ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
«ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Г.С. Воробьева, А.И. Селезнев

ИНТЕРФЕЙСЫ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Учебное пособие

Издательство
Томского политехнического университета
2008

ББК
УДК 681.325.5-181.48.001.63

Воробьева Г.С.

В Интерфейсы микропроцессорных систем: учебное пособие / Г.С. Воробьева, А.И. Селезнев. – Томск: Изд-во Томского политехнического университета, 2008. – 190 с.

Изложены теоретические основы построения сетей I2C, а так же описаны технические характеристики различных микросхем с данным интерфейсом, даны алгоритмы программирования встроенного блока I2C для микроконтроллеров семейств MCS-51, PIC, AVR.

Пособие содержит необходимые сведения (карты памяти, описания сред разработки, систем команд) для реализации данного интерфейса. Предназначено для бакалавров, магистрантов, аспирантов и инженеров, занимающихся проектированием микропроцессорных систем с I2C интерфейсом.

УДК 681.325.5-181.48.001.63

Рекомендовано к печати Редакционно-издательским советом
Томского политехнического университета

Рецензенты

Доктор ук, профессор ТГУ
Заведующий кафедрой медицинской и биологической
кибернетики ГОУВПО Сиб МУ РОС ЗДРАВа,
заслуженный работник высшей школы РФ, профессор
Пеккер Я.С.

Зам. нач. СКБ ОАО «Манотомь», доцент, к.т.н.
Бычков В.В.

Доцент, к.т.н.
Солдатов А.И.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Любое современное электронное устройство, как правило, содержит множество разнообразных микросхем, которые производят обмен и обработку информации. Использование параллельных шин для передачи данных приводит к увеличению числа соединительных контактов, проводников и мест пайки. Следствием этого является увеличение стоимости системы, габаритов, уменьшение её надежности. Кроме того, в высокой пропускной способности параллельных шин зачастую нет необходимости. По этой причине компанией Philips была разработана шина Inter-Integrated Circuit Bus (сокращенно I2C) – шина для передачи данных между интегральными схемами. Перечислим некоторые достоинства шины:

- имеется механизм адресации, что позволяет размещать несколько устройств на одной шине;
- интегральные схемы могут быть подсоединены к шине или отсоединены от нее без оказания какого-либо влияния на работу других подключенных устройств;
- не нужны драйверы линии и прочая интерфейсная логика;
- данные передаются только по двум линиям, благодаря чему требуется меньше соединений и минимизируются затраты;
- более высокая помехозащищенность по сравнению с параллельными интерфейсами;
- имеется возможность реализовать интерфейс чисто программными средствами.

Видно, что данная шина подходит для большинства приложений, не требующих передачи и обработки больших объемов данных.

ГЛАВА 1. УСТРОЙСТВО ШИНЫ I2C

§ 1.1. Скоростные режимы

На настоящий момент существует три режима работы шины, отличающиеся друг от друга максимальной скоростью передачи данных:

1. *Low-speed*
   Режим является стандартным и максимальная скорость передачи данных в этом режиме составляет 100 Кбит/с
2. *Fast-speed*
   Данный режим характеризуется максимальной скоростью 400 Кбит/с
3. *High-speed*
   Максимальная скорость передачи данных в этом режиме равна 3.4 Мбит/с

Следует отметить, что все режимы совместимы сверху вниз, т. е. устройство, поддерживающее режим работы High-speed, может нормально работать на одной шине с устройствами, поддерживающими режимы Fast-speed mode и Low-speed. Максимальная скорость передачи данных в этом случае будет определяться самым медленным устройством на шине I2C (если не используются специальные мосты).

§ 1.2. Структурная схема

Наиболее часто применяются системы с одним ведущим и несколькими ведомыми устройствами. Пример структурной схемы для данного случая приведен на рис. 1.1.

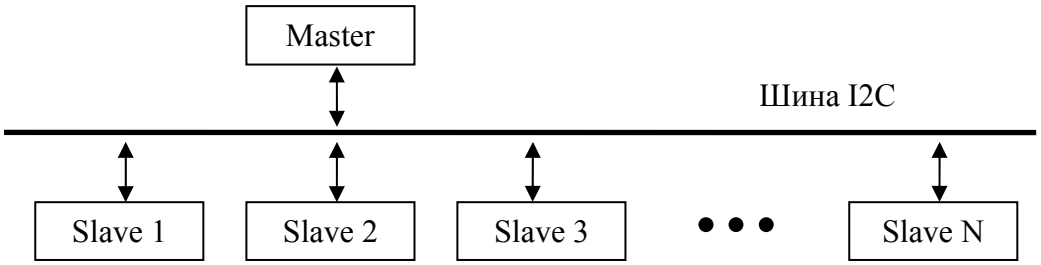


Рис. 1.1. Структурная схема системы с одним ведущим

Ведущим (Master) обычно является микроконтроллер. Он инициирует и контролирует обмен данными, формирует необходимые временные интервалы. Ведомыми (Slave) устройствами могут быть как микроконтроллеры, так другие микросхемы, например, часы реального времени, память EEPROM, различные датчики, декодеры и т. д.

Спецификация I2C также предусматривает возможность подключения к одной шине нескольких ведущих устройств. Структурная схема такой системы представлена на рис. 1.2.

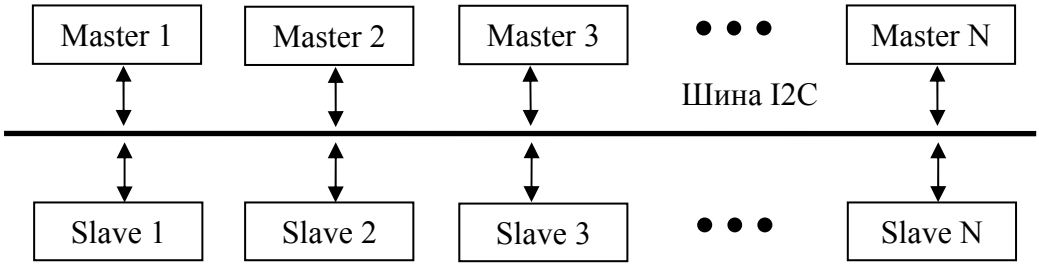


Рис. 1.2. Структурная схема системы с несколькими ведущими

Для обеспечения целостности данных и предотвращения конфликтов на шине при работе систем с несколькими ведущими в спецификации I2C предусмотрены процедуры арбитража и синхронизации.

§ 1.3. Аппаратная реализация

Как уже было сказано выше, шина I2C состоит из двух линий:

- SCL (Serial Clock Line) – линия последовательной передачи синхроимпульсов;
- SDA (Serial Data Line) – линия последовательной передачи данных.

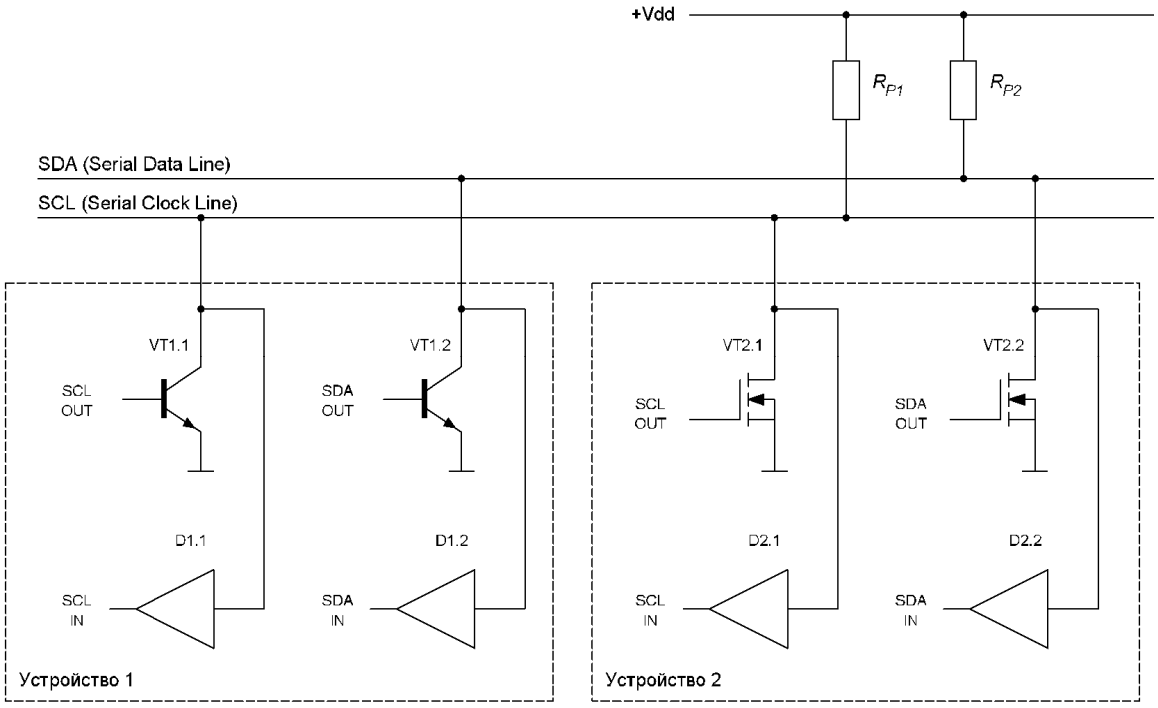


Рис. 1.3. Структура входных и выходных блоков

На рис. 1.3 приведен фрагмент схемы, на котором показаны два подключенных к шине устройства.

С точки зрения аппаратной реализации входных и выходных каскадов интерфейса нет разницы между Master-устройствами и Slave-устройствами. И те и другие имеют в своем составе по два каскада с открытым стоком/коллектором и два буферных усилителя с большим входным сопротивлением.

Резисторы R_{P1} и R_{P2} служат для формирования уровня лог. 1 в том случае, когда все транзисторы выходных каскадов, подключенных к линии, закрыты.

Если устройство 1 желает сформировать на линии SCL уровень лог. 0, оно открывает транзистор VT1.1 и на лини появляется сигнал низкого логического уровня. Устройство 2 узнает об изменении уровня на линии SCL по изменению сигнала SCL IN на выходе буферного усилителя D2.1.

Если же устройство 1 желает сформировать на линии SCL уровень лог. 1, то оно закрывает транзистор VT1.1 и за счет подтягивающего резистора R_{P1} на линии появляется высокий логический уровень (разумеется, при условии, что транзистор VT2.1 устройства 2 закрыт).

Из схемы, приведенной на рис. 1.3, видно, что все выводы устройств, подключенные к линии SCL (как, впрочем, и выводы, подключенные к линии SDA), объединены по схеме «монтажное И». Благодаря этому у ведомых устройств появляется возможность «затягивать» низкий уровень сигнала на линии SCL. Затягивание низкого уровня дает ведомому время на подготовку к приему либо передаче данных. Кроме того, особенности схемы «монтажное И» используются при арбитраже и синхронизации, которые описанны в 0.

Формирование высокого и низкого уровней на линии SDA происходит таким же образом, как и на линии SCL.

Значение сопротивления подтягивающих резисторов R_{P1} и R_{P2} зависит от следующих параметров:

- напряжения питания;
- емкости шины;
- количества подключенных к линии устройств (определяют входной ток и ток утечки).

Напряжение питания определяет минимальное сопротивление резисторов, т. к. максимальный ток через открытый транзистор выходного каскада любого подключенного к шине устройства не должен превышать 3 мА. На рис. 1.4 приведена зависимость минимального сопротивления подтягивающих резисторов от напряжения питания.

Емкость шины складывается из емкости проводника (по отношению к земле), емкости соединений и емкости подключенных к линии выво-

дов, и определяет максимально допустимое сопротивление подтягивающих резисторов, т. к. длительность фронта сигнала (которая увеличивается с ростом емкости) не должна превышать значения, определенного спецификацией. Ниже приведена зависимость максимального сопротивления подтягивающих резисторов от суммарной емкости шины.

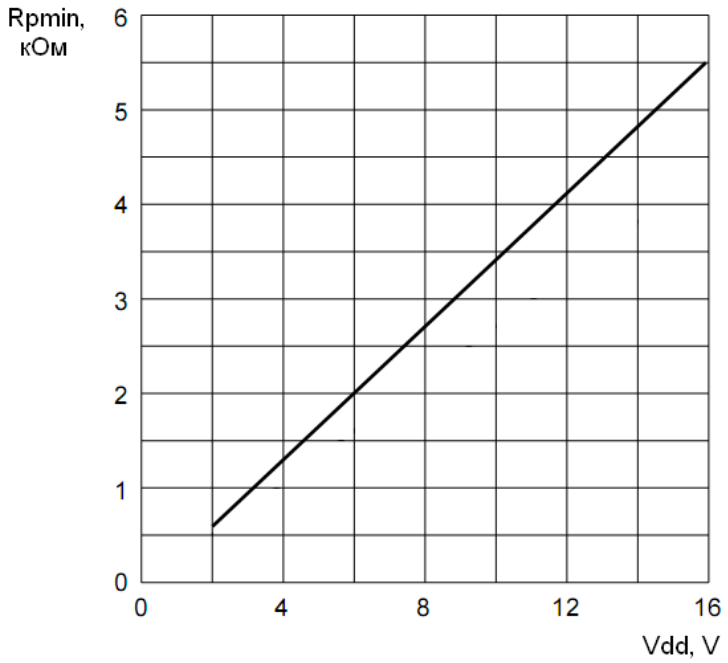


Рис. 1.4. Зависимость минимального сопротивления подтягивающих резисторов R_P от напряжения питания Vdd

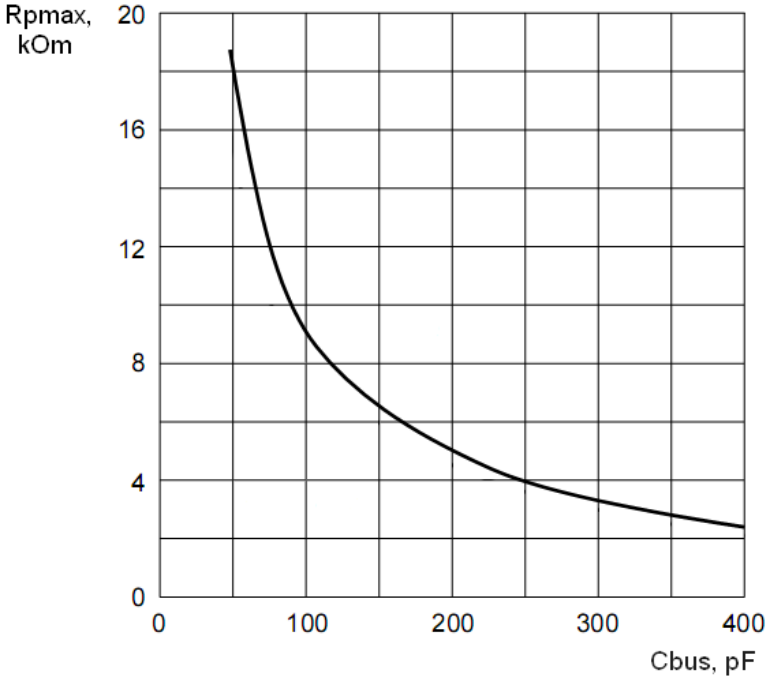


Рис. 1.5. Зависимость максимального сопротивления подтягивающих резисторов R_P от суммарной емкости шины Cbus

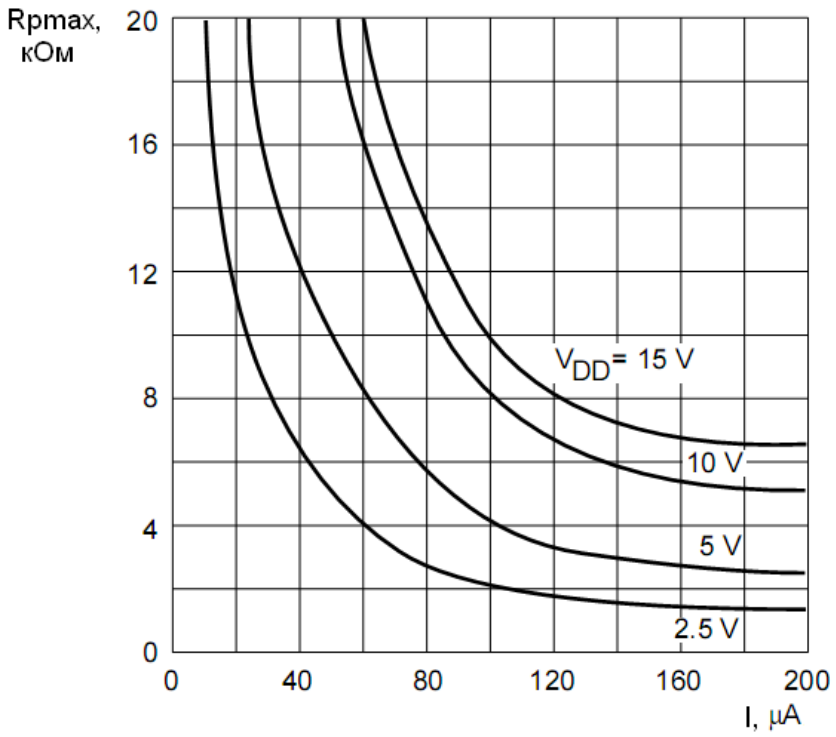


Рис. 1.6. Зависимость максимального сопротивления R_P от суммарного входного тока устройств, подключенных к шине

Спецификацией установлено, что входной ток каждого подключенного к шине вывода при высоком логическом уровне не должен превышать 10 мкА.

Таким образом, максимально допустимое сопротивление также зависит от суммарного входного тока устройств, подключенных к линии, и от напряжения питания. Данная зависимость представлена на рис. 1.6.

§ 1.4. Протокол обмена

Каждое устройство на шине распознается по его уникальному адресу и может работать как приемник или передатчик, в зависимости от выполняемых им функций. Также устройства могут быть классифицированы как ведущие и ведомые при передаче данных. Любое адресуемое устройство считается ведомым по отношению к ведущему. В табл. 1.1 приведены основные термины, используемые при описании шины.

Таблица 1.1

Основные термины

Термин	Описание
Transmitter (передатчик)	Устройство, посылающее данные на шину
Receiver (приемник)	Устройство, принимающее данные с шины

Окончание табл. 1.1

Термин	Описание
Master (ведущий)	Устройство, которое инициирует передачу данных, генерирует тактовый сигнал и заканчивает передачу
Slave (ведомый)	Устройство, адресуемое ведущим
Multi-master	Режим работы шины с несколькими ведущими. Несколько ведущих могут попытаться захватить шину одновременно, при этом не происходит потери передаваемых данных
Arbitration (арбитраж)	Процедура, которая обеспечивает контроль над шиной только одному из контроллеров при попытке одновременного захвата шины несколькими ведущими
Synchronization (синхронизация)	Процедура синхронизации тактового сигнала от двух или более устройств

Шина I2C поддерживает режим multi-master. Это означает, что к шине может быть подключено несколько устройств, способных управлять ею.

В качестве ведущих устройств обычно выступают микроконтроллеры, поэтому в качестве примера рассмотрим случай передачи данных между двумя микроконтроллерами, подключенными к шине (см. рис. 1.7).

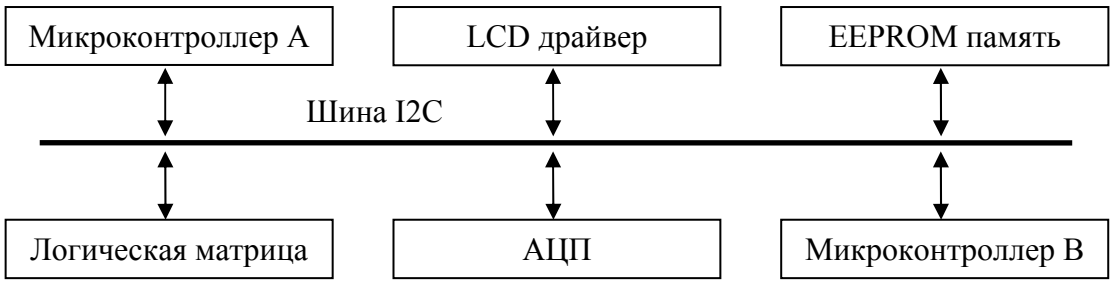


Рис. 1.7. Пример конфигурации шины I2C с двумя микроконтроллерами

Этот пример показывает взаимоотношения ведущий-ведомый и передатчик-приемник. Следует отметить, что эти отношения не постоянны и зависят только от направления передачи данных в конкретный момент времени. Передача данных будет происходить следующим образом:

1. Предположим, микроконтроллер A хочет передать информацию микроконтроллеру B:
   - Микроконтроллер A (ведущий) адресует микроконтроллер B (ведомый)
   - Микроконтроллер A (ведущий-передатчик) передает данные микроконтроллеру B (ведомый-приемник)

- Микроконтроллер А завершает передачу.

2. Предположим, микроконтроллер А желает принять информацию от микроконтроллера В:
   - Микроконтроллер А (ведущий) адресует микроконтроллер В (ведомый)
   - Микроконтроллер А (ведущий-приемник) принимает данные от микроконтроллера В (ведомый-передатчик)
   - Микроконтроллер А завершает передачу.

В том и в другом случае ведущее устройство (микроконтроллер А) генерирует сигнал синхронизации и завершает передачу.

Возможность подключения к одной шине нескольких микроконтроллеров означает, что несколько ведущих могут попытаться инициировать передачу данных одновременно. Для устранения хаоса, который может при этом возникнуть, была разработана процедура арбитража. Даная процедура основана на том, что все соединения на шине I2C выполнены по схеме «монтажное И» (процедура арбитража более подробно описана в § 1.4.6)

Генерация тактового сигнала – это всегда обязанность ведущего; каждый ведущий генерирует свой собственный тактовый сигнал при передаче данных по шине I2C. Тактовый сигнал может быть изменен только если он «затягивается» медленным ведомым устройством (путем удержания линии в низком состоянии), или другим ведущим во время процедуры арбитража.

§ 1.4.1. Передача бита

Данные на линии SDA должны быть стабильными в течение высокого лог. уровня на линии SCL. Состояние линии SDA должно изменяться, только если на линии SCL присутствует низкий логический уровень.

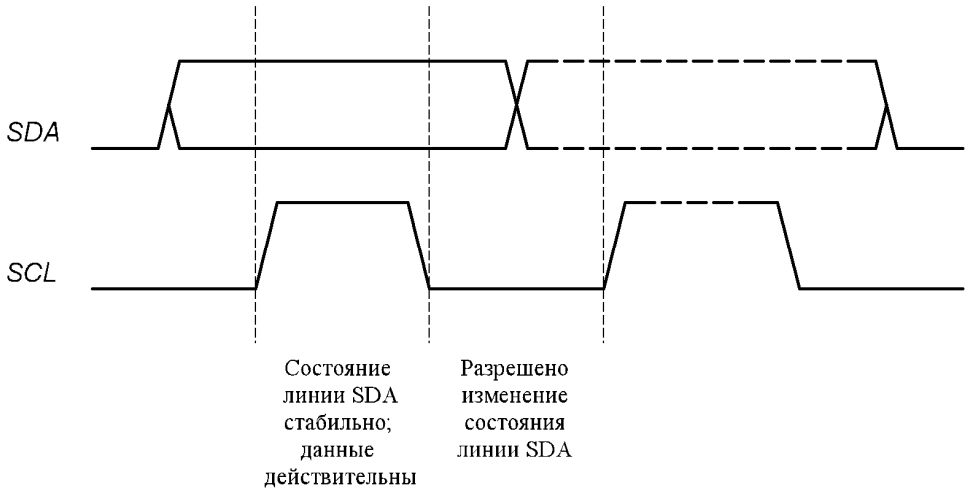


Рис. 1.8. Передача бита по шине I2C

На каждый передаваемый по линии SDA бит данных приходится один тактовый импульс на линии SCL (см. рис. 1.8).

§ 1.4.2. Условия START и STOP

Спецификацией шины I2C предусмотрены специальные условия, называемые START и STOP.

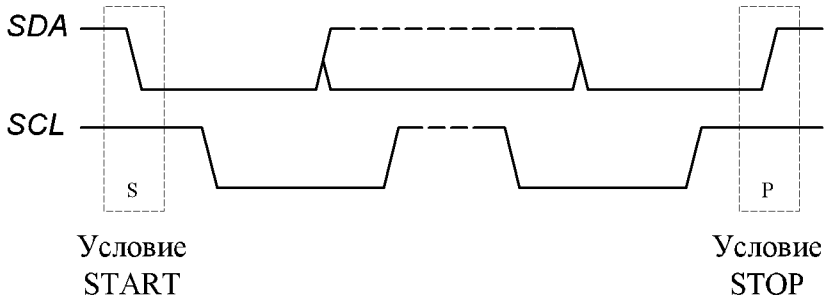


Рис. 1.9. Условия START и STOP

Условие START – это переход логического уровня сигнала на линии SDA из высокого в низкий, в то время как на линии SCL присутствует высокий логический уровень.

Условие STOP – это переход логического уровня сигнала на линии SDA из низкого в высокий, в то время как на линии SCL присутствует высокий логический уровень.

Условия START и STOP всегда формирует ведущее устройство. После того, как было сформировано условие START, шина считается занятой. Шина считается свободной спустя некоторое время после формирования условия STOP.

Если вместо условия STOP было сформировано условие repeated START (RESTART), то шина продолжает оставаться занятой.

Определение условий START и STOP не представляет сложностей для устройств, в которые встроены специальные аппаратные средства. Однако микроконтроллеры, которые не имеют таких средств, должны осуществлять считывание значения линии SDA как минимум дважды за период синхронизации, чтобы отследить изменение состояния на линии.

§ 1.4.3. Формат байта

Передача данных по линии SDA происходит побайтно.

Количество байт, передаваемых за один сеанс связи, неограниченно. Каждый байт должен оканчиваться битом подтверждения. Данные передаются, начиная со старшего бита.

Если ведомый не может принять/передать еще один целый байт, пока он не выполнит какую-либо другую функцию, он может удер-

живать на линии SCL низкий логический уровень, переводя ведущего в состояние ожидания. Пересылка данных возобновится, когда ведомый будет готов к приему/передаче следующего байта и отпустит линию SCL.

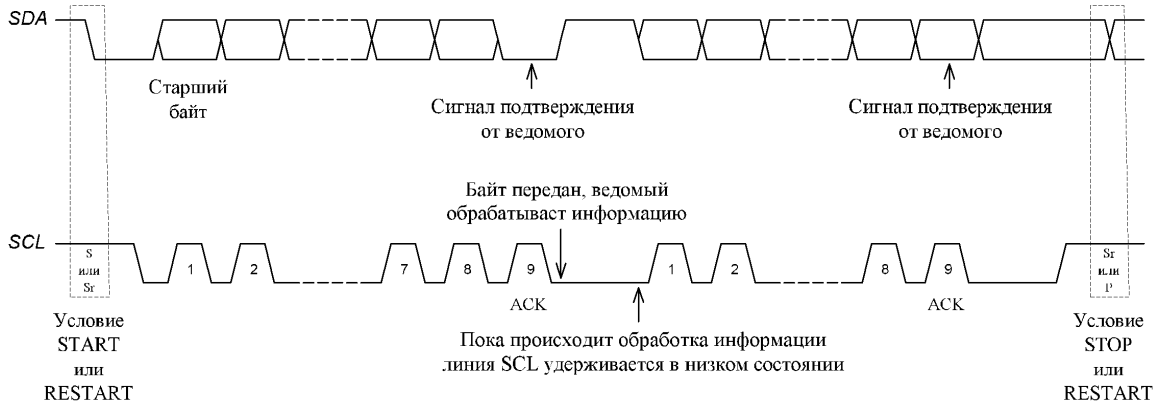


Рис. 1.10. Передача данных по шине I2C

§ 1.4.4. Формирование Бита Подтверждения

Подтверждение при передаче данных обязательно. Тактовый импульс, соответствующий биту подтверждения, формируется ведущим. На время тактового импульса, соответствующего биту подтверждения, передатчик отпускает линию SDA. Во время тактового импульса подтверждения приемник должен стабильно удерживать на линии SDA низкий логический уровень.

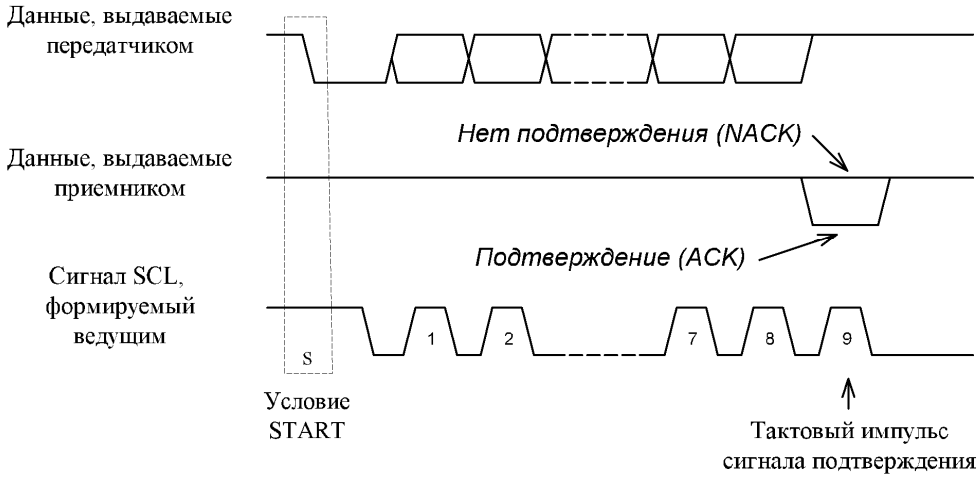


Рис. 1.11. Формирование бита подтверждения

Если ведомый не может подтвердить свой адрес (например, когда он выполняет какие-либо функции реального времени), на линии SDA должен быть оставлен высокий логический уровень. После этого веду-

щий может сформировать условие STOP для прекращения текущего сеанса связи или условие RESTART для начала нового.

Если ведомый-приемник подтвердил свой адрес, и через некоторое время больше не может принимать данные, ведущий также должен прервать передачу. Для этого ведомый не подтверждает байт, «отпускает» линию данных и ведущий формирует условие STOP или RESTART.

Если в передаче участвует ведущий-приемник, то он должен сообщить об окончании передачи ведомому-передатчику с помощью не подтверждения последнего байта. После этого ведомый-передатчик должен отпустить линию данных, чтобы позволить ведущему сформировать условие STOP или RESTART.

§ 1.4.5. Синхронизация

При передаче данных по шине I2C каждый ведущий генерирует свой собственный тактовый сигнал. Данные на линии SDA действительны только во время присутствия высокого логического уровня на линии SCL.

Синхронизация осуществляется благодаря тому, что устройства подключены к линии SCL по схеме «монтажное И». Это означает, что если одно из устройств выдает на линию SCL низкий лог. уровень, то на выводах SCL всех устройств, подключенных к шине будет низкий логический уровень.

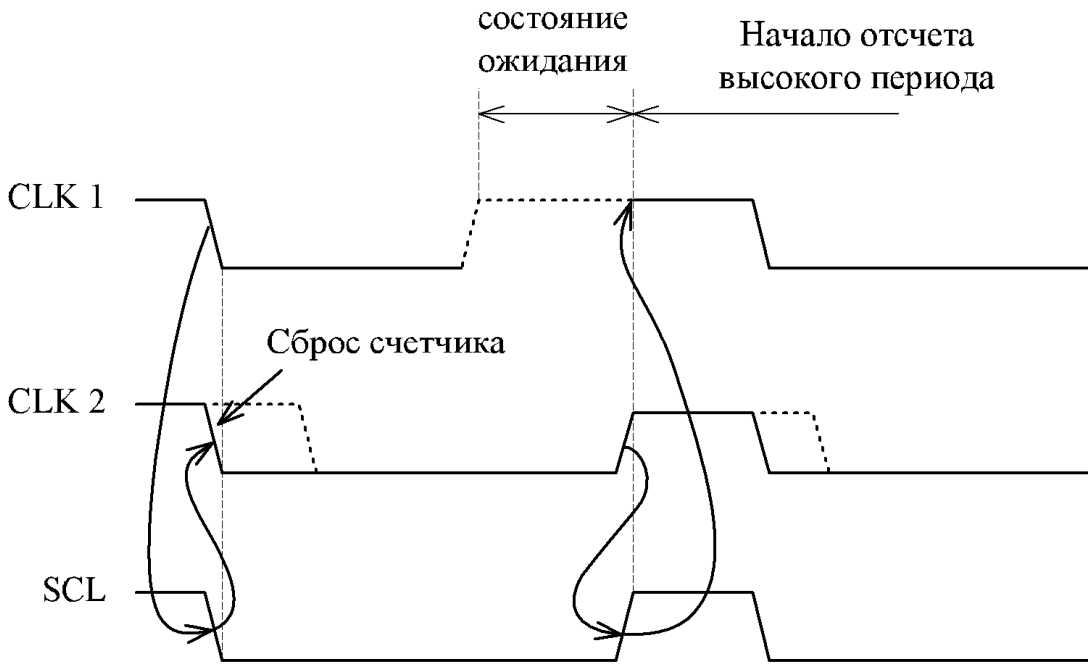


Рис. 1.12. Синхронизация во время процедуры арбитража

Однако, если одно из устройств желает сформировать на лини SCL высокий логический уровень в то время как другое устройство удерживает на линии низкий лог. уровень, то линия будет оставаться в состоянии лог. «0». Таким образом, на линии SCL будет находиться низкий логический уровень на протяжении самого длинного низкого периода из всех тактовых сигналов. Устройства с более коротким низким периодом тактового сигнала будут находиться в состоянии ожидания, пока не кончится длинный период.

Когда у всех ведущих устройств закончится низкий период тактового сигнала, на линии SCL появится высокий логический уровень. После этого все ведущие устройства начнут отсчитывать высокий период их тактового сигнала. Первое устройство, у которого закончится этот период, снова установит на линии SCL низкий логический уровень.

Таким образом, низкий период на линии SCL определяется самым длинным низким периодом тактового сигнала из всех подключенных к шине устройств, а высокий период определяется самым коротким периодом тактового сигнала устройств.

§ 1.4.6. Арбитраж

Ведущий может начать передачу данных только если шина свободна. Несколько ведущих могут сформировать условие START одновременно.

Арбитраж происходит на шине SDA в период высокого уровня на линии SCL. Если один ведущий удерживает на линии SDA низкий логический уровень, в то время как другой ведущий пытается сформировать на ней уровень лог. «1», то последний отключается от линии, т. к. уровень SDA не соответствует уровню его внутренней линии данных.

Арбитраж может продолжаться в течение передачи нескольких бит. Сначала арбитраж продолжается при адресации ведомого. Если ведущие адресуют одного и того же ведомого, то арбитраж продолжается при передаче бит данных данных, если это ведущие-передатчики, или бит подтверждения, если это ведущие-приемники. Так как адрес и данные на шине определяются ведущим, выигравшим арбитраж, то потери данных не происходит.

Ведущий, который проиграл арбитраж, может продолжать генерировать тактовые импульсы до конца байта, при передаче которого арбитраж был проигран.

Если ведущий также имеет функции ведомого и проигрывает арбитраж на этапе адресации, то возможно, что ведущий, выигравший арбитраж, желает обратиться к проигравшему. Поэтому проигравший ведущий должен немедленно переключиться в режим ведомого.

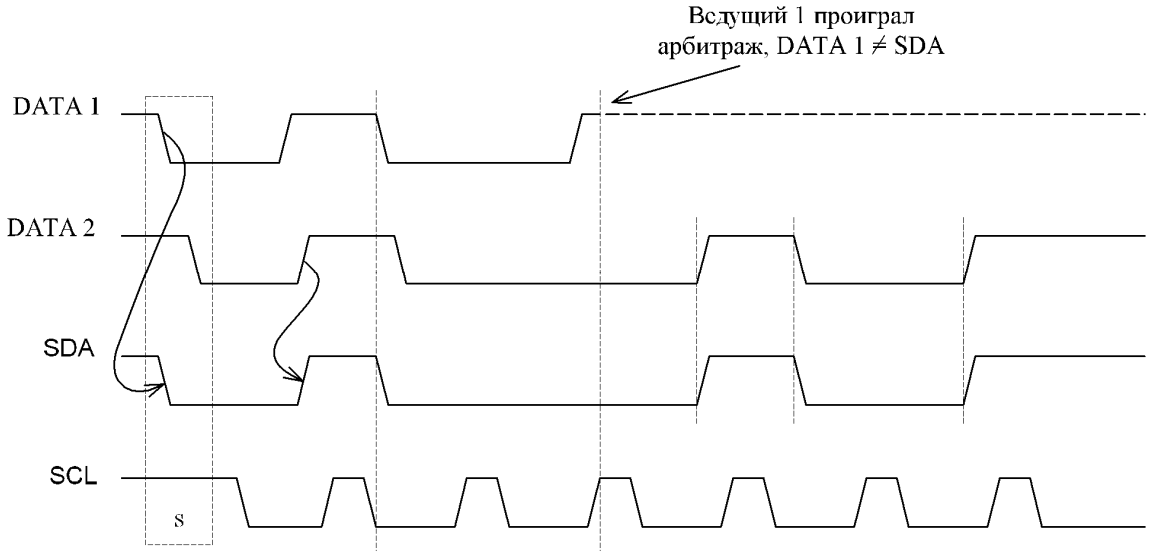


Рис. 1.13. Процедура арбитража двух ведущих

На рис. 1.13 показана процедура арбитража двух ведущих устройств. Разумеется, в процесс арбитража может быть вовлечено большее количество ведущих. В момент, когда обнаруживается различие между уровнем внутренней линии данных и уровнем, присутствующим на линии SDA, ведущий 1 отключается от линии, не влияя, таким образом, на передачу данных выигравшего ведущего.

Так как арбитраж зависит только от адреса и данных, передаваемых конкурирующими ведущими, не существует ни центрального ведущего, ни приоритетного доступа к шине.

Особое внимание следует уделить ситуации, когда во время процедуры арбитража на шине формируется условие RESTART или STOP. Если такая ситуация возможна, то ведущие должны формировать условие RESTART или STOP в одинаковых позициях кадра. Другими словами, арбитраж запрещен между:

- Условием RESTART и битом данных
- Условием STOP и битом данных
- Условием RESTART и условием STOP

Ведомые устройства не участвуют в процедуре арбитража.

§ 1.4.7. Использование синхронизации для управления связью

Кроме использования в процедуре арбитража, механизм синхронизации может быть использован приемниками как средство управления пересылкой данных на байтовом или битовом уровнях.

На уровне байта устройство может принимать/передавать данные на большой скорости, но требует некоторого времени для сохранения принятого байта или подготовки следующего отправляемого байта. В

этом случае устройство может удерживать на линии SCL низкий логический уровень после приема байта и выдачи бита подтверждения для перевода ведущего устройства в состояние ожидания. Передача продолжится после того, как устройство будет готово и «отпустит» линию SCL (см. рис. 1.10).

На уровне бита такое устройство, как микроконтроллер без аппаратно реализованного модуля I2C, может замедлить тактовую частоту путем продления низкого периода импульсов синхронизации. Таким образом, скорость передачи любого ведущего согласуется со скоростью ведомого устройства.

§ 1.4.8. Форматы с 7-битным адресом

Передача данных происходит в формате, показанном на рис. 1.14.

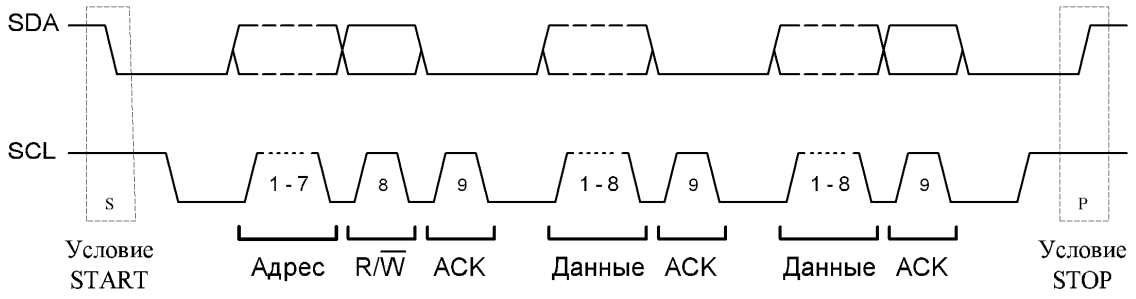
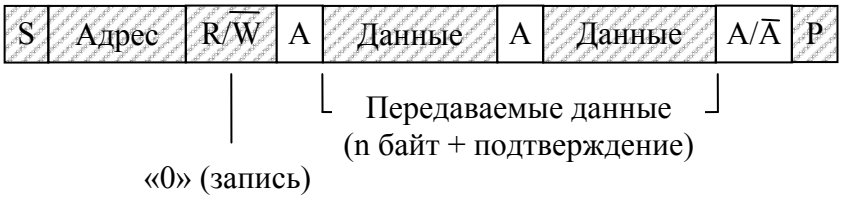
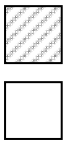


Рис. 1.14. Сеанс связи по шине I2C





A = подтверждение (SDA = «0»)
$\overline{A}$ = нет подтверждения (SDA = «1»)
S = условие START
P = условие STOP

Рис. 1.15. Ведущий-передатчик адресует ведомого-приемника с 7-битным адресом. Направление передачи не изменяется

После формирования условия START посылается адрес ведомого устройства. После 7-го бита адреса передается бит направления (бит R/W). Если этот бит равен «0», значит производится передача (запись), если же он равен «1», то производится прием (чтение). Передача данных

всегда заканчивается условием STOP, формируемым ведущим. Однако, если ведущий желает сохранить контроль над шиной, он формирует условие RESTART и адресует другого ведомого, без предварительного формирования условия STOP. При такой передаче данных возможны различные комбинации чтения/записи.

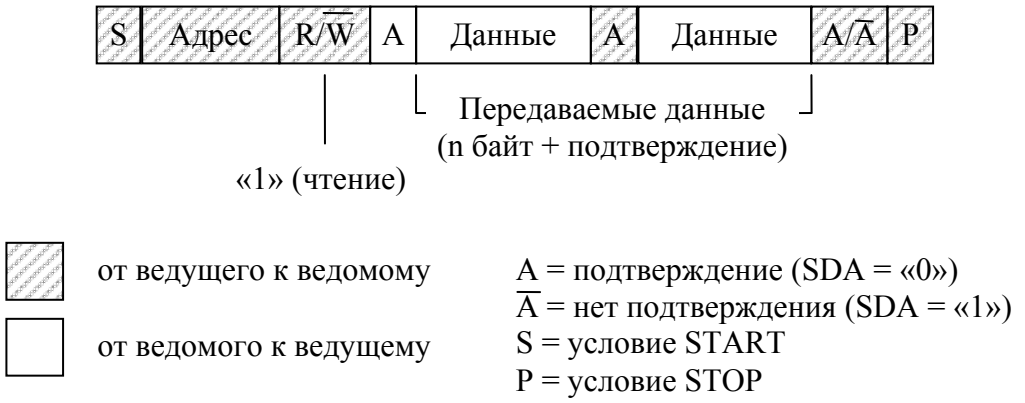


Рис. 1.16. Ведущий читает ведомого сразу после передачи адреса

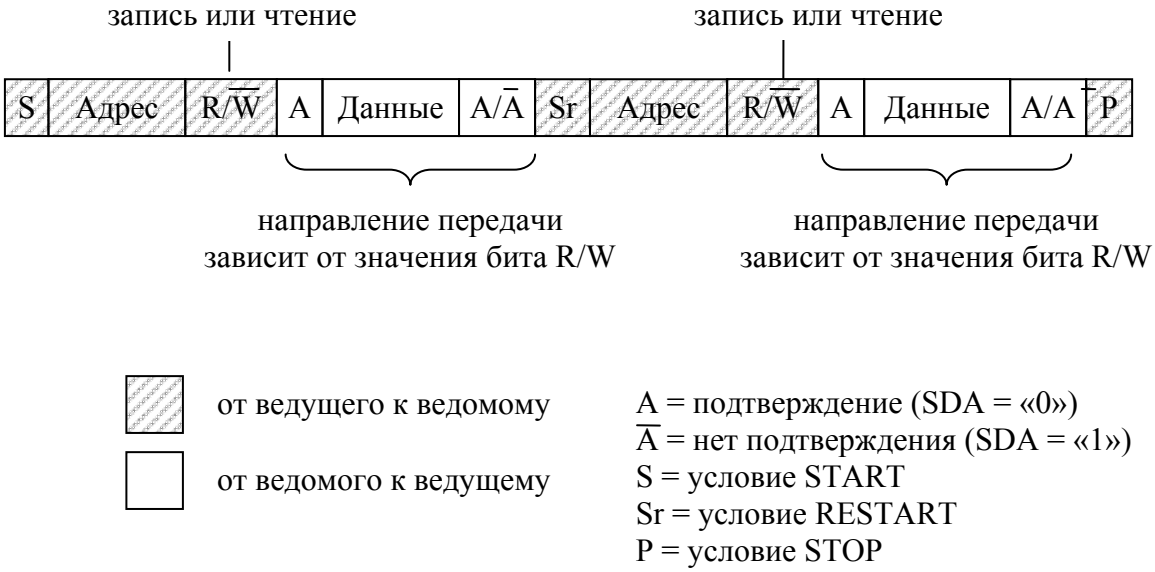


Рис. 1.17. Комбинированный формат передачи данных

Примечания:

- Комбинированные форматы могут быть использованы, например, для управления последовательной памятью. Первый передаваемый байт задает адрес внутренней ячейки памяти, который сохраняется во внутреннем регистре микросхемы. После формирования условия RESTART и передачи адреса ведомого, могут быть прочитаны данные из памяти.

- Все решения об авто-инкременте или декременте адреса, по которому было обращение, принимаются разработчиком устройства.
- После каждого байта формируется бит подтверждения.
- I2C-совместимые устройства должны сбрасывать логику шины при обнаружении условий START или RESTART и ожидать передачи адреса ведомого, даже если условие START/RESTART было сформировано в неправильной позиции кадра.
- Не разрешается передача пустых сообщений (таких, в которых сразу после условия START следует условие STOP).

Возможные форматы передачи данных:

- Ведущий-передатчик передает данные ведомому-приемнику. Направление передачи данных не изменяется (см. рис. 1.15).
- Ведущий читает ведомого сразу после передачи адреса (см. рис. 1.16). В момент первого подтверждения ведущий-передатчик становится ведущим-приемником, а ведомый-приемник становится ведомым-передатчиком. Первое подтверждение генерируется ведомым. Перед тем, как сформировать условие STOP, ведущий формирует бит «неподтверждения» (NACK).
- Комбинированный формат (см. рис. 1.17). При изменении направления передачи данных ведущий формирует условие RESTART и снова отправляет адрес ведомого устройства, проинвертировав бит R/W. Если ведущий-приемник формирует условие RESTART, он должен предварительно сформировать бит «неподтверждения» (NACK).

§ 1.4.9. 7-битная адресация

Процедура адресации на шине I2C заключается в том, что первый байт после условия START определяет, с каким ведомым будет работать ведущее устройство. Исключение – адрес общего вызова, который адресует все устройства на шине. Когда передается этот адрес, теоретически все устройства на шине должны сформировать подтверждение. Однако, устройства могут игнорировать этот адрес. Байт, следующий за адресом общего вызова, определяет действие, которое будет выполнено.

§ 1.4.10. Назначение битов байта адреса

Первые 7 бит первого байта составляют адрес ведомого устройства. Восьмой (младший) бит определяет направление передачи данных. Если он равен «0», значит ведущий будет передавать данные выбранному ведомому устройству. Если бит направления равен «1», значит ведущий будет принимать информацию от ведомого.

После того, как адрес отправлен, каждое устройство на шине сравнивает первые семь бит байта адреса со своим собственным адресом. Если адрес совпадает, то устройство считается адресованным как ведомый-приемник либо как ведомый-передатчик, в зависимости от бита направления R/W.

Адрес ведомого может состоять из фиксируемой и программируемой части. Так как возможно, что в системе будет несколько одинаковых устройств, использование программируемой части адреса позволяет подключить к шине максимально возможное количество таких устройств. Количество программируемых бит в адресе зависит от числа свободных выводов микросхемы. Например, если устройство имеет 4 фиксированных и 3 программируемых адресных бита, то всего к шине может быть подключено 8 таких устройств.

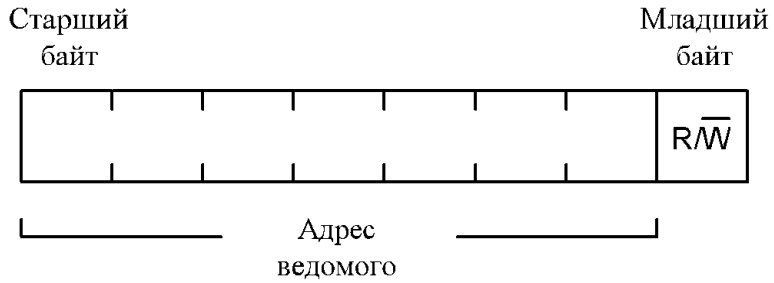


Рис. 1.18. Назначение битов байта адреса

Комитет I2C координирует выделение I2C адресов. Две группы по 8 адресов (0000xxx и 1111xxx) зарезервированы для целей, указанных в табл. 1.2.

Таблица 1.2

Зарезервированные адреса

Адрес	Значение бита направления R/W	Описание
0000 000	0	Адрес общего вызова
0000 000	1	байт START(1)
0000 001	x	CBUS-адрес(2)
0000 010	x	Адрес, зарезервированный для шин другого формата(3)
0000 011	x	Зарезервировано
0000 1xx	x	Код ведущего режима High-speed
1111 1xx	x	Зарезервировано
1111 0xx	x	10-битная адресация

Примечания:

1. Устройствам запрещается подтверждать прием байта START
2. Адрес CBUS был зарезервирован, чтобы была возможность использовать CBUS-совместимые и I2C-совместимые устройства в одной системе. I2C-совместимые устройства не должны реагировать на прием этого байта.
3. Адрес, зарезервированный для шин другого формата. Также предназначен для смешанного использования I2C и других протоколов. На этот адрес могут отвечать только те I2C-устройства, которые поддерживают эти форматы.

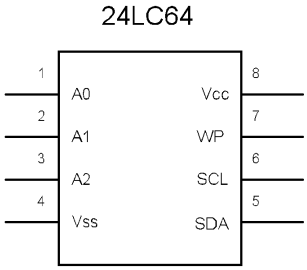
Подробную информацию по использованию адреса общего вызова, байта START, CBUS-адреса, 10-битной адресации, а также режима High-speed можно найти в [1].

ГЛАВА 2. МИКРОСХЕМЫ С I2C

§ 2.1. Микросхема EEPROM-памяти 24LC64

Данная микросхема – это микросхема памяти EEPROM-типа объемом 64 Кбита (8К х 8), выполненная по технологии CMOS. Диапазон напряжений питания от 2.5 В до 5.5 В. Максимальный потребляемый ток при записи данных составляет 3 мА, при чтении – 400 мкА, в состоянии простоя – 100 нА. Фирма-производитель гарантирует до 1 000 000 циклов стирания/записи. Передача данных по интерфейсу I2C может происходить на частоте до 400 кГц. Расположение выводов микросхемы приведено на рис. 2.1.

Микросхема 24LC64 имеет 7-разрядный адрес. Старшие четыре бита адреса заданы производителем («1010»), младшие три бита задаются с помощью адресных входов A2-A0. Адресные входы могут быть напрямую подключены к Vss или Vcc. Если, к примеру, на выводы A2 и A1 подан уровень лог. «1», а на вывод A0 подан уровень лог. «0», то микросхема будет иметь адрес «1010110». Таким образом, к одной шине I2C может быть подключено до восьми микросхем 24LC64.



Назначение выводов:

A0, A1, A2	– адресные входы
Vss	– общий провод
Vcc	– напряжение питания
WP	– вывод защиты от записи
SCL	– линия передачи синхроимпульсов (I2C)
SDA	– линия передачи данных (I2C)

Рис. 2.1. Расположение выводов микросхемы 24LC64

Запись данных в 24LC64

Для того чтобы записать байт данных в микросхему, нужно после условия START на шину выдать байт адреса со сброшенным битом R/W, затем выдать старший и младший байты адреса ячейки памяти, в которую будет производиться запись. После этого ведущий шины передает записываемый байт/байты (до 32 байт) и формирует условие STOP. Если было передано несколько байт данных, то первый байт будет сохранен в той ячейке, адрес которой был указан, второй запишется в следующую ячейку и т. д. На рис. 2.2 изображен процесс записи данных.

Переданный на запись байт (байты) предварительно сохраняется во внутренних регистрах микросхемы, запись в EEPROM-память происходит после формирования условия STOP. Такая внутренняя организация позволяет сократить время, затрачиваемое на передачу данных (т. к. запись в регистры происходит намного быстрее, чем сохранение в EEPROM-памяти).

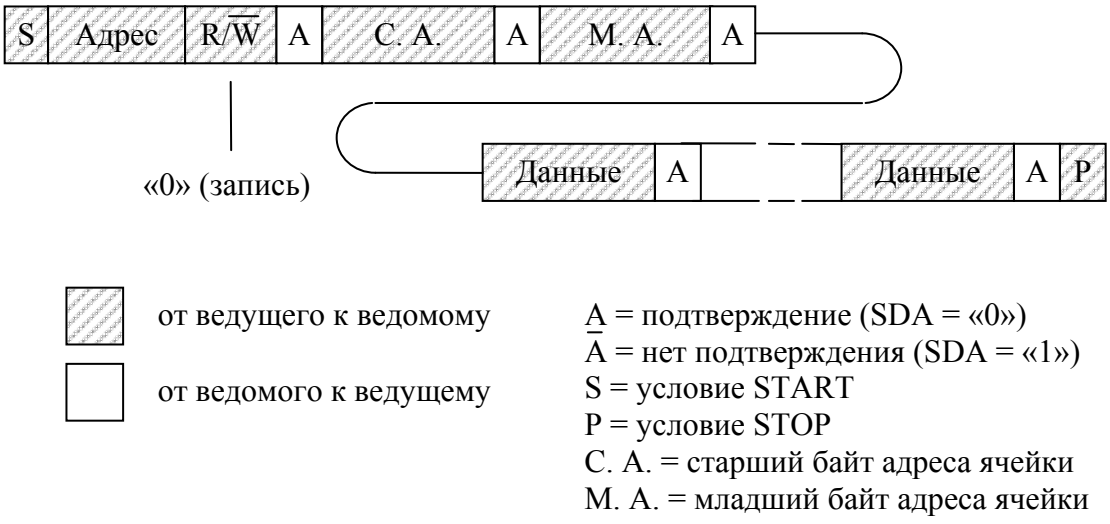


Рис. 2.2. Запись данных в 24LC64

Если обратиться к микросхеме 24LC64 в тот момент, когда она производит запись информации в энергонезависимую память, то бит подтверждения не будет сформирован.

Таким образом, чтобы продолжить работу с микросхемой, нужно дождаться момента, когда она вновь начнет формировать бит подтверждения при обращении к ней.

Чтение данных из 24LC64

Чтение данных из микросхемы производится следующим образом. После формирования условия START на шину выдается байт адреса со сброшенным битом R/W, затем передаются старший и младший байты адреса ячейки памяти, данные из которой нужно прочитать. Затем формируется условие RESTART и на шину снова выдается байт адреса микросхемы, но уже с установленным битом R/W. После этого ведущий шины читает один или несколько байт данных. На рис. 2.3 изображен процесс чтения данных.

Первый байт, передаваемый ведомым, – это содержимое ячейки, адрес которой был указан, второй байт – содержимое следующей ячейки и т. д. Таким образом можно прочитать все ячейки микросхемы за один сеанс.

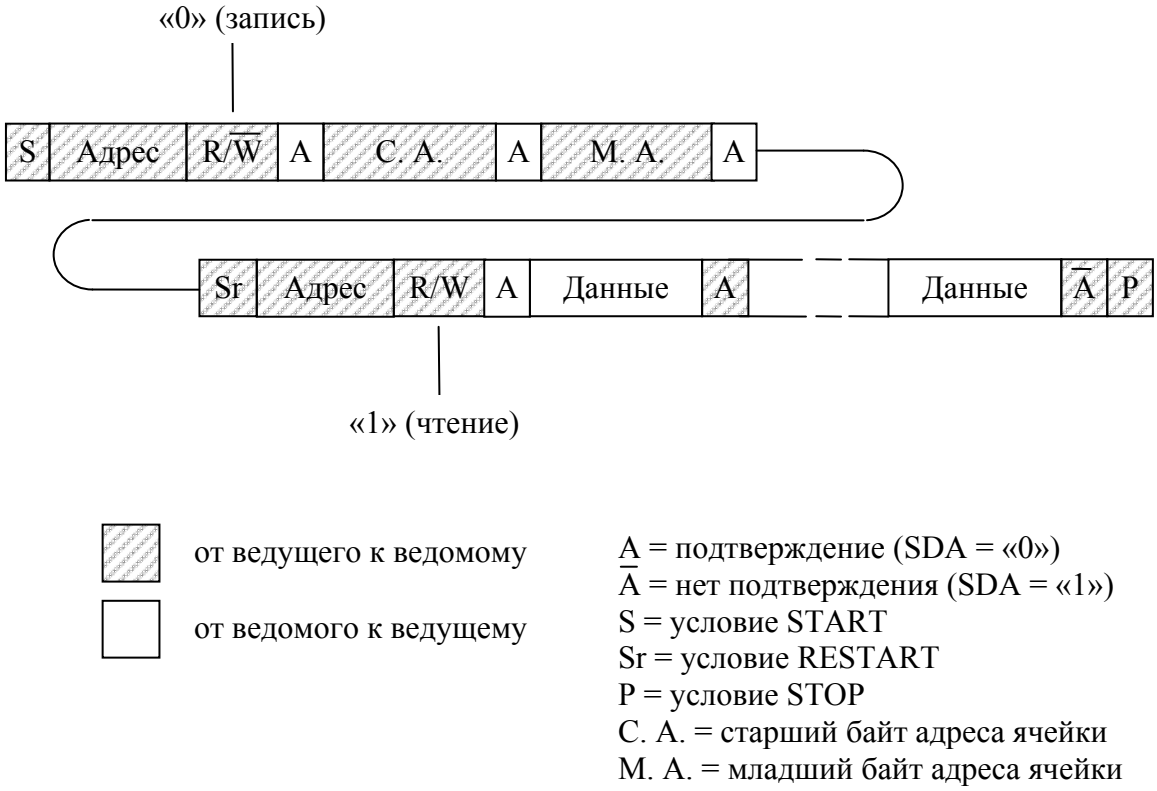


Рис. 2.3. Чтение данных из 24LC64

Чтение данных из микросхемы также может быть организовано следующим образом. После формирования условия START на шину выдается байт адреса с установленным битом R/W и производится чтение одного или нескольких байт данных. При таком способе чтения первым передается байт данных, адрес которого содержался во внутреннем указателе адреса микросхемы. Недостатком такого способа является то, что не всегда заранее известно, на какой байт указывает внутренний указатель микросхемы (например, если на шине имеется более одного ведущего устройства).

Более подробную информацию можно найти в [2].

§ 2.2. Датчик температуры DS1621

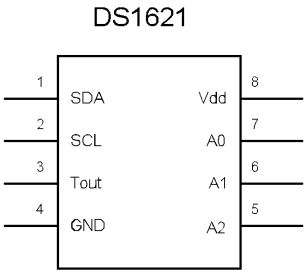
Данная микросхема представляет собой цифровой термометр и термостат. Позволяет измерять температуру в диапазоне от –55 °С до +125 °С с точностью 0.5 °С. Преобразование температуры в цифровой код происходит менее чем за одну секунду.

Расположение выводов микросхемы приведено на рис. 2.4.

Датчик температуры имеет три программно доступных регистра:

- регистр конфигурации;
- регистр TH;

- регистр TL.
  Регистр конфигурации содержит восемь бит (см. рис. 2.5).



Назначение выводов:

SDA	– линия передачи данных (I2C);
SCL	– линия передачи синхроимпульсов (I2C);
Tout	– выходной сигнал термостата;
GND	– общий провод;
Vdd	– напряжение питания;
A0, A1, A2	– адресные входы.

Рис. 2.4. Расположение выводов микросхемы DS1621

Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
DONE	THF	TLF	NVB	-	-	POL	1SHOT

Рис. 2.5. Регистр конфигурации

Рассмотрим назначение каждого бита.

- DONE – это флаг, показывающий, завершено ли преобразование температуры в цифровой код. Если преобразование завершено, бит DONE равен «1», если еще продолжается то «0».
- THF – флаг, который устанавливается в лог. «1» в случае, когда температура датчика поднялась выше значения, хранящегося в регистре TH. Флаг сбрасывается записью лог. «0» в бит THF, либо при отключении питания.
- TLF – флаг, который устанавливается в лог. «1» если температура датчика опустилась ниже значения, хранящегося в регистре TL. Флаг сбрасывается записью лог. «0» в бит TLF, либо при отключении питания.
- NVB – флаг, который отображает состояние процесса записи в энергонезависимую память датчика. Если NVB = «1», то запись производится, если NVB = «0», то запись завершена.
- POL – бит, определяющий какой уровень будет являться активным для вывода Tout. Если POL = «1», то активный уровень – высокий, если POL = «0», то активный уровень – низкий. Этот бит хранится в энергонезависимой памяти.
- 1SHOT – бит, определяющий, в каком режиме будет работать датчик температуры. Если 1SHOT = «1», то датчик будет работать в режиме однократных преобразований, т. е. после прихода команды «Start Convert T» датчик совершит только одно преобразование. Если же 1SHOT = «0», то команда «Start Convert T» запустит цикл преобразований, остановить который можно с по-

мощью команды «Stop Convert T». Этот бит хранится в энергонезависимой памяти.

Второй и третий биты зарезервированы.

В регистрах TH и TL хранятся значения температуры, используемые при формировании сигнала на выводе Tout.

Если текущая температура датчика превысила значение, хранящееся в регистре TH, то на выводе Tout появляется уровень логической единицы (если в байте конфигурации бит POL = «1»). Высокий уровень напряжения на выводе Tout сохраняется до тех пор, пока температура датчика не упадет ниже значения, хранящегося в регистре TL (см. рис. 2.6).

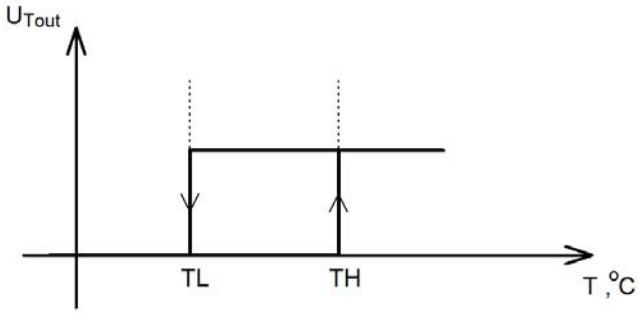


Рис. 2.6. Работа DS1621 в режиме термостата

Таблица 2.1

Код, соответствующий некоторым значениям температур

Температура	Цифровой код (в двоичной системе счисления)	Цифровой код (в шестнадцатеричной системе счисления)
+ 125 °C	01111101 00000000	7D 00
+ 25 °C	00011001 00000000	19 00
+ 0.5 °C	00000000 10000000	00 80
0 °C	00000000 00000000	00 00
–0.5 °C	11111111 10000000	FF 80
–25 °C	11100111 00000000	E7 00
–125 °C	11001001 00000000	C9 00

Выходной сигнал на выводе Tout обновляется сразу после того, как завершается преобразование температуры в цифровой код.

Для передачи значения температуры используется два байта. При этом из 16 бит значащими являются только 9 (используется левостороннее выравнивание), а само значение температуры представлено в дополнительном коде. В табл. 2.1 приведены некоторые значения температур и соответствующий им код.

Датчик DS1621 имеет 7-разрядный адрес. Старшие четыре бита адреса заданы производителем («1001»). Младшие три задаются с помощью соответствующего включения адресных выводов A2, A1 и A0. Если, к примеру, на выводы A2 и A1 подан уровень логической единицы, а на вывод A0 подан уровень логического нуля, то полный адрес датчика будет «1001110». Таким образом, к одной шине I2C может быть подключено до восьми датчиков DS1621.

Основные команды

Read Temperature [AAh]

Эта команда позволяет прочитать из датчика результат последнего преобразования температуры в цифровой код.

Access TH [A1h]

Данная команда позволяет прочитать или записать (в зависимости от значения бита R/W) содержимое двухбайтного регистра TH. Формат данных такой же, как и при чтении значения температуры.

Access TL [A2h]

Данная команда позволяет прочитать или записать (в зависимости от значения бита R/W) содержимое двухбайтного регистра TL. Формат данных такой же, как и при чтении значения температуры.

Access Config [ACh]

Эта команда позволяет прочитать или записать байт в регистр конфигурации.

Start Convert T [EEh]

При получении этой команды запускается процесс преобразования температуры. Если датчик настроен на работу в режиме многократных преобразований, то запускается цикл преобразований.

Stop Convert T [22h]

Данная команда останавливает преобразование температуры. Она может быть использована для прерывания цикла преобразований (если датчик работает в режиме многократных преобразований). Преобразования возобновятся при получении датчиком команды «Start Convert T».

Основные операции

Прежде всего, следует настроить датчик на нужный режим работы, для чего в регистр конфигурации записывается соответствующий байт.

Делается это следующим образом. После условия START на шину выдается байт адреса со сброшенным битом R/W, затем передается код команды доступа к регистру конфигурации (Access Config [ACh]) и сам байт, после чего формируется условие STOP.

Для того чтобы прочитать данные из регистра конфигурации, необходимо после условия START выдать на шину байт адреса со сброшенным битом R/W. После этого на шину выдается команда доступа к регистру конфигурации «Access Config» и формируется условие RESTART. Затем ведущий выдает на шину байт адреса с установленным битом R/W и читает байт данных, после чего формируется условие STOP.

На рис. 2.7 и 2.8 показаны процессы записи и чтения данных из регистра конфигурации.

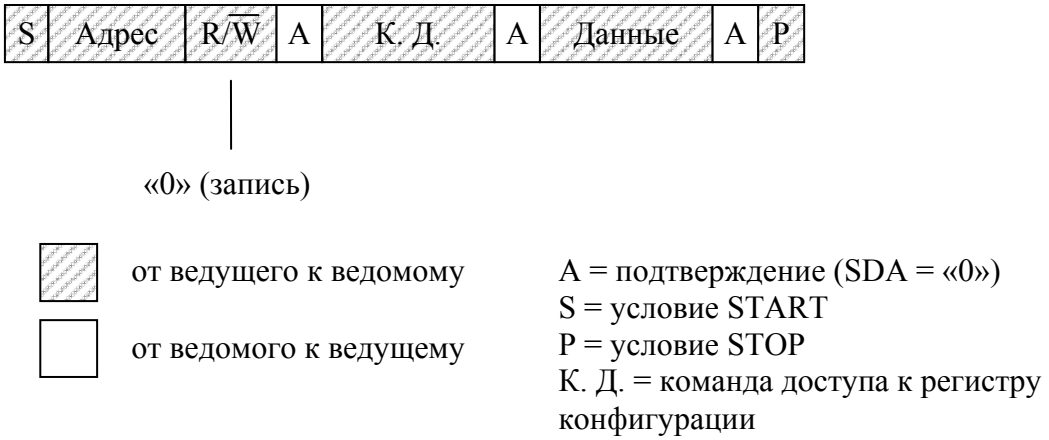


Рис. 2.7. Запись байта в регистр конфигурации

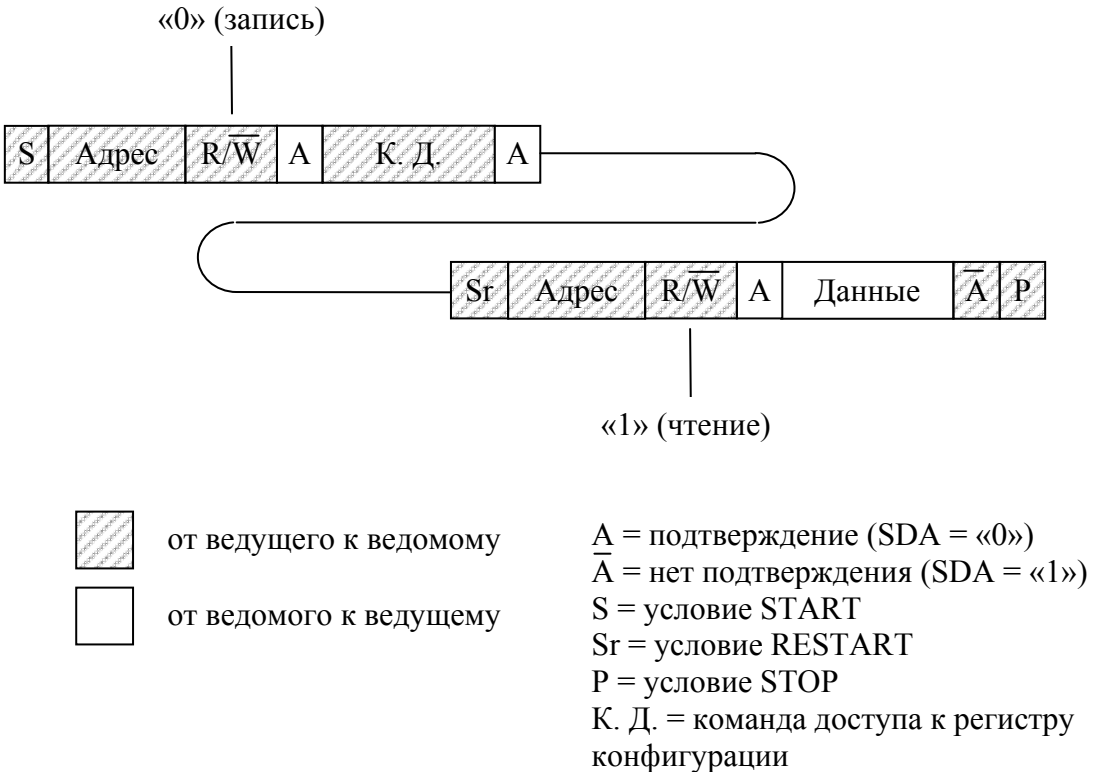


Рис. 2.8. Чтение регистра конфигурации

Если в схеме используется сигнал вывода Tout, то в регистры TH и TL нужно записать соответствующие значения температуры.

Для того чтобы записать данные в регистр TH нужно после формирования условия START выдать на шину байт адреса со сброшенным битом R/W, после этого передать код команды «Access TH» и двухбайтное значение температуры. Затем ведущий шины формирует условие STOP.

Если нужно прочитать данные из регистра TH, то поступают следующим образом. После условия START передается байт адреса со сброшенным битом R/W и код команды «Access TH». Затем формируется условие RESTART, снова передается байт адреса, но уже с установленным битом R/W, после чего ведущий шины принимает два байта от датчика и формирует условие STOP.

На рис. 2.9 и 2.10 показаны процессы записи и чтения данных из регистра TH.

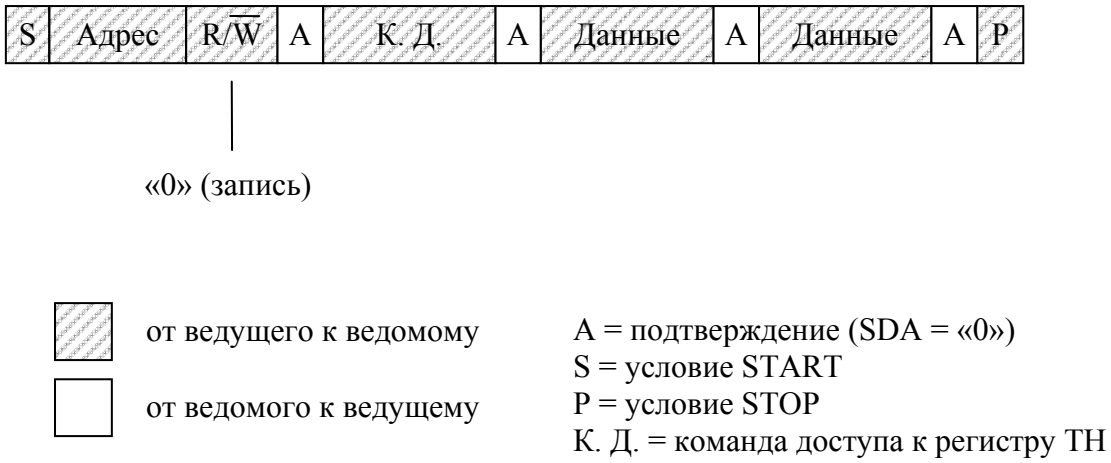


Рис. 2.9. Запись в регистр TH

Запись и чтение данных из регистра TL производится аналогично, только вместо команды «Access TH» нужно использовать команду «Access TL».

После того, как датчик был настроен на нужный режим работы, нужно запустить его на преобразование температуры. Для этого нужно после условия START выдать на шину байт адреса со сброшенным битом R/W, и передать код команды «Start Convert T», после чего сформировать условие STOP. Процесс запуска датчика на преобразование изображен на рис. 2.11.

Если датчик был настроен на работу в режиме однократного преобразования, то после получения команды «Start Convert T» он совершит только одно преобразование. Об окончании процесса преобразования можно судить по установке бита DONE в единицу.

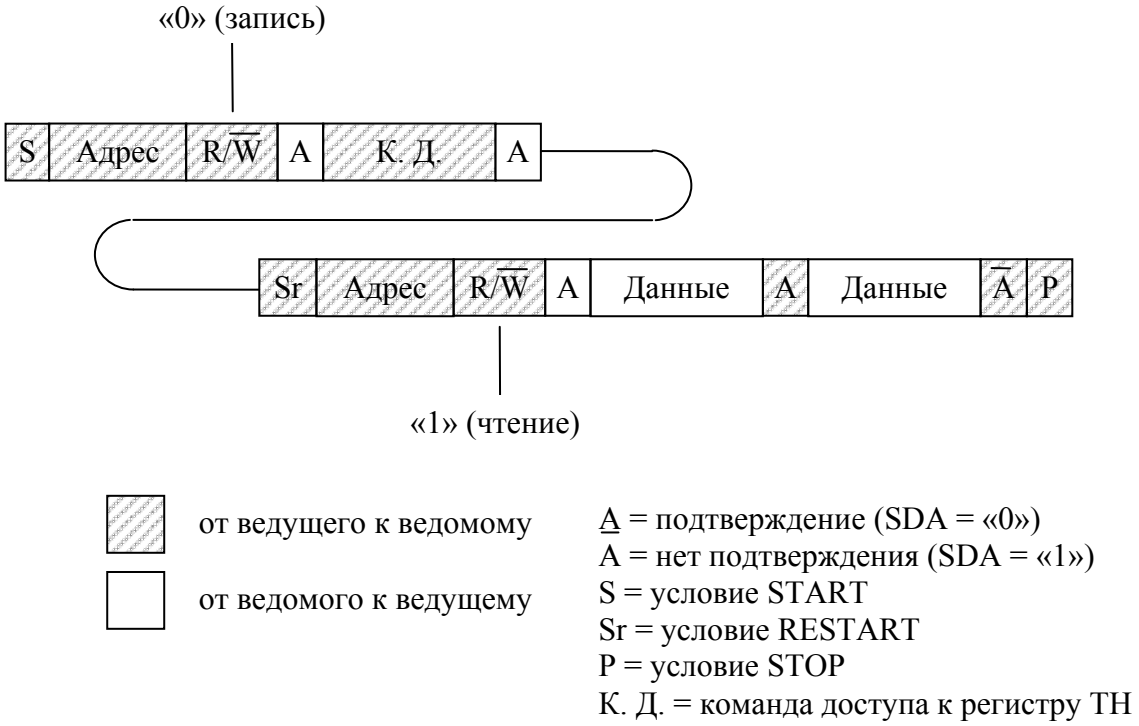


Рис. 2.10. Чтение регистра TH

Рис. 2.11. Запуск датчика на преобразование

Если датчик был настроен на работу в режиме многократных преобразований, то команда «Start Convert T» запустит серию преобразований, остановить которую можно с помощью команды «Stop Convert T». Чтобы остановить серию преобразований, нужно произвести те же действия, что и при запуске датчика, заменив при этом команду «Start Convert T» на команду «Stop Convert T».

Результаты преобразования читают следующим образом. После формирования условия START на шину выдается байт адреса со сбро-

шенным битом R/W, затем выдается команда «Read Temperature». Потом формируется условие RESTART, снова передается байт адреса, но уже с установленным битом R/W, после чего ведущий шины принимает два байта от датчика и формирует условие STOP. На рис. 2.12 показан процесс чтения результата преобразования.

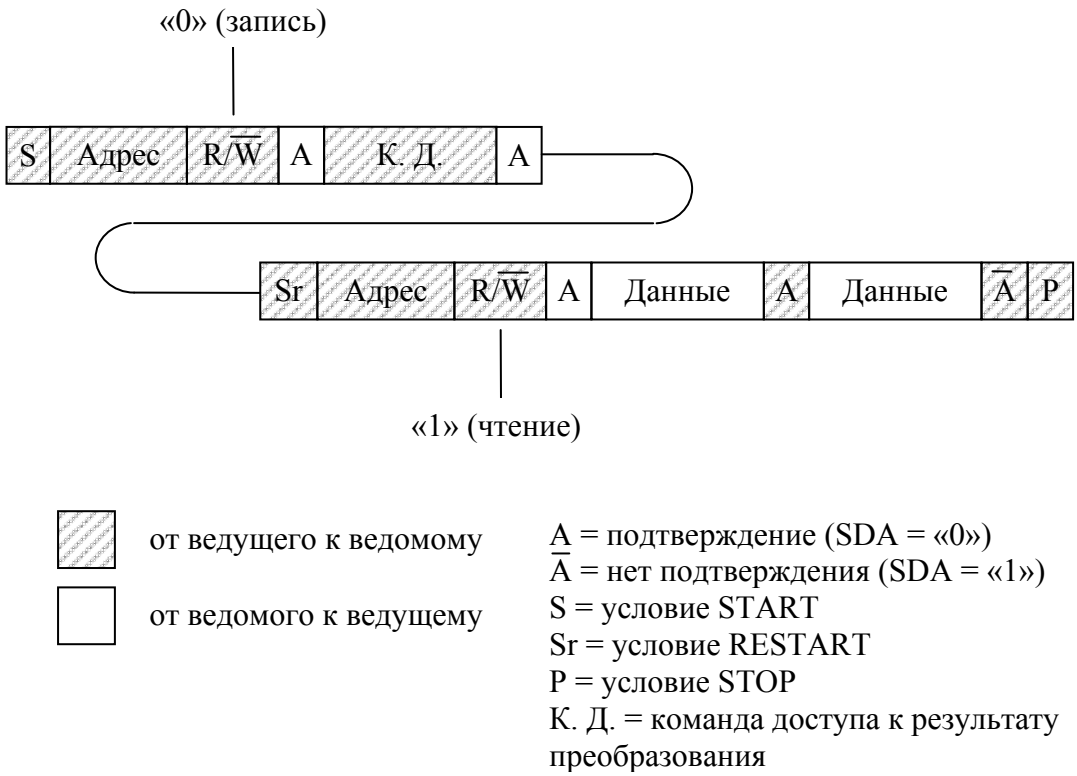


Рис. 2.12. Чтение результата преобразования

Более подробную информацию можно найти в [3].

§ 2.3. Часы реального времени МК41Т56

Данная микросхема включает в себя часы реального времени и оперативное запоминающее устройство емкостью 56 байт. Часы ведут отсчет секунд, минут, часов, дней недели, чисел, месяцев и лет (с учетом високосных). Кроме того, имеется возможность программной настройки точности хода часов.

Благодаря встроенному детектору питания при исчезновении питающего напряжения микросхема автоматически подключает альтернативный источник питания, в качестве которого выступает литиевый элемент напряжением 3 В. Так как микросхема МК41Т56 изготовлена по CMOS-технологии, она имеет очень низкое энергопотребление, и элемента питания с емкостью 50 мА·ч достаточно для поддержания работы в течение более чем 10-ти лет.

Типовая схема включения приведена на рис. 2.13.

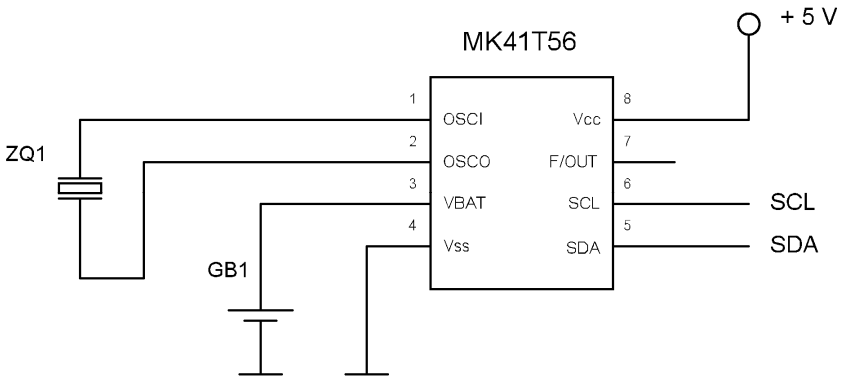


Рис. 2.13. Типовая схема включения MK41T56

С точки зрения пользователя микросхема представляет собой ОЗУ емкостью 64 байта, первые восемь байт которого задействованы для хранения текущего времени и даты, а так же байта калибровки. В табл. 2.2 представлено распределение памяти микросхемы.

Таблица 2.2

Распределение памяти MK41T56

Адрес ячейки	Содержимое ячейки	Диапазон значений (в шестнадцатеричной системе счисления)
0	секунды	00…59
1	минуты	00…59
2	часы	00…23
3	день недели	01…07
4	число месяца	01…31
5	месяц	01…12
6	год	00…99
7	регистр конфигурации	–

Данные о времени и дате хранятся в BCD (двоично-десятичном) формате. Для настройки часов нужно записать в ячейки с адресами 0…6 текущее время и дату (данные также должны быть представлены в BCD формате).

Запись данных в МК41Т56

Для того чтобы записать байт в микросхему, нужно после условия START на шину выдать байт адреса (7-разрядный адрес микросхемы «1101000») со сброшенным битом R/W, после чего передается адрес ячейки, в которую будет производиться запись. Далее следует сам запи-

сываемый байт (или несколько байт) и условие STOP. На рис. 2.14 изображен процесс записи данных.

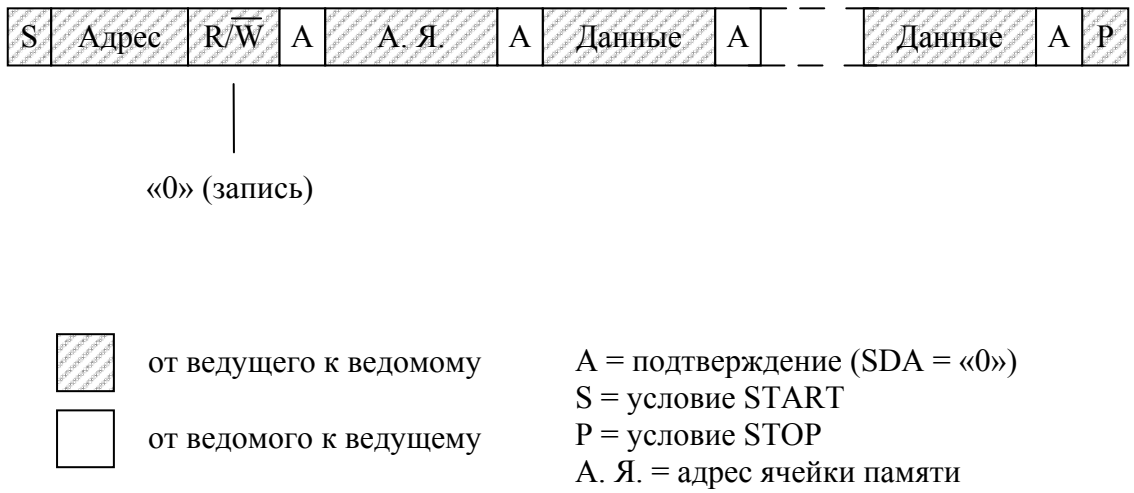


Рис. 2.14. Запись данных в МК41Т56

Если было передано несколько байт, то первый из них сохраняется по указанному адресу, второй – в следующей ячейке и т. д. Таким образом, за один сеанс можно заполнить все 64 ячейки микросхемы.

Чтение данных из МК41Т56

Чтение данных происходит следующим образом. После условия START и байта адреса устройства со сброшенным битом R/W следует адрес ячейки, содержимое которой нужно прочитать. После этого формируется условие RESTART и на шину выдается байт адреса микросхемы с установленным битом R/W. Затем ведущий читает из устройства один или несколько байт данных и формирует условие STOP. На рис. 2.15 изображен процесс чтения данных.

Если было прочитано несколько байт данных, то первый из них – это содержимое адресованной ячейки, второй – содержимое следующей ячейки и т. д. Таким образом, за один сеанс можно прочитать все 64 ячейки микросхемы.

Чтение данных из микросхемы также может быть организовано следующим образом. После формирования условия START на шину выдается байт адреса с установленным битом R/W и производится чтение одного или нескольких байт данных.

При таком способе чтения первым передается байт данных, адрес которого содержался во внутреннем указателе адреса микросхемы. Недостатком такого способа является то, что не всегда заранее известно, на какой байт указывает внутренний указатель микросхемы (например, если на шине имеется более одного ведущего устройства).

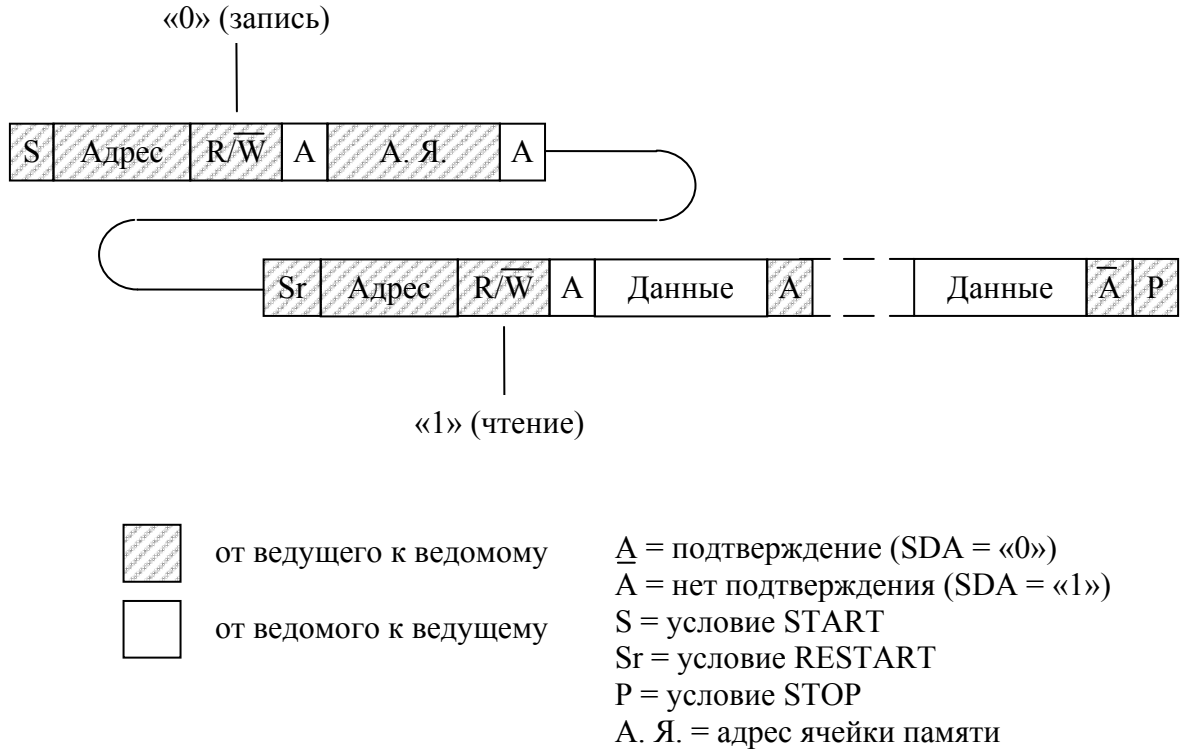


Рис. 2.15. Чтение данных из MK41T56

Более подробную информацию можно найти в [4].

ГЛАВА 3. ШИНА I2C В МИКРОКОНТРОЛЛЕРАХ СЕМЕЙСТВА PIC

§ 3.1. Описание микроконтроллера и среды разработки

§ 3.1.1. Общие сведения о PIC16F877

Контроллеры PIC имеют высокопроизводительную RISC-архитектуру, позволяющую выполнять любую из 35 инструкций за один цикл (кроме команд перехода – они выполняются за 2 цикла). Память данных и память программ имеют раздельные шины данных и адреса, что позволяет осуществлять параллельный доступ и организовывать конвейерное выполнение инструкций.

Основные характеристики:

- высокоскоростная RISC-архитектура
- 35 инструкций
- все команды выполняются за один цикл (4 такта), кроме инструкций переходов
- максимальная тактовая частота 20 МГц (тактовый сигнал), 200 нс (один машинный цикл)
- до 8к x 14 слов FLASH памяти программ
- до 368 x 8 бит памяти данных (ОЗУ)
- до 256 x 8 бит EEPROM памяти данных
- система прерываний – до 14 источников
- 8-уровневый аппаратный стек (13 разрядов – все команды занимают 1 слово ПП)
- прямой, косвенный и относительный режимы адресации
- сброс по включению питания (POR)
- таймер сброса (PWRT) и таймер ожидания запуска генератора (OST) после включения питания
- сторожевой таймер (WDT) с собственным RC-генератором
- программируемая защита памяти программ
- режим энергосбережения SLEEP
- программирование в готовом устройстве (используется два вывода МК)
- низковольтный режим программирования
- режим внутрисхемной отладки (используется два вывода МК)
- напряжение питания 2.0…5.5 В
- энергопотребление: ~ 0.6 мА (3.0В, 4 МГц)
  20 мкА (3.0В, 32 кГц)

< 1 мкА (режим энергосбережения)

- таймер 0: 8-разрядный таймер/счетчик с 8-разрядным программируемым предделителем
- таймер 1: 16-разрядный таймер/счетчик с возможностью подключения внешнего резонатора
- таймер 2: 8-разрядный таймер/счетчик с 8-разрядным программируемым предделителем и выходным делителем
- два модуля сравнения/захвата/ШИМ (CCP)
- 8-канальный 10-разрядный АЦП
- последовательный синхронный порт MSSP:
  ведущий/ведомый режим SPI
  ведущий/ведомый режим I2C
- последовательный синхронно-асинхронный приемопередатчик USART с поддержкой детектирования адреса
- ведомый 8-разрядный параллельный порт PSP с поддержкой внешних сигналов -RD, -WR, -CS (имеют инверсные значения)
- детектор пониженного напряжения (BOD) для сброса МК по снижению напряжения питания (BOR)

Память программ (ПП)

Микроконтроллеры PIC16F87x имеют 13-разрядный счетчик команд PC, способный адресовать 8К × 14 слов ПП. Физически реализовано все 8К × 14 слов (FLASH ПП).

Адрес вектора сброса – 0000h.

Адрес вектора прерываний – 0004h.

Память данных (ПД)

Таблица 3.1

Карта памяти микроконтроллера PIC16F877

INDF*	00h	**INDF***	80h	**INDF***	100h	**INDF***	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	**PCL**	82h	**PCL**	102h	**PCL**	182h
STATUS	03h	**STATUS**	83h	**STATUS**	103h	**STATUS**	183h
FSR	04h	**FSR**	84h	**FSR**	104h	**FSR**	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD	08h	TRISD	88h		108h		188h
PORTE	09h	TRISE	89h		109h		189h

Окончание табл. 3.1

PCLATH	0Ah	**PCLATH**	8Ah	**PCLATH**	10Ah	**PCLATH**	18Ah
INTCON	0Bh	**INTCON**	8Bh	**INTCON**	10Bh	**INTCON**	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Резерв(1)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Резерв(1)	18Fh
T1CON	10h		90h	Регистры общего назначения 16 байт	110h	Регистры общего назначения 16 байт	190h
TMR2	11h	SSPCON2	91h				
T2CON	12h	PR2	92h				
SSPBUF	13h	SSPADD	93h				
SSPCON	14h	SSPSTAT	94h				
CCPR1L	15h		95h				
CCPR1H	16h		96h				
CCP1CON	17h		97h				
RCSTA	18h	TXSTA	98h				
TXREG	19h	SPBRG	99h				
RCREG	1Ah		9Ah				
CCPR2L	1Bh		9Bh				
CCPR2H	1Ch		9Ch				
CCP2CON	1Dh		9Dh				
ADRESH	1Eh	ADRESL	9Eh				
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
Регистры общего назначения 96 байт	20h	Регистры общего назначения 80 байт	A0h EFh	Регистры общего назначения 80 байт	120h 16Fh	Регистры общего назначения 80 байт	1A0h 1EFh
	7Fh	Доступ к 70h-7Fh	F0h FFh	Доступ к 70h-7Fh	170h 17Fh	Доступ к 70h-7Fh	1F0h 1FFh
Банк 0		Банк 1		Банк 2		Банк 3	

* – регистр косвенной адресации – не физический регистр (при использовании косвенной адресации: обращение к регистру, адрес которого находится в FSR).

1 – зарезервированные регистры, их нельзя использовать.

Закрашенные участки ПД не реализованы, значение при чтении 00h.

Жирным шрифтом выделены регистры, которые доступны из любого банка памяти.

Вся ПД (регистры) разделена на 4 банка, которые содержат регистры общего (РОН) и специального (SFR) назначения. Переключение между банками ведется битами RP1 и RP0 в регистре STATUS, которые являются старшими битами адреса ПД

Объем банков ПД – 128 байт (до адреса 7Fh).

В начале банка размещаются SFR, затем РОН, выполненные как статическое ОЗУ. Некоторые часто используемые SFR могут быть доступны из любого банка ПД (без предварительного выбора соответствующего банка памяти).

Регистры общего назначения (РОН)

Предназначены для записи/считывания данных в процессе работы. Обратиться к ним можно либо прямой адресацией, либо косвенной (через регистр FSR).

Регистры специального назначения (SFR – Special Function Registers)

С помощью SFR осуществляется управление функциями ядра МК и периферийными модулями (таймерами, АЦП, портами).

Все регистры – РОН и SFR – представлены в табл. 3.1.

Регистр STATUS

В регистре STATUS содержатся флаги состояния АЛУ, флаг причины сброса МК и биты управления банками памяти.

Регистр STATUS может быть адресован любой командой, как и любой другой регистр ПД. Если обращение выполняется командой, которая воздействует на флаги Z, DC и C, то изменение этих трех битов командой заблокировано. Эти биты устанавливаются/сбрасываются согласно логике ядра МК. Команды изменения регистра STATUS также не воздействуют на биты -TO и -PD. Поэтому, результат выполнения команды с регистром STATUS может отличаться от ожидаемого.

Поэтому при изменении битов регистра STATUS рекомендуется использовать команды, не влияющие на флаги АЛУ (SWAPF, MOVWF, BCF и BSF).

Примечание. Флаг заема имеет инверсное значение. Вычитание выполняется путем прибавления дополнительного кода второго операнда. При выполнении команд сдвига (RRF и RLF) бит C загружается старшим или младшим битом сдвигаемого регистра.

Регистр STATUS (адрес 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	**RP1**	**RP0**	**-TO**	**-PD**	**Z**	**DC**	**C**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **IRP**: Бит выбора банка при косвенной адресации
1 = банк 2, 3 (100h – 1FFh)
0 = банк 0, 1 (000h - 0FFh)

биты 6-5: **RP1:RP0**: Биты выбора банка при непосредственной адресации
11 = банк 3 (180h – 1FFh)
10 = банк 2 (100h – 17Fh)
01 = банк 1 (080h – 0FFh)
00 = банк 0 (000h – 07Fh)

бит 4: **-TO**: Флаг переполнения сторожевого таймера
1 = после POR или выполнения команд CLRWDT, SLEEP
0 = после переполнения WDT

бит 3: **-PD**: Флаг включения питания
1 = после POR или выполнения команды CLRWDT
0 = после выполнения команды SLEEP

бит 2: **Z**: Флаг нулевого результата
1 = нулевой результат выполнения арифметической или логической операции
0 = не нулевой результат выполнения арифметической или логической операции

бит 1: **DC**: Флаг десятичного переноса/заема (для команд ADDWF, ADDWL, SUBWF, SUBWL), заем имеет инверсное значение
1 = был перенос из младшего полубайта
0 = не было переноса из младшего полубайта

бит 0: **C**: Флаг переноса/заема (для команд ADDWF, ADDWL, SUBWF, SUBWL), заем имеет инверсное значение
1 = был перенос из старшего бита
0 = не было переноса из старшего бита

Регистр PIR1

Регистр PIR1 доступен для чтения и записи, содержит биты прерываний периферийных модулей.

Регистр PIR1 (адрес 0Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	**ADIF**	**RCIF**	**TXIF**	**SSPIF**	**CCP1IF**	**TMR2IF**	**TMR1IF**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **PSPIF** : Флаг прерывания от ведомого параллельного порта
1 = произошла операция чтения или записи (сбрасывается программно)
0 = операции чтения или записи не происходило

бит 6: **ADIF**: Флаг прерывания от модуля АЦП
1 = преобразование АЦП завершено
0 = преобразование АЦП не завершено

бит 5: **RCIF**: Флаг прерывания от приемника USART
1 = буфер приемника USART полон
0 = буфер приемника USART пуст

бит 4: **TXIF**: Флаг прерывания от передатчика USART
1 = буфер передатчика USART пуст
0 = буфер передатчика USART полон

бит 3: **SSPIF**: Флаг прерываний от модуля MSSP
1 = выполнено условие возникновения прерывания от модуля MSSP (сбрасывается программно).
Условия возникновения прерывания:
- SPI
  - Выполнен прием/передача данных.
- Ведомый I2C
  - Выполнен прием/передача данных.
- Ведущий I2C
  - Выполнен прием/передача данных.
  - Завершено формирование на шине бита START.
  - Завершено формирование на шине бита STOP.
  - Завершено формирование на шине бита повторный START.
  - Завершено формирование на шине бита подтверждения.
  - Обнаружено на шине формирование бита START (для режима с несколькими ведущими).
  - Обнаружено на шине формирование бита STOP (для режима с несколькими ведущими).

0 = условие возникновения прерывания от модуля MSSP не выполнено

бит 2: **CCP1IF**: Флаг прерывания от модуля CCP1
Режим захвата
1 = выполнен захват значения TMR1 (сбрасывается программно)
0 = захвата значения TMR1 не происходило
Режим сравнения
1 = значение TMR1 достигло указанного в регистрах CCPR1H:CCPR1L(сбрасывается программно)
0 = значение TMR1 не достигло указанного в регистрах CCPR1H:CCPR1L
ШИМ режим
Не используется

бит 1: **TMR2IF**: Флаг прерывания по переполнению TMR2
1 = произошло переполнение TMR2 (сбрасывается программно)
0 = переполнения TMR2 не было

бит 0: **TMR1IF**: Флаг прерывания по переполнению TMR1
1 = произошло переполнение TMR1 (сбрасывается программно)
0 = переполнения TMR1 не было

Примечание. Флаги прерываний устанавливаются при возникновении условий прерываний вне зависимости от соответствующих битов разрешения и бита общего разрешения прерываний GIE (INTCON<7>). Программное обеспечение пользователя должно сбрасывать соответствующие флаги при обработке прерываний от периферийных модулей.

Регистр PIR2

Регистр PIR2 доступен для чтения и для записи, содержит флаги прерываний от модуля CCP2, возникновения коллизий на шине и окончания записи в EEPROM память данных.

Примечание. Флаги прерываний устанавливаются при возникновении условий вне зависимости от соответствующих битов разрешения и бита общего разрешения прерываний GIE (INTCON<7>). Программное обеспечение пользователя должно сбрасывать соответствующие флаги при обработке прерываний от периферийных модулей.

Регистр PIR2 (адрес 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	**Резерв**	-	**EEIF**	**BCLIF**	-	-	**CCP1IF**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **Не реализован**: читается как '0'

бит 6: **Резерв**: всегда должен равняться нулю

бит 5: **Не реализован**: читается как '0'

бит 4: **EEIF**: Флаг прерывания по окончанию записи в EEPROM данных
1 = запись в EEPROM данных завершена (сбрасывается программно)
0 = запись в EEPROM данных не завершена или не была начата

бит 3: **BCLIF**: Флаг прерывания возникновения коллизий на шине
1 = на шине обнаружены коллизии (только в режиме ведущего I^2C)
0 = коллизий не обнаружено

биты 2-1: **Не реализованы**: читаются как '0'

бит 0: **CCP2IF**: Флаг прерывания от модуля CCP2
Режим захвата
1 = выполнен захват значения TMR1 (сбрасывается программно)
0 = захвата значения TMR1 не происходило
Режим сравнения
1 = значение TMR1 достигло указанного в регистрах CCPR2H:CCPR2L(сбрасывается программно)
0 = значение TMR1 не достигло указанного в регистрах CCPR2H:CCPR2L
ШИМ режим
Не используется

§ 3.1.2. Краткое описание отладочных средств

Существует множество инструментальных средств, предназначенных для облегчения и интенсификации труда разработчиков. Основное назначение любого отладчика – помочь разработчику разобраться на программно-аппаратном уровне в процессах происходящих в устройстве, а затем добиться желаемых характеристик функционирования. Кроме этого инструментальные средства могут обладать дополнительными возможностями, которые избавляют разработчика от множества утомительных процедур.

Наиболее универсальными и мощными средствами отладки микропроцессорных систем на сегодня являются внутрисхемные эмуляторы, к которым относится и MPLAB ICD компании Microchip Technology Inc.

MPLAB IDE (Integrated Development Environment) – интегрированная среда разработки, представляет собой программный продукт, работающий под управлением операционной системы WINDOWS и предназначенный для написания, отладки и оптимизации программ для Microchip PIC микроконтроллеров. MPLAB обеспечивает полную среду разработки и включает в себя:

- MPLAB Project Manager– менеджер проекта;
- MPLAB Editor – текстовый редактор;

- MPLAB-SIM Simulator – симулятор;
- MPASM – универсальный Ассемблер, а так же MPLINK (Линковщик) и MPLIB (Библиотекарь);
- MLPAB-ICE Emulator – эмулятор;
- программаторы PICSTART Plus и PRO MATE 2;
- а также инструментальные средства третьих лиц – большое количество других компаний делают инструментальные средства разработки, работающие с MPLAB.

MPLAB-ICD (In Circuit Debugging) – внутрисхемный отладчик, позволяет выполнять внутрисхемную отладку программы в реальной схеме, используя генератор и периферию отлаживаемого контроллера. Для отладки используются только два вывода контроллера, обмен данными идёт по ICSPTM интерфейсу.

Используя ICD, можно отлаживать аппаратно-зависимые участки кода, которые трудно воспроизвести в симуляторе, например: работа с АЦП, измерение временных параметров входного сигнала, организация обратной связи с управляемым объектом, отладка интерфейсов USART, SPI, I2C и т. п.

MPLAB-ICD

MPLAB-ICD работает под управлением универсальной программной среды MPLAB и обладает следующими возможностями:

- связь с компьютером по RS-232;
- отладка в режиме реального времени и пошаговая отладка;
- одна задаваемая точка останова;
- просмотр и модификация содержимого управляющих регистров, RAM и EEPROM;
- внутрисхемная отладка и встроенная система программирования PIC-контроллеров серии PIC16F87x;
- работа от источника питания отлаживаемой конструкции в диапазоне от 3,0 до 5,5 В;
- диапазон рабочих частот от 32 кГц до 20 МГц.

В комплект MPLAB-ICD входят:

1. Эмуляционная плата MPLAB ICD header с микроконтроллером PIC16F877-20/P;
2. Основной модуль MPLAB ICD module;
3. Демонстрационная макетная плата MPLAB ICD demo board;
4. Кабель для подключения к компьютеру через RS 232;
5. 40-выводная и 28-выводнвя панели для подключения микроконтроллера или эмуляционной платы;

6. 9-дюймовый 6-выводной кабель;
7. CD с программным обеспечением и документацией;
8. Инструкции по эксплуатации.

Внешний вид комплекта MPLAB ICD показан на рис. 3.1.

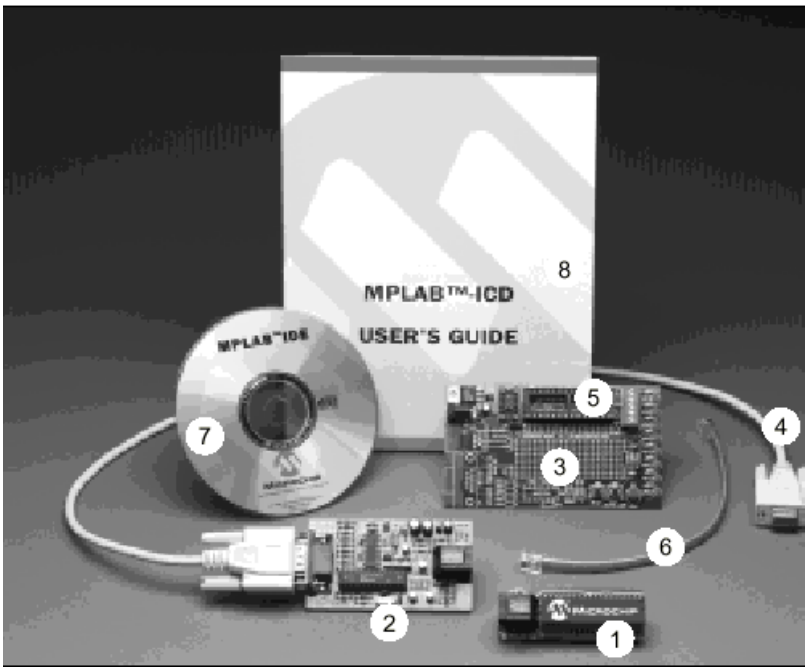


Рис. 3.1. Внешний вид комплекта MPLAB ICD

Вследствие использования внутрисхемного программирования, MPLAB-ICD задействует следующие ресурсы микроконтроллера:

1. MCLR/Vpp используется для программирования;
2. RB6 и RB7 зарезервированы для программирования и внутрисхемной отладки;
3. Шесть регистров общего назначения зарезервированы для DEBUG MONITOR – 70h, 1EBh – 1EFh;
4. Первая ячейка памяти программ должна содержать инструкцию NOP;
5. Память программ с адреса 0x1F00 по 0x1FFF зарезервирована для кода отладки;
6. Один уровень стека недоступен.

Примечание. MPLAB-ICD не поддерживает низковольтное программирование. При использовании ICD функция низковольтного программирования должна быть отключена

Демонстрационная плата MPLAB ICD DEMO BOARD

Демонстрационная плата предназначена для демонстрации PIC16F8XX и изучения его возможностей. Плата подключается к основному модулю через эмуляционную плату MPLAB-ICD Header.

PIC16f8XX может быть вставлен непосредственно в демонстрационную плату в обход эмуляционной головки.

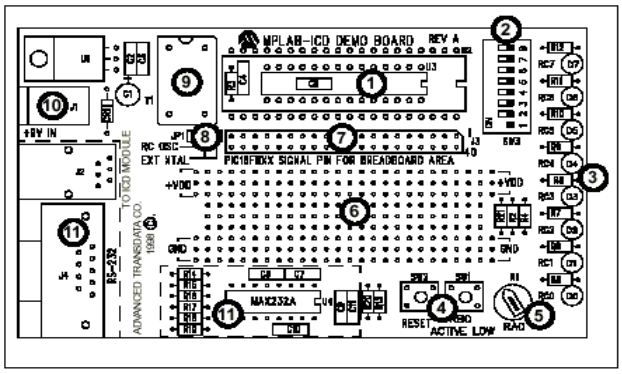


Рис. 3.2. Демонстрационная плата MPLAB ICD DEMO BOARD

На плате имеются:

1. 40- и 28-штырьковые разъемы;
2. Восемь DIP переключателей для соединения/разъединения каждого из восьми светодиодов с ответствующей линией порта C;
3. Восемь красных светодиодов, подключенных к порту C для отображения 8-битных двоичных величин;
4. Две кнопки. Одна для сброса, вторая подает внешнее воздействие (лог. «1») на RB0;
5. Потенциометр для подачи аналогового сигнала заданного уровня на RA0;
6. Область макетирования;
7. Разъем подключения внешних устройств к ножкам микроконтроллера – для расширения макетирования;
8. Перемычка выбора генератора – внутренний RC генератор (приблизительно 2 МГц) или внешний кристалл;
9. Разъем для внешнего кристалла;
10. Разъем питания – 9В, 0,75А;
11. Выход интерфейса RS-232;
12. Разъем для подключения основной платы MPLAB ICD.

Работа в среде MPLAB и работа с MPLAB-ICD

Работа с MPLAB возможна в двух режимах: MPLAB SIM – Simulator и MPLAB ICD Debugger. В первом режиме программа функционирует в режиме эмуляции PIC микроконтроллера, во втором работа идет с внутрисхемным отладчиком (непосредственно с микроконтроллером).

Работа с MPLAB идет в рамках проекта, который включает в себя файлы с исходным текстом, конечные файлы (*.HEX) и подключаемые библиотеки.

Создание нового проекта

1. Выберите из меню Project > New Project.
2. В появившемся диалоге (см. рис. 3.3) необходимо ввести имя проекта в поле «File Name» например «student.pjt».
3. Нажмите кнопку «ОК».
4. В следующем диалоге (см. рис. 3.4) в поле «include Path» указать путь к фалу «p16f877.inc».

Рис. 3.3. Диалоговое окно нового проекта

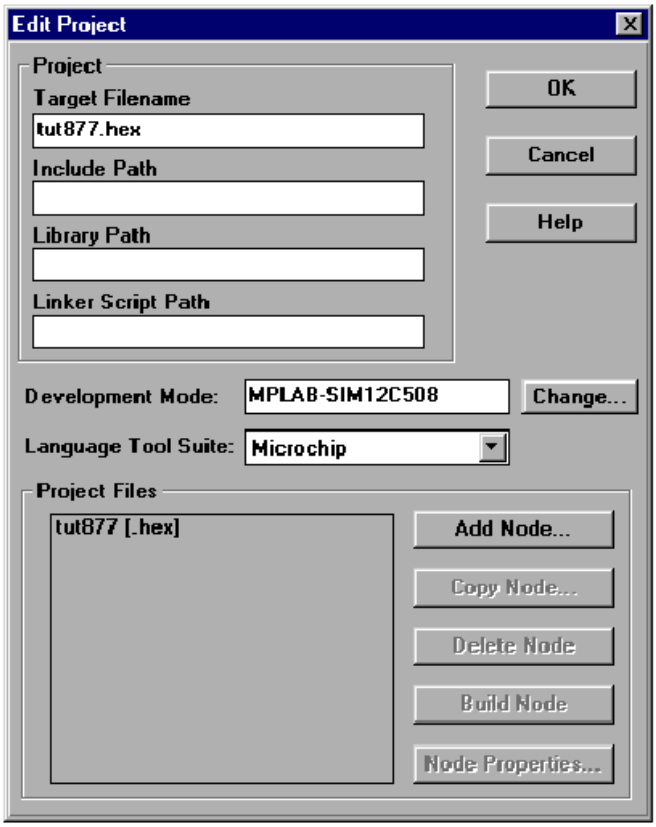


Рис. 3.4. Диалоговое окно редактирования проекта

5. Нажмите «Change»; появится окно опций MPLAB-ICD (см. рис. 3.5).

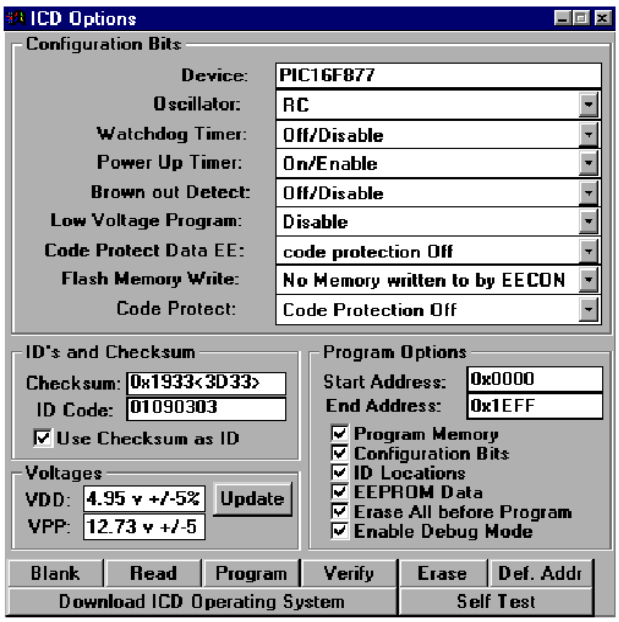


Рис. 3.5. Окно опций MPLAB-ICD

6. В поле «Processor» выберите PIC16F877
7. Выберите нужный режим работы MPLAB IDE – режим эмуляции (MPLAB SimSimulator) или режим внутрисхемного программирования (MPLAB-ICD – Debugger). В начале работы выберите режим «Simulator», затем, когда проект будет набран, скомпилирован и отлажен в симуляторе, переключите MPLAB в режим «MPLAB-ICD Debugger». Настройки (MPLAB-ICD Debugger) приведены ниже: Device – PIC16f877 Oscillator – RC Watchdog Timer – OFF Power On Timer – ON Brown out Detect – OFF Low Voltage Program – Disable Code Protect – Data EE code protection off Flash Memory – Write No Memory written to by EECON Code Protect – Code protection OFF
8. Нажмите «Add Node».
9. В поле «File Name» введите имя исходного ASM файла. Рекомендуется называть файл исходных текстов тем же именем, что и проект.
10. Выберите из меню File>New.
11. Выберите File>Save As… в появившемся диалоге в поле «File Name» укажите имя файла, совпадающее с именем файла исходных текстов указанного в проекте.

Функции, необходимые для работы MPLAB-IDE/ICD и клавиши быстрого доступа к ним

1. Изменение параметров проекта – Project>Edit Project (CTRL+F3).
2. Компиляция проекта – Project>Make Project (F10).
3. Пошаговое выполнение программы – Debug>Run>Step (F7).
4. Остановка выполнения программы – Debug>Run>Halt (F5).
5. Сброс контроллера – Debug>Run>Reset (F6)

6. Запуск на выполнение – Debug>Run>Run (F9)

Для просмотра состояния микроконтроллера и значений регистров в пакете MPLAB предусмотрены так называемые «Окна просмотра».

Доступ к любому из них осуществляется через пункт меню «*WINDOWS*».

Special function register window – окно регистров специальных функций.

Program memory – Окно отображающее содержимое памяти программ.

Absolute listing – Листинг программы.

Stack – Окно стека.

Symbol list – Окно отображающее присвоенные имена.

Project – Окно позволяющее редактировать настройки проекта.

Watch windows – Данный пункт позволяет создать собственное «окно просмотра»:

- *New watch window* – создание нового окна;
- *Load watch window* – загрузка сохраненного окна;
- *Add to active watch window* – Добавить переменную к активному окну;
- *Edit active watch window* – Редактировать активное окно;
- *Save active watch window* – сохранить активное окно.

При создании нового окна в меню необходимо выбрать переменную, которую необходимо отслеживать и нажать на кнопку ADD. При необходимости можно редактировать параметры отображения данной переменной.

Краткие сведения об ассемблере MPASM

Формат представления данных

В MPASM возможно представление данных не только в численном виде, но и в форме выражений. Ниже приведены некоторые элементы, которые могут быть в выражениях:

– минус;
* умножение;
/ деление;
+ прибавить;
<< сдвиг влево;
>> сдвиг вправо;
= равно;
=<> не равно.

Числа задаются в следующем формате. (Для примера взято число $162_{(10)} = A2_{(16)} = 242_{(8)} = 10100010_{(2)}$).

Двоичные (Binary) числа: 0b10100010 или b’10100010’ – эти записи эквивалентны.

Восьмеричные (Octal) числа: o’242’ или 242o – эти записи эквивалентны (признак восьмеричного числа – буква o, а не цифра ноль).

Десятичные (Decimal) числа: .162, или d’162’ – эти записи эквивалентны.

Шестнадцатеричные (Hexadecimal) числа: 0xA2, или h’A2’, или 0A2h – все эти записи эквивалентны (в последней форме записи число должно начинаться с 0..9).

ASCII коды: a’B’, или ’B’ – в обоих случаях Ассемблер сгенерирует ASCII код буквы B.

Основные директивы Ассемблера

Кроме команд процессора MPASM, как и другие языки Ассемблера позволяет использование специальных управляющих слов – директив. Ниже дано описание и формат представления некоторых директив MPASM.

<label> **equ** <expr> эта директива определяет константу.

Пример:

four equ 4 ;присваивает имени four значение 4.

<label> **set** <expr> эта директива определяет переменную (подобна equ, но может быть переопределена другой директивой set).

Пример:

width set 0x12

length set 0x14

area set length width*

[<label>] **org** <address> расположить программу с адреса address.

Пример:

int_1 org 0x20 ;Метка int_1 организована с адреса 20h

nop ;Начало подпрограммы

include <<filename>> или include «<include_file>« – подключает другой исходный файл, (включение может быть вложенным).

Пример:

include «c:\sys\sysdefs.inc» ;подключить системные определения

include <regs.h> ;подключить описания регистров

list [<list_option>, …, <list_option>] директива позволяющая управлять листингом. Запускается с ключами; рассмотрим только ключ **p** – тип процессора.

Пример:

list p=16f877

banksel <const> – по этой директиве Ассемблер генерирует команды выбора банка, в котором находится предварительно определенная константа – const.

Пример:
banksel TRISA
movwf TRISA

end – конец ассемблерной программы, все, что после этой директивы будет игнорировано.

Пример:
list p=16f877
. *; выполняемый код*
: *;*
end; Конец команд

Система команд PIC16F877

Система команд PIC16F877 показана в табл. 3.2. Всего имеется 35 инструкций. Инструкции могут быть байт-ориентированными (операндом является байт) и бит-ориентированные (операнд – отдельный бит регистра).

Таблица 3.2

Система команд PIC16F877

№	Мнемокод	Операнд	Выполняемая операция	Изменяемые флаги	Циклы	Примечания
Байт-ориентированные команды с регистровым файлом						
1.	ADDWF f,d	0<f<127 d-[0,1]	Сложение W с f	C, DC, Z	1	1,2
2.	ANDWF f,d	0<f<127 d-[0,1]	Логическое И W и f		1	1,2
3.	CLRF f	0<f<127	Сброс регистра f	Z	1	2
4.	CLRW	–	Сброс регистра W	Z	1	
5.	COMF f,d	0<f<127 d-[0,1]	Инверсия регистра f	Z	1	1,2
6.	DECF f,d	0<f<127 d-[0,1]	Декремент регистра f	Z	1	1,2
7.	DECFSZ f,d	0<f<127 d-[0,1]	Декремент f и пропуск следующей команды, если результат декремента равен 0	–	1(2)	1,2,3
8.	INCF f,d	0<f<127 d-[0,1]	Инкремент регистра f	Z	1	1,2

Продолжение табл. 3.2

№	Мнемокод	Операнд	Выполняемая операция	Изменяемые флаги	Циклы	Примечания
9.	INCFSZ f,d	0<f<127 d-[0,1]	Инкремент f и пропуск следующей команды, если результат декремента равен 0	–	1(2)	1,2,3
10.	IORWF f,d	0<f<127 d-[0,1]	Логическое ИЛИ W и f	Z	1	1,2
11.	MOVF f,d	0<f<127 d-[0,1]	Пересылка регистра f	Z	1	1,2
12.	MOVWF f	0<f<127	Пересылка W в f		1	
13.	NOP	–	Холостая команда	–	1	
14.	RLF f,d	0<f<127 d-[0,1]	Сдвиг f влево через перенос	C	1	1,2
15.	RRF f,d	0<f<127 d-[0,1]	Сдвиг f вправо через перенос	C	1	1,2
16.	SUBWF f,d	0<f<127 d-[0,1]	Вычитание W из f	C, DC, Z	1	1,2
17.	SWAPF f,d	0<f<127 d-[0,1]	Обмен местами тетрад в f	–	1	1,2
18.	XORWF f,d	0<f<127 d-[0,1]	Исключающее ИЛИ W и f	Z	1	1,2
Бит-ориентированные команды						
19.	BCF f,b	0<f<127 0<b<7	Сброс бита b в регистре f	–	1	1,2
20.	BSF f,b	0<f<127 0<b<7	Установка бита b в регистре f	–	1	1,2
21.	BTFSC f,b	0<f<127 0<b<7	Пропустить следующую команду, если бит b, в регистре f равен нулю	–	1(2)	3

Окончание табл. 3.2

№	Мнемокод	Операнд	Выполняемая операция	Изменяемые флаги	Циклы	Примечания
22.	BTFSS f,b	0<f<127 0<b<7	Пропустить следующую команду, если бит b, в регистре f равен единице	–	1(2)	3
Операции с константами и команды управления						
23.	ADDLW k	0<k<255	Сложение константы с W	C, DC, Z	1	
24.	ANDLW k	0<k<255	Логическое И константы и W	Z	1	
25.	CALL k	0<k<2047	Вызов подпрограммы	–	2	
26.	CLRWDT	–	Сброс сторожевого таймера (WDT)	TO,PD	1	
27.	GOTO k	0<k<2047	Переход по адресу k	–	2	
28.	IORLW k	0<k<255	Логическое ИЛИ константы и W	Z	1	
29.	MOVLW k	0<k<255	Пересылка константы в W	–	1	
30.	RETFIE	–	Возврат из прерывания.	–	2	
31.	RETLW k	0<k<255	Возврат из подпрограммы с загрузкой константы в W	–	2	
32.	RETURN	–	Возврат из подпрограммы	–	2	
33.	SLEEP	–	Переход в режим SLEEP	TO,PD	1	
34.	SUBLW k	0<k<255	Вычитание W из константы.	C, DC,Z	1	
35.	XORLW k	0<k<255	Исключающее ИЛИ константы и W	Z	1	

Примечания.

1. При выполнении операции «чтение-модификация-запись» с портом ввода/вывода исходные считываются с выводов порта, а не из выходных защелок. Например, если в выходной защелке было записано «1», а на соответствующем выходе низкий уровень сигнала, то обратно будет записано значение «0».
2. Если команда выполняется над регистром TMR0 (когда d=1, результат записывается в регистр таймера 0), то предделитель, будет обнулен.
3. Если счетчик программ (PC) изменяется или результат проверки условия истинен, то команда выполняется за два цикла. Во втором цикле выполняется команда NOP.

Таблица 3.3

Обозначения в таблице системы команд (табл. 3.2)

Символ	Описание
f	адрес регистра (0x00-0x7F) – файл
W:	Рабочий регистр
b:	Номер бита в 8-ми разрядном регистре
k:	Константа
x:	Не используется. Ассемблер формирует код с x=0
d:	Регистр назначения: d=0 – результат в регистре W d=1 – результат в регистре f. По умолчанию d=1
label:	Имя метки
TOS:	Вершина стека
PC:	Счетчик команд
TO:	Тайм-аут
PD:	Выключение питания
dest:	Регистр назначения: рабочий регистр W или регистр, заданный в команде
[]:	Необязательные параметры
():	Содержание
-->:	Присвоение
<>:	Битовое поле
∈:	Из набора

§ 3.2. Модуль MSSP

Модуль ведущего синхронного последовательного порта (MSSP) может использоваться для связи с периферийными микросхемами или другими микроконтроллерами. Периферийными микросхемами могут

быть: EEPROM память, сдвиговые регистры, драйверы ЖКИ, АЦП и др. Модуль MSSP может работать в одном из двух режимов:

- Последовательный периферийный интерфейс (SPI);
- Inter-Integrated Circuit (I2C).

§ 3.2.1. Режим I2C

Модуль MSSP полностью поддерживает все функции ведущих и ведомых устройств, включая поддержку общего вызова, аппаратные прерывания по детектированию битов START и STOP для определения занятости шины I2C в режиме ведущего (при конкуренции на шине). В MSSP модуле реализована поддержка стандартного режима 7, 10-разрядной адресации.

Фильтр «glitch» подключен к выводам SDA и SCL, когда они настроены на вход. Фильтр работает в режимах 100кГц и 400кГц. В режиме 100кГц, когда выводы SDA и SCL настроены на выход, фильтр контролирует длительность формируемых сигналов в не зависимости от тактовой частоты микроконтроллера.

Для работы с шиной I2C используется два вывода SCL (сигнал синхронизации) и SDA (данные). Выводы SDA и SCL автоматически настраиваются при включении режима I2C. Включение модуля MSSP выполняется установкой бита SSPEN (SSPCON<5>) в «1».

Для управления модулем MSSP в режиме I2C используется шесть регистров:

- SSPCON, регистр управления MSSP;
- SSPCON2, регистр управления 2 MSSP;
- SSPSTAT, регистр статуса MSSP;
- SSPBUF, буфер приемника/передатчика;
- SSPSR, сдвиговый регистр (пользователю не доступен);
- SSPADD, регистр адреса.

Регистр SSPSTAT (адрес 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	**CKE**	**D/-A**	**P**	**S**	**R/-W**	**UA**	**BF**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **SMP:** Фаза выборки бита

Ведущий режим SPI
1 = опрос входа в конце периода вывода данных
0 = опрос входа в середине периода вывода данных

Ведомый режим SPI
Для режима ведомого SPI этот бит всегда должен быть сброшен в '0'

Ведущий или ведомый режим I^2C
1 = управление длительностью фронта выключено в стандартном режиме (100кГц и 1МГц)
0 = управление длительностью фронта включено в скоростном режиме (400кГц)

бит 6: **CKE:** Выбор фронта тактового сигнала (см. рис. 9-2, 9-3 и 9-4)

SPI режим, CKP=0

1 = данные передаются по переднему фронту сигнала на выводе SCK

0 = данные передаются по заднему фронту сигнала на выводе SCK

SPI режим, CKP=1

1 = данные передаются по заднему фронту сигнала на выводе SCK

0 = данные передаются по переднему фронту сигнала на выводе SCK

Ведущий или ведомый режим I^2C

1= входные уровни соответствуют спецификации SMBus

0= входные уровни соответствуют спецификации I^2C

бит 5: **D/-A:** Бит Данные/Адрес (только для режима I^2C)

1 = последний принятый или переданный байт является информационным

0 = последний принятый или переданный байт является адресным

бит 4: **P:** Бит STOP (только для режима I^2C)

Этот бит сбрасывается в '0' когда модуль MSSP выключен, SSPEN=0.

1 = указывает, что бит STOP был обнаружен последним (этот бит равен '0' после сброса)

0 = бит STOP не является последним

бит 3: **S:** Бит START (только для режима I^2C)

Этот бит сбрасывается в '0' когда модуль MSSP выключен, SSPEN=0.

1 = указывает, что бит START был обнаружен последним (этот бит равен '0' после сброса)

0 = бит START не является последним

бит 2: **R/-W:** Бит чтения/записи (только для режима I^2C)

Значение бита действительно только после совпадения адреса и до приема бита START, STOP или -ACK.

Ведомый режим I^2C

1 = чтение

0 = запись

Ведущий режим I^2C

1 = выполняется передача данных

0 = передачи данных не происходит

Логическое ИЛИ этого бита с битами SEN, RSEN, PEN, RCEN или ACKEN укажет на неактивное состояние модуля MSSP.

бит 1: **UA:** Флаг обновления адреса устройства (только для режима 10-разрядного I^2C)

1 = необходимо обновить адрес в регистре SSPADD

0 = обновление адреса не требуется

бит 0: **BF:** Бит статуса буфера

Прием (SPI и I^2C режимы)

1 = прием завершен, буфер SSPBUF полон

0 = прием не завершен, буфер SSPBUF пуст

Передача (только I^2C режима)

1 = выполняется передача данных (исключая биты -ACK и STOP), буфер SSPBUF полон

0 = передача данных завершена (исключая биты -ACK и STOP), буфер SSPBUF пуст

Регистр SSPCON (адрес 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	**SSPOV**	**SSPEN**	**CKP**	**SSPM3**	**SSPM2**	**SSPM1**	**SSPM0**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **WCOL:** Бит конфликта записи

Ведущий режим

1 = запись в SSPBUF была выполнена при не выполнении условий шины I^2C

0 = конфликта не было

Ведомый режим

1 = была предпринята попытка записи в SSPBUF во время передачи предыдущего байта

0 = конфликта не было

бит 6: **SSPOV:** Бит переполнения приемника

SPI режим

1 = принят новый байт в то время как SSPBUF содержит предыдущие данные(байт в SSPSR будет потерян). В ведомом режиме пользователь должен прочитать содержимое регистра SSPBUF даже, если только передает данные. В ведущем режиме бит в '1' не устанавливается, т.к. каждая операция инициализируется записью в SSPBUF. (сбрасывается в '0' программно)

0 = нет переполнения

I^2C режим

1 = принят новый байт в то время как SSPBUF содержит предыдущие данные. Значение бита не действительно при передаче данных. (сбрасывается в '0' программно)

0 = нет переполнения

бит 5: **SSPEN:** Бит включения модуля MSSP

Когда модуль включен, соответствующие порты ввода/вывода настраиваются на выход или вход

SPI режим

1 = модуль MSSP включен, выводы SCK, SDO, SDI, -SS используются модулем MSSP

0 = модуль MSSP выключен, выводы работают как цифровые порты ввода/вывода

I^2C режим

1 = модуль MSSP включен, выводы SDA, SCL используются модулем MSSP

0 = модуль MSSP выключен, выводы работают как цифровые порты ввода/вывода

бит 4: **CKP:** Бит выбора полярности тактового сигнала

SPI режим

1 = пассивный высокий уровень сигнала

0 = пассивный низкий уровень сигнала

Ведомый режим I^2C

Управление тактовым сигналом SCK

1 = не управлять тактовым сигналом

0 = удерживать тактовый сигнал в низком логическом уровне (используется для подготовки данных)

Ведущий режим I^2C

Не имеет значения

биты 3-0: **SSPM3:SSPM0:** Режим работы модуля MSSP

0000 = ведущий режим SPI, тактовый сигнал = F_{OSC}/4

0001 = ведущий режим SPI, тактовый сигнал = F_{OSC}/16

0010 = ведущий режим SPI, тактовый сигнал = F_{OSC}/64

0011 = ведущий режим SPI, тактовый сигнал = выход TMR2 / 2

0100 = ведомый режим SPI, тактовый сигнал с вывода SCK. Вывод -SS подключен к MSSP

0101 = ведомый режим SPI, тактовый сигнал с вывода SCK. Вывод -SS не подключен к MSSP

0110 = ведомый режим I^2C, 7-разрядная адресация

0111 = ведомый режим I^2C, 10-разрядная адресация

1000 = ведущий режим I^2C, тактовый сигнал = F_{OSC}/(4 * (SSPADD+1))

1011 = программная поддержка ведущего режима I^2C (ведомый режим выключен)

1110 = программная поддержка ведущего режима I^2C, 7-разрядная адресация с разрешением прерываний по приему бит START и STOP

1111 = программная поддержка ведущего режима I^2C, 10- разрядная адресация с разрешением прерываний по приему бит START и STOP

1001, 1010, 1100, 1101 = резерв

Регистр SSPCON2 (адрес 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	**ACKSTAT**	**ACKDT**	**ACKEN**	**RCEN**	**PEN**	**RSEN**	**SEN**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **GCEN:** Бит разрешения поддержки общего вызова (только для ведомого режима I^2C)

1 = разрешить прерывания при приеме в регистр SSPSR адреса общего вызова (0000h)

0 = поддержка общего вызова выключена

бит 6: **ACKSTAT:** Бит статуса подтверждения (только для ведущего режима I^2C)

Передача ведущего I^2C

1 = подтверждения не было получено от ведомого

0 = подтверждение от ведомого было получено

бит 5: **ACKDT:** Бит подтверждения (только для ведущего режима I^2C)
Прием ведущего I^2C
Значение этого бита передается при разрешении формирование бита подтверждения.
1 = подтверждение
0 = нет подтверждения

бит 4: **ACKEN:** Сформировать бит подтверждения (только для ведущего режима I^2C)
1 = на выводах SCL, SDA формируется бит ACKDT. Аппаратно сбрасывается в '0'
0 = подтверждение не формируется

бит 3: **RCEN:** Разрешить прием данных (только для ведущего режима I^2C)
1 = разрешить прием данных с шины I^2C
0 = приемник выключен

бит 2: **PEN:** Сформировать бит STOP (только для ведущего режима I^2C)
1 = на выводах SCL, SDA формируется бит STOP. Аппаратно сбрасывается в '0'
0 = бит STOP не формируется

бит 1: **RSEN:** Сформировать бит повторный START (только для ведущего режима I^2C)
1 = на выводах SCL, SDA формируется бит повторный START. Аппаратно сбрасывается в '0'
0 = бит повторный STAT не формируется

бит 0: **SEN:** Сформировать бит START (только для ведущего режима I^2C)
1 = на выводах SCL, SDA формируется бит START. Аппаратно сбрасывается в '0'
0 = бит START не формируется

Примечание. Для битов ACKEN, RCEN, PEN, RSEN, SEN. Если I2C модуль находится в пассивном состоянии, то ни один из битов не может быть установлен в «1» (поставлен в очередь), не может быть выполнена запись в регистр SSPBUF (или запись в регистр SSPBUF заблокирована).

В регистре SSPCON устанавливается требуемый режим I2C. С помощью четырех битов (SSPCON<3:0>) можно выбрать один из режимов I2C:

- Ведомый режим I2C, 7-разрядная адресация;
- Ведомый режим I2C, 10-разрядная адресация;
- Ведущий режим I2C, тактовый сигнал FSCL = FOSC/(4 * (SSPADD+1));
- Программная поддержка ведущего режима I2C (реализовано для совместимости с другими PICmicro).

При выборе любого режима I2C выводы SCL и SDA должны быть настроены на вход, установкой соответствующих битов регистра TRISC в «1». После выбора режима I2C и установки бита SSPEN в «1» выводы SDA, SCL подключаются к модулю MSSP. Для нормальной работы модуля I2C к выводам SCL, SDA должны быть подключены внешние подтягивающие резисторы.

Бит CKE (SSPSTAT<6>) устанавливает уровни сигналов на выводах SCL, SDA в ведущем и ведомом режимах. Если CKE =1, то выходные уровни соответствуют спецификации SMBus. Когда CKE=0, выходные уровни соответствуют спецификации I2C.

Регистр SSPSTAT содержит биты статуса передачи данных: обнаружение на шине битов START (S) или STOP (P), флаг приема байта данных или адреса, указатель загрузки старшего байта 10-разрядного адреса, бит операции приема/передачи.

В регистр SSPBUF загружаются данные для передачи по шине I2C, и из него читаются принятые данные. Регистр SSPSR выполняет сдвиг принимаемых/передаваемых данных. При приеме данных регистры SSPBUF, SSPSR работают как двухуровневый буфер приемника. Буфер позволяет принимать следующий байт до чтения предыдущего принятого байта из регистра SSPBUF. Когда байт полностью загружен в SSPSR, он передается в регистр SSPBUF и устанавливается флаг прерывания SSPIF в «1». Если полностью принят следующий байт до чтения предыдущего байта из SSPBUF, то устанавливается бит SSPOV (SSPCON<6>) в «1», а байт в регистре SSPSR будет потерян.

В регистр SSPADD записывается адрес ведомого устройства. В 10-разрядном режиме пользователь должен сначала записывать старший байт адреса (1111 0 A9 A8 0). После совпадения старшего байта адреса необходимо загрузить младший байт адреса (A7:A0).

§ 3.2.2. Режим ведомого I2C

В режиме ведомого I2C выводы SCL, SDA должны быть настроены на вход. Модуль MSSP автоматически изменит направление вывода SDA при передаче данных ведомым. Структурная схема модуля MSSP в режиме ведомого I2C показана на рис. 3.6.

При совпадении адреса или после приема байта данных (если предварительно совпал адрес) аппаратно генерируется бит подтверждения (-ACK), а затем данные из регистра SSPSR загружаются в SSPBUF.

Существует несколько условий, при которых бит -ACK не формируется (эти условия могут возникать одновременно):

- бит BF (SSPSTAT<0>) = 1 перед приемом данных;
- бит переполнения SSPOV (SSPSTAT<6>) = 1 перед приемом данных.

Если бит BF = 1, то значение из SSPSR не переписывается в регистр SSPBUF, а биты SSPIF и SSPOV устанавливаются в «1». В табл. 3.4 показаны операции после приема байта при различных значениях битов BF, SSPOV.

Таблица 3.4

Операции после приема байта при различных значениях битов BF, SSPOV

Биты статуса приемника		Запись из SSPSR в SSPBUF	Формирование бита -ACK	Установка флага прерываний SSPIF
BF	SSPOV			
0	0	Есть	Есть	Есть
1	0	Нет	Нет	Есть
1	1	Нет	Нет	Есть
0	1	Есть	Нет	Есть

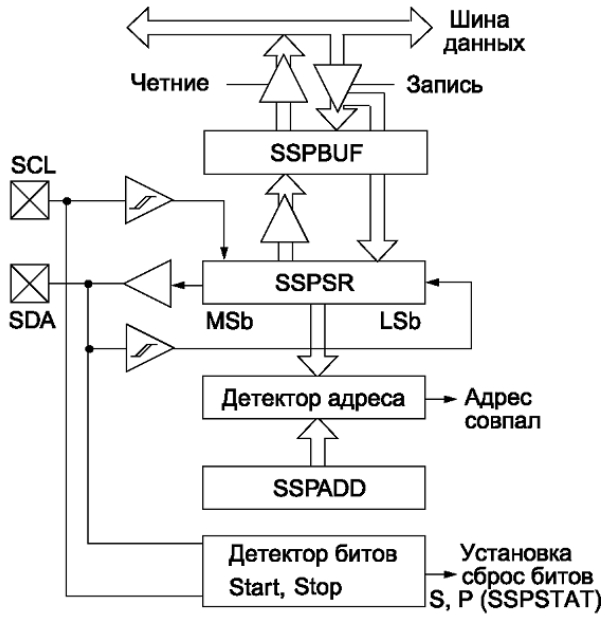


Рис. 3.6. Структурная схема модуля MSSP в режиме ведомого I2C

В затененных ячейках показана ситуация, когда вовремя не был сброшен бит переполнения SSPOV в «0». Заметьте, что бит BF аппаратно сбрасывается в «0» при чтении из регистра SSPBUF, а бит SSPOV необходимо сбрасывать в «0» программно.

Адресация

После включения модуля MSSP ожидается формирование на шине бита START. Получив бит START, принимается 8 бит в сдвиговый регистр SSPSR. Выборка битов происходит по переднему фронту синхронизирующего сигнала на выводе SCL. По заднему фронту восьмого такта сигнала SCL значение в регистре SSPSR<7:1> сравнивается с содержимым регистра SSPADD. Если значение адреса совпадает, а биты BF и SSPOV равны нулю, то выполняются следующие действия:

1. Значение регистра SSPSR загружается SSPBUF по 8-му заднему фронту сигнала SCL;
2. Устанавливается флаг BF в «1» (буфер полон) по 8-му заднему фронту сигнала SCL;
3. Генерируется бит -ACK;
4. Устанавливается флаг прерываний SSPIF (PIR1<3>) в «1» по 9-му заднему фронту сигнала SCL.

В режиме ведомого при 10-разрядной адресации необходимо принять два байта адреса. Пять старших бит первого байта определяют, является ли полученный байт первым байтом 10-разрядного адреса. Бит R/-W(SSPSTAT<2>) должен быть настроен для приема второго байта

адреса. Для 10-разрядной адресации первый байт адреса должен иметь формат '1111 0 A9 A8 0', где A9:A8 два старших бита адреса. Рекомендуемая последовательность действий при 10-разрядной адресации (шаги 7–9 для передачи ведомым):

1. Принять старший байт адреса (устанавливаются биты SSPIF, BF и UA (SSPSTAT<1> в «1»).
2. Записать младший байт адреса в регистр SSPADD (аппаратно сбрасывается бит UA в «0» и «отпускается» линия SCL).
3. Выполнить чтение из регистра SSPBUF (сбрасывается бит BF в «0») и сбросить флаг SSPIF в «0».
4. Принять младший байт адреса (устанавливаются биты SSPIF, BF и UA (SSPSTAT<1> в «1»).
5. Записать старший байт адреса в регистр SSPADD (аппаратно сбрасывается бит UA в «0» и «отпускается» линия SCL).
6. Выполнить чтение из регистра SSPBUF (сбрасывается бит BF в «0») и сбросить флаг SSPIF в «0».
7. Принять бит повторный START.
8. Принять старший байт адреса (устанавливаются биты SSPIF и BF в «1»).
9. Выполнить чтение из регистра SSPBUF (сбрасывается бит BF в «0») и сбросить флаг SSPIF в «0».

Примечание. В 10-разрядном режиме после команды повторный START (шаг 7) не требуется обновлять значение в регистре SSPADD. В данном случае требуется соответствие только первого байта адреса.

Прием данных ведомым

Если бит R/-W в адресном байте равен нулю, а принятый адрес совпадает с адресом устройства, то бит R/-W в регистре SSPSTAT сбрасывается в «0». Принятый адрес загружается в регистр SSPBUF.

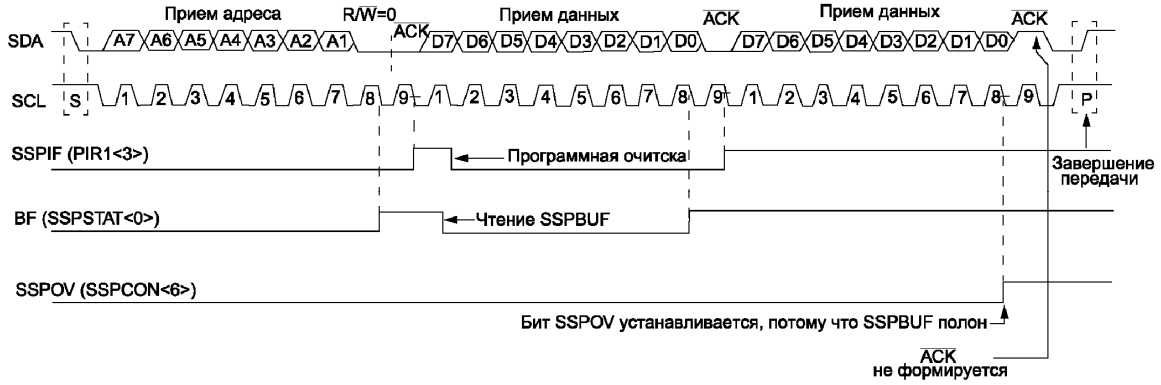


Рис. 3.7. Прием данных ведомым I2C (7-разрядная адресация)

Если бит BF (буфер полон) или SSPOV (переполнение буфера) установлен в «1», то бит подтверждения -ACK не формируется. Эту ошибку необходимо обработать программно.

Прерывание от модуля MSSP генерируются при каждом принятом байте с шины I2C, установкой флага SSPIF (PIR<3>) в «1» (сбрасывается программно). Регистр SSPSTAT используется для определения типа принятого байта.

Примечание. Значение регистра SSPBUF будет обновлено, если SSPOV=1 и BF=0. Если было выполнено чтение из регистра SSPBUF, но не был сброшен бит SSPOV в «0», то бит -ACK не формируется.

Работа в SLEEP режиме

Ведомый I2C может принимать адресные байты или байты данных в SLEEP режиме микроконтроллера. После приема байта микроконтроллер выходит из SLEEP режима, если разрешены прерывания от MSSP модуля.

Передача данных ведомым

Если бит R/-W в адресном байте равен «1», а принятый адрес совпадает с адресом устройства, то бит R/-W в регистре SSPSTAT устанавливается в «1». Принятый адрес загружается в регистр SSPBUF. Бит -ACK формируется девятым битом, после чего линия SCL удерживается в низком логическом уровне. Передаваемые данные должны быть записаны в регистр SSPBUF, после чего они автоматически переписываются в регистр SSPSR. После записи данных необходимо «отпустить» сигнал SCL установкой бита CKP(SSPCON<4>) в «1». Ведущий шины контролирует состояние линии SCL, ожидая смены уровня сигнала. Восемь бит загруженных данных последовательно сдвигаются по заднему фронту сигнала SCL, что гарантирует достоверное значение данных на линии SDA (см. рис. 3.8).

Модуль MSSP генерирует прерывание по каждому переданному байту, устанавливая бит SPPIF в «1» по заднему фронту девятого такта сигнала SCL. Флаг SSPIF должен быть сброшен программно. Регистр SSPSTAT используется для определения статуса передачи данных.

Ведущее устройство формирует бит подтверждения ACK на девятом такте сигнала SCL для каждого принятого байта. Если бит подтверждения ACK не сформирован (высокий уровень сигнала SDA), передача данных завершена. Логика ведомого устройства настраивается на обнаружение бита START.

Если бит подтверждения ACK был получен (низкий уровень сигнала SDA), в регистр SSPBUF необходимо записать новый байт для передачи. Линию SCL также необходимо «отпустить», установкой бита CKP в «1».

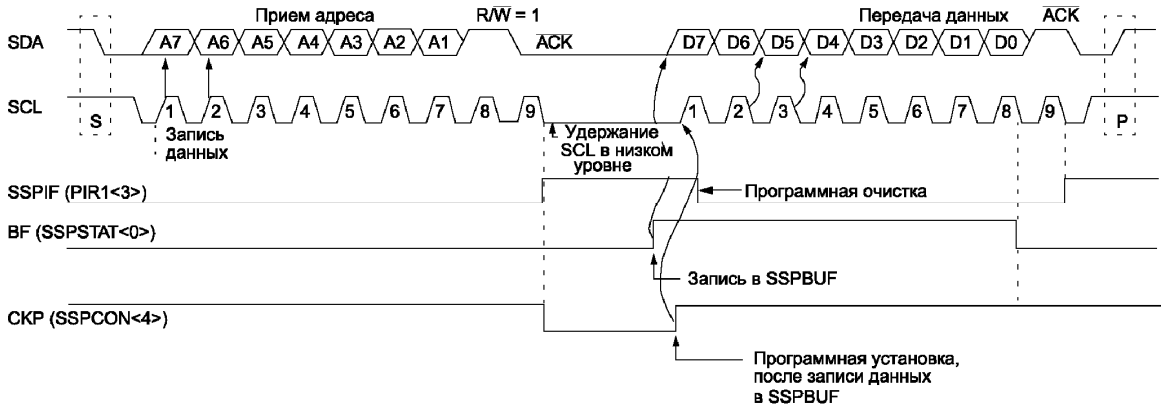


Рис. 3.8. Передача данных ведомым I2C (7-разрядная адресация)

Эффект сброса

При сбросе микроконтроллера модуль MSSP выключается, и прекращается любой обмен данными.

§ 3.2.3. Режим ведущего I2C

В режиме ведущего поддерживается генерация прерываний при обнаружении на шине битов START и STOP. Биты STOP (P) и START (S) в регистре SSPSTAT равны «0» после сброса микроконтроллера или при выключенном модуле MSSP. Шина находится в неактивном состоянии, если бит P=1 или оба бита S, P равны «0».

В режиме ведущего выводы SCL, SDA управляются аппаратно.

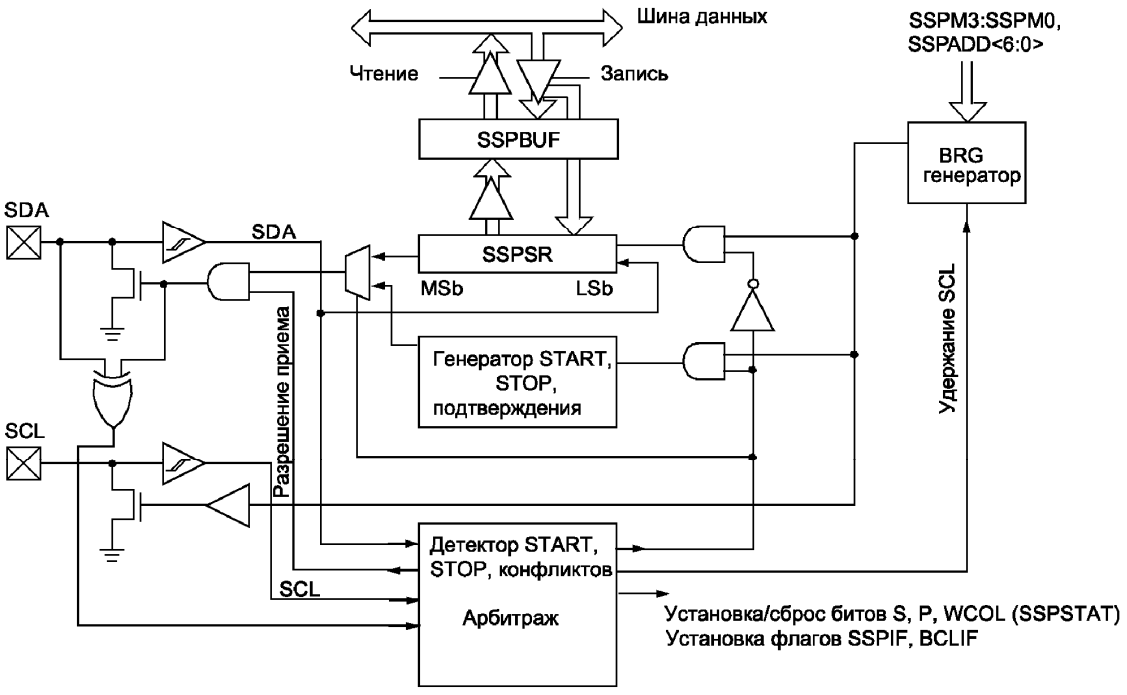


Рис. 3.9. Структурная схема модуля MSSP в режиме ведущего I2C

Следующие события на шине I2C могут привести к установке флага прерываний SSPIF в «1»:

- Выполнено условие START;
- Выполнено условие STOP;
- Передан/принят байт данных;
- Передан бит подтверждения;
- Выполнено условие повторный START.

Режим конкуренции

В режиме конкуренции прерывания после START и STOP позволяют определить, когда шина I2C свободна. Биты S и P сбрасываются в «0» при сбросе микроконтроллера или при выключении модуля MSSP. Управление шиной может быть перехвачено, когда бит P=1 или шина простаивает (S=0 и P=0). Если шина занята, можно разрешить прерывания от MSSP для обнаружения бита STOP на шине.

При конкуренции линия SDA должна проверяться на соответствие уровня, при ожидаемом высоком уровне на выходе. Эта проверка производится автоматически, а результат помещается в бит BCLIF (PIR2<3>).

Арбитраж на шине I2C может быть потерян во время:

- Передачи адреса;
- Передачи данных;
- Формирования бита START;
- Формирования бита повторный START;
- Формирования бита ACK.

Поддержка режима ведущего I2C

Ведущий режим включается соответствующей настройкой битов SSPM в регистре SSPCON и установкой в «1» бита SSPEN. После включения ведущего режима аппаратно могут выполняться следующие функции:

- Формирование бита START на линии SCL и SDA;
- Формирование бита повторный START на линии SCL и SDA;
- Записью в регистр SSPBUF инициализируется передача байта данных/адреса;
- Формирование бита STOP на линии SCL и SDA;
- Настройка порта I2C на прием данных;
- Формирование бита подтверждения ACK после приема байта на линии SCL и SDA.

Примечание. Модуль MSSP в ведущем режиме не имеет стека событий. Это означает, что пользователь не может, к примеру, инициро-

вать передачу бита START и произвести запись в SSPBUF до того, как START будет завершен. При попытке осуществления подобной операции будет установлен бит WCOL в «1», указывая, что запись в регистр SSPBUF не произошла.

Работа в режиме ведущего I2C

Ведущий формирует на шине I2C тактовый сигнал и биты START, STOP. Текущий обмен данными завершается после формирования бита STOP или повторный START. Поскольку бит повторный START инициирует новый обмен данными, шина I2C остается занятой.

Передатчик ведущего выдает данные на линию SDA, а тактовый сигнал на линию SCL. Первый передаваемый байт содержит 7-разрядный адрес приемника (при 7-разрядной адресации устройств) и бит направления данных R/-W=0. После каждого переданного байта принимается бит подтверждения ACK. Биты START и STOP формируется для указания начала и завершения передачи данных.

В режиме приема ведущим на шину I2C сначала выдается байт, содержащий 7-разрядный адрес передатчика (при 7-разрядной адресации устройств) и бит направления данных R/-W = 1. Данные принимаются с линии SDA, а на линии SCL формируется тактовый сигнал. После каждого принятого байта формируется бит подтверждения. Биты START и STOP формируется для указания начала и завершения передачи данных.

Генератор скорости обмена BRG используется для установки требуемой частоты тактового сигнала на линии SCL – 100кГц, 400кГц или 1МГц. Значение для перезагрузки BRG берется из 7 младших бит регистра SSPADD. BRG начинает работу сразу после записи данных в регистр SSPBUF. Как только операция завершена (передан последний бит байта и принят бит подтверждения) генератор BRG останавливается, вывод SCL «отпускается».

Рекомендованная последовательность действий при передаче данных:

1. Инициировать START установкой бита SEN в регистре SSPCON2.
2. Ожидать прерывание (если оно разрешено) или установку бита SSPIF после завершения выполнения START.
3. Записью в SSPBUF инициируется передача адреса.
4. 7 бит адреса (при 7-разрядной адресации) и бит направления данных выдается на SDA.
5. Принять подтверждение ACK от приемника, результат записывается в бит ACKSTAT регистра SSPCON2.
6. По заднему фронту девятого такта устанавливается бит SSPIF в «1».
7. Записью в SSPBUF инициируется передача данных.
8. 8 бит данных выдаются на SDA.

9. Принимается подтверждение ACK от приемника, результат записывается в бит ACKSTAT регистра SSPCON2.
10. По заднему фронту девятого такта устанавливается бит SSPIF в «1».
11. Инициировать STOP установкой бита PEN в регистре SSPCON2.
12. Ожидать прерывание (если оно разрешено) или установку бита SSPIF после завершения выполнения STOP.

Генератор скорости обмена

В ведущем режиме значение для перезагрузки BRG берется из младших 7 бит регистра SSPADD (см. рис. 3.10).

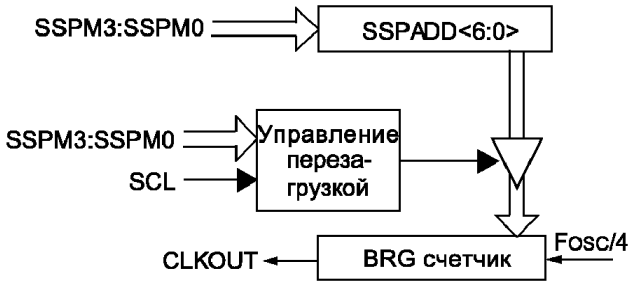


Рис. 3.10. Структурная схема генератора скорости обмена

После загрузки SSPADD в BRG, счетчик BRG считает, декрементируя до нуля (в тактах Q2 и Q4), и останавливается до следующей перезагрузки, которая не всегда производится автоматически. Если после окончания счета сигнал на линии SCL должен перейти в высокий уровень, перезагрузка производится только после этого перехода (см. рис. 3.11).

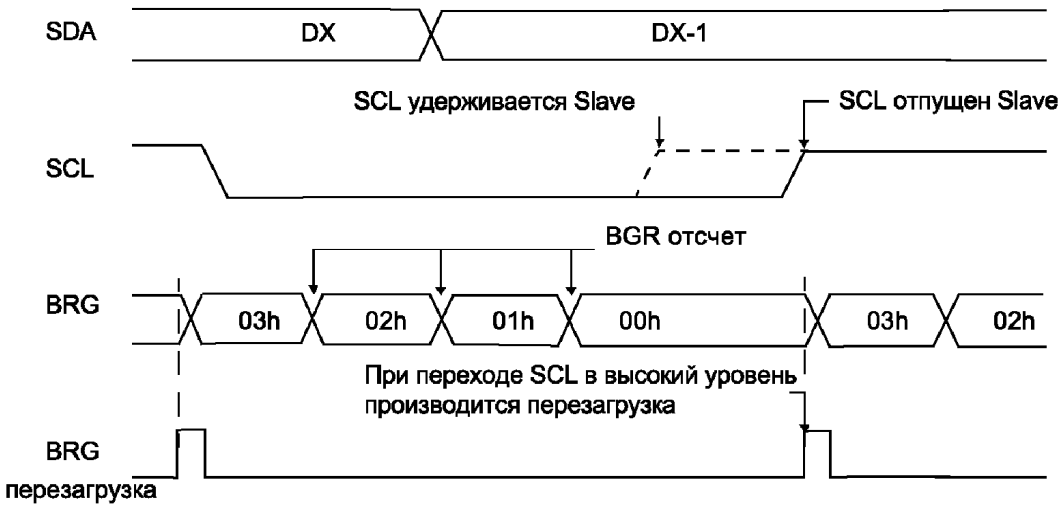


Рис. 3.11. Временная диаграмма работы BRG с арбитражем SCL

Примечание. Скорость обмена = FOSC / (4 · (SSPADD + 1))

Формирование бита START в режиме ведущего I2C

Чтобы инициировать формирование бита START на шине I2C, необходимо установить бит SEN (SSPCON2<0>) в «1». Если на линиях SCL и SDA высокий уровень сигнала, BRG загружается значением из регистра SSPADD <6:0> и начинает счет. Если по окончанию отсчета времени TBRG сохраняется высокий уровень на SCL и SDA, сигнал SDA переводится в низкий логический уровень. Перевод SDA в низкий уровень, в то время когда на линии SCL высокий, и есть бит START на шине I2C. После формирования бита START устанавливается бит S и флаг прерывания SSPIF в «1», BRG загружается новым значением и начинает счет. По окончании счета бит SEN (SSPCON2<0>) автоматически сбрасывается в «0», генератор останавливается, на SDA остается низкий уровень сигнала. Формирование бита START завершено.

Примечание. Если в начале формирования бита START на SDA или SCL присутствует низкий уровень или во время выполнения START низкий уровень на SCL появляется раньше, чем на SDA, устанавливается флаг прерывания BCLIF (конфликт шины), выполнение START прекращается, MSSP переходит в состояние ожидания.

Если во время формирования бита START производится попытка записи в SSPBUF, устанавливается бит WCOL, а запись не происходит.

Примечание. Поскольку MSSP не имеет стека событий, установка любого из младших 5 битов регистра SSPCON2 до завершения формирования бита START запрещена.

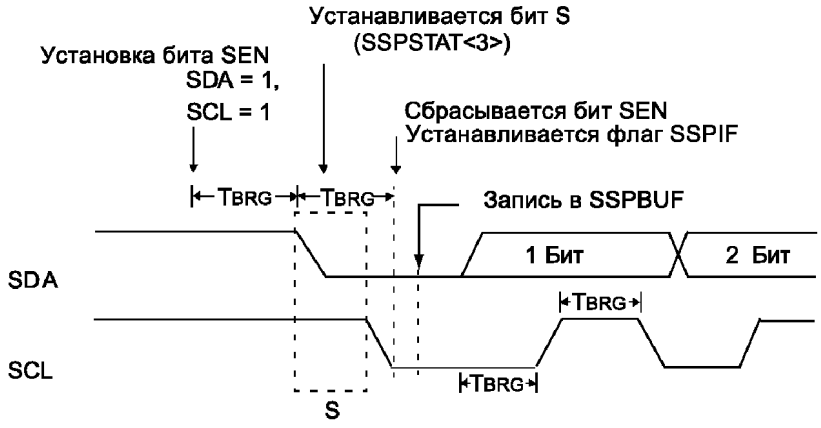


Рис. 3.12. Временная диаграмма формирования бита START

Формирование бита повторный START в режиме ведущего I2C

Чтобы инициировать формирование бита повторный START, необходимо установить бит RSEN (SSPCON2<1>) в то время, когда

модуль MSSP находится в режиме ожидания. При включении формирования бита повторный START линия SCL переводится в низкий логический уровень. Когда на SCL устанавливается низкий уровень сигнала, BRG перезагружается содержимым регистра SSPADD<6:0> и начинает отсчет, при этом SDA «отпускается» в высокий уровень. Если по окончанию счета BRG, на линии SDA сохраняется высокий уровень, SCL также «отпускается». BRG вновь перезагружается и начинает отсчет. Если по окончании отсчета времени TBRG сохраняется высокий уровень на линиях SCL и SDA, сигнал SDA переводится в низкий уровень. После формирования бита повторный START на шине I2C устанавливается бит S (SSPSTAT<3>) в «1». Флаг SSPIF не будет установлен в «1» до тех пор, пока BRG не перезагрузится новым значением и начнет счет.

Примечания:

1. Если бит RSEN установлен в «1» во время выполнения какой либо операции на шине, то не будет выполнено никаких действий.
2. Если на SDA низкий уровень при переходе SCL из низкого уровня в высокий или низкий уровень на SCL появляется раньше, чем на SDA, устанавливается флаг прерывания BCLIF (конфликт шины), формирование бита повторный START прекращается, MSSP переходит в состояние ожидания.

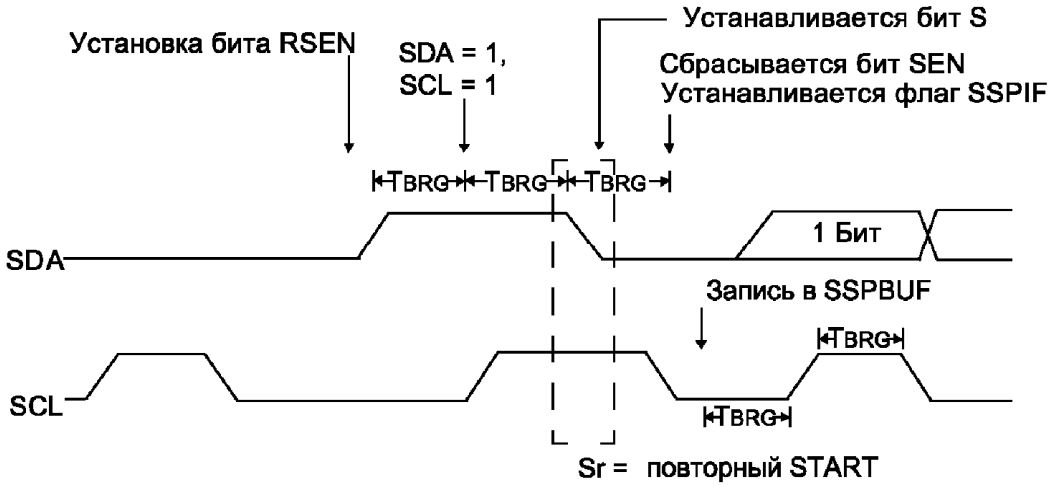


Рис. 3.13. Временная диаграмма формирования бита повторный START

Сразу после установки бита SSPIF пользователь может загрузить регистр SSPBUF 7-разрядным адресом (для 7-разрядного режима адресации) или старшим байтом 10-разрядного адреса. По завершении передачи 8 бит и получении подтверждения ACK, можно передать байт данных или младший байт адреса.

Если во время формирования бита повторный START производится попытка записи в SSPBUF, устанавливается бит WCOL, а запись не происходит.

Примечание. Поскольку MSSP не имеет стека событий, установка любого из младших 5 битов регистра SSPCON2 до завершения формирования бита повторный START запрещена.

Передача данных в режиме ведущего I2C

Для инициализации передачи байта данных, 7-разрядного адреса или любой части 10-разрядного адреса нужно просто записать байт в регистр SSPBUF. В результате чего установится бит BF в «1», а BRG начнет формировать сигнал для передачи данных. Каждый передаваемый бит будет выдаваться на SDA по заднему фронту сигнала SCL. Низкий уровень на SCL удерживается в течение одного периода BRG. Данные должны поступать на SDA до прихода переднего фронта на SCL. После отпускания SCL в высокий уровень на время TBRG данные должны удерживаться на SDA в том же состоянии. По окончании передачи 8-го бита сбрасывается флаг BF в «0», а ведущий «отпускает» SDA с тем, чтобы принять бит подтверждения. По заднему фронту 9-го такта значение ACK записывается в бит ACKSTAT регистра SSPCON2. В этот же момент устанавливается флаг SSPIF в «1», а BRG отключается до следующей операции на шине, оставляя низкий уровень на SCL и отпуская SDA (см. рис. 3.14).

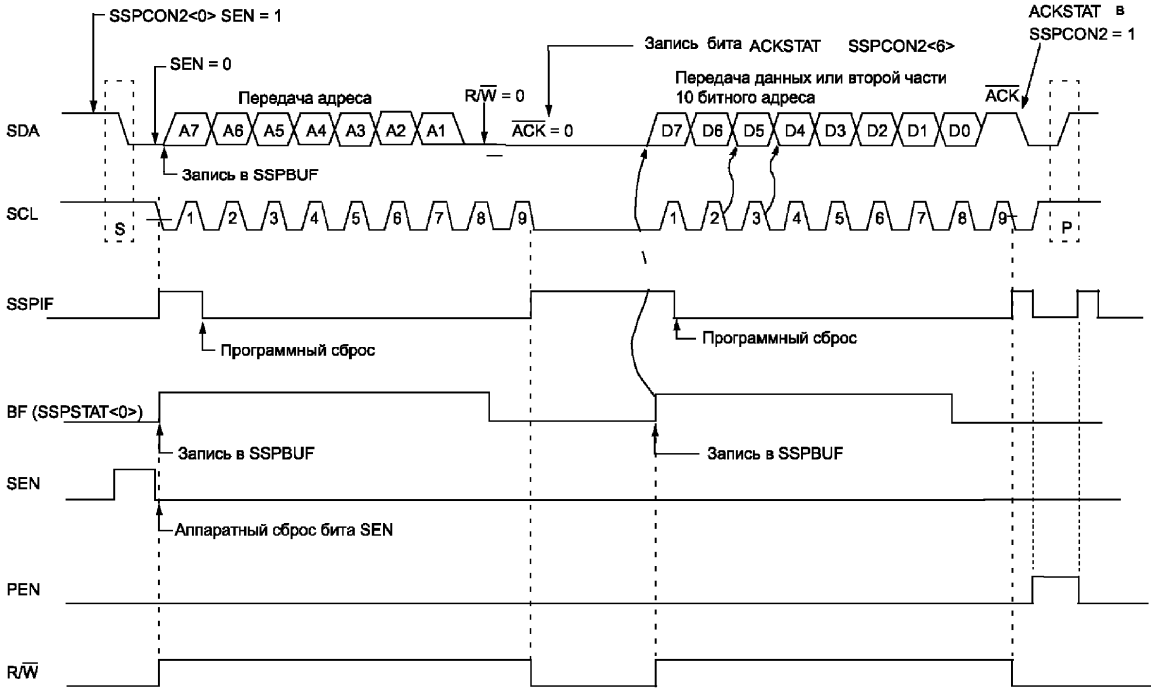


Рис. 3.14. Временная диаграмма передачи данных в режиме ведущего I2C

В режиме передачи данных бит BF (SSPSTAT<0>) аппаратно устанавливается в «1» после записи данных в регистр SSPBUF и аппаратно сбрасывается после передачи 8 бит данных.

Если во время передачи данных производится попытка записи в регистр SSPBUF, устанавливается бит WCOL в «1», а запись не происходит. Бит WCOL сбрасывается программно.

В режиме передачи данных бит ACKSTAT(SSPCON2<6>) равен нулю, если ведомый сформировал подтверждение. Ведомый посылает подтверждение, если он распознал адрес (включая общий вызов) или корректно принял данные.

Прием данных в режиме ведущего I2C

Прием данных ведущим шины I2C разрешается установкой бита RCEN(SSPCON2<3>) в «1».

Примечание. При установке бита RCEN в «1» модуль MSSP должен находиться в режиме ожидания.

BRG начинает формировать тактовый сигнал SCL для приема данных в сдвиговый регистр SSPSR. Каждый бит данных будет приниматься с SDA по заднему фронту SCL. По заднему фронту 8-го такта, значение из SSPSR переписывается в SPPBUF, устанавливается бит BF и SSPIF в «1», BGR останавливается, удерживая SCL в низком уровне, а модуль MSSP переходит в режим ожидания. После чтения регистра SSPBUF аппаратно сбрасывается бит BF в «0». По окончанию приема, ведущий может сформировать бит подтверждения установкой бита ACKEN (SSPCON2<4>) в «1».

В режиме приема данных бит BF (SSPSTAT<0>) аппаратно устанавливается в «1» после загрузки данных в регистр SSPBUF и аппаратно сбрасывается после чтения регистра SSPBUF.

При приеме данных бит SSPOV устанавливается в «1», если в момент приема 8-го бита следующего байта бит BF=1 после приема предыдущего байта.

Если во время приема данных производится попытка записи в регистр SSPBUF, устанавливается бит WCOL в «1», а запись не происходит. Бит WCOL сбрасывается программно.

Формирование бита подтверждения в режиме ведущего I2C

Для инициализации формирования бита подтверждения на шине I2C необходимо установить бит ACKEN (SSPCON2<4>) в «1». При установке этого бита на SCL выдается низкий уровень сигнала,

а на SDA содержимое бита ACKDT. Если нужно подтвердить прием, бит ACKDT должен быть равен нулю. По окончанию счета BRG линия SCL «отпускается». Как только SCL перейдет из низкого уровня в высокий, BRG опять начнет счет. После окончания счета SCL переводится в низкий уровень, бит ACKEN автоматически сбрасывается в «0», устанавливается флаг прерывания SSPIF в «1», BGR останавливается, а модуль MSSP переходит в режим ожидания (см. рис. 3.15).

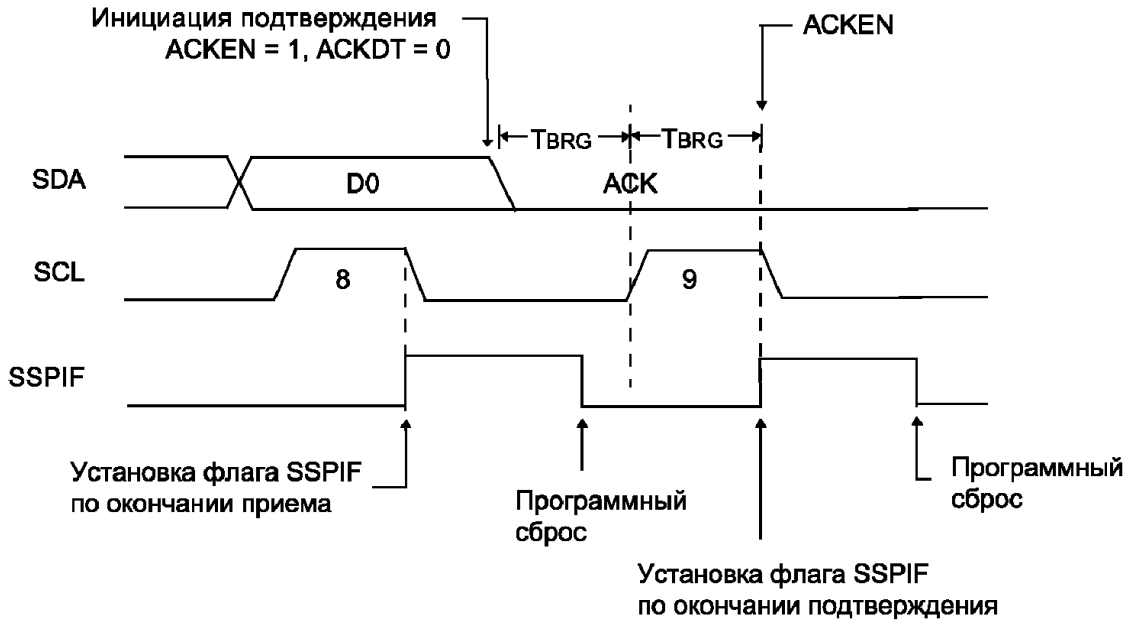


Рис. 3.15. Временная диаграмма формирования бита подтверждения

Если во время формирования бита подтверждения производится попытка записи в SSPBUF, устанавливается бит WCOL в «1», а запись не происходит.

Формирование бита STOP в режиме ведущего I2C

Чтобы инициировать формирование бита STOP, необходимо установить бит PEN (SSPCON2<2>) в «1». По окончании приема/передачи данных, после прохождения заднего фронта тактового сигнала на SCL удерживается низкий уровень сигнала. При установке бита PEN ведущий выдает низкий уровень на линию SDA, перезагружает BRG и начинает счет до нуля. По окончании счета линия SCL «отпускается». Через время TBRG, после установки высокого уровня на SCL, «отпускается» SDA. Когда на SDA появляется высокий уровень сигнала, устанавливаются биты P и SSPIF в «1», бит PEN автоматически сбрасывается в «0», а генератор BRG останавливается (см. рис. 3.17).

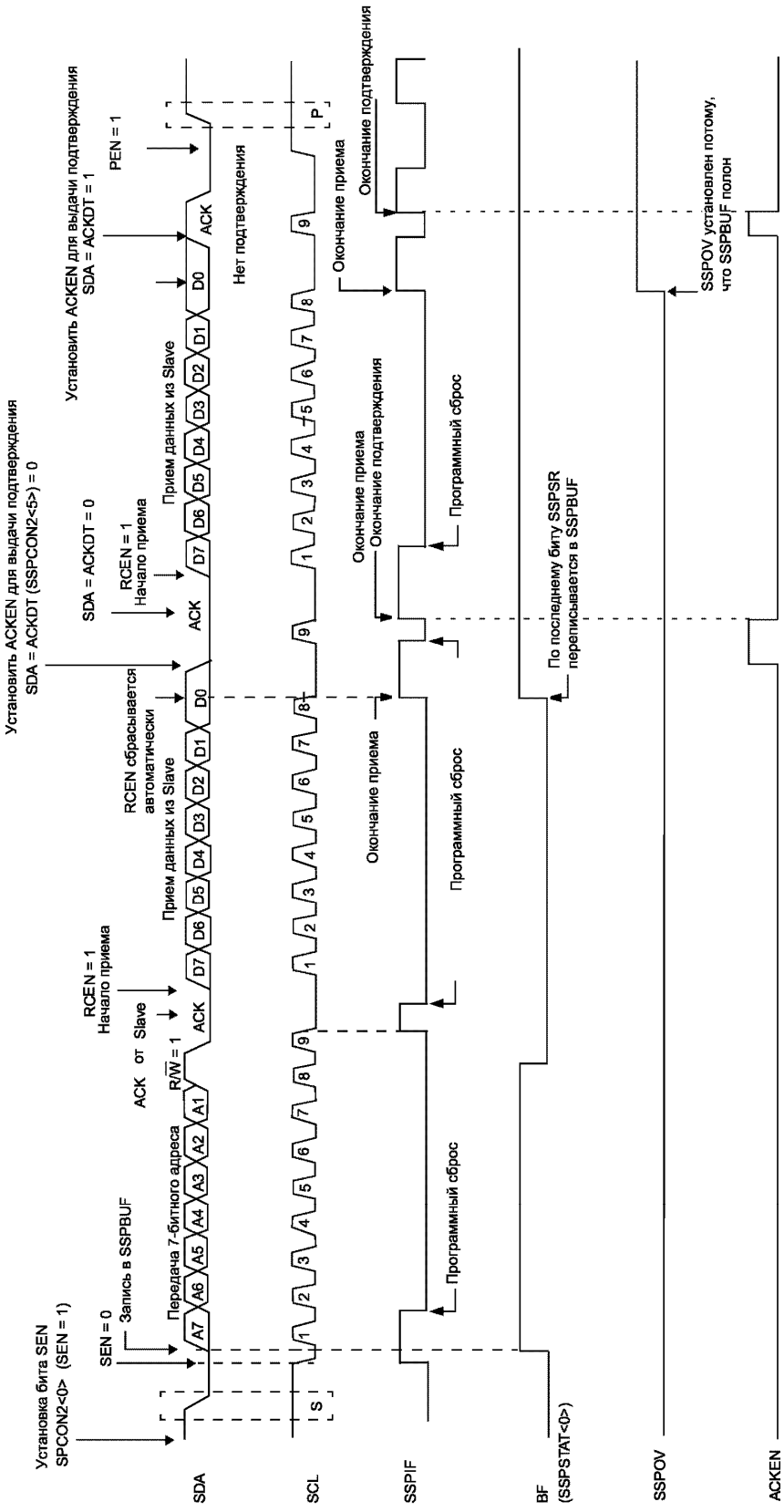


Рис. 3.16. Временная диаграмма приема данных в режиме ведущего I2C (7-разрядная адресация)

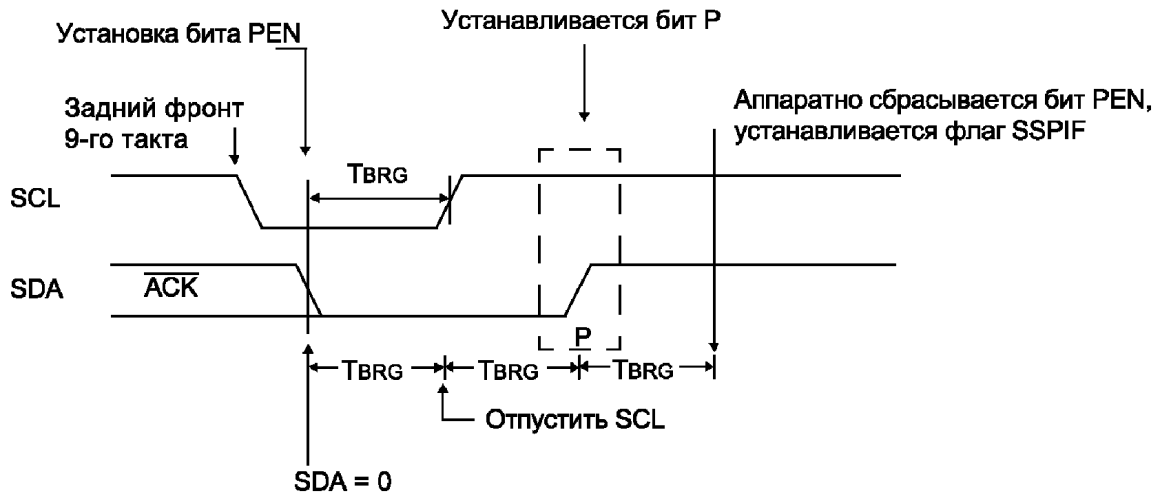


Рис. 3.17. Временная диаграмма формирования бита STOP

Если во время формирования бита STOP производится попытка записи в SSPBUF, устанавливается бит WCOL в «1», а запись не происходит.

Синхронизация тактового сигнала

Синхронизация тактового сигнала производится каждый раз во время приема/передачи данных, формирования бита START или STOP и т. д.

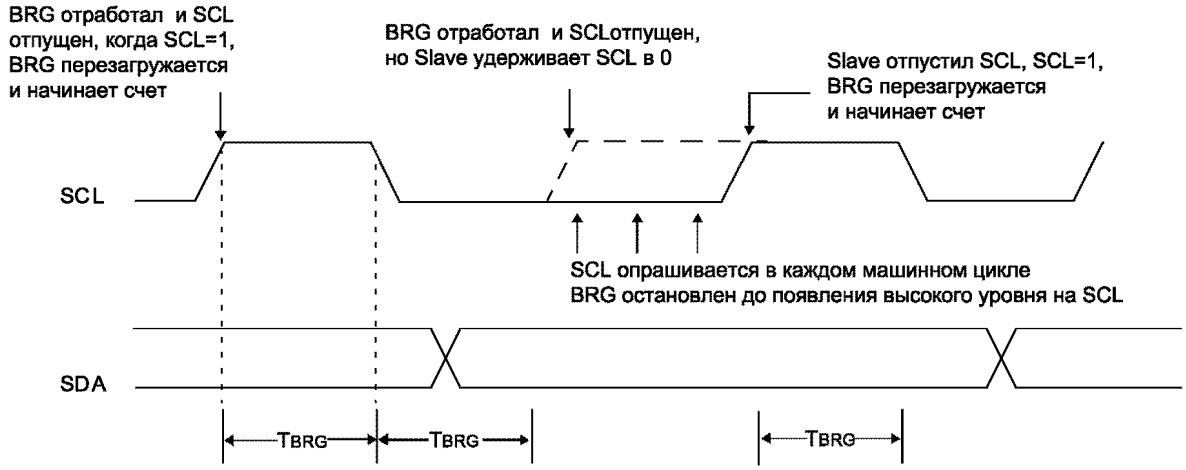


Рис. 3.18. Временная диаграмма синхронизации тактового сигнала при передаче данных ведущим

При «отпускании» ведущим SCL BRG приостанавливается, пока на SCL не появится высокий уровень сигнала. При появлении сигнала высокого уровня на SCL генератор BRG перегружается значением из SSPADD<6:0> и начинает счет. Это гарантирует, что длительность высокого уровня сигнала на SCL всегда будет не меньше TBRG, даже если другое устройство на шине удерживает тактовый сигнал.

Режим конкуренции, арбитраж и конфликты шины

В режиме конкуренции необходимо поддерживать правила арбитража шины. Во время передачи адреса/данных на SDA ведущий может потерять арбитраж, если он формирует высокий уровень сигнала, а другой ведущий сформировал низкий уровень на SDA. При переходе SCL в высокий уровень, сигнал на SDA изменяться не может. Если на SDA ожидается высокий уровень, а в действительности низкий, значит, возник конфликт шины. Обнаружив конфликт шины, ведущий устанавливает флаг прерывания BCLIF в «1», прекращает текущую операцию на шине и переводит порт I2C в режим ожидания (см. рис. 3.19).

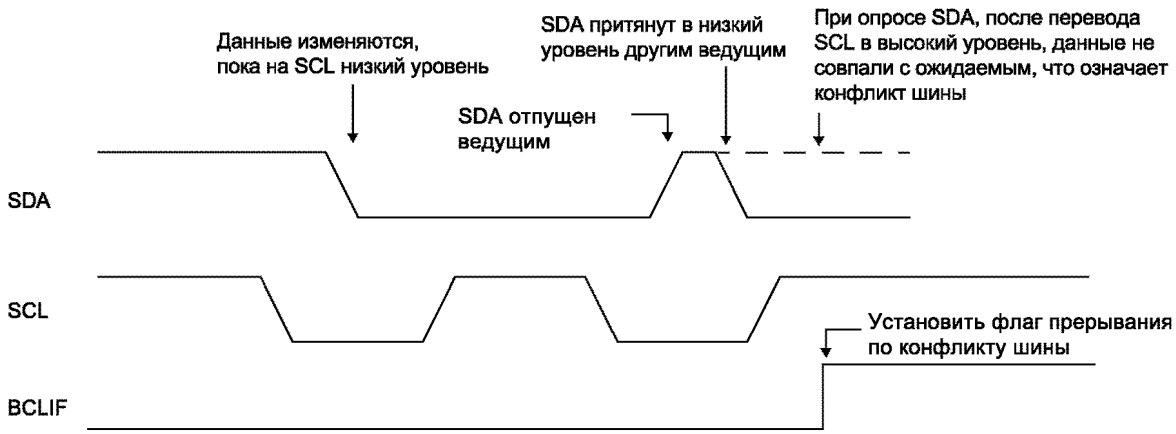


Рис. 3.19. Временная диаграмма конфликта шины при передаче данных и бита подтверждения

Если при возникновении конфликта шины выполнялась передача данных, она обрывается, устанавливается бит BF в «1», а линии SCL и SDA «отпускаются» в высокое состояние. В регистр SSPBUF может быть произведена запись, причем запись в SSPBUF инициирует передачу независимо от того, в какой момент передатчик отключился при возникновении конфликта шины. Если пользователь обрабатывает прерывания по конфликту шины, после освобождение шины он может продолжить обмен, сформировав бит START.

Если при возникновении конфликта выполнялось формирование бита START, повторный START, STOP или ACK, выполняемая операция обрывается, SCL и SDA «отпускаются», а соответствующий бит управления в SSPCON2 сбрасывается в «0». Если пользователь обрабатывает прерывания по конфликту шины, после освобождение шины он может продолжить обмен, сформировав бит START.

Ведущий продолжает следить за состоянием шины, и при появлении бита STOP устанавливается флаг прерывания SSPIF в «1».

В режиме конкуренции использование прерывания при обнаружении битов START и STOP позволяет определить занятость шины. Управление шиной может быть перехвачено при установленном бите P или сброшенных битах S и P.

Конфликт шины при формировании бита START

Во время формирования бита START конфликт шины возникает, если:

- В начале START на SDA или SCL низкий уровень сигнала (см. рис. 3.20);
- На SCL низкий уровень появляется раньше, чем на линии SDA (см. рис. 3.21).

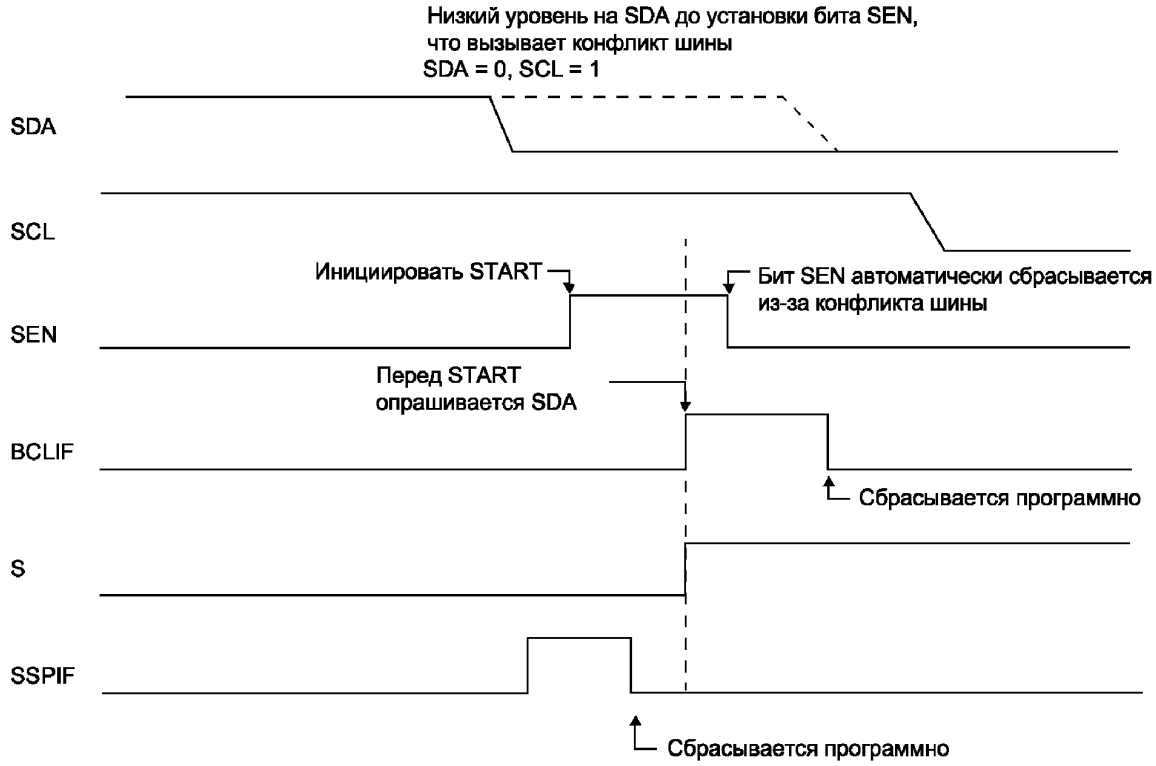


Рис. 3.20. Временная диаграмма конфликта шины во время формирования бита START (только SDA)

Во время формирования бита START сигналы SCL и SDA продолжают отслеживаться. Если SCL или SDA имеют низкий уровень сигнала, то формирование бита START прекращается, устанавливается флаг BCLIF в «1», а модуль MSSP переходит в режим ожидания (см. рис. 3.20).

Бит START начинается при наличии высокого уровня сигнала на линиях SCL и SDA. Если на SCL появляется низкий уровень раньше, чем на SDA, возникает конфликт шины, поскольку это подразумевает, что другой ведущий пытается в это время передать данные.

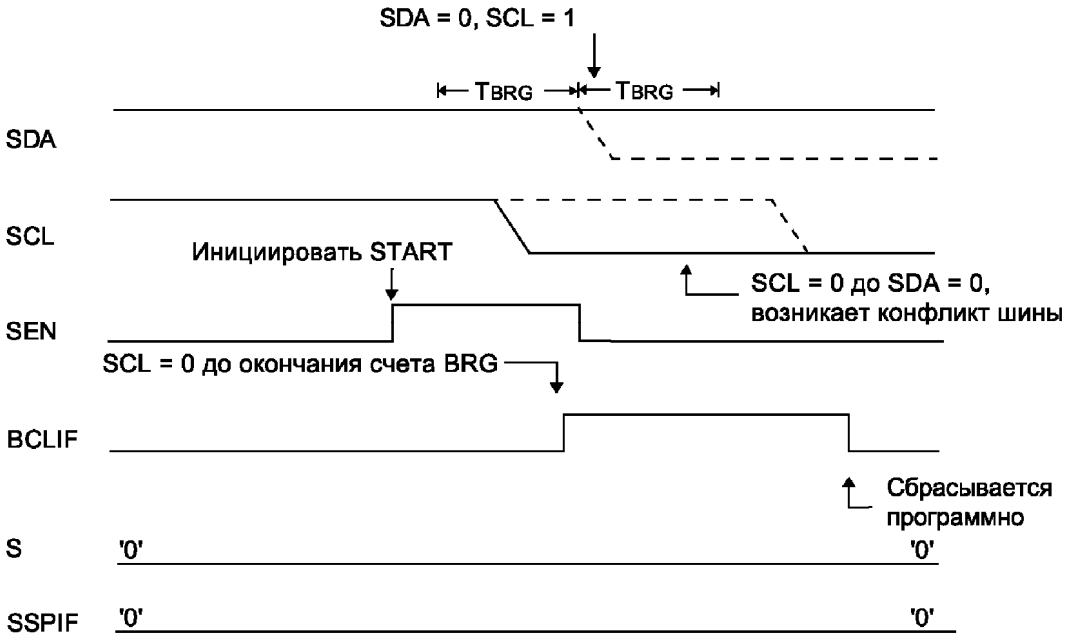


Рис. 3.21. Временная диаграмма конфликта шины во время формирования бита START (SCL = 0)

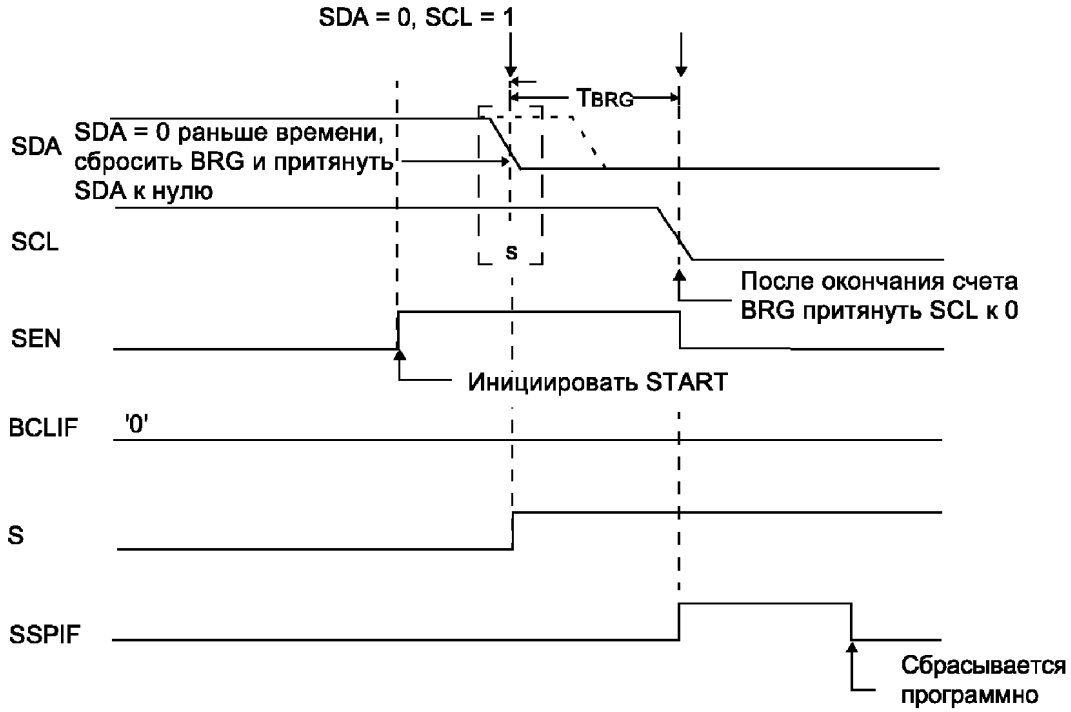


Рис. 3.22. Временная диаграмма сброса BRG при проверке линии SDA во время формирования бита START

Если во время счета BRG на SDA появляется низкий уровень сигнала, BRG сбрасывается, а на SDA формируется низкий уровень раньше времени (см. рис. 3.22).

Если же на SDA высокий уровень, низкий уровень формируется в конце счета BRG. Генератор BRG перезагружается и считает до нуля. Если в это время на SCL появится низкий уровень, конфликт шины не возникает. В конце счета BRG SCL переводится в низкий уровень.

Примечание. Конфликт шины во время START не возникает, потому что два или более ведущих могут сформировать START одновременно, но при этом один из них первым переведет SDA в низкий уровень. Конфликт шины не возникает, поскольку ведущие могут продолжить арбитраж во время передачи адреса, данных, формировании бита повторный START и STOP.

Конфликт шины при формировании бита повторный START

Во время формирования бита повторный START конфликт шины возникает, если:

- На SDA низкий уровень при переходе SCL из низкого уровня в высокий (см. рис. 3.23);
- SCL переходит в низкий уровень раньше SDA, что указывает на то, что другой ведущий пытается передать данные (см. рис. 3.24).

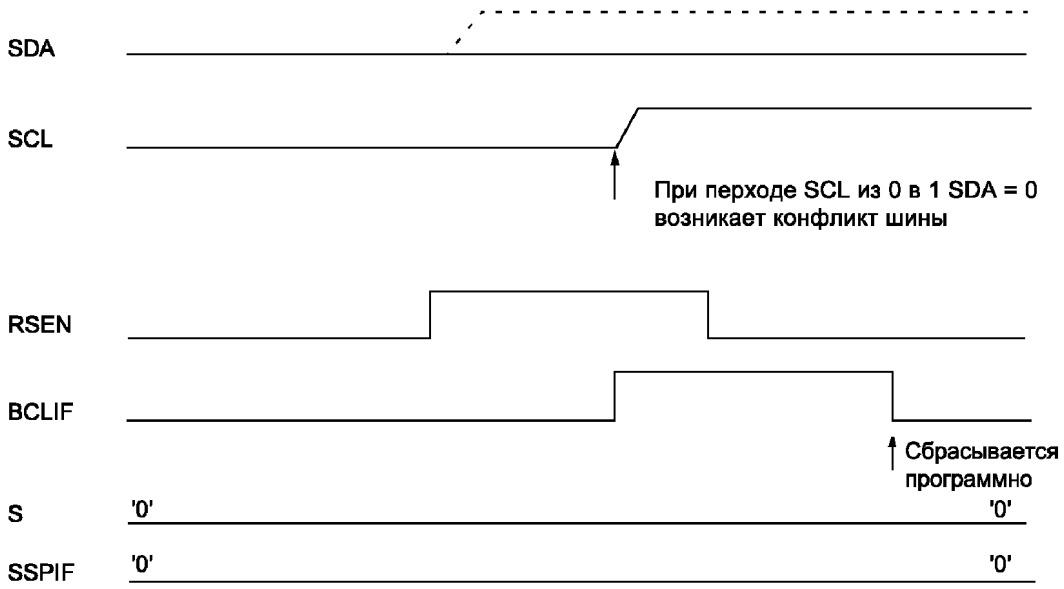


Рис. 3.23. Временная диаграмма конфликта шины во время формирования бита повторный START (случай 1)

После «отпускания» линии SDA сигнал на выводе должен перейти в высокий уровень, после чего BRG перезагружается и начинает счет. Затем «отпускается» линия SCL и при появлении на ней высокого уровня опрашивается SDA. Если на SDA низкий уровень сигнала, значит, произошел конфликт шины, т. е. другой ведущий пытается передать

данные. Если на SDA высокий уровень, то BRG снова перезагружается и начинается счет. Если SDA переходит в низкий уровень до окончания счета, конфликт шины не происходит, поскольку два или более ведущих могут пытаться получить доступ к шине одновременно.

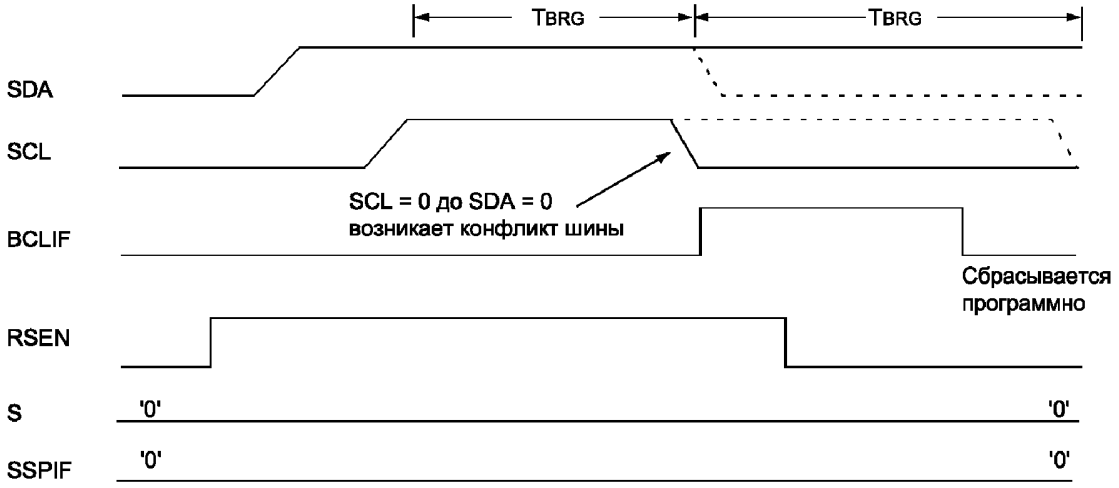


Рис. 3.24. Временная диаграмма конфликта шины во время формирования бита повторный START (случай 2)

Если на линии SCL сигнал переходит в низкий уровень до окончания счета, а на SDA сохраняется высокий уровень, значит, произошел конфликт шины, т. е. другой ведущий пытается передать данные.

Если по окончании счета BGR на SCL и SDA высокий уровень, то SDA переводится в низкий уровень, а BRG перезагружается и начинает счет. По окончании счета, независимо от уровня сигнала на SCL он переводится в низкий уровень (см. рис. 3.24).

Конфликт шины при формировании бита STOP

Во время формирования бита STOP конфликт шины возникает если:

- После «отпускания» линии SDA и окончания счета BRG на SDA по-прежнему низкий уровень сигнала (см. рис. 3.25);
- После «отпускания» линии SDA сигнал на SCL переходит в низкий уровень до того, как на SDA установится высокий уровень (см. рис. 3.26).

Формирование бита STOP начинается с перевода линии SDA в низкий уровень, затем SCL «отпускается». После появления на SCL высокого уровня BRG перезагружается и начинает счет. По окончании счета SDA «отпускается», BRG перезагружается и снова начинает счет и опрашивает SDA.

Если на нем низкий уровень или на SCL появился низкий уровень до перехода SDA в высокий, значит, произошел конфликт шины, т. е. другой ведущий пытается передать данные.

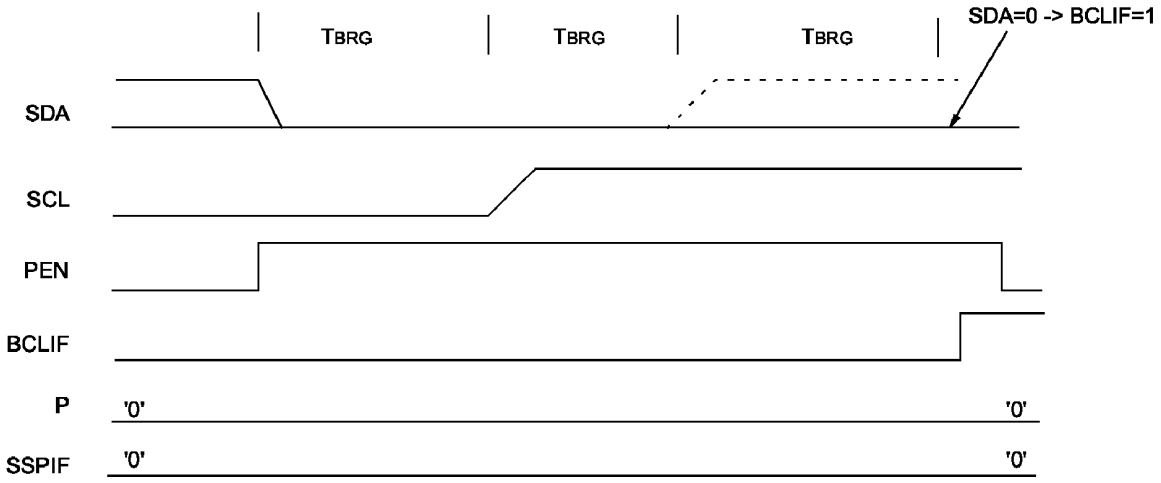


Рис. 3.25. Временная диаграмма конфликта шины во время формирования бита STOP (случай 1)

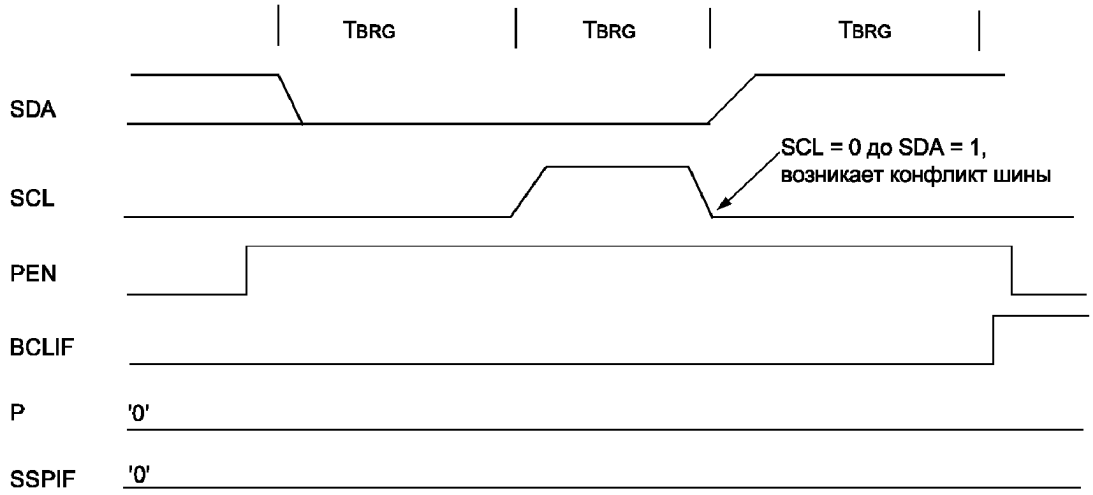


Рис. 3.26. Временная диаграмма конфликта шины во время формирования бита STOP (случай 2)

§ 3.3. Алгоритмы работы с модулем MSSP

Работа модуля MSSP в режиме I2C может быть организована как с использованием прерываний, так и без них. Ниже приведены алгоритмы работы с модулем без использования прерываний. Кроме того, в данных алгоритмах не предусмотрена обработка ошибок, что также необходимо учитывать при отладке программы.

§ 3.3.1. Алгоритм инициализации модуля MSSP

Прежде чем начать обмен данными по шине I2C необходимо проинициализировать модуль MSSP контроллера. Ниже приведен один из возможных алгоритмов инициализации.

1. Бит SSPEN (регистр SSPCON, бит 5) = 0 (выключение модуля MSSP)
2. Настройка на ввод линий порта, мультиплексированных с линиями SCL и SDA.
3. Бит SMP (регистр SSPSTAT, бит 7) = 0 (включение управления длительностью фронта в скоростном режиме 400 кГц)
4. Бит CKE (регистр SSPSTAT, бит 6) = 0 (установка уровней, соответствующих спецификации I2C)
5. Бит CKP (регистр SSPCON, бит 4) =
   Ведомый режим
   1 (не управлять тактовым сигналом)
   Ведущий режим
   не имеет значения
6. Биты SSPM3:SSPM0 (регистр SSPCON, биты 3-0) =
   Ведомый режим
   0110 (7-разрядная адресация)
   0111 (10-разрядная адресация)
   Ведущий режим
   1000
7. Бит GCEN (регистр SSPCON2, бит 7) =
   Ведомый режим
   1 (разрешить прерывания при приеме адреса общего вызова 00h)
   0 (запретить прерывания при приеме адреса общего вызова 00h)
   Ведущий режим
   не имеет значения
8. *Ведомый режим*
   Запись в регистр SSPADD адреса, который присваивается ведомому
   Ведущий режим
   Запись в регистр SSPADD числа, задающего скорость обмена. При этом скорость обмена данными по шине I2C будет определяться по формуле:

$$Скорость\,обмена = \frac{F_{OSC}}{4 \cdot (SSPADD + 1)}$$

9. Сброс бита статуса буфера BF (регистр SSPSTAT, бит 0). Сброс бита производится аппаратно при чтении регистра SSPBUF.
10. Программный сброс бита переполнения приемника SSPOV (регистр SSPCON, бит 6).
11. Программный сброс флага прерывания SSPIF (регистр PIR1, бит 3)
    Бит SSPEN (регистр SSPCON, бит 5) = 1 (включение модуля MSSP)

§ 3.3.2. Алгоритм формирования условия START

Для формирования условия START на шине I2C нужно выполнить следующие действия:

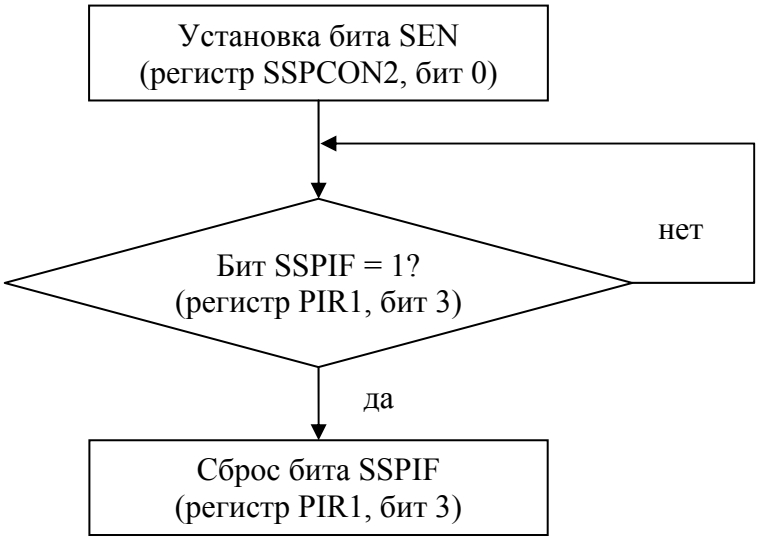


Рис. 3.27. Алгоритм формирования условия START

§ 3.3.3. Алгоритм формирования условия RESTART

Условие RESTART может быть сформировано по следующему алгоритму:

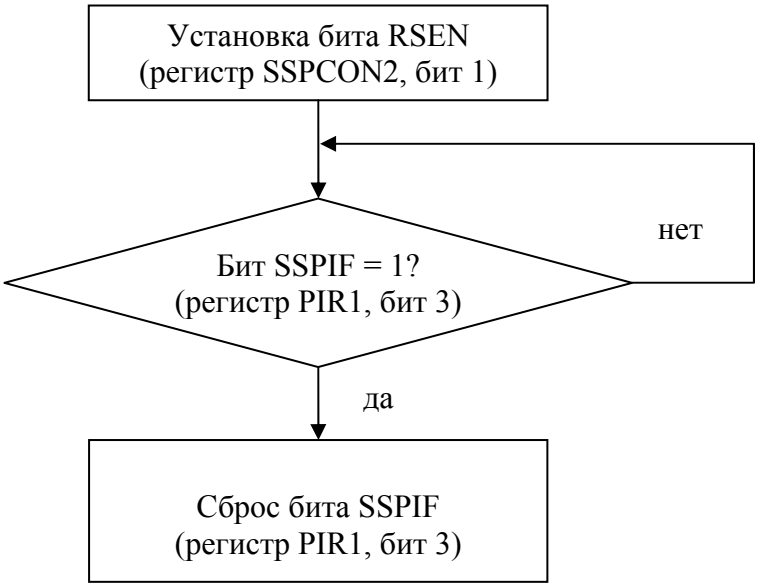


Рис. 3.28. Алгоритм формирования условия RESTART

§ 3.3.4. Алгоритм формирования условия STOP

Одним из алгоритмов формирования условия STOP является следующий:

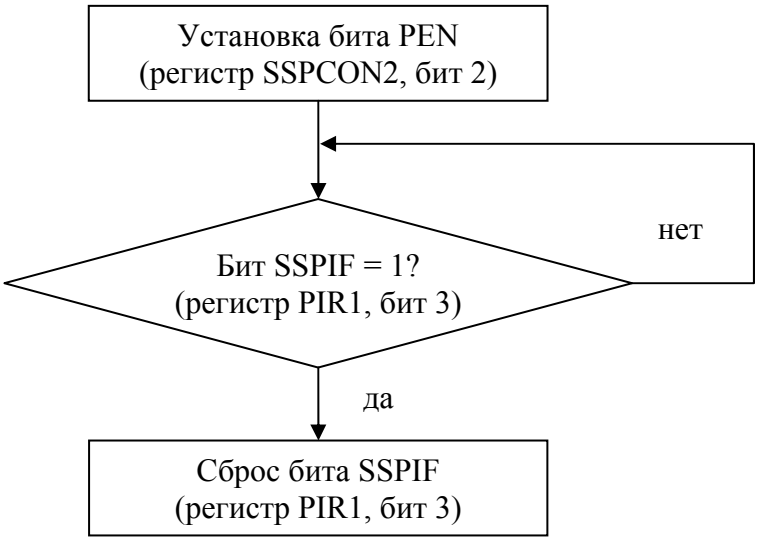


Рис. 3.29. Алгоритм формирования условия STOP

§ 3.3.5. Алгоритм приема байта ведомым

Для организации приема байта ведомым устройством может быть применен следующий алгоритм:

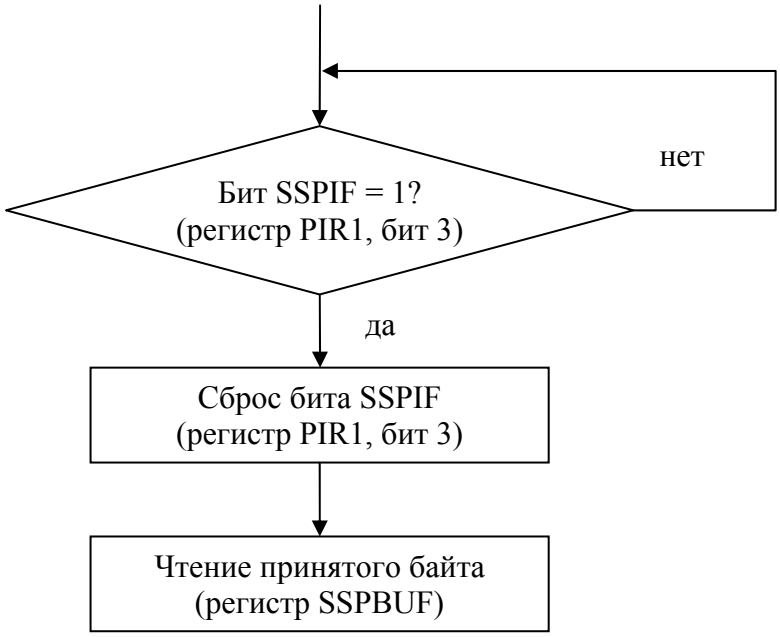


Рис. 3.30. Алгоритм приема байта ведомым

§ 3.3.6. Алгоритм передачи байта ведомым

Передача байта ведомым устройством может быть организована следующим образом:

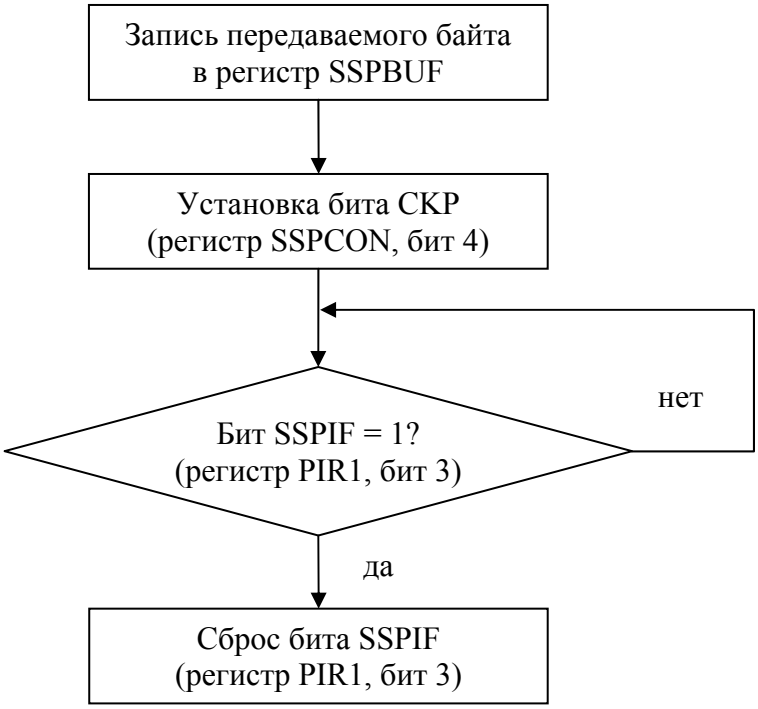


Рис. 3.31. Алгоритм передачи байта ведомым

§ 3.3.7. Алгоритм передачи байта ведущим

Ведущее устройство может передать байт, используя следующий алгоритм:

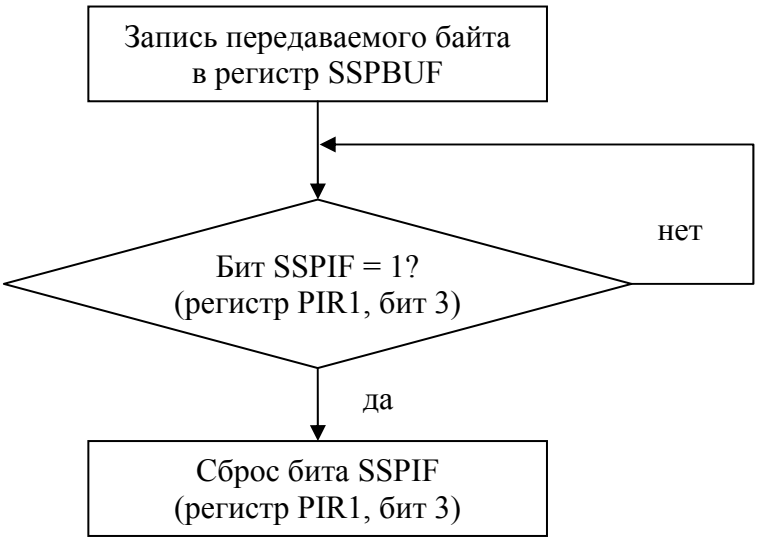


Рис. 3.32. Алгоритм передачи байта ведущим

§ 3.3.8. Алгоритм приема байта ведущим

Прием байта ведущим устройством с шины I2C можно организовать по следующему алгоритму:

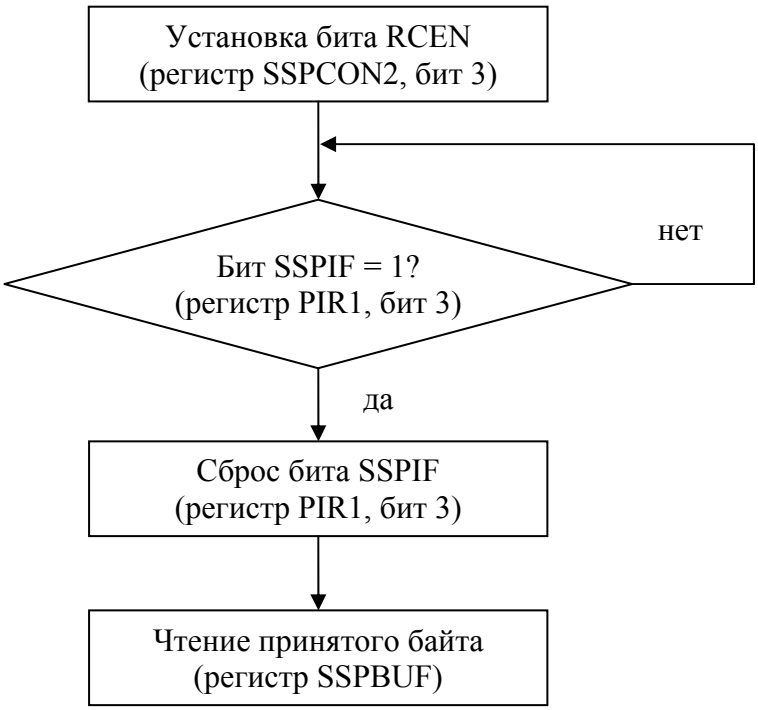


Рис. 3.33. Алгоритм приема байта ведущим

§ 3.3.9. Алгоритм формирования бита подтверждения ACK ведущим

Для формирования бита подтверждения нужно выполнить следующую последовательность действий:

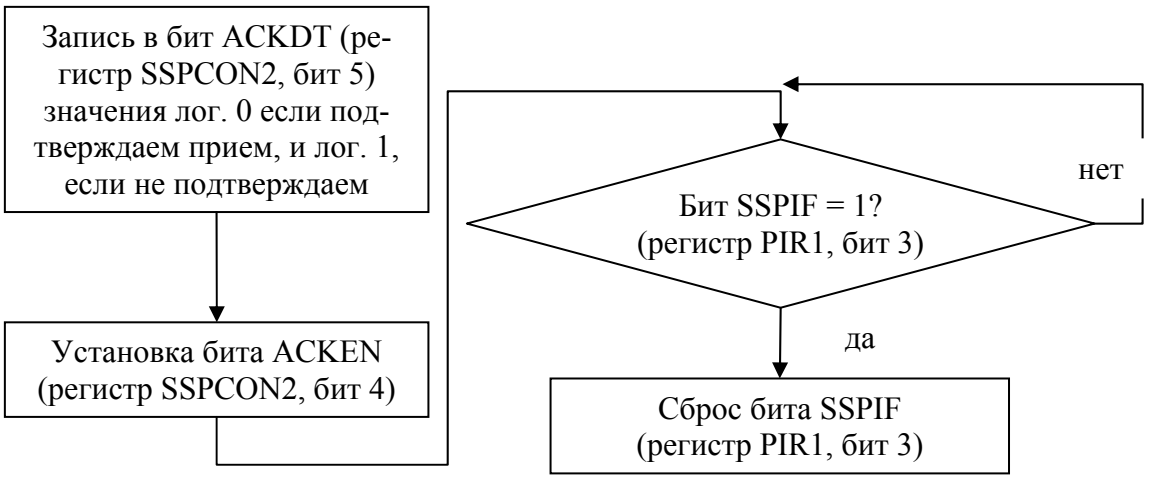


Рис. 3.34. Алгоритм формирования бита подтверждения ACK/NACK ведущим

§ 3.4. Примеры программ

§ 3.4.1. Пример программы ведущего микроконтроллера

В качестве примера рассмотрим программу, предназначенную для работы с цифровым датчиком температуры DS1621. Программа осуществляет запуск датчика на преобразование температуры в цифровой код, дожидается окончания преобразования, считывает результат и сохраняет его в регистрах микроконтроллера. При этом предполагается, что датчик уже настроен на работу в режиме однократного преобразования и находится в режиме ожидания. Алгоритм данной программы представлен на рис. 3.35, 3.36 и 3.37.

Алгоритмы подпрограмм, формирующих условия START, RESTART, STOP, а так же осуществляющих прием, передачу байта и формирование бита подтверждения, приведены в 0.

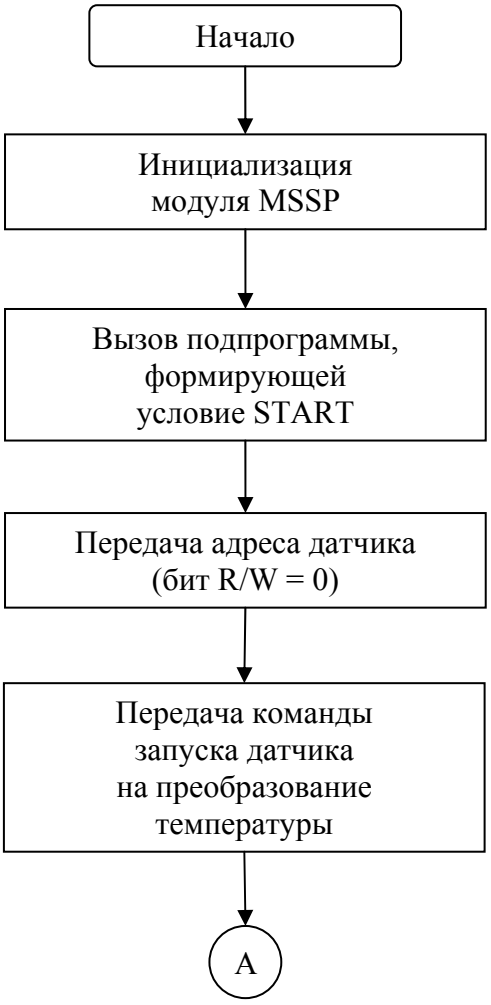


Рис. 3.35. Алгоритм программы ведущего микроконтроллера (начало)

Рис. 3.36. Алгоритм программы ведущего микроконтроллера (продолжение)

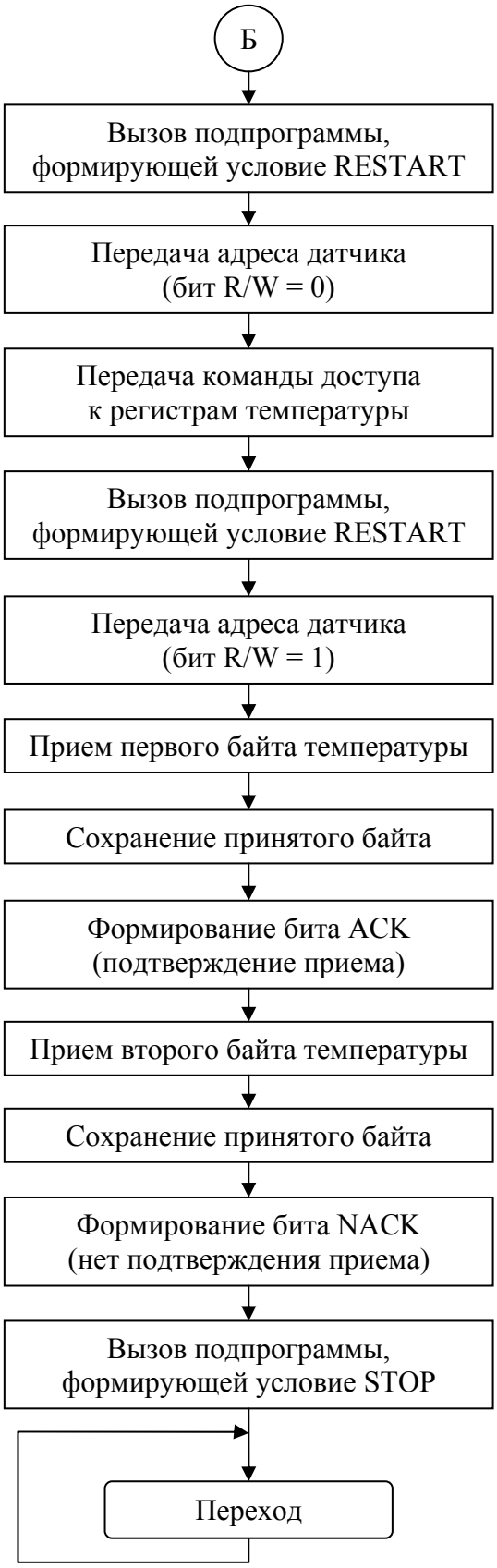


Рис. 3.37. Алгоритм программы ведущего микроконтроллера (окончание)

Ниже приведена программа, реализующая алгоритм, изображенный на рис. 3.35, 3.36 и 3.37. Программа написана для микроконтроллера PIC166F877 с кварцевым резонатором 4 МГц, обмен данными по шине I2C ведется на частоте 100 кГц.

```
list p=16f877
include «p16f877.inc»

ADR_W equ b'10010000'    ;байт адреса датчика (для записи)
ADR_R equ b'10010001'    ;байт адреса датчика (для чтения)
COM_ST equ 0xEE          ;команда запуска датчика на преобразование
COM_CON equ 0xAC         ;команда доступа к регистру конфигурации датчика
COM_TP equ 0xAA          ;команда доступа к регистрам температуры датчика
TEMP_1 equ 0x7A          ;регистр для хранения первого байта данных
TEMP_2 equ 0x7B          ;регистр для хранения второго байта данных

;-------------------------------------------------------------------------------
;инициализация
;-------------------------------------------------------------------------------

;инициализация модуля MSSP
        banksel SSPCON      ;выбор банка с регистром SSPCON
        bcf SSPCON, SSPEN   ;выключение модуля MSSP

        banksel TRISC       ;выбор банка с регистром TRISC
        bsf TRISC, 3        ;настройка на ввод линий порта C
        bsf TRISC, 4        ;мультиплексированных с модулем MSSP

        bcf SSPSTAT, SMP    ;включение управления длительностью фронта в
                            ;скоростном режиме
        bcf SSPSTAT, CKE    ; установка уровней, соответствующих; спецификации I2C

        banksel SSPCON      ;выбор банка с регистром SSPCON
        bsf SSPCON, SSPM3   ;задание режима
        bcf SSPCON, SSPM2   ;работы модуля MSSP
        bcf SSPCON, SSPM1   ;в режиме
        bcf SSPCON, SSPM0   ;ведущего

        banksel SSPADD      ;выбор банка с регистром SSPADD
        movlw 0x09          ;загрузка в SSPADD делителя,
        movwf SSPADD        ;обеспечивающего частоту обмена 100 кГц

        banksel SSPBUF      ;выбор банка с регистром SSPBUF
        movf SSPBUF, W      ;сброс бита статуса буфера BF
        bcf SSPCON, SSPOV   ;сброс бита переполнения приемника SSPOV

        bcf PIR1, SSPIF     ;сброс флага прерывания от модуля MSSP
```

```
        bsf SSPCON, SSPEN  ;включение модуля MSSP


;-------------------------------------------------------------------------------
;основная программа
;-------------------------------------------------------------------------------

;запуск датчика на преобразование
        call START         ;вызов подпрограммы, формирующей условие START

        movlw ADR_W        ;передача байта адреса датчика (бит R/W = 0)
        call SEND_B        ;на шину I2C

        movlw COM_ST       ;передача команды запуска
        call SEND_B        ;датчика на преобразование температуры

rest

;доступ к регистру конфигурации
        call RESTART       ;вызов подпрограммы, формирующей условиеRESTART

        movlw ADR_W        ;передача байта адреса датчика (бит R/W = 0)
        call SEND_B        ;на шину I2C

        movlw COM_CON      ;передача команды доступа
        call SEND_B        ;к регистру конфигурации датчика

;чтение регистра конфигурации
        call RESTART       ;вызов подпрограммы, формирующей условие RESTART

        movlw ADR_R        ;передача байта адреса датчика (бит R/W = 1)
        call SEND_B        ;на шину I2C

        call REC_B         ;прием содержимого регистра конфигурации датчика

        call NACK          ;формирование бита NACK (не подтверждаем прием)

;завершено ли преобразование?
        andlw 0x80         ;если преобразование не завершено (т. е. если бит 7 регистра
        btfsc STATUS, Z    ;конфигурации не установлен), то
        goto rest          ;переход на метку rest

;доступ к регистрам темературы
        call RESTART       ;вызов подпрограммы, формирующей условие RESTART
        movlw ADR_W        ;передача байта адреса датчика (бит R/W = 0)
        call SEND_B        ;на шину I2C

        movlw COM_TP       ;передача команды доступа
```

```
        call SEND_B         ;к регистрам температуры датчика

;чтение регистров температуры
        call RESTART        ;вызов подпрограммы, формирующей условие RESTART

        movlw ADR_R         ;передача байта адреса датчика (бит R/W = 1)
        call SEND_B         ;на шину I2C

        call REC_B          ;прием первого байта температуры

        movwf TEMP_1        ;сохранение принятого байта

        call ACK            ;вызов подпрограммы, формирующей бит подтвержде-
                            ;ния ACK

        call REC_B          ;прием второго байта температуры

        movwf TEMP_2        ;сохранение принятого байта

        call NACK           ;вызов подпрограммы, формирующей бит NACK

        call STOP           ;вызов подпрограммы, формирующей условие STOP

m_loop
        goto m_loop         ;переход на метку m_loop

;=========================================
;подпрограммы работы с модулем MSSP
;=========================================

;подпрограмма формирования условия START
START
        banksel SSPCON2     ;выбор банка с регистром SSPCON2
        bsf SSPCON2, SEN    ;запуск формирования условия START
        banksel PIR1        ;выбор банка с регистром PIR1
m_st
        btfss PIR1, SSPIF   ;проверка, закончено ли формирование
        goto m_st           ;если нет, то переход на метку m_st
        bcf PIR1, SSPIF     ;сброс флага прерывания SSPIF
return ;возврат из подпрограммы

;подпрограмма формирования условия RESTART
RESTART
        banksel SSPCON2     ;выбор банка с регистром SSPCON2
        bsf SSPCON2, RSEN   ;запуск формирования условияRESTART

        banksel PIR1        ;выбор банка с регистром PIR1
m_rst
```

```
        btfss PIR1, SSPIF       ;проверка, закончено ли формирование
        goto m_rst              ;если нет, то переход на метку m_rst

        bcf PIR1, SSPIF         ;сброс флага прерывания SSPIF
return  ;возврат из подпрограммы

;подпрограмма формирования условия STOP
STOP
        banksel SSPCON2         ;выбор банка с регистром SSPCON2
        bsf SSPCON2, PEN        ;запуск формирования условия STOP

        banksel PIR1            ;выбор банка с регистром PIR1
m_sp
        btfss PIR1, SSPIF       ;проверка, закончено ли формирование
        goto m_sp               ;если нет, то переход на метку m_sp

        bcf PIR1, SSPIF         ;сброс флага прерывания SSPIF
return  ;возврат из подпрограммы

;подпрограмма передачи байта
SEND_B
        banksel SSPBUF          ;выбор банка с регистром SSPBUF
        movwf SSPBUF            ;загрузка отправляемого байта в регистр SSPBUF

        banksel PIR1            ;выбор банка с регистром PIR1
m_sb
        btfss PIR1, SSPIF       ;проверка, закончена ли передача байта
        goto m_sb               ;если нет, то переход на метку m_sb

        bcf PIR1, SSPIF         ;сброс флага прерывания SSPIF
return  ;возврат из подпрограммы

;подпрограмма приема байта
REC_B
        banksel SSPCON2         ;выбор банка с регистром SSPCON2
        bsf SSPCON2, RCEN       ;запуск процесса приема байта

        banksel PIR1            ;выбор банка с регистром PIR1
m_rb
        btfss PIR1, SSPIF       ;проверка, закончен ли прием байта
        goto m_rb               ;если нет, то переход на метку m_rb

        bcf PIR1, SSPIF         ;сброс флага прерывания SSPIF
        movf SSPBUF, W          ;загрузка принятого байта в аккумулятор
                                ;и сброс бита статуса буфера BF (аппаратно)
return  ;возврат из подпрограммы
```

```
;подпрограмма формирования бита подтверждения
ACK
        banksel SSPCON2     ;выбор банка с регистром SSPCON2
        bcf SSPCON2, ACKDT  ;загрузка значения (лог. 0), выдаваемого на линию
                            SDA во время
                            ;формирования бита подтверждения
        bsf SSPCON2, ACKEN  ;запуск процесса формирования бита подтвер-
                            ждения

        banksel PIR1        ;выбор банка с регистром PIR1
m_ack
        btfss PIR1, SSPIF   ;проверка, закончено ли формирование
        goto m_ack          ;если нет, то переход на метку m_ack

        bcf PIR1, SSPIF     ;сброс флага прерывания SSPIF
return  ;возврат из подпрограммы

;подпрограмма формирования бита «неподтверждения»
NACK
        banksel SSPCON2     ;выбор банка с регистром SSPCON2
        bsf SSPCON2, ACKDT  ;загрузка значения (лог. 1), выдаваемого на линию
                            SDA во время
                            ;формирования бита подтверждения
        bsf SSPCON2, ACKEN  ;запуск процесса формирования бита подтвер-
                            ждения

        banksel PIR1        ;выбор банка с регистром PIR1
m_nack
        btfss PIR1, SSPIF   ;проверка, закончено ли формирование
        goto m_nack         ;если нет, то переход на метку m_nack

        bcf PIR1, SSPIF     ;сброс флага прерывания SSPIF
return  ;возврат из подпрограммы

        end
```

§ 3.4.2. Пример программы ведомого микроконтроллера

В качестве примера рассмотрим программу, которая будет осуществлять обмен информацией с ведущим устройством так, как это делал бы датчик температуры DS1621. Предполагаем, что ведущее устройство работает по алгоритму, описанному в § 3.4.1, тогда ведомое устройство должно реализовывать алгоритм, изображенный на рис. 3.38.

Алгоритмы подпрограмм, осуществляющих прием и передачу байта, приведены в 0.

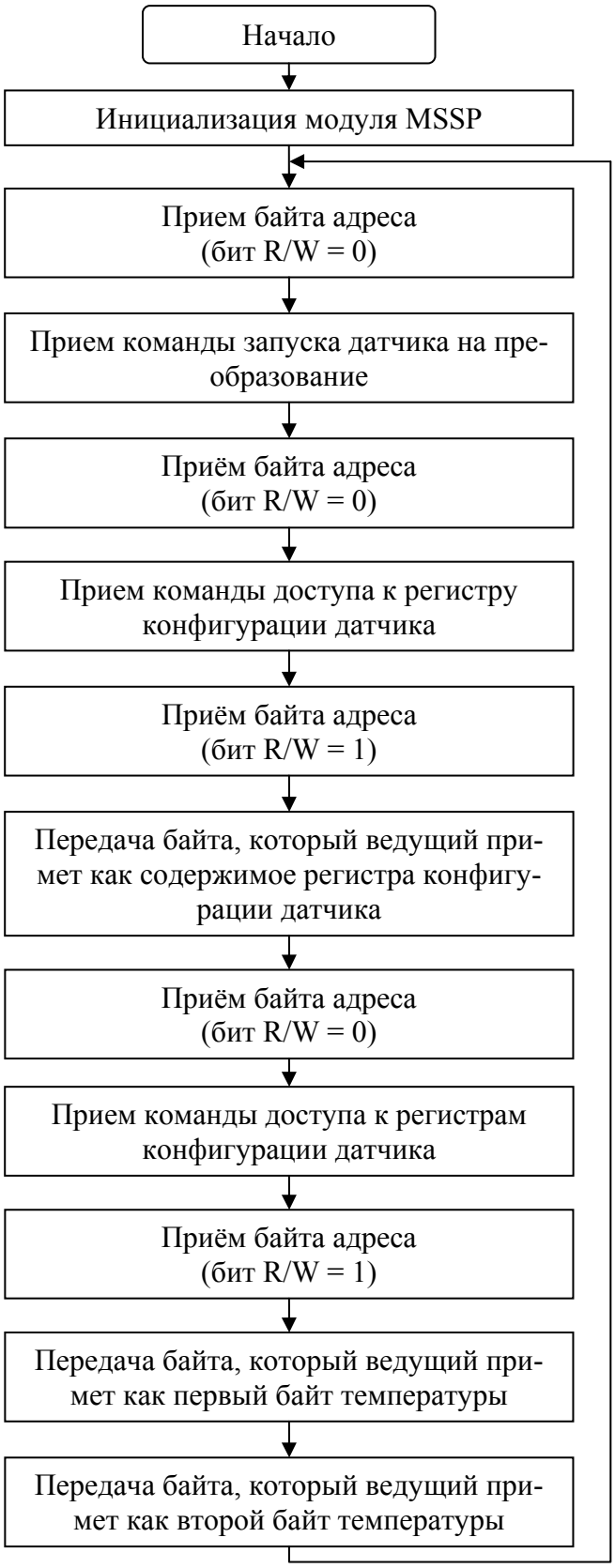


Рис. 3.38. Алгоритм программы ведомого контроллера

Ниже приведена программа, реализующая алгоритм, изображенный на рис. 3.38. Программа написана для микроконтроллера PIC166F877, обмен данными по шине I2C ведется на частоте 100 кГц.

```
list p=16f877
include «p16f877.inc»

ADR equ b'10010000'      ;адрес ведомого устройства
REG_CON equ 0x81         ;значение, которое передается ведущему как содер-
                         ;жимое
                         ;регистра конфигурации датчика температуры
T_1 equ 0xED             ;значение, которое передается как первый байт
                         температуры
T_2 equ 0x80             ;значение, которое передается как второй байт
                         температуры

;-------------------------------------------------------------------------------
;инициализация
;-------------------------------------------------------------------------------

;инициализация модуля MSSP
        banksel SSPCON       ;выбор банка с регистром SSPCON
        bcf SSPCON, SSPEN    ;выключение модуля MSSP

        banksel TRISC        ;выбор банка с регистром TRISC
        bsf TRISC, 3         ;настройка на ввод линий порта C
        bsf TRISC, 4         ;мультиплексированных с модулем MSSP

        bcf SSPSTAT, SMP     ;включение управления длительностью фронта
                             ;в скоростном режиме
        bcf SSPSTAT, CKE     ;установка уровней, соответствующих
                             ;спецификации I2C
        banksel SSPCON       ;выбор банка с регистром SSPCON
        bsf SSPCON, CKP      ;не управлять тактовым сигналом

        bcf SSPCON, SSPM3    ;задание режима
        bsf SSPCON, SSPM2    ;работы модуля MSSP
        bsf SSPCON, SSPM1    ;в режиме ведомого
        bcf SSPCON, SSPM0    ;с 7-разрядной адресацией

        banksel SSPCON2      ;выбор банка с регистром SSPCON2
        bcf SSPCON2, GCEN    ;запретить прерывания при приеме в регистр SSPSR
                             ;адреса общего вызова 0000h
        movlw ADR            ;запись в регистр SSPADD адреса,
        movwf SSPADD         ;который присваивается ведомому
        banksel SSPBUF       ;выбор банка с регистром SSPBUF
        movf SSPBUF, W       ;сброс бита статуса буфера BF
```

```
        bcf SSPCON, SSPOV   ;сброс бита переполнения приемника SSPOV

        bcf PIR1, SSPIF     ;сброс флага прерывания от модуля MSSP

        bsf SSPCON, SSPEN   ;включение модуля MSSP

;-------------------------------------------------------------------------------
;основная программа
;-------------------------------------------------------------------------------

sta
        call REC_B          ;прием байта адреса (бит R/W = 0)
        call REC_B          ;прием команды запуска датчика на преобразование
                            температуры
        call REC_B          ;прием байта адреса (бит R/W = 0)

        call REC_B          ;прием команды доступа к регистру конфигурации
                            датчика

        call REC_B          ;прием байта адреса (бит R/W = 1)

        movlw REG_CON       ;загрузка в аккумулятор байта, который ведущий примет
                            ; как содержимое регистра конфигурации датчика
        call SEND_B         ;передача «содержимого регистра конфигурации»

        call REC_B          ;прием байта адреса (бит R/W = 0)

        call REC_B          ;прием команды доступа к регистрам температуры

        call REC_B          ;прием байта адреса (бит R/W = 1)

        movlw T_1

        call SEND_B         ;передача первого байта температуры

        movlw T_2

        call SEND_B         ;передача второго байта температуры

        goto sta

;===============================================
;подпрограммы работы с модулем MSSP
;===============================================
;подпрограмма приема байта
REC_B
        banksel PIR1        ;выбираем банк с регистром PIR1
```

```
m_rb
        btfss PIR1, SSPIF       ;проверка, закончен ли прием байта
        goto m_rb               ;если нет, то переход на метку m_rb

        bcf PIR1, SSPIF         ;сброс флага прерывания от модуля MSSP
        movf SSPBUF, W          ;загрузка принятого байта в аккумулятор
                                ;и сброс бита статуса буфера BF (аппаратно)
        return                  ;возврат из подпрограммы

;подпрограмма передачи байта
SEND_B
        banksel SSPBUF          ;выбор банка с регистром SSPBUF
        movwf SSPBUF            ;загрузка отправляемого байта в регистр SSPBUF
        bsf SSPCON, CKP         ;запуск процесса отправки байта

m_sb
        btfss PIR1, SSPIF       ;проверка, закончена ли передача байта
        goto m_sb               ;если нет, то переход на метку m_sb

        bcf PIR1, SSPIF         ;сброс флага прерывания от модуля MSSP
        return                  ;возврат из подпрограммы

        end
```

ГЛАВА 4. ШИНА I2C В МИКРОКОНТРОЛЛЕРАХ СЕМЕЙСТВА MCS-51

§ 4.1. Описание микроконтроллера и среды разработки

§ 4.1.1. Общие сведения о C8051F060

Микроконтроллеры фирмы Silicon Laboratories C8051Fxxx оптимально подходят для построения устройств, требующих высокой производительности, точности измерений, большой степени интеграции и малого потребления. Они программно совместимы с 8051-м стандартом, но одновременно имеют рекордно высокую производительность – до 100 MIPS. Микроконтроллеры Silabs также уникальны своей интеграцией с аналоговыми узлами. Многие модели имеют до двух встроенных независимых АЦП, встроенный ЦАП, компараторы напряжения, программируемые усилители напряжения и др.

Основные характеристики:

- Два 16-разрядных АЦП последовательного приближения
  - Нелинейность +-0,75 МЗР
  - Программируемая скорость преобразования (до 1 млн преобразований в секунду)
  - Прямой доступ к памяти, данные сохраняются в ОЗУ без вмешательства со стороны программы
  - Формирование прерывания при попадании результата преобразования в заданный диапазон значений
- 10-разрядный АЦП последовательного приближения
  - Программируемая скорость преобразования (до 200 тыс. преобразований в секунду)
  - До 8-ми внешних входов
  - Встроенный датчик температуры
- Два 12-разрядных ЦАП
  - Синхронизация выходов с таймерами для генерации сигнала без фазовых искажений
- Три аланоговых компаратора
  - Программируемые гистерезис/время отклика
- Источник опорного напряжения
  - Прецизионная схема слежения за напряжением питания/детектор снижения напряжения питания

- Встроенный JTAG отладчик и интерфейс граничного сканирования
  - Высокопроизводительное 8051-совместимое процессорное ядро
  - Конвейерная арихтектура, 70 % команд выполняются за 1 или 2 системных тактовых цикла
  - Производительность до 25MIPS при тактовой частоте 25MHz
  - Гибкая система прерываний
- Память
  - 4352 (4К + 256) байт внутреннего ОЗУ данных
  - 64 Кбайт FLASH-памяти, возможно внутрисистемное программирование FLASH-памяти секторами по 512 байт
  - Интерфейс внешней (64 Кбайт) памяти данных с возможностью мультиплексированного и немультиплексированного режимов работы
- Цифровые периферийные модули
  - 59 портов ввода/вывода общего назначения
  - Встроенный контроллер CAN 2.0B
  - Последовательные интерфейсы SMBus (I2C-совместимый), SPI и два УАПП (доступны одновременно)
  - Программируемый массив 16-разрядных таймеров/счетчиков с шестью модулями захвата/сравнения
  - Пять 16-разрядных таймеров/счетчиков общего назначения
  - Отдельный сторожевой таймер
  - Двунаправленный вывод сброса
- Источники тактовых импульсов
  - Внутренний калибруемый прецизионный генератор 24.5 МГц
  - Внешний генератор: кварцевый, RC-, C-, или счетчик
- Напряжение питания: 2.7…3.6 В
- Рабочая температура: –40…+85 ОС

Память программ

CIP-51 имеет адресное пространство памяти программ 64 Кбайт. В МК C8051F060/1/2/3/4/5 физически реализовано 64 Кбайт этой памяти программ, которая является внутрисистемной перепрограммируемой Flash-памятью, занимающей непрерывный блок адресов от 0x0000 до 0xFFFF. Следует иметь ввиду, что 1024 байт (0xFC00 – 0xFFFF) этой памяти зарезервированы для целей производителя и не доступны для хранения программ пользователя.

По умолчанию память программ настраивается только для чтения. Однако CIP-51 может записывать данные в память программ (с использованием команды MOVX), для чего необходимо разрешить запись во Flash-память

программ. Эта возможность позволяет МК обновлять программный код и использовать память программ для долговременного хранения данных.

Память данных

Физически реализовано 256 байт внутреннего ОЗУ, отображенного в пространстве памяти данных с адресами от 0x00 до 0xFF. Младшие 128 байт памяти данных используются для регистров общего назначения (РОН) и сверхоперативного ЗУ (СОЗУ). Для доступа к младшим 128 байтам памяти данных можно использовать либо прямую, либо косвенную адресацию. Ячейки с адресами от 0x00 до 0x1F разбиты на четыре банка РОН (Регистры общего Назначения), каждый банк состоит из восьми однобайтовых регистров. Следующие 16 байт (0x20 – 0x2F) могут адресоваться побайтно или побитно как 128 бит, доступные в режиме прямой битовой адресации.

Старшие 128 байт памяти данных доступны только в режиме косвенной адресации. Эта область памяти занимает то же самое адресное пространство, что и регистры специального назначения (Special Function Registers – SFR), но физически отделена от них. При обращении к ячейкам памяти с адресами 0x7F – 0xFF использующийся в команде режим адресации определяет, к чему осуществляется доступ: к страшим 128 байтам памяти данных или к SFR. Команды, которые используют режим прямой адресации, будут обращаться к SFR. Команды, использующие режим косвенной адресации, будут обращаться к старшим 128 байтам памяти данных.

Страничная организация SFR

В CIP-51 используется страничная организация SFR, что позволяет отображать в адресном пространстве 0x80 – 0xFF большое количество регистров SFR. Пространство памяти SFR имеет 256 страниц. Таким образом, каждая ячейка памяти из области 0x80 – 0xFF может адресовать до 256 регистров SFR. В микроконтроллерах семейства C8051F06x используются пять SFR страниц: 0, 1, 2, 3 и F. SFR страницы выбираются при помощи регистра выбора страницы SFRPAGE. Последовательность действий при чтении и записи SFR следующая:

1. Выбрать номер соответствующей SFR страницы, используя регистр SFRPAGE.
2. Прочитать или записать регистр SFR, используя режим прямой адресации (команда MOV).

Стек

Программный стек может быть размещен в любом месте 256-байтной памяти данных. Область стека определяется с использованием указателя стека (Stack Pointer – SP, 0x81). SP будет указывать на последнюю использо-

ванную ячейку. Следующее значение, загружаемое в стек, размещается по адресу SP+1, и затем SP инкрементируется. При сбросе SP инициализируется значением 0x07. Поэтому первое значение, загружаемое в стек, размещается по адресу 0x08, которое также является первым регистром (R0) регистрового банка 1. Таким образом, если требуется использовать более одного банка регистров, SP следует инициализировать адресом ячейки ОЗУ, не используемой для хранения данных. Стек может иметь глубину до 256 байт.

МК также имеет встроенный аппаратный регистратор стековых операций, который представляет собой 32-разрядный сдвиговый регистр. Каждая команда PUSH или инкремент SP загружают один регистрационный бит в этот регистр, каждая команда CALL или прерывание загружают два регистрационных бита в этот регистр. Команда POP или декремент SP извлекают один регистрационный бит, а команда RET извлекает два регистрационных бита из этого регистра. Схема регистратора стековых операций способна определять переполнение или опустошение стека и может уведомлять программные средства отладки, даже если МК отлаживается в режиме реального времени.

§ 4.1.2. Среда SILICON LABORATORIES

Существует множество инструментальных средств, предназначенных для облегчения и интенсификации труда разработчиков. Основное назначение любого отладчика – помочь разработчику разобраться на программно – аппаратном уровне в процессах происходящих в устройстве, а затем добиться желаемых характеристик функционирования. Кроме этого инструментальные средства могут обладать дополнительными возможностями, которые избавляют разработчика от множества утомительных процедур.

Общее описание внутрисхемного отладчика и Silicon Laboratories IDE

Silicon Laboratories IDE – интегрированная среда разработки, представляет собой программный продукт, работающий под управлением операционной системы Windows и предназначенный для написания, отладки и оптимизации программ для контроллеров Cygnal. Silicon Labs IDE включает в себя:

Project Manager – менеджер проекта, позволяет создавать, сохранять текущие проекты;

Editor – текстовый редактор для написания программы;

Simulator – симулятор для проверки функционирования программы;

Keil Assembler – язык программирования

Linker – утилита объединения всех файлов проекта

Compilar – утилита преобразования программы в машинные коды

Внутрисхемный отладчик ICD

ICD работает под управлением программной среды Silicon Laboratories и обладает следующими возможностями:

- Отладка в режиме реального времени с возможностью пошаговой отладки;
- Связь с компьютером по USB – интерфейсу;
- Просмотр и модификация содержимого управляющих регистров, RAM и EEPROM.

Работа в среде Silicon Laboratories и работа с внутрисхемным отладчиком

Всплывающее меню File

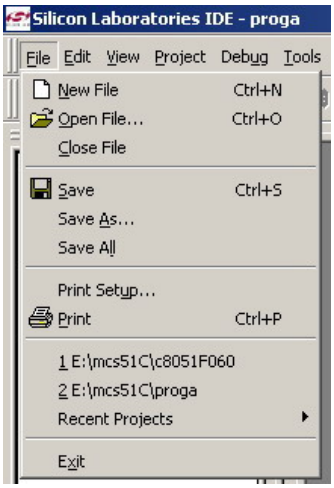


Рис. 4.1. Меню File

Во всплывающем меню File доступны следующие опции: New File, Open File, Close File, Save, Save As, Save All, Print Setup, Print. Они предназначены для создания, сохранения, открытия и закрытия файлов.

Всплывающее меню View

Во всплывающем меню View доступны следующие опции: Debug Windows, Project Window, Output Window, Toolbars, Status bar, Workbook Mode.

Debug Windows

Для просмотра и отработки программы предусмотрены специальные окна, отображающие содержимое специальных регистров, памяти и т. д. Чтобы вызвать данные окна, нужно выбрать View>Debug Window:

- SFR’s – окно регистров специального назначения;
- Watch Window – собственное окно просмотра;
- Registers – окно просмотра содержимого регистров;
- RAM – окно просмотра содержимого ячеек памяти;

- Code Memory – окно просмотра содержимого области памяти;
- Disassembly – окно, отображающее команды на языке ассемблер;
- External Memory – окно просмотра содержимого ячеек внешней памяти;
- Stack – окно просмотра содержимого стека.

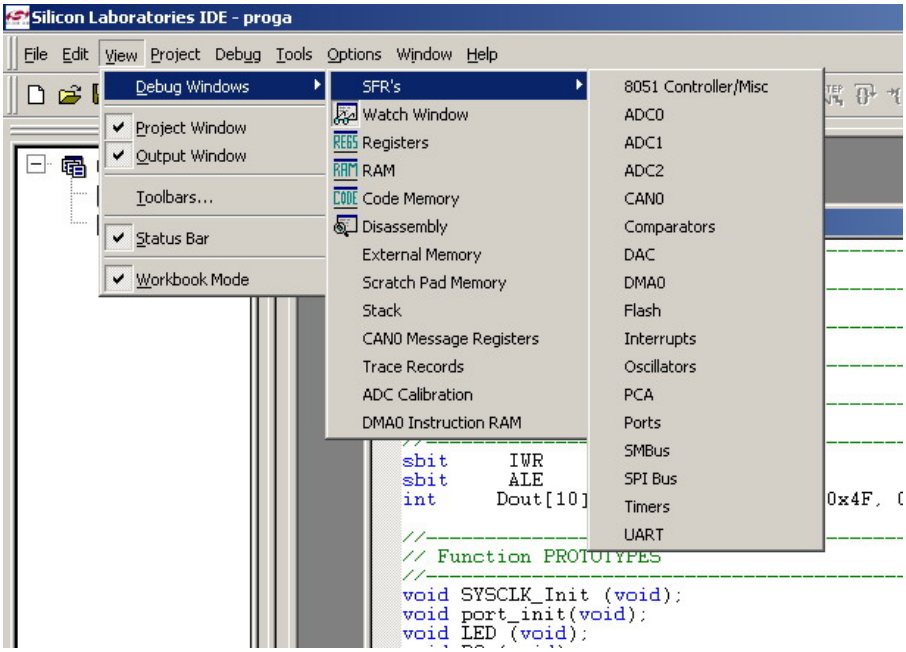


Рис. 4.2. Меню View

Создание нового проекта

1. Выбрать из меню Project>New Project. В результате в «Окне проекта», находящегося слева, появится «дерево», как показано на рис. 4.3.

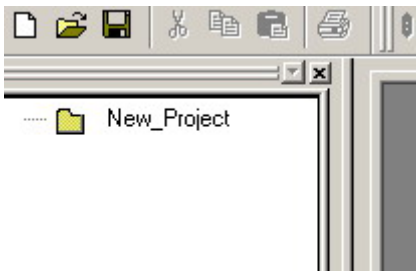


Рис. 4.3. «Дерево» проекта

2. Выбрать из меню File>New file, тем самым создать бланк программы.
3. Сохранить этот бланк нажав File>Save file, указав его название и расширение (например blink.asm)
4. Нажать правой кнопкой на «корень дерева» Вашего проекта, как показано на рис. 4.4, и добавить созданный Вами бланк программы
5. Сохранить созданный проект с указанием его названия, нажав Project>Save Project.

Теперь можно приступить к написанию программы.

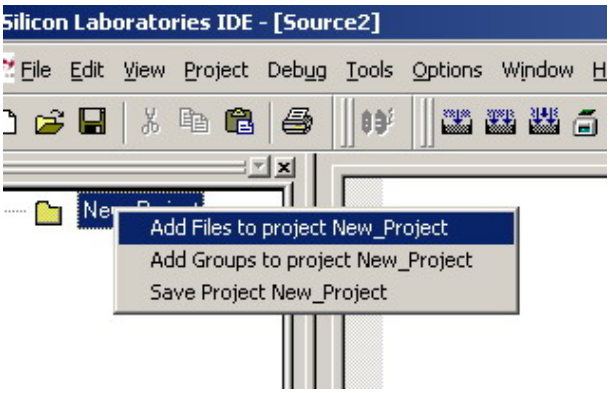


Рис. 4.4. Добавление бланка программы

Функции, необходимые для работы:

- Подключение Debug>Connect;
- Компиляция проекта Project>Assemble/Compile File;
- Запуск программы Debug>Run.
- Пошаговое выполнение программы Debug>Step;
- Сброс контроллера Debug>Reset;

В контроллер можно загрузить только откомпилированный проект, не содержащий ошибок. Компиляция не произойдет, если на пути к каталогу, содержащему компилируемый файл, имеются русскоязычные символы.

Отладка программы

Написав программу, можно посмотреть, как она функционирует. Для этого необходимо откомпилировать программу и записать в контроллер, после чего можно приступить к отладке.

Внизу экрана располагается область, называемая Output Window. Она предназначена для сообщения пользователю о наличии или отсутствии ошибок в программе.

Это сообщение появится после компиляции проекта. Если программа содержит ошибки, то в сообщении будет указанно, в какой строке допущена ошибка, и что именно неверно.

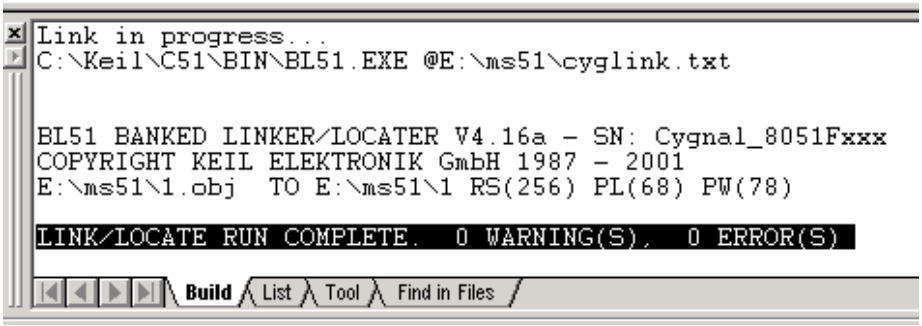


Рис. 4.5. Output Window

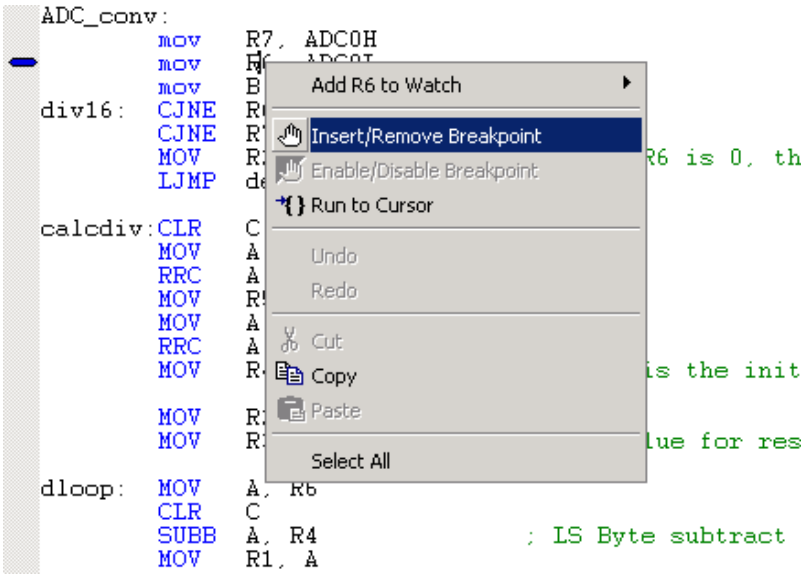


Рис. 4.6. Создание точки останова (breakpoint)

При наличии ошибок программа не будет записана в контроллер.

Одним из доступных средств отладки является Breakpoint, то есть точка останова. Для того чтобы поставить её на какой-нибудь строке программы, нужно нажать на данной строке правой кнопкой мыши и выбрать Insert/Remove Breakpoint.

Если точка останова была установлена, то на поле слева появится значок, означающий, что на этом месте программа остановится, и будет ждать дальнейших указаний.

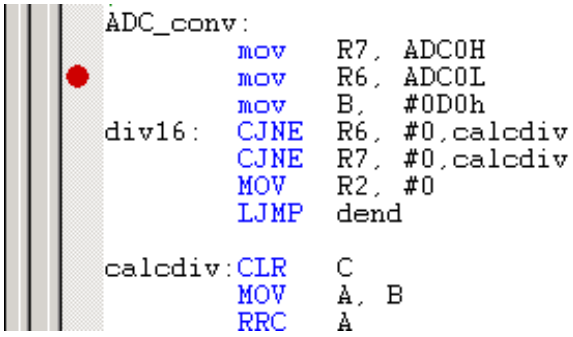


Рис. 4.7. Точка останова (breakpoint)

Для того чтобы снять эту точку, необходимо ещё раз проделать данную процедуру. Метка синего цвета указывает на ту строчку, которая будет выполнена на следующем шаге.

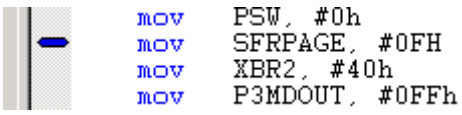


Рис. 4.8. Обозначение следующей выполняемой команды

Также имеется возможность запустить выполнение программы до определенной строки без установки breakpoint. Это может оказаться полезным в том случае, если необходима разовая проверка выполнения программы до этой строки. В этом случае, на нужной строке следует нажать правой кнопкой мыши и выбрать Run to Cursor.

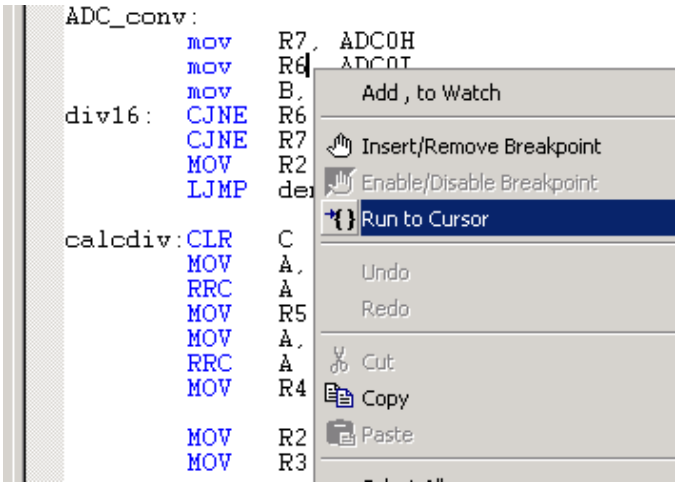


Рис. 4.9. Запуск выполнения программы до текущей строки

Программа выполнится до указанной строки и остановится. После этого можно просмотреть содержимое регистров и т. д., чтобы удостовериться в правильности исполнения программой поставленной задачи. Затем можно снова запустить программу, она продолжит выполнение с того места, на котором была остановлена.

Система команд

Таблица 4.1

Система команд микроконтроллера C8051F060

Мнемоника команды	Описание	Байты	Циклы
	АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ		
ADD A,Rn	Сложение аккумулятора с регистром (n = 0…7)	1	1
ADD A,direct	Сложение аккумулятора с прямо-адресуемым байтом	2	2
ADD A,@Ri	Сложение аккумулятора с косвенно-адресуемым байтом ОЗУ	1	2
ADD A,#data	Сложение аккумулятора с константой	2	2
ADDC A,Rn	Сложение аккумулятора с регистром и переносом	1	1
ADDC A,direct	Сложение аккумулятора с прямо-адресуемым байтом и переносом	2	2

Продолжение табл. 4.1

Мнемоника команды	Описание	Байты	Циклы
ADDC A,@Ri	Сложение аккумулятора с косвенно-адресуемым байтом ОЗУ и переносом	1	2
ADDC A,#data	Сложение аккумулятора с константой и переносом	2	2
SUBB A,Rn	Вычитание из аккумулятора регистра и заема	1	1
SUBB A,direct	Вычитание из аккумулятора прямо-адресуемого байта и заема	2	2
SUBB A,@Ri	Вычитание из аккумулятора косвенно-адресуемого байта ОЗУ и заема	1	2
SUBB A,#data	Вычитание из аккумулятора константы и заема	2	2
INC A	Инкремент аккумулятора	1	1
INC Rn	Инкремент регистра	1	1
INC direct	Инкремент прямо-адресуемого байта	2	2
INC @Ri	Инкремент косвенно-адресуемого байта ОЗУ	1	2
DEC A	Декремент аккумулятора	1	1
DEC Rn	Декремент регистра	1	1
DEC direct	Декремент прямо-адресуемого байта	2	2
DEC @Ri	Декремент косвенно-адресуемого байта ОЗУ	1	2
INC DPTR	Инкремент указателя данных	1	1
MUL AB	Умножение аккумулятора на регистр B	1	4
DIV AB	Деление аккумулятора на регистр B	1	8
DA A	Десятичная коррекция аккумулятора	1	1
	ЛОГИЧЕСКИЕ ОПЕРАЦИИ		
ANL A,Rn	Логическое И аккумулятора и регистра	1	1
ANL A,direct	Логическое И аккумулятора и прямо-адресуемого байта	2	2
ANL A,@Ri	Логическое И аккумулятора и косвенно-адресуемого байта ОЗУ	1	2
ANL A,#data	Логическое И аккумулятора и константы	2	2
ANL direct,A	Логическое И прямо-адресуемого байта и аккумулятора	2	2
ANL direct,#data	Логическое И прямо-адресуемого байта и константы	3	3
ORL A,Rn	Логическое ИЛИ аккумулятора и регистра	1	1
ORL A,direct	Логическое ИЛИ аккумулятора и прямо-адресуемого байта	2	2
ORL A,@Ri	Логическое ИЛИ аккумулятора и косвенно-адресуемого байта ОЗУ	1	2
ORL A,#data	Логическое ИЛИ аккумулятора и константы	2	2

Продолжение табл. 4.1

Мнемоника команды	Описание	Байты	Циклы
ORL direct,A	Логическое ИЛИ прямо-адресуемого байта и аккумулятора	2	2
ORL direct,#data	Логическое ИЛИ прямо-адресуемого байта и константы	3	3
XRL A,Rn	Исключающее ИЛИ аккумулятора и регистра	1	1
XRL A,direct	Исключающее ИЛИ аккумулятора и прямо-адресуемого байта	2	2
XRL A,@Ri	Исключающее ИЛИ аккумулятора и косвенно-адресуемого байта ОЗУ	1	2
XRL A,#data	Исключающее ИЛИ аккумулятора и константы	2	2
XRL direct,A	Исключающее ИЛИ прямо-адресуемого байта и аккумулятора	2	2
XRL direct,#data	Исключающее ИЛИ прямо-адресуемого байта и константы	3	3
CLR A	Сброс аккумулятора	1	1
CPL A	Инверсия аккумулятора	1	1
RL A	Сдвиг аккумулятора влево циклический	1	1
RLC A	Сдвиг аккумулятора влево через перенос	1	1
RR A	Сдвиг аккумулятора вправо циклический	1	1
RRC A	Сдвиг аккумулятора вправо через перенос	1	1
SWAP A	Обмен местами тетрад в аккумуляторе	1	1
	КОМАНДЫ ПЕРЕДАЧИ ДАННЫХ		
MOV A,Rn	Пересылка в аккумулятор из регистра (n = 0…7)	1	1
MOV A,direct	Пересылка в аккумулятор прямо-адресуемого байта	2	2
MOV A,@Ri	Пересылка в аккумулятор косвенно-адресуемого байта ОЗУ	1	2
MOV A,#data	Загрузка в аккумулятор константы	2	2
MOV Rn,A	Пересылка в регистр из аккумулятора	1	1
MOV Rn,direct	Пересылка в регистр прямо-адресуемого байта	2	2
MOV Rn,#data	Загрузка в регистр константы	2	2
MOV direct,A	Пересылка по прямому адресу аккумулятора	2	2
MOV direct,Rn	Пересылка по прямому адресу регистра	2	2
MOV direct,direct	Пересылка прямо-адресуемого байта по прямому адресу	3	3
MOV direct,@Ri	Пересылка косвенно-адресуемого байта ОЗУ по прямому адресу	2	2
MOV direct,#data	Пересылка по прямому адресу константы	3	3

Продолжение табл. 4.1

Мнемоника команды	Описание	Байты	Циклы
MOV @Ri,A	Пересылка в косвенно-адресуемую ячейку ОЗУ аккумулятора	1	2
MOV @Ri,direct	Пересылка в косвенно-адресуемую ячейку ОЗУ прямо-адресуемого байта	2	2
MOV @Ri,#data	Пересылка в косвенно-адресуемую ячейку ОЗУ константы	2	2
MOV DPTR,#data16	Загрузка указателя данных	3	3
MOVC A,@A+DPTR	Пересылка в аккумулятор байта из памяти программ	1	3
MOVC A,@A+PC	Пересылка в аккумулятор байта из памяти программ	1	3
MOVX A,@Ri	Пересылка в аккумулятор байта из внешней памяти данных	1	3
MOVX @Ri,A	Пересылка байта из аккумулятора во внешнюю память данных	1	3
MOVX A,@DPTR	Пересылка в аккумулятор из расширенной внешней памяти данных	1	3
MOVX @DPTR,A	Пересылка из аккумулятора в расширенную внешнюю память данных	1	3
PUSH direct	Загрузка в стек	2	2
POP direct	Извлечение из стека	2	2
XCH A,Rn	Обмен аккумулятора с регистром	1	1
XCH A,direct	Обмен аккумулятора с прямо-адресуемым байтом	2	2
XCH A,@Ri	Обмен аккумулятора с косвенно-адресуемым байтом ОЗУ	1	2
XCHD A,@Ri	Обмен младшей тетрады аккумулятора с младшей тетрадой косвенно-адресуемого байта ОЗУ	1	2
	ОПЕРАЦИИ С БИТАМИ		
CLR C	Сброс переноса	1	1
CLR bit	Сброс бита	2	2
SETB C	Установка переноса	1	1
SETB bit	Установка бита	2	2
CPL C	Инверсия переноса	1	1
CPL bit	Инверсия бита	2	2
ANL C,bit	Логическое И бита и переноса	2	2
ANL C,/bit	Логическое И инверсии бита и переноса	2	2

Окончание табл. 4.1

Мнемоника команды	Описание	Байты	Циклы
ORL C,bit	Логическое ИЛИ бита и переноса	2	2
ORL C,/bit	Логическое ИЛИ инверсии бита и переноса	2	2
MOV C,bit	Пересылка бита в перенос	2	2
MOV bit,C	Пересылка переноса в бит	2	2
JC rel	Переход, если перенос равен единице	2	2/3
JNC rel	Переход, если перенос равен нулю	2	2/3
JB bit,rel	Переход, если бит равен единице	3	3/4
JNB bit,rel	Переход, если бит равен нулю	3	3/4
JBC bit,rel	Переход, если бит установлен, с последующим сбросом бита	3	3/4
	ПРОГРАММНЫЕ ПЕРЕХОДЫ		
ACALL addr11	Абсолютный вызов подпрограммы в пределах страницы в 2 Кбайта	2	3
LCALL addr16	Длинный вызов подпрограммы	3	4
RET	Возврат из подпрограммы	1	5
RETI	Возврат из подпрограммы обработки прерывания	1	5
AJMP addr11	Абсолютный переход внутри страницы в 2 Кбайта	2	3
LJMP addr16	Длинный переход в полном объеме памяти программ	3	4
SJMP rel	Короткий относительный переход внутри страницы в 256 байт	2	3
JMP @A+DPTR	Косвенный относительный переход	1	3
JZ rel	Переход, если аккумулятор равен нулю	2	2/3
JNZ rel	Переход, если аккумулятор не равен нулю	2	2/3
CJNE A,direct,rel	Сравнение аккумулятора с прямо-адресуемым байтом и переход, если не равно	3	3/4
CJNE A,#data,rel	Сравнение аккумулятора с константой и переход, если не равно	3	3/4
CJNE Rn,#data,rel	Сравнение регистра с константой и переход, если не равно	3	3/4
CJNE @Ri,#data,rel	Сравнение косвенно-адресуемого байта ОЗУ с константой и переход, если не равно	3	4/5
DJNZ Rn,rel	Декремент регистра и переход, если не нуль	2	2/3
DJNZ direct,rel	Декремент прямо-адресуемого байта и переход, если не нуль	3	3/4
NOP	Холостая команда	1	1

§ 4.2. Модуль SMBus

SMBus может работать в режимах ведущего и/или ведомого, а также может функционировать на шине с несколькими ведущими. SMBus обеспечивает управление линией SDA (последовательные данные), генерацию тактовых импульсов SCL и синхронизацию, арбитраж, управление битами START/STOP и их генерацию. Интерфейс SMBus способен работать при любом напряжении от 3.0 до 5.0 В, а различные устройства на шине могут иметь различные напряжения питания. Линии SCL (тактовые импульсы) и SDA (последовательные данные) являются двунаправленными. Необходимо подать на них положительное напряжение питания через подтягивающие резисторы или подобную схему. Каждое устройство, подключенное к шине, должно иметь выход с открытым стоком или с открытым коллектором как для линии SCL, так и для линии SDA; тогда при свободной шине обе линии будут «подтянуты» к напряжению высокого логического уровня.

§ 4.2.1. Режимы работы модуля SMBus

Модуль SMBus может быть настроен для работы как в режиме ведущего, так и в режиме ведомого. В любой конкретный момент времени он может работать в одном из четырех режимов: ведущий передатчик, ведущий приемник, ведомый передатчик, ведомый приемник. Значение регистра состояния SMB0STA определяет состояние режима передачи модуля SMBus (см. табл. 4.2). Приведенные ниже описания режимов показывают применение модуля SMBus с использованием управления по прерываниям; кроме этого работа с модулем SMBus возможна в режиме опроса.

Таблица 4.2

Коды состояния модуля SMBus

Код состояния (SMB0STA)	Режим	Состояние модуля SMBus	Типичное действие
0x00	Все режимы	Ошибка шины (т.е. некорректный START, некорректный STOP, …)	Установка STO для сброса SMBus
0x08	Ведущий передатчик/приемник	Передан бит START.	Загрузка SMB0DAT адресом ведомого + R/W. Сброс STA.
0x10	Ведущий передатчик/приемник	Передан бит «повторный START».	Загрузка SMB0DAT адресом ведомого + R/W. Сброс STA.
0x18	Ведущий передатчик	Передан адрес ведомого + W. Получен ACK.	Загрузка SMB0DAT данными для передачи
0x20	Ведущий передатчик	Передан адрес ведомого + W. Получен NACK.	Повтор опроса подтверждения. Установка STO + STA.
0x28	Ведущий передатчик	Передан байт данных. Получен ACK.	1) Загрузка SMB0DAT следующим байтом, или 2) Установка STO, или 3) Сброс STO, а затем установка STA для передачи бита «повторный START»

Окончание табл. 4.2

Код состояния (SMB0STA)	Режим	Состояние модуля SMBus	Типичное действие
0x30	Ведущий передатчик	Передан байт данных. Получен NACK.	1) Повтор передачи, или 2) Установка STO
0x38	Ведущий передатчик	Потерян арбитраж.	Сохранение текущих данных
0x40	Ведущий приемник	Передан адрес ведомого + R. Получен ACK.	Если необходимо принять только один байт, то сброс AA (передача NACK после приема байта). Ожидание принимаемых данных
0x48	Ведущий приемник	Передан адрес ведомого + R. Получен NACK.	Повтор опроса подтверждения. Установка STO + STA.
0x50	Ведущий приемник	Получен байт данных. Передан ACK.	Чтение SMB0DAT. Ожидание следующего байта. Если следующий байт является последним, то сброс AA.
0x58	Ведущий приемник	Получен байт данных. Передан NACK.	Установка STO.
0x60	Ведомый приемник	Получен собственный адрес ведомого + W. Передан ACK.	Ожидание данных.
0x68	Ведомый приемник	При передаче в ведущем режиме адреса ведомого + R/W потерян арбитраж. Получен собственный адрес ведомого + W. Передан ACK.	Сохранить текущие данные для повтора передачи, когда шина освободится. Ожидание данных.
0x70	Ведомый приемник	Получен адрес общего вызова (0x00). Передан ACK.	Ожидание данных.
0x78	Ведомый приемник	При передаче в ведущем режиме адреса ведомого + R/W потерян арбитраж. Получен адрес общего вызова (0x00). Передан ACK.	Сохранить текущие данные для повтора передачи, когда шина освободится.
0x80	Ведомый приемник	Получен собственный адрес ведомого + W. Получен байт данных. Передан ACK.	Чтение SMB0DAT. Ожидание следующего байта или STOP.
0x88	Ведомый приемник	Получен собственный адрес ведомого + W. Получен байт данных. Передан NACK.	Установка STO для сброса SMBus.
0x90	Ведомый приемник	Получен адрес общего вызова (0x00). Получен байт данных. Передан ACK.	Чтение SMB0DAT. Ожидание следующего байта или STOP.
0x98	Ведомый приемник	Получен адрес общего вызова (0x00). Получен байт данных. Передан NACK.	Установка STO для сброса SMBus.
0xA0	Ведомый приемник	Получен бит STOP или «повторный START», когда устройство адресуется в качестве ведомого.	Никаких действий не требуется.
0xA8	Ведомый передатчик	Получен собственный адрес ведомого + R. Передан ACK.	Загрузка SMB0DAT данными для передачи.
0xB0	Ведомый передатчик	При передаче в ведущем режиме адреса ведомого + R/W потерян арбитраж. Получен собственный адрес ведомого + R. Передан ACK.	Сохранить текущие данные для повтора передачи, когда шина освободится. Загрузка SMB0DAT данными для передачи.
0xB8	Ведомый передатчик	Передан байт данных. Получен ACK.	Загрузка SMB0DAT данными для передачи
0xC0	Ведомый передатчик	Передан байт данных. Получен NACK.	Ожидание бита STOP.
0xC8	Ведомый передатчик	Передан последний байт данных (AA=0). Получен ACK.	Установка STO для сброса SMBus.
0xD0	Ведомый передатчик/приемник	Истек таймаут высокого уровня на линии SCL, определяемый значением регистра SMB0CR (при FTE=1)	Установка STO для сброса SMBus.
0xF8	Все режимы	Простой (ожидание)	Флаг SI не установлен.

Режим ведущего передатчика

Последовательные данные выдаются на линию SDA, а тактовые импульсы выдаются на линию SCL. SMBus генерирует бит START и затем передает первый байт, который содержит адрес целевого ведомого устройства и бит направления. В этом случае бит направления (R/W) должен быть сброшен в 0, инициируя операцию записи. Затем модуль SMBus передает один или несколько байт последовательных данных, ожидая подтверждения (ACK) от ведомого после каждого байта. Для обозначения конца сеанса передачи последовательных данных ведущее устройство генерирует бит STOP.

Режим ведущего приемника

Последовательные данные принимаются с линии SDA, а тактовые импульсы выдаются на линию SCL. Модуль SMBus генерирует бит START и затем передает первый байт, который содержит адрес целевого ведомого устройства и бит направления. В этом случае бит направления (R/W) должен быть установлен в 1, инициируя операцию чтения. Модуль SMBus принимает последовательные данные от ведомого по линии SDA, при этом генерирует тактовые импульсы на линии SCL. После приема каждого байта модуль SMBus генерирует биты подтверждения (ACK) или неподтверждения (NACK) в зависимости от состояния бита AA регистра SMB0CN. Для обозначения конца сеанса передачи последовательных данных ведущий генерирует бит STOP.

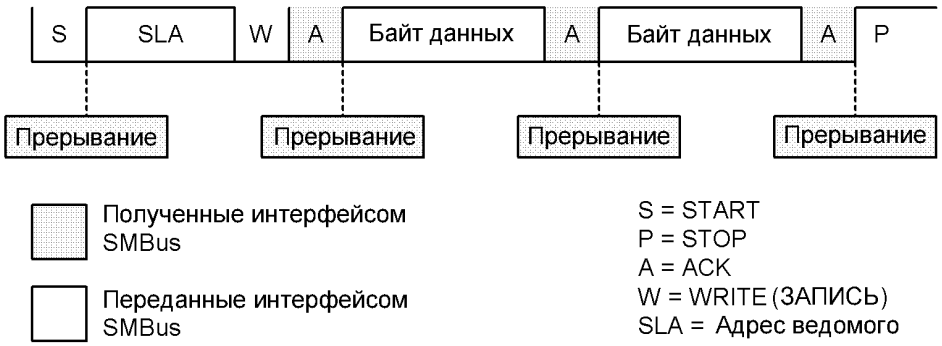


Рис. 4.10. Передача данных в режиме ведущего

Режим ведомого передатчика

Последовательные данные выдаются на линию SDA, а тактовые импульсы принимаются с линии SCL. Модуль SMBus принимает бит START, а вслед за ним байт данных, который содержит адрес ведомого и бит направления. В этом случае бит направления (R/W) должен быть установлен в 1, инициируя операцию чтения.

Рис. 4.11. Прием данных в режиме ведущего

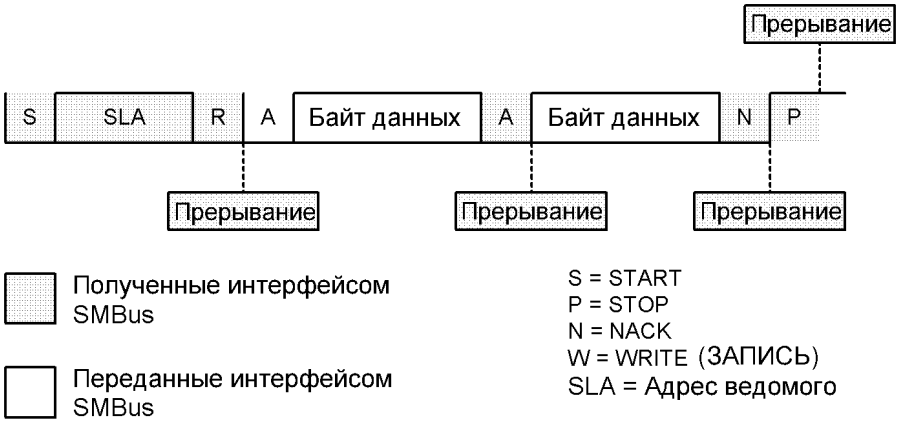


Рис. 4.12. Передача данных в режиме ведомого

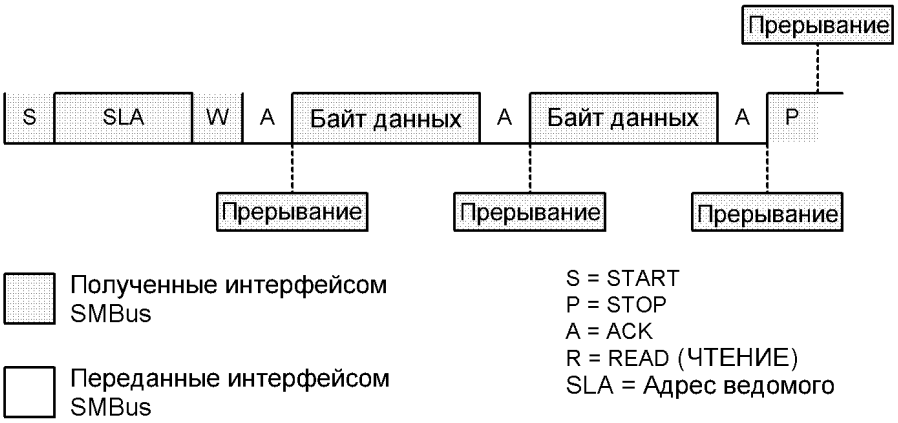


Рис. 4.13. Прием данных в режиме ведомого

Если принятый адрес ведомого соответствует адресу, хранящемуся в регистре SMB0ADR, то модуль SMBus генерирует бит подтверждения (ACK). Модуль SMBus также будет генерировать бит подтверждения (ACK), если принятый адрес является адресом общего вызова (0x00) и бит разрешения адреса общего вызова (SMB0ADR.0) установлен в 1. Модуль SMBus принимает тактовые импульсы по линии SCL и передает ведущему один или несколько байт последовательных данных, ожидая подтвержде-

ния (ACK) от ведущего после каждого байта. Модуль SMBus выходит из режима ведомого после приема бита STOP от ведущего.

Режим ведомого приемника

Последовательные данные принимаются с линии SDA, а тактовые импульсы принимаются с линии SCL. Модуль SMBus принимает бит START, а вслед за ним байт данных, который содержит адрес ведомого и бит направления. В этом случае бит направления (R/W) должен быть сброшен в 0, инициируя операцию записи. Если принятый адрес ведомого соответствует адресу, хранящемуся в регистре SMB0ADR, то модуль SMBus генерирует бит подтверждения (ACK). Модуль SMBus также будет генерировать бит подтверждения (ACK), если принятый адрес является адресом общего вызова (0x00) и бит разрешения адреса общего вызова (SMB0ADR.0) установлен в 1. Модуль SMBus принимает один или несколько байт последовательных данных; после приема каждого байта модуль SMBus передает биты подтверждения (ACK) или неподтверждения (NACK) в зависимости от состояния бита AA регистра SMB0CN. Модуль SMBus выходит из режима ведомого приемника после приема бита STOP от ведущего.

§ 4.2.2. Регистры специального назначения модуля SMBus

Для доступа к интерфейсу SMBus и управления им используются пять регистров SFR:

- регистр управления SMB0CN;
- регистр установки тактовой частоты SMB0CR;
- регистр адреса SMB0ADR;
- регистр данных SMB0DAT;
- регистр состояния SMB0STA.

Регистр управления SMB0CN

Регистр управления SMB0CN используется для управления модулем SMBus и его настройки. Все биты этого регистра можно читать и записывать программно. Два из управляющих битов также устанавливаются модулем SMBus аппаратно. Флаг прерывания от последовательного порта (SI, SMB0CN.3) устанавливается в 1 аппаратно при возникновении прерывания от модуля SMBus. Он может быть сброшен только программно. Флаг STOP (STO, SMB0CN.4) устанавливается в 1 программно. Этот флаг сбрасывается в 0 аппаратно при обнаружении на шине бита STOP.

Установка в 1 флага ENSMB включает модуль SMBus. Сброс в 0 флага ENSMB отключает модуль SMBus и удаляет его с шины. Сброс флага ENSMB и затем повторная его установка в 1 приведут к сбросу модуля SMBus. Однако флаг ENSMB не следует использовать для временного удаления устройства с шины, т. к. информация о состо-

янии шины будет потеряна. Вместо этого для временного удаления устройства с шины следует использовать флаг назначения подтверждения AA (описание флага AA приведено ниже).

Установка в 1 флага запуска (STA, SMB0CN.5) переведет модуль SMBus в режим ведущего. Если шина свободна, модуль SMBus сгенерирует бит START. Если шина занята, то модуль SMBus будет ожидать бита STOP, свидетельствующего об освобождении шины, и затем сгенерирует бит START через 5мкс после задержки, определяемой значением регистра SMB0CR. (В соответствии с протоколом SMBus, модуль SMBus также будет считать шину свободной, если шина простаивает в течение 50мкс и бит STOP не обнаружен). Если бит STA устанавливается в 1 в то время, когда модуль SMBus находится в режиме ведущего и уже переданы один или несколько байт, то будет сгенерировано событие «повторный START».

Если флаг окончания передачи (STO, SMB0CN.4) устанавливается в 1 в то время, когда модуль SMBus находится в режиме ведущего, то модуль SMBus сгенерирует на шине бит STOP. В режиме ведомого флаг STO можно использовать для восстановления из состояния сбоя. В этом случае бит STOP не генерируется, но модуль SMBus ведет себя так, как будто бит STOP уже получен, и переходит в режим «неадресованного» ведомого приемника. Следует иметь ввиду, что этот условный бит STOP не вызовет освобождения шины. Шина будет оставаться занятой до тех пор, пока на ней не появится бит STOP или пока не произойдет условие таймаута освобождения шины. При обнаружении на шине бита STOP модуль SMBus автоматически сбрасывает в 0 флаг STO.

Флаг прерываний от последовательного порта (SI, SMB0CN.3) устанавливается аппаратно в 1, если интерфейс SMBus переходит к одному из 27 возможных состояний. Если прерывания от модуля SMBus разрешены, то при установке в 1 флага SI генерируется запрос прерывания. Флаг SI должен быть сброшен программно.

Примечание. Если флаг SI установлен в 1 в то время, когда на линии SCL удерживается низкий уровень сигнала, то период тактового импульса будет «растягиваться» (на участке с низким уровнем сигнала) и передача последовательных данных по шине приостановится до тех пор, пока не будет сброшен в 0 флаг SI. На длительность высокого уровня сигнала на линии SCL установка флага SI не влияет.

Флаг назначения подтверждения AA (AA, SMB0CN.2) используется для задания уровня сигнала на линии SDA во время тактового импульса подтверждения на линии SCL. Установка в 1 флага AA приведет к передаче бита подтверждения ACK (низкий уровень сигнала на линии SDA) во время тактового импульса подтверждения на линии SCL, если устройство распознало свой адрес.

Сброс в 0 флага AA приведет к передаче бита «нет подтверждения» NACK (высокий уровень сигнала на линии SDA) во время тактового импульса подтверждения на линии SCL. После передачи байта в режиме ведомого ведомое устройство можно временно удалить с шины путем сброса в 0 флага AA. Собственный адрес ведомого и адрес общего вызова будут игнорироваться. Для восстановления работы на шине необходимо установить в 1 флаг AA, чтобы разрешить ведомому распознавать свой адрес.

Установка в 1 бита разрешения таймера освобождения шины SMBus (FTE, SMB0CN.1) включит таймер отсчета таймаута освобождения шины, который определяется значением регистра SMB0CR. Если на линии SCL удерживается высокий уровень сигнала, то таймер отсчитывает таймаут, определяемый регистром SMB0CR. Переполнение таймера означает истечение таймаута освобождения шины: если модуль SMBus ожидает момента для генерации бита START, то он сгенерирует его после истечения данного таймаута. Период освобождения шины должен быть не более 50мкс.

Когда бит (TOE, SMB0CN.0) установлен в 1, Таймер 4 используется для отсчета таймаута низкого уровня сигнала на линии SCL. Если Таймер 4 включен, то он будет перезагружаться, когда на линии SCL присутствует сигнал высокого уровня, и будет отсчитывать таймаут, когда на линии SCL присутствует сигнал низкого уровня.

Если Таймер 4 включен и настроен на переполнение через 25мс (и бит TOE установлен в 1), то переполнение Таймера 4 означает истечение таймаута низкого уровня сигнала на линии SCL; в этом случае для сброса модуля SMBus можно использовать процедуру обработки прерывания от Таймера 4.

Регистр управления SMB0CN

R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
BUSY	ENSMB	STA	STO	SI	AA	FTE	TOE	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0 (доступен в битовом режиме адресации)	SFR Адрес: 0xC0 SFR страница: 0

Бит 7: BUSY: Флаг занятости шины SMBus.
0: Шина SMBus свободна
1: Шина SMBus занята

Бит 6: ENSMB: Включение модуля SMBus0.
Этот бит включает/отключает последовательный интерфейс SMBus0.
0: SMBus0 отключен.
1: SMBus0 включен.

Бит 5: STA: Флаг запуска модуля SMBus0.
0: Бит START не передается.
1: При работе в режиме ведущего бит START передается, если шина свободна. (Если шина не свободна, то бит START передается после приема бита STOP.) Если бит STA устанавливается после передачи или приема одного или нескольких байт и до приема бита STOP, то передается бит «повторный START».

Бит 4: STO: Флаг окончания передачи модуля SMBus0.
0: Бит STOP не передается.
1: Установка в 1 бита STO приведет к передаче бита STOP. При приеме бита STOP флаг STO аппаратно сбрасывается в 0. Если оба флага STA и STO установлены в 1, то вслед за битом STOP передается бит START. В режиме ведомого установка флага STO заставит модуль SMBus0 вести себя так, как будто получен бит STOP.

Бит 3: SI: Флаг прерывания от последовательного порта SMBus.
Этот бит устанавливается аппаратно при переходе модуля SMBus к одному из 27 возможных состояний. (Состояние с кодом 0xF8 не вызывает установку бита SI.) Если прерывание от SI разрешено, то установка этого бита приведет к переходу на процедуру обслуживания прерывания от модуля SMBus. Этот бит автоматически аппаратно не сбрасывается и должен быть сброшен программно.

Бит 2: AA: Флаг назначения подтверждения .
Этот бит определяет тип бита подтверждения, передаваемого во время тактового цикла подтверждения на линии SCL.
0: Во время тактового цикла подтверждения передается бит «нет подтверждения» (высокий уровень сигнала на линии SDA).
1: Во время тактового цикла подтверждения передается бит «подтверждение» (низкий уровень сигнала на линии SDA).

Бит 1: FTE: Бит разрешения таймера освобождения шины SMBus.
0: Не используется таймаут высокого уровня на линии SCL
1: Если время удержания высокого уровня на линии SCL превышает предел, определяемый значением регистра SMB0CR, то происходит условие таймаута.

Бит 0: TOE: Бит разрешения таймаута SMBus.
0: Не используется таймаут низкого уровня на линии SCL.
1: Если время удержания низкого уровня на линии SCL превышает предел, определяемый Таймером 4 (если он включен), то происходит условие таймаута.

Регистр установки тактовой частоты SMB0CR

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xCF SFR страница: 0

Биты 7-0: SMB0CR.[7:0]: Установка тактовой частоты модуля SMBus0

Регистр установки тактовой частоты SMB0CR управляет частотой тактовых импульсов, выдаваемых на линию SCL в режиме ведущего. 8-разрядное слово, сохраненное в регистре SMB0CR, загружается в специальный 8-разрядный таймер. Этот таймер считает в прямом направлении и когда он переполнится (из состояния 0xFF в состояние 0x00), состояние сигнала на линии SCL изменится на противоположное.

Значение SMB0CR ограничивается следующим уравнением:

$SMB0CR < ((288 - 0.85 * SYSCLK) / (1.125 * 10^6))$, где

SMB0CR – 8-разрядное значение (без знака) регистра SMB0CR;
SYSCLK – системная тактовая частота в [Гц].

Длительность удержания низкого и высокого уровней тактового сигнала на линии SCL определяется следующими уравнениями:

$T_{LOW} = (256 - SMB0CR) / SYSCLK$

$T_{HIGH} = (258 - SMB0CR) / SYSCLK + 625нс$

Значение регистра SMB0CR определяет также таймаут освобождения шины в соответствии со следующим уравнением:

$T_{BFT} = 10 * [(256 - SMB0CR) + 1] / SYSCLK$

Регистр данных SMB0DAT

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xC2 SFR страница: 0

Биты 7-0: SMB0DAT: Данные модуля SMBus0.

Регистр SMB0DAT содержит байт данных, которые должны передаваться последовательному интерфейсу SMBus, или данные, только что принятые от последовательного интерфейса SMBus. Читать из этого регистра или записывать в этот регистр можно всегда, когда флаг прерывания от последовательного порта SI (SMB0CN.3) установлен в 1. Когда флаг SI не установлен в 1, система может находится в процессе сдвига данных в регистр (или из регистра) SMB0DAT и обращаться к этому регистру нельзя.

Регистр данных модуля SMBus0 SMB0DAT содержит байт последовательных данных, который необходимо передать, или байт последовательных данных, который только что принят. Программа может прочитать из регистра или записать в регистр данные, когда флаг SI установлен в 1; программа не должна пытаться обратиться к регистру SMB0DAT, когда модуль SMBus включен и флаг SI сброшен в 0, т. к. в этот момент может осуществляться аппаратный сдвиг байта данных в регистр или из регистра.

Данные всегда сдвигаются старшими разрядами вперед. После приема байта первый бит принятых данных занимает старший разряд регистра SMB0DAT. Когда данные выдвигаются из регистра, они одновременно появляются на шине. Поэтому регистр SMB0DAT всегда содержит последний байт данных, присутствующий в настоящий момент на шине. Таким образом, в случае потери арбитража переход от ведущего передатчика к ведомому приемнику осуществляется с корректными данными в регистре SMB0DAT.

Регистр адреса SMB0ADR

Регистр адреса SMB0ADR содержит адрес ведомого для интерфейса SMBus0. В ведомом режиме семь старших значащих битов образуют 7-битный адрес ведомого. Младший значащий бит, бит 0, используется для разрешения распознавания адреса общего вызова (0x00). Если бит 0 установлен в 1, адрес общего вызова будет распознаваться. В противном случае, адрес общего вызова будет игнорироваться. Содержимое этого регистра игнорируется, если модуль SMBus0 работает в ведущем режиме.

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
SLV6	SLV5	SLV4	SLV3	SLV2	SLV1	SLV0	GC	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xC3 SFR страница: 0

Биты 7-1: SLV6-SLV0: Адрес ведомого SMBus
Эти биты загружаются 7-разрядным адресом ведомого, на который будет отвечать модуль SMBus при работе в качестве ведомого передатчика или ведомого приемника. SLV6 является старшим значащим битом адреса и соответствует первому биту адресного байта, полученного по шине SMBus.

Бит 0: GC: Разрешение адреса общего вызова.
Этот бит используется для разрешения распознавания адреса общего вызова (0x00).
0: Адрес общего вызова игнорируется.
1: Адрес общего вызова распознается.

Регистр состояния SMB0STA

Регистр состояния SMB0STA содержит 8-битный код состояния, показывающий текущее состояние модуля SMBus0. Существует 28 возможных состояний модуля SMBus, каждому из которых соответствует уникальный код состояния. Пять старших значащих битов кода состояния могут иметь различные значения, а три младших значащих бита для корректных кодов состояния всегда равны нулю, когда SI = 1. Поэтому все возможные коды состояния кратны восьми. Это позволяет применять в программе код состояния в качестве индекса, используемого для перехода на соответствующую процедуру обслуживания (используя 8 байт кода для обслуживания состояния или для перехода на более сложную процедуру обслуживания).

Для нужд программы пользователя содержимое регистра SMB0STA определено только тогда, когда флаг SI установлен в 1. Программа никогда не должна записывать данные в регистр SMB0STA. Это приведет к неопределенному результату. В табл. 4.2 приведены все 28 состояний модуля SMBus вместе с соответствующими им кодами.

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение
STA7	STA6	STA5	STA4	STA3	STA2	STA1	STA0	при сбросе: 11111000
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xC1 SFR страница: 0

Биты 7-3: STA7-STA3: Код состояния модуля SMBus0.
Эти биты содержат код состояния модуля SMBus0. Существует 28 возможных кодов состояния. Каждый код состояния соответствует единственному состоянию модуля SMBus0. Корректный код состояния присутствует в регистре SMB0STA, когда флаг SI (SMB0CN.3) установлен в 1. Содержимое регистра SMB0STA не определено, когда флаг SI равен нулю. Запись в регистр SMB0STA в любое время даст неопределенный результат.

Биты 2-0: STA2-STA0: Три младших значащих бита регистра SMB0STA всегда читаются как 0, когда флаг SI установлен в 1

§ 4.3. Алгоритмы работы с модулем SMBus

Работа модуля SMBus в режиме I2C может быть организована как с использованием прерываний, так и без них. Ниже приведены алгоритмы работы с модулем без использования прерываний. Кроме того, в данных алгоритмах не предусмотрена обработка ошибок, что также необходимо учитывать при отладке программы.

§ 4.3.1. Алгоритм инициализации модуля SMBus

Прежде чем начать обмен данными по шине I2C необходимо проинициализировать модуль SMBus контроллера. Ниже приведен один из возможных алгоритмов инициализации.

1. Бит ENSMB (регистр SMB0CN, бит 6) = 0 (выключение модуля SMBus)
2. *Режим ведомого*
   Бит AA (регистр SMB0CN, бит 2) = 1 (подтверждение приема байта адреса/данных)

Режим ведущего

Не имеет значения

3. Если нужно учитывать таймаут высокого уровня на линии SCL, то бит FTE (регистр SMB0CN, бит 1) = 1
4. Если нужно учитывать таймаут низкого уровня на линии SCL, то бит TOE (регистр SMB0CN, бит 1) = 1
5. *Режим ведомого*

   Запись в регистр SMB0ADR адреса, который присваивается ведомому. Если нужна поддержка адреса общего вызова, то бит 0 адреса должен быть установлен в «1». Если поддержка адреса общего вызова не нужна, то бит 0 адреса должен быть сброшен в «0».

   Режим ведущего

   Запись в регистр SMB0CR значения, задающего скорость обмена данными по шине I2C. При этом длительность удержания низкого и высокого уровней тактового сигнала на линии SCL определяется следующими выражениями:

$$T_{LOW} = \frac{256 - SMB0CR}{SYSCLK}$$

$$T_{HIGH} = \frac{258 - SMB0CR}{SYSCLK} + 625\,нс$$

6. Бит SI (регистр SMB0CN, бит 3) = 0 (сброс флага прерывания от модуля SMBus)
7. Бит ENSMB (регистр SMB0CN, бит 6) = 1 (включение модуля SMBus)

Следует отметить, что обмен данными по шине I2C возможен только в случае правильной настройки приоритетного декодера матрицы. Более подробно приоритетный декодер описан в [5].

§ 4.3.2. Алгоритм формирования условия START

Для формирования условия START на шине I2C нужно выполнить следующие действия:

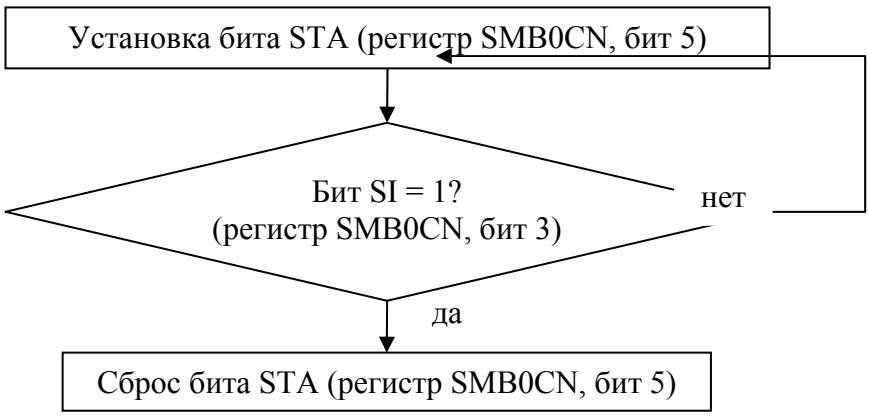


Рис. 4.14. Алгоритм формирования условия START

§ 4.3.3. Алгоритм формирования условия RESTART

Условие RESTART может быть сформировано по следующему алгоритму:

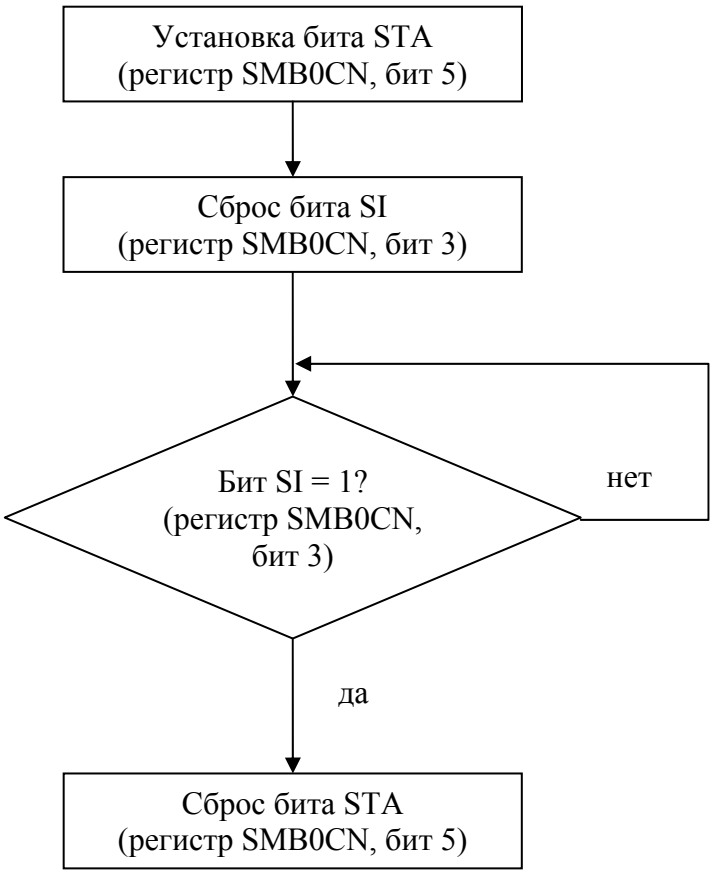


Рис. 4.15. Алгоритм формирования условия RESTART

§ 4.3.4. Алгоритм формирования условия STOP

Одним из алгоритмов формирования условия STOP является следующий:

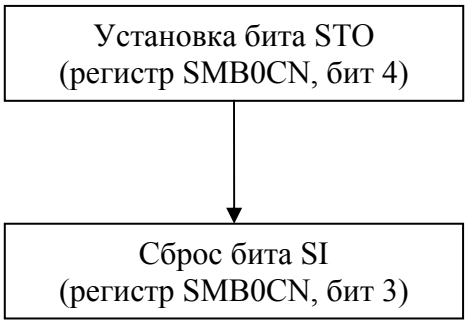


Рис. 4.16. Алгоритм формирования условия STOP

§ 4.3.5. Алгоритм приема байта ведомым

Для организации приема байта ведомым устройством может быть применен следующий алгоритм:

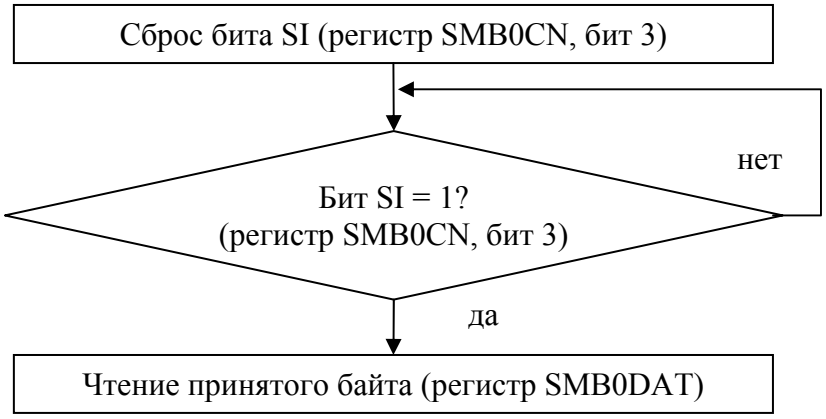


Рис. 4.17. Алгоритм приема байта ведомым

§ 4.3.6. Алгоритм приема ведомым последнего байта

Для организации приема последнего байта ведомым устройством может быть применен следующий алгоритм:

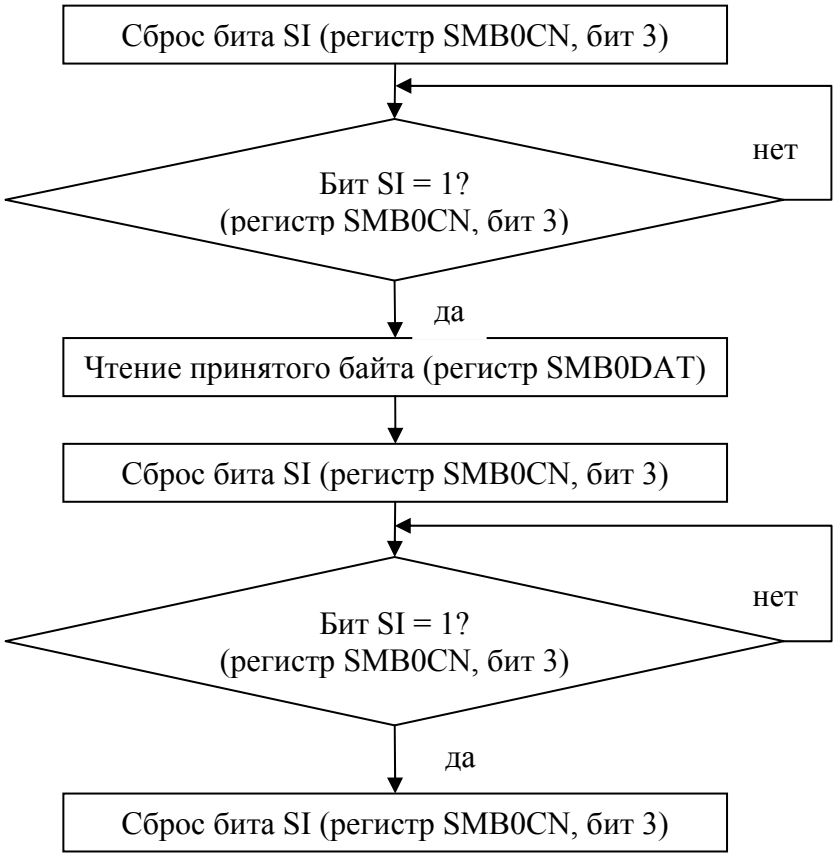


Рис. 4.18. Алгоритм приема ведомым последнего байта

§ 4.3.7. Алгоритм передачи байта ведомым

Передача байта ведомым устройством может быть организована следующим образом:

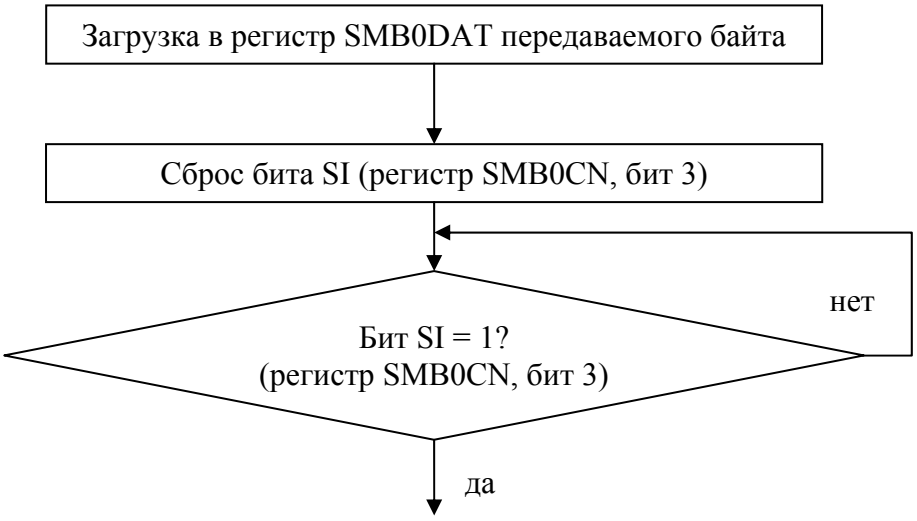


Рис. 4.19. Алгоритм передачи байта ведомым

§ 4.3.8. Алгоритм передачи ведомым последнего байта

Передача последнего байта ведомым устройством может быть организована следующим образом:

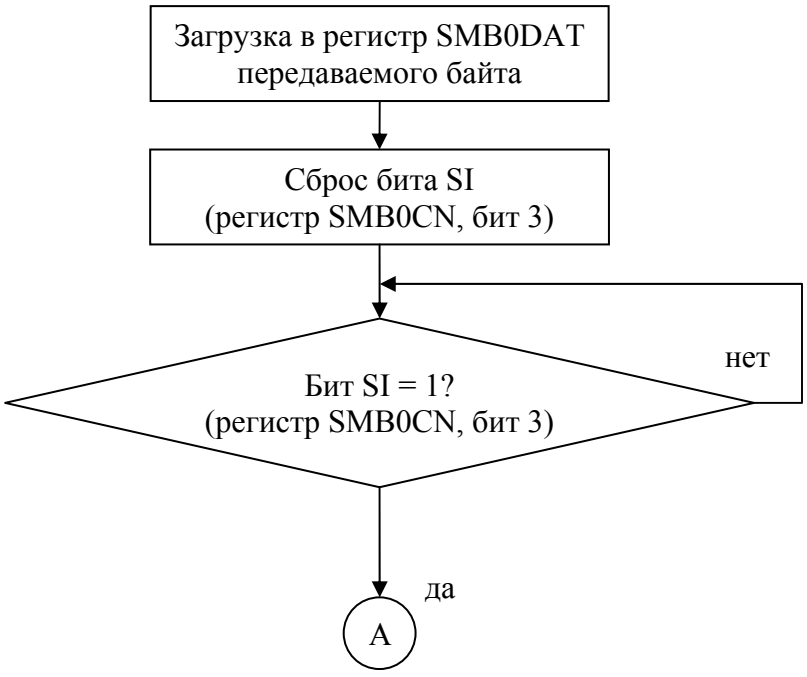


Рис. 4.20. Алгоритм передачи ведомым последнего байта (начало)

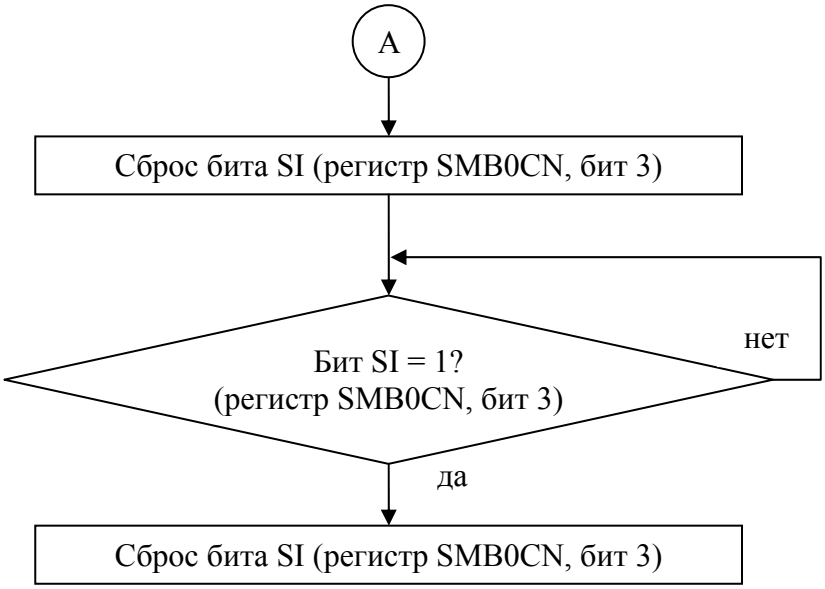


Рис. 4.21. Алгоритм передачи ведомым последнего байта (окончание)

§ 4.3.9. Алгоритм передачи байта ведущим

Ведущее устройство может передать байт, используя следующий алгоритм:

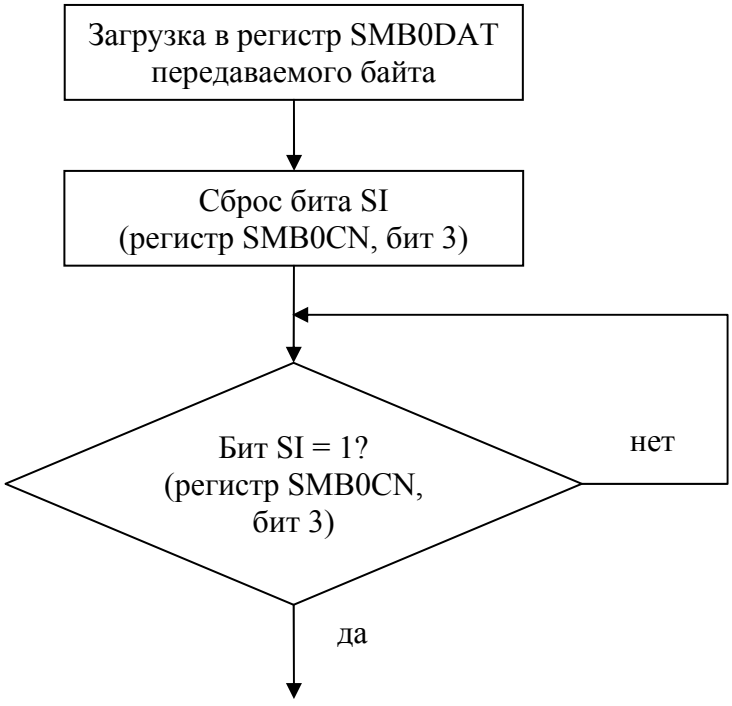


Рис. 4.22. Алгоритм передачи байта ведущим

§ 4.3.10. Алгоритм приема байта ведущим

Прием байта ведущим устройством с шины I2C можно организовать по следующему алгоритму:

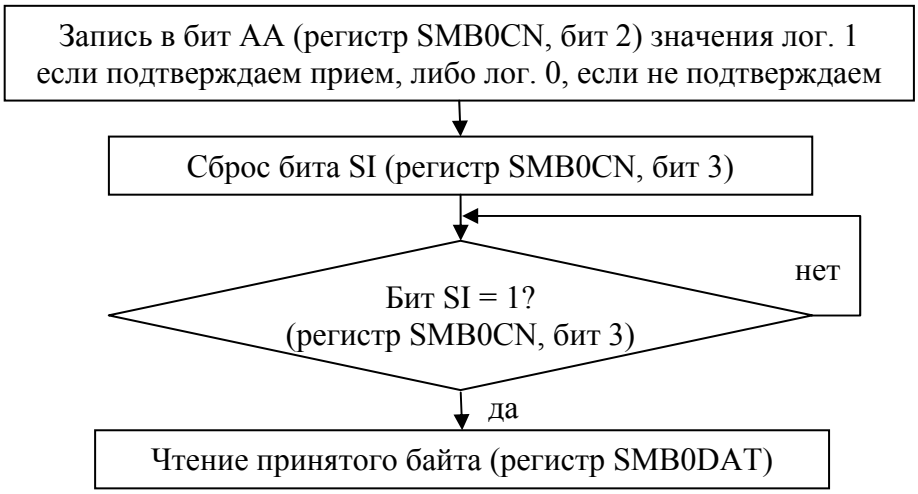


Рис. 4.23. Алгоритм приема байта ведущим

Рис. 4.24. Алгоритм программы ведущего микроконтроллера (начало)

§ 4.4. Примеры программ

§ 4.4.1. Пример программы ведущего микроконтроллера

В качестве примера рассмотрим программу, предназначенную для работы с микросхемой EEPROM памяти 24LC64. Программа производит запись и последующее чтение двух байт данных. Алгоритм программы представлен на рис. 4.24 и 4.25.

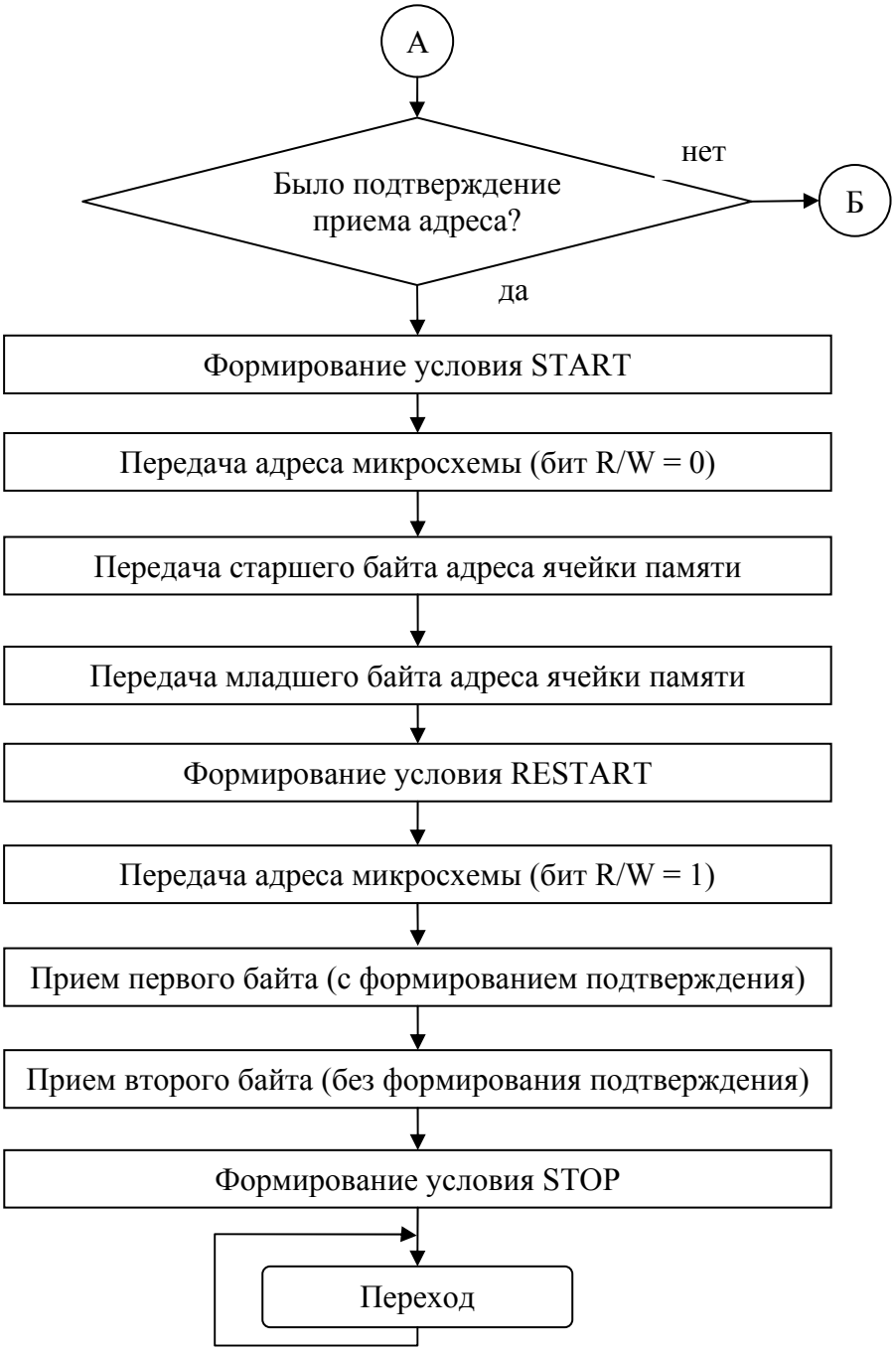


Рис. 4.25. Алгоритм программы ведущего микроконтроллера (окончание)

Алгоритмы подпрограмм, формирующих условия START, RE-START, STOP, а так же осуществляющих прием и передачу байта приведены в § 4.3.

Ниже приведена программа, реализующая описанный алгоритм. Программа написана для микроконтроллера C8051F060. Предполагается, что микросхема памяти 24LC64 имеет адрес 0xA8, а в качестве линий SDA и SCL используются соответственно линии 5 и 6 порта P0 микроконтроллера. Обмен данными по шине I2C ведется на частоте 100 кГц.

```
$NOMOD51
$include (c8051f060.inc)

CONST_ADR_W equ 0A8h      ;байт адреса микросхемы (для записи, бит R/W = 0)
CONST_ADR_R equ 0A9h      ;байт адреса микросхемы (для чтения, бит R/W = 1)

CONST_ACK equ 018h        ;код состояния модуля «было подтверждение адреса»

HIGH_A equ 012h           ;старший байт адреса ячейки памяти
LOW_A equ 034h            ;младший байт адреса ячейки памяти

BYTE_1 equ 0AAh           ;первый сохраняемый байт
BYTE_2 equ 0BBh           ;второй сохраняемый байт

TEMP_1 equ 030h           ;регистр для хранения первого принятого байта
TEMP_2 equ 031h           ;регистр для хранения второго принятого байта

;-------------------------------------------------------------------------------
;инициализация
;-------------------------------------------------------------------------------

;запрещение сторожевого таймера
        mov WDTCN, #0DEh
        mov WDTCN, #0ADh

;настройка модуля SPI на трехпроводный режим работы
        mov SFRPAGE, #00h     ;выбор страницы SFR с регистром SPI0CN
        mov SPI0CN, #02h      ;настройка модуля SPI на трехпроводной режим работы
                              ;чтобы обеспечить подключение линий SDA и SCL
                              ;к линиям P0.5 и P0.6
;настройка и включение матрицы
        mov SFRPAGE, #0Fh     ;выбор страницы SFR с регистрами XBR0 и XBR2
        mov XBR0, #07h        ;подключение входов/выходов
                              ;модулей UART0, SPI и SMBus
        mov XBR2, #40h        ;включение матрицы

;настройка модуля SMBus
        mov SFRPAGE, #00h     ;выбор страницы SFR с регистрами модуля SMBus
        mov SMB0CR, #241      ;задание скорости обмена данными 100 кГц
        clr SMB0CN.3          ;сброс флага прерывания от модуля SMBus
        setb SMB0CN.6         ;включение модуля SMBus
```

```
;-------------------------------------------------------------------------------
;основная программа
;-------------------------------------------------------------------------------

;запись двух байт данных в микросхему памяти
        acall START             ;формирование условия START

        mov A, #CONST_ADR_W     ;передача байта адреса микросхемы (бит R/W = 0)
        acall SEND_B            ;на шину I2C

        mov A, #HIGH_A          ;передача старшего байта адреса
        acall SEND_B            ;ячейки памяти

        mov A, #LOW_A           ;передача младшего байта адреса
        acall SEND_B            ;ячейки памяти

        mov A, #BYTE_1          ;передача первого
        acall SEND_B            ;сохраняемого байта

        mov A, #BYTE_2          ;передача второго
        acall SEND_B            ;сохраняемого байта

        acall STOP              ;формирование условия STOP

;ожидание окончания цикла записи
polling:

        acall START             ;формирование условия START

        mov A, #CONST_ADR_W     ;передача байта адреса микросхемы (бит R/W = 0)
        acall SEND_B            ;на шину I2C

        mov A, SMB0STA          ;сохранение в аккумуляторе кода состояния
                                ;модуля SMBus

        acall STOP              ;формирование условия STOP

        xrl A, #CONST_ACK       ;проверка, было ли сформировано подтверждение
        jnz polling             ;и если нет, то переход на метку polling

;чтение записанных байтов
        acall START             ;формирование условия START

        mov A, #CONST_ADR_W     ;передача байта адреса микросхемы (бит R/W = 0)
        acall SEND_B            ;на шину I2C

        mov A, #HIGH_A          ;передача старшего байта адреса
        acall SEND_B            ;ячейки памяти

        mov A, #LOW_A           ;передача младшего байта адреса
        acall SEND_B            ;ячейки памяти

        acall RESTART           ;формирование условия RESTART

        mov A, #CONST_ADR_R     ;передача байта адреса микросхемы (бит R/W = 1)
        acall SEND_B            ;на шину I2C
```

```
        acall REC_ACK           ;прием первого байта и подтверждение приема
        mov TEMP_1, A           ;сохранение принятого байта

        acall REC_NACK          ;прием второго байта и неподтверждение приема
        mov TEMP_2, A           ;сохранение принятого байта

        acall STOP              ;формирование условия STOP

loop:
        jmp loop;               ;переход на метку loop

;================================================
;подпрограммы работы с модулем SMBus
;================================================

;подпрограмма формирования условия START
START:
        setb SMB0CN.5           ;установка бита запуска формирования
m_st:
        jnb SMB0CN.3, m_st      ;ожидание окончания формирования условия START
        clr SMB0CN.5            ;сброс бита запуска
        ret                     ;возврат из подпрограммы

;подпрограмма формирования условия RESTART
RESTART:
        setb SMB0CN.5           ;установка бита запуска формирования
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к запуску формирования условия)
m_rst:
        jnb SMB0CN.3, m_rst     ;ожидание окончания формирования условия RESTART
        clr SMB0CN.5            ;сброс бита запуска
        ret                     ;возврат из подпрограммы

;подпрограмма формирования условия STOP
STOP:
        setb SMB0CN.4           ;установка флага запуска формирования
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к запуску формирования условия)
        ret                     ;возврат из подпрограммы

;подпрограмма передачи байта
SEND_B:
        mov SMB0DAT, A          ;загрузка в регистр SMB0DAT отправляемого байта
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к запуску процесса передачи байта)
m_sp:
        jnb SMB0CN.3, m_sp      ;ожидание окончания передачи байта
        ret                     ;возврат из подпрограммы

;подпрограмма приема байта с формированием подтверждения
REC_ACK:
        setb SMB0CN.2           ;установка бита AA (после приема байта ведущий
                                ;сформирует подтверждение)
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
```

```
                                ;приводит к запуску процесса приема байта)
m_ra:
        jnb SMB0CN.3, m_ra      ;ожидание окончания приема байта
        mov A, SMB0DAT          ;сохранение принятого байта в аккумуляторе
        ret                     ;возврат из подпрограммы

;подпрограмма приема байта без формирования подтверждения
REC_NACK:
        clr SMB0CN.2            ;сброс бита AA (после приема байта ведущий
                                ;не сформирует подтверждение)
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к запуску процесса приема байта)
m_rna:
        jnb SMB0CN.3, m_rna     ;ожидание окончания приема байта
        mov A, SMB0DAT          ;сохранение принятого байта в аккумуляторе
        ret                     ;возврат из подпрограммы
        END
```

§ 4.4.2. Пример программы ведомого микроконтроллера

Рис. 4.26 Алгоритм программы ведомого микроконтроллера (начало)

В качестве примера рассмотрим программу, которая будет осуществлять обмен информацией с ведущим устройством так, как это делала бы микросхема EEPROM памяти 24LC64. Предполагаем, что ведущее устройство работает по алгоритму, описанному в § 4.4.1, тогда ведомое устройство должно реализовывать алгоритм, приведенный на рис. 4.26 и 4.27.

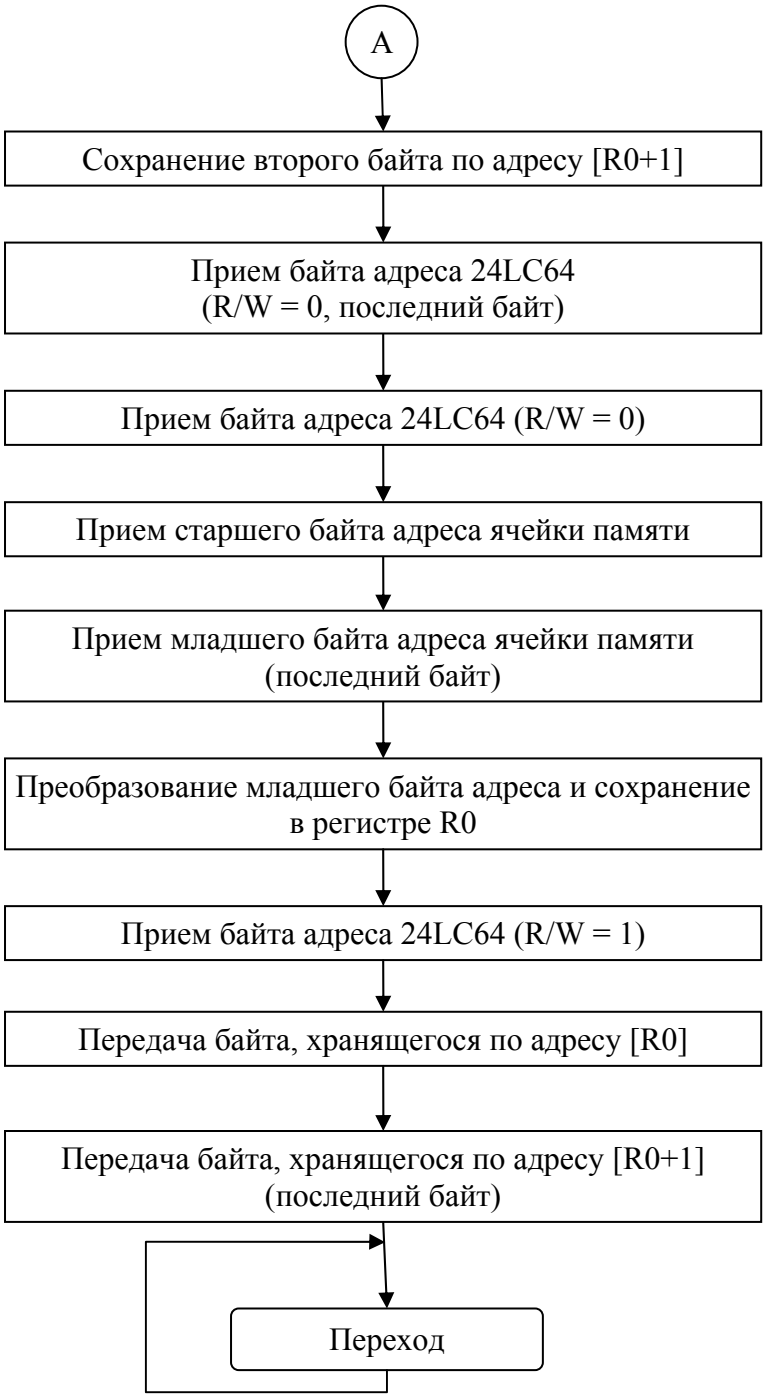


Рис. 4.27 Алгоритм программы ведомого микроконтроллера (окончание)

Алгоритмы подпрограмм, осуществляющих прием и передачу байта, приведены в § 4.3.

Ниже приведена программа, реализующая описанный алгоритм. Программа написана для микроконтроллера C8051F060. Предполагается, что в качестве линий SDA и SCL используются соответственно линии 5 и 6 порта P0 микроконтроллера.

```
$NOMOD51
$include (c8051f060.inc)

;-------------------------------------------------------------------------------
;инициализация
;-------------------------------------------------------------------------------

;запрещение сторожевого таймера
        mov WDTCN, #0DEh
        mov WDTCN, #0ADh

;настройка модуля SPI на трехпроводной режим работы
        mov SFRPAGE, #00h       ;выбор страницы SFR с регистром SPI0CN
        mov SPI0CN, #02h        ;настройка модуля SPI на трехпроводной режим работы
                                ;чтобы обеспечить поключение линий SDA и SCL
                                ;к линиям P0.5 и P0.6
;настройка и включение матрицы
        mov SFRPAGE, #0Fh       ;выбор страницы SFR с регистрами XBR0 и XBR2
        mov XBR0, #07h          ;подключение входов/выходов
                                ;модулей UART0, SPI и SMBus
        mov XBR2, #40h          ;включение матрицы

;настройка модуля SMBus
        mov SFRPAGE, #00h       ;выбор страницы SFR с регистрами модуля SMBus
        setb SMB0CN.2           ;установка бита AA (формировать подтверждение)
        mov SMB0ADR, #0A8h      ;задание адреса ведомого (без поддержки адреса общего
                                вызова)
        clr SMB0CN.3            ;сброс флага прерывания от модуля SMBus
        setb SMB0CN.6           ;включение модуля SMBus

;-------------------------------------------------------------------------------
;основная программа
;-------------------------------------------------------------------------------

        acall REC_B             ;прием байта адреса (R/W = 0)

        acall REC_B             ;прием старшего байта адреса
                                ;ячейки памяти (байт игнорируется)

        acall REC_B             ;прием младшего байта адреса ячейки памяти
        anl A, #07Fh            ;обнуление старшего бита (чтобы не выйти из
                                ;допустимого диапазона адресов)
        add A, #020h            ;прибавление смещения
        mov R0, A               ;сохранение преобразованного байта адреса в регистре R0
```

```
        acall REC_B             ;прием первого сохраняемого байта
        mov @R0, A              ;сохранение принятого байта по адресу [R0]

        acall REC_LB            ;прием второго сохраняемого байта (последний байт)
        inc R0                  ;инкремент R0
        mov @R0, A              ;сохранение второго байта по адресу [R0]

        acall REC_LB            ;прием байта адреса (R/W = 0, последний байт)


        acall REC_B             ;прием байта адреса (R/W = 0)

        acall REC_B             ;прием старшего байта адреса
                                ;ячейки памяти (байт игнорируется)

        acall REC_LB            ;прием младшего байта адреса
                                ;ячейки памяти (последний байт)
        anl A, #07Fh            ;обнуление старшего бита (чтобы не выйти из
                                ;допустимого диапазона адресов)
        add A, #020h            ;прибавление смещения
        mov R0, A               ;сохранение преобразованного байта адреса в регистре R0


        acall REC_B             ;прием байта адреса (R/W = 1)

        mov A, @R0              ;передача байта
        acall SEND_B            ;хранящегося по адресу [R0]

        inc R0                  ;вычисление адреса следующего байта
        mov A, @R0              ;передача байта
        acall SEND_LB           ;хранящегося по адресу [R0] (последний байт)


loop:
        jmp loop                ;переход на метку loop

;=========================================
;подпрограммы работы с модулем SMBus
;=========================================

;подпрограмма приема байта
REC_B:
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к «отпусканию» лини SCL)
m_rb:
        jnb SMB0CN.3, m_rb      ;ожидание окончания приема байта
        mov A, SMB0DAT          ;сохранение принятого байта в аккумуляторе
        ret                     ;возврат из подпрограммы

;подпрограмма приема последнего байта
REC_LB:
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
```

```
                                ;приводит к «отпусканию» линии SCL)
m_rlb:
        jnb SMB0CN.3, m_rlb     ;ожидание окончания приема байта
        mov A, SMB0DAT          ;сохранение принятого байта в аккумулятор

        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к «отпусканию» линии SCL)
m_rlb2:
        jnb SMB0CN.3, m_rlb2    ;ожидание окончания формирования условия STOP
        clr SMB0CN.3            ;сброс флага прерывания от модуля
        ret                     ;возврат из подпрограммы

;подпрограмма передачи байта
SEND_B:
        mov SMB0DAT, A          ;загрузка в регистр SMB0DAT отправляемого байта
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к «отпусканию» линии SCL)
m_sb:
        jnb SMB0CN.3, m_sb      ;ожидание окончания передачи байта
        ret                     ;возврат из подпрограммы

;подпрограмма передачи последнего байта
SEND_LB:
        mov SMB0DAT, A          ;загрузка в регистр SMB0DAT отправляемого байта
        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к «отпусканию» линии SCL)
m_slb:
        jnb SMB0CN.3, m_slb     ;ожидание окончания передачи байта

        clr SMB0CN.3            ;сброс флага прерывания от модуля (сброс флага
                                ;приводит к «отпусканию» линии SCL)
m_slb2:
        jnb SMB0CN.3, m_slb2    ;ожидание окончания формирования условия STOP
        clr SMB0CN.3            ;сброс флага прерывания от модуля
        ret                     ;возврат из подпрограммы
        END
```

ГЛАВА 5. ШИНА I2C В МИКРОКОНТРОЛЛЕРАХ СЕМЕЙСТВА AVR

§ 5.1. Описание микроконтроллера и среды разработки

§ 5.1.1. Общие сведения об ATmega16

ATmega16 – это 8-битный микроконтроллер, выполненный по CMOS технологии. Благодаря применению гарвардской архитектуры и одноуровневого конвейера команд производительность МК при выполнении большинства команд достигает 1 MIPS на 1 МГц.

Основные характеристики:

- производительность до 16 MIPS при частоте 16 МГц;
- аппаратный умножитель;
- 16 Кбайт flash-памяти;
- 1 Кбайт внутренней SRAM;
- 512 байт EEPROM;
- JTAG
- два 8-битных таймера/счетчика с предделителем и режимом сравнения;
- 16-битный таймер/счетчик с предделителем, режимом сравнения/захвата;
- часы реального времени;
- 8-канальный 10-битный АЦП;
- I2C;
- USART;
- SPI;
- встроенный аналоговый компаратор;
- тактовая частота 0…16 МГц;
- напряжение питания 4.5…5.5 В.

Память программ

Память программ микроконтроллера составляет 16 Кбайт организованных как 8К x 16. Такая организация связана с тем, что все инструкции имеют размер 16 либо 32 бита. Для обеспечения большей гибкости память программ разделена на две секции: секция загрузчика и секция прикладной программы.

Память программ гарантированно выдерживает не менее 10 000 циклов записи/стирания. Также имеется возможность изменения содержимого памяти программ во время работы МК (самопрограммирование).

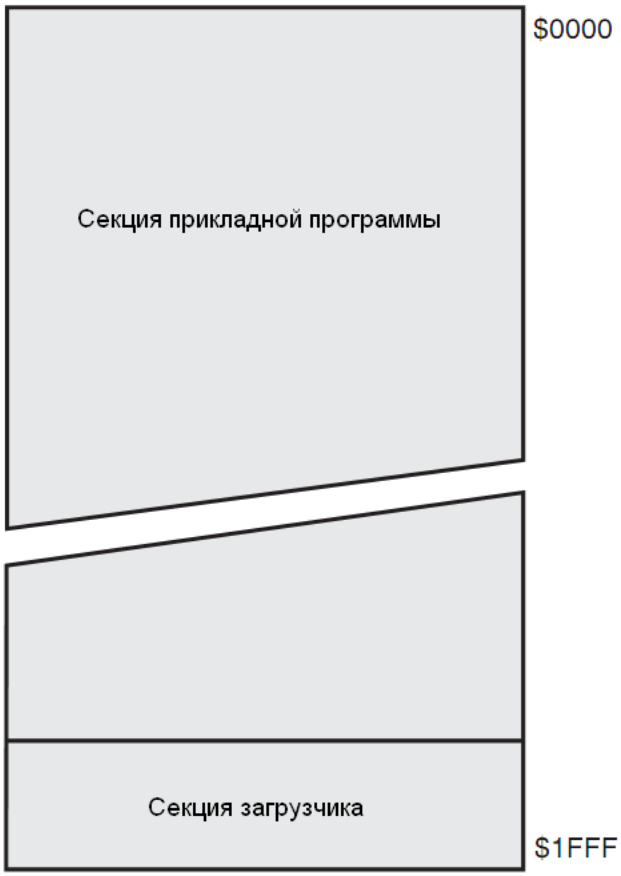


Рис. 5.1. Распределение памяти программ микроконтроллера ATmega16

Регистры общего назначения

На рис. 5.2 представлена структура регистрового файла микроконтроллера.

Большинство инструкций, взаимодействующих с регистрами общего назначения, выполняются за один такт. К каждому из регистров регистрового файла также можно обратиться косвенно, используя приведенные адреса.

Регистры ввода/вывода

Все порты ввода/вывода и регистры периферийных устройств расположены в области регистров ввода/вывода. Данная область доступна с помощью команд IN и OUT. К регистрам с адресами $00…$1F могут быть применены команды прямой побитной адресации SBI и CBI. Значения битов этих же регистров могут быть проверены с помощью инструкций SBIC и SBIC.

Память данных

Распределение адресов памяти данных приведено на рис. 5.3.

Рис. 5.2. Распределение регистров общего назначения микроконтроллера ATmega16

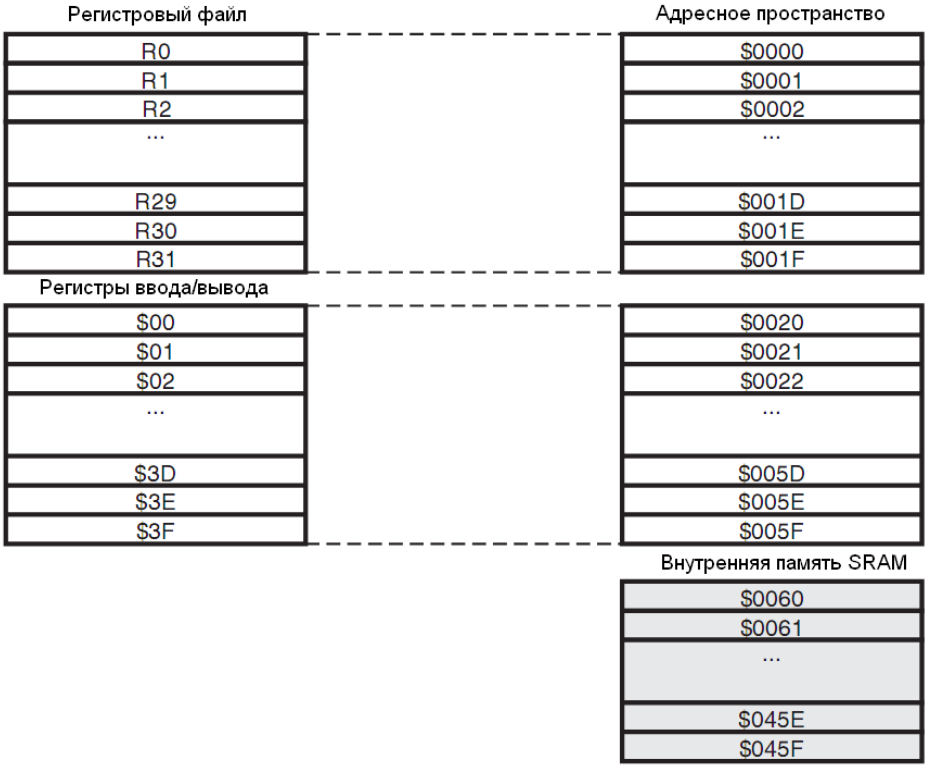


Рис. 5.3. Распределение памяти данных микроконтроллера ATmega16

Каждый из 32 регистров общего назначения, 64 регистров ввода/вывода и 1024 байт памяти данных SRAM может быть адресован любым из пяти возможных способов:

1. Прямая адресация
2. Косвенная адресация со смещением
3. Косвенная адресация
4. Косвенная адресация с преддекрементом
5. Косвенная адресация с постинкрементом

§ 5.1.2. Краткое описание отладочных средств

Производитель микроконтроллеров AVR – компания ATMEL – выпустила отладочный пакет AVR Studio для работы с этими контроллерами. В состав AVR Studio входят:

- Project management tool – менеджер проекта;
- Source file editor – текстовый редактор;
- AVRASM – ассемблер;
- Chip simulator – симулятор;
- In-circuit emulator interface – эмулятор.

Работа в среде AVR STUDIO 4.0

Создание нового проекта

1. Из меню выберите Project > New project.
2. Введите имя проекта, имя файла ASM и укажите его местоположение.
3. Поставьте галочки «Create initial File» и «Create Folder».
   Нажмите кнопку NEXT.

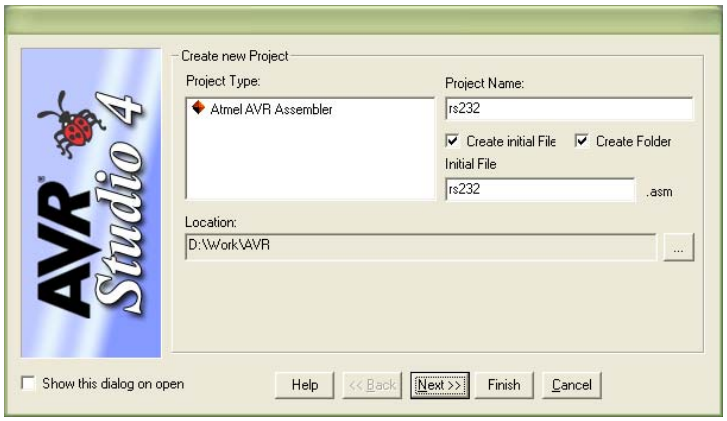


Рис. 5.4. Создание нового проекта

4. В появившемся далее окне выберите Debug Platform = Simulator (слева) и нужный контроллер – ATmega16 (справа). Нажмите кнопку «Finish».

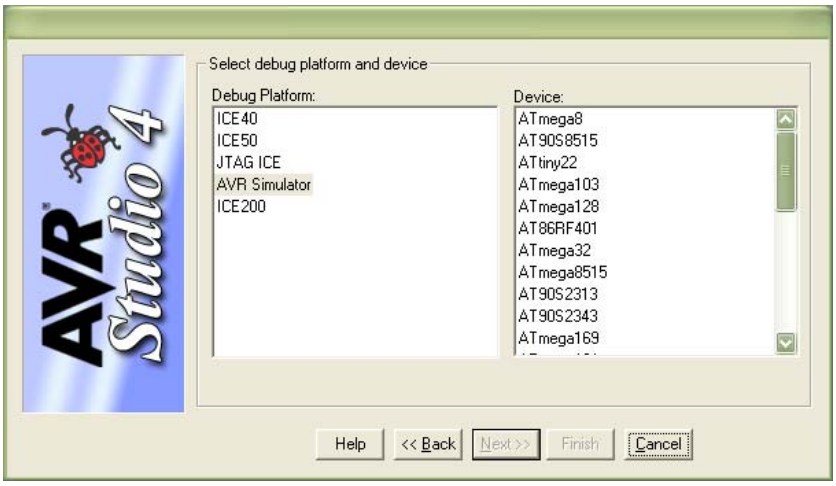


Рис. 5.5. Выбор отладочной платформы

Проект создан, можно приступать непосредственно к написанию программы.

Основные элементы среды AVR Studio

Элементы среды AVR Studio (см. рис. 5.6):

1. Окно проекта. Здесь показаны файлы, включенные в проект.
2. Вкладка для открытия окна проекта.
3. Вкладка для открытия окна просмотра регистров.
4. Вкладка для открытия окна помощи.
5. Окно созданного ассемблерного файла.
6. Окно состояния (AVR Studio сообщает здесь о своих действиях).

Первой строкой программы должно быть, подключение файла .inc для микроконтроллера, где всем используемым именам регистров присвоены их адреса, а названиям битов – их номера:

```
.include «m16def.inc»
```

Затем нужно инициализировать стек:

```
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
```

Работа с симулятором

После написания программы ее необходимо оттранслировать (меню Project > Build) и запустить в симуляторе (меню Debug > Start Debugging). Если в программе имеются ошибки, об этом будет ука-

зано в окне состояния. Если грубых ошибок нет, то запускается пошаговая симуляция работы микроконтроллера ATmega16 (Debug). Следующая команда не выполняется, пока не будет нажата кнопка Step Into (см. на панели инструментов). Текущие значения всех регистров контроллера отображаются в окне просмотра регистров. Можно запустить и автоматическое исполнение инструкций (кнопка Run). В таком случае выполнение инструкций происходит автоматически; программа остановится либо по нажатию кнопки Stop, либо если была достигнута строка, отмеченная точкой останова (breakpoint). Точку останова можно поставить, кликнув в программе на инструкцию, на которой нужно остановиться. Выполнение программы после точки останова продолжается только по команде пользователя (например, Run).

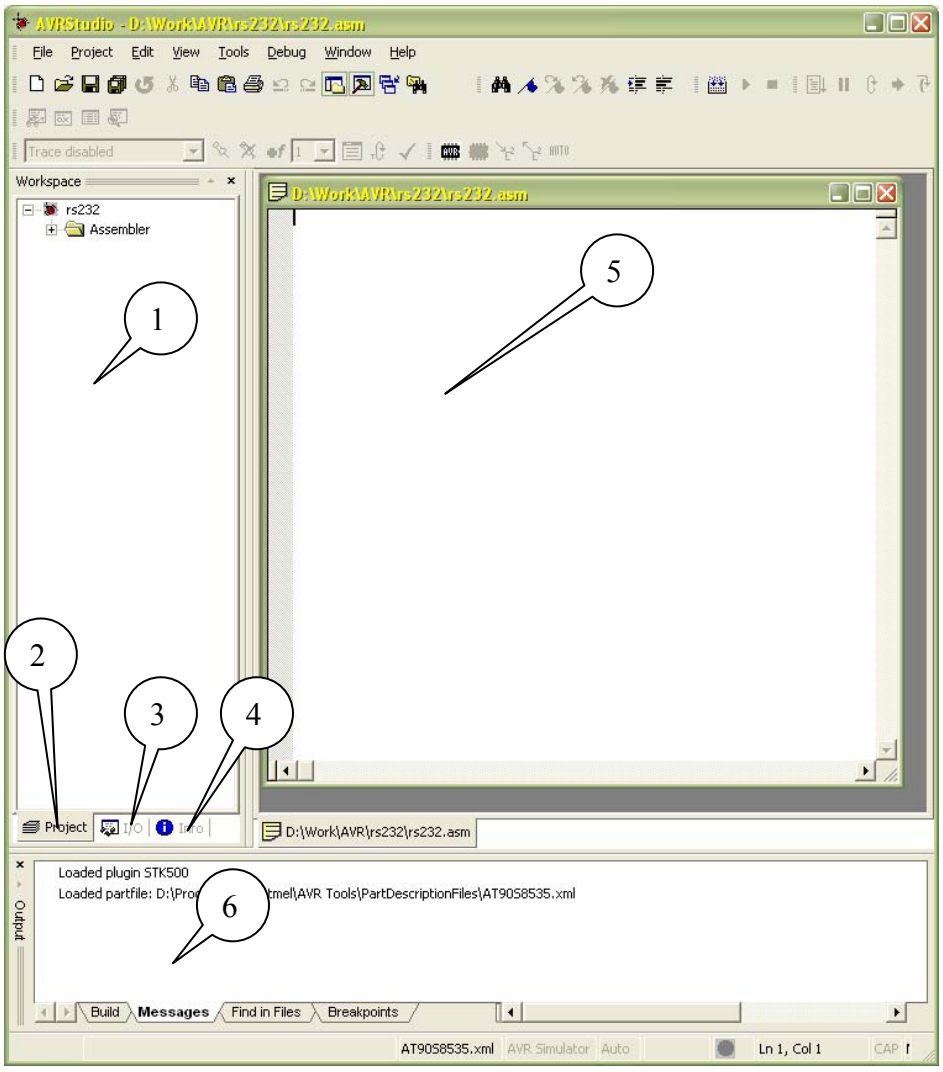


Рис. 5.6. Основное окно среды AVR Studio

Система команд

Таблица 5.1

Система команд микроконтроллера ATmega16

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ					
ADD	Rd, Rr	Сложение двух регистров без переноса	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Сложение двух регистров с переносом	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Сложение регистровой пары с константой	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Вычитание двух регистров	Rd ← Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Вычитание константы из регистра	Rd ← Rd – K	Z,C,N,V,H	1
SBC	Rd, Rr	Вычитание двух регистров с заемом	Rd ← Rd – Rr – C	Z,C,N,V,H	1
SBCI	Rd, K	Вычитание константы с заемом из регистра	Rd ← Rd – K – C	Z,C,N,V,H	1
SBIW	Rdl,K	Вычитание константы из регистровой пары	Rdh:Rdl ← Rdh:Rdl – K	Z,C,N,V,S	2
AND	Rd, Rr	«Логическое И» двух регистров	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	«Логическое И» регистра и константы	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	«Логическое ИЛИ» двух регистров	Rd ← Rd ∨ Rr	Z,N,V	1
ORI	Rd, K	«Логическое ИЛИ» регистра и константы	Rd ← Rd ∨ K	Z,N,V	1
EOR	Rd, Rr	«Исключающее ИЛИ» двух регистров	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	Перевод числа в обратный код	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Перевод числа в дополнительный код	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Установка бита(-ов) в регистре	Rd ← Rd ∨ K	Z,N,V	1
CBR	Rd,K	Сброс бита(-ов) в регистре	Rd ← Rd • ($FF – K)	Z,N,V	1
INC	Rd	Инкремент	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Декремент	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Проверка на отрицательное или нулевое значение	Rd ← Rd • Rd	Z,N,V	1

Продолжение табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
CLR	Rd	Очистка регистр	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Установка всех битов регистра	Rd ← $FF	–	1
MUL	Rd, Rr	Умножение беззнаковых чисел	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Умножение знаковых чисел	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Умножение знакового числа на беззнаковое	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Умножение дробных беззнаковых чисел	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Умножение дробных знаковых чисел	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Умножение дробного знакового числа на беззнаковое	R1:R0 ← (Rd x Rr) << 1	Z,C	2
КОМАНДЫ ПЕРЕХОДА					
RJMP	k	Относительный безусловный переход	PC ← PC + k + 1	–	2
IJMP		Косвенный безусловный переход (Z)	PC ← Z	–	2
JMP	k	Прямой переход	PC ← k	–	3
RCALL	k	Относительный вызов подпрограммы	PC ← PC + k + 1	–	3
ICALL		Косвенный вызов подпрограммы (Z)	PC ← Z	–	3
CALL	k	Прямой вызов подпрограммы	PC ← k	–	4
RET		Возврат из подпрограммы	PC ← STACK	–	4
RETI		Возврат из подпрограммы обратотки прерывания	PC ← STACK	I	4
CPSE	Rd,Rr	Сравнить, пропустить след. команду, если равно	Если (Rd = Rr) тогда PC ← PC + 2 или 3	–	1 / 2 / 3
CP	Rd,Rr	Сравнить	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Сравнить с учетом переноса	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Сравнить регистр с константой	Rd – K	Z, N,V,C,H	1

Продолжение табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
SBRC	Rr, b	Пропустить, если бит в регистре сброшен	Если (Rr(b)=0) тогда PC ← PC + 2 или 3	–	1 / 2 / 3
SBRS	Rr, b	Пропустить, если бит в регистре установлен	Если (Rr(b)=1) тогда PC ← PC + 2 или 3	–	1 / 2 / 3
SBIC	P, b	Пропустить, если бит в регистре ввода/вывода сброшен	Если (P(b)=0) тогда PC ← PC + 2 или 3	–	1 / 2 / 3
SBIS	P, b	Пропустить, если бит в регистре ввода/вывода установлен	Если (P(b)=1) тогда PC ← PC + 2 или 3	–	1 / 2 / 3
BRBS	s, k	Переход, если флаг STATUS установлен	Если (SREG(s) = 1) тогда PC←PC + k + 1	–	1 / 2
BRBC	s, k	Переход, если флаг STATUS сброшен	Если (SREG(s) = 0) тогда PC←PC + k + 1	–	1 / 2
BREQ	k	Переход, если равно	Если (Z = 1) тогда PC ← PC + k + 1	–	1 / 2
BRNE	k	Переход, если не равно	Если (Z = 0) тогда PC ← PC + k + 1	–	1 / 2
BRCS	k	Переход, если установлен флаг переноса	Если (C = 1) тогда PC ← PC + k + 1	–	1 / 2
BRCC	k	Переход, если сброшен флаг переноса	Если (C = 0) тогда PC ← PC + k + 1	–	1 / 2
BRSH	k	Переход, если больше или равно (беззнаковое)	Если (C = 0) тогда PC ← PC + k + 1	–	1 / 2
BRLO	k	Переход, если меньше (беззнаковое)	Если (C = 1) тогда PC ← PC + k + 1	–	1 / 2

Продолжение табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
BRMI	k	Переход, если отрицательное	Если (N = 1) тогда PC ← PC + k + 1	–	1 / 2
BRPL	k	Переход, если положительное	Если (N = 0) тогда PC ← PC + k + 1	–	1 / 2
BRGE	k	Переход, если больше или равно (знаковое)	Если (N ⊕ V= 0) тогда PC ← PC + k + 1	–	1 / 2
BRLT	k	Переход, если меньше нуля (знаковое)	Если (N ⊕ V= 1) тогда PC ← PC + k + 1	–	1 / 2
BRHS	k	Переход, если флаг половинного переноса установлен	Если (H = 1) тогда PC ← PC + k + 1	–	1 / 2
BRHC	k	Переход, если флаг половинного переноса сброшен	Если (H = 0) тогда PC ← PC + k + 1	–	1 / 2
BRTS	k	Переход, если флаг T установлен	Если (T = 1) тогда PC ← PC + k + 1	–	1 / 2
BRTC	k	Переход, если флаг T сброшен	Если (T = 0) тогда PC ← PC + k + 1	–	1 / 2
BRVS	k	Переход, если флаг переполнения установлен	Если (V = 1) тогда PC ← PC + k + 1	–	1 / 2
BRVC	k	Переход, если флаг переполнения сброшен	Если (V = 0) тогда PC ← PC + k + 1	–	1 / 2
BRIE	k	Переход, если прерывания разрешены	Если (I = 1) тогда PC ← PC + k + 1	–	1 / 2
BRID	k	Переход, если прерывания запрещены	Если (I = 0) тогда PC ← PC + k + 1	–	1 / 2

Продолжение табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ					
MOV	Rd, Rr	Пересылка из регистра в регистр	Rd ← Rr	–	1
MOVW	Rd, Rr	Пересылка регистровой пары в регистровую пару	Rd+1:Rd ← Rr+1:Rr	–	1
LDI	Rd, K	Загрузка константы в регистр	Rd ← K	–	1
LD	Rd, X	Косвенное чтение	Rd ← (X)	–	2
LD	Rd, X+	Косвенное чтение с постинкрементом	Rd ← (X), X ← X + 1	–	2
LD	Rd, – X	Косвенное чтение с преддекрементом	X ← X – 1, Rd ← (X)	–	2
LD	Rd, Y	Косвенное чтение	Rd ← (Y)	–	2
LD	Rd, Y+	Косвенное чтение с постинкрементом	Rd ← (Y), Y ← Y + 1	–	2
LD	Rd, – Y	Косвенное чтение с преддекрементом	Y ← Y – 1, Rd ← (Y)	–	2
LDD	Rd,Y+q	Косвенное чтение со смещением	Rd ← (Y + q)	–	2
LD	Rd, Z	Косвенное чтение	Rd ← (Z)	–	2
LD	Rd, Z+	Косвенное чтение с постинкрементом	Rd ← (Z), Z ← Z+1	–	2
LD	Rd, –Z	Косвенное чтение с преддекрементом	Z ← Z – 1, Rd ← (Z)	–	2
LDD	Rd, Z+q	Косвенное чтение со смещением	Rd ← (Z + q)	–	2
LDS	Rd, k	Прямое чтение из памяти	Rd ← (k)	–	2
ST	X, Rr	Косвенная запись	(X) ← Rr	–	2
ST	X+, Rr	Косвенная запись с постинкрементом	(X) ← Rr, X ← X + 1	–	2
ST	–X, Rr	Косвенная запись с преддекрементом	X ← X – 1, (X) ← Rr	–	2
ST	Y, Rr	Косвенная запись	(Y) ← Rr	–	2
ST	Y+, Rr	Косвенная запись с постинкрементом	(Y) ← Rr, Y ← Y + 1	–	2
ST	–Y, Rr	Косвенная запись с преддекрементом	Y ← Y – 1, (Y) ← Rr	–	2
STD	Y+q,Rr	Косвенная запись со смещением	(Y + q) ← Rr	–	2
ST	Z, Rr	Косвенная запись	(Z) ← Rr	–	2

Продолжение табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
ST	Z+, Rr	Косвенная запись с постинкрементом	(Z) ← Rr, Z ← Z + 1	–	2
ST	–Z, Rr	Косвенная запись с преддекрементом	Z ← Z – 1, (Z) ← Rr	–	2
STD	Z+q,Rr	Косвенная запись со смещением	(Z + q) ← Rr	–	2
STS	k, Rr	Прямая запись в память	(k) ← Rr	–	2
LPM		Загрузка из памяти программ	R0 ← (Z)	–	3
LPM	Rd, Z	Загрузка из памяти программ	Rd ← (Z)	–	3
LPM	Rd, Z+	Загрузка из памяти программ с постинкрементом	Rd ← (Z), Z ← Z+1	–	3
SPM		Запись в память программ	(Z) ← R1:R0	–	–
IN	Rd, P	Чтение из регистра ввода/вывода	Rd ← P	–	1
OUT	P, Rr	Запись в регистр ввода/вывода	P ← Rr	–	1
PUSH	Rr	Сохранение байта в стеке	STACK ← Rr	–	2
POP	Rd	Извлечение байта из стека	Rd ← STACK	–	2
КОМАНДЫ ОПЕРАЦИЙ С БИТАМИ					
SBI	P,b	Установить бит в регистре ввода/вывода	I/O(P,b) ← 1	–	2
CBI	P,b	Сбросить бит в регистре ввода/вывода	I/O(P,b) ← 0	–	2
LSL	Rd	Логический сдвиг влево	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Логический сдвиг вправо	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Циклический сдвиг влево через перенос	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z,C,N,V	1

Продолжение табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
ROR	Rd	Циклический сдвиг вправо через перенос	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z,C,N,V	1
ASR	Rd	Арифметический сдвиг вправо	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Обмен местами тетрад	Rd(3..0) ← Rd(7..4), Rd(7..4) ← Rd(3..0)	–	1
BSET	s	Установка флага	SREG(s) ← 1	SREG(s)	1
BCLR	s	Сброс флага	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Копирование бита регистра в T	T ← Rr(b)	T	1
BLD	Rd, b	Копирование T в бит регистра	Rd(b) ← T	–	1
SEC		Установка флага переноса	C ← 1	C	1
CLC		Сброс флага переноса	C ← 0	C	1
SEN		Установка флаг отрицательного результата	N ← 1	N	1
CLN		Сброс флага отрицательного результата	N ← 0	N	1
SEZ		Установка флага нулевого результата	Z ← 1	Z	1
CLZ		Сброс флага нулевого результата	Z ← 0	Z	1
SEI		Глобальное разрешение прерываний	I ← 1	I	1
CLI		Глобальное запрещение прерываний	I ← 0	I	1
SES		Установка флага знака	S ← 1	S	1
CLS		Сброс флага знака	S ← 0	S	1
SEV		Установка флага переполнения	V ← 1	V	1
CLV		Сброс флага переполнения	V ← 0	V	1
SET		Установка бита T	T ← 1	T	1
CLT		Сброс бита T	T ← 0	T	1

Окончание табл. 5.1

Мнемоника	Операнды	Описание	Операция	Флаги	Такты
SEH		Установка флага половинного переноса	H ← 1	H	1
CLH		Сброс флага половинного переноса	H ← 0	H	1
КОМАНДЫ УПРАВЛЕНИЯ MCU					
NOP		холостая команда	–	–	1
SLEEP		Переход в «спящий» режим	См. [6]	–	1
WDR		Сброс сторожевого таймера	См. [6]	–	1
BREAK		Останов	Используется только внутрисхемным отладчиком	–	–

§ 5.2. Модуль TWI

§ 5.2.1. Устройство модуля TWI

Модуль TWI состоит из нескольких подмодулей (см. рис. 5.7). Все регистры, выделенные жирной линией, доступны через шину данных микроконтроллера.

Выводы SCL и SDA

Данные выводы связывают микроконтроллер с остальными устройствами в системе. Драйверы выходов содержат ограничитель скорости изменения фронтов. Входные каскады содержат блок подавления помех, задача которого состоит в игнорировании импульсов длительностью менее 50 нс. Обратите внимание, что к каждой из этих линий можно подключить внутренний подтягивающий резистор путем установки разрядов PORTC.0 (SCL), PORTC.1 (SDA). Использование встроенных подтягивающих резисторов в ряде случаев позволяет отказаться от применения внешних.

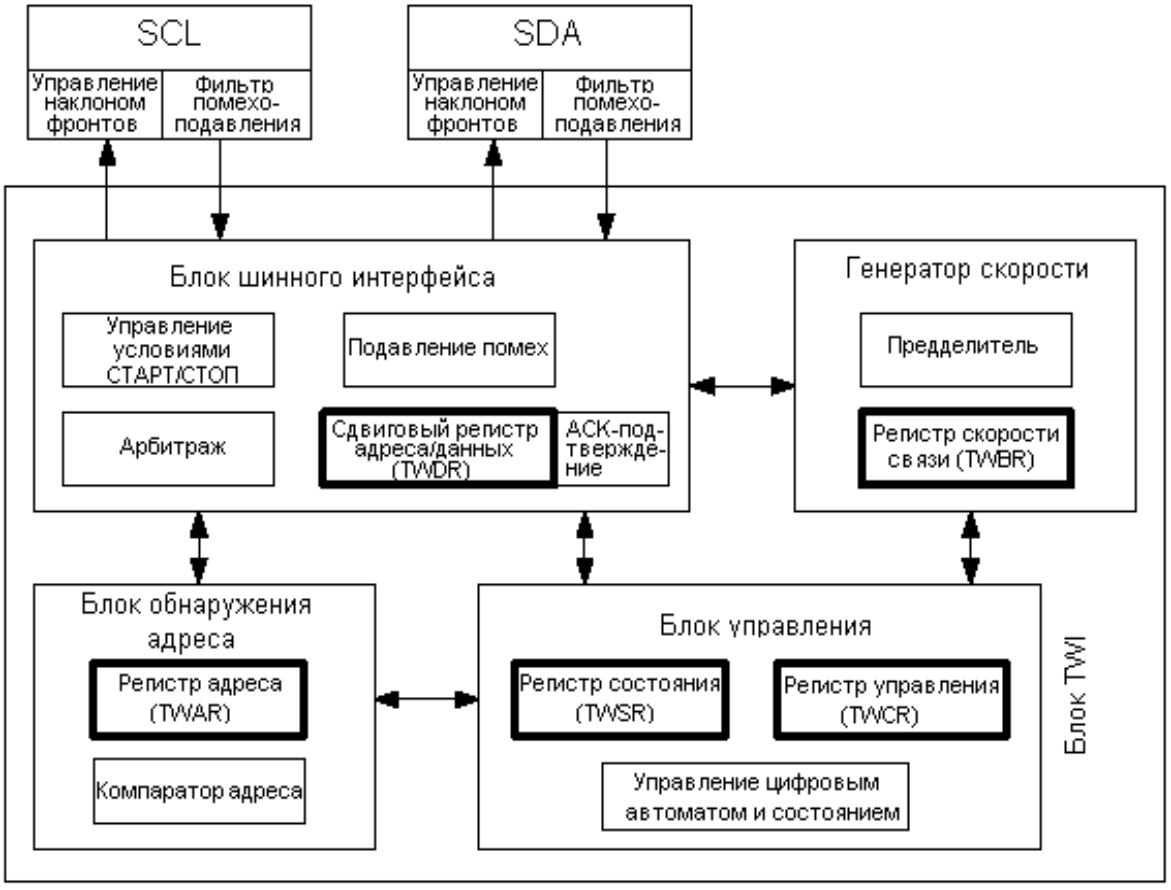


Рис. 5.7. Функциональная схема модуля TWI

Блок генератора скорости связи

Данный блок управляет периодом импульсов SCL в режиме ведущего устройства. Период SCL задается регистром скорости TWI (TWBR) и значением бит управления предделителем в регистре состояния TWI (TWSR). В подчиненном режиме значение скорости или установки предделителя не оказывают влияния на работу, но тактовая частота подчиненного устройства должна быть минимум в 16 раз выше частоты SCL. Обратите внимание, что подчиненные могут продлевать длительность низкого уровня на линии SCL, тем самым, уменьшая среднюю частоту синхронизации шины I2C. Частота SCL определяется в соответствии со следующим выражением:

$$f_{SCL} = \frac{f_{CPU}}{16 + 2 \cdot (TWBR) \cdot 4^{TWPS}},$$

где TWBR – значение регистра скорости TWI; TWPS – значение бит предделителя в регистре состояния TWI.

Примечание. Значение регистра TWBR должно быть равно или больше 10, если TWI работает в ведущем режиме. Если TWBR меньше 10, то ведущий может генерировать некорректные состояния на линиях SDA и SCL.

Блок шинного интерфейса

Данный блок содержит сдвиговый регистр адреса и данных (TWDR), контроллер условий START/STOP и схему арбитража. TWDR содержит передаваемый байт адреса или данных, или принятый байт адреса или данных. Помимо 8-разр. регистра TWDR в состав блока шинного интерфейса также входит регистр, хранящий значение передаваемого или принятого бита ACK (NACK). К данному регистру нет прямого доступа со стороны программного обеспечения. Однако во время приема он может устанавливаться или сбрасываться путем манипуляций с регистром управления TWI (TWCR). В режиме передатчика значение принятого бита ACK (NACK) можно определить по значению регистра TWSR.

Контроллер условий START/STOP отвечает за генерацию и детекцию условий START, RESTART и STOP. Контроллер условий START/STOP позволяет обнаружить условия START и STOP, даже если микроконтроллер находится в одном из спящих режимов. Этим обеспечивается возможность пробуждения микроконтроллера по запросу ведущего шины.

Если TWI инициировал передачу в качестве ведущего, то схема арбитража непрерывно контролирует передачу, определяя возможность

дальнейшей передачи. Если TWI теряет арбитраж, то блок формирует соответствующий сигнал блоку управления, который выполняет адекватные действия и генерирует соответствующий код состояния.

Блок обнаружения адреса

Блок обнаружения адреса проверяет, равен ли принятый адрес значению 7-разр. адреса из регистра TWAR. Если установлен бит разрешения обнаружения общего вызова TWGCE в регистре TWAR, то все входящие адресные биты будут дополнительно сравниваться с адресом общего вызова. При совпадении адреса подается сигнал блоку управления, что позволяет выполнить ему необходимые действия. От установки регистра TWCR зависит, будет ли формироваться подтверждение адреса. Блок обнаружения адреса способен функционировать даже, когда микроконтроллер переведен в спящий режим, тем самым, позволяя возобновить нормальную работу микроконтроллера по запросу мастера шины.

Блок управления

Блок управления наблюдает за шиной I2C и генерирует отклики в соответствии с установками регистра управления TWI (TWCR). Если на шине возникает событие, которое требует внимания со стороны программы, то устанавливается флаг прерывания TWINT. Следующим тактом обновляется содержимое регистра статуса TWI – TWSR, в котором будет записан код, идентифицирующий возникшее событие. Даная информация хранится в TWSR только тогда, когда установлен флаг прерывания TWI. Остальное время в регистре TWSR содержится специальный код состояния, который информирует о том, что нет информации о состоянии TWI. До тех пор пока установлен флаг TWINT, линия SCL остается в низком состоянии. Это позволяет программе завершить все задачи перед продолжением сеанса связи.

Флаг TWINT устанавливается в следующих ситуациях:

- после передачи условия START/RESTART;
- после передачи SLA+R/W;
- после передачи адресного байта;
- после потери арбитража;
- после того как TWI адресован собственным подчиненным адресом или общим вызовом;
- после приема байта данных;
- после приема условия STOP или RESTART в режиме подчиненной адресации;
- после возникновения ошибки по причине некорректного условия START или STOP.

§ 5.2.2. Описание регистров TWI

Регистр скорости – TWBR

Бит	7	6	5	4	3	2	1	0	
	TWBR 7	TWBR 6	TWBR 5	TWBR 4	TWBR 3	TWBR 2	TWBR 1	TWBR 0	TWBR
Чтение / запись	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	
Значение после сброса МК	0	0	0	0	0	0	0	0	

- Биты 7..0 – Биты регистра скорости связи шины TWI

TWBR задает коэффициент деления частоты генератора скорости связи. Генератор частоты скорости связи – делитель частоты, который формирует сигнал синхронизации SCL в режимах «Ведущий». В разделе (§ 5.2.1 Устройство модуля TWI) показана методика вычисления скоростей связи.

Регистр управления – TWCR

Бит	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Чтение / запись	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	
Значение после сброса МК	0	0	0	0	0	0	0	0	

Регистр TWCR предназначен для управления работой TWI. Он используется для разрешения работы TWI, для инициации сеанса связи ведущего путем генерации условия START на шине, для генерации подтверждения приема, для генерации условия STOP и для останова шины во время записи в регистр TWDR. Он также сигнализирует о попытке ошибочной записи в регистр TWDR, когда доступ к нему был запрещен.

- Бит 7 – TWINT: Флаг прерывания TWI

Данный бит устанавливается аппаратно, если TWI завершает текущее задание и ожидает реакции программы. Если бит I в SREG и бит TWIE в TWCR установлены, то микроконтроллер переходит на вектор прерывания TWI. Линия SCL остается в низком состоянии, пока установлен флаг TWINT. Флаг TWINT сбрасывается программно путем записи в него логической 1. Обратите внимание, что данный флаг не сбрасывается автоматически при переходе на вектор прерывания. Также нужно учесть, что

очистка данного флага приводит к возобновлению работы TWI. Из этого следует, что программный сброс данного флага необходимо выполнить после завершения опроса регистров TWAR, TWSR и TWDR.

- Бит 6 – TWEA: Бит разрешения подтверждения

Бит TWEA управляет генерацией импульса подтверждения. Если в бит TWEA записана лог. 1, то импульс ACK генерируется на шине TWI, если выполняется одно из следующих условий:

1. Принят собственный подчиненный адрес.
2. Принят общий вызов, когда установлен бит TWGCE в регистре TWAR.
3. Принят байт данных в режиме ведущего приемника или подчиненного приемника.

Запись лог. 0 в бит TWEA позволяет временно «отключиться» от двухпроводной последовательной шины. Для возобновления распознавания адреса необходимо записать в данный бит лог.1.

- Бит 5 – TWSTA: Бит условия START

Программист должен установить данный бит при необходимости стать ведущим на двухпроводной последовательной шине. TWI аппаратно проверяет доступность шины и генерирует условие START, если шина свободна. Однако если шина занята, то TWI ожидает появления условия STOP, а затем генерирует новое условие START для перехвата состояния ведущего шины. TWSTA необходимо сбрасывать программно после передачи условия START.

- Бит 4 – TWSTO: Бит условия STOP

Установка бита TWSTO в режиме ведущего приводит к генерации условия STOP на двухпроводной последовательной шине. Когда условие STOP сформировано, бит TWSTO автоматически сбрасывается. В подчиненном режиме установка бита TWSTO может использоваться для выхода из состояния ошибки. В этом случае условие STOP не генерируется, но интерфейс TWI возвращается к безадресному подчиненному режиму и переводит линии SCL и SDA в высокоимпедансное состояние.

- Бит 3 – TWWC: Флаг ошибки записи

Бит TWWC устанавливается при попытке записи в регистр данных TWDR, когда бит TWINT имеет низкий уровень. Флаг сбрасывается при записи регистра TWDR, когда TWINT = 1.

- Бит 2 – TWEN: Бит разрешения работы TWI

Бит TWEN разрешает работу TWI и активизирует интерфейс TWI. Если бит TWEN установлен, то TWI берет на себя функции управления линиями ввода-вывода SCL и SDA. При этом разрешается работа ограничителей скорости изменения фронтов и помехоподавляющих фильтров. Если данный бит равен нулю, то TWI отключается и все передачи прекращаются независимо от текущего состояния работы.

- Бит 1 – Зарезервирован
  Данный бит является резервным и считывается как 0.
- Бит 0 – TWIE: Разрешение прерывания TWI

Если в данный бит записана лог. 1 и установлен бит I в регистре SREG, то запрос на прерывание TWI будет генерироваться до тех пор, пока установлен флаг TWINT.

Регистр состояния – TWSR

Бит	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Чтение / запись	Ч	Ч	Ч	Ч	Ч	Ч	Ч/З	Ч/З	
Значение после сброса МК	1	1	1	1	1	0	0	0	

- Биты 7..3 – TWS: Состояние TWI

Данные 5 бит отражают состояние логики блока TWI и двухпроводной последовательной шины. Коды состояния будут представлены далее в этом разделе.

Обратите внимание, что считываемое значение из регистра TWSR содержит и 5-разр. код состояния и 2-разр. значение, управляющее предделителем. Программист должен сбросить в 0 биты предделителя во время проверки бит состояния. В этом случае проверка состояния не будет зависеть от настройки предделителя.

- Бит 2 – Зарезервирован
  Данный бит является резервным и считывается как 0.
- Биты 1..0 – TWPS: Биты предделителя TWI

Данные биты отличаются полным доступом (чтение/запись) и позволяют управлять предделителем скорости связи.

Формула для вычисления скорости связи представлена в разделе (§ 5.2.1 Устройство модуля TWI). Значение бит TWPS1..0 используется в ней.

Регистр данных – TWDR

Бит	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Чтение / запись	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	
Значение после сброса МК	1	1	1	1	1	1	1	1	

В режиме передатчика регистр TWDR содержит следующий байт для передачи. В режиме приемника регистр TWDR содержит последний

принятый байт. Запись в регистр возможна только когда TWI не выполняет процесс сдвига данных. Такое состояние наступает, когда происходит аппаратная установка флага прерывания TWINT. Обратите внимание, что регистр данных не может инициализироваться пользователем до возникновения первого прерывания. Данные в регистре TWDR остаются стабильными пока установлен бит TWINT. Во время сдвига последовательной передачи данных одновременно происходит сдвиг для последовательного ввода. TWDR всегда содержит последний байт представленный на шине, исключая ситуацию возобновления нормальной работы микроконтроллера по прерыванию TWI. В этом случае состояние TWDR является неопределенным. В случае потери арбитража шины данные, передаваемые от ведущего к подчиненному, не теряются. Управление битом ACK происходит автоматически под управлением схемы TWI, непосредственного доступа к биту ACK нет.

- Биты 7..0 – TWD: Регистр данных шины TWI

Данные 8 бит составляют байт данных, который необходимо передать следующим, или последний принятый байт по двухпроводной последовательной шине.

Регистр подчиненного адреса – TWAR

Бит	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Чтение / запись	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	
Значение после сброса МК	1	1	1	1	1	1	1	0	

Если TWI настроен на режим подчиненного передатчика или приемника, то будет реагировать только на адрес, записанный в этот регистр (в 7 старших разрядов TWAR). В многомастерных системах регистр TWAR настраивается в том ведущем, к которому адресуются как к подчиненному другие ведущие шины.

Младший бит регистра TWAR используется для разрешения обнаружения адреса общего вызова ($00). Специальный компаратор выполняет сравнение подчиненного адреса (или адреса общего вызова) с принятым адресом. Если обнаруживается совпадение, то генерируется запрос на прерывание.

- Биты 7..1 – TWA: Регистр подчиненного адреса TWI

  Данные семь бит составляют подчиненный адрес блока TWI.
- Бит 0 – TWGCE: Бит разрешения обнаружения адреса общего вызова

После установки данного бита разрешается работа схемы обнаружения адреса общего вызова, передаваемого по шине I2C.

§ 5.2.3. Режимы передачи

TWI может работать в одном из 4-х режимов работы. Они называются: ведущий передатчик (MT), ведущий приемник (MR), подчиненный передатчик (ST) и подчиненный приемник (SR). Некоторые из этих режимов могут использоваться в рамках одного и того же приложения. Например, TWI может использовать режим MT для записи данных в 2-проводную последовательную EEPROM память, а режим MR для считывания данных из EEPROM. Если в системе имеются другие ведущие (мастера), один из которых передает данные, то у остальных используется режим SR. Какой из режимов должен использоваться определяется программно.

Передача TWI приостанавливается до тех пор, пока программно не будет сброшен флаг TWINT.

После установки флага TWINT по значению кода состояния из регистра TWSR определяется, какое действие выполнить программе. В табл. 5.2–5.5 представлена информация о том, какие программные действия должны быть предприняты при различных значениях кода состояния. Обратите внимание, что в таблицах биты предделителя сброшены.

Режим ведущего передатчика

В режиме ведущего передатчика байты данных передаются подчиненному приемнику. Для ввода режима ведущего необходимо передать условие START. Формат следующего адресного пакета определяет какой режим вводится: ведущий передатчик или ведущий приемник. Если передается SLA+W, то вводится режим MT (ведущий передатчик), а если SLA+R, то вводится режим MR (ведущий приемник). Все упоминаемые в этом разделе коды состояния в позиции бит предделителя имеют нулевые значения.

Передача условия START инициируется путем записи в TWCR следующего значения:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	1	0	x	1	0	x

Для разрешения работы двухпроводного последовательного интерфейса необходимо установить бит TWEN. Запись лог. 1 в TWSTA инициирует передачу условия START, а запись лог. 1 в TWINT приводит к сбросу флага TWINT. После записи данного значения TWI тестирует двухпроводную последовательную шину и генерирует условие START сразу после освобождения шины. После передачи условия START аппаратно устанавливается флаг INT, а в регистр TWSR помещается код состояния $08 (см. табл. 5.2). Для перевода в режим ведущего передатчика необходимо передать SLA+W. Это выполняется путем записи значения

SLA+W в регистр TWDR. После этого необходимо сбросить флаг TWINT (путем записи в него лог. 1) для продолжения сеанса связи. Данное условие выполняется путем записи следующего значения в TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	0	0	x	1	0	x

После передачи SLA+W и приема бита подтверждения флаг TWINT снова устанавливается, а в регистр TWSR помещается код состояния, который может иметь несколько значений. В режиме ведущего код состояния может быть $18, $20 или $38. Для каждого из этих кодов состояний необходимо выполнить адекватные действия, что отражено в табл. 5.2.

После успешной передачи SLA+W должен быть передан пакет данных. Его передача инициируется записью байта данных в TWDR. Доступ на запись к TWDR разрешен только тогда, когда флаг TWINT равен 1. В противном случае доступ блокируется и устанавливается флаг ошибочной записи TWWC в регистре TWCR. После обновления TWDR необходимо сбросить бит TWINT (путем записи в него лог. 1) для продолжения сеанса связи. Это можно выполнить путем записи следующего значения в регистр TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	0	0	x	1	0	x

Данная последовательность повторяется до тех пор, пока не будет передан последний байт. После этого генерируется условие STOP или RESTART. Условие STOP генерируется путем записи следующего значения TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	0	1	x	1	0	x

Условие RESTART генерируется путем записи следующего значения в TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	1	0	x	1	0	x

После передачи условия RESTART (состояние $10) двухпроводной последовательный интерфейс может обращаться к тому же подчиненному устройству или же к новому, при этом не требуется передача условия STOP. Таким образом, RESTART полезно использовать для смены подчиненного устройства в режимах ведущий передатчик и ведущий приемник без потери управления шиной.

Таблица 5.2

Коды состояния модуля TWI в режиме ведущего передатчика

Код состояния (TWSR) Биты предделителя равны 0	Состояние шины I2C и логики модуля TWI	Действия программы					Следующее действие логики модуля TWI
		В/из TWDR	В TWCR				
			TWSTA	TWSTO	TWINT	TWEA	
1	2	3	4	5	6	7	8
$08	Передано условие START	Загрузка SLA+W	0	0	1	x	Будет передан SLA+W; ACK или NACK будет принят
$10	Передано условие RESTART	Загрузка SLA+W или	0	0	1	x	Будет передан SLA+W; ACK или NACK будет принят
		Загрузка SLA+R	0	0	1	x	Будет передан SLA+W; логика модуля переключится в режим ведущего приемника
$18	Передан SLA+W; принят ACK	Загрузка байта данных или	0	0	1	x	Будет передан байт данных и принят ACK или NACK
		Нет действий или	1	0	1	x	Будет сформирован RESTART
		Нет действий или	0	1	1	x	Будет сформирован STOP и сброшен флаг TWSTO
		Нет действий	1	1	1	x	Вслед за условием START будет сформирован STOP и сброшен флаг TWSTO
$20	Передан SLA+W; принят NACK	Загрузка байта данных или	0	0	1	x	Будет передан байт данных и принят ACK или NACK
		Нет действий или	1	0	1	x	Будет сформирован RESTART
		Нет действий или	0	1	1	x	Будет сформирован STOP и сброшен флаг TWSTO

Окончание табл. 5.2

1	2	3	4	5	6	7	8
		Нет действий	1	1	1	x	Вслед за условием START будет сформирован STOP и сброшен флаг TWSTO
$28	Передан байт данных; принят ACK	Загрузка байта данных или	0	0	1	x	Будет передан байт данных и принят ACK или NACK
		Нет действий или	1	0	1	x	Будет сформирован RESTART
		Нет действий или	0	1	1	x	Будет сформирован STOP и сброшен флаг TWSTO
		Нет действий	1	1	1	x	Вслед за условием START будет сформирован STOP и сброшен флаг TWSTO
$30	Передан байт данных; принят NACK	Загрузка байта данных или	0	0	1	x	Будет передан байт данных и принят ACK или NACK
		Нет действий или	1	0	1	x	Будет сформирован RESTART
		Нет действий или	0	1	1	x	Будет сформирован STOP и сброшен флаг TWSTO
		Нет действий	1	1	1	x	Вслед за условием START будет сформирован STOP и сброшен флаг TWSTO
$38	Потеря арбитража при передаче SLA+W или байта данных	Нет действий или	0	0	1	x	Шина I2C будет освобождена и модуль TWI перейдет в неадресованный режим
		Нет действий	1	0	1	x	Будет сформировано условие START когда шина I2C освободится

Режим ведущего приемника

В режиме ведущего приемника принимается несколько байт данных от подчиненного передатчика. Для ввода режима ведущего необходимо передать условие START. Формат следующего адресного пакета определит, будет ли введенный режим ведущий передатчик или приемник. Если передается SLA+W, то вводится режим ведущий передатчик, если же передается SLA+R, то вводится режим ведущий приемник. Во всех кодах состояния, приведенных в этом разделе, биты предделителя равны нулю.

Передача условия START инициируется путем записи следующего значения в TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	1	0	x	1	0	x

Для разрешения работы двухпроводного последовательного интерфейса необходимо установить бит TWEN. Передача условия START инициируется записью лог. 1 в TWSTA. Для сброса флага TWINT необходимо записать в него лог. 1. TWI выполнит генерацию условия START только после тестирования шины и ее освобождения. После передачи условия START флаг TWINT устанавливается аппаратно, а в регистр TWSR помещается код состояния $08 (см. табл. 5.3). Для ввода режима ведущий приемник необходимо передать SLA+R. Это выполняется путем записи значения SLA+R в TWDR. После этого необходимо сбросить флаг TWINT (путем записи в него лог. 1) для продолжения сеанса связи. Для этого в регистр TWCR необходимо поместить следующее значение:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	0	0	x	1	0	x

После передачи SLA+R и приема бита подтверждения снова устанавливается флаг TWINT, а в регистр TWSR помещается код состояния, который может иметь несколько значений: $38, $40 или $48. Действия, которые выполняются при каждом из этих значений, представлены в табл. 5.3. Принятые данные хранятся в регистре TWDR после аппаратной установки флага TWINT. Данная последовательность повторяется до приема последнего байта. После этого ведущий приемник информирует подчиненный передатчик отправкой бита NACK после приема последнего байта данных. Сеанс связи завершается генерацией условия STOP или RESTART. Условие STOP генерируется путем записи в регистр TWCR следующего значения:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	0	1	x	1	0	x

Таблица 5.3

Коды состояния модуля TWI в режиме ведущего приемника

Код состояния (TWSR) Биты предделителя равны 0	Состояние шины I2C и логики модуля TWI	Действия программы					Следующее действие логики модуля TWI
		В/из TWDR	В TWCR				
			TWSTA	TWSTO	TWINT	TWEA	
1	2	3	4	5	6	7	8
$08	Передано условие START	Загрузка SLA+W	0	0	1	x	Будет передан SLA+W; ACK или NACK будет принят
$10	Передано условие RE-START	Загрузка SLA+W или	0	0	1	x	Будет передан SLA+W; ACK или NACK будет принят
		Загрузка SLA+R	0	0	1	x	Будет передан SLA+W; логика модуля переключится в режим ведущего приемника
$38	Потеря арбитража при передаче SLA+W или байта данных	Нет действий или	0	0	1	x	Шина I2C будет освобождена и модуль TWI перейдет в неадресованный режим
		Нет действий	1	0	1	x	Будет сформировано условие START когда шина I2C освободится
$40	Передан SLA+R; принят ACK	Нет действий или	0	0	1	0	Будет принят байт данных и передан NACK
		Нет действий	0	0	1	1	Будет принят байт данных и передан ACK

Окончание табл. 5.3

1	2	3	4	5	6	7	8
$48	Передан SLA+R; принят NACK	Нет действий или	1	0	1	x	Будет сформирован RESTART
		Нет действий или	0	1	1	x	Будет сформирован STOP и сброшен флаг TWSTO
		Нет действий	1	1	1	x	Вслед за условием START будет сформирован STOP и сброшен флаг TWSTO
$50	Принят байт данных; принят ACK	Чтение байта данных или	0	0	1	0	Будет принят байт данных и передан NACK
		Чтение байта данных	0	0	1	1	Будет принят байт данных и передан ACK
$58	Принят байт данных; принят NACK	Чтение байта данных или	1	0	1	x	Будет сформирован RESTART
		Чтение байта данных или	0	1	1	x	Будет сформирован STOP и сброшен флаг TWSTO
		Чтение байта данных	1	1	1	x	Вслед за условием START будет сформирован STOP и сброшен флаг TWSTO

Условие RESTART генерируется путем записи в TWCR следующего значения:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	1	x	1	0	x	1	0	x

После генерации условия RESTART (состояние $10) двухпроводной последовательный интерфейс может обращаться к тому же подчиненному или к новому подчиненному без генерации условия STOP. RESTART позволяет ведущему переключаться между подчиненными, режимом ведущего передатчика и ведущего приемника без потери управления над шиной.

Режим подчиненного приемника

В режиме подчиненного приемника принимается несколько байт данных от ведущего передатчика. Во всех кодах состояния, приведенных в этом разделе, биты предделителя равны нулю.

Для ввода режима подчиненного приемника необходимо выполнить инициализацию регистров TWAR и TWCR следующим образом:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Значение	Собственный подчиненный адрес устройства							

Старшие семь разрядов образуют адрес, который присваивается подчиненному приемнику. Если в младшем разряде записана лог. 1, то TWI будет отвечать на адрес общего вызова ($00). В противном случае он игнорирует адрес общего вызова.

В регистр TWCR должно быть записано следующее значение:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	0	1	0	0	0	1	0	x

Для разрешения работы TWI необходимо записать лог. 1 в TWEN. Для разрешения подтверждения собственного подчиненного адреса или адреса общего вызова в TWEA записывается 1. Битам TWSTA и TWSTO необходимо присвоить нулевое значение.

После инициализации TWAR и TWCR схема TWI ожидает получения собственного подчиненного адреса (или, если разрешено, адреса общего вызова), а вслед за ним – бита направления данных. Если бит направления равен «0» (запись), то TWI переходит в режим «подчиненный приемник», в противном случае вводится режим «подчиненный передатчик». После приема собственного подчиненного адреса и бита записи устанавливается флаг TWINT, а в регистр TWSR помещается код состояния. По коду состояния определяется, какие программные действия необходимо предпринять. В табл. 5.4 собрана информация о предпринимаемых действиях для каждого возможного значения кода состояния. Режим подчиненного приемника также вводится, если теряется арбитраж, когда TWI находился в режиме ведущего (см. состояния $68 и $78).

Таблица 5.4

Коды состояния модуля TWI в режиме подчиненного приемника

Код состояния (TWSR) Биты предделителя равны 0	Состояние шины I2C и логики модуля TWI	Действия программы					Следующее действие логики модуля TWI
		В/из TWDR	В TWCR				
			TWSTA	TWSTO	TWINT	TWEA	
1	2	3	4	5	6	7	8
$60	Был принят собственный адрес и передан ACK	Нет действий или	x	0	1	0	Будет принят байт данных и передан NACK
		Нет действий	x	0	1	1	Будет принят байт данных и передан ACK
$68	При передаче SLA+R/W в режиме ведущего был потерян арбитраж; был принят собственный SLA+W и передан ACK	Нет действий или	x	0	1	0	Будет принят байт данных и передан NACK
		Нет действий	x	0	1	1	Будет принят байт данных и передан ACK
$70	Был принят адрес общего вызова и передан ACK	Нет действий или	x	0	1	0	Будет принят байт данных и передан NACK
		Нет действий	x	0	1	1	Будет принят байт данных и передан ACK

Продолжение табл. 5.4

1	2	3	4	5	6	7	8
$78	При передаче SLA+R/W был потерян арбитраж; был принят адрес общего вызова и передан ACK	Нет действий или	x	0	1	0	Будет принят байт данных и передан NACK
		Нет действий	x	0	1	1	Будет принят байт данных и передан ACK
$80	Был принят байт данных и передан ACK (ведомый был адресован собственным SLA+W)	Чтение байта данных или	x	0	1	0	Будет принят байт данных и передан NACK
		Чтение байта данных	x	0	1	1	Будет принят байт данных и передан ACK
$88	Был принят байт данных и передан NACK (ведомый был адресован собственным SLA+W)	Чтение байта данных или	0	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова)
		Чтение байта данных или	0	0	1	1	Модуль переключится в неадресованный ведомый режим; не будет распознаваться SLA; GCA (адрес общего вызова) будет распознаваться, если бит TWGCE = «1»
		Чтение байта данных или	1	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова); когда шина I2C освободится, будет сформировано условие START

Продолжение табл. 5.4

1	2	3	4	5	6	7	8
		Чтение байта данных	1	0	1	1	Модуль переключится в неадресованный ведомый режим; будет распознаваться собственный SLA; адрес общего вызова будет распознаваться, если бит TWGCE = «1»; когда шина I2C освободится, будет сформировано условие START
$90	Был принят байт данных и передан ACK (ведомый был адресован адресом общего вызова)	Чтение байта данных или	x	0	1	0	Будет принят байт данных и передан NACK
		Чтение байта данных	x	0	1	1	Будет принят байт данных и передан ACK
$98	Был принят байт данных и передан NACK (ведомый был адресован адресом общего вызова)	Чтение байта данных или	0	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова)
		Чтение байта данных или	0	0	1	1	Модуль переключится в неадресованный ведомый режим; не будет распознаваться SLA; GCA (адрес общего вызова) будет распознаваться, если бит TWGCE = «1»
		Чтение байта данных или	1	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова); когда шина I2C освободится, будет сформировано условие START

Окончание табл. 5.4

1	2	3	4	5	6	7	8
		Чтение байта данных	1	0	1	1	Модуль переключится в неадресованный ведомый режим; будет распознаваться собственный SLA; адрес общего вызова будет распознаваться, если бит TWGCE = «1»; когда шина I2C освободится, будет сформировано условие START
$A0	Был принят STOP или RESTART в то время как ведомый был в адресованном режиме	Нет действий	0	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова)
			0	0	1	1	Модуль переключится в неадресованный ведомый режим; не будет распознаваться SLA; GCA (адрес общего вызова) будет распознаваться, если бит TWGCE = «1»
			1	0	1	0	Модуль переключится в неадресованный ведомый режим;не будут распознаваться SLA и GCA (адрес общего вызова); когда шина I2C освободится, будет сформировано условие START
			1	0	1	1	Модуль переключится в неадресованный ведомый режим; будет распознаваться собственный SLA; адрес общего вызова будет распознаваться, если бит TWGCE = «1»; когда шина I2C освободится, будет сформировано условие START

Если бит TWEA сбросить во время передачи, то TWI сформирует бит NACK (нет подтверждения) на линии SDA после приема следующего байта данных. Данное свойство может использоваться для сигнализации состояния, когда подчиненный не может больше принимать байты данных. Если TWEA равен нулю, то TWI не подтверждает свой подчиненный адрес. Однако последовательная шина остается под контролем, и функция распознавания адреса может быть активизирована в любой момент путем установки бита TWEA. Это означает, что бит TWEA можно использовать для временной изоляции TWI от шины I2C.

Во всех режимах сна, кроме холостого хода (Idle), синхронизация TWI отключается. Если бит TWEA установлен, то интерфейс останется способным подтверждать прием своего собственного подчиненного адреса или адреса общего вызова за счет использования сигнала синхронизации шины в качестве тактового источника. При обнаружении запроса микроконтроллер выходит из режима сна, при этом линия SCL остается на низком уровне в процессе пробуждения и до сброса флага TWINT (записью в него «1»). Далее выполняется прием данных обычным способом при обычной системной синхронизации. Учтите, что если для микроконтроллера выбрано большое время запуска, то линия SCL может оказаться длительно в низком состоянии и заблокировать другой обмен информацией.

Обратите внимание, что после пробуждения регистр данных TWDR не отражает последний байт, присутствовавший на шине во время выхода из указанных выше режимов сна.

Режим подчиненного передатчика

В режиме подчиненного передатчика выполняется передача байт данных ведущему приемнику. Во всех кодах состояния, приведенных в этом разделе, биты предделителя равны нулю.

Для ввода режима подчиненного передатчика необходимо инициализировать регистры TWAR и TWCR следующим образом:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Значение	Собственный подчиненный адрес устройства							

Старшие семь разрядов образуют адрес, который присваивается подчиненному приемнику. Если в младшем разряде записана лог. 1, то TWI будет отвечать на адрес общего вызова ($00). В противном случае он игнорирует адрес общего вызова.

В регистр TWCR должно быть записано следующее значение:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Значение	0	1	0	0	0	1	0	x

Для разрешения работы TWI необходимо записать лог. 1 в TWEN. Для разрешения подтверждения собственного подчиненного адреса или адреса общего вызова в TWEA записывается 1. Битам TWSTA и TWSTO необходимо присвоить нулевое значение.

После инициализации TWAR и TWCR схема TWI ожидает получения собственного подчиненного адреса (или, если разрешен, адреса общего вызова), а вслед за ним – бита направления данных. Если бит направления равен «1» (чтение), то TWI переходит в режим «подчиненный передатчик», иначе вводится режим «подчиненный приемник». После приема собственно подчиненного адреса и бита записи устанавливается флаг TWINT, а в регистр TWSR помещается код состояния. Код состояния позволяет определить, какие программные действия необходимо выполнить. Подробности по использованию кодов состояний представлены в табл. 5.5. Режим «подчиненный передатчик» также вводится, если теряется арбитраж когда TWI находился в режиме ведущего (см. состояние $B0). Если бит TWEA обнулить во время передачи, то TWI передаст последний байт. Вводится состояние $C0 или состояние $C8 в зависимости от того передал ведущий приемник бит ACK (есть подтверждение) или NACK (нет подтверждения) за последним байтом. TWI переходит в безадресный подчиненный режим и далее игнорирует ведущего, если тот продолжает передачу. Таким образом, ведущий приемник будет принимать все «1». Состояние $C8 вводится, если ведущий требует передачи дополнительных байт данных (путем передачи ACK), даже если подчиненный передал последний байт (TWEA равен нулю и ожидается прием NACK от ведущего).

Пока TWEA равен нулю, TWI не отвечает на собственный подчиненный адрес. Однако последовательная шина остается под контролем и функция распознавания адреса может быть активизирована в любой момент путем установки бита TWEA. Это означает, что бит TWEA можно использовать для временной изоляции TWI от двухпроводной последовательной шины.

Во всех режимах сна, кроме холостого хода (Idle), синхронизация TWI отключается. Если бит TWEA установлен, то интерфейс останется способным подтверждать прием своего собственного подчиненного адреса или адреса общего вызова за счет использования сигнала синхронизации шины в качестве тактового источника. При обнаружении запроса микроконтроллер выходит из режима сна, при этом линия SCL остается на низком уровне в процессе пробуждения и до сброса флага TWINT (записью в него «1»). Далее выполняется прием данных обычным способом при обычной системной синхронизации. Учтите, что если для микроконтроллера выбрано большое время запуска, то линия SCL может оказаться длительно в низком состоянии и заблокировать другой обмен информацией.

Таблица 5.5

Коды состояния модуля TWI в режиме подчиненного передатчика

Код состояния (TWSR) Биты предделителя равны 0	Состояние шины I2C и логики модуля TWI	Действия программы					Следующее действие логики модуля TWI
		В/из TWDR	В TWCR				
			TWSTA	TWSTO	TWINT	TWEA	
1	2	3	4	5	6	7	8
A8	Был принят собственный адрес и передан ACK	Загрузка байта данных или	x	0	1	0	Будет передан последний байт данных; ожидается прием NACK
		Загрузка байта данных	x	0	1	1	Будет переданы байт данных; ожидается прием ACK
B0	При передаче SLA+R/W в режиме ведущего был потерян арбитраж; был принят собственный SLA+R и передан ACK	Загрузка байта данных или	x	0	1	0	Будет передан последний байт данных; ожидается прием NACK
		Загрузка байта данных	x	0	1	1	Будет переданы байт данных; ожидается прием ACK
B8	Был передан байт данных, содержащийся в регистре TWDR; принят ACK	Загрузка байта данных или	x	0	1	0	Будет передан последний байт данных; ожидается прием NACK
		Загрузка байта данных	x	0	1	1	Будет переданы байт данных; ожидается прием ACK

Продолжение табл. 5.5

1	2	3	4	5	6	7	8
C0	Был передан байт данных, содержащийся в регистре TWDR; принят NACK	Нет действий или	0	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова)
		Нет действий или	0	0	1	1	Модуль переключится в неадресованный ведомый режим; не будет распознаваться SLA; GCA (адрес общего вызова) будет распознаваться, если бит TWGCE = «1»
		Нет действий или	1	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова); когда шина I2C освободится, будет сформировано условие START
		Нет действий	1	0	1	1	Модуль переключится в неадресованный ведомый режим; будет распознаваться собственный SLA; адрес общего вызова будет распознаваться, если бит TWGCE = «1»; когда шина I2C освободится, будет сформировано условие START

Продолжение табл. 5.5

1	2	3	4	5	6	7	8
C8	Был передан последний байт данных, содержащийся в регистре TWDR (TWEA = 0); принят ACK	Нет действий или	0	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова)
		Нет действий или	0	0	1	1	Модуль переключится в неадресованный ведомый режим; не будет распознаваться SLA; GCA (адрес общего вызова) будет распознаваться, если бит TWGCE = «1»
		Нет действий или	1	0	1	0	Модуль переключится в неадресованный ведомый режим; не будут распознаваться SLA и GCA (адрес общего вызова); когда шина I2C освободится, будет сформировано условие START
		Нет действий	1	0	1	1	Модуль переключится в неадресованный ведомый режим; будет распознаваться собственный SLA; адрес общего вызова будет распознаваться, если бит TWGCE = «1»; когда шина I2C освободится, будет сформировано условие START

Обратите внимание, что после пробуждения регистр данных TWDR не отражает последний байт, присутствовавший на шине во время выхода из указанных выше режимов сна.

Прочие состояния

Имеются два кода состояния, которые отличаются от упомянутых выше (см. табл. 5.6). Состояние $F8 индицирует, что нет доступной информации, т. к. не установлен флаг TWINT. Это может произойти между другими состояниями и когда TWI не участвует в последовательной передаче данных.

Таблица 5.6

Прочие состояния модуля TWI

Код состояния (TWSR) Биты предделителя равны 0	Состояние шины I2C и логики модуля TWI	Действия программы					Следующее действие логики модуля TWI
		В/из TWDR	В TWCR				
			TWSTA	TWSTO	TWINT	TWEA	
$F8	Нет информации о состоянии модуля TWI; TWINT = «0»	Нет действий	Нет действий				Ожидание или продолжение текущей передачи
$00	Ошибка во время последовательной ередачи данных на шине; START или STOP в неверной позиции формата посылки	Нет действий	0	1	1	x	Влияет только на внутреннюю логику модуля, условие STOP на шине не формируется; шина освобождается и сбрасывается бит TWSTO

Состояние $00 индицирует, что во время последовательной передачи данных на шине возникла ошибка. Ошибка возникает, если условия START или STOP формируются в неверной позиции формата по-

сылки. Примеры таких неточных позиций могут существовать во время передачи адресного байта, байта данных и бита подтверждения. После возникновения ошибки устанавливается флаг TWINT. Для выхода из состояния ошибки необходимо установить флаг TWSTO и сбросить TWINT путем записи в него «1». Это приводит к переводу TWI в безадресный режим и к сбросу флага TWSTO (другие биты в TWCR не затрагиваются). Линии SDA и SCL освобождаются и условие STOP не передается.

Сочетание нескольких режимов

В некоторых случаях сочетаются несколько режимов TWI для обеспечения желаемого действия. В качестве примера рассмотрим чтение данных из последовательного EEPROM. Обычно, сеанс связи организовывается в такой последовательности:

1. Инициируется сеанс связи
2. В EEPROM отправляется инструкция с указанием адреса считываемой ячейки
3. Выполняется чтение
4. Завершается сеанс связи

Обратите внимание, что данные передаются как от ведущего к подчиненному, так и обратно, от подчиненного к ведущему. Ведущий инструктирует подчиненного какую ячейку он желает считать, для чего используется режим «Ведущий передатчик». В дальнейшем данные передаются подчиненным, что требует использования режима «Ведущий приемник». Следовательно, направление передачи данных изменяется. Ведущий должен сохранить управление над шиной на каждом из этапов, а каждый из шагов должен быть выполнен как элементарное действие. Если данный принцип нарушить в многомастерной системе, то другой ведущий может обратиться к EEPROM на шагах 2 и 3 и изменить указатель данных. Это приведет к тому, что ведущий считает данные из ячейки с неверным адресом. Таким образом, направление передачи данных необходимо изменять только передачей условия RESTART между передачей адресного байта и приемом байта. Передачей RESTART мастер сохранит свое «господство» на шине.

Системы с несколькими ведущими и арбитраж

Если к одной шине подключено несколько ведущих, то передача может быть инициирована одновременно одним или несколькими из них. Стандарт I2C гарантирует, что в таких ситуациях разрешается передача только одному ведущему, при этом не будет происходить потеря данных.

Ниже приведены несколько различных сценариев, возникающих в процессе арбитража:

- Два или более ведущих выполняют идентичную связь с одним и тем же подчиненным. В этом случае ни один подчиненный и ни один из ведущих не узнает об этой конфликтной ситуации.
- Два или более ведущих обращаются к тому же подчиненному с различными данными или битом направления данных. В этом случае возникает арбитраж во время передачи бита R/W или же бит данных. Ведущие, которые выводят на SDA лог. 1, в то время, как другие выводят лог. 0, теряют арбитраж. Ведущие, которые проиграли арбитраж, переходят в безадресный подчиненный режим или ожидают освобождения шины и затем передают новое условие START, что зависит от программы.
- Два или более ведущих обращаются к различным подчиненным. В этом случае арбитраж возникает во время передачи бит SLA. Ведущие, которые выводят на SDA лог. 1, в то время как другие выводят лог. 0, теряют арбитраж. Ведущие, которые проиграли арбитраж во время передачи SLA, переходят в подчиненный режим для проверки, не обращается ли к ним выигравший арбитраж ведущий. Если такая адресация действительно выполняется, то они переключаются в режим «Подчиненный приемник» или «Подчиненный передатчик» в зависимости от значения бита направления. Если адресации не было, то они перейдут в безадресный подчиненный режим или будут ожидать освобождения шины и после этого передадут новое условие START (определяется программой).

§ 5.3. Алгоритмы работы с модулем TWI

Работа модуля TWI может быть организована как с использованием прерываний, так и без них. Ниже приведены алгоритмы работы с модулем без использования прерываний. Кроме того, в данных алгоритмах не предусмотрена обработка ошибок, что также необходимо учитывать при отладке программы.

§ 5.3.1. Алгоритм инициализации модуля TWI

Прежде чем начать обмен данными по шине I2C необходимо проинициализировать модуль TWI контроллера. Ниже приведен один из возможных алгоритмов инициализации.

1. Регистр TWCR = 0x00 (выключение модуля TWI)
2. Настройка на ввод тех линий порта, которые мультиплексированы с линиями SCL и SDA.

3. *Режим ведущего*

Запись в регистры TWBR и TWSR значений, задающих скорость обмена данными по шине I2C. При этом частота тактовых импульсов на линии SCL определяется из следующего выражения:

$$f_{SCL} = \frac{F_{CPU}}{16 + 2(TWBR) \cdot 4^{TWPS}},$$

где F_{CPU} – тактовая частота контроллера; $TWBR$ – содержимое регистра TWBR, $TWPS$ – значение двух младших бит регистра TWSR. Значение, хранящееся в регистре TWBR должно быть больше 10, это необходимо для корректной работы модуля.

Режим ведомого

Запись в старшие 7 бит регистра TWAR адреса, который присваивается ведомому. Если нужна поддержка адреса общего вызова, то младший бит (бит 0) регистра TWAR должен быть установлен в «1». Если поддержка адреса общего вызова не нужна, то младший бит регистра TWAR должен быть сброшен в «0».

§ 5.3.2. Алгоритм формирования условия START

Для формирования условия START на шине I2C нужно выполнить следующие действия:

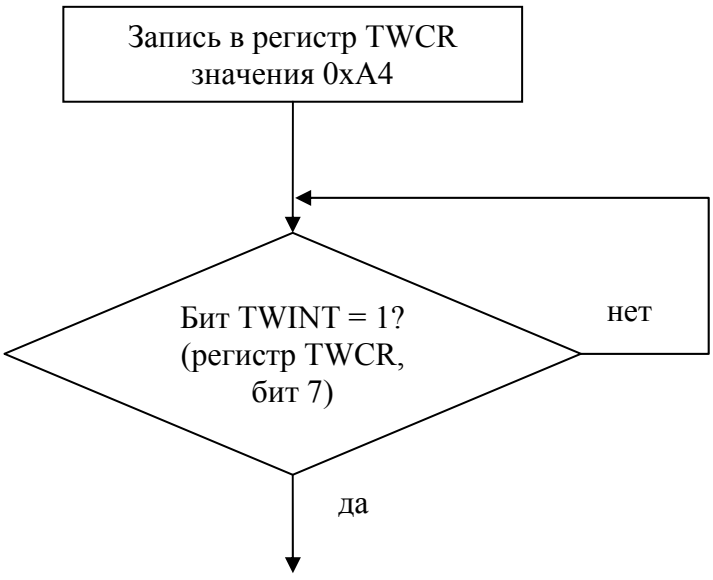


Рис. 5.8. Алгоритм формирования условия START

§ 5.3.3. Алгоритм формирования условия RESTART

Условие RESTART формируется таким же образом, как и условие START (§ 5.3.2).

§ 5.3.4. Алгоритм формирования условия STOP

Одним из алгоритмов формирования условия STOP является следующий:

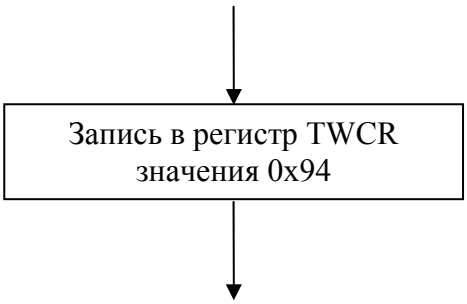


Рис. 5.9. Алгоритм формирования условия STOP

§ 5.3.5. Алгоритм передачи байта ведущим

Ведущее устройство может передать байт, используя следующий алгоритм:

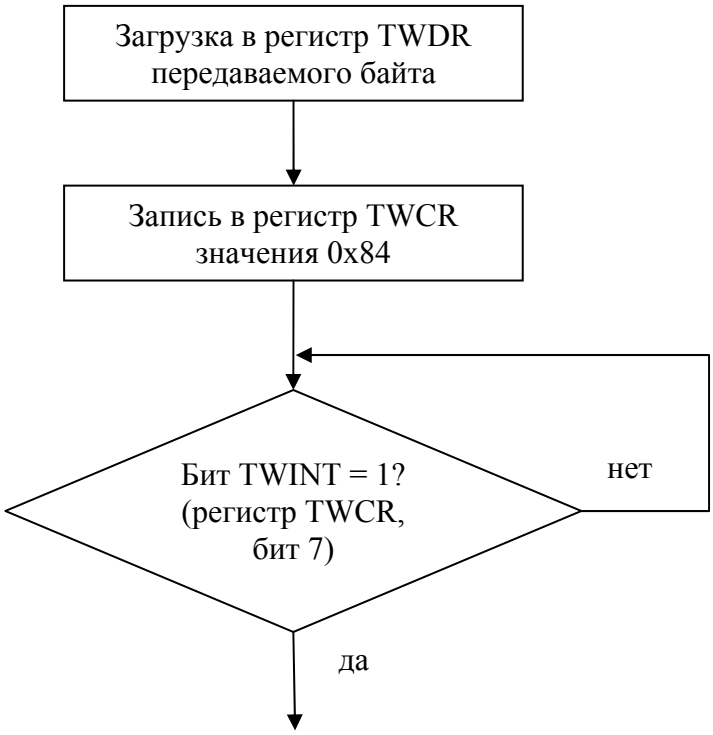


Рис. 5.10. Алгоритм передачи байта ведущим

§ 5.3.6. Алгоритм приема байта ведущим с формированием подтверждения

Прием байта ведущим устройством с формированием подтверждения можно организовать по следующему алгоритму:

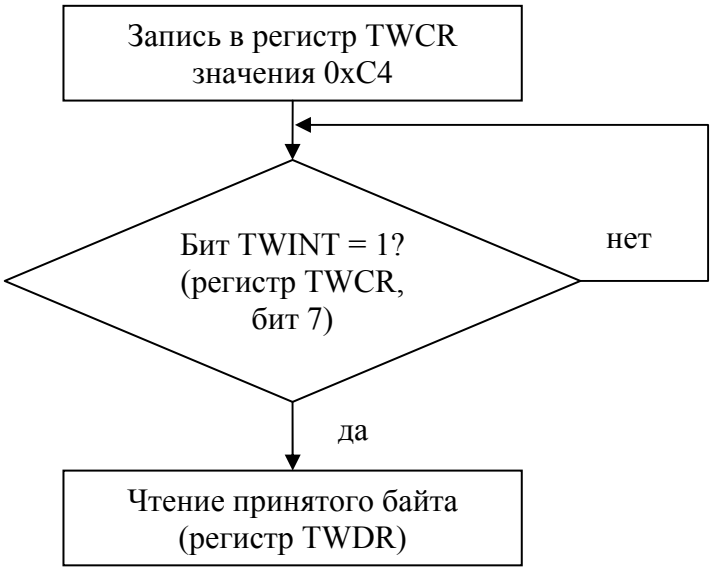


Рис. 5.11. Алгоритм приема байта ведущим с формированием подтверждения

§ 5.3.7. Алгоритм приема байта ведущим без формирования подтверждения

Прием байта ведущим устройством без формирования подтверждения можно организовать по следующему алгоритму:

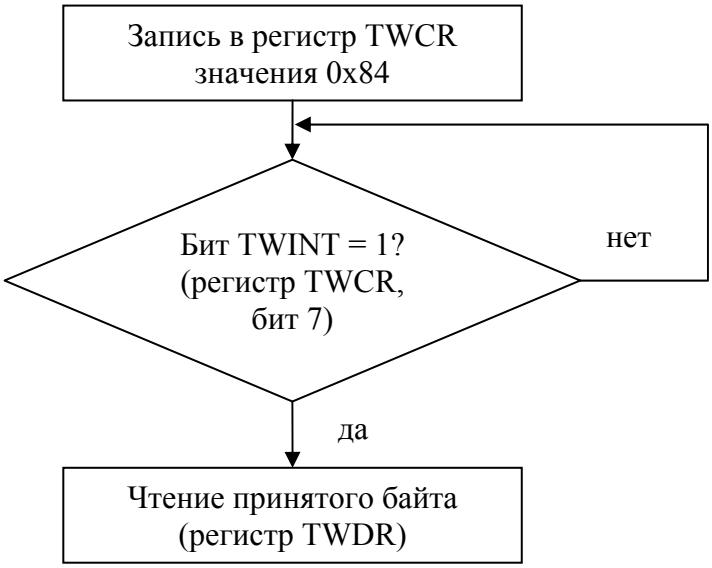


Рис. 5.12. Алгоритм приема байта ведущим без формирования подтверждения

§ 5.3.8. Алгоритм приема байта ведомым

Для организации приема байта ведомым устройством может быть применен следующий алгоритм:

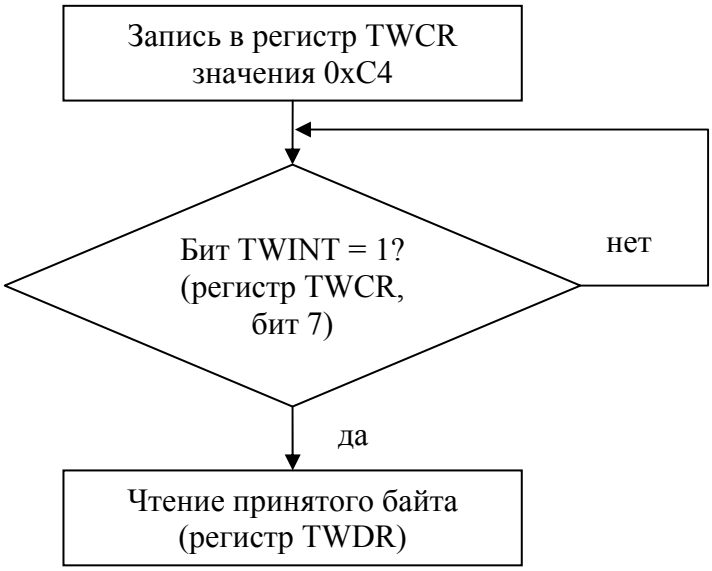


Рис. 5.13. Алгоритм приема байта ведомым

§ 5.3.9. Алгоритм передачи байта ведомым

Передача байта ведомым устройством может быть организована следующим образом:

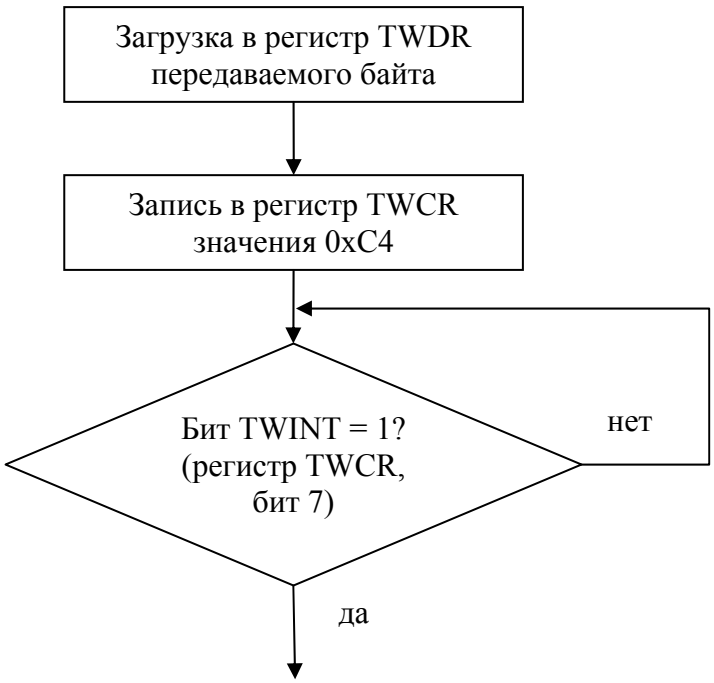


Рис. 5.14. Алгоритм передачи байта ведомым

§ 5.3.10.Алгоритм приема ведомым последнего байта

Для организации приема последнего байта ведомым устройством может быть применен следующий алгоритм:

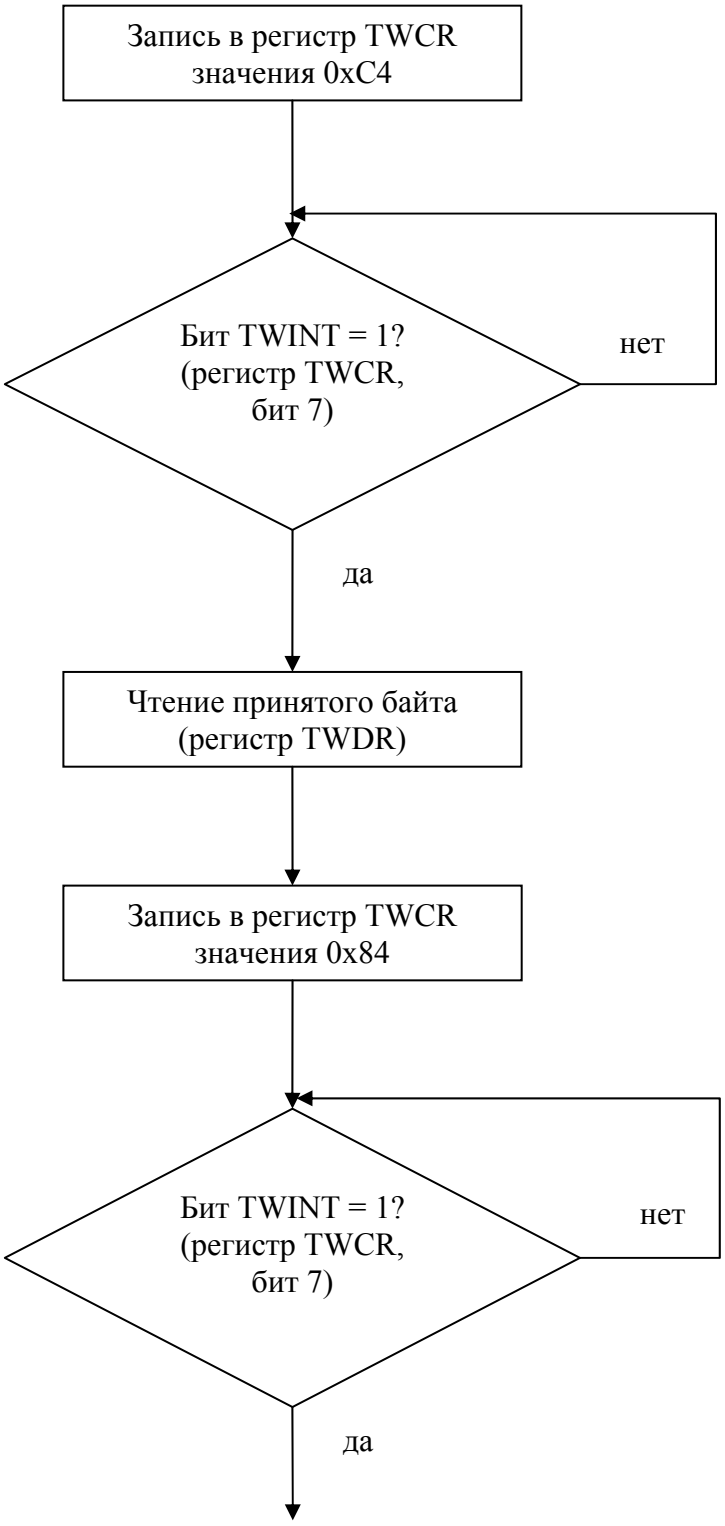


Рис. 5.15. Алгоритм приема ведомым последнего байта

§ 5.3.11.Алгоритм передачи ведомым последнего байта

Передача последнего байта ведомым устройством может быть организована следующим образом:

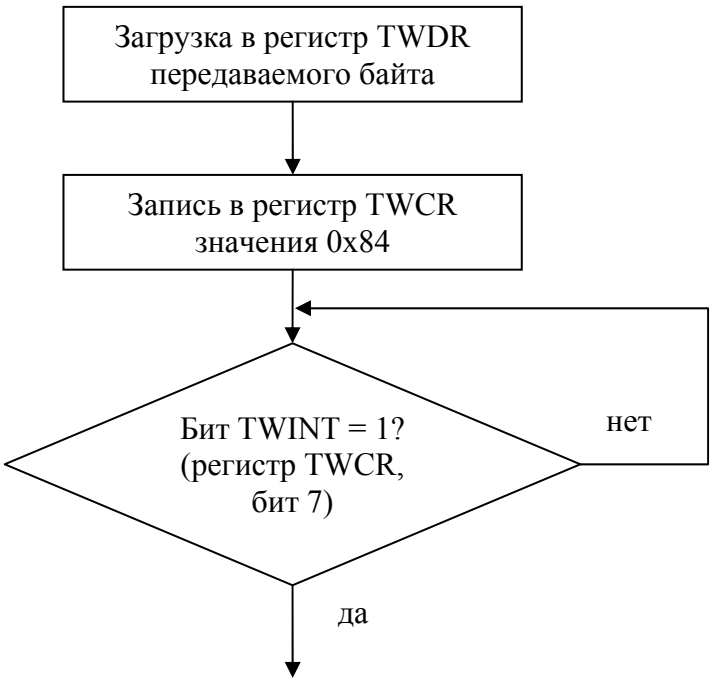


Рис. 5.16. Алгоритм передачи ведомым последнего байта

§ 5.4. Примеры программ

§ 5.4.1. Пример программы ведущего микроконтроллера

В качестве примера рассмотрим программу, предназначенную для работы с часами реального времени MK41T56 (предполагается, что часы уже настроены). Ведущий микроконтроллер производит считывание текущего времени (секунды, минуты, часы) и его сохранение в регистрах. Алгоритм программы представлен на рис. 5.17 и 5.18.

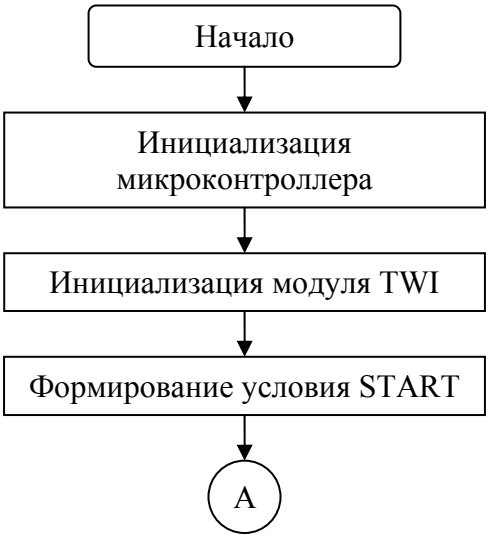


Рис. 5.17. Алгоритм программы ведущего микроконтроллера (начало)

Рис. 5.18. Алгоритм программы ведущего микроконтроллера (окончание)

Алгоритмы подпрограмм, формирующих условия START, RE-START, STOP, а так же осуществляющих прием и передачу байта приведены в § 5.3.

Ниже приведена программа, реализующая описанный алгоритм. Программа написана для микроконтроллера ATmega16 с кварцевым резонатором 4 МГц. Микросхема часов реального времени имеет адрес 0xD0, обмен данными по шине I2C ведется на частоте 100 кГц.

```
.INCLUDE «m16def.inc»
.DEVICE ATmega16

.DEF temp = r16
.DEF byte = r17

.DEF seconds = r0          ;регистр для хранения секунд
.DEF minutes = r1          ;регистр для хранения минут
.DEF hours = r2            ;регистр для хранения часов
.EQU ADR_W = 0xD0          ;байт адреса микросхемы (для записи, бит R/W = 0)
.EQU ADR_R = 0xD1          ;байт адреса микросхемы (для чтения, бит R/W = 1)
.EQU ADR_SEC = 0x00        ;адрес ячейки, в которой хранятся секунды

.CSEG

;-------------------------------------------------------------------------------
;инициализация
;-------------------------------------------------------------------------------

;инициализация стека
        ldi r16, low(RAMEND)
        out SPL, r16
        ldi r16, high(RAMEND)
        out SPH, r16

;настройка модуля TWI
        ldi temp, 0x00          ;выключение
        out TWCR, temp          ;модуля TWI

        ldi temp, 0xFC          ;настройка линий
        out DDRC, temp          ;SCL и SDA порта C
        out PORTC, temp         ;на ввод

        ldi temp, 0x0C          ;задание
        out TWBR, temp          ;частоты передачи
        ldi temp, 0x00          ;данных
        out TWSR, temp          ;по шине I2C
```

```
;-------------------------------------------------------------------------------
;основная программа
;-------------------------------------------------------------------------------

        call START              ;формирование условия START

        ldi byte, ADR_W         ;передача байта адреса микросхемы (бит R/W = 0)
        call SEND_B             ;на шину I2C

        ldi byte, ADR_SEC       ;передача адреса ячейки памяти
        call SEND_B             ;в которой хранятся секунды

        call RESTART            ;формирование условия RESTART

        ldi byte, ADR_R         ;передача байта адреса микросхемы (бит R/W = 1)
        call SEND_B             ;на шину I2C

        call REC_ACK            ;прием байта секунд и подтверждение приема
        mov seconds, byte       ;сохранение принятого байта

        call REC_ACK            ;прием байта минут и подтверждение приема
        mov minutes, byte       ;сохранение принятого байта

        call REC_NACK           ;прием байта часов и неподтверждение приема
        mov hours, byte         ;сохранение принятого байта

        call STOP               ;формирование условия STOP

loop:
        jmp loop                ;переход на метку loop

;================================================
;подпрограммы работы с модулем TWI
;================================================

;подпрограмма формирования условия START (RESTART)
START:
RESTART:
        ldi temp, 0xA4          ;запуск формирования
        out TWCR, temp          ;условия START (RESTART)
m_st:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_st               ;и, если нет, то переход на метку m_st
        ret                     ;возврат из подпрограммы

;подпрограмма формирования условия STOP
```

```
STOP:
        ldi temp, 0x94          ;запуск формирования
        out TWCR, temp          ;условия STOP
        ret                     ;возврат из подпрограммы

;подпрограмма передачи байта
SEND_B:
        out TWDR, byte          ;запись в регистр TWDR передаваемого байта
        ldi temp, 0x84          ;запуск процесса
        out TWCR, temp          ;передачи байта
m_sb:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_sb               ;и, если нет, то переход на метку m_sb
        ret                     ;возврат из подпрограммы

;подпрограмма приема байта с формированием подтверждения (ACK)
REC_ACK:
        ldi temp, 0xC4          ;запуск процесса
        out TWCR, temp          ;приема байта
m_ra:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_ra               ;и, если нет, то переход на метку m_ra
        in byte, TWDR           ;сохранение принятого байта в регистре byte
        ret                     ;возврат из подпрограммы

;подпрограмма приема байта без формирования подтверждения (NACK)
REC_NACK:
        ldi temp, 0x84          ;запуск процесса
        out TWCR, temp          ;приема байта
m_rna:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_rna              ;и, если нет, то переход на метку m_rna
        in byte, TWDR           ;сохранение принятого байта в регистре byte
        ret                     ;возврат из подпрограммы
```

§ 5.4.2. Пример программы ведомого микроконтроллера

В качестве примера рассмотрим программу, которая будет осуществлять обмен информацией с ведущим устройством так, как это делали бы часы реального времени MK41T56. Предполагаем, что ведущее устройство работает по алгоритму, описанному в § 5.4.1, тогда ведомое устройство должно реализовывать алгоритм, приведенный на рис. 5.19.

Алгоритмы подпрограмм, осуществляющих прием и передачу байта, приведены в § 5.3.

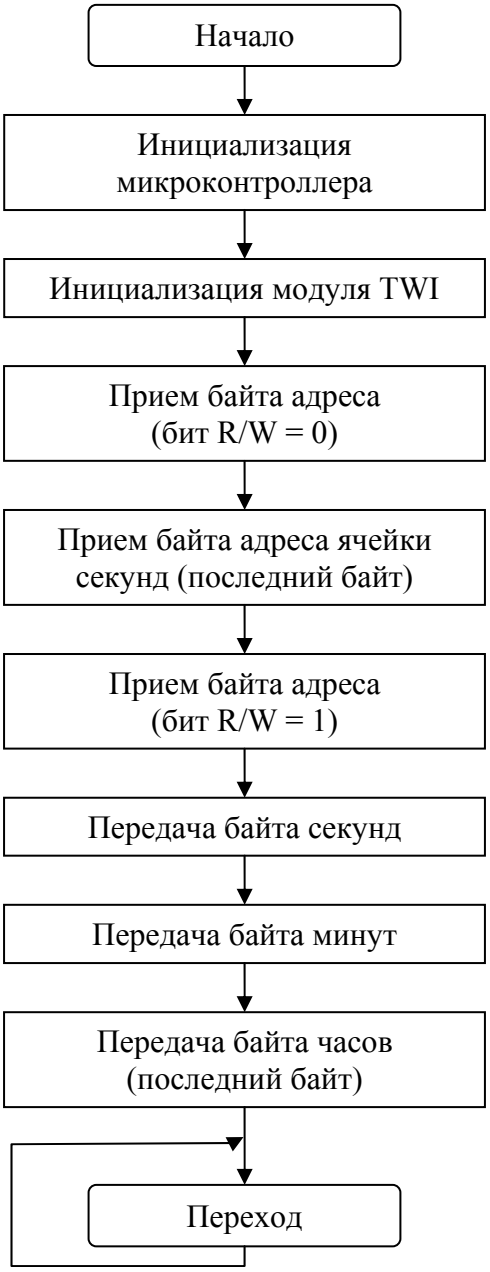


Рис. 5.19. Алгоритм программы ведомого микроконтроллера

Ниже приведена программа, реализующая описанный алгоритм. Программа написана для микроконтроллера ATmega16.

```
.INCLUDE «m16def.inc»
.DEVICE ATmega16

.DEF temp = r16
.DEF byte = r17

.EQU SECONDS = 0x28          ;значение, которое предается, как число секунд
.EQU MINUTES = 0x46          ;значение, которое предается, как число минут
.EQU HOURS = 0x11            ;значение, которое предается, как число часов
```

```
.EQU ADR_W = 0xD0          ;адрес ведомого (без поддержки общего вызова)

.CSEG

;-------------------------------------------------------------------------------
;инициализация
;-------------------------------------------------------------------------------

;инициализация стека
        ldi r16, low(RAMEND)
        out SPL, r16
        ldi r16, high(RAMEND)
        out SPH, r16

;настройка модуля TWI
        ldi temp, 0x00          ;выключение
        out TWCR, temp          ;модуля TWI

        ldi temp, 0xFC          ;настройка линий
        out DDRC, temp          ;SCL и SDA порта C
        out PORTC, temp         ;на ввод

        ldi temp, ADR_W         ;присвоение адреса
        out TWAR, temp          ;ведомому

;-------------------------------------------------------------------------------
;основная программа
;-------------------------------------------------------------------------------

        call REC_B              ;прием байта адреса (бит R/W = 0)

        call REC_LB             ;прием байта адреса ячейки секунд (последний байт)

        call REC_B              ;прием байта адреса (бит R/W = 1)

        ldi byte, SECONDS       ;передача байта
        call SEND_B             ;секунд

        ldi byte, MINUTES       ;передача байта
        call SEND_B             ;минут


        ldi byte, HOURS         ;передача байта
        call SEND_LB            ;часов

loop:
        jmp loop                ;переход на метку loop

;=========================================
;подпрограммы работы с модулем TWI
;=========================================

;подпрограмма приема байта
REC_B:
        ldi temp, 0xC4          ;запуск процесса
```

```
        out TWCR, temp          ;приема байта
m_rb:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_rb               ;и, если нет, то переход на метку m_rb
        in byte, TWDR           ;сохранение принятого байта в регистре byte
        ret                     ;возврат из подпрограммы

;подпрограмма приема последнего байта
REC_LB:
        ldi temp, 0xC4          ;запуск процесса
        out TWCR, temp          ;приема байта
m_rlb:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_rlb              ;и, если нет, то переход на метку m_rlb
        in byte, TWDR           ;сохранение принятого байта в регистре byte

        ldi temp, 0x84          ;освобождение
        out TWCR, temp          ;линий SCL и SDA
m_rlb2:
        in temp, TWCR           ;ожидание
        sbrs temp, TWINT        ;формирования
        rjmp m_rlb2             ;условия STOP
        ret                     ;возврат из подпрограммы

;подпрограмма передачи байта
SEND_B:
        out TWDR, byte          ;запись в регистр TWDR передаваемого байта
        ldi temp, 0xC4          ;запуск процесса
        out TWCR, temp          ;передачи байта
m_sb:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_sb               ;и, если нет, то переход на метку m_sb
        ret                     ;возврат из подпрограммы

;подпрограмма передачи последнего байта
SEND_LB:
        out TWDR, byte          ;запись в регистр TWDR передаваемого байта
        ldi temp, 0x84          ;запуск процесса
        out TWCR, temp          ;передачи байта
m_slb:
        in temp, TWCR           ;проверка, установился ли флаг
        sbrs temp, TWINT        ;прерывания от модуля TWI
        rjmp m_slb              ;и, если нет, то переход на метку m_slb
        ret                     ;возврат из подпрограммы
```

СПИСОК ЛИТЕРАТУРЫ

1. The I2C-BUS specification. Version 2.1, Philips Semiconductors, 2000 January.
2. 24LC64 64K I2C CMOS Serial EEPROM. Microchip Inc, 1998.
3. DS1621 Digital Thermometer and Thermostat. Dallas Semiconductor, 090905.
4. MK41T56 512 bit (64b x 8) Serial Access TIMEKEEPER SRAM. ST, November 2000.
5. C8051F060/1/2/3/4/5/6/7 Mixed Signal ISP FLASH MCU Family. Silicon Laboratories.
6. Atmel 8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash ATmega16, ATmega16L. 2466J-AVR-10/04.
7. MPLAB ICD USER'S GUIDE. DS51184D Microchip Technology Inc, 2000.
8. PIC16F87X Однокристальные 8-разрядные FLASH CMOS микроконтроллеры компании Microchip Incorporated. – М.: ООО «МикроЧип», 2002.
9. Семенов Б.Ю. Шина I2C в радиотехнических конструкциях. – М.: СОЛОН-Р, 2002
10. О. Николайчук x51-совместимые микроконтроллеры фирмы Cygnal. – М.: ООО «ИД СКИМЕН», 2002. – 472 с.
11. www.gaw.ru
12. www.microchip.com

Учебное издание

ВОРОБЬЕВА Галина Степановна
СЕЛЕЗНЕВ Антон Иванович

ИНТЕРФЕЙСЫ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Учебное пособие

Научный редактор
доктор технических наук,
профессор *Г.С. Евтушенко*

Издано в авторской редакции

Компьютерная верстка *К.С. Чечельницкая*
Дизайн обложки *О.Ю. Аршинова*

Подписано к печати 28.09.2011. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 11,05. Уч.-изд. л. 9,99.
Заказ ___-11. Тираж 35 экз.

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту BS EN ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru