ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
“ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Лабораторная работа №3

Интерфейсы микропроцессорных систем

Учебное пособие к выполнению лабораторной работы по дисциплине «Микропроцессорная техника» и курсового проекта по дисциплине «Электроника и микроэлектроника» для студентов ФТФ специальности 140306

Томск – 2004

УДК 681.322

Горюнов А.Г. Ливенцов С.Н. Интерфейсы микропроцессорных систем: Учеб. пособие.

Учебное пособие посвящено вопросам организации интерфейсов ввода-вывода встраиваемых микропроцессорных систем, а также особенностям программирования микроконтроллеров на языке СИ. Данное учебное пособие ориентировано на курс лабораторных работ с использованием учебно-лабораторных стендов SDK-1-1 и содержит примеры программ работы с периферией данного стенда.

Пособие подготовлено на кафедре «Электроника и автоматика физических установок» ТПУ и предназначена для студентов очного обучения специальности 200600.

Учебное пособие рассмотрено и рекомендовано
к изданию методическим семинаром кафедры электроники и автоматики
физических установок "___"_____ 2004г.

Зав. кафедрой
доцент, к.т.н. _____ В. Ф. Дядик

Содержание

1 Цель работы

Целью работы является изучение интерфейсов периферийных устройств и полевых сетей встраиваемых микропроцессорных систем, знакомство с их протоколами и особенностями практического использования.

2 Содержание работы

1. Изучение следующих вопросов организации интерфейсов

Параллельная передача данных [1].
Последовательная передача данных [1]
Синхронный последовательный интерфейс.
Асинхронный последовательный интерфейс.
Способы обмена информацией в микропроцессорной системе [1].
Последовательные интерфейсы встраиваемых микропроцессорных систем
Классификация и терминология.
Последовательный интерфейс RS-232.
Последовательный периферийный интерфейс SPI.
Синхронный последовательный интерфейс I^2C.
Протоколы нижнего уровня CAN.

2. Практическая часть лабораторной работы

Анализ реализации интерфейсов на примерах.
Выполнение индивидуального задания.

3. Оформление отчёта.

3 Последовательные интерфейсы встраиваемых микропроцессорных систем

3.1 Классификация и терминология

По определению интерфейс [2] представляет собой совокупность унифицированных аппаратных, программных, конструктивных средств, необходимых для реализации алгоритмов взаимодействия различных функциональных блоков МП-систем, а также функциональных блоков, входящих в состав автоматизированных систем управления.

Стандартизации в интерфейсе подлежат состав и тип линий связи, электрические и временные параметры сигналов, форматы передаваемой информации, команды и состояния, алгоритмы функционирования, конструктивное исполнение соединений.

Интерфейсы ЭВМ и средств промышленной автоматизации по функциональному назначению подразделяют на локальные, мезонинные, системные, интерфейсы периферийных устройств, приборные интерфейсы, интерфейсы локальных вычислительных сетей.

Встроенные в 8-разрядные МК модули последовательных приемопередатчиков в основном используются для реализации **интерфейсов периферийных устройств, приборных интерфейсов и интерфейсов локальных вычислительных сетей**. Понятие "интерфейс" пришло в область встраиваемых МП-систем управления из вычислительной техники, поэтому определения различных типов интерфейсов также даются в применении к ЭВМ.

Интерфейс периферийных устройств служит для подключения к системному интерфейсу ЭВМ различных по принципу действия периферийных устройств (накопители на жестких магнитных дисках, принтеры, сканеры, клавиатура и др.), каждое из которых имеет специфичный приборный интерфейс. Примеры параллельных интерфейсов периферийных устройств: Centronics, IEEE-48B. Хорошо известны последовательные интерфейсы периферийных устройств RS-232C, RS-422A, RS-485.

Под **приборным интерфейсом** понимают совокупность неунифицированных сигналов, которая обеспечивает обмен информацией и управление некоторым конкретным прибором. Функциональное назначение интерфейса периферийных устройств и приборного интерфейса одно и то же: связь ЭВМ с устройством ввода информации или с объектом управления. Но в первом случае эта связь осуществляется на основе уже стандартного решения, а во втором – произвольно выбранного разработчиком.

В приложении к встраиваемым МП-системам интерфейс периферийных устройств выступает в двух ипостасях. Первая из них – МК в составе встраиваемой МП-системы управления периферийным устройством. В этом случае МК как раз и выполняет функцию преобразования потока данных в стандарте какого-либо периферийного интерфейса к неунифицированным сигналам приборного интерфейса. Примером может служить клавиатура и манипулятор "мышка" персональных компьютеров. Однако системы подобного типа не ограничиваются компьютерной периферией. Так, многие современные средства измерения, способные работать в автономном режиме, предусматривают возможность связи с компьютером для передачи результатов измерения с целью обработки полученной информации и ее документирования. Причем МК, который обеспечивает обмен по последовательному каналу, не обязательно должен выполнять также функции управления измерительным прибором. Так, осциллограф смешанных сигналов HP54645D фирмы Hewlett Packard имеет в своем составе мультипроцессорную систему управления с достаточно большим объемом памяти. Поэтому он способен запоминать некоторую группу измерений и воспроизводить на экране электронно-лучевой трубки ранее полученную осциллограмму в измененном масштабе времени. Но этот осциллограф имеет также специальную навесную карту сопряжения с персональным компьютером в стандарте RS-232C, которая обеспечивает возможность переноса данных.

Еще один пример: система мониторинга электропривода насосов глубинных скважин. Встроенный в блок управления двигателем МК снимает показания с датчиков скорости вращения, тока, напряжения и передает их по последовательному интерфейсу в удаленный на несколько сот метров компьютер. Подобных примеров можно привести великое множество. Двухсторонний обмен информацией с персональным компьютером по последовательному интерфейсу (постоянному или коммутируемому) – типовая функция МК встраиваемой МП-системы управления.

Вторая ипостась интерфейса периферийных устройств в системах с 8-разрядными МК – связь МК с другими ИС платы встраиваемой МП-системы (рис. 1). Несмотря на то, что современные МК имеют в своем составе все основные компоненты системы обработки информации, технические характеристики встроенных модулей МК не всегда удовлетворяют условиям задачи. Один из наиболее ярких примеров – необходимость включения в состав устройства энергонезависимой памяти данных объемом от 64 Кбайт и выше. В качестве такой памяти не может быть использовано резидентное ПЗУ 8-разрядных МК, т.к. МК с таким объемом ПЗУ не существует. Следовательно, должна быть использована внешняя ИС памяти. Но 8-разрядные МК имеют закрытую архитектуру. Поэтому традиционное решение с использованием магистралей адреса и данных не может быть использовано для сопряжения МК со схемой памяти. Однако выход из положения был найден: память снабдили специальной схемой управления, которая способна принимать данные в последовательном коде, преобразовывать их в параллельный код, а затем производить операцию записи принятых данных в ячейки памяти с заданными адресами. В режиме считывания эта же схема управления преобразует параллельный код в последовательный и передает содержимое ячейки памяти в МК. Скорость обмена в последовательном коде между МК и периферийной ИС чрезвычайно высока – до 10 Мбит/с, поэтому рассматриваемое решение оказалось приемлемым с точки зрения быстродействия и очень привлекательным с точки зрения минимизации габаритов МП-системы. Число линий связи сокращается, и можно отойти от многослойной технологии при сохранении минимальной площади платы: очень выгодное решение для дешевых систем, коими системы с 8-разрядными МК и являются. Предложенный способ сопряжения оказался столь удачным, что его стали использовать в других периферийных ИС, таких, как АЦП, ЦАП, часы реального времени и т.д. Сопряжение с периферийными ИС по высокоскоростному последовательному интерфейсу стало основой схемотехники встраиваемых МП-систем на базе 8-разрядных МК. Последовательные интерфейсы, используемые для связи ИС на плате, стандартизировали. В настоящее время наиболее популярными являются последовательный синхронный интерфейс SPI (Serial Peripheral Interface), предложенный компанией Motorola, и двухпроводной интерфейс I2C (Inter-Integrated Circuit), разработанный компанией Philips. Появился новый класс ИС периферийных устройств с последовательным выходом, которые поддерживают протоколы SPI и I2C. По определению такие интерфейсы тоже должны быть отнесены к интерфейсам периферийных устройств, но если раньше МК выступал в роли части периферийного устройства, то теперь он превратился в маленькую ЭВМ, которая связывается уже со своими периферийными устройствами, используя стандартный периферийный интерфейс. Желая подчеркнуть отличие периферийного интерфейса первого типа от второго, последний часто определяют как "внутриплатный", обращая тем самым внимание на то, что связь осуществляется на плате встраиваемой МП-системы. Аналогичные решения используются также для связи МК с панелью индикатора или пультом управления, которые конструктивно выполнены на другой плате. Не все периферийные ИС, особенно это относится к датчикам физических величин с последовательным выходом, поддерживают протоколы SPI и I2C. Тогда, в соответствии с классификацией интерфейсов, интерфейс связи с такими ИС следует характеризовать как приборный. С технической точки зрения такие ИС могут потребовать использования метода программной эмуляции контроллера связи вместо применения модуля встроенного

последовательного приемопередатчика. Однако сущность не изменится: такой способ сопряжения обеспечивает малое число проводов связи или малое число дорожек платы. Справедливости ради, следует отметить, что **приборные интерфейсы с наиболее удачными протоколами как раз и становятся впоследствии периферийными.** Сопряжение по последовательному интерфейсу имеет и еще одно преимущество: возможность дешевой **потенциальной развязки, т.к. число каналов невелико.**

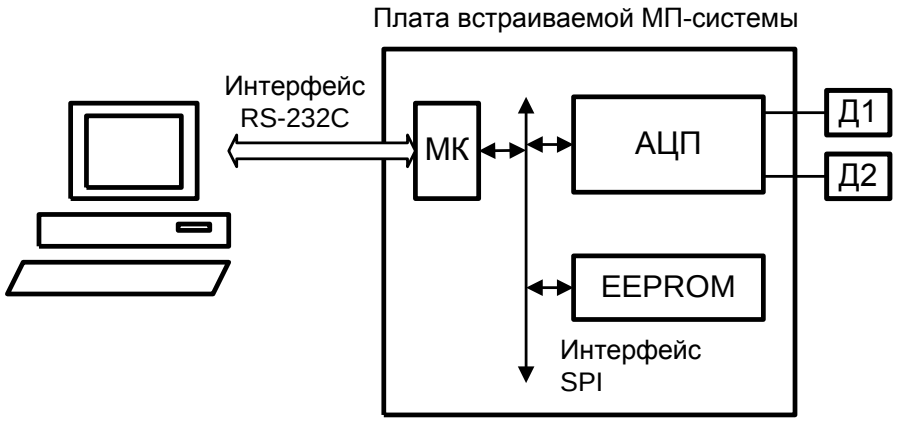


Рис. 1. Пример системы сбора данных.

Интерфейс ЛВС – интерфейс локальной вычислительной сети – используется для включения встраиваемой МП-системы в локальную вычислительную сеть, которая представляет собой систему рабочих станций на базе персональных компьютеров и программируемых контроллеров, связанных между собой каналами передачи данных и территориально расположенных, как правило, в пределах одного здания. Наиболее близок разработчику встраиваемых МП-систем пример локальной вычислительной сети на основе нескольких МП-контроллеров. Например, ЛВС овощехранилища, где в каждом помещении управление режимом хранения продуктов ведется автоматизированным способом при помощи программируемого контроллера. Режим хранения задается с персонального компьютера оператора, об отклонениях климата в помещении контроллер сообщает на пульт оператора. Внимательный читатель может заметить, что если в качестве рабочих станций ЛВС используются сплошь МП-контроллеры и только один узел представляет собой ЭВМ, а обмен данными между контроллерами без участия ЭВМ не предусмотрен, то такую систему можно вполне квалифицировать как интерфейс периферийных устройств. Действительно, в случае распределенной системы управления грань между этими типами интерфейсов только на основе функционального описания провести трудно. Отличительным признаком может служить характер протокола обмена. Если протокол предусматривает арбитраж, т.е. управление доступом к шине при попытке инициализации связи более чем одним узлом, то такую сеть следует считать локальной. А если арбитраж не предусмотрен и все сеансы обмена инициируются только ПК, то такая система по характеристикам ближе к интерфейсу периферийных устройств.

Локальные вычислительные сети на уровне распределенных систем управления наиболее часто встречаются при решении задач промышленной автоматизации. Поэтому в связи со встраиваемыми МП-системами, объединенными в локальную сеть, чаще используется термин "промышленная сеть". Промышленные интерфейсы связи используют протоколы, отличные от локальных и глобальных вычислительных сетей. Компании, специализирующиеся в области средств автоматизации, разрабатывают собственные стандарты. Например, DH-485 компании Allen Bradley или Profibus фирмы Siemens. А для относительно простых устройств, состоящих из двух-трех (обычно до

пяти) МК часто используются собственные протоколы. Но в основе всех **промышленных** сетей лежит **последовательный интерфейс**.

Непрерывное совершенствование промышленных последовательных интерфейсов, широкое внедрение МК в автомобильную электронику, которая предъявляет повышенные требования к надежности передачи информации между МП-системами управления разными агрегатами автомобиля (например, между контроллерами педали управления и тормозной системы), привели к созданию нового стандарта последовательной связи – CAN-интерфейса (Controller Area Network). Прогнозируется, что в ближайшем будущем CAN-интерфейс станет основным стандартом в области промышленных сетей связи и в автомобильной электронике. Поэтому многие фирмы – производители элементной базы в течение 1999 г. выпустили на рынок МК с модулем контроллера CAN-сети.

По организации связи интерфейсы подразделяются на магистральные, радиальные (сеть звезда), кольцевые, иерархические, радиально-магистральные (рис. 2). На уровне встраиваемых МП-систем обычно приходится иметь дело с первым или вторым вариантом, причем для локальных сетей используется преимущественно магистральная структура, а для интерфейса периферийных устройств – как первый, так и второй варианты.

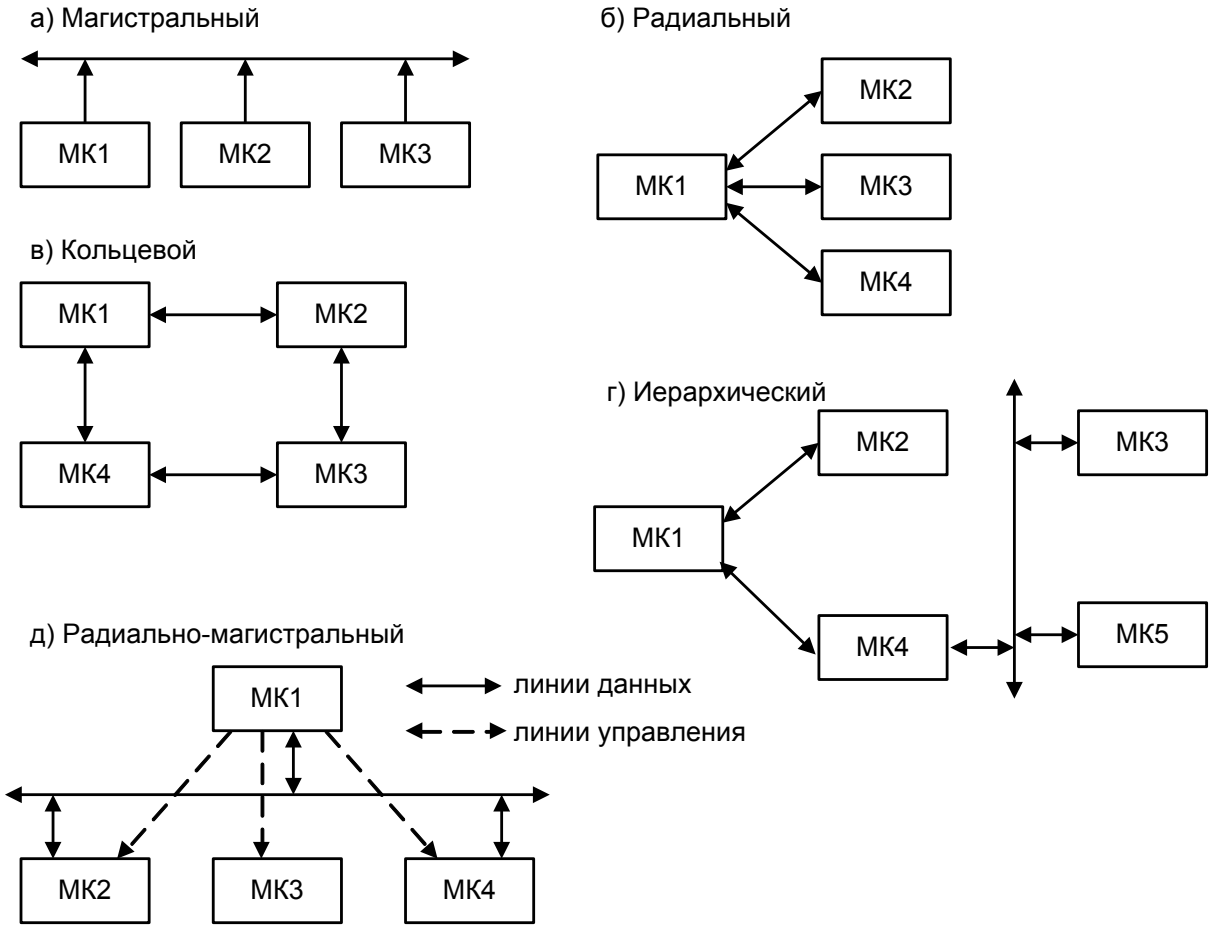


Рис. 2. Способы организации связи.

По режиму обмена информацией интерфейсы подразделяют на симплексные, полудуплексные, дуплексные, мультиплексные (рис. 3). В интерфейсах с симплексным режимом обмена информацией возможна лишь однонаправленная передача информации от одного абонента к другому. Соответственно, и буферы приемника и передатчика информации выполнены однонаправленными. В интерфейсах с полудуплексным режимом

обмена в произвольный момент времени может производиться либо только прием, либо только передача данных между двумя абонентами; буферы приемопередатчика каждого из абонентов связи выполнены двунаправленными. В интерфейсах с дуплексным режимом обмена в любой произвольный момент времени может производиться одновременный прием и передача данных между двумя абонентами.

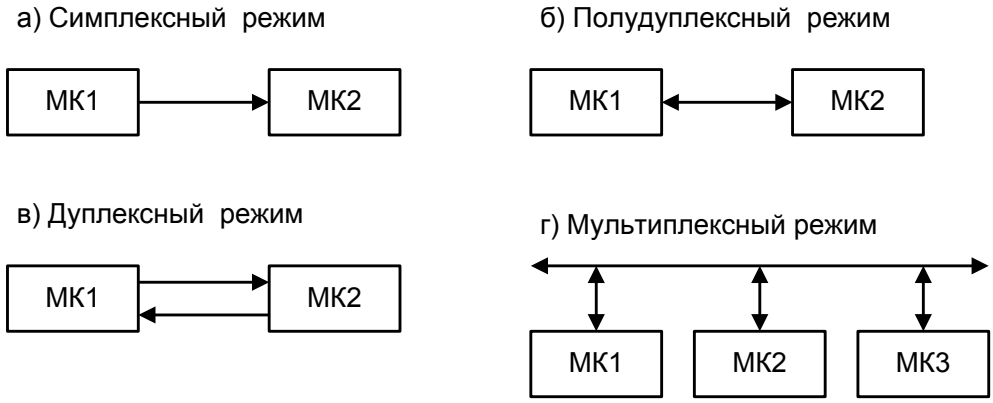


Рис. 3. Структура интерфейсов с различными режимами обмена информацией.

Линии приема и передачи информации физически разделены, соответственно, контроллер обмена каждого абонента имеет два вывода (приемника и передатчика) и буферы этих выводов – однонаправленные. В интерфейсах с мультиплексным режимом обмена в каждый момент времени может осуществляться прием или передача данных между парой любых абонентов сети.

Одним из определяющих моментов начальной стадии проектирования МП-системы является выбор стандарта связи, позволяющего оптимально решить требуемый объем задач контроля и управления путем установления режима гибкого обмена информацией между функциональными блоками интеллектуальной системы. Поэтому при разработке систем на основе разветвленных сетей управления особое внимание стоит уделить изучению общих принципов осуществления связи при возникновении необходимости в обмене информацией между отдельными устройствами в многоабонентской системе. Основой большинства таких систем является **шинная топология (рис. 2, а)**. Прием и передача информации осуществляется по одним и тем же линиям связи, являющимися общими для всех абонентов системы, эти линии и представляют собственную **шину данных системы**. Отдельно взятое устройство взаимодействует с шиной данных посредством подключения выводов своего последовательного интерфейса к соответствующим линиям шины.

В соответствии с существующими на сегодняшний день основными **протоколами связи**, представляющими собой комплексы правил, положений, рекомендаций, определяющих основные принципы взаимодействия между отдельными устройствами (**узлами**) системы, все многообразие многоабонентских систем можно условно разделить на две основные категории. Условность деления объясняется многообразием сетевых протоколов связи последовательных интерфейсов, которые могут сочетать в себе черты обоих категорий сетей.

К первой категории можно отнести **многоабонентские системы**, в состав которых входят равноправные по отношению друг к другу устройства. Последовательная связь может быть инициализирована любым устройством (узлом), подключенным к последовательной шине данных, в произвольный момент времени. Прием сообщения по шине данных может производиться одновременно всеми узлами системы. Нормальное функционирование сети обеспечивается аппаратно поддерживаемыми правилами

арбитража, установленными для используемой разновидности протокола связи, в совокупности с логикой обнаружения и предупреждения ошибочных состояний шины. Под арбитражем следует понимать управление доступом к шине при попытке инициализации связи более чем одним узлом с целью исключения возможных ситуаций столкновения данных на шине, а также для предотвращения "состязаний" выходных буферов устройств. Правила арбитража индивидуальны для каждого сетевого протокола. Узел, выигравший арбитраж, продолжает передачу данных по шине, остальные пытаются сделать это позже.

С целью повышения эффективности процесса передачи данных используется метод программной адресации узлов. Суть метода заключается в следующем. Любое сообщение, передаваемое по шине, имеет определяющий его содержание идентификатор или адрес, который является, как правило, первым кадром передаваемого сообщения. Узлы, получившие идентификатор, сравнивают его значение со значением собственного внутреннего адреса. Узел, которому было адресовано соответствующее послание (значения переданного и собственного адресов совпали), продолжает прием дальнейшей информации. Остальные устройства системы устанавливаются в режим ожидания, не реагируя на передаваемые данные до начала следующего цикла последовательной передачи. Многоабонентские сети, подобные вышеописанным, могут быть построены на основе протоколов последовательных интерфейсов, работающих в синхронном или в асинхронном режимах передачи данных, речь о которых пойдет ниже.

Вторая категория из числа рассматриваемых предполагает наличие в системе одного главного устройства (**MASTER**) и совокупности подчиненных (**SLAVE**). Вся связь в подобных системах, а также инициализация передач последовательных данных осуществляется главным (ведущим) устройством. Подчиненные (ведомые) устройства, как правило, работают в режиме приемников информации. Передача данных от SLAVE к MASTER может быть осуществлена по требованию ведущего, что определяется посылкой соответствующего кода в первом кадре сообщения ведущего. SLAVE, получив указанный код, производит выдачу данных в течение оставшегося времени цикла последовательной связи. Характер выводимой информации определяется содержанием предварительно посланного кода. В рассматриваемой категории сетей для организации индивидуальных связей между устройствами используется метод аппаратной адресации узлов. Выбор подчиненного устройства с целью инициализации связи между ним и ведущим производится в индивидуальном порядке посредством программной установки активного уровня сигнала на аппаратном входе выбора соответствующего ведомого устройства. Так что в цикле последовательной связи в любой момент времени могут принимать участие только два узла из общего количества входящих в сеть: MASTER и выбранный им SLAVE. Рассмотренная категория сетей характерна для интерфейсов, работающих в режиме синхронной передачи данных.

При последовательном обмене данными (бит за битом) требуется обеспечить побитную и покадровую синхронизацию. **Побитная синхронизация** необходима для правильного приема передаваемых битов, **покадровая синхронизация** – для выделения сообщения из принятой последовательности битов.

Известные в настоящее время интерфейсы периферийных устройств с последовательной передачей информации могут работать как в **асинхронном**, так и в **синхронном** режимах.

В синхронном режиме параллельно с передачей по линии данных последовательности информационных битов по линии синхросигналов передается последовательность синхроимпульсов, что позволяет, как правило, повысить скорость передачи и решить проблемы побитной синхронизации передатчика и приемника при передаче длинных информационных сообщений.

В асинхронном режиме побитная синхронизация приемника и передатчика осуществляется обычно по первому (стартовому) биту и затем поддерживается

абонентами в течение времени передачи кадра стабильностью тактовых частот генераторов передатчика и приемника, частоты которых равны и, как правило, минимум в 16 раз превышают частоту передачи данных. С учетом этих обстоятельств, скорость передачи в асинхронном режиме ниже и число бит в информационной посылке (кадре) меньше.

Покадровая синхронизация в асинхронном режиме осуществляется обрамлением информации при передаче по линии стартовым и стоповым битами. Стандартный кадр для асинхронного режима передачи данных представлен на рис. 4.

Покадровая синхронизация в синхронном режиме осуществляется использованием специальных кодовых последовательностей (флагов или специальных знаков) в общем случае в начале и конце кадра. Поскольку в синхронном режиме информационные биты сообщения передаются непрерывным потоком, то для кодирования и декодирования кадров используют специальные договоренности по форматам кадров (протоколам обмена).

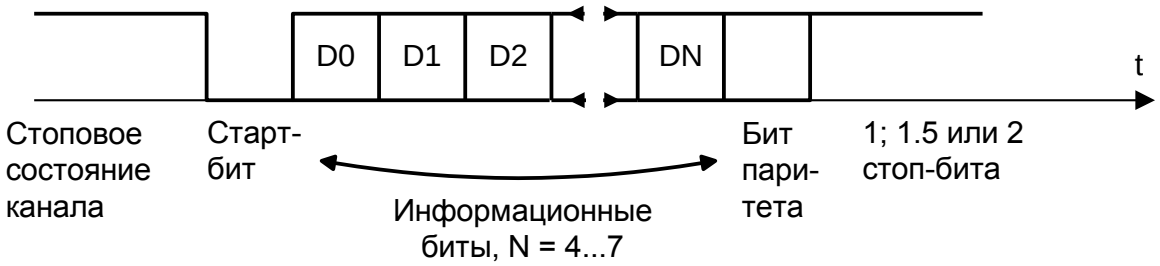


Рис. 4. Формат информационного кадра для асинхронного режима.

В случаях синхронного режима передачи данных протоколы обмена используют один из возможных общепринятых кадров:

- кадр бит-ориентированного протокола типа HDLC (протокол высокоуровневого управления каналом передачи данных);
- кадр бит-ориентированного протокола типа SDLC (протокол синхронного управления звеном данных);
- кадр байт-ориентированного протокола типа Monosync;
- кадр байт-ориентированного протокола типа Bisync.

Форматы кадров, используемых при синхронном режиме передачи данных, представлены на рис. 5.

Для обеспечения кодовой прозрачности аппаратных средств интерфейсов при последовательной передаче информации, под которой понимается свойство последних обеспечивать обмен данными с любыми кодовыми комбинациями, в том числе и с флаговыми (01111110), применяется **процедура битстаффинга**. Процедура битстаффинга предусматривает вставление передающим устройством в поток передаваемых данных двоичного нуля после каждых переданных подряд пяти двоичных единиц и исключение из потока принятых данных одного нуля после каждых принятых пяти единиц приемным устройством.

Поскольку в случае бит-ориентированных протоколов процедурой битстаффинга обрабатывается вся последовательность кадра, кроме флаговых байтов, то внутри кадра (в пределах между открывающим и закрывающим флагами) не оказывается кодовых комбинаций, содержащих подряд более пяти передаваемых в линии двоичных единиц.

В байт-ориентированных протоколах покадровая синхронизация осуществляется передачей двух или одного знака СИН соответственно. Кодовая прозрачность аппаратуры в этом случае обеспечивается процедурой байтстаффинга, а именно введением в пределах информационного и контрольного полей кадра специального символа АР1 перед каждым

управляющим символом СИН1 или СИН2 и перед каждым символом АР1. На приемном конце линии связи осуществляется операция удаления символов АР1, вставленных перед передачей кадра в линию связи.

В последовательных интерфейсах применяют различные методы кодирования последовательной информации. Наиболее часто используются следующие коды:

- Код без возвращения к нулю (БВН) (англоязычная аббревиатура – NRZ).
- Код без возвращения к нулю с инверсией (БВНИ).
- Код Манчестер 2.

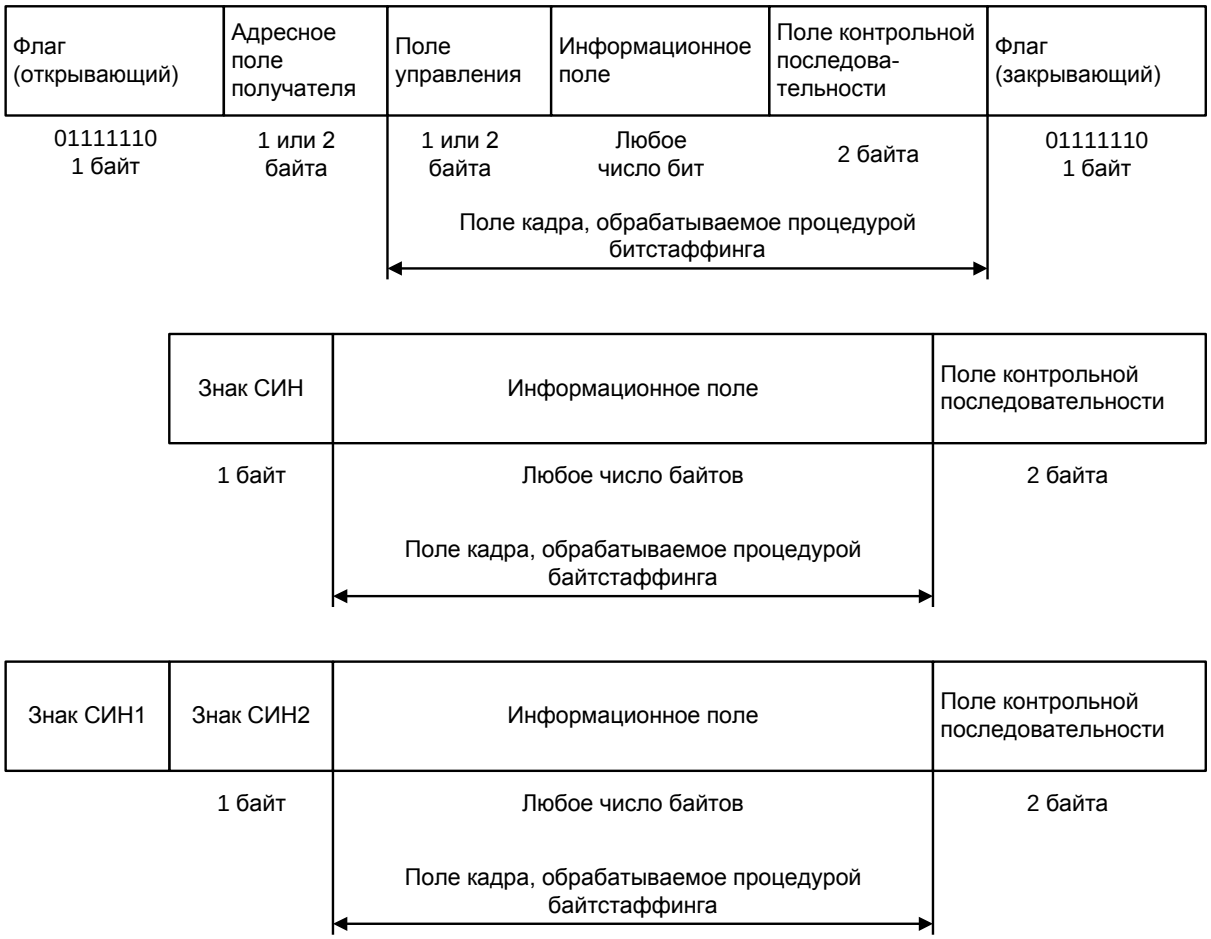


Рис. 5. Примеры информационных кадров для синхронного режима.

Код **без возвращения к нулю** отображает последовательность двоичных битов последовательностью уровней напряжения, постоянных на интервале каждого передаваемого двоичного разряда. В коде **без возвращения к нулю с инверсией** логическая единица передается отсутствием изменения уровня напряжения предшествующего бита, а логический нуль – инверсией этого уровня. Код **Манчестер 2** отображает каждый бит двоичной последовательности переходом уровней: если в середине битового интервала низкий уровень сменяется высоким, то передается логический нуль, если в середине битового интервала высокий уровень сменяется низким, то передается логическая единица. Примеры представления последовательности двоичных битов 10111000011 в кодах БВН, БВНИ и Манчестер 2 приведены на **рис. 6.**

Физический уровень реализации последовательных интерфейсов определяет число линий связи, электрические уровни сигналов на линиях, скорость передачи, максимально допустимую длину линии.

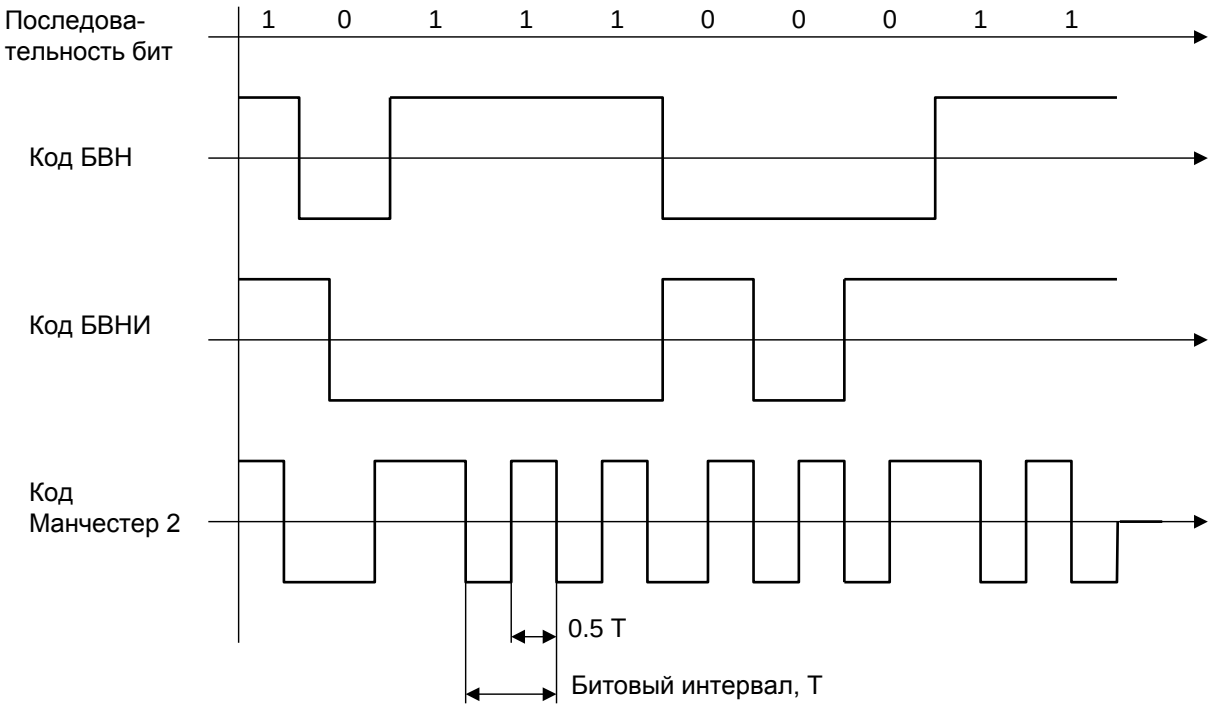


Рис. 6. Примеры кодирования последовательных кодов.

На рис. 7 ... 9 приведены структурные схемы организации физического уровня последовательных интерфейсов, наиболее часто встречающихся в системах с МК. Использованы следующие обозначения: T (передатчик – Transmitter), R (приемник – Receiver), T/R (двунаправленный приемопередатчик) – специальные ИС приемопередатчиков и трансиверов. Обратите внимание, что физический уровень внутриплатных интерфейсов SPI и I2C реализуется логическими сигналами, поэтому специальные схемы формирователей уровней не требуются (см. рис. 12 и 17). Основные параметры рассматриваемых интерфейсов приведены в табл. 1.

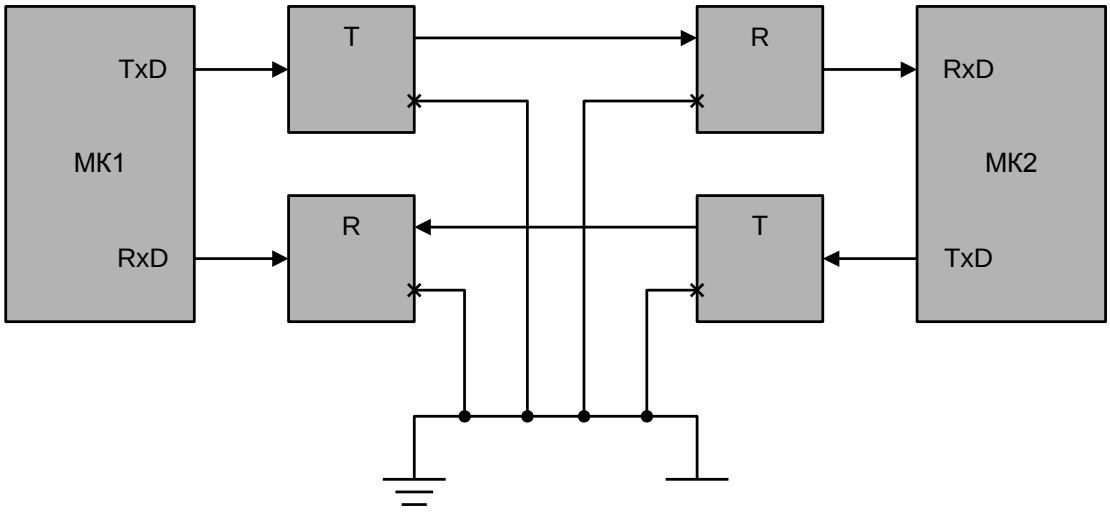


Рис. 7. Организация физического уровня интерфейса RS-232C.

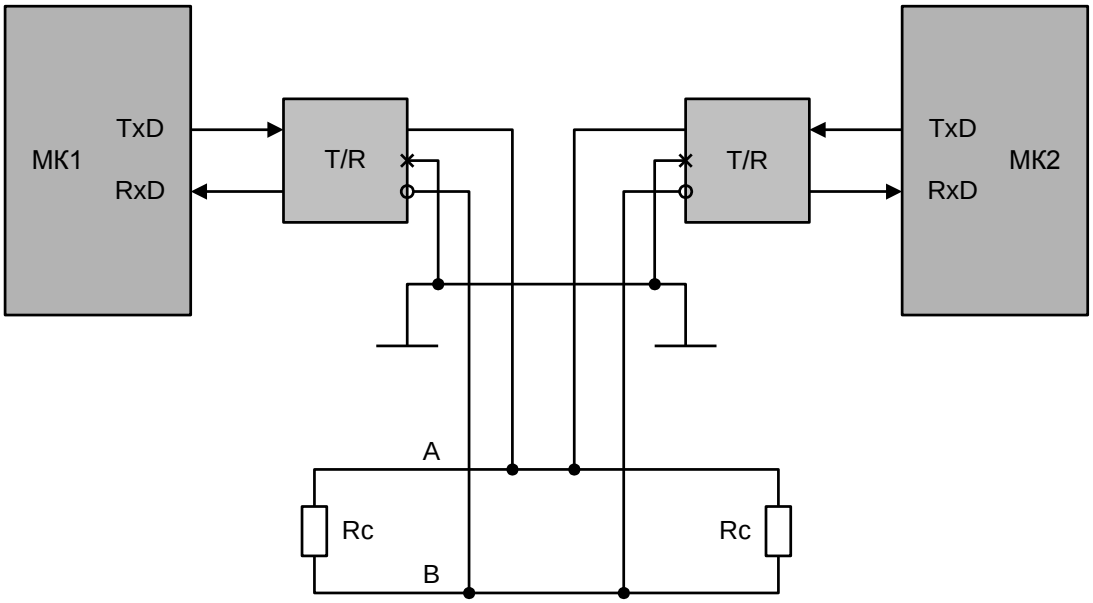


Рис. 8. Организация физического уровня интерфейса RS-485.

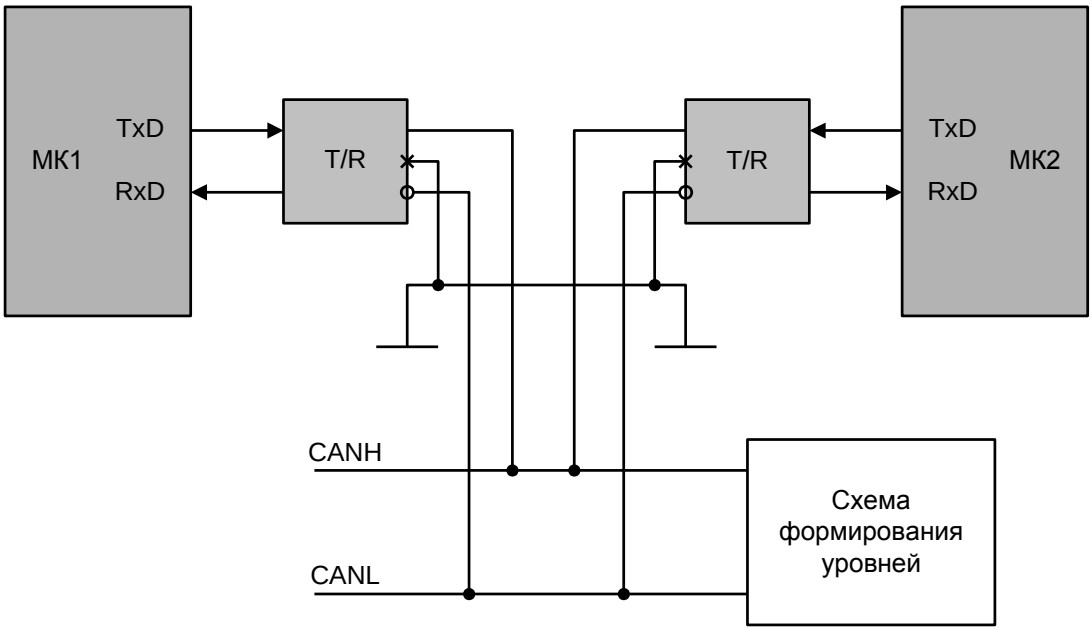


Рис. 9. Организация физического уровня CAN-сети.

Табл. 1. Основные параметры последовательных интерфейсов [2].

Параметры	SPI	12C	RS-232C	RS-485C	CAN
Максимальная скорость передачи	10 Мбод	400 кбод	115 кбод	10 Мбод	1 Мбод
Макс, протяженность линий связи [м]	10	500	15	1200 при 100 кбод 12 при 10 Мбод	5000
Число линий связи	3 + n*	2	2	2	2
Наименование линий связи	MISO, MOSI, SCK	SDA, SCL	TxD, RxD	TxD, RxD	CANL, CANH
Число двунаправленных линий связи	Нет	2	Нет	2	2
“1” передача данных	Соответствует уровню “1” логического сигнала	Соответствует уровню “1” логического сигнала	-2.5В<U<-3.0В	A<B -7В<U_{AB}<-0.3В	> 2 $U_{сети}$ / 3
"0" передача данных	Соответствует уровню "0" логического сигнала	Соответствует уровню "0" логического сигнала	+3.0В<U<+25 В	A>B +0.3В<U_{AB}<+7В	< 2 $U_{сети}$ / 3
Число информационных бит в кадре	8		8	8	До 64
Число служебных бит в кадре сообщения	Нет	3...4	3...4	3...4	До 64
Наименование служебных бит	Нет	Условия СТАРТ, СТОП, флаг АСК, бит направления R/W	Старт-бит, бит паритета, 1 или 2 стоп-бита	Старт-бит, бит паритета, 1 или 2 стоп-бита	SOF, 11-18 бит ID, RTR, IDE, r0, r1, 4бит DLC,16бит CRC, ASK, EOF
Требования к передатчику	Нет	Нет	Iвых. макс. = 500мА $t_{ФР}$ < 0.04 $t_{БИТА}$	Iвых.мдкс = 150мА $t_{ФР}$ < 0.3 $t_{БИТА}$	
Требования к приемнику	Нет	Нет	Rвх. = 3...7кОм, Свх < 2500пФ	Rвх > 12 кОм, Uвх.диф. > 200 мВ |Uвх.син.|<12В	
Элементная база приемопередатчика	Специальные ИС не требуются	Специальные ИС не требуются	Maxim MAX 1488/ 9E MAX 200/235 MAX562 MAX202	Maxim MAX253 MAX3480A MAX481/489 MAX1480 MAX485	Motorola MC33388 MC33389 Siemens TLE6252 TLE6260

Примечание: n* - число ведомых устройств в системе.

3.2 Последовательный интерфейс RS-232C

Последовательный интерфейс RS-232C синхронной и асинхронной передачи данных пока остается наиболее распространенным интерфейсом периферийного оборудования компьютеров [2]. Но на смену ему все активнее внедряется интерфейс USB. Интерфейс RS-232C, определенный стандартом Ассоциации электронной промышленности (EIA) и рекомендациями V.24 CCITT, подразумевает наличие оборудования двух видов: терминального (DTE) и связного (DCE). В качестве терминального оборудования может быть использован персональный компьютер, способный производить прием или передачу данных по последовательному интерфейсу. Под связным оборудованием понимаются устройства, которые могут упростить последовательную передачу данных совместно с терминальным оборудованием. Наглядным примером связного оборудования служит модем. В настоящее время интерфейс используется в самых различных устройствах, в том числе для обеспечения коммуникаций между компьютером и встраиваемой МП-системой управления. Обмен данными производится по двум линиям: линия RxD используется для приема данных, линия TxD – для передачи данных. Линия передачи одного устройства соединяется с линией приема другого, и наоборот (полный дуплекс). Для управления соединенными устройствами используется программное подтверждение (введение в поток передаваемых данных соответствующих управляющих символов). Возможна организация аппаратного подтверждения путем организации дополнительных линий RS-232 для обеспечения функций определения статуса и управления. Максимальная скорость передачи данных по линиям шины RS-232C составляет 115 кбит/с. Максимальная протяженность линий связи, по которым может быть осуществлена передача данных, составляет 15м.

Большинство систем используют асинхронный режим передачи данных интерфейса RS-232, несмотря на то что его спецификация предусматривает также синхронный режим. В асинхронном режиме каждый пакет содержит один символ кода ASCII (американский стандартный код для обмена информацией) или один байт произвольно закодированной информации. Символы кода ASCII представляются семью битами, например, буква A имеет код 1000001.

Наиболее широко распространен формат, включающий в себя один стартовый бит, один бит паритета и два стоповых бита. Сигнал с уровнями ТТЛ, формируемый на выходе TxD модуля контроллера последовательного приемопередатчика при передаче кода буквы A, показан на рис. 10. Начало пакета данных всегда отмечает низкий уровень стартового бита. После него следует 7 бит данных символа кода ASCII. Бит паритета содержит 1 или 0 так, чтобы общее число единиц в 8-битной группе было четным (четный паритет) или нечетным (нечетный паритет). Последними передаются два стоповых бита, представленных высоким уровнем напряжения.

Таким образом, полное асинхронно передаваемое слово данных состоит из 11 бит (фактически данные содержат только 7 бит, остальные биты выполняют служебные функции). Другой распространенный формат, часто используемый в МП-системах для произвольно закодированной информации: один стартовый бит, 8 бит данных, один стоповый бит.

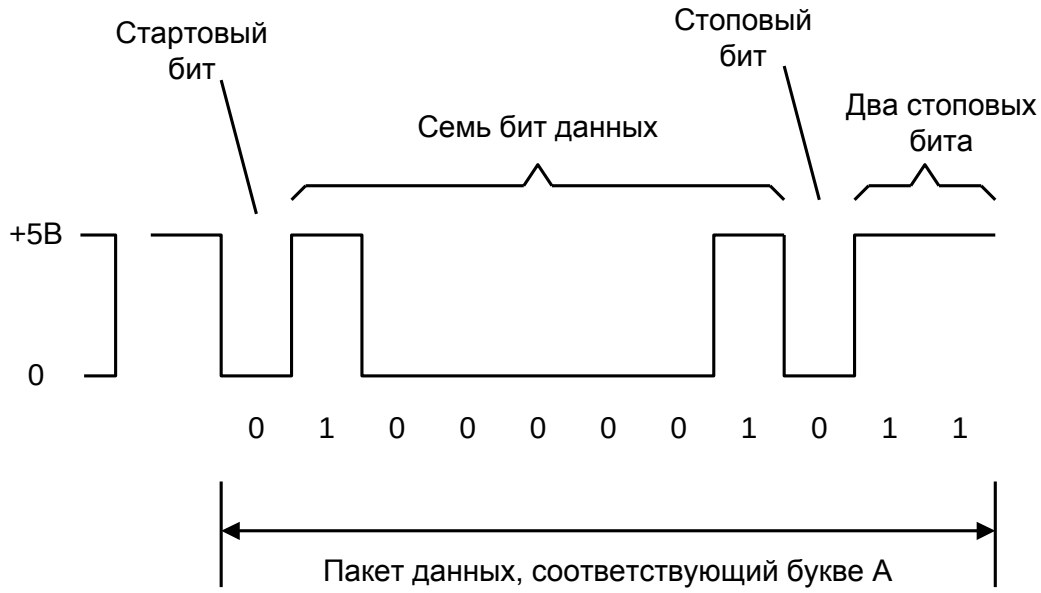


Рис. 10. Представление кода буквы А сигналами уровня ТТЛ.

Сигналы интерфейса RS-232C делят на следующие классы:

- ✓ Последовательные данные (например, TxD, RxD). Интерфейс RS-232C обеспечивает два независимых последовательных канала данных: первичный (главный) и вторичный (вспомогательный). Оба канала могут работать в дуплексном режиме, т.е. одновременно осуществлять передачу и прием информации.
- ✓ Управляющие сигналы квитирования (например, RTS, CTS). Сигналы квитирования – это средство, с помощью которого обмен сигналами позволяет DTE начать диалог с DCE до фактических передачи или приема данных по последовательным линиям связи.
- ✓ Сигналы синхронизации (например, TC, RC). В синхронном режиме (в отличие от более распространенного асинхронного) между устройствами необходимо передавать сигналы синхронизации, которые осуществляют тактирование принимаемого сигнала в целях его декодирования.

На практике вспомогательный канал RS-232C применяется редко, и в асинхронном режиме из 25 сигнальных линий интерфейса обычно используются только девять, которые приведены в табл. 2.

Используемые в интерфейсе RS-232 уровни сигналов отличаются от уровней сигналов ТГЛ, действующих в МК. Логический 0 (SPACE) представляется положительным напряжением в диапазоне от +3 до +25 В, а логическая 1 (MARK) – отрицательным напряжением в диапазоне от -3 до -25 В. На рис. 11 показан сигнал пакета данных для кода буквы А в том виде, в каком он существует на линиях TxD или RxD интерфейса RS-232.

Сдвиг уровня, т.е. преобразование ТТЛ-уровней в уровни интерфейса RS-232, и наоборот, производится специальными микросхемами драйвера линии и приемника линии.

Табл. 2. Основные линии интерфейса RS-232C.

Номер контакта	Сигнал	Выполняемая функция
1	FG	Основная или защитная земля, подключаемая к стойке или шасси оборудования
2	TxD	Последовательные данные, передаваемые от DTE к DCE
3	RxD	Последовательные данные, принимаемые DTE от DCE
4	RTS	Запрос передачи. Активным уровнем этого сигнала DTE указывает, что оно хочет послать данные в DCE
5	CTS	Сброс передачи. Активным уровнем этого сигнала DCE указывает готовность воспринимать данные от DTE
6	DSR	Готовность модема. Активным уровнем этого сигнала DCE сообщает, что связь установлена
7	SG	Возвратный тракт общего сигнала (сигнальная земля)
8	DCD	Обнаружение несущей данных. Активным уровнем этого сигнала DTE показывает, что оно работает и DCE может
9	-	Не задействован

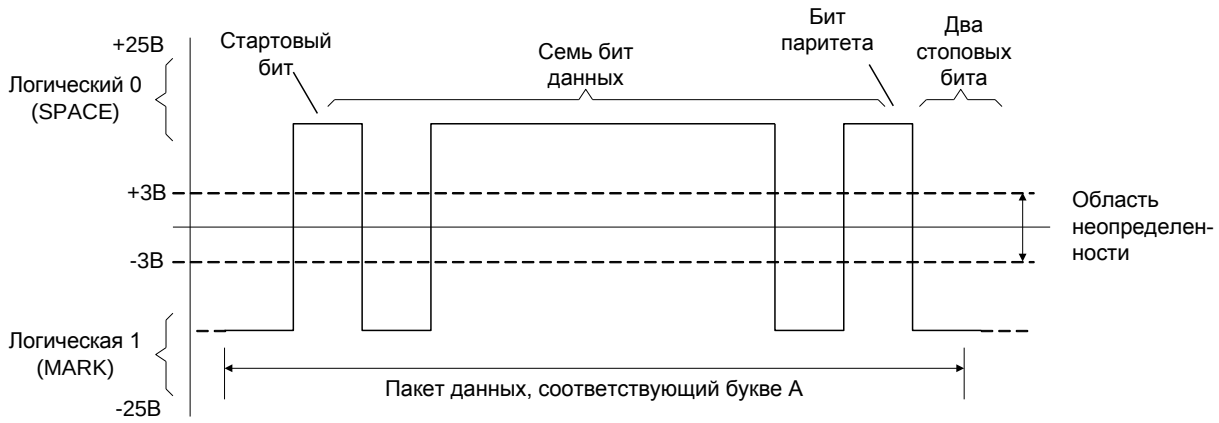


Рис. 11. Вид кода буквы А на линиях связи TxD или RxD интерфейса RS-232.

3.3 Последовательный периферийный интерфейс SPI

Последовательный периферийный интерфейс SPI (Serial Peripheral Interface) предназначен для связи МК с периферийными устройствами МП-системы, основой которой он является [2]. Наиболее часто эти устройства расположены на одной плате с МК, реже – это вынесенные пульты управления, индикаторные панели и т.п. В качестве периферийных устройств могут использоваться как простейшие сдвиговые регистры, так и сложные периферийные ИС со встроенными контроллерами управления, такие, как ЦАП, сигма-дельта АЦП с цифровой фильтрацией, последовательные запоминающие устройства типа FLASH или EEPROM, энергонезависимые ОЗУ и т.д. Рынок периферийных компонентов с интерфейсом, поддерживающим один из протоколов обмена SPI, чрезвычайно широк. Знание этого рынка элементной базы – путь к проектированию встраиваемых МП-систем на современном уровне. В табл. 3 приведены некоторые примеры периферийных ИС с интерфейсом SPI.

Табл. 3. Периферийные компоненты МП-систем с интерфейсом SPI.

Обозначение	Фирма-производитель	Функциональное назначение
MC68HC68T1	Motorola	Часы реального времени, календарь, энергонезависимое ОЗУ, мониторинг напряжения питания
AT45D021	Atmel	Память FLASH 8 Мбит
AD7715 AD7705 AD7706	Analog Devices	16-разрядный сигма-дельта АЦП со встроенным коммутатором, программируемым коэффициентом усиления и самокалибровкой
MAX132	Maxim	18-разрядный АЦП
MC14489	Motorola	Схема динамического управления 5-символьным 7-сегментным индикатором.
AD420AD421	Analog Devices	16-разрядный ЦАП с выходом “токовая петля”

На рис. 12 представлена структурная схема сопряжения МК и двух периферийных ИС с использованием интерфейса SPI.

В рассматриваемом примере МК является ведущим устройством, он инициирует обмен при передаче информации между МК и одной из периферийных ИС. Каждая из периферийных ИС является ведомым устройством. SPI-шина представлена тремя общими линиями связи (MISO, MOSI, SCK) и двумя линиями выбора ведомого устройства (SS1, SS2), которые индивидуальны для каждой периферийной ИС:

- MOSI – линий передачи данных от ведущего к ведомому (Master Output Slave Input).
- MISO – линия передачи данных от ведомого к ведущему (Master Input Slave Output).
- SCK – линия сигнала стробирования данных.
- SS1 и SS2 – линии сигналов выбора ведомого устройства.

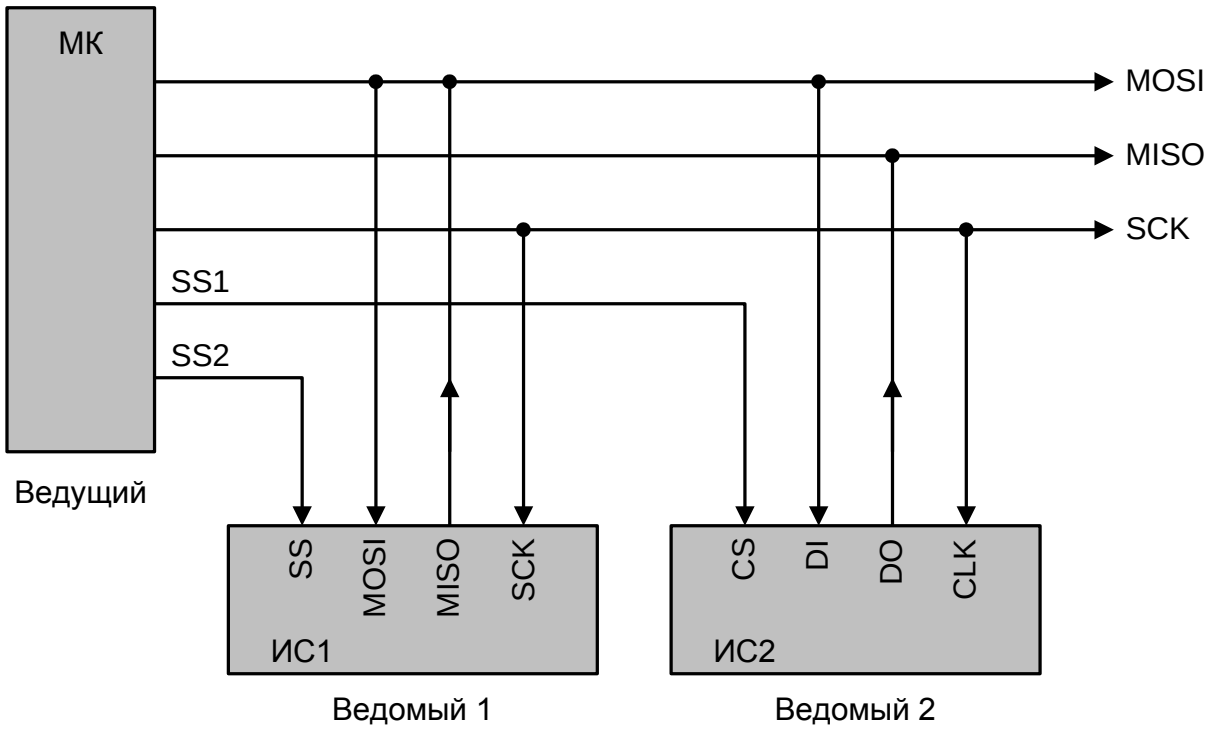


Рис. 12. Сопряжение МК с периферийными ИС посредством интерфейса SPI.

Как видно из рис. 12, образованная на основе интерфейса SPI минисеть относится к классу магистрально-радиальных. Линии передачи данных и линия синхронизации являются примером шинной организации, а линии выбора ведомого устройства – элементом системы радиального типа. Перед началом обмена (рис. 13) ведущее устройство отмечает одно ведомое устройство, с которым будет производиться обмен. Для этого на линии выбора устройства SSi устанавливается низкий активный уровень сигнала. Затем ведущее устройство последовательно выставляет на линию MOSI восемь бит информации, сопровождая каждый бит сигналом синхронизации SCK. Ведомое устройство дешифрирует переданный байт информации и определяет, в каком направлении будет производиться дальнейший обмен. Если ведомое устройство должно принимать информацию, то ведущее устройство, не снимая сигнала выбора ведомого SSi, продолжит передачу по линии MOSI. Если ведомое устройство должно передавать информацию, то оно активизирует линию MISO и в ответ на каждый импульс синхронизации от ведущего будет выставлять один бит информации. Длина посылки обмена в общем случае не ограничена и может составлять даже не целое число байтов. Завершение обмена также инициируется ведущим посредством установки в неактивное состояние сигнала выбора ведомого SSi.

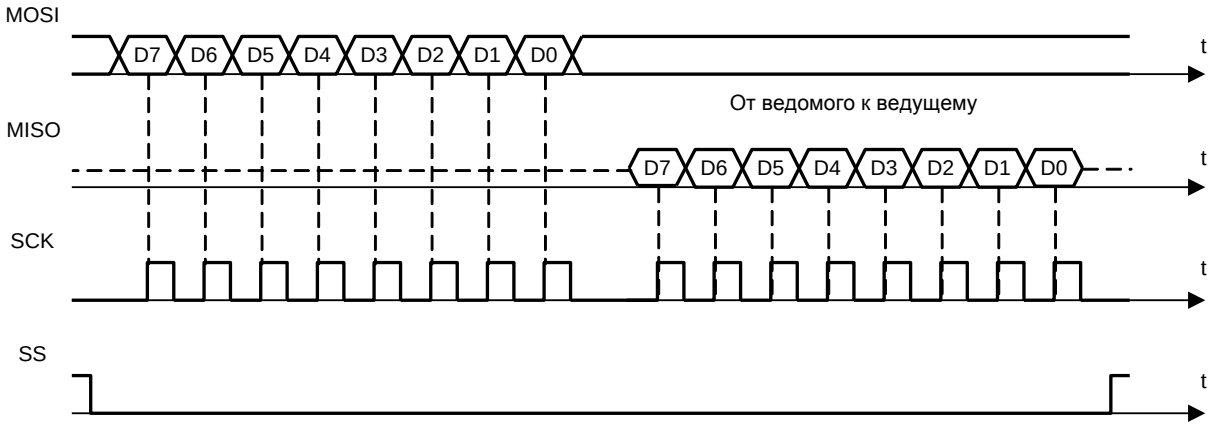


Рис. 13. Временные диаграммы, поясняющие принцип действия SPI-интерфейса.

Значительно реже SPI-интерфейс используется для организации межпроцессорной связи. По мнению автора, чрезвычайно удобным может оказаться его применение в достаточно простых, но многопроцессорных системах управления устройствами силовой электроники, которые конструктивно требуют смещения устройств управления разными потенциалами. И именно полнодуплексный SPI-интерфейс позволяет организовать индивидуальную потенциальную развязку для каждого МК системы (рис. 14).

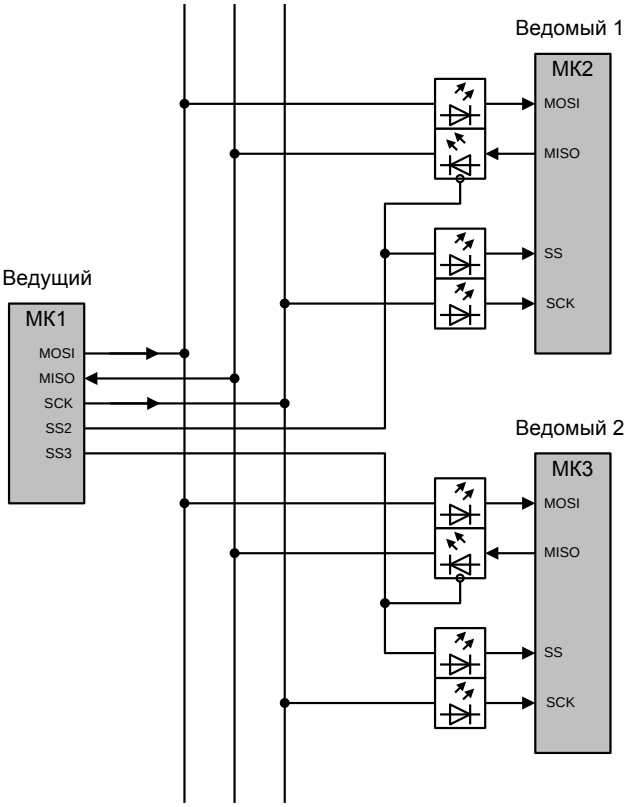


Рис. 14. Связь МК посредством интерфейса SPI с потенциальной развязкой.

Для подключения к SPI-шине встроенный контроллер SPI имеет четыре вывода: MOSI, MISO, SCK, SS. Встроенный контроллер SPI может работать как в ведущем, так и в ведомом режиме. Скорость приема и передачи определяется частотой тактирования

межмодульных магистралей МК f_{bus}: в ведущем режиме скорость обмена не может превышать $f_{bus}/2$, в ведомом режиме максимальная скорость обмена равна f_{bus}. Поэтому для большинства МК максимальная скорость обмена в ведущем режиме составляет 1 Мбит/с, в ведомом – 2 Мбит/с.

При работе встроенного контроллера в ведущем режиме к выводу MOSI подключается выходная линия данных, а к MISO – входная. При работе в ведомом режиме выводы меняются ролями. Вывод SCK является выходом, если контроллер SPI работает в ведущем режиме, и входом, если в ведомом. В системах с несколькими ведущими устройствами все выводы SCK соединяются вместе. То же делается с выводами MOSI и MISO. На время отсутствия связи буферы выводов встроенного контроллера SPI переводятся в высокоимпедансное состояние. Последнее позволяет избежать конфликтов на шине SPI. В противном случае несколько выводов MISO ведомых устройств одновременно были бы активными, что не позволило бы ведущему устройству произвести прием достоверной информации.

Только одно из устройств системы в каждый момент времени может работать в ведущем режиме, остальные – только в ведомом. Ведущее устройство формирует сигналы на выводах MOSI и SCK, которые поступают на одноименные выводы ведомых устройств. Одно из выбранных ведомых устройств передает данные через вывод MISO на вывод MOSI ведущего устройства. Автоматическое управление направлением передачи выводов MOSI, MISO и SCK позволяет обойтись без изменений во внешних схемах управления, когда новое устройство начинает работать в ведущем режиме.

Вывод SS встроенного контроллера SPI используется в зависимости от того, в каком режиме работает данное устройство. При работе в ведомом режиме при подаче высокого уровня сигнала на вход SS устройство игнорирует сигналы SCK и удерживает вывод MISO в высокоимпедансном состоянии. Если же в ведомом режиме работы на входе SS установлен низкий логический уровень, то буферы линий MOSI и SCK разворачиваются на ввод, линия MISO – на вывод. При работе в ведущем режиме вывод SS может быть использован как обычная линия вывода. В системах со сложной логикой работы этот вывод может использоваться как вход сигнала обнаружения ошибки для индикации состояния шины в случаях, если более чем одно устройство пытается стать ведущим.

Схема управления контроллера SPI-интерфейса позволяет выбрать один из двух протоколов обмена и полярность импульсов синхронизации SCK. При работе в ведущем режиме возможно также программно выбрать частоту импульсов синхронизации.

Протоколы связи SPI. Два бита регистра управления любого контроллера SPI-интерфейса определяют временную диаграмму обмена по шине SPI:

- Бит CPHA назначает протокол обмена.
- Бит CPOL определяет полярность сигнала синхронизации SCK.

В соответствии с комбинацией битов CPHA:CPOL принято различать четыре режима работы интерфейса SPI. Комбинации CPHA:CPOL = 00 соответствует режим 0, комбинации CPHA:CPOL = 11 – режим 3. Встроенный контроллер SPI МК позволяет программно настраивать режим SPI в процессе инициализации, в то время как периферийные ИС реализуют один или два режима SPI, которые определяются их техническим описанием. Наиболее часто это режимы 0 и 3.

На рис. 15 представлены временные диаграммы сигналов для протокола передачи CPHA = 0. Для сигнала SCK приводятся две диаграммы, различающиеся полярностью сигнала. Первая соответствует режиму 0, вторая – режиму 1. Диаграммы относятся как к ведущему, так и к ведомому устройству, поскольку выводы MISO и MOSI ведущего соединены с аналогичными выводами ведомого. Сигнал SS подается только на ведомое устройство. Поэтому вывод SS у ведущего остается незадействованным и его диаграмма не представлена, но имеется в виду, что она соответствует неактивному состоянию. Встроенные контроллеры SPI выполнены таким образом, что длина посылки составляет один байт, что и отражено на временных диаграммах.

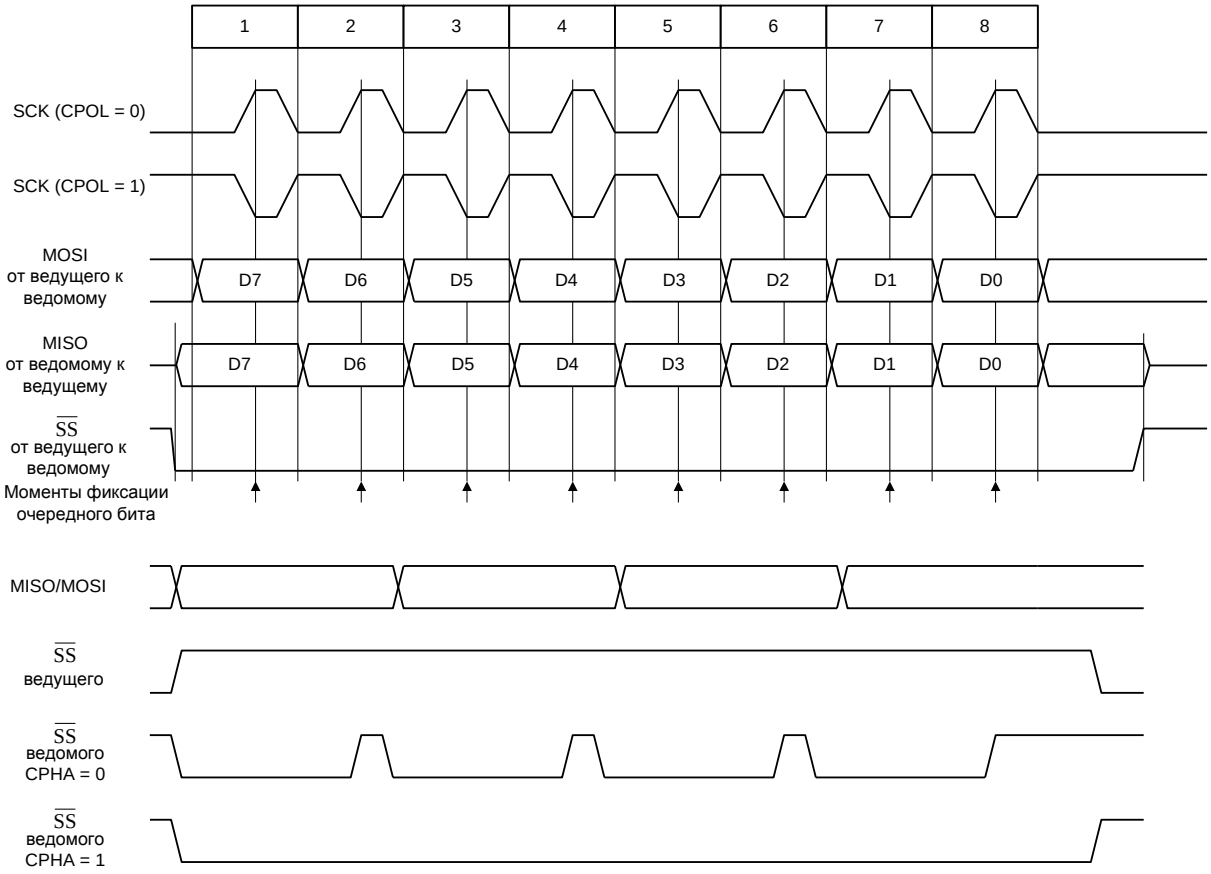


Рис. 15. Временные диаграммы обмена SPI-интерфейса в режимах 0 и 1.

Начало обмена рассматриваемого протокола определяется установкой сигнала выбора ведомого SS в активное состояние SS = 0. При направлении передачи от ведущего к ведомому первый перепад сигнала синхронизации SCK используется ведомым устройством для запоминания очередного бита во внутреннем сдвиговом регистре контроллера SPI. Ведущий выставляет очередной бит посылки на линии MOSI по каждому четному фронту сигнала SCK. При передаче данных от ведомого к ведущему старший бит передаваемого байта должен быть выставлен ведомым на линию MISO сразу после изменения уровня сигнала SS = 0. По первому фронту SCK уровень сигнала на линии MISO будет запомнен в младшем разряде сдвигового регистра ведущего устройства. По этой причине сигнал на линии выбора ведущего должен быть возвращен в неактивное состояние SS = 1 после передачи каждого байта в любом направлении. Тогда передача каждого нового байта будет сопровождаться предварительной установкой SS в 0.

Начало обмена для протокола опции CPHA = 1 (рис. 16) определяет первое изменение уровня сигнала на линии SCK после установки сигнала выбора ведомого SS в активное состояние SS = 0. При передаче данных от ведущего к ведомому и в обратном направлении все нечетные перепады SCK вызывают выдвижение очередного бита посылки из сдвигового регистра передатчика на линию. Каждый четный перепад используется для записи этого бита в сдвиговый регистр приемника. Сигнал выбора ведомого может оставаться в активном состоянии S3 = 0 в течение передачи нескольких байт информации. Это несколько упрощает логику программного драйвера SPI.

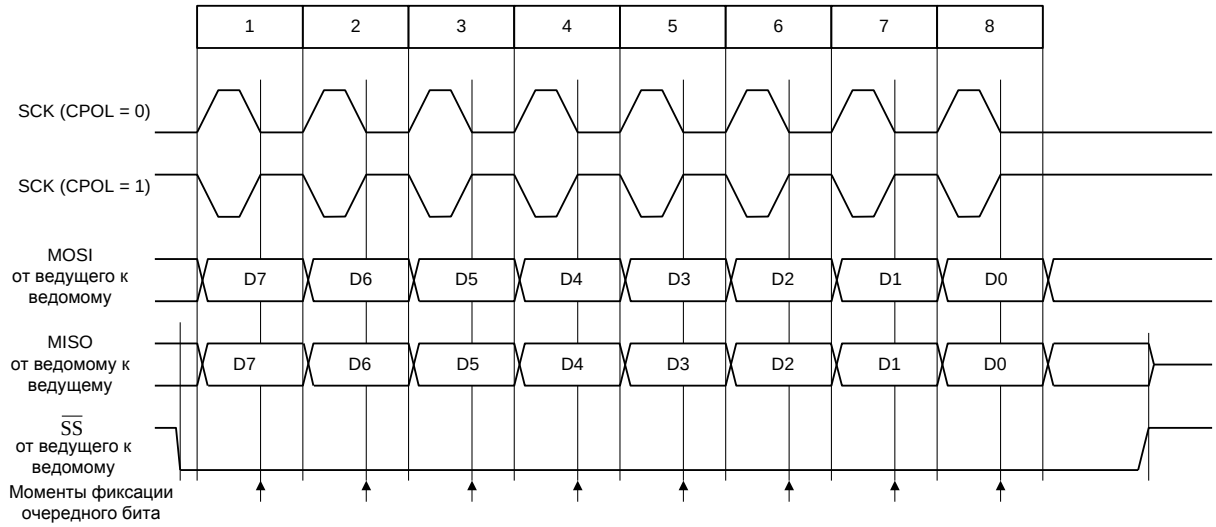


Рис. 16. Временные диаграммы обмена SPI-интерфейса в режимах 2 и 3.

Рассмотренные протоколы обмена не имеют различий по скорости и надежности передачи информации. Выбор протокола диктуется периферийным устройством. В некоторых случаях полярность и фаза сигнала SCK изменяются между передачами, для того чтобы обеспечить связь с устройствами, имеющими различный протокол.

Системные ошибки SPI. Предусмотрено обнаружение двух системных ошибок в системах, использующих SPI-стандарт. Первая ошибка носит название "ошибка режима", вторая – "ошибка записи".

Ошибка режима. Эта ошибка возникает при работе SPI в ведущем режиме, если сигнал на выводе SS изменяется с 1 на 0. Это происходит, когда какое-то из ведомых устройств пытается стать ведущим. Для буферных устройств микроконтроллеров это может привести к катастрофическим последствиям.

В некоторых случаях защита не может быть осуществлена в полной мере. Например, если второе устройство перешло в ведущий режим, но не изменило сигнал SS сразу. Или два ведомых устройства пытаются одновременно использовать линию MISO. В этом случае может быть состязание их выходных буферов, но сигнала "ошибки системы" не будет и буферы останутся незащищенными. Можно дать несколько советов разработчику по предотвращению необратимых последствий.

Полезный совет. Включайте гасящий резистор от 1 до 10 кОм последовательно с выводами SPI. (При нормальной работе резистор не оказывает заметного воздействия на сигнал, но при состязании буферов он ограничивает ток и предотвращает возможность их повреждения.) Или используйте режим "монтажное ИЛИ". Следует отметить, что оба совета связаны с некоторым снижением скорости передачи, которая начинает зависеть от емкости соединительных проводов.

Ошибка записи. Эта ошибка возникает при записи в регистр данных в процессе передачи данных и определяется особенностями логики встроенных систем, реализующих протокол связи в стандарте SPI. Поскольку регистр данных не имеет буфера для передаваемых данных, запись производится непосредственно в сдвиговый регистр. Для предотвращения искажения передаваемых данных предусматривается сигнал **ошибки записи**. Передача данных продолжается без нарушений, но новые данные в сдвиговый регистр не записываются. Эта ошибка обычно возникает в ведомых устройствах, так как они не располагают точной информацией о том, когда ведущее устройство выйдет на связь. Ведущее устройство само инициирует передачу и знает, когда работает его передатчик, поэтому ошибка такого рода для него маловероятна, хотя она обнаруживается так же, как и в ведомом устройстве.

3.4 Синхронный последовательный интерфейс I²C

Интерфейс I2C (синхронный последовательный интерфейс I2C) является двухпроводным последовательным интерфейсом, разработанным фирмой Philips [2]. Изначально в стандартном режиме шина была предназначена для скорости передачи данных до 100 кбит/с. В усовершенствованном быстром режиме поддерживается передача данных со скоростью до 400 кбит/с. На одной шине одновременно могут работать устройства и стандартного, и быстрого режима. Основными свойствами шины являются следующие:

- Двунаправленная передача данных между главными и подчиненными устройствами.
- Многоабонентская связь (нет центрального главного узла).
- Арбитраж между одновременно передающими устройствами без разрушения целостности передаваемых данных.
- Последовательная тактовая синхронизация позволяет приборам с различными скоростями передачи битов осуществлять связь через одну последовательную шину.
- Последовательная тактовая синхронизация может использоваться в качестве механизма квитирования установления связи, чтобы приостанавливать и возобновлять последовательную передачу.

I2C-шина может использоваться в целях тестирования и диагностики.

I2C-шина использует два провода (SDA и SCL) для передачи информации между приборами, соединенными этой шиной. Типичная конфигурация I2C-шины показана на рис. 17.

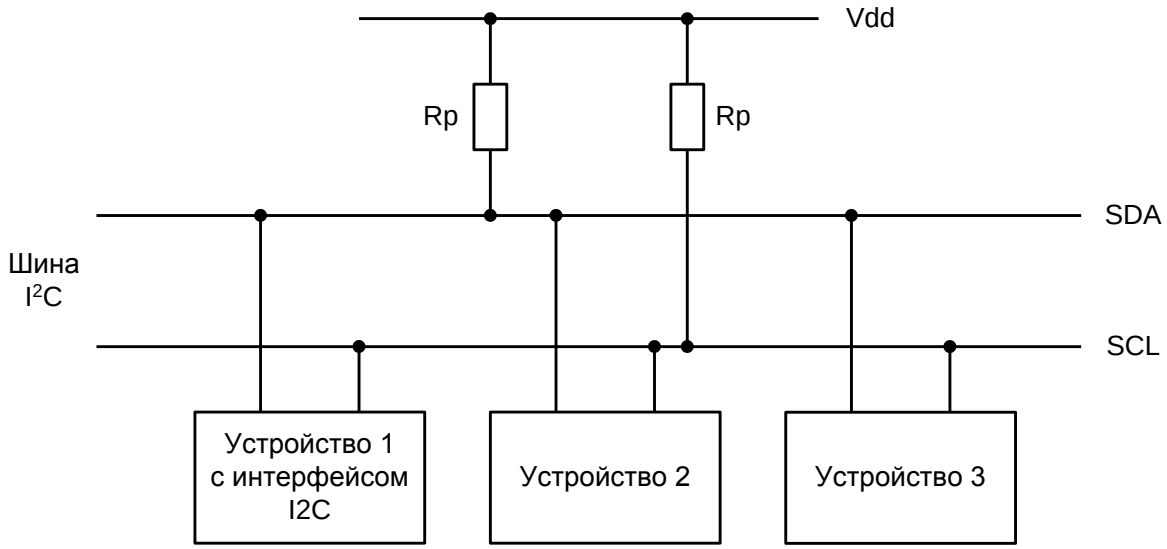


Рис. 17. Конфигурация I²C-шины.

Протокол связи I²C. Интерфейс I²C использует протокол с битом подтверждения ASK для обеспечения надежной передачи и приема. При передаче одно устройство – ведущее (выдает тактовый сигнал), а другое – ведомое. Все составляющие протокола ведомого, включая поддержку общего вызова, а также протокол ведущего реализованы аппаратно в модуле I²C.

Адресация на шине I²C. В протоколе интерфейса I²C каждое устройство имеет адрес. Когда ведущий хочет инициировать передачу данных, он сначала передает адрес устройства SLA, к которому хочет обратиться. Все устройства на шине следят за выставляемым на шину адресом и сравнивают его с собственным. Вместе с адресом

передается бит направления передачи R/W, который определяет, будет ли ведущий читать из ведомого или писать в него.

Существуют два формата адреса. Простейший – 7-битный формат с битом R/W (рис. 18). Адрес – это 7 старших битов байта. Более сложным является 10-битный формат с битом R/W (рис. 19). В 10-битном адресе первые пять битов определяют, что это 10-битный адрес.

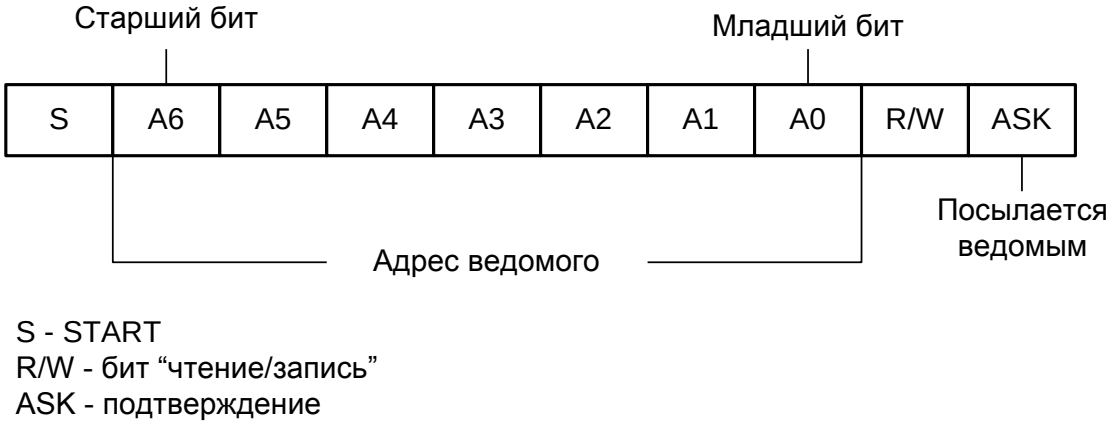


Рис. 18. Формат 7-битного адреса.

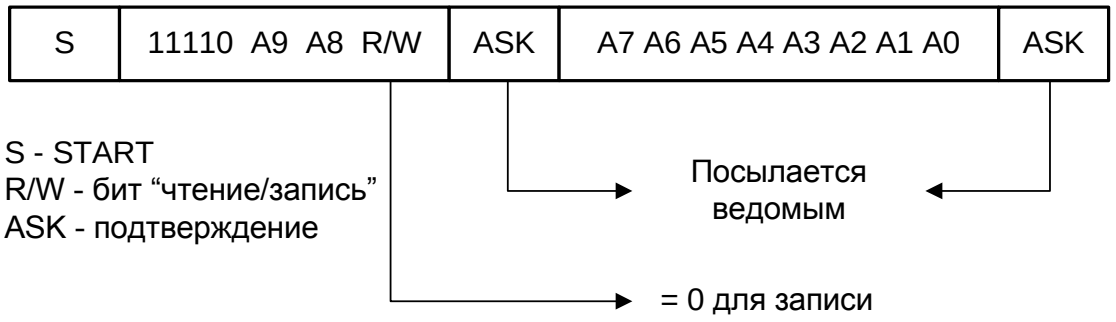


Рис. 19. Формат 10-битного адреса.

Основные типы передачи данных. В зависимости от направления передачи битов (R/W) для I^2C-шины возможны два типа передачи данных:

- Передача данных от главного передатчика к подчиненному приемнику. Первый байт, передаваемый передатчиком, является адресом подчиненного приемника. Затем следует несколько байтов данных. Подчиненный приемник возвращает бит подтверждения после каждого принятого байта.
- Передача данных от подчиненного передатчика к главному приемнику. Первый байт (адрес подчиненного передатчика) передается главным устройством. Затем подчиненный передатчик возвращает бит подтверждения. Следующие несколько байтов данных передаются подчиненным устройством к главному. Главное устройство возвращает бит подтверждения после всех принятых байтов, кроме последнего байта. В конце последнего принятого байта возвращается "нет подтверждения".

Инициализация и прекращение передачи данных. Когда нет передачи данных (режим ожидания), линии тактирования SCL и данных SDA приведены подтягивающим резистором к высокому уровню. Главное устройство генерирует все последовательные

синхроимпульсы и условия START и STOP, определяющие начало и конец передачи данных.

Условие START определяется как переход SDA из высокого уровня в низкий при высоком уровне SCL, а условие STOP – как переход SDA из низкого уровня в высокий при высоком уровне SCL. Ввиду такого способа определения условий START и STOP при передаче данных, линия SDA может изменять свое состояние только при низком уровне SCL.

Когда ведущий не желает освобождать шину (выставив STOP), он должен выставить повторное условие START, которое идентично START (переход SDA из высокого уровня в низкий при высоком уровне на SCL), но выдается вслед за подтверждением, являясь началом следующей последовательной передачи, таким образом шина не освобождается. На **рис. 20** показано, как осуществляется передача данных по шине в соответствии с принятым стандартом протокола связи.

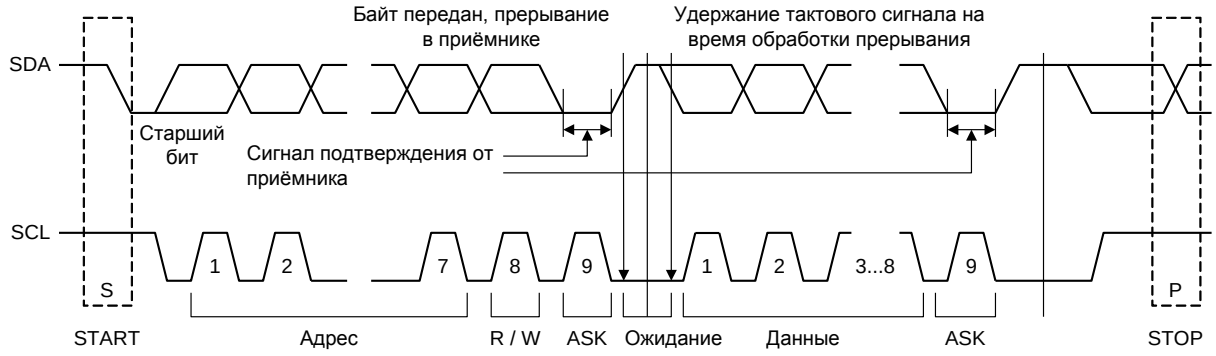


Рис. 20. Пример обмена данными в I^2C.

Режимы работы I^2C-логики. I^2C-логика устройств связи, обеспечивающая последовательный интерфейс, который удовлетворяет спецификации I^2C-шины и поддерживает два вышеперечисленных типа передачи данных, может работать в следующих четырех режимах.

Режим главного передатчика. Последовательный вывод данных через SDA-линию, в то время как SCL выдает последовательные синхроимпульсы. Первый переданный байт содержит адрес подчиненного приемного устройства (7 бит) и бит направления данных. В этом случае бит направления данных (R/W) будет логическим 0, мы говорим, что передается "W". Таким образом, первый переданный байт представляет собой SLA+W. Последовательные данные передаются по 8 бит. Условия START и STOP выводятся для указания начала и конца последовательной передачи.

Режим главного приемника. Первый переданный байт содержит адрес подчиненного передающего устройства (7 бит) и бит направления данных. В этом случае бит направления данных (R/W) будет логической 1 и мы говорим, что передается "R". Таким образом, первый переданный байт представляет собой SLA+R. Последовательные данные передаются через линию SDA, в то время как линия SCL выдает последовательные синхроимпульсы. Последовательные данные передаются по 8 бит. После того как принят каждый байт, передается бит подтверждения. Условия START и STOP выводятся для индикации начала и конца последовательной передачи.

Режим подчиненного приемника. Последовательные данные и последовательные синхроимпульсы передаются через линии SDA и SCL. После того как принят каждый байт, передается бит подтверждения. Условия START и STOP распознаются как начало и конец последовательной передачи. Распознавание адреса выполняется аппаратным обеспечением после приема адреса подчиненного устройства и бита направления.

Режим подчиненного передатчика. Первый байт принимается и обрабатывается как в режиме подчиненного приемника. Однако в этом режиме бит направления будет указывать, что направление передачи изменено на обратное. Последовательные данные передаются через линию SDA, в то время как последовательные синхроимпульсы вводятся через SCL. Условия START и STOP выводятся для индикации начала и конца последовательной передачи.

В подчиненном режиме аппаратные средства I^2C-интерфейса осуществляют поиск своего собственного подчиненного адреса и адреса общего вызова. Если детектируется один из этих адресов, запрашивается прерывание. Когда МК желает, чтобы шина стала главной, аппаратные средства ожидают, пока шина освободится, прежде чем ввести главный режим, так что возможное действие в качестве подчиненного не прерывается. Если арбитраж шины потерян в главном режиме, то соответствующий передатчик переключается в подчиненный режим немедленно и может детектировать свой собственный подчиненный адрес в той же последовательной передаче.

Работа при конкуренции. Протокол I^2C допускает наличие более одного ведущего в системе. Это называется конкуренцией. Когда два или более ведущих пытаются передать данные одновременно, необходимы арбитраж и синхронизация.

Арбитраж. Арбитраж осуществляется на шине данных SDA, пока на SCL высокий уровень. В режиме главного передатчика арбитражная логика проверяет, чтобы каждая переданная логическая единица действительно появлялась в виде логической 1 на I^2C-шине. Если другой прибор на шине отменяет логическую 1 и переводит линию SDA в низкий уровень, арбитраж теряется и соответствующий ведущий немедленно переходит из режима главного передатчика в режим подчиненного приемника, поскольку устройство, выигравшее арбитраж, может обратиться к нему. Устройство будет продолжать выдавать синхроимпульсы (на SCL) до тех пор, пока передача текущего последовательного байта не будет завершена.

Арбитраж может также быть потерян в режиме главного приемника. Потеря арбитража в этом режиме может произойти только в то время, когда ведущее устройство возвращает сигнал "нет подтверждения" (логическая 1) на шину. Арбитраж теряется, когда другой прибор на шине изменит этот сигнал на низкий уровень. Поскольку это может произойти только в конце последовательного байта, устройство не генерирует в дальнейшем тактовые импульсы. Арбитраж не допускается между повторными сигналами START, условием STOP и битом данных, повторным START и STOP. Ведущий должен обеспечить невозможность возникновения приведенных условий. **Рис. 21** показывает процедуру арбитража.

Синхронизация тактовых сигналов. Синхронизация тактовых сигналов возникает, когда устройства начинают арбитраж, и реализуется благодаря использованию соединения "монтажного И" с SCL. Переход SCL из высокого уровня заставляет все устройства, вовлеченные в арбитраж, начать отсчет длительности низкого уровня. После того как тактовый сигнал устройства перешел, а низкий уровень, оно будет удерживать этот уровень на SCL, до тех пор, пока тактовый сигнал не перейдет в высокий уровень, но при этом на SCL может по-прежнему сохраняться низкий уровень, если уровень тактового сигнала другого устройства все еще низкий. Низкий уровень на SCL удерживается устройством с самым длинным полупериодом низкого уровня. Устройства с более коротким полупериодом выдают высокий уровень на SCL и переходят в состояние ожидания, до тех пор, пока на SCL не появится высокий уровень. После этого все устройства начинают отсчет длительности высокого уровня. Устройство с самым коротким полупериодом высокого уровня по завершении его переведет SCL в низкий уровень. Длительность высокого уровня на SCL определяется устройством с самым коротким полупериодом высокого уровня. Процедура синхронизации представлена на **рис. 22**.

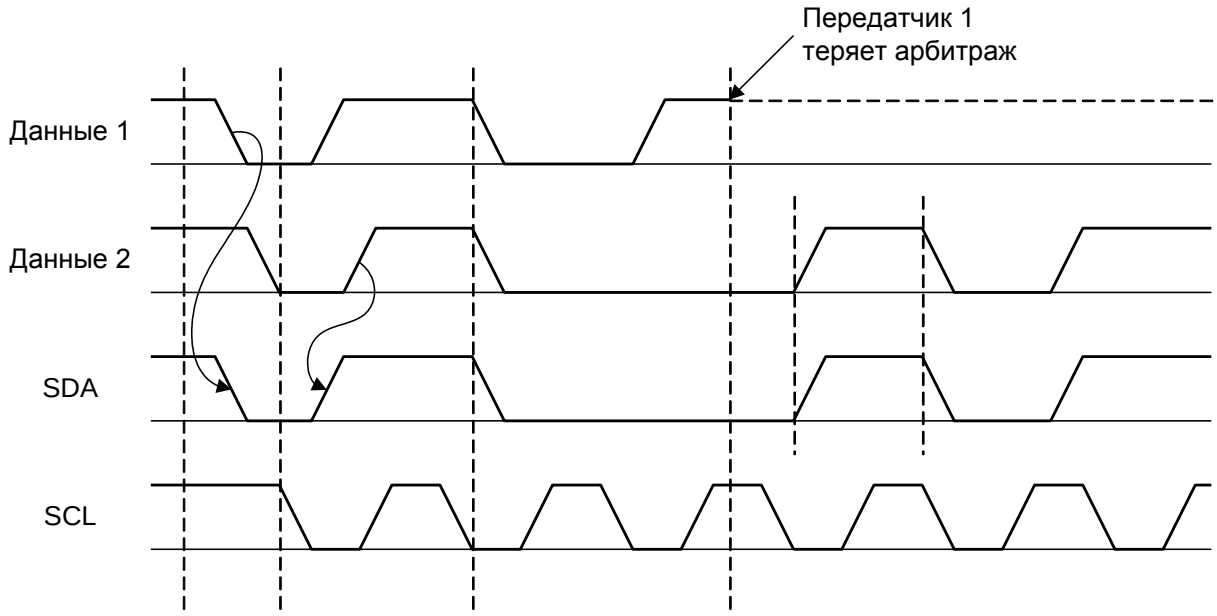


Рис. 21. Арбитраж при конкуренции (2 ведущих).

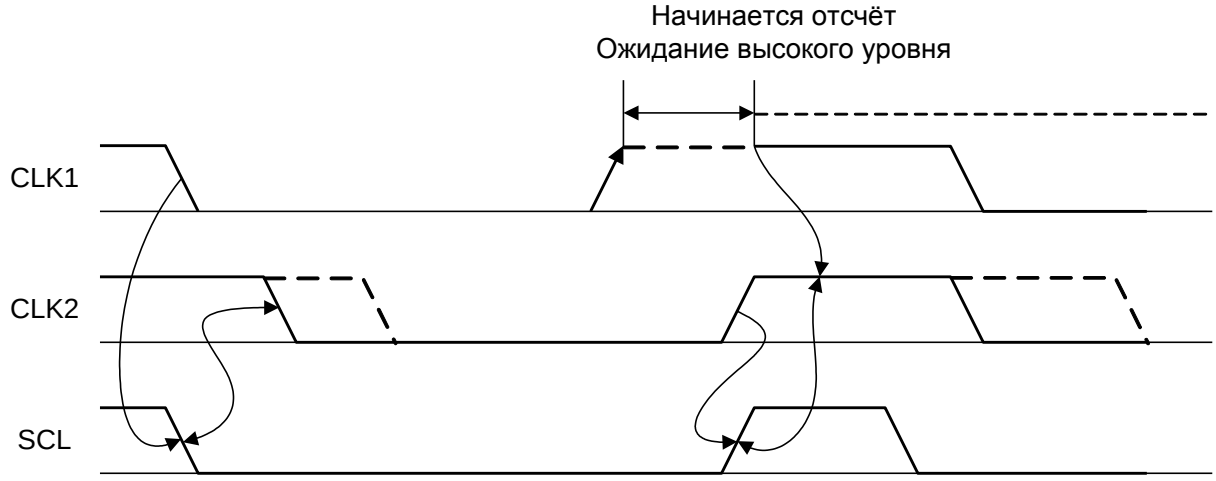


Рис. 22. Синхронизация тактовых сигналов интерфейса I^2C.

Коллизии на линиях шины и их обработка PC-логикой. Аппаратные средства I^2C-интерфейса имеют возможность обрабатывать некоторые специальные случаи, которые могут возникать во время последовательной передачи.

Одновременно повторяющиеся условия START от двух главных устройств. Повторяющиеся условия START могут генерироваться в режиме главного передатчика или главного приемника. Особая ситуация возникает, если два главных устройства одновременно генерируют повторяющееся условие START (рис. 23). Пока это происходит, арбитраж не теряется ни одним из главных устройств, так как они оба передают одни и те же данные. Если аппаратные средства I^2C детектируют повторяющееся условие START на I^2C-шине до того, как они сами генерируют повторное условие START, то они освободят шину и запрос на прерывание генерироваться не будет. Если другое главное устройство "освобождает" шину путем подачи условия STOP,

исходное устройство будет передавать нормальное условие START и пытаться повторно начать прерванную передачу последовательных данных.

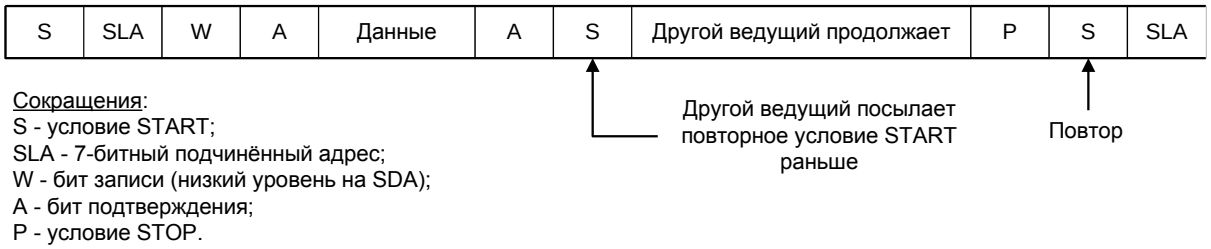


Рис. 23. Одновременно повторяющиеся условия START от двух главных устройств.

Принудительный доступ к I^2C-шине. В некоторых случаях неконтролируемый источник может вызвать неожиданный останов (зависание) шины. В подобных ситуациях проблема может быть вызвана взаимными помехами, временным прерыванием шины или временным коротким замыканием между SDA и SCL.

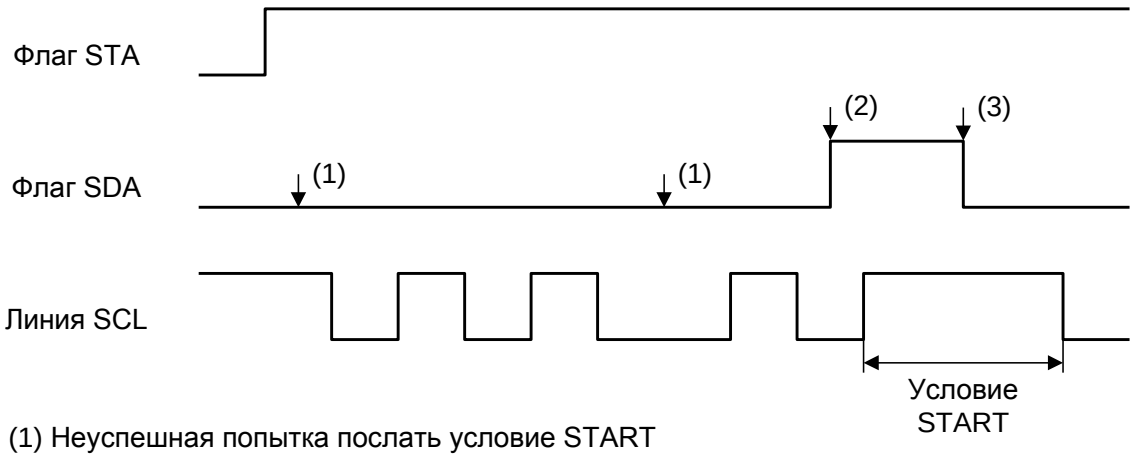
Если неконтролируемый источник генерирует ненужный START или маскирует условие STOP, тогда I^2C-шина остается занятой неопределенное время. Если доступ к шине не обеспечивается в течение разумного промежутка времени, возможен принудительный доступ. Никакого условия STOP в этом случае не передается. Аппаратные средства I^2C ведут себя так, как если бы условие STOP было принято, и способны передавать условие START.

Блокировка I^2C-шины низким уровнем SCL или SDA. Неожиданный останов I^2C-шины происходит, если на линии SDA или SCL неконтролируемый источник установит логический 0. Если линия SCL блокируется (подключается к низкому уровню) прибором на шине, дальнейшая последовательная передача невозможна и аппаратные средства I^2C не могут решить проблему этого типа. Когда это происходит, проблема должна быть решена со стороны того прибора, который подключил линию SCL к низкому уровню.

Если линия SDA блокируется другим прибором на шине (например, подчиненным прибором вне бита синхронизации), проблема может быть решена путем передачи дополнительных тактовых импульсов на линию SCL (рис. 24). Аппаратные средства интерфейса пытаются генерировать условие START после каждых двух дополнительных тактовых импульсов на линии SCL. Когда линия SDA в конце концов освобождается, передается нормальное условие START и последовательная передача продолжается.

Если происходит принудительный доступ к шине или передается повторное условие START, в то время когда линия SDA заблокирована (подключена к низкому уровню), аппаратные средства выполняют действия, аналогичные вышеописанным.

Ошибка шины. Ошибка шины возникает в случае, если условия START или STOP устанавливаются в неправильной позиции по отношению к разрядной сетке. Неправильные позиции возникают во время последовательной передачи адресного байта, данных или бита подтверждения. Аппаратные средства I^2C реагируют на ошибку шины только тогда, когда они участвуют в последовательной передаче в главном или в адресуемом подчиненном режимах. Когда детектируется ошибка шины, интерфейс сразу же переключается в неадресуемый подчиненный режим, освобождает линии SDA и SCL и устанавливает флаг прерывания. Далее производится попытка вновь повторить прерванную последовательную передачу.



(1) Неуспешная попытка послать условие START
(2) Линия SDA освободилась
(3) Успешная попытка послать условие START

Рис. 24. Возврат из блокировки шины, вызванной низким уровнем на SDA.

В качестве завершения обсуждения протокола I^2C следует заметить, что в области локальных сетей МК он достаточно активно замещается протоколом CAN, а в области связи с периферийными БИС он продолжает активно использоваться по причине наличия периферийных БИС с высокими техническими характеристиками.

3.5 Протоколы нижнего уровня CAN.

Общая характеристика. Разработанный в середине 80-х гг. фирмой Bosch для систем управления узлами автомобиля протокол CAN (Controller Area Network – сеть контроллеров) является последовательным протоколом высокоскоростной и высоконадежной передачи данных в широковещательном (broadcast) режиме и мультимастерной среде [2]. Удачное сочетание низкой стоимости подключения, простоты и надежности с проверкой временем и широкой доступностью элементной базы и инструментальных средств разработки – одни из основных достоинств CAN-технологии.

Положения, закрепленные в используемой на сегодня спецификации 2.0 A/B фирмы Bosch и международном стандарте ISO 11898, соответствуют двум начальным уровням (физическому и канальному) 7-уровневой модели взаимодействия открытых систем ISO/OSI. Ряд оригинальных технических решений, реализованных при разработке протокола, наилучшим образом позволили сориентировать его на решение задач контроля и управления.

Шинная топология, являющаяся основой CAN, требует наличия механизма адресации узлов, однако в CAN нет адресов как таковых: сообщение принимается всеми узлами. Любое передаваемое сообщение имеет определяющий его содержание уникальный идентификатор (ID), на основании которого каждый узел фильтрует "свои" сообщения и "решает" – реагировать или нет на сообщение, транслируемое в данный момент. Неоспоримыми преимуществами отсутствия адресации являются теоретически неограниченное количество узлов и простота их добавления и отключения.

Физическая среда передачи данных в CAN может быть самой разной – витая пара, плоский кабель, оптоволокно, а также радио и ИК каналы и даже линии электропередач. Основным ограничением протяженности шины является лишь предельно допустимая суммарная задержка распространения сигнала для заданной скорости передачи (в кабеле, трансиверах, входных цепях контроллеров и т.д.). В соответствии с рекомендациями ISO11898 при использовании стандартных трансиверов и быстродействующих оптопар (для гальванической развязки) максимальная протяженность сети при скорости передачи 1 Мбит/с ограничена девятью метрами. Предельная рекомендуемая протяженность сети в соответствии с тем же стандартом достигается при снижении скорости передачи до 50 кбит/с. А в документах промышленной CAN-группы CiA (CAN in Automation) приведены следующие полученные практическим путем соотношения скорость-протяженность для проводной сети без гальванической развязки: 1 Мбит/с – 30 м; 500 кбит/с – 100 м; 125 кбит/с – 500 м; 20 кбит/с – 2500 м; 10 кбит/с – 5000 м.

Форматы передаваемых сообщений. Сообщения, передаваемые по CAN-шине, именуются кадрами. Форматы кадров передаваемых данных приведены на **рис. 25, 26**. В зависимости от инициатора передачи и ее цели, существует четыре типа кадров:

- ✓ Data Frame – кадр данных;
- ✓ Remote Frame – кадр запроса данных;
- ✓ Error Frame – кадр ошибки;
- ✓ Overload Frame – кадр перегрузки.

Рис. 25. Кадры данных и запроса данных.

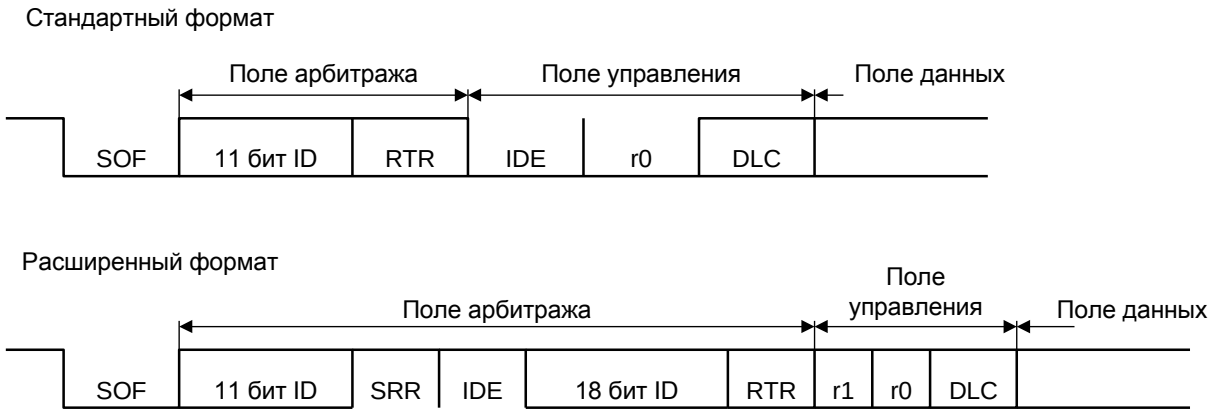


Рис. 26. Два формата кадров.

Собственно для передачи данных используется **Data Frame**, в поле данных которого (Data Field) могут находиться от 0 до 8 байт данных.

Поле арбитража (Arbitration Field) кадра включает в себя идентификатор (ID), однозначно определяющий содержание и приоритет сообщения. Стандартным форматом сообщений (CAN Specification 2.0 A) предусмотрен 11-битный идентификатор, позволяющий различать до 20348 типов сообщений (на практике обычно до 2032), а расширенный (CAN Specification 2.0 B) – 29-битный (стандартный 11-битный с 18-битным расширением) с теоретически возможным числом типов сообщений более 536 млн. Кадры, соответствующие стандартному и расширенному форматам сообщений, приведены на **рис. 26**.

Бит RTR (Remote Transmission Request – запрос передачи данных) для кадра должен иметь доминантный уровень. В расширенном формате кадра бит SRR (Substitute Remote Request) с рецессивным уровнем заменяет следующий (в стандартном формате) за 11-разрядным идентификатором бит RTR. Бит распознавания формата кадра IDE (ID Extension) имеет доминантный уровень для стандартного формата кадра и рецессивный – для расширенного. Биты r0 и r1 – резервные.

В поле управления (Control Field) содержится 4-разрядный код, задающий длину поля данных (0...8 байт), – DLC – Data Lenght Code. Поле контрольной суммы – CRC Field – включает в себя контрольную сумму сообщения (15 бит) и бит-разделитель. В поле подтверждения ACK (Acknowledgement) передающий узел всегда выставляет рецессивный

уровень. В случае если передача прошла успешно, приемный узел сигнализирует об этом установкой в этом поле доминантного уровня.

Начинается кадр доминантным битом SOF (Start of Frame), служащим также для синхронизации битового потока, а заканчивается семью битами рецессивного уровня поля EOF (End of Frame) и 3-битным того же уровня промежутком между кадрами. Для исключения потери синхронизации при передаче длинной последовательности одинаковых битов в пределах полей начала кадра, арбитража, управления, данных и контрольной суммы используется битстаффинг – вставка дополнительного бита противоположного значения после подряд идущих пяти одинаковых. При приеме производится обратная операция (дебитстаффинг).

Для запроса данных от удаленного узла служит кадр запроса данных **Remote Frame**, также имеющий стандартный и расширенный форматы. Отличия кадра запроса данных от кадра данных – в отсутствии поля данных и рецессивном уровне бита RTR. При получении кадра запроса данных запрашиваемый узел отвечает передачей кадра данных.

Сигнализация об ошибках происходит посредством передачи кадра **Error Frame**, формат которого приводится на **рис. 27**. Он инициируется любым узлом (в CAN правильность передачи контролируется каждым узлом), обнаружившим ошибку.

Шесть доминантных бит флага ошибки (активный флаг ошибки) перекрывают остаток ошибочно переданного кадра и создают глобальную ошибку в сети – ошибку битстаффинга, которая воспринимается остальными узлами, если им не удалось обнаружить первоначальную (локальную) ошибку. Далее они выставляют свои флаги ошибки. Ввиду этого обстоятельства, последовательность доминантных бит (суперпозиция флагов ошибки) может иметь длину от 6 до 12 бит. Ненадежным или частично поврежденным узлам (см. ниже) при обнаружении ошибки разрешено посылать лишь пассивный флаг ошибки – последовательность шести рецессивных бит.

Для задержки передачи данных или посылки кадра запроса данных (при неготовности приемника или наличии доминантных бит в промежутке между кадрами) служит кадр перегрузки **Overload Frame**. В отличие от кадра ошибки он не влияет на счетчик ошибок (см. ниже) и не вызывает повторную передачу сообщения.

Обнаружение коллизий и арбитраж. Несколько необычно решается проблема коллизий (столкновений в сети), присущая шинной топологии. В этом случае снова используется идентификатор сообщения в сочетании со схемой подключения к шине типа "монтажное ИЛИ", где узел, выставляющий на шину "0" – доминантный уровень, подавляет "1" – рецессивный уровень, выставленный другим узлом. Победителем в арбитраже является узел, имеющий идентификатор с наименьшим численным значением и, как следствие, наивысший приоритет. Только победивший узел продолжает передачу данных, остальные пытаются сделать это позже. На рис. 28 наглядно отражена процедура арбитража при попытке передачи сообщений одновременно четырьмя узлами сети.

Подобный режим доступа к шине известен как CSMA/CD+AMP (Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority) – множественный доступ с контролем несущей, обнаружением коллизий и арбитражем на основе приоритета сообщений. Этот режим не позволяет поспорившим узлам устраивать столкновение на шине, а сразу выявляет победителя.

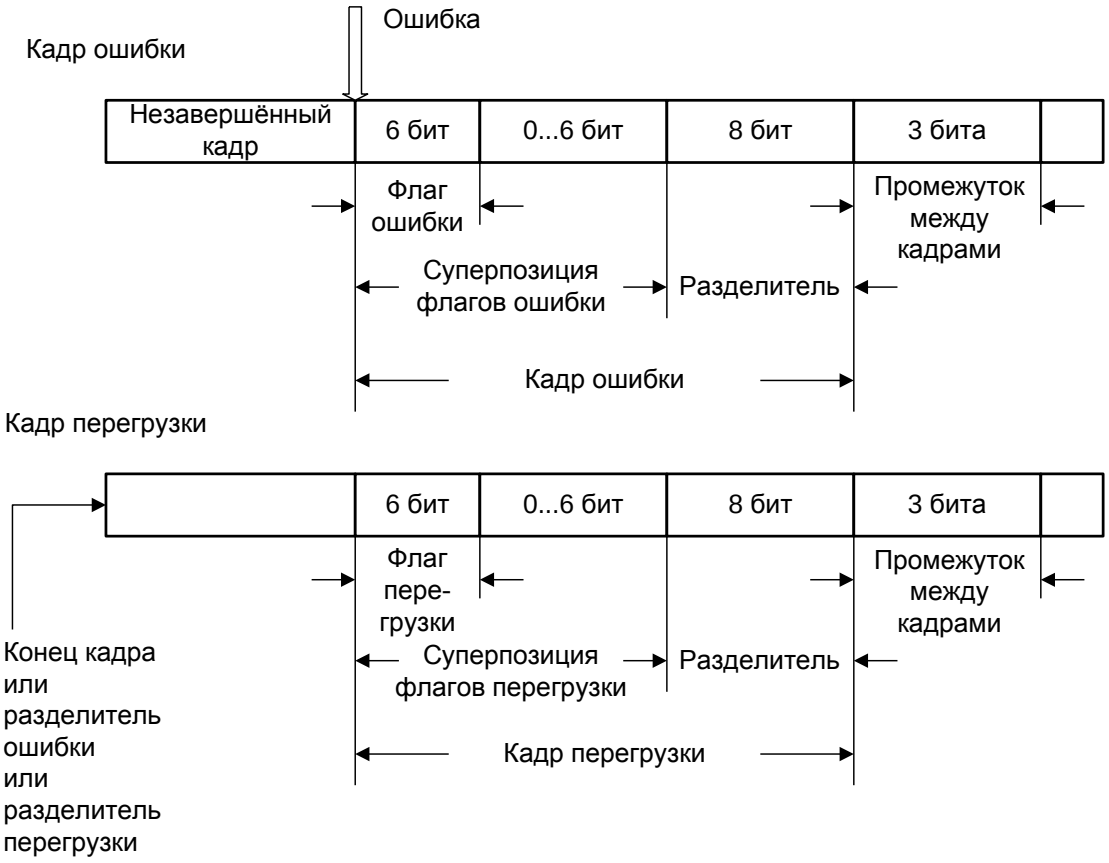


Рис. 27. Кадры ошибки и перегрузки.

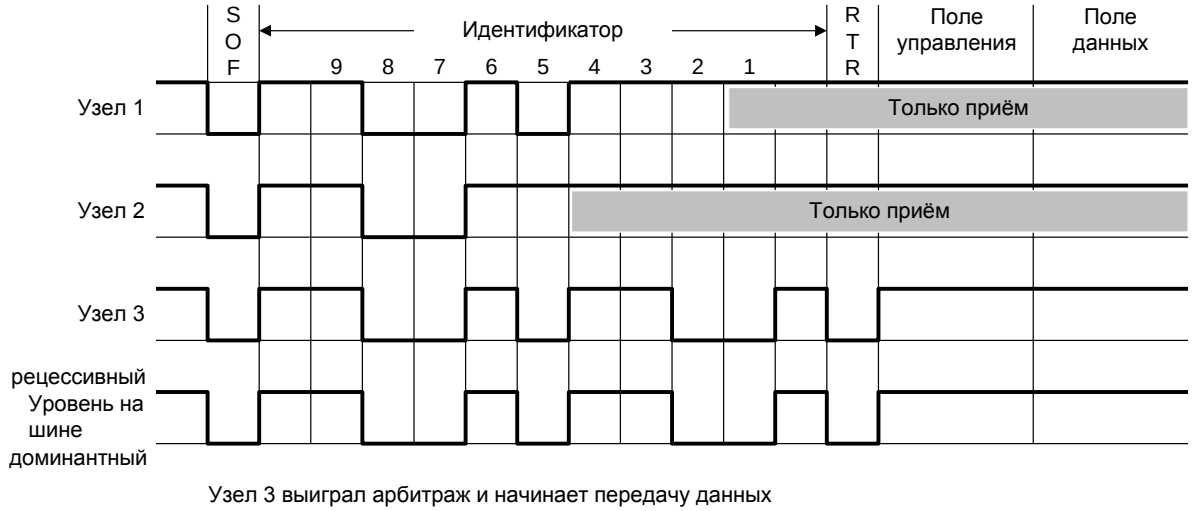


Рис. 28. Процедура арбитража.

Обнаружение ошибок и "живучесть" сети. CAN-протокол, изначально разработанный специально для систем управления жизненно важными узлами автомобилей, критичных к уровню безопасности и степени достоверности передаваемых данных, обладает эффективными средствами обнаружения ошибок.

В отличие от других сетевых протоколов, в CAN не используются подтверждающие сообщения, а при обнаружении одной или более ошибок хотя бы одним узлом (в CAN все узлы принимают все сообщения и участвуют в проверке сообщения на наличие ошибок – вычисляют контрольную сумму и т.п.) текущая передача прерывается (при условии, что ошибку обнаружил как минимум один узел со статусом Error Active) генерацией кадра ошибки с флагом ошибки. Передатчик, сообщение которого было прервано, повторяет передачу.

В CAN обнаруживается три разновидности ошибок на уровне сообщений:

- ✓ **CRC Error.** Ошибка контрольной суммы – несовпадение вычисленной и произведенной в поле CRC контрольной суммы сообщения.
- ✓ **Frame Error.** Ошибка формата кадра – несоответствие форматов и размеров полей кадра стандартным.
- ✓ **Ack Error.** Ошибка подтверждения – неполучение передатчиком подтверждения приема (доминантный уровень) в поле АСК.

А также два типа ошибок на битовом уровне:

- ✓ **Bit Error.** Передающий узел обнаруживает (в любых полях, кроме полей арбитража и подтверждения) расхождение, например, из-за замыкания шины, между посланным в шину логическим уровнем и фактически на ней установленным (в CAN каждый передатчик имеет возможность контролировать свой собственный сигнал).
- ✓ **Stuff Error.** Нарушение правил битстаффинга в сообщении наличие в поле сообщения, подлежащего битстаффингу, последовательности из шести бит с одинаковым значением.

В целях предотвращения блокирования сети неисправными узлами и повышения ее "живучести", "права доступа" к шине любого узла могут меняться в зависимости от числа зарегистрированных им ошибок. Каждый узел сети содержит два внутренних счетчика: ошибок приема и ошибок передачи. В зависимости от текущих значений этих счетчиков, узел будет переключать себя в одно из нижеприведенных состояний.

Error Active – узел принимает полноправное участие в обмене данными и при обнаружении ошибки выставит активный флаг ошибки, прервав любое ошибочное сообщение.

Error Passive – узел участвует в обмене данными, но в случае обнаружения ошибки выставляет лишь пассивный флаг ошибки и не может, таким образом, прервать любое ошибочное сообщение, за исключением своего собственного. Кроме этого, такой узел между своими последовательными передачами будет выдерживать паузу не менее 11 рецессивных бит (Suspend Transmission).

Bus Off – узел не принимает участия в обмене данными и не оказывает никакого влияния на сеть (выходные драйверы отключены).

В процессе работы узел может переходить из состояния Error Active в состояние Error Passive и обратно (например, при изменении помеховой обстановки). Выход из состояния Bus Off возможен только после программного или аппаратного сброса контроллера.

4 Программирование микроконтроллеров на языке Си

4.1 Побитовые операторы

Побитовые операторы обращаются с переменными как с наборами битов, а не как с числами [3]. Эти операторы используются в тех случаях, когда необходимо получить доступ к отдельным битам данных, например при выводе графических изображений на экран. Побитовые операторы могут выполнять действия только над целочисленными значениями. В отличие от логических операторов, с их помощью сравниваются не два числа целиком, а отдельные их биты. Существует три основных побитовых оператора: И(&), ИЛИ(|) и исключающее ИЛИ(^). Сюда можно также отнести унарный оператор побитового отрицания (~), который инвертирует значения битов числа.

Побитовое И. Оператор & записывает в бит результата единицу только в том случае, если оба сравниваемых бита равны 1, как показано ниже:

Бит 0	**Бит 1**	**Результат**
0	0	0
0	1	0
1	0	0
1	1	1

Этот оператор часто используют для маскирования отдельных битов числа.

Побитовое ИЛИ. Оператор | записывает в бит результата единицу в том случае, если хотя бы один из сравниваемых битов равен 1, как показано ниже:

Бит 0	**Бит 1**	**Результат**
0	0	0
0	1	1
1	0	1
1	1	1

Этот оператор часто используют для установки отдельных битов числа.

Побитовое исключающее ИЛИ. Оператор ^ записывает в бит результата единицу в том случае, если сравниваемые биты отличаются друг от друга, как показано ниже:

Бит 0	**Бит 1**	**Результат**
0	0	0
0	1	1
1	0	1
1	1	0

Ниже показаны примеры использования этих операторов с шестнадцатеричными, восьмеричными и двоичными числами.

```
0xf1          &         0x35      результат 0x31
                                  (шестнадцатеричное)
0361          &         0065      результат 061 (восьмеричное)
11110011      &         00110101  результат 00110001 (двоичное)

0xf1          |         0x35      результат 0xf5
                                  (шестнадцатеричное)
0361          |         0065      результат 0365 (восьмеричное)
11110011      |         00110101  результат 11110111 (двоичное)

0xf1          ^         0x35      результат 0xc4
                                  (шестнадцатеричное)
0361          ^         0065      результат 0304 (восьмеричное)
11110011      ^         00110101  результат 11000110 (двоичное)
```

```
~0xf1                    результат 0xff0e
                         (шестнадцатеричное)
~0361                    результат 0177416
(восьмеричное)
~11110011                результат 11111111 00001100
                         (двоичное)
```

В последнем примере результат состоит из двух байтов, так как в процессе операции побитового отрицания осуществляется повышение целочисленности операнда – его тип преобразуется к типу с большей размерностью. Если операнд беззнаковый, то результат получается вычитанием его значения из самого большого числа повышенного типа. Если операнд знаковый, то результат вычисляется посредством приведения операнда повышенного типа к беззнаковому типу, выполнения операции ~ и обратного приведения его к знаковому типу.

4.2 Операторы сдвига

Языки C/C++ содержат два оператора сдвига: сдвиг влево (<<) и сдвиг вправо (>>) [3]. Первый сдвигает битовое представление целочисленной переменной, указанной слева от оператора, влево на количество битов, указанное справа от оператора. При этом освобождающиеся младшие биты заполняются нулями, а соответствующее количество старших битов теряется.

Сдвиг беззнакового числа на одну позицию влево с заполнением младшего разряда нулём эквивалентен умножению числа на 2, как показано в следующем примере:

Пример 1.

```
unsigned int valuel = 65;     // младший байт: 0100 0001
valuel <<= 1;                 // младший байт: 1000 0010
```

При правом сдвиге происходят аналогичные действия, только битовое представление числа сдвигается на указанное количество битов вправо. Значения младших битов теряются, а освобождающиеся старшие биты заполняются нулями в случае беззнакового операнда и значением знакового бита в противной ситуации. Таким образом, сдвиг беззнакового числа на одну позицию вправо эквивалентен делению числа на два:

Пример 2

```
unsigned int valuel = 10;     // младший байт: 0000 1010
valuel >>= 1;                 // младший байт: 0000 0101
```

4.3 Указатели

Суть концепции указателей состоит в том, что вы работаете с адресом ячейки памяти, получая лишь косвенный доступ к ее содержимому, благодаря чему появляется возможность динамически создавать и уничтожать переменные. Хотя с помощью указателей можно создавать чрезвычайно эффективные алгоритмы, сложность программы, в которой используются указатели, значительно возрастает.

В языках C/C++ вопросы, связанные с указателями, массивами и строками, тесно связаны друг с другом. Для начинающего программиста глава может показаться чересчур сложной, но лишь в полной мере изучив тему указателей, можно начать создавать действительно профессиональные приложения.

4.3.1 Указатели как особый тип переменных

Начинающие программисты, как правило, работают только с обычными переменными. Для таких переменных память выделяется автоматически при запуске программы или функции, в которой они объявлены, и удаляется также автоматически при завершении программы или функции. Доступ к ним организуется достаточно просто – по имени [3]:

```
imeraorycell_contents += 10;
```

Другой способ получения доступа к значению переменной заключается в использовании другой переменной, которая содержит ее адрес. Предположим, имеется переменная типа `int` с именем `imemorycell_contents` и переменная `pimemor_cell_address`, являющаяся указателем на нее. Как вы уже знаете, в C/C++ есть оператор взятия адреса `&`, который возвращает адрес своего операнда. Поэтому вам будет нетрудно разобраться в синтаксисе присвоения одной переменной адреса другой переменной:

```
pimemorycselI_address = &imemorycell_contents;
```

Переменные, которые хранят адреса других переменных, называются **указателями.** Например, переменная imemoirycell_contents представлена в памяти компьютера ячейкой с адресом 7751. После выполнения показанной выше строки программы адрес этой переменной будет присвоен указателю pimemorycell_address.

Обращение к переменной, чей адрес хранится в другой переменной, осуществляется путем помещения перед указателем оператора `*`: `*pimemorycell_address`. Такая запись означает, что будет произведен косвенный доступ к ячейке памяти через имя указателя, содержащего адрес ячейки. Например, если выполнить две показанные ниже строки, то переменная `imemorycell_contents` примет значение 20:

```
pimemoryciell_address = &imemorycell_contents;
*pimemorycell_address = 20;
```

С учетом того, что указатель `pimemorycell_address` хранит адрес переменной `irnemorycell_contents`, обе следующие строки приведут к одному и тому же результату: присвоению переменной `imemorycell_contents` значения 20.

```
imemorycell_contents = 20;
*pimemorycell_address = 20;
```

4.3.2 Объявление указателей

В языках C/C++ все переменные должны быть предварительно объявлены. Объявление указателя `pimemorycell_address` выглядит следующим образом:

```
int *pimemorycell_address;
```

Символ `*` говорит о том, что создается указатель. Этот указатель будет адресовать переменную типа `int`. Следует подчеркнуть, что в C/C++ указатели могут хранить адреса только переменных конкретного типа. Если делается попытка присвоить указателю

одного типа адрес переменной другого типа, возникнет ошибка либо во время компиляции, либо во время выполнения программы.

Рассмотрим пример 3:

```
int   *pi
float real_value = 98.26;
pi = &real_value ;
```

В данном случае переменная `pi` объявлена как указатель типа `int`. Но в третьей строке делается попытка присвоить этому указателю адрес переменной `real_value`, имеющей тип `float`. В результате компилятор выдаст предупреждение вида "несовместимые операнды в операции присваивания", а программа, использующая указатель `pi`, будет работать неправильно.

4.3.3 Использование указателей

В следующем фрагменте программы посредством указателей производится обмен значений между переменными `iresult_a` и `iresult_b`:

Пример 4

```
int iresult_a = 15, iresult_b = 37, iternporary;
int *piresult;

piresult = &iresult_a;
iteraporary = *piresult;
*piresult = iresult_b;
iresult_b = itemporary;
```

Первая строка содержит традиционные объявления переменных. При этом в памяти компьютера резервируются три ячейки для хранения целочисленных значений, каждой ячейке присваивается имя, и две из них инициализируются начальными значениями. Предположим, что переменная `iresult_a` хранит свои значения в ячейке с адресом 5328, переменная `iresult_b` связана с ячейкой 7916, а `itemporary` – с ячейкой 2385.

Во второй строке программы создается указатель `piresult`. При этом также происходит резервирование именованной ячейки памяти (скажем, с адресом 1920). Поскольку инициализация не производится, то в данный момент указатель содержит пустое значение. Если попытаться применить к нему оператор `*`, то компилятор не сообщит об ошибке, но и не возвратит никакого адреса.

В третьей строке происходит присваивание указателю `piresult` адреса переменной `iresult_a`.

В следующей строке в переменную `itemporary` записывается содержимое переменной `iresult_a`, извлекаемое с помощью выражения `*piresult`:

```
itemporary = *piresult;
```

Таким образом, переменной `itemporary` присваивается значение 15. Если перед именем указателя `piresult` не поставить символ `*`, то в результате в переметшую `itemporary` будет ошибочно записано содержимое самого указателя, т.е. 5328. Это очень коварная ошибка, поскольку многие компиляторы не выдают в подобных ситуациях предупреждений или сообщений об ошибке. Компилятор Visual C++ отобразит предупреждение вида "разные уровни косвенной адресации в операции присваивания".

В пятой строке содержимое переменной `iresult_b` копируется в ячейку памяти, адресуемую указателем `piresult`:

```
*piresult = iresult_b;
```

В последней строке число, хранящееся в переменной `itemporary`, просто копируется в переменную `iresult_b`.

В следующем фрагменте программы демонстрируется возможность манипулирования адресами, хранящимися в указателях. В отличие от предыдущего

примера, где переменные обменивались значениями, здесь осуществляется обмен адресами переменных.

Пример 5

```
char cswitch1 = 'S', cswitch2 = 'T';
char *pcswitchl, *pcswitch2, *pctemporary;

pcswitch1 = &cswitch1;
pcswitch2 = &cswitch2;
pctemporary = pcswitch1;
pcswitchl = pcswitch2;
pcswitch2 = pctemporary;
printf("%c%c", *pcswitch1, *pcswitch2);
```

В третьей и четвёртой строчках указателям `pcswitch1` и `pcswitch2` присваиваются адреса переменных `cswitch1` и `cswitch2` соответственно.

В пятой строке содержимое указателя `pcswitch1` копируется в переменную `pctemporary`, в результате чего оба указателя адресуют одну переменную: `cswitchl`.

В следующей строке содержимое указателя `pcswitch2` копируется в указатель `pcswitchl`, после чего оба будут содержать адрес переменной `cswitch2`:

```
pcswitchl = pcswitch2;
```

Обратите внимание, что если бы содержимое указателя `pcswitchl` не было продублировано во временной переменной `pcteraporary`, то в результате выполнения предыдущего выражения ссылка на адрес переменной `cswitchl` была бы утеряна.

В предпоследней строке происходит копирование адреса из указателя `pctemporary` в указатель `pcswitch2`. В результате работы функции printf () получаем:

```
TS
```

Заметьте: в ходе выполнения программы исходные значения переменных `cswitchl` и `cswitch2` не изменялись. Описанный метод может пригодиться вам в дальнейшем, так как, в зависимости от размеров объектов, часто бывает проще копировать их адреса, чем перемещать содержимое.

4.3.4 Инициализация указателей

Указатели можно инициализировать при их объявлении, как и любые другие переменные. Например, в следующем фрагменте создаются две именованные ячейки памяти: `iresult` и `piresult`.

```
int iresult;
int *piresult = &iresult;
```

Идентификатор `iresult` представляет собой обычную целочисленную переменную, а `piresult` – указатель на переменную типа `int`. Одновременно с объявлением указателя `piresult` ему присваивается адрес переменной `iresult`. Будьте внимательны: здесь инициализируется содержимое самого указателя, т.е. хранящийся в нем адрес, но не содержимое ячейки памяти, на которую он указывает. Переменная `iresult` остается неинициализированной.

В приведенной ниже программе объявляется и инициализируется указатель на строку-палиндром, одинаково читаемую как слева направо, так и справа налево:

Пример 6

```
/*
*    psz.c
*    Эта программа на языке С содержит пример
*    инициализации указателя.
*/
#include <stdio.h>
```

```
#include <string,h>
void main()
{
     char *pszpalindrome = "Дом мод";
     int i;
     for(i = strlen(pszpalindrome) - 1;i >= 0; i--)
     printf("%c", pszpalindrome[i]);
     printf("%s", pszpalindrome);
}
```

В указателе `pszpalindrome` сохраняется адрес только первого символа строки. Но это не значит, что оставшаяся часть строки пропадает: компилятор заносит все строковые константы, обнаруженные им в программе, в специальную скрытую таблицу, встраиваемую в программу. Таким образом, в указатель записывается адрес ячейки таблицы, связанной с данной строкой.

Функция `strlen()`, объявленная в файле STRING.H, в качестве аргумента принимает указатель на строку, заканчивающуюся нулевым символом, и возвращает число символов в строке, не считая последнего. В нашем примере строка состоит из семи символов, но счетчик цикла `for` инициализируется значением 6, поскольку строка в нем интерпретируется как массив, содержащий элементы от нулевого до шестого. На первый взгляд, может показаться странной взаимосвязь между строковыми указателями и массивами символов, но если мы вспомним, что имя массива по сути своей является указателем на первый элемент массива, то станет понятным, почему переход от имени указателя к имени массива в программе не вызвал возражений компилятора.

4.3.5 Ограничения на использование оператора &

Оператор взятия адреса (&) можно применять далеко не с каждым выражением. Ниже иллюстрируются ситуации, когда оператор & используется неправильно:

Пример 7

```
/*
     с константами
*/
     pivariable = &48;
/*
     в выражениях с арифметическими операторами
*/
int iresult = 5;
pivariable = &(iresult + 15);
/*
     с переменными   класса   памяти   register
*/
register int   register1;
pivariable = &register1;
```

В первом случае делается недопустимая попытка получить адрес константного значения. Поскольку с константой 48 не связана ни одна ячейка памяти, операция не может быть выполнена.

Во втором случае программа пытается найти адрес выражения `iresult+15`. Поскольку результатом этого выражения является число, находящееся в программном стеке, его адрес не может быть получен.

В последнем примере объявляется регистровая переменная, смысл которой состоит в том, что предполагается частое её использование, поэтому она должна располагаться не в памяти, а непосредственно в регистрах процессора. Компилятор может проигнорировать подобный запрос и разместить переменную в памяти, но в любом случае операция взятия адреса считается неприменимой по отношению к регистровым переменным.

4.4 Обработчики прерываний

Пусть требуется осуществить обработку прерываний от таймера 0 MCS51 (прерывание 0Bh). В программу на языке Си (компилятор фирмы Keil Software) можно вставить следующий код:

Пример 8

```
void T0_ISR(void) interrupt 1      // Обработчик прерывания
                                   //от таймера 0
{
          // Действия, выполняемые обработчиком
}
```

Цифра после `interrupt` указывает на номер прерывания таймера 0 в таблице векторов прерываний MCS51, а имя подпрограммы может быть любой.

В данном случае после трансляции программы по адресу `0Bh` будет расположена команда:

```
jmp addr
```

где `addr` – адрес начала программы – обработчика прерывания от таймера 0 (непосредственно подпрограммы `T0_ISR`).

Из примера 8 видно, что установка обработчика прерывания для микроконтроллеров MCS51 в Си фирмы Keil Software выполняется достаточно просто. Однако, в изучаемом в курсе лабораторных работ по микропроцессорной технике учебно-лабораторном стенде вектора прерываний необходимо располагать в пользовательской таблице. В результате чего в данную таблицу вектор прерывания необходимо помещать самостоятельно, например, при помощи следующей функции.

Пример 9. Функция, устанавливающая вектор прерывания в ОЗУ.

```
#define RAM_TABLE 0x2000 // начало пользовательской таблицы
// векторов прерываний
#define LJMP 0x2 //код команды LJMP
void SetVect(unsigned char num, voide code *handler)
{
     // расчёт адреса вектора прерывания (адреса куда необходимо
     // установить команду перехода)
     unsigned char xdata *p = RAM_TABLE + (num<<3) + 3;
     *p++ = LJMP; // запись по адресу p команды LJMP
     // запись адреса перехода (метки) для команды LJMP
     *(unsigned short xdata *)p = (unsigned short)handler;
}
```

Вызов данной функции можно выполнять следующим образом. Для случая примера 8 получаем:

Пример 10

```
void main(void)
{
     /* … */
     SetVect(1,T0_ISR); // установка вектора в //пользовательской
таблице
     ET0 = 1;  // разрешение прерывания от таймера 0
     EA = 1;   // разрешение прерываний
}
```

5 Примеры программ ввода/вывода информации для учебно-лабораторного стенда SDK-1-1.

Ниже перечисленные примеры выполнены с применением пользовательских функций работы с периферией учебно-лабораторного стенда SDK-1-1. Данные функции можно найти в файлах по следующему пути:

L:\Study\МПС\SDK_1_1\EXAMPLES

5.1 Работа с UART с использованием обработчика прерывания

Ниже приведён пример работы с последовательным портом ввода/вывода с использованием обработчика прерывания.

Пример 11

```
#include <ADuC812.h>
#include "max.h"
#include "vect.h"

unsigned char buf;
bit flagRi;

// Обработчик прерывания от UART
void isr_uart(void) interrupt 4
{
     if(RI == 1)
     {
          RI = 0;
          buf = SBUF;
          flagRi = 1;
     }
     if(TI == 1)
     {
          TI = 0;
     }
}

void main(void)
{
     WriteMax(0x7,0xAA); // поджег светодиодов

     //-----------Настройка UART ---------------
     SCON = 0x40;
     REN = 1; /* Разрешение приёма */
     // ------ Настройка таймера 2
     RCLK = 1;
     TCLK = 1; /* Тактовые импульсы от таймера 2 */
     RCAP2L = 0xDC;
     RCAP2H = 0xFF; /* Скорость 9600 бит/сек  */
     TR2 = 1; /*запуск таймера 2 */
     ES = 1;   /* Разрешение прерывания от uart */
     EA = 1;
     //----------------------------------------
     buf = ':';
     flagRi = 0;
     TI = 0;RI = 0;
```

```
     SetVect(4,isr_uart); //установка вектора прерывания
                              //от   UART   в   пользовательской
таблице
     SBUF = buf;
     while(1)
     {
          if(flagRi == 1)
          {
               SBUF = buf;
               flagRi = 0;
          }
     }
}
```

Разберитесь с программой примера 11, допишите недостающие комментарии, начертите блок-схему программы.

Модифицируйте программу так, чтобы при приёме байта загорался светодиод №1, а при передаче - №2.

5.2 Работа с внешним EEPROM по шине I²C

Ниже приведён пример программы работы с внешним EEPROM, который подключен к микроконтроллеру ADuC812 по шине I²C. Данная программа производит запись строки “hello” в EEPROM по адресу 0x10h, производит чтение и проверяет идентичность записанной, и считанной информации.

Пример 12

```
#include <ADuC812.h>
#include "max.h"
#include "I2C.h"
#include "eeprom.h"

void main(void)
{
bit Weeprom;
bit Reeprom;
unsigned char xdata buf[5] = "hello";
unsigned char xdata buf2[5]="";
unsigned char length;
unsigned char i;

     Weeprom = 0;
     Reeprom = 0;

     WriteMax(0x7,0xAA);

     length = 5;
     //Запись строки buf в EEPROM по I2C
     Weeprom = WriteBlockEEPROM(0x10,buf,length);
     if(Weeprom == 1) WriteMax(0x7,0x01); // ошибка записи
     //Чтение строки из EEPROM по адресу 0x10 в buf2
     Reeprom = ReadBlockEEPROM(0x10,buf2,length);
     if(Reeprom == 1) WriteMax(0x7,0x02);// ошибка чтения

     //Проверка строк на идентичность
     for(i = 0;i < 5;i++)
          {
               if(buf2[i]!=buf[i]) WriteMax(0x7,0xff);
          }

     while(1);
}
```

Разберитесь с программой примера 12, допишите недостающие комментарии, начертите блок-схему программы. Изучите работу функций WriteBlockEEPROM и **ReadBlockEEPROM. Изучите модули eeprom.c и i2c.c.**

5.3 Работа с ЖКИ и клавиатурой стенда

Ниже приведён пример программы работы с матричной клавиатурой и ЖКИ учебно-лабораторного стенда SDK-1-1. Данная программа производит считывание клавиши и отображает её на ЖКИ.

Пример 13

```
#include <ADuC812.h>
#include "max.h"
#include "kb.h"
#include "lcd.h"

void main(void)
{
unsigned char ch;
unsigned char bufch;
bit kbOnce;

     kbOnce = 0;
     ch = 0;
     bufch = 0;

     InitLCD(); // Инициализация ЖКИ
     LCD_Clear(); // Очистка экрана ЖКИ

     WriteMax(0x7,0xAA);
     while(1)
     {
          while(kbOnce == 0)
          {
               bufch = ch;
               kbOnce = ScanKBOnce(&ch); // Чтение клавиши
          }
          kbOnce = 0;
          WriteMax(0x7,0xAA);
          if(ch != bufch) // Защита от дребезга контактов
               {
                    LCD_Putch(ch);
                    WriteMax(0x7,0x00);
               }
          if(ch == '#') LCD_Clear(); //Очистка экрана
     }
}
```

Разберитесь с программой примера 13, допишите недостающие комментарии, начертите блок-схему программы. Изучите работу функций ScanKBOnce и LCD_Putch. Изучите модули kb.c и lcd.c.

Доработайте программу так, чтобы отображался курсор, и можно было набирать один и тот же символ более чем один раз.

6 Индивидуальное задание

Вариант индивидуального задания указывается преподавателем.

7 Содержание отчёта

1. Цель работы.
2. Структурная схема параллельного интерфейса связи микроконтроллера ADuC812 с ПЛИС и внешней ОЗУ.
3. Структурная схема контроллера последовательной синхронной передачи и приёма с кратким описанием.
4. Структурная схема контроллера последовательной асинхронной передачи и приёма с кратким описанием.
5. Формат информационного кадра для асинхронного режима.
6. Последовательные интерфейсы встраиваемых микропроцессорных систем

Структурная схема сопряжения МК и периферийных ИС с использованием UART с кратким описанием.

Структурная схема сопряжения МК и периферийных ИС с использованием интерфейса SPI с кратким описанием.

Структурная схема сопряжения МК и периферийных ИС с использованием интерфейса I^2C с кратким описанием.

7. Результаты выполнения заданий

Листинг примеров с комментариями и блок-схемами.

Листинг программы индивидуального задания с комментариями. Блок-схема программы.

8. Выводы по проделанной лабораторной работе.

8 Контрольные вопросы

1. Понятие интерфейса.
2. Параллельная передача данных. Шина данных. Шина адреса. Шина управления.
3. Минимально необходимый набор управляющих сигналов в синхронном параллельном интерфейсе.
4. Отличительные особенности асинхронной передачи в параллельном интерфейсе.
5. Каким образом в параллельном интерфейсе формируются сигналы начала и конца передачи.
6. Последовательный интерфейс. Отличительные особенности последовательного интерфейса от параллельного.
7. Синхронная последовательная передача данных.
8. Асинхронная последовательная передача данных.
9. Синхронный последовательный интерфейс. Основные элементы структуры интерфейса.
10. Асинхронный последовательный интерфейс. Основные элементы структуры интерфейса.
11. Приборный интерфейс.
12. Интерфейс локальной вычислительной сети.
13. Типы общепринятых кадров синхронного режима передачи данных.
14. Процедура битстаффинга.
15. Бит-ориентированные протоколы.
16. Байт-ориентированные протоколы.
17. Общепринятые кадры синхронного режима передачи данных.
18. Методы кодирования последовательной информации.
19. Организация физического уровня интерфейса RS-232C.
20. Организация физического уровня интерфейса RS-485.
21. Организация физического уровня CAN-сети.
22. Последовательный интерфейс RS-232C. Основные сигналы. Уровни сигналов.
23. Последовательный периферийный интерфейс SPI. Сигналы интерфейса и их уровни.
24. Сопряжение МК с периферийными ИС посредством интерфейса SPI.
25. Протоколы связи интерфейса SPI.
26. Синхронный последовательный интерфейс I2C. Сигналы интерфейса и их уровни.
27. Конфигурация I2C-шины.
28. Протокол связи I2C.
29. Адресация на шине I2C. Основные типы передачи данных.
30. Режимы работы I2C-логики.
31. Работа при конкуренции в I2C. Арбитраж. Синхронизация тактовых сигналов.
32. Интерфейс CAN. Особенности данного интерфейса.
33. Программно управляемый ввод-вывод.
34. Ввод-вывод, управляемый программой обработки прерываний.

Перечень источников

1. Горюнов А.Г. Ливенцов С.Н. Основы микропроцессорной техники L:\Study\МПС\Методички\мп.pdf

2. Ремизевич Т. В. Микроконтроллеры для встраиваемых приложений: от общих подходов – к семействам HC05 и HC08 фирмы Motorola. /под ред Кирюхина И. С. – М.: ДОДЭКА, 2000. – 272 с.

3. В. Кораблёв. С и С++ – К.: Издательская группа BHV, 2002. – 432 с.