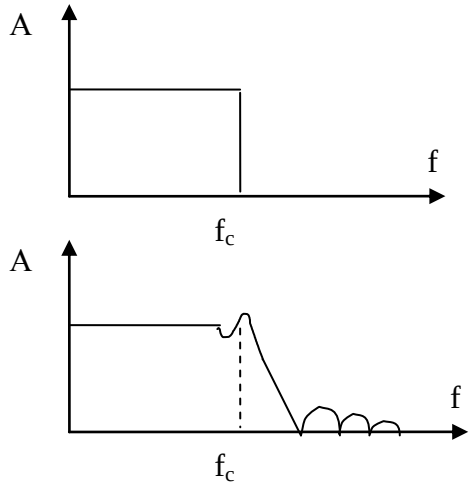


Рис.8.1.2. Амплитудно-частотная характеристика фильтра

Существует математическое описание фильтров низкой частоты (ФНЧ), с помощью которых может быть рассчитана любая амплитудно-частотная характеристика и построен цифровой фильтр с конечной импульсной характеристикой (КИХ). Фильтр КИХ описывается выражением:

$$Y(n) = \sum_{k=0}^{N} C(k)X(n-k),$$

где X(n-k) – дискретные отсчеты амплитуды аналоговой величины, C(k) – коэффициенты фильтра, заранее рассчитанные, Y(n) – значение дискретной величины на выходе фильтра.

Амплитудно-частотная характеристика фильтра определяется коэффициентами C(k). Качественные характеристики фильтрации определяются числом k.

На рис. 8.1.3 показана структурная схема цифрового фильтра с конечной импульсной характеристикой.

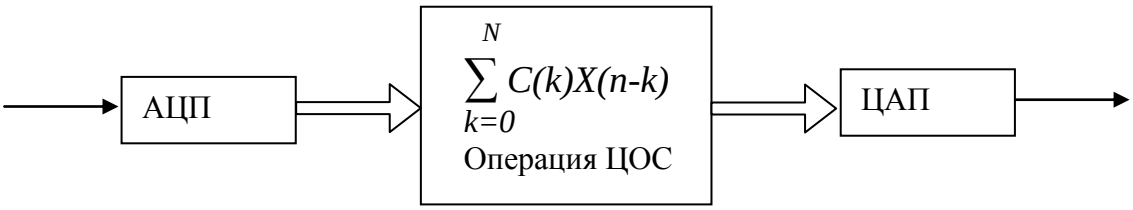


Рис. 8.1.3. Структурная схема цифрового фильтра

На рис.8.1.4 показан пример вычислительного устройства для реализации цифрового фильтра КИХ. В память устройства записываются две таблицы: таблица коэффициентов C(k) и таблица значений X(n-k), причем при каждом поступлении нового значения дискретного отсчета амплитуды аналоговой величины все предыдущие значения сдвигаются на один отсчет.

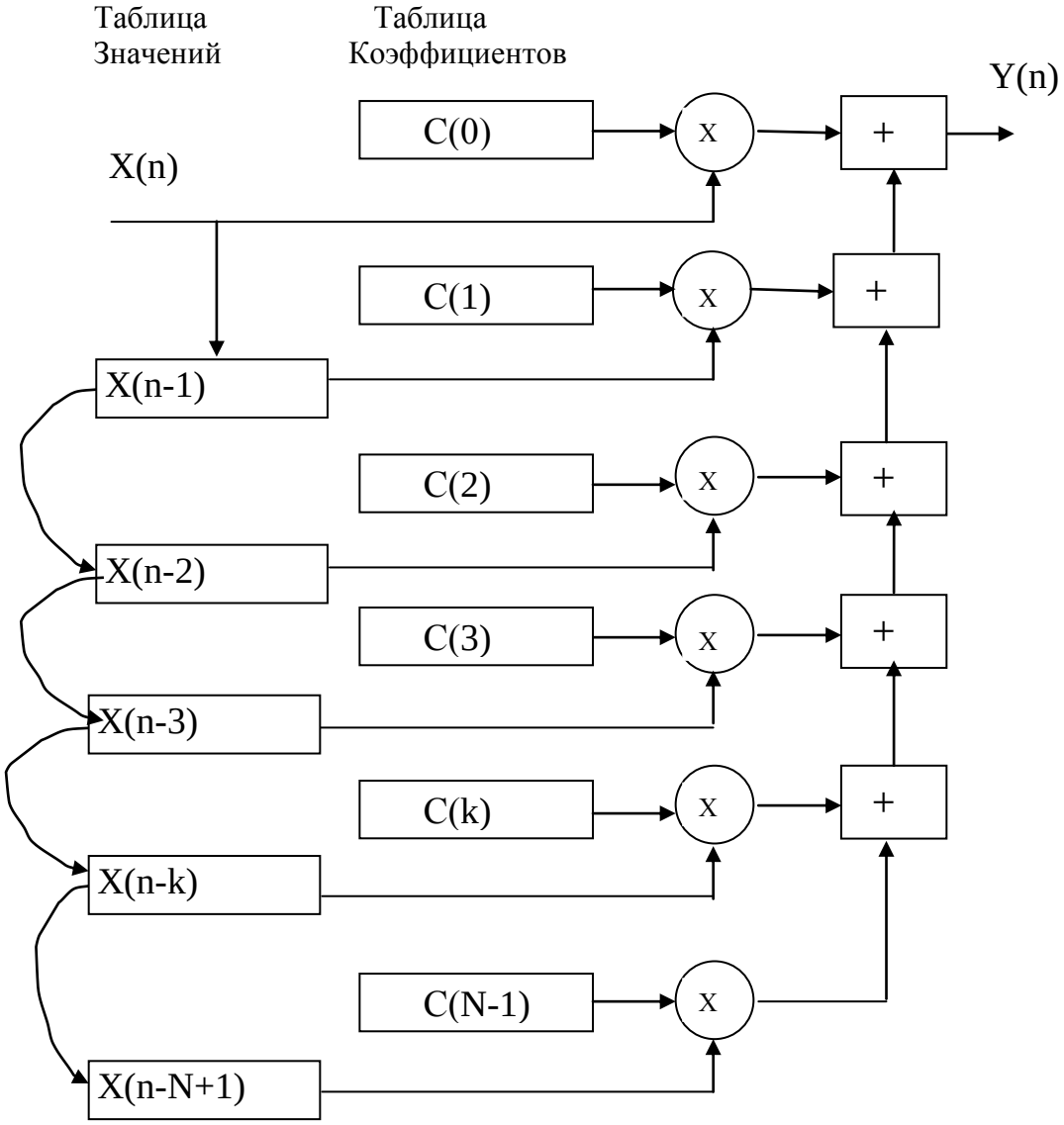


Рис. 8.1.4. Реализация цифрового фильтра КИХ

Самое старое значение при сдвиге теряется. Для вычисления значения Y(n) необходимо произвести N последовательных умножений и суммирования их результатов за короткое время Δt, если цифровая обработка сигнала осуществляется в реальном масштабе времени.

Процесс вычисления может быть разбит на операции умножения с накоплением, которые обозначаются аббревиатурой MAC (S=A*B+S).

Укажем еще два примера ЦОС, часто встречающихся на практике. Один пример относится к спектральному анализу с использованием преобразования Фурье. Анализатор спектра позволяет определить зависимости параметров входного сигнала от частот, т.е. любой сигнал может быть представлен в частотной области как совокупность графиков в координатах частота –амплитуда ($X(f)$). Дискретное преобразование Фурье в частотной области задается формулой:

$$X(f) = \Delta t \sum_{n=-\infty}^{+\infty} x(n\Delta t) e^{-j2\pi f n \Delta t},$$

где $x(nT)$ дискретные значения отсчетов сигнала, взятых через интервалы Δt. Для сокращения объема вычислений используют алгоритмы быстрого преобразования Фурье (БПФ), в которых число временных отсчетов n ограничивают в зависимости от точности вычислений. Для реализации дискретного преобразования Фурье кроме операции MAC требуется быстрое вычисление значений функции типа $y=exp(x)$.

Второй пример относится к задаче управления скоростью электродвигателя. В этом случае может быть использован цифровой регулятор, в основу которого положена модель пропорционально-интегрально-дифференциального регулятора (ПИД). Управляющее воздействие $u(k)$ ПИД регулятора описывается выражением:

$$u(k) = u(k-1) + \Delta u(k),$$

где к предыдущему значению управляющего воздействия $u(k-1)$ добавляется приращение $\Delta u(k)$, рассчитанное по формуле:

$$\Delta u(k) = c_0 u(k) - c_1 u(k-1) + c_2 u(k-2),$$

где c_0, c_1, c_2 – коэффициенты, рассчитанные заранее. В выражении для вычисления управляющего воздействия $u(k)$ присутствуют операции умножения с накоплением, причем их также необходимо выполнять в реальном масштабе времени.

Анализ приведенных примеров показывает, что цифровая обработка сигналов ЦОС должна выполняться в реальном масштабе времени с заданной точностью при ограничении времени обработки.

Преимущества цифровых методов обработки сигналов по сравнению с аналоговыми состоит в расширении используемых математических операций, усложнения алгоритмов обработки и повышения точности вычислений. К недостаткам ЦОС следует отнести нехватку времени

обработки для некоторых приложений. Для преодоления указанного недостатка многими производителями вычислительной техники разработан ряд специализированных процессоров с высокой производительностью. Такие процессоры называют сигнальными (DSP).

Лекция 16

8.2. Сигнальные микропроцессоры

Для повышения производительности сигнальных процессоров широко используются все преимущества архитектуры RISC. Большинство команд выполняются за один машинный такт. Для сокращения времени выполнения команд применяются аппаратные способы вычисления некоторых математических операций: умножение, сдвиг влево или вправо на несколько разрядов, вычисление экспоненциальных функций, операции над комплексными числами. Для большей эффективности выполнения математических операций в состав операционного устройства включают несколько АЛУ. В систему команд закладывают специализированные операции. Например, в команде умножения с накоплением MAC может быть указано правило изменения адресов ячеек памяти, где хранятся операнды массивов A и B, и число циклов выполнения команды MAC. Реализация указанных решений позволяет использовать сравнительно низкие тактовые частоты работы сигнальных процессоров, что облегчает их применение в промышленных условиях.

Сигнальные процессоры производятся многими компаниями. Сложившееся к настоящему времени распределение рынка между ведущими поставщиками (табл. 8.2.1) показывает, что 4 компании, стоящие в начале списка, поставляют более 80% всех используемых в мире DSP. Именно эти компании наиболее известны и на российском рынке, и их продукция часто упоминается.

Таблица 8.2.1

Основные производители DSP и принадлежащие им доли рынка

№ п/п	Наименование компании	Доля рынка DSP
1	Texas Instruments	54,3%
2	Freescale Semiconductor	14,1%
3	Analog Devices	8,0%
4	Philips Semiconductors	7,5%
5	Agere Systems	7,3%
6	Toshiba	4,9%
7	DSP Group	2,2%
8	NEC Electronics	0,6%
9	Fujitsu	0,4%

10	Intersil	0,3%
11	Другие	0,5%

Из таблицы 8.2.1 видно, что в настоящее время наибольшее распространение получили сигнальные процессоры компаний Analog Device, Texas Instruments, Freescale. Сигнальные процессоры компании Analog Devices удобны для приложений, требующих выполнения больших объемов математических вычислений (таких как цифровая фильтрация сигнала, вычисление корреляционных функций и т.п.), поскольку их производительность на подобных задачах выше, чем у процессоров других компаний. В то же время для задач, требующих выполнения интенсивного обмена с внешними устройствами, предпочтительнее использовать процессоры Texas Instruments, обладающие высокоскоростными встроенными интерфейсными устройствами. Сигнальные процессоры компании и Freescale более дешевы, и используются в промышленных роботах, бытовых приборах, средствах беспроводной связи.

Перечислим основные характеристики сигнальных процессоров. Формат данных и разрядность – одна из основных характеристик цифровых сигнальных процессоров. В сигнальных процессорах используется формат с фиксированной точкой (целые числа со знаком) либо формат с плавающей точкой. Разрядность данных 16 или 32 бита.

На производительность процессора DSP влияет организация внутренней памяти. Это связано с тем, что ключевые команды DSP являются многооперандными, и для уменьшения времени их выполнения требуется одновременное чтение нескольких ячеек памяти. Например, команда MAC требует одновременного чтения 2 операндов и самой команды для того, чтобы ее можно было выполнить за 1 такт. Это достигается различными методами, среди которых применение многопортовой памяти, разделение на память программ и память данных (Гарвардская архитектура), использование кэша команд и т.д. DSP-процессоры широко используются в мобильных устройствах, где потребление мощности является основной характеристикой.

Для снижения энергопотребления используется множество методов, в том числе уменьшение напряжения питания и введение функций управления потреблением, например, динамического изменения тактовой частоты, переключения в спящий или дежурный режим или отключения неиспользуемой в данный момент периферии.

Большое значение для микропроцессорных систем приобретает возможность отладки и коррекции программ в готовом устройстве. Почти все современные процессоры DSP поддерживают внутрисхемную эмуляцию в соответствии со стандартом IEEE 1149.1 JTAG. При использовании технологии JTAG осуществляется переход от эмуляции процессора внешним устройством к непосредственному контролю над процессором при

выполнении программы, что позволяет значительно увеличить степень соответствия макета реальному устройству и, следовательно, повысить надежность процесса отладки.

8.3. Сигнальные процессоры фирмы Texas Instruments

Сигнальные процессоры компания Texas Instruments производит с 1982 года. К наиболее распространенным относятся процессоры семейства TMS320, которые разделяются на процессоры с фиксированной точкой и на процессоры с плавающей точкой. К процессорам с фиксированной точкой относятся семейства TMS320C2xxx, TMS320C5xxx, TMS320C6xxx. К процессорам с плавающей точкой относятся семейства TMS320C3xxx, TMS320C4xxx, TMS320C8xxx. Каждое семейство содержит высокопроизводительное ядро центрального процессорного устройства (CPU), встроенную комбинированную память и широкий набор встроенных периферийных устройств.

Обладая производительностью до 40 MIPS, 16-битные контроллеры семейства C24x позволяют реализовывать различные алгоритмы управления. Набор однотактовых инструкций обеспечивает быстрое выполнение сложных математических вычислений в режиме реального времени, а гарвардская архитектура имеет ряд удобств при использовании векторной математики, часто используемой в задачах промышленной автоматизации. Модернизированная гарвардская архитектура контроллеров C24x обеспечивает максимальную скорость обработки данных благодаря наличию раздельных шин для программы и данных, позволяя одновременно читать данные и программные инструкции. Передача данных между двумя пространствами поддерживается программно. Структурная схема сигнального процессора TMS320C24x показана на рис.8.2.5.

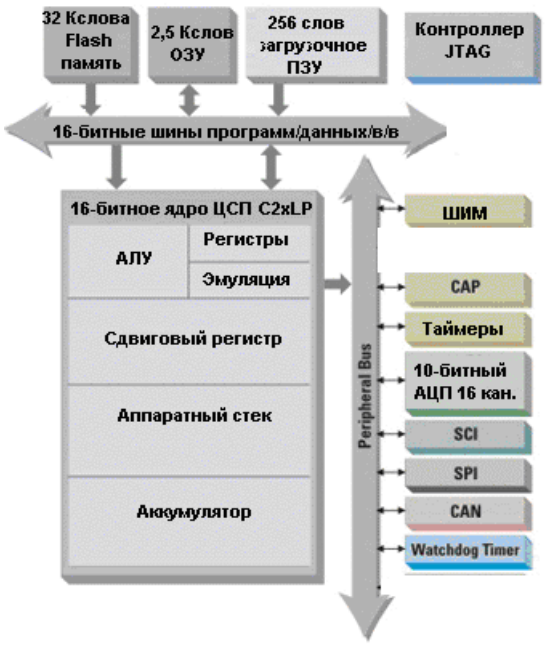


Рис. 8.2.5. Структурная схема сигнального процессора TMS320C24x

В процессор встроена высокопроизводительная память ОЗУ (DRAM), в которой осуществляются две операции за один цикл. Память программ внутрисхемно перепрограммируемая (Flash-память). С этой целью в ПЗУ зашита загрузочная программа.

Семейство процессоров TMS320C2000 обладает высокой степенью интеграции периферийных устройств.

Три устройства ШИМ (PWM), CAP, Таймеры выполняют роль модуля обработчиков событий (PCA – массив программируемых счетчиков):

до 16 выходов ШИМ;
до четырех входов захвата состояния таймера (CAP);
до шести 32-разрядных/шести 16-разрядных таймеров.

В состав процессора входит 10-разрядный 16-канальный АЦП. Время преобразования 375 нс, запуск начала преобразования по внутренним или по внешним сигналам. Внутренний или внешний источник опорного напряжения.

В состав последовательных интерфейсов входят:

многоканальный буферизованный последовательный порт (McBSP), котрый включает модули синхронных SCI и асинхронных UART приемников передатчиков;
до 4 модулей последовательных периферийных интерфейсов SPI;
до двух модулей интерфейса CAN, версии 2.0B;
шина I^2C (версия 2.1).

В состав семейства TMS320C5000 входит цифровой сигнальный процессор TMS320C54x, который является на сегодняшний день одним из самых популярных в мире, обладая полноценной линейкой из 15

разнообразных устройств с диапазоном производительности от 30 до 532 MIPS. Передовая архитектура и набор инструкций семейства C54x обеспечивает исключительно малый объем кода для типовых задач ЦОС, позволяя использовать встроенное ОЗУ с максимальной эффективностью. Широкие возможности цифровых сигнальных процессоров позволяют использовать их в мобильных устройствах для подключения к сети Internet, высокоскоростной радиосвязи и многих других.

Семейство 16-битных цифровых сигнальных процессоров с фиксированной точкой C5000 обладает широчайшим выбором требуемых разработчикам встроенных периферийных устройств (рис.8.2.6).

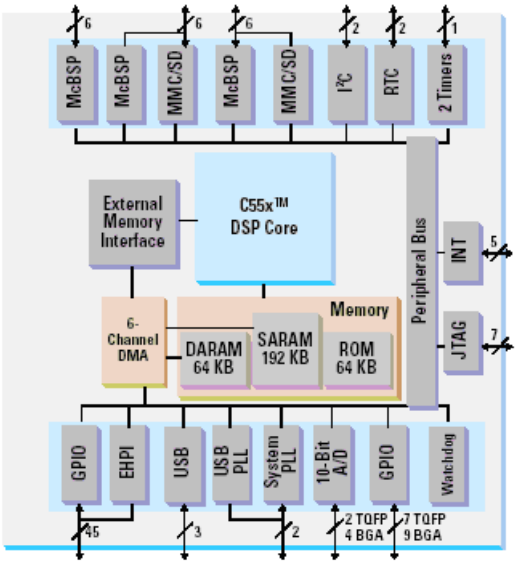


Рис. 8.2.6. Структурная схема процессоров семейства TMS320C5000

Ориентированное на высокоэффективные портативные устройства с батарейным питанием, семейство процессоров C5000 обеспечивает высокую гибкость при разработке микропроцессорных систем.

В состав периферийных устройств входят:

универсальный последовательный интерфейс USB 2.0 Full-Speed;
последовательны интерфейс UART;
последовательный интерфейс Inter Integrated-Circuit (I2C);
аналогово-цифровой преобразователь (АЦП)
поддержка карт Multimedia Card / Secure Digital (MMC/SD)
аппаратная поддержка видео
многоканальные буферизованные последовательные порты (McBSP)
непосредственный доступ к памяти (DMA)
параллельный 8/16-битный интерфейс Enhanced Host Port Interface (EHPI)

Сигнальные процессоры семейства TMS320C6000 применяются в медицине, промышленных приложениях, системах многоканальной телефонии, домашних охранных системах, цифровой обработке изображений, 3D-графике и распознавании голоса

Процессоры выполнены на основе архитектуры VelociTI очень длинного слова инструкции (VLIW). На тактовой частоте 200 МГц процессоры C62x позволяют развить производительность 1600 миллионов инструкций в секунду, что делает их эффективным инструментом для решения сложных задач цифровой обработки. Процессоры C62x обладают операционной гибкостью высокоскоростных контроллеров и вычислительными способностями матричных процессоров. Процессор содержит 32 32-разрядных регистра общего назначения и 8 независимых функциональных блоков. В состав этих 8 блоков входят: 6 арифметико-логических устройств (АЛУ) для достижения высокой степени параллелизма и два 16-разрядных умножающих устройства с формированием 32-разрядного результата. Процессоры могут выполнить за один цикл два умножения-накопления (MAC), что позволяет достичь общей производительности 466 млн. MAC в секунду. Процессор также содержат специфическую аппаратную логику, встроенную память и дополнительные периферийные устройства. Процессор содержат большой банк встроенной памяти и содержат мощный и разнообразный набор периферийных устройств, в том числе многоканальные буферизованные последовательные порты (McBSP), таймеры общего назначения, параллельный интерфейс хост-порта (TMS320C6201, TMS320C6211), 32-разрядную шину расширения (6203, 6204), интерфейс PCI (TMS320C6205) и интерфейс внешней памяти для непосредственного подключения к SDRAM или SBSRAM, а также асинхронной памяти. Процессор C62x поддерживается полным набором инструментальных средств для проектирования, в том числе: новый Си-компилятор, оптимизатор ассемблирования для упрощения программирования и распределения процессорного времени, а также Windows-интерфейс отладчика для визуализации выполнения программного кода.

Лекция 17

9. Комплексная отладка МПС

Особенность написания и отладки программного обеспечения для однокристальных микро-ЭВМ (микроконтроллеров) и процессоров цифровой обработки сигналов (ЦОС) состоит в том, что для этого, как правило, совершенно недостаточно иметь системы, состоящей только из программируемого микроконтроллера или процессора ЦОС.

Все это заставляет при разработке программ для микроконтроллеров использовать специальные средства, называемыми инструментальными средствами разработки и отладки. Инструментальные средства отладки микроконтроллеров предназначены для повышения эффективности труда разработчика.

Существуют различные варианты построения инструментальных средств разработки и отладки. К ним относятся:

— Внутрисхемные эмуляторы (ВСЭ);
— Программные симуляторы; •
— Платы разработки и отладки (ПРО);
— Мониторы отладки; •
– Эмуляторы ПЗУ.

В настоящее время проектирование различных цифровых устройств на базе микропроцессоров и микроконтроллеров невозможно представить без применения разнообразных программных и аппаратно-программных отладочных средств. К наиболее развитым аппаратно-программным средствам комплексной отладки относятся внутрисхемные эмуляторы. Внутрисхемные эмуляторы подключаются к отлаживаемой или тестируемой системе вместо целевого микропроцессора или микроконтроллера и позволяют гибко управлять поведением системы на протяжении процесса отладки, собирать данные о состоянии ее различных объектов, выполнять программы пользователя в различных режимах: в режиме реального времени (непрерывное выполнение программы с заданного адреса), в пошаговом режиме, в режиме с остановками функционирования по заданному условию. Зачастую они позволяют эмулировать не только целевой процессор, но и память, тактовый генератор, устройства ввода - вывода. Обычно в самом начале отладка ведется с применением узлов эмулятора, функционирующих вместо блоков отлаживаемой системы. В процессе отладки происходит постепенная замена узлов внутрисхемного эмулятора на соответствующие блоки устройства пользователя.

Таким образом, внутрисхемные эмуляторы представляют собой весьма мощные средства для комплексной отладки микропроцессорных систем.

Внутрисхемный эмулятор содержит следующие функциональные блоки:

1. Отладчик;
2. Блок эмуляции микроконтроллера;
3. Эмуляционная память;
4. Процессор точек останова;
5. Трассировщик;
6. Анализатор эффективности программного кода;
7. Таймер реального времени;
8. Программно-аппаратные средства, обеспечивающие возможность чтения и модификации ресурсов эмулируемого процессора в реальном масштабе времени.

Функционально внутрисхемные эмуляторы делятся на стыкуемые с ПЭВМ и функционирующие автономно. Автономные внутрисхемные эмуляторы имеют индивидуальные вычислительные ресурсы, средства ввода-вывода

Внутрисхемный эмулятор стыкуется с отлаживаемой системой при помощи специальной эмуляционной головки. Эмуляционная головка вставляется вместо микроконтроллера в отлаживаемую систему.
Внутрисхемный эмулятор точно воспроизводит электрические и временные характеристики работы микроконтроллера, заменив отлаживаемый микроконтроллер на внутрисхемный эмулятор

Симулятор – программное средство, способное имитировать работу микроконтроллера и его памяти. Как правило, симулятор содержит в своем составе отладчик и модель ЦПУ и памяти.
Более продвинутые симуляторы содержат в своём составе модели встроенных периферийных устройств, таких, как таймеры, порты, АЦП и системы прерываний.
Симулятор должен уметь загружать файлы программ во всех популярных форматах, максимально полно отображать информацию о состоянии ресурсов симулируемого микроконтроллера, а также предоставлять возможности по симуляции выполнения загруженной программы в различных режимах. В процессе отладки модель "выполняет" программу, и на экране компьютера отображается текущее состояние модели.

Загрузив программу в симулятор, пользователь имеет возможность запускать её в пошаговом или непрерывном режимах, задавать условные или безусловные точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров симулируемого микропроцессора. С помощью симулятора можно быстро проверить логику выполнения программы, правильность выполнения арифметических операций.
В зависимости от класса используемого отладчика различные симуляторы могут поддерживать высокоуровневую символьную отладку программ.
Некоторые модели симуляторов могут содержать ряд дополнительных программных средств, таких, например, как интерфейс внешней среды, встроенную интегрированную среду разработки.
В реальной системе микроконтроллер обычно занимается считыванием информации с подключенных внешних устройств (датчиков), обработкой этой информации и выдачей управляющих воздействий на исполнительные устройства.
Чтобы в симуляторе, не обладающем интерфейсом внешней среды, смоделировать работу датчика, нужно вручную изменять текущее состояние модели периферийного устройства, к которому в реальной системе подключён датчик. Если, например, при приёме байта через последовательный порт взводится некоторый флажок, а сам байт попадает в определённый регистр, то оба эти действия нужно производить в таком симуляторе вручную. Наличие же интерфейса внешней среды позволяет пользователю создавать и гибко использовать модель внешней среды микроконтроллера, функционирующую и взаимодействующую с отлаживаемой программой по заданному алгоритму.

Платы разработки и отладки (ПРО) являются конструкторами для макетирования прикладных систем. В последнее время при выпуске новой

модели микроконтроллера фирма-производитель обязательно выпускает и соответствующую плату ПРО. Обычно это печатная плата с установленным на ней микроконтроллером плюс вся необходимая ему стандартная обвязка. На этой плате также устанавливают схемы связи с внешним компьютером. Как правило, там же имеется свободное поле для монтажа прикладных схем пользователя. Иногда имеется уже готовая разводка для установки дополнительных устройств, рекомендуемых фирмой. Например, ПЗУ, ОЗУ, ЖКИ-дисплей, клавиатура, АЦП и др. Кроме учебных или макетных целей, такие доработанные пользователем платы используются в качестве одноплатных контроллеров, встраиваемых в мало серийную продукцию.

Для большего удобства платы ПРО комплектуются ещё и простейшим средством отладки на базе монитора отладки. Однако здесь проявились два разных подхода: один используется для микроконтроллеров, имеющих внешнюю шину, а второй – для микроконтроллеров, не имеющих внешней шины.

В первом случае отладочный монитор поставляется фирмой в виде микросхемы ПЗУ, которая вставляется в специальную розетку на плате ПРО. Плата также имеет ОЗУ для программ пользователя и канал связи с внешним компьютером или терминалом.

Во втором случае плата ПРО имеет встроенные схемы программирования внутреннего ПЗУ микроконтроллера, которые управляются от внешнего компьютера. В этом случае программа монитора просто заносится в ПЗУ микроконтроллера совместно с прикладными кодами пользователя. Прикладная программа при этом специально должна быть подготовлена: в нужные её места вставляют вызовы отладочных подпрограмм монитора. Затем осуществляется пробный прогон. Чтобы внести в программу исправления пользователю надо стереть ПЗУ и произвести повторную запись. Готовую прикладную программу получают из отлаженной путём удаления всех вызовов мониторных функций и самого монитора отладки. Примерами могут служить платы ПРО фирмы Microchip для своих PIC контроллеров. Такой же принцип и у плат для отладки микроконтроллеров 80C750 Philips или 89C2051 Atmel.

Важно отметить, что плюс к монитору, иногда платы ПРО комплектуются ещё и программами отладки, которые запускаются на внешнем компьютере в связке с монитором. Эти программы в последнее время заметно усложнились и зачастую имеют высокопрофессиональный набор отладочных функций, например отладчик- симулятор, или различные элементы, присущие в чистом виде интегрированным средам разработки. В состав поставляемых комплектов могут входить и программы прикладного характера, наиболее часто встречающиеся на практике.

Эмулятор ПЗУ – программно-аппаратное средство, позволяющее замещать ПЗУ на отлаживаемой плате и подставляющее вместо него ОЗУ, в которое может быть загружена программа с компьютера через один из стандартных каналов связи. Это устройство позволяет пользователю избежать многократных циклов перепрограммирования ПЗУ. Эмулятор ПЗУ

имеет смысл только для микроконтроллеров, которые в состоянии обращаться к внешней памяти программ. Эмулятор ПЗУ может работать с любыми типами микроконтроллеров.
Сейчас появились модели интеллектуальных эмуляторов ПЗУ, которые позволяют контролировать состояние внутренних элементов микроконтроллера на плате пользователя и вообще, по управлению отладкой, стали похожими на внутрисхемный эмулятор.

Интеллектуальные эмуляторы ПЗУ представляют собой гибрид из обычного эмулятора ПЗУ, монитора отладки и схем быстрого переключения шины от эмулятора к монитору и наоборот. Этим создаётся эффект, как если бы монитор отладки был установлен на плате пользователя.
Таким образом, ещё каких-нибудь 15 – 20 лет назад наиболее распространённым способом создания макета (прототипа) будущего устройства был "живой": подобрав электронные компоненты, разработчик брал в руки паяльник и собирал на макетных платах отдельные узлы или устройство в целом. Затем начинался процесс отладки: исправление ошибок принципиальной схемы, установка режимов работы, уточнение параметров применяемых компонентов и т.д.
Этот вариант не потерял своей актуальности и по сей день, но применяется сейчас в модифицированном виде:по блок-схеме прибора с учётом планируемых технических характеристик выбирается микроконтроллер;
выбирается отладочная плата для данного микроконтроллера;
на основе анализа аналоговой части устройства принимается решение либо о полном её макетировании, либо использовании подходящих модулей с макетированием схемы их объединения и схем дополнительных устройств, входящих в прибор. Этот этап макетирования выполняется "в живую".

К преимуществам такого способа разработки можно отнести: уменьшение времени выхода готовой продукции; уменьшение материальных затрат и риска при разработке; использование собственных ресурсов для ускорения разработки; свободное использование собственных разработок в дальнейшем.