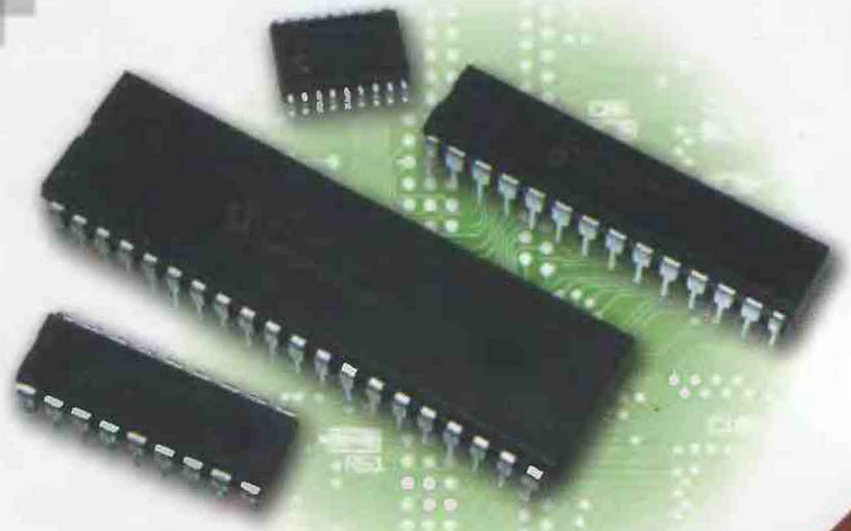
ПРАКТИКА ИНЖЕНЕРНОЙ ЭЛЕКТРОНИКИ

Анна и Манфред Кёниг

Полное руководство по PIC микроконтроллерам PIC18, PIC10F, rfPIC



+CD

МК-ПРЕСС

Анна и Манфред Кёниг

Полное руководство по PIC-микроконтроллерам

PIC18, PIC10F, rfPIC

Перевод с немецкого:
В. И. Кириченко, Ю. А. Шпак

"МК-Пресс"
Киев, 2007

Содержание

Предисловие

Ежегодно на рынке появляются десятки новых типов микроконтроллеров. Также и в сфере периферии уже нельзя довольствоваться только испытанными, давно известными устройствами.

В напряженном распорядке дня разработчика зачастую не остается свободного времени на поиск информации — именно поэтому авторы и решили написать эту книгу. При этом преследовалась цель отобрать из огромного объема сведений только самые важные, и затем упорядочить их. Само собой разумеется, в одну книгу нельзя вместить всю существующую информацию, тем более, что она постоянно изменяется.

По многочисленным просьбам, вводную главу мы посвятили теоретическим основам, чтобы в новых проектах мог разобраться даже новичок. Тем не менее, эта книга не рассчитана на начинающих, поскольку авторы изначально предполагали, что читатель имеет фундаментальные познания в электронике и базовые — в сфере обработки данных.

ГЛАВА

1 Основы

О микроконтроллерах PIC уже накопилось столько информации, что ею можно было бы заполнить тысячи страниц. Мы попытаемся изложить только основополагающие факты, объяснить их и при этом поделиться собственным опытом.

В частности, наша задача — дать беглый обзор большого объема разнообразных сведений. Для того чтобы не перегружать главу излишними деталями, мы в ней обсудим только фундаментальные особенности микроконтроллеров PIC, которые можно обнаружить уже у первых двух поколениях PIC, хотя они образуют также и базу для PIC18.

Новые и дополнительные свойства микроконтроллеров PIC будут рассмотрены в отдельной главе "PIC18". Авторы решили, что лучше всего будет представить семейство PIC18 как логическое продолжение предыдущих серий PIC. Несколько раз новшества PIC18 кратко упоминаются уже во вступительной главе.

В былые времена различные представители микроконтроллеров PIC одного поколения разнились между собой, в основном, объемом памяти, количеством выводов и типом корпуса. Между тем, существует много функций, которые у ряда производных устройств реализованы по-разному.

Различные типы микроконтроллеров могут отличаться множеством мелочей. Так, к примеру, во многих случаях мы не будем останавливаться на том, какие специальные разряды находятся в каких регистрах. Компания Microchip хорошо потрудилась над тем, чтобы сохранить совместимость "снизу вверх". Каждый опытный разработчик знает как это трудно.

Таким образом, эта книга не может быть заменой литературе от Microchip. Мы настоятельно рекомендуем каждому пользователю микроконтроллеров PIC читать справочные руководства, содержащие немало интересных советов. Особенности конкретного устройства необходимо выяснять по соответствующим техническим описаниям, даже если их не всегда комфортно читать. Кроме того, существует бесчисленное множество полезных рекомендаций для особых случаев применения (так называемые "Application Notes").

Для выбора типа микроконтроллера PIC с определенными свойствами существует **Product Selector Guide** (руководство по выбору продукта). Некоторые из функций PIC18 теоретически могут быть реализованы даже в представителях среднего подсемейства микроконтроллеров PIC, поэтому перед тем как приступить к новому проекту следует всегда изучать самую свежую информацию.

Поскольку при выборе устройства также учитывается и цена, очень важна возможность беспрепятственного ознакомления с состоянием цен. С некоторого времени это осуществимо с помощью раздела Buy домашней Web-страницы компании Microchip (`www.microchip.com`).

С точки зрения пользователя, необходимыми компонентами каждого микроконтроллера являются:

- центральный процессор;
- память данных и память программ;
- тактовый генератор и таймер;
- порты ввода-вывода;
- схема сброса.

Можно также назвать компоненты, от которых отказались в первых микроконтроллерах PIC:

- схема прерываний;
- аппаратные модули, работающие без участия центрального процессора.

Разумеется, существуют и другие компоненты, однако они доступны пользователю лишь косвенно (например, регистр стека).

Что сразу же впечатлило авторов при первом знакомстве с микроконтроллерами PIC, так это базовая концепция архитектуры и принципа функционирования.

Со времен первого поколения, устройства PIC значительно "выросли" — причем не только в размере, но и в интеллектуальности технологии. В первую очередь, PIC занимают особое место в мире микроконтроллеров, благодаря множеству хорошо продуманных аппаратных модулей.

1.1. Архитектура и принцип функционирования

Что касается архитектуры и структуры команд, то базовая концепция микроконтроллеров PIC особо не изменились.

- **Память данных и память программ разделены.** Особенностью микроконтроллеров PIC является то, что память программ и память данных разделены (Гарвардская архитектура). Эта архитектура — основа большинства других особенностей принципа функционирования.

  Присутствует шина данных, которая во всех микроконтроллерах PIC имеет разрядность 8 бит и разделена с шиной адреса, которая соединяет центральный процессор с памятью программ. В результате процессор в состоянии одновременно выполнять доступ к данным и к словам команд.

- **Каждая команда имеет разрядность слова (за исключением PIC18).** Гарвардская архитектура допускает, чтобы разрядность ячеек в памяти программ не зависела от разрядности ячеек в памяти данных. Вследствие этого возможно выбрать ячейки памяти программ такого размера, чтобы для каждой команды требовалось только одно обращение к шине адреса. В одном слове команды содержится совокупная информация о коде команды и аргументах.

  Для этого необходим хорошо продуманный, сокращенный набор команд. В результате микроконтроллеры PIC становятся очень быстродействующими и простыми в обслуживании, хотя существуют также ограничения и недостатки, о которых мы говорим далее, когда речь пойдет о системе команд.

- **Конвейер команд.** Гарвардская архитектура, наряду с применением команд в одно слово, позволяет воспользоваться особым "трюком" под названием "конвейеризация". Одновременно с выполнением команды центральный процессор выбирает из памяти программ следующую команду. Этим обработка ускоряется почти в два раза при той же тактовой частоте.

Только в том случае, когда обрабатываемая в данный момент команда является командой перехода, подготовленная команда отбрасывается и выбирается команда по новому адресу перехода. Таким образом, длительность команд перехода больше на один командный цикл (для команд условного перехода — только если переход выполняется).

- **Разрядность команд: 12, 14 и 16 бит**. Разрядность памяти программ в первом поколении микроконтроллеров PIC составляет 12 бит. Эти устройства, называемые компанией Microchip "базовой серией" (Base-line), — быстрые, хотя, с современной точки зрения, довольно "спартанские". Впрочем они с успехом используются и по сей день.

  В последнее время на рынке появились новые интересные представители базовой серии. Само собой разумеется, они поддерживают технологию Flash (например, крошечные PIC10F2xx с шестью или большие PIC16F59 с 40 выводами).

  Среднее подсемейство (Mid-range) с разрядностью памяти программ 14 бит предоставляет уже заметно больше комфорта. Удлинение слова команды используется преимущественно в пользу расширения диапазона адресов, поэтому сам набор команд в сравнении с базовой серией микроконтроллеров PIC не изменился. Большой объем адресуемой памяти данных — основа для модулей аппаратного обеспечения среднего подсемейства PIC.

  Наиболее современное старшее подсемейство (Enhanced) имеет 16-тиразрядные слова команд, которые используются не только для дальнейшего расширения адресного пространства, но также и для увеличения набора команд. В системе команд этого подсемейства PIC присутствует также несколько команд длиной в два слова, однако, благодаря ухищренной кодировке, их обработка, по сути, не меняется.

Старшее подсемейство PIC обычно обозначают как "PIC18", поскольку данное обозначение присутствует во всех предыдущих микроконтроллерах PIC этого поколения. Поскольку в этом поколении появились некоторые структурные изменения и технологические новшества, ему посвящена отдельная глава книги.

Хотелось бы упомянуть об еще одной ветке старшего подсемейства PIC — "High-End", которое обозначается как "PIC17". Эти микроконтроллеры уже присутствовали некоторое время на рынке перед Enhanced PIC. Хотя PIC17 и отличаются в некоторых аспектах от других семейств, с ними нетрудно разобраться с помощью технических описаний.

1.2. Структура команды

Следующее описание структуры команд касается команд длиной в одно слово. Особенности Enhanced PIC описываются в главе "PIC18".

Каждая команда состоит из двух компонентов: кода операции и аргументов. Код операции определяет, какую операцию необходимо выполнять. Аргументы могут быть константами, адресами или прочими параметрами. Оба компонента заключены в одном слове команды, разряды которого используются оптимально, поскольку отсутствует выделение строго определенного количества разрядов под код операции и аргументы. Так, в некоторых командах аргументы очень длинные, в то время как в других командах их вообще нет. В некоторых случаях даже тяжело сказать, какие разряды относятся к коду операции, а какие — к аргументу.

Наибольший класс команд образуют **логические и арифметические операции с аргументом данных и рабочим регистром W**. Команд с двумя аргументами данных не существует, в чем заключается один из недостатков сжатой структуры команды.

Этот класс команд имеет следующую структуру:

12 разрядов	OOOOO,D,FFFFF
14 разрядов	OOOOO,D,FFFFFFF
16 разрядов	OOOOOO,D,FFFFFFFF

При этом "O" обозначает код операции, "D" — селектор регистра назначения, а "F" — разряды указателя на аргумент данных.

Очень удобно в этих командах то, что пунктом назначения для результата операции может быть, на выбор, регистр W или указанный в аргументе файловый регистр.

В языке ассемблера, поддерживаемом компанией Microchip, с которым пользователь, наверняка, имел дело, если не предпочитает язык высокого уровня, такая команда записывается следующим образом:

```
ADDWF    WERT,W
```

Эта команда выполняет сложение содержимого файлового регистра под именем WERT и содержимого рабочего регистра W. **Регистр назначения — W**. Аргумент D в слове команды в этом случае устанавливается равным 0. Иногда в команде ассемблера встречают также запись в виде "0" вместо "W".

Если в качестве регистра назначения выступает WERT, тогда записывают:

```
ADDWF    WERT,F
```

или по-другому

```
ADDWF    WERT
```

Аргумент D в слове команды в этом случае устанавливается равным 1. Иногда в команде ассемблера используют также тип записи "1" вместо "F". Если регистр назначения явно не указан, то в его качестве автоматически выбирается файловый регистр (D=1).

Авторы предпочитают последний, выделенный полужирным шрифтом тип записи, при котором аргумент отбрасывается, поскольку он более наглядный.

1.3. Набор команд

В системе команд микроконтроллеров PIC среднего подсемейства всего 35 кодов операции, где есть все, что требуется для программирования (табл. 1.1 – 1.3). В базовой серии PIC используется почти такой же набор команд. Хотя для удобства программирования было бы неплохо получить еще несколько дополнительных команд, существующих команд достаточно для решения любых задач.

Таблица 1.1. Команды с файловым регистром N в качестве аргумента

Команда + операнды	Операция	Флаги состояния	Примечание
ADDWF F,D	D = W + F	CY,DC,ZR	
ANDWF F,D	D = W AND F	ZR	
CLRF F	F = 0	ZR	
CLRW	W = 0	ZR	

Таблица 1.1. Окончание

Команда + операнды	*Операция*	*Флаги состояния*	*Примечание*
COMF F,D	D = NOT F	ZR	
DECF F,D	D = F – 1	ZR	
DECFSZ F,D	D = F – 1, пропускаем следующую команду, если ZR = 1	Нет	
INCF F,D	D = F + 1	ZR	
INCFSZ F,D	D = F + 1, пропускаем следующую команду, если ZR = 1	Нет	
IORWF F,D	D = W OR F	ZR	
MOVF F,D	D = F	ZR	
MOVWF F	F = W	Нет	
NOP	Нет операции	Нет	
RLF F,D	Циклический сдвиг влево через CY	CY	
RRF F,D	Циклический сдвиг вправо через CY	CY	
SUBWF F,D	D = F – W	CY, DC, ZR	
SWAPF F,D	D = F, переставить полубайты	Нет	
XORWF F,D	D = W XOR F	ZF	

Таблица 1.2. Команды с адресами разрядов в качестве аргументов; B = 0...7

Команда + операнды	*Операция*	*Флаги состояния*	*Примечание*
BCF F,B	Очистить F(B)	Нет	
BSF F,B	Установить F(B)	Нет	
BTFSC F,B	Пропустить следующую команду, если F(B) = 0	Нет	
BTFSS F,B	Пропустить следующую команду, если F(B) = 1	Нет	

Таблица 1.3. Команды с константами, команды перехода и другие

Команда + операнды	*Операция*	*Флаги состояния*	*Примечание*
ADDLW K	W = W + K	CY, DC, ZR	только PIC16CXX
ANDLW K	W = W AND K	ZR	
CALL ADR	Вызов подпрограммы	Нет	
CLRWDT	Сброс сторожевого таймера	TO, PD	
GOTO ADR	Переход по адресу	Нет	
IORLW K	W = W OR K	ZF	
MOWLW K	W = K	Нет	
RETFIE	Выход из подпрограммы обработки прерывания	GIE	только PIC16CXX
RETLW K	Возврат с W = K	Нет	
RETURN	Возврат	Нет	только PIC16CXX
SLEEP	Переход в "спящий" режим	TO, PD	
SUBLW K	W = K – W	CY, DC, ZR	только PIC16CXX
TRIS F	Регистр TRIS = W		только PIC16CXX
XORLW K	W = W XOR K	ZR	

Команды очень хорошо документированы в технических описаниях, где можно найти даже коды операций.

В описании команд следует всегда обращать внимание на колонку, в которой отмечены флаги, изменяемые в результате выполнения той или иной команды. Флаги находятся в регистре состояния STATUS, в котором также присутствуют и некоторые дополнительные разряды.

В микроконтроллерах PIC базовой серии и среднего подсемейства пристствуют только три флага: ZR, CY и DC. В документации Microchip флаги ZR и CY обозначаются соответственно Z и C. Авторы находят это неудобным и потому используют в книге собственный способ записи.

Внимание!

При операциях вычитания флаг CY устанавливается, даже если не было переполнения. По команде MOVF устанавливается флаг ZR (хотя это и непривычно, зато практично ввиду отсутствия команды TEST).

1.4. Память данных

Для адресации файловых регистров в распоряжении имеется ограниченное число разрядов в словах команд (в базовой серии PIC — 5 разрядов, с помощью которых можно адресовать 32 файловых регистра). Файловые регистры от 0 до 7 применяются для специальных целей. Их называют Special Function Register (SFR) — регистрами специального назначения. Таким образом, для использования в качестве пользовательских переменных остается еще 24 разряда.

В среднем подсемействе микроконтроллеров PIC используется 7 разрядов, которых достаточно уже для адресации 128 файловых регистров. Файловые регистры от 0 до 31 — регистры специального назначения (SFR), так что для переменных остается 96 байт.

Именно в малом размере адресного пространства заключается главный недостаток сжатых команд. Хотя большинство микросхем предоставляют до 4 банков по 32 и 128 файловых регистров, операции по переключению банков не совместимы с удобством программирования. При каждом обращении к файловому регистру необходимо обращать внимание на то, правильно ли выбран банк.

Разряды для выбора банка расположены в FSR для базовой серии и в регистре STATUS для среднего подсемейства микроконтроллеров PIC. С некоторого времени MPASM позволяет выбирать банк с помощью ассемблерного макроса:

```
BANKSEL     WERT        ;WERT - пример любой переменной
```

По этому макросу ассемблер вставляет команды, принадлежащие определенному типу микроконтроллеров, для выбора банка, в котором находится переменная WERT.

Основной способ избежать ошибок — помещать переключение банка перед каждым доступом к памяти данных. Тем не менее, несмотря на всю надежность такого подхода, он делает программу большой и не удобочитаемой. Такой способ используют компиляторы с языков высокого уровня.

При программировании на ассемблере многие разработчики применяют другой метод. В качестве основного состояния задается банк 0, и переключение выполняется только при доступе к другим банкам. Для того, чтобы этот метод был действительно эффективен, необходимо хорошо продумывать структуру данных — особенно, если реализованы подпрограммы, не зависящие от главной программы.

Для упрощения этой процедуры компанией Microchip при разработке микроконтроллеров PIC был принят ряд мер. Так, в PIC с четырьмя банками присутствуют области адресов, расположенные друг над другом, к которым можно обращаться независимо от выбора банка (так называемая "область доступа"). Было бы бессмысленно дополнительно перед каждым доступом к переменной выполнять переключение банка. В этой области доступа располагают, например, рабочие переменные стандартных подпрограмм. Наиболее важные регистры специального назначения, к которым часто обращаются (рабочие регистры), обычно располагаются в банке 0. Регистры специального назначения, расположенные в банке 1, используются, преимущественно, для инициализации.

1.5. Косвенная адресация данных

Косвенная адресация подразумевает, что указатель на аргумент размещен не явно в аргументе, а в регистре-указателе. В базовой серии, а также в среднем подсемействе микроконтроллеров PIC этот регистр называется FSR (File Select Register — регистр выбора файла). Команды с косвенной адресацией не требуют аргумента, поскольку он загружается заранее в FSR.

Без косвенной адресации микроконтроллер немыслим. Поскольку код операции имеет ограниченный размер, в PIC применили следующий трюк: команды для косвенной адресации имеют такой же код операции, что и команды для прямой адресации. Указание на то, что речь идет о косвенной адресации, содержится в аргументе.

Это означает, что должен задаваться какой-то особенный аргумент, который не является "правильным" аргументом данных. В базовой серии и среднем подсемействе микроконтроллеров PIC это — адрес 0, следовательно, с таким адресом не может быть задан физический файловый регистр. Таким образом, файловый регистр с адресом 0, — это символический регистр, который в ассемблере обозначается именем `INDF` (косвенный файловый регистр).

Кроме того, в языке ассемблера нет никакого дополнительного кода операции для косвенной адресации, например:

```
ADDWF   INDF,W
```

В старшем подсемействе микроконтроллеров PIC присутствует три регистра FSR, что повышает удобство программирования. Более подробно об этом сказано в главе "PIC18".

1.6. Память программ

Следующее описание памяти программ касается исключительно базовой серии и среднего подсемейства микроконтроллеров PIC (см. также главу "PIC18").

Указатель на слово программы называют **счетчиком команд**. Центральный процессор всегда выполняет ту команду, на которую указывает этот счетчик, после чего увеличивает его содержимое, если речь не идет о команде перехода.

Младший байт счетчика команд — адресуемый регистр специального назначения (PCL). Регистр PCL изменяют в случае реализации косвенных программных переходов. Однако старший байт счетчика команд умышленно сделан недоступным напрямую. Если необходимо изменить старший байт программного указателя, то его нужно записывать в промежуточный буфер памяти — регистр PCLATH. Регистр PCLATH копируется в старший байт счетчика команд только при доступе к регист-

ру PCL — будь то по команде перехода или по арифметической команде с аргументом PCL.

Для программного перехода по адресу ADRH:ADRL используют, как правило, команды CALL и GOTO, которым в качестве аргумента передается адрес. При этом существуют трудности, аналогичные тем, которые возникают при адресации памяти данных.

Базовая серия и среднее подсемейство микроконтроллеров PIC предоставляют в распоряжение 11 разрядов для адреса назначения, чтобы охватить все 2К слов программы. Количество в 2К слов обозначается как "страница".

В базовой серии PIC команда GOTO имеет лишь 9 разрядов для адреса назначения, а команда CALL — даже 8. Иногда для адресации требуется немного ловкости, поскольку необходимо экономно обращаться с имеющейся памятью.

Подобно тому, как это происходит при выборе банка, при адресации памяти программ перед каждой командой перехода должна предварительно выбираться корректная страница (если этого еще не сделано). В среднем подсемействе PIC это реализовано путем правильной установки регистра PCLATH.

При этом перед командами CALL и GOTO необходимо позаботиться только о разрядах 3 и 4 регистра PCLATH. Младшие три разряда затираются адресом перехода, а старшие три не используются, поскольку на данный момент не существует микроконтроллеров PIC среднего подсемейства с более, чем четырьмя страницами.

Наиболее практичный способ выбора страницы перед выполнением команд CALL LABEL и GOTO LABEL:

```
MOVLW     HIGH_LABEL
MOVWF     PCLATH
```

Измененное значение PCLATH копируется в счетчик команд только по следующим командам CALL и GOTO.

При программировании на ассемблере мы обходимся с управлением страницами подобно тому, как это происходит для памяти данных. Отдельные страницы содержат, по возможности, законченные программные модули. При таком подходе переходы с одной страницы на другую должны использоваться как можно реже. Таблицы и списки размещаются, если необходимо, на собственной странице.

Такие часто используемые служебные программы как, например, умножение и деление, можно скопировать в каждую страницу, в которой они используются. Для этого требуется меньше места, чем в том случае, когда перед каждым вызовом выполняются команды выбора страницы.

Такой способ организации программы требует особой тщательности, что идет на пользу не только программе, но, прежде всего, — программисту. Исходный файл становится более наглядным и удобочитаемым.

Еще один способ реализации переходов — прямая запись в счетчик команд. Такой подход применяют всегда, когда хотят перейти по переменному адресу.

Для программного перехода по адресу ADRH:ADRL используют следующие команды:

```
MOVF    ADRH,W
MOVWF   PCLATH
MOVF    ADRL,W
MOVWF   PCL
```

Содержимое регистра PCLATH загружается в старший байт счетчика команд только последней командой, по которой происходит переход по новому адресу.

Подобный прямой доступ к счетчику команд называется "вычисленным GOTO" и применяется для переменных ветвлений.

Предположим, байтовая переменная FALL может принимать значения от 0 до 5. Каждому значению FALL поставлена в соответствие некоторая служебная подпрограмма (PROG0, PROG1, ...). Возможны различные способы ветвления, и один из них выглядит следующим образом:

```
MOVF    FALL,W
ADDWF   PCL
GOTO    PROG0
GOTO    PROG1
GOTO    PROG2
GOTO    PROG3
GOTO    PROG4
GOTO    PROG5
```

Перед вызовом этих строк программы следует проверить, действительно ли переменная FALL меньше или равна 5, иначе выполнение команды ADDWF PCL может привести к переходу в непредвиденную область памяти.

Внимание!

В случае рассмотренного выше ветвления необходимо удостовериться, что при выполнении команды ADDWF PCL не происходит переполнение.

Важным примером представленного выше варианта ветвления является следующая подпрограмма:

```
GETWERT     MOVF    TABPTR,W
            ADDWF   PCL
            RETLW   WERT0
            RETLW   WERT1
            RETLW   WERT2
            RETLW   WERT3
            RETLW   WERT4
            RETLW   WERT5
            и т. д.
```

С помощью команды CALL GETWERT в регистр W можно загрузить табличное значение, соответствующее указателю TABPTR.

1.7. Порты ввода-вывода

Выводы портов связаны с регистрами портов и сгруппированы по восемь и меньше. Каждый вывод порта может включаться как выход или как вход (третье состояние).

Регистры портов обозначаются, как правило, PORTA, PORTB, PORTC и т. д. Количество портов разнообразно. У самых маленьких микроконтроллеров PIC, оснащенных всего лишь несколькими выводами портов, выходной регистр называется GPIO.

Каждому регистру PORT поставлен в соответствие регистр направления передачи данных, который поразрядно определяет, какие выводы (по номеру разряда)

ются входами (1), а какие – выходами (0). Это можно легко запомнить по за-ным буквам английского алфавита (1: Input — вход, 0: Output — выход).

Регистры направления передачи данных обозначаются как "TRIS", т.е. TRISA, TRISB и т. д. Регистр TRIS включает или выключает связанные с выводами выход-усилители-формирователи.

Когда значение W записывают в регистр PORT (командой MOVWF), это значение ается на соответствующие выводы, если соответствующие им выходные усили-формирователи включены.

По команде чтения из регистра PORT те-ее физическое состояние выводов порта ывается в регистр PORT. Таким образом, команде MOVF PORTB,W необязательно ывается именно то значение, которое бы-аписано в регистр PORTB.

Если выходной усилитель формирователь чен, то, как правило, исходят из того, что анное значение появляется также и на вы-де, однако есть случаи, в которых это не так. Например, светодиод включается по ко-анде BCF LED, и может случиться, что све-диод так сильно нагружает порт, что при росе вывода считывается состояние 1 (рис. 1.1).

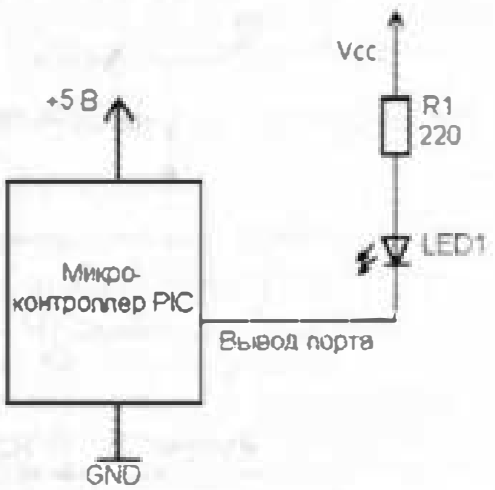


Рис. 1.1. Нагрузка вывода светодиодом

Команды типа "чтение/модификация/запись"

Каждая команда, для которой в качестве аргумента указан регистр порта, явля-ся командой типа "чтение/модификация/запись" (за исключением команд MOVWF MOVF). Например, по команде BCF PORTB, 0 считывается весь PORTB, обнуляется разряд 0 и затем измененное таким образом значение записывается обратно в ре-истр PORTB. Если бы вышеназванный светодиод был подключен, например, к вы-оду 7 порта B, то выключить его можно было бы по команде BCF PORTB, 0.

Для того чтобы понять принцип функционирования портов, рекомендуем изу-ить соответствующие блок-схемы в технических описаниях (рис. 1.2).

Отдельные выводы порта имеют различающиеся входные характеристики. Если спользуют свойства триггера Шмитта, то на это следует обращать внимание.

В среднем подсемействе микроконтроллеров PIC многие выводы портов имеют пециальные дополнительные возможности для поддержки аппаратных модулей. За информацией о распределении назначения выводов всегда следует обращаться к ехнической документации.

Если вывод порта ведет себя странно, то это может объясняться следующими причинами:

- проблематика, связанная с чтением/модификацией/записью
- неправильно установлен соответствующий регистр TRIS;
- аппаратный модуль включен, переняв на себя работу вывода;
- вывод порта поврежден, например, по причине короткого замыкания (да, и такое случается!).

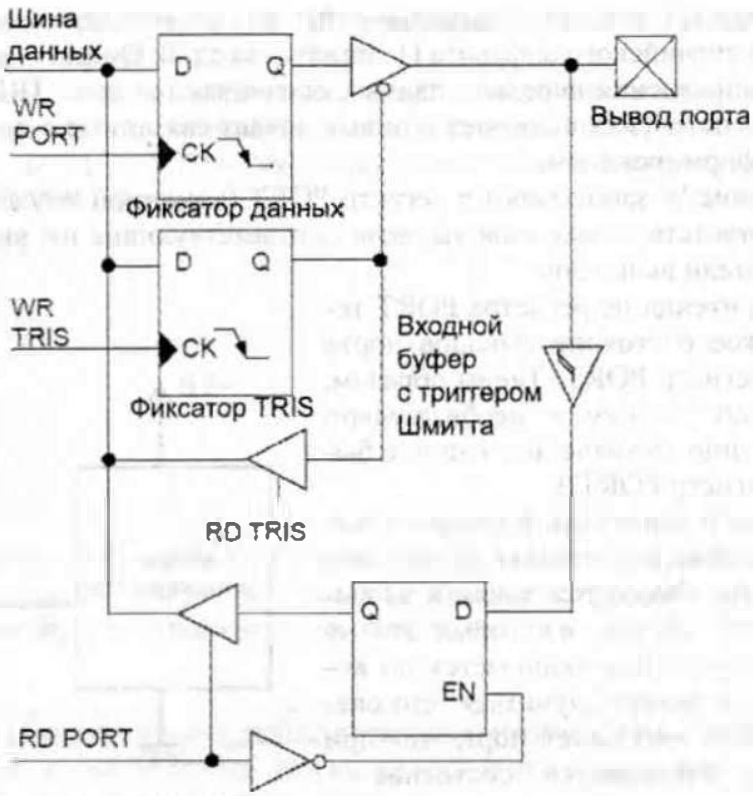


Рис. 1.2. Типичный порт ввода-вывода

1.8. Регистры специального назначения

Регистры специального назначения (SFR) образуют интерфейс между центральным процессором и аппаратной частью микроконтроллеров PIC. Они используются в качестве рабочих регистров, регистров конфигурации или для опроса состояния аппаратного обеспечения.

В среднем подсемействе микроконтроллеров PIC классическим является следующее расположение первых восьми файловых регистров (табл. 1.4).

Таблица 1.4. Классическое расположение первых восьми файловых регистров в среднем подсемействе микроконтроллеров PIC

Номер регистра	*Банк 0*	*Банк 1*
0	INDF	INDF
1	TMR0	OPTION
2	PCL	PCL
3	STATUS	STATUS
4	FSR	FSR
5	PORTA	TRISA
6	PORTB	TRISB
7	PORTC	TRISC

Описание отдельных регистров специального назначения можно найти в соответствующих разделах книги.

1.9. Прерывания

В базовой серии микроконтроллеров PIC прерывания не используются. В бы-
времена это означало необходимость творческого подхода в программирова-
Мы и по сей день пользуемся стратегиями, разработанными для реализации
ной многозадачности без прерываний (даже если они присутствуют).

Позже мы опишем структуру прерываний в среднем подсемействе микроконт-
леров PIC, которая по причине совместимости образует также основу структу-
рерываний в микроконтроллерах старшего подсемейства. Что касается старше-
дсемейства PIC, то с прерываниями в нем стало работать намного удобнее, о
подробно рассказано в главе "PIC18".

Первые три источника прерываний одинаковы у всех представителей среднего
емейства PIC:

- переполнение таймера 0 (флаг прерывания T0IF);
- внешние импульсы на выводе 0 порта B (флаг прерывания INT0IF);
- изменение в старшем полубайте порта B (флаг прерывания RBIF).

Все последующие прерывания называются "прерываниями периферии". Для
ждого аппаратного модуля имеется, по крайней мере, одно прерывание.

Каждому прерыванию соответствует флаг прерывания (флаг оповещения), ко-
рый, за некоторыми исключениями, должен сбрасываться программно.

Кроме того, для каждого прерывания присутствует разряд, с помощью которого
рерывание можно разрешить (англ.: enable) или запретить (англ.: disable).

Для первых трех прерываний эти разряды и флаги находятся в регистре
NTCON. Для прерываний периферии флаги находятся в регистрах **PIR1** и **PIR2** а
разряды разрешения — в регистрах **PIE1** и **PIE2**.

Кроме того, в регистре INTCON присутствуют два глобальных разряда разре-
шения. Сбросив разряд GIE, можно установить общий запрет прерываний. Для того
чтобы были разрешены прерывания периферии, кроме разряда GIE должен быть
также установлен разряд PEIE.

Структура регистра INTCON:

7	6	5	4	3	2	1	0
GIE	PEIE	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

Позиции разрядов для прерываний периферии следует сверять по соответст-
вующим техническим описаниям. К сожалению, эти позиции необходимо выяснять
самому, поскольку в MPASM не определены какие либо простые имена для соот-
ветствующих разрядов. Их можно определить самостоятельно с помощью директи-
вы `#define` во включаемом файле. Мы не рекомендуем изменять заголовочные
файлы, поскольку они могут обновляться при обновлении MPASM.

Важно!

Флаг устанавливается после поступления запроса на прерывание, даже если
оно запрещено. Таким образом, о возникновении прерывания можно узнать путем
опроса соответствующего флага.

Пример структуры прерываний показан на рис. 1.3

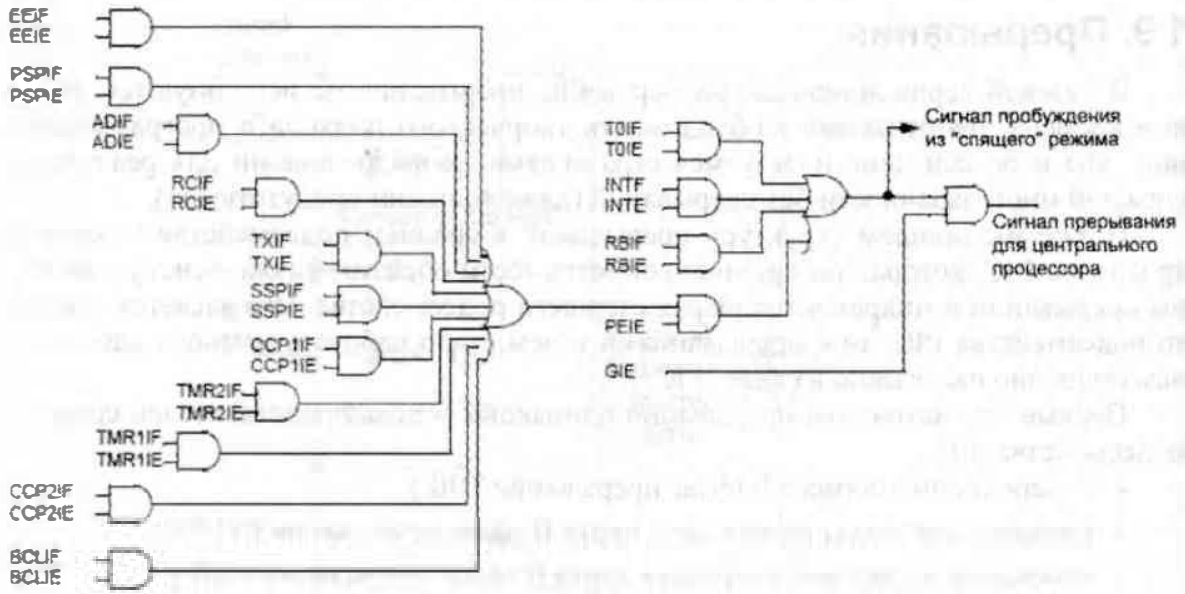


Рис. 1.3. Пример структуры прерываний

1.9.1. Обслуживание прерывания

Если некоторое прерывание разрешено, то после его возникновения текущее состояние счетчика команд сохраняется, после чего в него загружается адрес вектора прерывания. Время, которое необходимо для этого, называют **латентным периодом**. Оно составляет три командных цикла, если событие инициируется синхронно с тактами команд, иначе оно может длиться до четырех командных циклов. Подпрограмма обработки прерывания завершается по команде `RETFIE`, которая загружает сохраненный программный адрес обратно в счетчик команд. В среднем подсемействе микроконтроллеров PIC адрес прерывания имеет значение 4 и используется только один вектор прерывания. Управление приоритетностью отсутствует.

Программа обработки прерывания не может прерываться другим прерыванием. После вхождения в процедуру обработки прерывания автоматически сбрасывается разряд GIE, который опять устанавливается по команде `RETFIE`. По этой причине нет смысла сбрасывать разряд GIE внутри подпрограммы обработки прерывания.

1.9.2. Подпрограмма обработки прерываний

Подпрограмма обработки прерываний должна начинаться по адресу вектора прерывания (для среднего подсемейства микроконтроллеров PIC это адрес 4).

Каждая подпрограмма обработки прерывания вначале должна сохранить все переменные, которые она совместно использует с основной программой. К ним относятся, как правило, регистры W, STATUS и PCLATH, которые в среднем подсемействе PIC должны сохранятся программно, а восстанавливаться — перед командой `RETFIE`.

При сохранении и восстановлении используют регистр W, который, следовательно, необходимо сохранять первым, а восстанавливать последним.

Тот, кто еще не имеет опыта работы с микроконтроллерами PIC, скорее всего использовал бы для сохранения и восстановления регистров W и STATUS следующие последовательности команд:

Сохранение W и STATUS (неверный вариант):

```
MOVWF   W_STACK       ; Выбор банка???
MOVF    STATUS,W
MOVWF   S_STACK       ; Выбор банка???
...
```

Восстановление W и STATUS (неверный вариант):

```
MOVF    S_STACK,W     ; Выбор банка???
MOVWF   STATUS
MOVF    W_STACK,W     ; неверно: MOVF может изменить STATUS,ZR!!!
```

Эти последовательности команд содержат одну возможную и одну явную ошибку. Явная ошибка находится в последней строке. После того как восстановлен регистр состояния, далее не может следовать команда MOVF, так как она снова изменяет регистр STATUS. У этой проблемы есть простое решение: заменить команду MOVF двумя командами SWAPF (см. ниже).

Но вот со второй проблемой, порой, все далеко не так просто. При входе в подпрограмму обработки прерывания действительно зачастую не известно, какой банк выбран, поэтому обозначим его как "Банк X".

В микроконтроллерах PIC с четырьмя банками обычно используется "область доступа" (см. раздел 1.4, "Память данных"). Если переменные W_STACK и S_ STACK разместить в такой области, то проблема с выбором банка решена. Если же область доступа не существует, то проблему можно решить следующим образом.

Сохранение W и STATUS (правильно):

```
MOVWF   W_STACK       ; Регистр W сохраняется в банке X
MOVF    STATUS,W
BANK0                 ; Любой банк. BANK0 - в качестве примера
MOVWF   S_STACK       ; Регистр STATUS сохраняется в банке 0
...
```

Восстановление W и STATUS (правильно):

```
MOVF    S_STACK,W     ; Предполагается: пока еще выбран BANK0
MOVWF   STATUS        ; Выбран опять банк X
SWAPF   W_STACK       ; SWAPF не изменяет STATUS
SWAPF   W_STACK,W
```

Компания Microchip предлагает шаблоны программ для каждого типа микроконтроллеров PIC, в которых, кроме всего прочего, можно найти подпрограммы для сохранения и восстановления важных регистров. Они находятся в папке, в которой была установлена среда MPLAB. При стандартной установке этой папке соответствует путь C:\Programme\MLAB IDE\MCHIP_Tools\TEMPLATE\Code или же ..\Object, в зависимости от того, пишется ли относительный код или абсолютный.

Мы избегаем проблемы с выбором банка тем, что мы выбираем в качестве основного банк 0 и, встречая смены банка, запрещаем прерывания. Этим разгружается подпрограмма обработки прерываний, что может потребоваться при решении задач, критичных ко времени.

Используя описанный способ, можно также позволить себе прерывания с короткой обработкой, в которых отказываются от сохранения и восстановления. В та-

ком варианте обработки применяют только команды, не использующие ни регистр W, ни регистр STATUS. Пример: BSF, BCF, BTFSS, BTFSC. GOTO.

Интересным трюком является применение команд INCFSZ+NOP вместо INCF или DECFSZ+NOP вместо DECF, поскольку команды INCFSZ и DECFSZ не изменяют регистр STATUS. Эту особенность можно применять, например, при подсчете событий прерываний.

Не забывайте сбрасывать флаг обрабатываемого прерывания внутри подпрограммы обработки, поскольку в противном случае прерывание сразу же повторится снова.

1.10. Осциллятор

Каждый микроконтроллер для выполнения команд и функционирования аппаратных модулей нуждается в системной синхронизации. Эта синхронизация "дирижирует" выполнением команд. Для того чтобы выполнить одну команду, требуется цикл из четырех тактов, которые обозначаются как "Q1"..."Q4":

- Q1 — декодирование команды;
- Q2 — выборка аргумента данных (если необходимо);
- Q3 — выполнение операции;
- Q4 — сохранение аргумента данных (если необходимо).

Для выборки команды из памяти программ, как правило, не требуется дополнительное время, поскольку в любом случае следующая команда загружается еще во время выполнения текущей (конвейерная обработка).

Осциллятор, который производит тактовые импульсы для центрального процессора, называется "**главным**" (кроме него, существуют дополнительные осцилляторы, например, для сторожевого таймера или для АЦП).

Тактовые импульсы главного осциллятора называют **системными**. В примерах применения используется понятие "**командный такт**", частота которого составляет четвертую долю частоты системных тактовых импульсов.

Для формирования тактовых импульсов существует множество возможностей. Классическим способом является подключение к микроконтроллерам PIC электрических элементов (кварц, резонатор, RC-цепь). Если осциллятор строится из внешних элементов, то мы говорим для краткости о **внешнем осцилляторе**, что, по большому счету, некорректно. Тактовые импульсы можно производить также и внешним осциллятором (кварцевым). С тех пор как на рынке появились маленькие микроконтроллеры PIC (8 выводов), используются также внутренние RC-осцилляторы, для которых больше не нужно выделять выводы под подключение внешнего осциллятора.

В общем случае, можно сказать следующее: внешний кварцевый осциллятор — наиболее точный способ формирования системных тактовых импульсов, но при этом и наиболее дорогостоящий. Кварцевый осциллятор считается чувствительным к помехам. Наименьшую точность получают с помощью внутреннего RC-осциллятора, поскольку он — температурно-зависимый. Внутренний RC-осциллятор можно подстраивать.

У новых представителей PIC положение дел с внутренними осцилляторами значительное улучшилось. Если речь идет о точности или малом потреблении мощности, то свойствам осциллятора следует уделить особое внимание. Наиболее важные данные можно найти в "Selector Guide" (ранее "Line card").

Хорошим компромиссом в отношении точности и цены является применение в качестве внешнего осциллятора резонатора. Точность резонаторов лежит в пределах около 1%.

Что касается разработки осцилляторов, то компания Microchip предоставляет много примеров в этой сфере. Полезную информацию по этой теме можно также найти в технических описаниях.

Если есть подозрение, что микроконтроллер PIC не тактируется или частота тактовых импульсов не соответствует требуемой, то проверить это можно с помощью простого трюка. В самом начале программы помещается маленький тестовый цикл, в котором некоторый вывод попеременно переключают в состояние то высокого, то низкого уровня (при этом следует не забыть запрограммировать тестовый вывод как выход!). Затем по осциллографу наблюдают, правильно ли работает PIC.

Примечание

Измерения на выводе OSC1 ненадежны, поскольку измерительный щуп сбивает работу осциллятора.

Допустимая частота системной синхронизации не ограничена снизу и, теоретически, с помощью микроконтроллера PIC можно управлять ключом Морзе. Сверху частота ограничена значением 20 МГц, для PIC17 — 33 МГц, а для PIC18 — 40 МГц.

Чем выше тактовая частота, тем быстрее выполняется программа, однако мы вовсе не одобряем идею возмещать небрежное или неумелое программирование более быстрым тактированием. Высокая частота имеет ряд недостатков. Например, быстро повышается потребление тока, а с ним — и тепловыделение. Можно также столкнуться с проблемой электромагнитных помех.

Исходя из тактовой частоты, необходимо выбирать специальных представителей микроконтроллеров PIC, и, как правило, более быстрые также являются более дорогостоящими.

В большинстве случаев применения мы используем частоту системной синхронизации 8 МГц. Таким образом, одна команда длится ровно 1 мкс. Пересчет на другие частоты происходит с помощью простых правил. Между прочим, мы не раз опровергали утверждение, что определенные программы "невозможно" составить для микроконтроллеров PIC с частотой системной синхронизации 4 МГц.

1.11. Таймеры

Таймеры относятся к аппаратным модулям, поскольку они работают без непосредственного вмешательства центрального процессора. Таймеры могут функционировать как счетчики командных циклов или как счетчики фронтов внешних импульсов. Первая функция обозначается как "режим таймера", а вторая — как "режим счетчика".

Каждый таймер содержит счетный регистр (один или два байта) и регистр конфигурации. Все таймеры микроконтроллеров PIC используют предварительные делители частоты, которые инициализируются с помощью соответствующих регистров конфигурации. Содержимое таймера увеличивается на единицу только тогда, когда возникает переполнение в соответствующем регистре предварительного делителя.

Регистры предварительных делителей не адресуются — они недоступны для чтения/записи и обнуляются при каждом доступе к таймерам на запись. Для кри-

тичных по времени задач следует применять технику программирования, в которой минимизирован регулярный доступ к таймерам на запись.

В базовой серии микроконтроллеров PIC используется только один таймер (Timer0), а в среднем подсемействе — до трех таймеров (Timer0, Timer1, Timer2). Для всех таймеров присутствует прерывание по переполнению.

Кроме того, каждый микроконтроллер PIC располагает **сторожевым таймером**, который можно отключить или активировать с помощью регистра конфигурации.

Рассмотрим таймеры базовой серии и среднего подсемейства микроконтроллеров PIC. В старшем подсемействе появилась масса новшеств, о которых речь пойдет в главе "PIC18".

1.11.1. Timer0

Счетный регистр, соответствующий Timer0, называется TMR0. Регистр конфигурации, с помощью которого инициализируется предварительный делитель, — это регистр OPTION. В базовой серии PIC этот регистр не адресуется и недоступен для чтения, а запись в него можно осуществить только по специальной команде `OPTION`.

В среднем подсемействе PIC таймер Timer0 остался без изменений за исключением того факта, что регистр OPTION является адресуемым. Поскольку, из соображений совместимости, команда `OPTION` еще поддерживается ассемблером, этот регистр теперь называется OPTION_REG.

Хорошая новость: предварительный делитель Timer0 позволяет выбирать коэффициент деления до 256. Плохая новость: Timer0 должен делить предварительный делитель со сторожевым таймером (строго говоря, для сторожевого таймера он выполняет роль постделителя, что для пользователя несущественно). Назначение предварительного делителя требуется задавать с помощью разряда PSA в регистре OPTION. Без предварительного делителя таймер тактируется командным тактом.

Timer0 может также тактироваться, на выбор, фронтами импульсов на внешнем входе (T0CKI). В этом случае, с помощью особого разряда в регистре OPTION определяют, какие должны подсчитываться фронты: положительные или отрицательные.

1.11.2. Timer1

Timer1 — двухбайтный. Его счетные регистры называются TMR1H и TMR1L, а регистр конфигурации — T1CON. В отличие от Timer0, Timer1 можно активизировать с помощью особого разряда в регистре конфигурации (TMR1ON).

Если разряд TMR1CS в регистре T1CON содержит 0, то установлен режим таймера, если же этот разряд содержит 1, то Timer1 подсчитывает положительные фронты на входе T1CKI/T1OSO.

Разряд /T1SYNC должен устанавливаться в 0 для того, чтобы синхронизировать внешние такты с главным осциллятором. Этот разряд должен быть сброшен также в том случае, если счетчику необходимо работать даже при остановленном главном осцилляторе.

Таймер Timer1 особенен тем, что оснащен собственным осциллятором, который рассчитан специально на кварц с частотой 32 кГц, включенный между выводами T1OSI и T1OSO (максимум 200 кГц). Осциллятор включается установкой в 1 разряда T1OSCEN в регистре T1CON. Когда осциллятор включен, на выводах T1OSI и T1OSO всегда считывается 0. Значение TRIS в этом случае игнорируется.

С помощью регистра T1CON также выбирается делитель частоты. Для Timer1 его возможные значения составляют 1, 2, 4 и 8.

Таймер Timer1 является основой для двух важных аппаратных модулей: **фик-** **и сравнения** (описаны в подразделе 1.14.3).

1.11.3. Timer2

Счетный регистр таймера Timer2 называется TMR2 (1 байт), а соответствую- регистр конфигурации — T2CON.

Timer2 — единственный таймер, который автоматически устанавливается после достижения заранее заданного значения (при последующем инкремен- нии), которое должно быть записано в регистр PR2 (регистр периода). Если регистр имеет значение 0, то не удивляйтесь, что Timer2 кажется неработаю-

Важно!

Время между двумя переполнениями Timer2 составляет (PR2+1) счетных цик-

Значения делителя частоты могут составлять 1, 4 и 16. В таймере Timer2 также используется постделитель с коэффициентом p = 1...16, который доступен только чтение и потому применим исключительно для формирования прерываний по переполнению (прерывание возникает только при каждом p-м переполнении).

Таймер Timer2 является основой для **аппаратных модулей широтно-импульсной модуляции (ШИМ)** (см. подраздел 1.14.3).

1.11.4. Сторожевой таймер

Сторожевой таймер выполняет особую роль и недоступен ни для чтения, ни для записи — его можно только сбросить в нуль по команде CLRWDT. Сброс сторожевого таймера выполняет также команда SLEEP.

Сторожевой таймер включается и отключается в соответствии с настройками конфигурации и оснащен собственным осциллятором, который работает даже во время "спящего" режима. Сторожевой таймер использует свой предварительный делитель совместно с таймером Timer0. Делитель частоты при этом может составлять максимум 128, что соответствует времени до возникновения переполнения около 18 мс (точность невысока). Когда возникает переполнение сторожевого таймера, то производится системный сброс.

Сторожевой таймер имеет двоякое назначение:

- **Обеспечение надежной работы программы**. Сторожевой таймер сбрасывают в определенных местах программы, которые выбирают таким образом, чтобы при переполнении сторожевого таймера можно было сделать вывод, что программа вышла из нормального режима. Если команд сброса будет слишком много, то работа сторожевого таймера становится безрезультатной.
- **Регулярное выведение из "спящего" режима**. Для снижения энергопотребления можно сделать так, чтобы программа активизировалась только в определенные моменты времени, в промежутках между которыми главный осциллятор должен быть отключен.

1.12. "Спящий" режим

Под "спящим" режимом подразумевают такое состояние, в котором главный осциллятор остановлен, а центральный процессор бездействует (а также все функ-

ции, зависящие от главного осциллятора). Таймеры продолжают работать, если они тактируются извне при отсутствии внутренней синхронизации с главным осциллятором.

Единственная причина перевода микроконтроллера в "спящий" режим — уменьшение энергопотребления. Потребляемая мощность в "спящем" режиме обычно лежит в пределах одного процента (или меньше) потребляемой мощности при тактовой частоте 4 МГц.

В "спящем" режиме почти все регистры сохраняют свои значения — особенно регистры ввода-вывода и их TRIS-регистры. Сторожевой таймер, если он включен, продолжает работать, поскольку не зависит от главного осциллятора.

Переход в "спящий" режим происходит по команде SLEEP, а для выхода из него существует три способа:

- импульс на входе /MCLR;
- истечение времени задержки сторожевого таймера;
- прерывание.

Для выхода из "спящего" режима по прерыванию должен быть установлен соответствующий флаг разрешения прерывания, а при прерываниях периферии — также и разряд PEIE в регистре INTCON. Подпрограмма обработки прерывания выполнятся после выхода из "спящего" режима только тогда, когда дополнительно установлен разряд GIE в регистре INTCON.

Вывод из "спящего" режима возможен только от тех источников прерываний, которые активны и в "спящем" режиме. Если требуется точность по времени, тогда имеет смысл использовать прерывание от таймера, но при этом таймер должен тактироваться от внешнего кварца при отсутствии синхронизации с главным осциллятором.

После выхода из "спящего" режима выполнение программы продолжается с команды, следующей за командой SLEEP. Рекомендуется после команды SLEEP размещать две команды NOP. Если выход из "спящего" режима происходит по разрешенному прерыванию, то оно выполняется до возобновления выполнения программы.

Рассмотрим типичную ситуацию, в которой прибор, питающийся от батарейки, должен производить замер параметров окружающей среды (например, температуры). Сторожевой таймер для этого случая применения слишком неточен. Кроме того, переполнение наступает довольно часто (приблизительно через каждые 18 мс), а внутренние таймеры не работают. Решением является осциллятор Timer1 с внешним кварцем на 32 кГц, с потреблением тока менее 10 мкА. Для вывода из "спящего" режима разряд T1SYNC в регистре T1CON должен быть равен 1 (отсутствие синхронизации!).

1.13. Сброс

Сбросом называют состояние, когда центральный процессор готовится к работе или же возобновляет работу. Для перехода в состояние сброса существуют различные причины.

Наиболее важным видом сброса является начальный момент, после подачи напряжения питания. Этот вид сброса так и называют: "сброс по включению питания" (Power-on RESET — POR). В более поздних моделях микроконтроллеров PIC реализованы схемы обнаружения провалов питания, которое производят сброс, если на-

напряжение питания оказывается ниже определенного порога и после этого снова достигает достаточного уровня (Brown-Out RESET — BOR).

Сброс может также быть обусловлен импульсом на выводе /MCLR или завершением счета сторожевым таймером. Необходимо четко различать возобновление работы после выхода из "спящего" состояния.

В старшем подсемействе микроконтроллеров PIC для сброса существует еще больше причин, которые описаны в главе "PIC18".

После старта программы иногда очень важна информация о причине сброса. В базовой серии и среднем подсемействе PIC эту причину можно определить с помощью двух разрядов регистра состояния STATUS: /TO и /PD.

Если /TO = 0, то это означает, что произошло переполнение сторожевого таймера, а /PD = 0 указывает на сброс после выхода из "спящего" режима. В случае сброса по импульсу на входе /MCLR оба разряда остаются неизменными.

После сбросов типа POR и BOR разряды /TO и /PD имеют значение 1. Для того чтобы можно было различить эти причины, в микроконтроллерах PIC с обнаружением провалов напряжения был введен еще один регистр специального назначения (PCON) с разрядами /POR и /BOR.

Представленное ниже описание имеет силу только для базовой серии и среднего подсемейства PIC. В PIC18 существуют дополнительные сбросы и некоторые различий во флагах (см. главу "PIC18").

1.13.1. Сброс по включению питания

Наибольше внимания мы посвятим сбросу по включению питания, который наступает, когда на микроконтроллер подается питающее напряжение. Для того чтобы микроконтроллер мог работать, на входе /MCLR (Master Clear) должен быть высокий уровень сигнала. При достижении достаточного уровня напряжения запускается осциллятор. Разумеется, перед тем как центральный процессор начнет свою работу, напряжение питания должно достичь устоявшегося, надежного уровня. Зачастую также желательно, чтобы осциллятор к этому времени работал стабильно. Кроме того, разумеется, должны быть инициализированы различные регистры. Прежде всего, при сбросе по включению питания должен устанавливаться в исходное состояние счетчик команд.

С одной стороны, следует уделить особое внимание росту напряжения, которое сопровождается ростом уровня на входе /MCLR. С другой стороны, необходимо тщательно организовать программный старт, поскольку за короткий период после сброса должна быть выполнена правильная инициализация окружения периферии.

После сброса по включению питания большинство регистров специального назначения имеют определенные начальные значения, которые можно найти в соответствующих технических описаниях. Так, например, все порты принудительно инициализируются как входы (все разряды регистров TRIS содержат 1).

После старта программы первым делом устанавливают на желаемые значения регистры специального назначения, если после сброса они должны отличаться от установленных по умолчанию. При этом следует обращать внимание на последовательность действий (например, сначала устанавливаются корректные значения порта, и только после этого порт определяется как выход). Кроме того, прерывания следует разрешать только после выполнения всех начальных условий.

Правильная практика — сброс всех обычных файловых регистров с последующей инициализацией тех регистров, которые должны содержать другие исходные значения.

Может возникнуть следующая типичная тупиковая ситуация: программа превосходно работает в эмуляторе, но потом не хочет функционировать в запрограммированном микроконтроллере. Кроме прочих причин, дело может быть в том, что эмулятор вначале сбрасывает все общие регистры, а в программе, возможно, не была выполнена инициализация переменных.

Между моментом, когда периферия получает питание, и моментом, к которому микроконтроллер завершает выполнение своей программы инициализации, проходит определенное время. В течение этого промежутка времени необходимо позаботиться о надлежащих установках (включить/выключить подтягивающие сопротивления!).

Также следует обратить внимание на физической процесс подачи напряжения питания. Микроконтроллер использует определенный, специфицированный диапазон напряжений. За пределами этого диапазона он, возможно, и будет работать, но без гарантий надежности и долговечности. Оптимальное напряжение питания быстро возрастает и затем остается на постоянном уровне. Если до оптимума настолько далеко, что внутренняя схема сброса больше не в состоянии правильно функционировать, тогда лучше использовать внешний генератор сброса (см. главу "Периферийные модули").

Последние модели микроконтроллеров PIC оснащены таймером задержки, который отвечает за то, чтобы после достижения порогового напряжения состояние сброса продолжалось еще некоторое время. Типичное время задержки составляет порядка 72 мс, но может варьироваться. Это иногда позволяет сэкономить на микросхеме сброса. Таймер задержки может включаться и выключаться с помощью настроек конфигурации.

Во взаимодействии с таймером задержки работает таймер запуска осциллятора, который вслед за интервалом задержки удерживает состояние сброса еще на 1024 цикла осциллятора, чтобы была гарантирована стабильность его работы до старта программы.

1.13.2. Сброс по провалу напряжения

Обнаружение провалов напряжения отсутствует в базовой серии микроконтроллеров PIC, а также реализовано не во всех представителях среднего подсемейства PIC. Под провалом напряжения понимают состояние, при котором напряжение питания оказывается ниже минимального предельного значения, гарантирующего надежное функционирование. Обнаружение провалов может включаться и выключаться с помощью настроек конфигурации.

Сброс по провалу напряжения возникает, если напряжение питания на определенное время (T_{BOR}) опускается ниже определенного минимального порога (V_{BOR}). Когда напряжение опять поднимается выше этого порога, следует точно такой же сброс, как при сбросе по включению питания. Разряд /BOR = 0.

1.13.3. Сброс по сигналу на входе /MCLR и сброс от сторожевого таймера

Сброс может быть также обусловлен импульсом отрицательной полярности на входе /MCLR. Еще одна причина сброса (намеренная или ненамеренная) — завершение счета сторожевым таймером. В обоих случаях большинство регистров специального назначения остаются неизменными, а некоторые возвращаются в исходное состояние так же, как при сбросе по включению питания (например, регистр TRIS). Подробности ищите в технических описаниях!

Кроме того, имеет значение, в каком режиме происходит один из этих сбросов: в нормальном или в "спящем". В последнем случае счетчик команд, само собой разумеется, в исходное значение не сбрасывается, и программа продолжает работу с команды, следующей за командой SLEEP. К тому же существуют различия в состоянии регистров после этих двух сбросов. При выходе из "спящего" режима неизменными остаются многие регистры, которые при обычном сбросе возвращаются к предустановленным значениям.

Все события сторожевого таймера — как в нормальном, так и в "спящем" режиме — устанавливают разряд /TO в значение 0.

Сброс по сигналу на входе /MCLR-сброс оставляет разряды /TO и /PD неизменными!

1.14. Аппаратные модули

Аппаратные модули работают без участия центрального процессора, роль которого заключается только в их конфигурировании и опросе состояния.

Аппаратные модули можно разбить на три категории:

- аналоговые (АЦП, компараторы);
- коммуникационные (USART, SSP, CAN, USB);
- модули, связанные с таймерами (фиксации, сравнения, ШИМ).

Внутренний осциллятор тоже является аппаратным модулем, но он, традиционно, под этой рубрикой не рассматривается. Строго говоря, таймеры и порты ввода-вывода также относятся к аппаратным модулям, но, учитывая их особую роль, они уже были рассмотрены ранее.

Практическая работа с аппаратными модулями проста и для всех модулей схожа. Каждому модулю отведены регистры специального назначения — результата, конфигурации и состояния. Окончания "con" или "stat" в конце одного из таких регистров не следует интерпретировать слишком строго.

В первом поколении базовой серии микроконтроллеров PIC аппаратные модули не использовались, поскольку в небольшом объеме RAM-памяти было недостаточно места для размещения требуемых регистров специального назначения. Однако, коль в микроконтроллере присутствует только один порт (GPIO), тогда высвобождаются две ячейки под дополнительные регистры. Одна из таких ячеек используется для регистра OSCCAL. В новых микроконтроллерах базовой серии мы находим компаратор, и не будет ничего удивительного, если однажды увидят свет "маленькие" PIC с аппаратными модулями.

Кроме всего прочего, почти для каждого аппаратного модуля задействован как минимум один флаг прерывания, информирующий о том, что произошло некоторое событие, связанное с модулем. Соответствующее прерывание возникает только в том случае, если оно разрешено.

Почти каждый аппаратный модуль для своего функционирования нуждается в одной или нескольких линиях ввода-вывода. Для того чтобы выводы микроконтроллера, выделенные для обмена данными с аппаратным модулем, использовать как обычные выводы, модуль должен быть выключен с помощью регистра конфигурации.

Если какой-либо вывод порта работает некорректно, это может быть связано с тем, что он занят аппаратным модулем. Обычно аппаратные модули после сброса выключены, и потому их необходимо включать принудительно из программы (исключение — аналоговые входы).

Аппаратные модули со всеми соответствующими регистрами специального назначения хорошо описаны в технических спецификациях. Искать в них перечень выводов, имеющих отношение к модулям, зачастую бессмысленно. За такой информацией мы рекомендуем, в случае необходимости, обращаться к описанию расположения выводов, которое можно найти вначале любой спецификации микроконтроллера.

Аппаратные модули, связанные с последовательным обменом данным, описаны в отдельной главе "Последовательный обмен данными".

1.14.1. Аналого-цифровой преобразователь

Аналого-цифровой преобразователь (АЦП) обслуживается двумя регистрами: ADCON0 и ADCON1. Регистр результата называется ADRES, если речь идет о восьмиразрядном преобразователе. Если же разрешение составляет больше восьми разрядов, то используют два регистра результата: ADRESH:ADRESL — выравнивание в которых может быть выбрано как по правому, так и по левому краю.

Первые АЦП имели разрешение 8 разрядов, а в более новых разрешение составляет 10 разрядов. Также есть представители с 12-тиразрядными АЦП, например, 16F77xx. Чем выше разрешение, тем выше точность измерений, однако высокое разрешение АЦП не является гарантией точных результатов измерений. Если входная схема неточная, то пользы не будет даже от 12-тиразрядного АЦП. Если на входе присутствует шум или другие стохастические помехи, то улучшение результатов зачастую достигается четырехкратным измерением с помощью 10-тиразрядного АЦП с последующим усреднением результата.

Регистру ADCON1 соответствует основная конфигурация, а регистр ADCON0 отвечает, преимущественно, за обслуживание. К основной конфигурации относится определение выводов микроконтроллера, выбранных в качестве аналоговых входов. Кроме того, с помощью регистра ADCON1 можно выбрать верхнее и нижнее опорное напряжение. Для верхней границы выбирают V_{DD} или специальное опорное напряжение, которое подается на определенный вход АЦП. Для нижней границы можно выбрать V_{SS} или также специальное опорное напряжение.

Еще один элемент конфигурации — генератор импульсов для тактирования преобразователя. По сути, модуль АЦП оснащен собственным RC-осциллятором, однако, при желании, от него можно отказаться, заменив его главным осциллятором с настраиваемым предварительным делителем частоты. Собственный осциллятор АЦП имеет то преимущество, что во время преобразования можно выключать главный осциллятор (команда SLEEP), чтобы он не вносил помех в процесс преобразования. При этом выход микроконтроллера из спящего режима происходит по прерыванию от АЦП.

По физическим причинам, тактирование АЦП должно происходить в пределах определенного диапазона скорости. Если оно слишком медленное, то напряжение на конденсаторе хранения между двумя шагами может понизиться настолько, что окончания преобразования можно не дождаться. Если же тактирование слишком быстрое, то пренебрегается время установления, что ведет к непредсказуемому поведению. Таким образом, следует придерживаться рекомендаций, представленных в технических описаниях.

Если в некотором случае применения используется несколько каналов АЦП, то перед измерением необходимо выполнить выбор канала, для чего предназначены три разряда в регистре ADCON0.

После переключения канала необходимо выждать определенное время, пока не установится полное новое напряжение на внутреннем конденсаторе хранения. Это время установления зависит от импеданса измерительного входа. Чем он выше, тем больше следует ожидать. Импеданс не должен быть более 10 кОм. Если время неизвестно, то время установления следует выбирать с запасом. Для него в технических описаниях есть формула — несложная, но если ею пренебречь, то это чревато последствиями.

Если модуль АЦП изначально верно один раз сконфигурировать, то в дальнейшем его обслуживание не вызывает затруднений.

Преобразование начинается по истечении **времени установления** записью 1 в разряд GO регистра ADCON0. В конце преобразования модуль АЦП возвращает этот разряд в исходное состояние. После этого можно считать цифровой результат из регистра ADRES или из регистровой пары ADRESH:ADRESL. Само событие об окончании преобразования можно узнать с помощью соответствующего флага прерывания. Регистры ADCON0 и ADRES (ADRESH) в среднем подсемействе микроконтроллеров PIC находятся в банке 0, а ADCON1 и ADRESL — в банке 1.

Пример простейшего цикла преобразования:

```
        BSF     ADCON0,GO
WAIAD   BTFSC   ADCON0,GO
        GOTO    WAIAD
```

Теоретически, в этом цикле можно находиться бесконечно, поэтому рационально выполнять в нем обратный отсчет с помощью интервального счетчика. Преобразование, как правило, не должно длиться более 30 мкс (точное время зависит от различных обстоятельств).

Времени установления обязательно следует уделять внимание также в том случае, когда наблюдается скачок измеряемого напряжения (например, после размыкания или замыкания ключа). При этом время установления может быть заметно больше, чем после переключения канала.

Пример — проверка аккумулятора. Сначала отключается нагрузка, после чего выжидается время восстановления батареи (в некоторых случаях — до 100 мс!), пока на конденсаторе хранения не установится напряжение холостого хода, и только тогда можно начинаться измерение. После этого батарея в течение нескольких секунд нагружается опорным током, после чего опять выполняется измерение, и разность оценивается как мера состояния аккумулятора.

АЦП часто применяют для измерения сопротивления датчиков (например, датчиков температуры или света), что в простейшем случае реализуют с помощью делителя напряжения. При этом необходимо обращать внимание на импеданс на входе АЦП — в крайних случаях может даже потребоваться помощь какого-нибудь преобразователя полных сопротивлений, например, операционного усилителя.

В заключение упомянем о существовании интересной возможности измерения сопротивлений без АЦП путем измерения времени заряда и разрядки конденсатора.

1.14.2. Компаратор

Компаратор сравнивает два напряжения и выдает результат в виде логического значения на выходе микроконтроллера. Модули компараторов PIC, как правило, состоят из двух компараторов и очень гибкие в конфигурировании. Регистр CMCON, кроме разрядов конфигурации, содержит также значения выходных разрядов (C1OUT, C2OUT).

С помощью разрядов C1INV и C2INV выбирают полярность на выходе, а с помощью разрядов CIS, CM0, CM1 и CM2 — конфигурацию компараторов. На рис. 1.4, взятом из технического описания микроконтроллера PIC16F87XA, все проиллюстрировано лучше всяких слов.

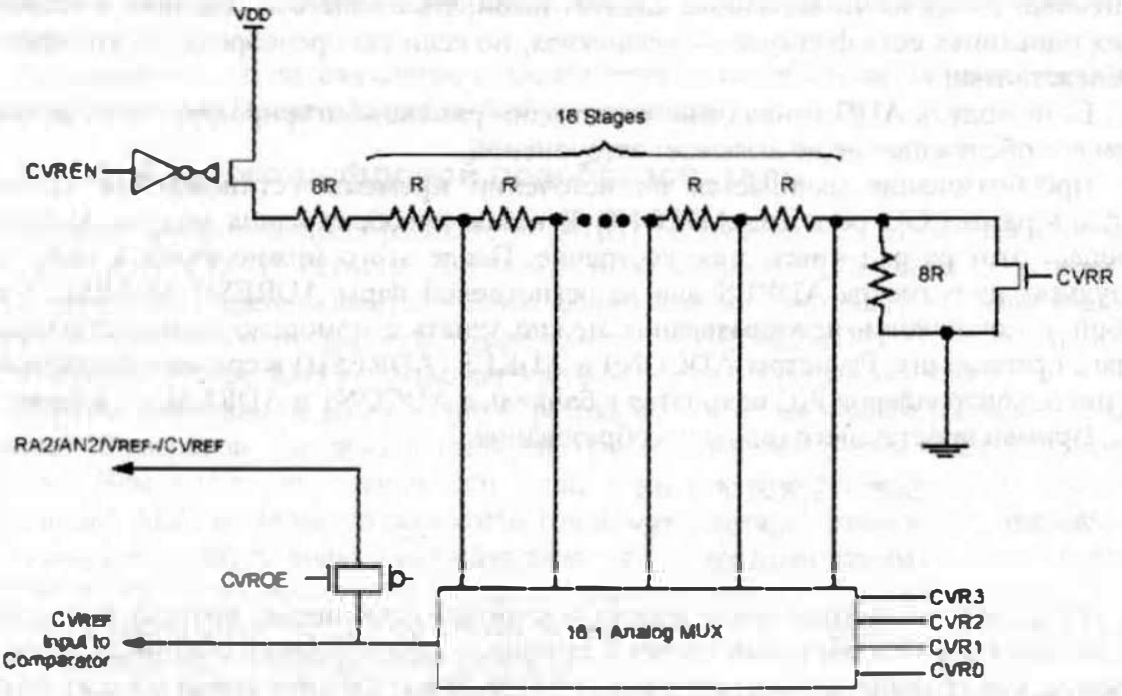


Рис. 1.4. Модуль опорного напряжения

При желании, биты результата можно выдать на выводы портов. Оба компаратора могут быть соединены между собой.

Напряжение для сравнения можно подать на вывод порта или воспользоваться внутренним модулем опорного напряжения. Генератор опорного напряжения состоит из цепи резисторов. Модуль опорного напряжения может гибко настраиваться с помощью конфигурационного регистра (CVRCON).

Безусловно, подробности следует выяснять по соответствующим техническим описаниям, поскольку между разными типами микроконтроллеров PIC существует множество различий.

Если вывод определен как вход компаратора, то он уже не в состоянии функционировать как цифровой вывод и всегда считывается как "0".

Подобно тому, как это происходит в случае с АЦП, соответствующие выводы портов после сброса всегда устанавливаются как аналоговые входы. Для того чтобы их можно было использовать как линии ввода-вывода, компаратор необходимо отключить.

1.14.3. Модуль CCP/ECCP

Модуль CCP поддерживает три различных режима: захвата (Capture) и сравнения (Compare) под управлением таймера Timer1, и режим ШИМ (PWM) под управлением Timer2. Некоторые микроконтроллеры содержат два (или более, до 5) модулей CCP, которые можно конфигурировать в различные режимы независимо друг от друга. Обозначение "ECCP" касается только режима ШИМ (Enhanced PWM).

В трех режимах используются одни и те же регистровые пары: CCP1RH: CCP1RL — для первого модуля и CCP2RH:CCP2RL — для второго. В дальнейшем

будем применять обозначение "CCPxRH:CCPxRL", поскольку упомянутые пары своим функциям почти идентичны. Эти двойные регистры будем называть "регистрами CCP". То же самое относится и к регистрам конфигурации CCP1CON и CCP2CON, общим для всех трех режимов (общее обозначение "CCPxCON").

Регистр CCPxCON содержит четыре разряда для выбора режима CCP. Если все четыре разряда содержат 0, то модуль отключен. Кроме того, в регистрах CCPxCON есть еще два разряда, которые служат только для режима ШИМ (см. текст ниже).

Каждому модулю CCP отведен один вывод микроконтроллера, который также является общим для всех трех режимов. Эти выводы (CCP1 или CCP2) обычно расположены в ряду порта C (см. расположение выводов).

Режим захвата

Режим захвата служит для фиксации момента времени (значение таймера Timer1) появления фронта на выводе CCP. В этом режиме вывод CCP, как правило, конфигурируется как вход. Если вывод CCP сконфигурировать как выход, то можно с помощью фронта на этом выводе вызывать прерывание — трюк, который при известных обстоятельствах может быть полезным.

Для функции захвата нет альтернативного решения, которое было бы настолько удобным. Постоянный опрос входов, как правило, неприемлем. Применение внешнего прерывания по сигналу на выводе 0 порта B имеет серьезный недостаток: необходимо разрешить прерывание, что в ряде случаев не позволяет использовать для другой цели.

В случае режима захвата, данные о времени появления фронта находятся в соответствующем регистре CCP до тех пор, пока не наступит новое событие появления фронта, если только за этот промежуток времени был сброшен флаг прерывания (CCPxIF). Для этого даже не обязательно разрешать прерывание — зачастую достаточно опрашивать флаг прерывания через достаточно короткие отрезки времени. В среднем подсемействе микроконтроллеров PIC флаг прерывания для первого модуля CCP находится в регистре PIR1 (CCP1IF), а для второго — в регистре PIR2 (CCP2IF).

Режим захвата очень удобен для сбора информации о времени оборота двигателя. При частоте вращения 6000 об/мин время оборота составляет 10 мс. Время, которое есть в распоряжении для чтения регистра CCPxR, как правило, заметно больше длительности основного цикла программы.

Существует четыре варианта режима захвата:

- захват по каждому отрицательному фронту;
- захват по каждому положительному фронту;
- захват по каждому четвертому положительному фронту;
- захват по каждому шестнадцатому положительному фронту.

Режим сравнения

В режиме сравнения заранее задается определенное значение таймера Timer1, при достижении которого устанавливается флаг прерывания. Это 16-тиразрядное значение записывается в соответствующий регистр CCP. Дополнительно, на выбор, может выполняться специальное аппаратное действие, которое определяют с помощью регистра CCPCON. В любом случае, при достижении состояния таймера Timer1 значения регистра CCPR устанавливается флаг прерывания,.

Вывод CCP, если он используется, должен быть сконфигурирован как выход!

К четырем вариантам дополнительных действий относятся:

- установка на выходе высокого уровня сигнала в случае совпадения;
- установка на выходе низкого уровня сигнала в случае совпадения;
- никакого дополнительного действия
- сброс Timer1, CCP2 дополнительно запускает аналого-цифровое преобразование.

Если, кроме дополнительных действий, основанных на аппаратной части, требуется выполнить еще и некоторые программные функции, то необходимо воспользоваться соответствующим прерыванием, поскольку смысл режима сравнения именно в том, чтобы действие происходило в точно определенный момент времени.

Режим сравнения практически незаменим, например, в тех случаях, когда через переменные интервалы времени необходимо выдавать на порт управления значения синуса для управления двигателем. Также, когда требуется реализовать линейно изменяющийся сигнал, то без режима сравнения это сделать очень сложно, поскольку, как правило, одновременно должны решаться еще и другие задачи.

Режим ШИМ

Режим ШИМ служит для формирования периодичных прямоугольных сигналов. Период и длительность импульса можно сконфигурировать в определенных пределах. Режим ШИМ связан с таймером Timer2.

Период ШИМ-сигнала определяется предварительным делителем таймера и регистром периода PR2. Поскольку коэффициент деления частоты не превышает 16, максимально возможный период — 4096 командных циклов. При частоте системной синхронизации 4 МГц это соответствует длительности периода около 4 мс.

Внутри периода на выводе CCP некоторое время установлен высокий уровень сигнала. Это время называется "рабочей фазой". В отличие от периода ШИМ, который можно выбирать кратным периоду таймера, рабочая фаза может быть запрограммирована с разрешением в четыре раза выше. Рабочая фаза задается десятью разрядами. Старшие 8 разрядов хранятся в регистре CCPxRL, а два младших — в регистре CCPxCON (разряды 4 и 5). Значение в регистре CCPxRL представляет собой период, кратный длительности цикла Timer2. Младшие разряды соответствуют четверти этой длительности цикла.

Конфигурация режима ШИМ происходит в следующей последовательности:

1. Установить коэффициента деления частоты (T2CON) как можно меньшим.
2. Вычислить длительность ШИМ периода в циклах таймера (PR2+1).
3. Вычислить рабочую фазу (CCPxRL, CCPxCON, 4 и 5).
4. Задать режим ШИМ в регистре CCPxCON.
5. Обнулить и включить таймер Timer2.
6. Определить ШИМ-вывод как выход.

Пример

Модуль CCP1 должен применяться в режиме ШИМ для формирования прямоугольных импульсов с длительностью периода 1,2 мс и скважностью импульсов 12,5%. Частота системной синхронизации — 4 МГц.

Коэффициент деления частоты выбираем равным 16, так как в случае значения 4 можно реализовать периоды максимум в 1,024 мс.

Длительность периода составляет 1200/16 = 75 циклов таймера. Отсюда следует, что должно устанавливаться PR2+1 = 75 (PR2=74).

Рабочая фаза составляет 75·0,125 = 9,375. Целую часть 9 записываем в регистр CPR1L и округляем остаток до четверти (в данном случае — до 9,5). Таким образом, два младших разряда равны 10.

Если бы мы пренебрегли остатком, то получили бы относительную погрешность почти 4%. С дополнительными разрядами относительная погрешность составляет всего лишь около 1,3%.

Во многих приложениях (например, при управлении двигателями) длительность периода зачастую удерживается постоянной, скважность импульсов (рабочая фаза) вычисляется согласно алгоритму управления.

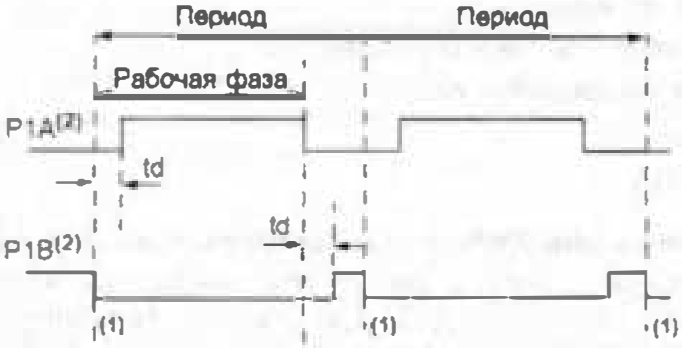
Пользователь знает, что во многих случаях определяющим фактором является точность не столько длительности периода, сколько скважности импульсов. Два дополнительных разряда особенно существенны при малых значениях регистра PxRL.

Усовершенствованная ШИМ

Усовершенствованная ШИМ (Enhanced PWM) служит для удобства управления полу- и полномостовыми схемами. Модуль CCP при этом может обслуживать не только один, а, на выбор, — два или четыре выхода одновременно.

Полумостовая схема состоит из двух полевых транзисторов: верхнего и нижнего ключа. Если оба ключа замкнуть, то это приведет к короткому замыканию, которое разрушит мост, если же они одновременно переключаются (например, верхний ключ размыкается, а нижний — замыкается), тогда короткое замыкание наступает по физическим причинам, так как процесс переключения занимает некоторое время (скорость затвора).

С помощью усовершенствованной ШИМ пользователь имеет возможность реализовывать на аппаратном уровне программируемое время задержки между двумя процессами переключения — так называемой "задержки мертвой зоны"(рис. 1.5).



(1) — момент времени, когда значение TMR2 равно значению регистра PR2;
(2) — выходы определяются как активные при высоком уровне сигнала

Рис. 1.5. Задержка мертвой зоны

Полезной функцией является **автоотключение** — автоматическое аварийное отключение при наступлении определенного события. Условия, определяющие аварийное отключение, можно выбрать с помощью регистра ECCPAS: INT0, компаратор 1, компаратор 2 и некоторые объединения по логическому "ИЛИ" этих трех событий. С помощью регистра ECCPAS можно также определять параметры состояния отключения (0, 1 или третье состояние).

Регистр ECCPAS содержит разряд EPPCASE, который во время состояния аварийного отключения имеет значение 1.

Для рестарта после аварийного выключения существуют две возможности:

- авторестарт сразу же по завершению условия отключения — для этого должен быть установлен в 1 разряд PRSEN в регистре PWMxCON;
- запуск через сброс разряда EPPCASE — это невозможно, если условие отключения еще активно. Если, несмотря на это, разряд EPPCASE все-таки необходимо сбросить, то должен быть установлен в 0 разряд PRSEN.

На практике обычно с рестартом очень осторожны, и перед тем как опять выполнить включение или выдерживают некоторое время после исчезновения условия выключения, или ожидают следующего события. Этим также предотвращаются постоянные включения/выключения.

Если, например, к аварийному выключению привело превышение температуры, то требуется или падение температуры за определенное время ниже некоторого предела, или достижение нижнего температурного порога (гистерезис).

1.15. Конфигурация

Под конфигурацией понимают выбор функций микроконтроллера PIC, которые устанавливаются при программировании и во время выполнении программы измениться уже не могут.

В зависимости от типа микроконтроллера, существует различнео количество свойств, определяемых разрядами конфигурации:

- выбор осциллятора;
- источники сброса:
  - таймер задержки;
  - таймер запуска главного осциллятора;
  - обнаружение провала напряжения;
- сторожевой таймер;
- монитор отказоустойчивости генератора;
- запуск с удвоенной скоростью;
- защита кода;
- режим отладки.

Рационально помещать конфигурационные установки в начале программы. Для этого служит директива `__config` (см. систему помощи ассемблера MPASM).

В папке `\MPLAB IDE\MCHIP_TOOLS` для каждого типа микроконтроллеров PIC представлен так называемый заголовочный файл (`.inc`). В его конце, наряду со всеми обозначениями, находится список всех разрядов конфигурации, возможных для соответствующей модели. Там же приведен образец полной конфигурации.

Кроме того, можно временно модифицировать конфигурацию в окне конфигурации перед программированием. Это, например, удобно, в тех случаях, когда требуется временно изменить защиту кода.

ГЛАВА

2 Последовательный обмен данными

Важным отличительным признаком микроконтроллера является его способность к быстрому и удобному обмену данными. Взаимодействие с электронными элементами, другими частями оборудования или ПК происходит почти исключительно с помощью различных последовательных интерфейсов.

В первом поколении микроконтроллеров PIC для обмена данными приходилось выдавать биты по одному на порты ввода-вывода, и таким же образом считывать их. В результате, или центральный процессор был занят этим большую часть времени, или приходилось изловчаться и время ожидания следующего бита использовать для мелких задач наблюдения. Сегодня подобную "акробатику" применяют лишь в исключительных случаях, когда очень большое число элементов оправдывает любой вид экономии аппаратного обеспечения.

Новое поколение микроконтроллеров PIC предлагает различные аппаратные модули для последовательного обмена данными. Центральный процессор больше не имеет никакого отношения к выдаче или чтению отдельных битов, а только уведомляется с помощью флагов о событиях последовательного интерфейса (например, получение байта или освобождения буфера передачи). Каждому событию последовательного интерфейса соответствует свое прерывание.

Несколько лет назад понятия "V.24" или "RS232" были для нас синонимами последовательной передачи данных, однако на сегодняшний день в микроконтроллерах PIC используются и другие последовательные интерфейсы. Они различаются физическими свойствами передачи, логическими определениями и организационными положениями обмена данными. Выбор того или иного типа интерфейса обычно обусловлен характером применения.

2.1. Свойства последовательных интерфейсов

Если к интерфейсу подключено только два участника, как это исконно было при RS232, то требуются совсем немного соглашений о передаче данных. Однако современные системы связи ориентированы на большое число участников. В этом случае возникает понятие шины.

Когда говорят об интерфейсе или шине, то под этим подразумевают совокупность всех соглашений и положений для протоколов, линий передач, физических уровней и соединителей. К примеру, интерфейсу RS232 соответствует очень простой протокол передачи. Интерфейсный модуль, работающий по этому протоколу, называется UART (также SCI). Для того чтобы из интерфейса UART получился интерфейс RS232, должны соблюдаться еще и другие положения, затрагивающие физические уровни, а также стандарты кабеля и соединителей.

2.1.1. Управление битами

Наиболее низкий уровень, на котором должен быть определен интерфейс, — уровень битов. Сначала необходимо установить, каким образом передается бит (то есть, "1" или "0"), — логически и физически. Распространенное решение: 1 — высокий уровень сигнала; 0 — низкий уровень сигнала. В качестве альтернативы, существует также и такое соглашение: 1 — отсутствие изменения уровня; 0 — изменение уровня. При беспроводном интерфейсе чаще всего принимают соглашение следующего вида: 1 — длинная пауза; 0 — короткая пауза в несущем сигнале.

Необходима также договоренность касательно синхронизации передаваемых битов, чтобы приемник знал, где заканчивается и начинается следующий бит.

В интерфейсах, обозначенных как **асинхронные**, следование сигнала жестко согласуется по времени. Для этого все участники должны обладать очень точным тактированием (внутренним). Как вариант, передатчик может перед каждым блоком данных передавать поле синхронизации, с помощью которого остальные участники могут подстраивать свой отсчет времени.

Еще одна концепция заключается в сопровождении обмена данными синхроимпульсами, каждый из которых сигнализирует о том, что бит в линии действенен (**синхронный интерфейс**).

При управлении битами также используют трюк под названием "вставка битов" (bit stuffing). Если было отправлено определенное число битов без смены уровня, то искусственно вставляется бит, который имитирует смену уровня. На стороне приемника он опять удаляется из потока битов.

2.1.2. Битовые поля

Биты имеют смысл только тогда, когда они объединяются в большие структуры, такие как байты, слова и т.д.

Самая простая битовая структура — это байт. Для того чтобы обозначить начало и конец байта, многие протоколы используют один стартовый бит и, в зависимости от соглашения, — один или два стоповых бита. На выбор, может также отправляться проверочный бит (бит четности).

Однако сложные шинные системы, кроме полезных данных, нуждаются в передаче большого количества дополнительных бит, содержащих коммуникационную техническую информацию. Эти биты образуют **кадр**. К ним относятся **битовые поля**, которые служат для подготовки приемника к предстоящему обмену данными, или уже упомянутые выше биты синхронизации, предназначенные для согласования тактирования.

Если к линии связи подключено несколько участников, то также требуется адрес или идентификационное поле, чтобы (главным образом) приемник смог определить, ему ли предназначена посылка.

От аппаратных модулей микроконтроллеров PIC ожидается, что они не только возьмут на себя управление отдельными битами, но и избавят нас от работы с кадрами. Это означает, что, выступая в роли приемника, мы хотим получать только полезные данные, а в роли передатчика — не заботиться о передаче битов кадра.

2.1.3. Ведущий и ведомый

В шинной системе с несколькими участниками также требуется соглашение о том, какой участник в определенный момент времени уполномочен на отправку.

Даже когда присутствует только два участника, может потребоваться договоренность о том, кто может "говорить", а кто обязан "слушать".

Распространенным решением этой проблемы является назначению одному из участников функций ведущего (Master). Ведущий берет на себя инициативу касательно всех передач данных по шине и распределяет "право слова". Кроме того, он обычно задает тактирование. Другие участники называются "ведомыми" (Slave).

Существуют системы с несколькими ведущими и даже такие, в которых ведущий вообще отсутствует. Известный пример, когда, благодаря хитроумной организации, обходятся без ведущего, — это шина CAN.

Системой с "авторитарным" ведущим является шина USB, которая, прежде, чем разрешить своим ведомым обмен данными, требует от них обширной "бюрократии" сообщений. В качестве ведущего при этом, как правило, выступаем ПК под управлением Windows. В USB-применениях микроконтроллер PIC всегда выполняет роль ведомого.

2.2. Модуль SSP (SPI и I2C)

Применение мнкроконтроллеров при подключении элементов с последовательной передачей данных, таких как память, цифровые датчнки или часовые микросхемы, имеет большое значение. В такой области PIC — ведущий передачи, а внешние элементы — ведомые. Ведущий решает, когда начинается передача, и выбирает, с каким участником он хотел бы вести обмен. Взаимодействие ведомых между собой невозможно и чаще всего не имеет смысла.

Подходящие шины для такого вида применения — SPI (Serial Peripheral Interface) и I^2C (Inter Integrated Circuit), однако они подходят для обмена данными исключительно между несколькими микроконтроллерами. Обе шины задуманы для связи в пределах одного устройства.

Как SPI, так и I^2C работает на основе синхроимпульсов, передаваемых ведущим устройством (то есть, это — синхронные интерфейсы). Таким образом, в этих двух шинах отсутствует жестко оговоренная скорость передачи данных. Хотя шина I^2C первоначально была задумана для частоты следования битов 400 кГц, сегодня она, в основном, гибкая в настройке и поддерживает скорость до 1 МГц. Допустимые пределы скорости передачи определяет ведущий. Он выдает синхроимпульсы, и ведомые принимают биты с заданной скоростью. Ведущий, разумеется, также отвечает и за адресацию отдельных участников.

Аппаратный модуль микроконтроллера PIC, который можно сконфигурировать как интерфейс I^2C или SPI, называется **SSP**. Интерфейсом I^2C на основе модуля SSP можно управлять только как подчиненным, а для функции ведущего в более новых моделях присутствует модуль **MSSP**. На практике наиболее важна работа в режиме ведущего, поскольку для реализации последовательного обмена данными с внешними элементами необходим именно такой режим. Учитывая тот факт, что модуль MSSP реализован далеко не во всех микроконтроллерах PIC, интерфейс I^2C зачастую обслуживают с помощью программного обеспечения, поскольку при выборе микроконтроллера необходимо принимать во внимание и другие важные аспекты.

Вопрос о том, какой интерфейс применять для подключения периферийных элементов: I^2C или SPI — решается, исходя из выбора используемых компонентов. К примеру, в случае подключения нескольких элементов, для интерфейса SPI необходимы линии выбора кристалла, а в интерфейсе I^2C адрес передается по шине. Компоненты имеют адресные входы, на которые подается их индивидуальный адрес, — по отдельным линиям, или через коммутаторы. По этой причине интерфейс

SPI более быстродействующий и простой в управлении, однако для него требуются линии выбора.

2.2.1. Принцип работы SPI

Шина SPI работает по принципу ковшового элеватора. Процесс начинается по записи ведущим байта в регистр SSPBUF. Байт копируется в сдвиговой регистр ведущего и сразу же передается в сдвиговой регистр выбранного ведомого. При этом ведущему одновременно передается ранее загруженное содержимое сдвигового регистра ведомого. Таким образом, нельзя никогда выбрать несколько подчиненных одновременно.

Если у ведущего есть несколько ведомых, то он должен организовать выбор кристалла с помощью любых выводы портов — модуль SSP за это не отвечает. Однако на стороне ведомых есть вывод под названием /SS (Slave Select), на котором должен быть установлен низкий уровень, чтобы модуль SSP воспринимал обращения.

Если обращение направлено только к одному ведомому, то он может оставаться выбранным постоянно. Модуль SPI поддерживает также режим ведомого без использования вывода /SS. В таком режиме ведомый всегда выбран. Тем не менее, учитывая потребление мощности, в этом случае все-таки рациональнее пожертвовать одной линией для сигнала /SS.

Сдвиговые регистры не адресуются и к началу передачи загружаются содержимым регистра SSPBUF. В процессе передачи сдвиговые регистры ведущего и ведомого обмениваются данными, а по окончании передачи их содержимое опять копируется в соответствующие регистры SSPBUF.

При передаче по SPI первым передается старший значащий разряд. Каждый участник уведомляется о завершении передачи с помощью флага прерывания SSPIF. Дополнительно можно также опросить разряд BF (Buffer Full) в регистре состояния SSPSTAT.

В обмене данными по интерфейсу SPI принимают участие следующие выводы:

- SDO — выход;
- SDI — вход;
- SCK — в случае ведущего — выход, в случае ведомого — вход для тактовых импульсов;
- /SS — выбор ведомого (только у ведомого, вход).

Таким образом, по SPI всегда передаются только простые байты. Насколько эти байты носят коммуникационно-технический характер — решать участникам.

2.2.2. Пример SPI

Зачастую требуется, чтобы при обмене данными по интерфейсу SPI инициативу проявлял ведомый. В этом случае для него создают специальную линию уведомления, которой он соединен с ведомым по любому выводу порта. Типичная стратегия для обмена данными между несколькими микроконтроллерами следующая.

1. Ведомый записывает в свой регистр SSPBUF код, содержащий запрос на выполнение желаемого действия и активирует линию уведомления.
2. Ведущий регулярно опрашивает линии уведомления. Если он обнаруживает сообщение, то выбирает соответствующего участника и записывает в свой регистр SSPBUF "ack" или "nak" — в зависимости от того, готов ли он выполнить за-

прос на обмен данными или нет. Во время последующей передачи данных ведущий одновременно принимает код. Если он готов к обмену, то оставляет выбор действительным и ожидает доставки от соответствующего ведомого посылки — непрерывно, байт за байтом.

Если ведомый получает "ack", то сразу же после этого он должен отправить последовательность байтов. Как только он обнаруживает установленный флаг прерывания SSPIF, сигнализирующий о передаче байта, ему следует как можно скорее записать следующий байт в свой регистр SSPBUF. Со стороны программного обеспечения ведущего должно быть предусмотрено предоставление ведомому столько времени между передачей двух байтов, сколько тому потребуется.

Извещение об окончании посылки реализуется с помощью отправки знака конца или числа, представляющего собой количество переданных байтов, или же освобождением линии уведомления.

Если ведомый получает "nak", то он должен повторить свою попытку позже.

После окончания посылки всегда желательно, чтобы ведущий отвечал подтверждением даже в том случае, когда ответ не требуется. При приеме байтов ведомый должен сразу же прочитать регистр SSPBUF как только флаг прерывания SSPIF укажет на завершение передачи. На это у него есть время до завершения следующей передачи. Если до того момента данные не будут считаны, это приведет к ошибке затирания.

При такой стратегии зачастую возникает следующая ситуация: хотя заявка на передачу данных исходит от ведомого, только ведущий определяет, кого из ведомых допустить к обмену данными и, что важнее всего, именно он определяет точный момент начала передачи.

Еще один вариант: для обслуживания какого-нибудь устройства подают заявки одновременно несколько ведомых. В этом случае управление передачей данных может стать и вовсе замысловатым. Все ведомые должны постоянно иметь в виду, что ведущий передает посылки, не дожидаясь приглашения, однако это может перекрываться по времени с активацией линии уведомления. В этом случае необходимо учитывать все возможные ситуации.

2.2.3. Инициализация SPI

При инициализации SPI должны быть приняты общепринятые меры.

Все требования в отношении условий передачи устанавливаются с помощью регистров SSPCON и SSPSTAT. Подробное описание этих двух регистров можно найти в спецификации конкретного микроконтроллера. Будьте внимательны! Не путайте эти регистры, поскольку они оба содержат разряды как конфигурации, так и состояния.

К конфигурированию относится, прежде всего, перевод модуля SSP в режим SPI и установка для него режима работы в качестве ведущего или ведомого. Кроме того, необходимо правильно сконфигурировать соответствующие выводы портов. В некоторых случаях дополнительно требуется установить относящиеся к делу разряды прерываний.

2.2.4. Проблемы с SPI?

Возможно, кто-то думает, что из-за простоты интерфейса SPI с ним не может возникнуть никаких проблем, однако это не так. Если обмен данными по SPI происходит не так, как ожидалось, тогда мы советуем в первую очередь (как для любого аппаратного модуля) проверить всю конфигурацию (прежде всего выводы портов).

Если ошибок обнаружено не будет, тогда необходимо проверить линии связи. Очень важен правильный выбор емкостей самих линий. Если, например, в линии синхроимпульсов возникают отражения, то нарушается процесс подсчета синхроимпульсов в передатчике. На помощью приходят RC-цепочки. Начинайте со значений сопротивления в 56 Ом и емкости в 100 пФ (рис. 2.1).

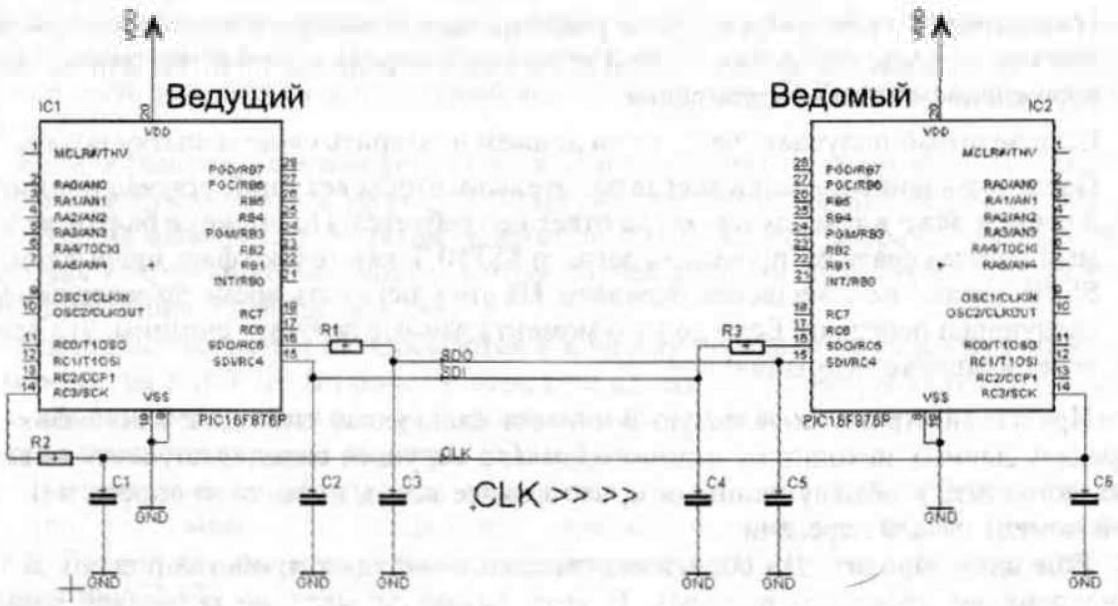


Рис. 2.1. Решение проблем с SPI

2.2.5. Принцип работы шины I²C на базе модуля SSP/MSSP

Кроме полезных данных, шина I²C должна передавать команды, адреса и биты подтверждения. По этой причине она намного сложнее для программирования, чем "оживленная" шина SPI. Затрудняющим фактором является наличие разных вариантов с различным размером поля адреса.

Раньше существовали только модули SSP без режима ведущего, который, тем не менее, является самым распространенным. При обращении к микросхемам периферии микроконтроллер PIC является ведущим устройством. Для обмена данными между двумя или более микроконтроллерами PIC нам едва ли бы пришла в голову мысль применить шину I²C.

Поскольку не было никакой модульной поддержки для режима ведущего, весь протокол должен был реализовываться программно. В результате единственным выходом из положения становилась специальная подпрограмма, которую можно было бы хранить в библиотеке и применять в случае необходимости. Однако все новое всегда чем-то да отличается от старого — и хорошо, если речь идет только о размере поля адреса, поскольку емкость памяти сейчас значительно больше.

Сейчас существует два режима ведущего, первый из которых называется "режимом ведущего с микропрограммным управлением". Звучит знакомо, не так ли?

овым в этом режиме является исключительно конфигурирование соответствующих выводов в виде выходов с открытым коллектором. До этого мы реализовывали эти выходы таким образом, что в выходном регистре находился нуль, и мы дополнительно обслуживали только разряды TRIS обоих выводов. Теперь же для ведущего с микропрограммным управлением используются выходы с открытым коллектором.

Второй режим — полуавтоматический режим ведущего, в котором программист устанавливает некоторый разряд в регистре управления (например, чтобы устройство I²C сформировало состояние начала передачи). Он должен всегда ожидать только флага прерывания, сигнализирующего о завершении предшествующих действий. Таким же образом определяется и момент выдачи ведущим байта управления. В этом случае после обнаружения установленного флага прерывания проверяется разряд ACKSTAT в регистре SSPCON2, и только потом можно записать адрес в регистр SSPBUF и т.д.

На самом деле, разрешать соответствующее прерывание, как правило, не следует, поскольку время выполнения зачастую меньше затрат на вызов подпрограммы обработки прерывания.

Подробное описание процесса записи байта в последовательную память EEPROM типа 24LC01 и чтение из нее для этого режима представлено в презентации, наподобие документа PowerPoint, которую можно загрузить с Web-сайта компании Microchip.

Это описание также находится на прилагаемом к книге компакт-диске в файле `Pdf\i2c\i2c.pdf`. Рекомендуем также ознакомиться с советами по применению из файла `AN735.pdf`.

2.2.6. Инициализация I²C

Для инициализации полуавтоматического режима ведущего необходимо выполнить следующие действия.

1. Сконфигурировать выводы SCL и SDA как выходы.
2. Включить модуль MSSP: SSPEN = 1.
3. Выбрать режим в SSPCON:SSPM3:0 = 1000.
4. Установить скорость передачи в SSPADD.
5. Определить скорость нарастания выходного напряжения для установки скорости.
6. Задать CKE = 0 для выбора I²C, а не SMBUS.

2.2.7. Принцип действия I²C без аппаратного модуля

Хотя теперь у нас есть модуль MSSP, по-прежнему можно использовать старые подпрограммы для I²C — и даже для любых выводов. Если, кроме интерфейса I²C, применяется быстрый обмен данными по SPI между несколькими микроконтроллерами PIC, то в распоряжении не остается подчиненных модулю MSSP выводов. Представителей же PIC с двумя модулями SSP до сих пор не существует.

2.3. Модуль USART

Модуль USART (Universal Synchronous Asynchronous Receive Transmit) — это, как следует из его названия, модуль универсального применения, который может

конфигурироваться не только как асинхронный (UART), но и как синхронный интерфейс, хотя последний применяется редко. В отличие от него, UART —интерфейс совершенно незаменимый.

С помощью драйверной схемы (MAX232, MAX232A или аналогичной) из интерфейса UART получают интерфейс RS232, который до сих пор играет очень важную роль. К примеру, RS232 используют для подключения к электронике микроконтроллеров PIC принтеров, модемов и другого удаленного аппаратного обеспечения. Также не следует забывать и об обмене данными с ПК.

Еще одним важным применением этого интерфейса является обмен данными между двумя микроконтроллерами на плате или в пределах некоторого прибора.

2.3.1. Асинхронный режим (UART)

В режиме UART выход и вход могут управляться совершенно независимо друг от друга, то есть интерфейс — "полнодуплексный". Выход одной стороны соединяется со входом другой. Передача и прием — два полностью независимых процесса, однако, разумеется, они должны происходить с одинаковой выбранной скоростью передачи. Выходной вывод называется TxD, а входной — RxD.

Передача

Передача байта выполняется подобно тому, как это происходит в случае с интерфейсом SPI в режиме ведущего. Байт, подлежащий передаче, записывают в регистр передачи TXREG. Последний копируется в сдвиговый регистр TSR, как только тот освобождается, и сразу же начинается сдвиг битов в выходную линию. В отличие от SPI, первым здесь передается младший значащий разряд. Регистр TSR не адресуется.

При этом с помощью аппаратного обеспечения автоматически передаются стартовый и стоповый биты. За раз может передаваться, на выбор, восемь или девять бит, однако, поскольку регистр TXREG имеет разрядность только восемь бит, девятый бит должен записываться в регистр состояния TXSTA (разряд TX9D).

Что касается состояния регистров TXREG и TSR, то важную роль здесь выполняют два флага.

- TXMT в регистре состояния TXSTA принимает значение 1, если освобождается регистр TSR. Если записывается значение из TXREG в TSR, то TXMT=0.
- Флаг прерывания TXIF принимает значение 1, если освобождается регистр TXREG. Этот флаг опять сбрасывается, как только в TXREG записывается некоторое значение. В отличие от многих других флагов прерываний, сбрасывать этот флаг программно не нужно.

Как правило, флаг TXIF опрашивают для того, чтобы узнать, можно ли выполнить запись в TXREG, чтобы при этом не затереть предыдущий байт. Флаг TXMT используется реже и указывает о завершении процесса передачи по UART.

Существует две типичные последовательности команд при обращении с флагом TXIF (при этом предполагается, что подлежащий к отправке байт находится в регистре W).

В первой последовательности программа ожидает освобождения TXREG.

```
WAIXMT    BTFSS   PIR1,TXIF
          GOTO    WAIXMT
          MOVWF   TXREG
```

Передача по UART (например, со скоростью 9600) длится чуть дольше одной миллисекунды, однако в случае критичных ко времени применений так долго ждать никто не будет. Тогда используют последовательность команд SENDIF, в которой передача происходит только если регистр TXREG свободен. В противном случае, по неудавшейся передаче, на следующем проходе цикла опять предпринимается попытка отправить байт. Однако при этом, во время следующего прохода цикла, необходимо знать, должен ли отправляться байт.

Рассмотрим типичный пример, в котором отправке подлежат несколько байтов, находящихся в буфере. Мы предполагаем, что запись в буфер происходит где-то по ходу программы. Когда буфер готов, на его начало устанавливается указатель (BUFPTR) и в переменную-счетчик (BUFCNT) загружается число байтов, подлежащих передаче.

Подпрограмма SENDIF постоянно выясняет, нужно ли отправлять байт. Это можно сделать, если BUFCNT <> 0 и TXREG = 1. В этом случае SENDIF извлекает из буфера следующий байт и записывает его в TXREG, после чего указатель инкрементируется, а BUFCNT декрементируется.

```
SENDIF    BTFSS    PIR1,TXIF
          RETURN
          MOVF     BUFCNT
          SKPNZ
          RETURN
          MOVF     BUFPTR,W
          MOVWF    FSR
          MOVF     INDF,W
          MOVWF    TXREG
          INCF     BUFPTR
          DECF     BUFCNT
          RETURN
```

Разумеется, для этого метода существует множество альтернатив. Так, кто-то предпочитает использовать прерывания. В том случае, если необходимо сэкономить число переменных, можно попытаться отказаться от переменной BUFCNT, а для того, чтобы выяснить, пустой ли буфер, опрашивать указатель.

Во многих случаях при отправке также не хотят ожидать заполнения буфера. В этом случае запись выполняют в кольцевой буфер, где в любой момент в одном его конце происходит запись, а в другом — чтение.

Если оба регистра пусты (TXIF = 1 и TXMT = 1), тогда можно в быстрой последовательности записать друг за другом два байта в TXREG. Как только первый байт записан в TXREG, он сразу же переносится в TSR, после чего TXREG опять свободен. Устанавливается флаг TXIF и можно записывать в TXREG второй байт, после чего TXIF сразу же становится равным 0. Только когда первый байт полностью отправлен, загружается второй байт из TXREG в TSR, и флаг прерывания TXIF опять становится равным 1.

Рекомендуется перед каждой записью в регистр TXREG выяснять, свободен ли он, путем опроса флага TXIF. В том случае, если запись в TXREG выполняется до того как он освободился, значение в этом регистре затирается, так и не быв передано. Если отрезки времени между процессами записи гарантировано большие, то от подобного опроса можно и отказаться.

Прием

Прием байтов относительно прост. Флаг прерывания RXIF указывает на то, что в регистр приема поступил байт. Этот флаг опять сбрасывается чтением регистра RXREG. Девятый бит считывают (если он применяется) из регистра состояния RXSTA (разряд RX9D).

Если следующий байт принимается еще до того, как был прочитан предыдущий, то возникает уведомление об ошибке путем установки в 1 разряда OERR в регистре состояния RXSTA. В регистре RXSTA есть еще один разряд, информирующий об ошибке, — FERR, который устанавливается, если обнаруживается нарушение кадрирования (например, если для передатчика и приемника установлены разные скорости передачи).

2.3.2. Адресуемый USART (AUSART)

При внутреннем использовании интерфейса UART девятый бит можно использовать на свое усмотрение. К примеру, он может выступать в качестве идентификатора, бита проверки или просто девятого бита данных.

Новые USART в отношении этого бита предлагают сервис, поддерживающий адресацию, если к интерфейсу UART подключено несколько участников. В таком случае USART еще называют "адресуемыми USART".

Для того чтобы воспользоваться этим сервисом, передатчик должен устанавливать девятый бит равным 1, если остальные восемь бит представляют собой адрес. Если речь идет о данных, тогда он устанавливает девятый бит равным 0.

Сервис заключается в том, что приемник блокирует прием байтов данных до тех пор, пока не получит идентифицирующий его адрес. Модуль решает эту задачу самостоятельно и тем самым избавляет от лишней работы центральный процессор.

В режиме простоя он блокирует прием данных, установив бит ADEN = 1 в регистре RCSTA. Блокировка препятствует загрузке содержимого сдвигового регистра в регистр RCREG и записи девятого бита в регистр RCSTA. Также не наступает прерывание.

В этом состоянии принимаются исключительно адреса (бит 9 =1). Если модуль приема распознает свой адрес, то он устанавливает разряд ADEN = 0, чтобы иметь возможность принять последующие данные.

2.3.3. Инициализация

Регистры TXSTA и RXSTA содержат не только разряды состояния, но и, преимущественно, разряды конфигурации. Все условия работы модуля USART должны определяться с помощью этих двух регистров. За подробной информацией мы, как всегда, рекомендуем обратиться к техническим описаниям.

Не забывайте конфигурировать выводы TxD и RxD как выход и вход соответственно. Также должны устанавливаться соответствующие разряды, отвечающие за прерывания. И, разумеется, нельзя забывать об установке скорости передачи.

Выбор скорости передачи

Скорость передачи устанавливается посредством записи некоторого значения в регистр SPBRG. В технических описаниях это значение обозначается как "X" и извлекается из таблиц спецификаций или определяется самостоятельно с помощью простой формулы.

Можно, конечно, сберечь силы, используя вместо формулы таблицы из технических описаний, однако мы уже нашли в этих таблицах несколько незначительных ошибок, поэтому рекомендуем взять в руки калькулятор.

Для выражения взаимосвязи между скоростью передачи и значением X существуют две различные формулы. Какую из них следует применить, легко узнать, проведя следующие рассуждения. Для "быстрого" варианта в регистре состояния TXSTA разряд BRGH устанавливают равным 1, а для "медленного" — равным 0.

Формулы для скорости передачи:

- Скорость = $F_{osc}/64 \cdot (X+1)$ — для BRGH = 0;
- Скорость = $F_{osc}/16 \cdot (X+1)$ — для BRGH = 1.

Поскольку X должно быть целым числом < 255, вычисленная скорость передачи как правило, не является целочисленным значением. По этой причине для X всегда следует выбирать такое целочисленное значение, для которого полученная скорость передачи как можно ближе к желаемой.

Разрешаем представленные выше формулы относительно X + 1:

- $X + 1 = F_{osc} /(64 \cdot$ Скорость) — для BRGH = 0;
- $X + 1 = F_{osc} /(16 \cdot$ Скорость) — для BRGH = 1.

Если здесь подставнть желаемую скорость передачи, то для X + 1, обычно, целочисленное значение не получается, и потому его округляют. Возникающая при округлении относительная погрешность соответствует относнтельной погрешности скорости передачи. Если X + 1 округляют вверх, то фактическая скорость передачи меньше желаемой, если же ее округляют вниз, то скорость передачи увеличивается.

Но как узнать, какое значение следует выбрать для BRGH? Ответ очень прост: значение X + 1 при BRGH = 1 в четыре раза больше, чем при BRGH = 0. До тех пор, пока X не больше 255, это — преимущество, поскольку при большом X + 1 округление до целочисленного значения дает незначительную относительную погрешность.

Пример

Желаемая скорость передачи 9600; F_{osc} = 4 МГц.

Вначале выбираем BRGH = 1 и получаем:

$X + 1 = 4000000/16 \cdot 9600 = 26{,}04.$

Если мы округляем до X + 1 = 26 (X=25), то погрешность округления составляет 0,04. Это соответствует относительной погрешность 0,16%.

Для BRGH = 0 мы бы получили

$X + 1 = 4000000/64 \cdot 9600 = 6{,}51.$

Если мы округляем до X + 1 = 7, то погрешность округления составляет 0,49. Относительная погрешность составляет целых 7% (для малой скорости передачи).

Если скорость передачи меньше на 7%, то длительность бита на 7% больше. Звучит безобидно, но это не так. Поскольку ошибки в длительностях битов суммируются, девятый бит поступает позже на 63% длительности бита. Рассчитывайте, чтобы опрос происходил приблизительно посередине бита. При этом, конечно же, предполагается, что другой участник также использует округленную скорость.

Самый безопасный вариант — когда для обоих участников погрешности округления одинаковы (например, два мнкроконтроллера PIC с одной и той же частотой F_{osc}), при условии, что значения F_{osc} реализованы достаточно точно.

Из представленных выше вычислений становится понятно, что некоторые комбинации F_{OSC} и желаемой скорости передачи или совсем не могут быть реализованы, или реализуются с недостаточной точностью. На это есть две причины.

Если само X при BRGH = 0 становится больше 255, то реализация невозможна. Это имеет место только при большом значении F_{OSC} и очень низкой скорости передачи. Таким образом, еще можно реализовать скорость передачи 1200 при F_{OSC} = 20 МГц (X + 1 = 132).

С другой стороны, при малом F_{OSC} и большой скорости передачи значение X + 1 может становиться очень малым, даже если выбирают BRGH = 1. В этом случае погрешность округления может, смотря по обстоятельствам, приводить к неприемлемой точности скорости передачи.

2.3.4. Улучшенный USART (EUSART)

Улучшенный (enhanced) USART реализован только в семействе PIC18, и улучшение касается скорости передачи. Своим существованием EUSART обязан шине LIN, суть которой заключается в том, что ведущее устройство в поле, предваряющем передачу данных, вначале посылает сигнал пробуждения из "спящего" режима, а затем — байт синхронизации (55_h) в целях дополнительной подстройки скорости передачи.

EUSART позволяет подстраивать с помощью байта синхронизации значение скорости передачи X в соответствии со скоростью передачи ведущего устройства. При этом, естественно, слишком малые значения X лучше не использовать. Для того, чтобы сделать подстройку скорости передачи более "тонкой", EUSART позволяет применять формулу, в которой значение X + 1 еще в четыре раза больше, чем при использовании вышеуказанной формулы для BRGH = 1.

Регистр SPRG, в который должно записываться значение скорости передачи X, состоит теперь из двух байтов (SPRGH:SPRG).

$X + 1 = F_{OSC} / (4 \cdot \text{Скорость})$ — для BRGH = 1.

2.3.5. Применение RS232

Интерфейс RS232 играет особую роль в лабораторных условиях, поскольку с его помощью организуют обмен данных опытных образцов с ПК.

Через модуль USART можно очень быстро и просто передавать ПК информацию об аппаратном обеспечении (например, значения характеристической кривой или переменные в зависимости от различных параметров процесса и управляющих воздействий). В простейшем случае достаточно в определенном месте программы записать соответствующее значение в регистр передачи (TXREG). Таким элегантным способом можно также реализовать поиск программных ошибок.

И наоборот, для целей тестирования через этот же простой интерфейс от ПК можно передавать команды и параметры для проверки аппаратного обеспечения микроконтроллера PIC. Например, при обнаружении нажатия клавиши ПК модулю USART микроконтроллера PIC через RS232 передается некоторый символ, определяющий ход выполнения или параметры программы (скажем, таким образом можно управлять параметрами работы двигателя: ускорение или замедление; останов; направление вращения).

При разработке устройств управления и регулирования подобная методика с применением простейших средств становится незаменимым инструментом.

Процедуры

При новых разработках мы выбираем для опытных образцов тип микроконтроллера PIC ресурсами, немного превышающими требуемые для работы аппаратного обеспечения, — нам потребуется несколько дополнительных выводов для целей диагностики. Выводы USART во всех случаях оставляем свободными. Это требуется, прежде всего, тогда, когда аппаратное обеспечение находится на этапе ознакомления (неизвестное устройство). Для этого на стороне ПК необходимо написать несколько небольших программ, чтобы организовать прием значений от RS232 с их последующим сохранением и представлением в виде таблицы или графика.

К сожалению, языки высокого уровня обходятся с RS232 очень сурово. Зачастую даже программное обеспечение ПК под Windows слишком медленное, чтобы обработать значение непосредственно во время приема. В таком случае значения записывают в файл и просматривают позже.

Рассмотрим простейший случай применения. Каждый раз при получении измеренного значения его записывают в регистр TXREG. При этом необходимо немного подумать о временных соотношениях. Если интервалы между считываниями измеряемых величин достаточно большие, то можно исходить из того, что в каждый момент при получении нового значения регистр TXREG пуст. Следует учитывать, что для передачи одного байта при скорости передачи 9600 требуется немногим более одной миллисекунды.

Если измеренное значение состоит из двух байтов, а интервал между измерениями заметно больше двух миллисекунд, тогда оба байта можно быстро записать один за другом в TXREG. При записи в регистр TXREG, когда он не пуст, один из байтов будет потерян. Если измеренные значения состоят только из одного байта, то обычно, не так уж и трагично, но в случае двух байтов старший байт может быть утерян. Проверка целостности принятых данных — задача программы на стороне ПК.

Если по RS232 требуется передавать целые наборы данных, то их записывают в корректной последовательности в некоторый буфер, а в определенном месте основного цикла для выдачи данных в RS232 вызывают описанную выше подпрограмму `SENDIF`.

Новые данные записываются в буфер только после передачи всего его содержимого, иначе возникнут искажения. Работа с кольцевым буфером требует намного больше внимания.

При передаче наборов целесообразно вводить в данные некоторую избыточность, чтобы приемник знал, где начинается новый набор данных. Согласитесь, что потеря байта вследствие ошибки передачи — не такой уж невероятный вариант.

2.4. Шина CAN

Совсем другие задачи возникают при передаче данных к отдаленным частям машины. В этой связи очень интересными решениями являются шины CAN и LIN, которые способны обеспечить надежный и упорядоченный обмен информацией между различными участниками с помощью хорошо продуманных физических и логических концепций и сравнительно простой протокольной надстройки.

Эти две шины первоначально были разработаны для обмена данными в пределах транспортных средств, но применяются также и в таких сферах как измерительная техника, промышленная автоматизация и домашние сети. Особым преимуществом шины CAN является то, что она может работать в зашумленном окружении.

Перед тем как мы должны были разработать первый проект для обмена данными по шине CAN, у нас сложилось впечатление, что эта шина сложная и для организации работы с ней потребуется масса времени, однако, к счастью, мы ошибались. Шина CAN проста по структуре и в обслуживании. Протокол работает с фильтрами и масками для того, чтобы логическое устройство приема (например микроконтроллер PIC) могло на аппаратном уровне быть разгружено от чтения ненужных сообщений. Если понять этот принцип, то обслуживание шины становится очень простым.

Для разработчиков, использующих микроконтроллеры PIC, есть две возможности простой реализации интерфейса CAN: они могут или выбрать представителя из семейства PIC18 с аппаратным модулем CAN, или же воспользоваться внешним контроллером CAN (MCP2515), который обслуживается через интерфейс SPI. Последнее решение имеет то преимущество, что можно применять любой микроконтроллер PIC, если он оснащен интерфейсом SPI.

О том, какое место отводится шине CAN компанией Microchip, можно судить по тому, что в руководстве по эксплуатации для PIC18 одному только модулю CAN посвящено около 80 страниц. Однако мы не рекомендуем читать эти страницы новичкам в качестве вступительных лекций.

Для начинающего разработчика будет полезно ознакомиться с демонстрационной платой CAN от Microchip, с помощью которой можно удостовериться, что все понято правильно. Кроме того, продвинутый пользователь может использовать эту плату в качестве инструмента разработки (см. главу 10).

2.4.1. Введение в CAN

Перечислим наиболее важные свойства шины CAN.

- Отсутствие ведущего устройства, распределяющего права шины. Каждый узел имеет равное право с другими. Для шины CAN речь идет об архитектуре со многими ведущими.
- Управление захватом шины реализовано в соответствии с приоритетностью сообщений. Отсюда следует короткое время задержки для сообщений с высоким приоритетом. Это важно, поскольку сеть на транспортном средстве должна соответствовать требованиям работы в режиме реального времени.
- Поддержка широковещательной передачи (то есть, одно сообщение может посылаться одновременно нескольким приемникам). Это возможно, поскольку шина CAN не использует адресов, а каждое сообщение обладает однозначным идентификатором, который одновременно содержит приоритет сообщения.
- Шина CAN была разработана для применения с незначительным потоком данных, поэтому она позволяет значительно сократить скорость передачи данных в случае, если требуется минимальная пропускная способность. Это положительно сказывается в плане уменьшения электромагнитного излучения. Для случаев применения с расширенным потоком данных в распоряжении есть скорость передачи данных до 1 Мбит/с.
- Поскольку шина была разработана для применения в производстве транспортных средств, в которых очень распространены электромагнитные помехи, она располагает обширной системой обработки ошибок. Обнаружение и обработка ошибок предотвращает паралич всей сети из-за одного неисправного участника на шине.

Физическое устройство CAN

Шина CAN состоит из двухпроводной линии, которая на обоих концах заканчивается резисторами, исключающими отражения (рис. 2.2). Двухпроводная линия может быть выполнена с параллельным расположением проводов или как витая пара.

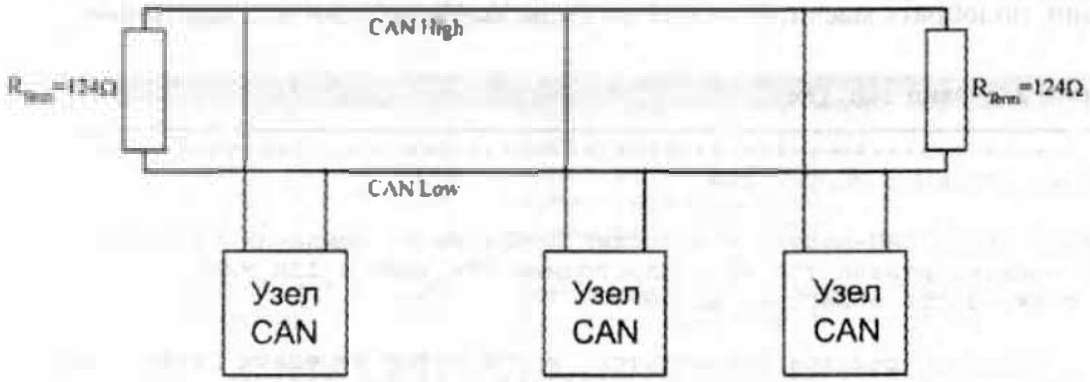


Рис. 2.2. Устройство сети CAN

При высокой частоте следования битов рекомендуется экранирование. Отдельные узлы подсоединяются к шине CAN также с помощью двухпроводной линии. Рекомендуется делать эти участки подключения как можно более короткими. Для сети со скоростью передачи 1 Мбит/с длина не должна превышать 0,3 м (рекомендация CiA Draft Standard 102 Version 2.0).

Сигналы в сети CAN

Для передачи шина CAN использует дифференциальный сигнал с дополняющими уровнями (рис. 2.3).

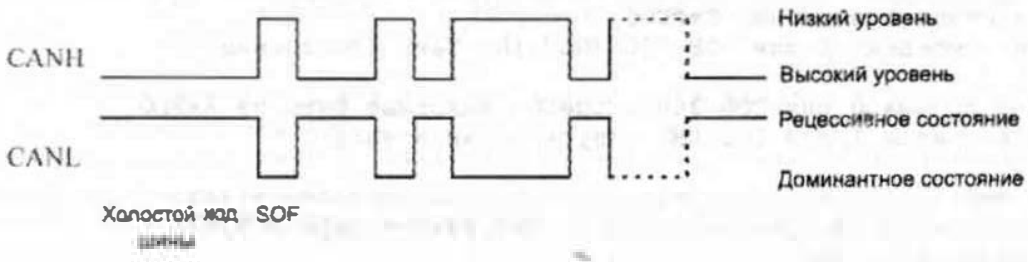


Рис. 2.3. Сигнал в шине CAN

В логических схемах высокий и низкий уровни обычно имеют одинаковую значимость, поэтому необходимо быть уверенным в том, что два логических уровня не смешиваются напрямую в одной точке, поскольку полученное напряжение может иметь непредвиденное значение и оказаться в запрещенной области. Для шины CAN низкий уровень — доминантное состояние, а высокий — рецессивное. Если к шине прикладываются одновременно два уровня, то в результате получается низкий уровень. Основное состояние шины CAN (состояние холостого хода) — рецессивное состояние, то есть высокий уровень.

2.4.2. Пример программы для CAN

Рассмотрим один небольшой проект для CAN.

Все исходные файлы проекта Buchcan находятся на прилагаемом к книге компакт-диске в папке CAN\Buchcan.

CAN.INC

Начнем с файла can.inc (листинг 2.1), содержащего все подпрограммы и инициализирующие значения для MCP2510. Соответствующие таблицы называются TAB_CAN_INI, TAB_TX0_INI и TAB_TX1_INI. В них можно, в зависимости от ситуации, подобрать маски, фильтры, разряды конфигурации или идентификатор кадра.

Листинг 2.1. Файл can.inc

```
;*********************************************************************
; Общее описание модуля CAN:
;
; Автор этого CAN-модуля - господин Кошински из компании Microchip.
; Мы модифицировали только подпрограммы ISR_RxB0 и_ISR_RxB1,
; а также часть обработки данных.
;
; В качестве параметра указывается, какой буфер передачи должен быть
; передан. Перед этим должны быть обновлены байты данных кадра.
; В случае тактового импульса первый байт данных - "Состояние". Он
; обновляется при изменении, т.е. по командам запуска и останова узла.
; При TxPDO1 в 'serv_tx' весь буфер обновляется и затем отправляется.
;
; Полученные кадры автоматически принимаются от MCP2510 (с учетом масок
; и фильтров, настроенных TAB_CAN_INI). Это устанавливает флаги
; в CANINTF, которые постоянно опрашиваются в подпрограмме 'servcan'.
; В части 'Обработка данных' выполняется обработка принятых сообщений.
;
; В начале узел отключен (knot_go = 0 ), т.е. обмен данными отсутствует.
; Однако тактовый импульс передается постоянно.
;
; Буфер передачи 0 для COB=180H+NodeID; входные биты и передача
;  аналогового значения TxPDO1
; Буфер передачи 1 для COB=700H+NodeID; Такт (состояние)
;
; Буфер приема 0 для COB=200H+NodeID: выходные биты из RxPDO1
; Буфер приема 1 для COB=0H; Запуск/останов узла
;
;*********************************************************************
; Набор команд SPI для автономного CAN-контроллера MCP2510
; с интерфейсом SPI

#define     CAN_Write     B'00000010'
#define     CAN_Read      B'00000011'
#define     CAN_RTS0      B'10000001'
#define     CAN_RTS1      B'10000010'
#define     CAN_RTS2      B'10000100'
#define     CAN_Status    B'10100000'
#define     CAN_BitMod    B'00000101'
#define     CAN_Reset     B'11000000'
#define     Dummy         0x00

; Некоторые важные адреса в MCP2510

#define     @TXB0         0x31   ;Адрес первого байта в буфере передачи 0
#define     @txb0d0       0x36
#define     @TXB1         0x41   ;Адрес первого байта в буфере передачи 1
#define     @TXB2         0x51   ;Адрес первого байта в буфере передачи 2
```

Листинг 2.1. Продолжение

```
#define    @RXB0          0x61   ;Адрес первого байта в буфере приема 0
#define    @rb0sidl       0x62
#define    @rb0d0         0x66
#define    @RXB1          0x71   ;Адрес первого байта в буфере приема 1
#define    @rb1sidl       0x72
#define    @rb1d0         0x76
#define    @CANCTRL       0x0F   ;Адрес регистра CANCTRL
#define    @CANSTAT       0x0E   ;Адрес регистра CANSTAT
#define    @CANINTF       0x2C   ;Адрес регистра CANINTF
#define    @CNF1          0x2A   ;Адрес регистра CNF1
#define    @CNF2          0x29   ;Адрес регистра CNF2
#define    @CNF3          0x28   ;Адрес регистра CNF3

#define    @TXRTSCTRL     0x0D   ;Адрес регистра RTS-Pin-Control
#define    @BFPCTRL       0x0C   ;Адрес регистра BF-Pin-Control

;**********************************************************************
; Макрос выбора кристалла
;**********************************************************************
; SCS_CAN
; Задача: Активизация выбора кристалла (CS) без передачи переменных

SCS_CAN    MACRO
           bcf    PORTC, 2      ;Выбор кристалла MCP2510 установлен
           ENDM
;**********************************************************************
; CCS_CAN
; Задача: Деактивация выбора кристалла (CS) без передачи переменных

CCS_CAN    MACRO
           bsf    PORTC, 2      ;Выбор кристалла MCP2510 сброшен
           ENDM

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Макроопределения для SPI
;**********************************************************************
; SPI_OUT
; Задача: Значение регистра W передаем по SPI и ожидаем готовности
; Вход: W
; Выход: SSPBUF, игнорируется

SPI_OUT    MACRO
           local  Loop_OUT
           movwf  SSPBUF        ;Новые данные для передачи
Loop_OUT   btfss  PIR1,SSPIF    ;Передача завершена
           goto   Loop_OUT
           bcf    PIR1,SSPIF    ;Сбрасываем флаг SPI
           ENDM

;**********************************************************************
; SPI_IN
; Задача: Считываем байт из SPI; готово, когда передача завершена
; Вход: -
; Выход: SSPBUF и W

SPI_IN     MACRO
           local  Loop_IN
```

Листинг 2.1. Продолжение

```
            movlw  Dummy
            movwf  SSPBUF          ;Передаем Dummy
Loop_IN     btfss  PIR1,SSPIF      ;Передача завершена
            goto   Loop_IN
            movf   SSPBUF,w        ;Считываем буфер
            bcf    PIR1,SSPIF      ;Сбрасываем флаг SPI
            ENDM

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Сброс MCP2510
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Hard_Reset
; Задача: Сброс MCP2510 по сигналу сброса
; Вход/выход отсутсвует

Hard_Reset  MACRO                  ;Аппаратный сброс MCP2510
            bcf    PORTC,1         ;Устанавливаем низкий уровень в линии
            NOP
            bsf    PORTC,1         ; Устанавливаем высокий уровень в линии
            ENDM

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Soft_Reset
; Задача: Сброс MCP2510 по команде
; Вход/выход отсутсвует

Soft_Reset  MACRO                  ;Программный сброс MCP2510
            SCS_CAN                ;Устанавливаем линию выбора кристалла
            movlw  CAN_Reset       ;Загружаем команду "Reset"
            SPI_OUT                ;Передаем команду
            CCS_CAN                ;Сбрасываем линию выбора кристалла
            ENDM

;**********************************************************************
;Таблица для инициализации MCP2510

;TAB_CAN_INI требует позицию регистра, записанной в регистр W
;Таблица начинается со старших адресов. Это означает, что первый регистр
;расположен в конце таблицы

; Таблица инициализационных значений для регистра MCP2510.
; Некоторые значения зависят от определенных настроек

TAB_CAN_INI
            addwf  PCL, f
            retlw  0x00            ;Dummy
            retlw  0x86            ;Регистр CANCTRL
            retlw  0x00            ;Регистр CANSTAT (только для чтения)
            retlw  0x00            ;Регистр EFLG
            retlw  0x00            ;Регистр CANINTF
            retlw  0xff            ;Регистр CANINTE
            goto   getcnf1         ; 0x03
                                   ;Регистр CNF1 (скорость передачи по CAN)
            retlw  0xF1            ;Регистр CNF2 (скорость передачи по CAN)
            retlw  0x05            ;Регистр CNF3 (скорость передачи по CAN)
            retlw  0x00            ;Регистр RXM1EID0
            retlw  0x00            ;Регистр RXM1EID8
```

Листинг 2.1. Продолжение

```
            retlw  0xE0          ;Регистр RXM1SIDL
            retlw  0xFF          ;Регистр RXM1SIDH
            retlw  0x00          ;Регистр RXM0EID0
            retlw  0x00          ;Регистр RXM0EID8
            retlw  0xC0          ;Регистр RXM0SIDL
            retlw  0xFF          ;Регистр RXM0SIDH

            retlw  0x9F          ;Регистр CANCTRL ;CLKout=16/8 МГц =2MHz
            retlw  0x00          ;Регистр CANSTAT (только для чтения)
            retlw  0x00          ;Регистр REC (только для чтения)
            retlw  0x00          ;Регистр TEC (только для чтения)
            retlw  0x00          ;Регистр RXF5EID0
            retlw  0x00          ;Регистр RXF5EID8
            retlw  0x00          ;Регистр RXF5SIDL
            retlw  0x00          ;Регистр RXF5SIDH
            retlw  0x00          ;Регистр RXF4EID0
            retlw  0x00          ;Регистр RXF4EID8
            retlw  0x00          ;Регистр RXF4SIDL
            retlw  0x00          ;Регистр RXF4SIDH
            retlw  0x00          ;Регистр RXF3EID0
            retlw  0x00          ;Регистр RXF3EID8
            retlw  0x00          ;Регистр RXF3SIDL
            retlw  0x00          ;Регистр RXF3SIDH

            retlw  0x97          ;Регистр CANCTRL    ;такт - ровно 500кГц
            retlw  0x00          ;Регистр CANSTAT (только для чтения)
            retlw  0x00          ;Регистр TXRTSCTRL  ;цифровой вход
            retlw  0x2C          ;Регистр BFPCTRL    ;цифровой выход
            retlw  0x00          ;Регистр RXF2EID0
            retlw  0x00          ;Регистр RXF2EID8
            retlw  0x00          ;Регистр RXF2SIDL
            retlw  0x00          ;Регистр RXF2SIDH   ;0H
            retlw  0x00          ;Регистр RXF1EID0
            retlw  0x00          ;Регистр RXF1EID8
            retlw  80h           ;Регистр RXF1SIDL
            retlw  0x4c          ;Регистр RXF1SIDH
            retlw  0x00          ;Регистр RXF0EID0
            retlw  0x00          ;Регистр RXF0EID8
            retlw  80h           ;Регистр RXF0SIDL
            retlw  0x4C          ;Регистр RXF0SIDH   ;200H+64h/65h

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; getcnf1
; Задача: Устанавливаем возвращаемое значение в зависимости от скорости
;         передачи. Выполняем переход к таблице ADDWF PCL.
; Вход: jmps (состояние перемычки, считывается только один раз в начале!)
; Выход: W

getcnf1     movf   jmps.w        ; Переменная, в которую записывается
            andlw  03h           ;  состояние перемычки CAN
            addwf  pcl
            retlw  03h           ;125kB
            retlw  03h           ; явно не реализовано
            retlw  01h           ;250kB
            retlw  00h           ;500kB

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Листинг 2.1. Продолжение

```
;TAB_TX0_INI требует позиции регистра, записанной в регистр W
;Таблица начинается со старшего адреса 03dh. Это означает, что первый
;регистр находится в конце таблицы
; TAB_TX0_INI и TAB_TX1_INI
; Задача: Заполнение буфера передачи.
;         Вызывается команда ADDWF PCL.
; Вход: W
; Выход: W

TAB_TX0_INI
              addwf  PCL,f
              retlw  0              ;dummy
              retlw  0x00           ;Регистр TXB0D7
              retlw  0x00           ;Регистр TXB0D6
              retlw  0x00           ;Регистр TXB0D5
              retlw  0x00           ;Регистр TXB0D4
              retlw  0x00           ;Регистр TXB0D3
              retlw  0x00           ;Регистр TXB0D2
              retlw  0x00           ;Регистр TXB0D1
              retlw  0x001          ;Регистр TXB0D0
              retlw  0x08           ;Регистр TXB0DLC
              retlw  0x00           ;Регистр TXB0EID0
              retlw  0x00           ;Регистр TXB0EID8
              retlw  80h            ;Регистр TXB0SIDL
              retlw  0x3c           ;Регистр TXB0SIDH
              retlw  0x00           ;Регистр TXB0CTRL   ;180H+64h/65h
;
;TAB_TX1_INI требует позиции регистра, записанной в регистр W
;Таблица начинается со старшего адреса 04dh. Это означает, что первый
;регистр находится в конце таблицы

TAB_TX1_INI
              addwf  PCL,f
              retlw  0              ;dummy
              retlw  0x07           ;Регистр TXB1D7
              retlw  0x06           ;Регистр TXB1D6
              retlw  0x05           ;Регистр TXB1D5
              retlw  0x04           ;Регистр TXB1D4
              retlw  0x03           ;Регистр TXB1D3
              retlw  0x02           ;Регистр TXB1D2
              retlw  0x01           ;Регистр TXB1D1
              retlw  0x00           ;Регистр TXB1D0
              retlw  0x08           ;Регистр TXB1DLC
              retlw  0x00           ;Регистр TXB1EID0
              retlw  0x00           ;Регистр TXB1EID8
              retlw  80h            ;Регистр TXB1SIDL
              retlw  0xec           ;Регистр TXB1SIDH
              retlw  0x00           ;Регистр TXB1CTRL   ;700H+64h/65h

;*****************************************************************************
; Функции SPI
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Записи регистра
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; SPI_Write
; Задача: Макрос, который загружает переменные SPI_Address и API_Data
;         и вызывает подпрограмму Write_reg
```

Листинг 2.1. Продолжение

```
  Входные параметры: адрес и данные

SPI_Write     MACRO  Address, Data
              movlw  Address
              movwf  SPI_Address
              movlw  Data
              movwf  SPI_Data
              call   Write_Reg
              ENDM

;*****************************************************************************
Write_Reg
  Задача: Запись SPI_Data по SPI_Address с помощью SPI_OUT
          Линия CS включается и выключается.
  Вход: SPI_Address и SPI_Data
  Выход: -

Write_Reg
              SCS_CAN                ;Устанавливаем линию выбора кристалла

              movlw  CAN_Write       ;Загружаем команду "Запись в регистр"
              SPI_OUT                ;Передаем команду

              movf SPI_Address,w     ;Загружаем адрес регистра
              SPI_OUT                ;Передаем адрес

              movf SPI_Data,w        ;Загружаем значение данных
              SPI_OUT                ;Передаем данные

              CCS_CAN                ;Сбрасываем линию выбора кристалла
              return                 ;Выходим из подпрограммы

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  Чтение регистра
;*****************************************************************************
  SPI_Read
  Задача: Макрос, загружает SPI_Address и вызывает Read_reg
  Входные параметры: Address
  Выход: SPI_Data

SPI_Read      MACRO  Address         ;Результат = SPI_Data
              movlw  Address
              movwf  SPI_Address
              call   Read_Reg
              ENDM
;*****************************************************************************
  SPI_Move
  Задача: Макрос, считывает в Target с помощью SPI_OUT и SPI_IN
          Используется линия.
  Входные параметры: Address и Target
  Выход: SPI_Data, W и 'Target'

SPI_Move      MACRO  Address, Target
              movlw  Address
              movwf  SPI_Address
              call   Read_Reg
              movf   SPI_Data,w
```

Листинг 2.1. Продолжение

```
            movwf  Target
            ENDM
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Read_Reg
; Задача: Считывает в SPI_Data с помощью SPI_OUT и SPI_IN
;         Используется линия CS.
; Вход: SPI_Address
; Выход: SPI_Data

Read_Reg
            SCS_CAN              ;Устанавливаем линию выбора кристалла
            movlw  CAN_Read      ;Загружаем команду "Чтение из регистра"
            SPI_OUT              ;Передаем команду

            movf SPI_Address,w   ;Загружаем адрес регистра
            SPI_OUT              ;Передаем адрес

            SPI_IN               ;Передаем Dummy, считываем данные в w
            movwf  SPI_Data      ;Записываем значение данных

            CCS_CAN              ;Сбрасываем линию выбора кристалла
            return               ;Выходим из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Модификация регистра
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; SPI_BitMod
; Задача: Макрос, загружает SPI_Adress, SPIData и SPI_Mask для Modify_Reg
;         Используется линия CS.
; Параметры: Address, Data, Mask

SPI_BitMod  MACRO  Address, Data, Mask
            movlw  Address
            movwf  SPI_Address
            movlw  Data
            movwf  SPI_Data
            movlw  Mask
            movwf  SPI_Mask
            call   Modify_Reg
            ENDM
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Modify_Reg
; Задача: Выполняет команду поразрядной модификации
; Вход: SPI_Address, SPI_Mask, SPI_Data
; Выход: -

Modify_Reg
            SCS_CAN              ;Устанавливаем линию выбора кристалла

            movlw  CAN_BitMod    ;Загружаем команду "Bit Modify"
            SPI_OUT              ;Передаем команду

            movf SPI_Address,w   ;Загружаем адрес регистра
            SPI_OUT              ;Передаем адрес

            movf SPI_Mask,w      ;Загружаем маску
            SPI_OUT              ;Передаем маску
```

Листинг 2.1. Продолжение

```
            movf   SPI_Data,w    ;Загружаем значение данных
            SPI_OUT              ;Передаем данные

            CCS_CAN              ;Сбрасываем линию выбора кристалла
            return               ;Выходим из подпрограммы

;*****************************************************************
; Запись сообщения
;-----------------------------------------------------------------
; SPI_Write_Msg
; Задача: Макрос, загружает SPI_Buffer и SPI_Adress для Write_Message
; Параметр: MsgBuffer(RAM-адрес), TxBuffer(регистр в MCP2510)
;          => SPI_Buffer  => SPI_Address

SPI_Write_Msg  MACRO MsgBuffer, TxBuffer
               movlw MsgBuffer
               movwf SPI_Buffer
               movlw TxBuffer
               movwf SPI_Address
               call  Write_Message
               ENDM

;-----------------------------------------------------------------
; Write_Message
; Задача: Передает SPI_Buffer (в RAM) в буфер сообщения
;                          ( SPI_Adress ) в MCP2510
; Вход: SPI_Buffer, SPI_Address
; Выход: -

Write_Message
; Расчет длины сообщения
            movlw  0x05          ;Количество регистров для записи
                                 ;  (5 ID-байтов)
            movwf  index
            movf SPI_Buffer, w   ;Считываем начальный адрес буфера
                                 ;  сообщения в PIC
            addlw  0x04          ;Указатель на байт длины данных
            movwf  FSR           ;Переносим адрес в регистр FSR
            movf   INDF, w       ;Считываем косвенно код длины данных
            andlw  0x0F          ;Маска разрядов длины данных в буфере
            addwf  index, f      ;Добавляем количество байтов данных к
                                 ;  к количеству ID-байтов
            SCS_CAN              ;Устанавливаем линию выборки кристалла
            movlw  CAN_Write     ;Загружаем команду "Запись в регистр"
            SPI_OUT              ;Передаем команду
            movf SPI_Address,w   ;Загружаем начальный адрес буфера
                                 ;  сообщения в MCP
            SPI_OUT              ;Передаем адрес
            movf   SPI_Buffer,w  ;Загружаем начальный адрес буфера в PIC
            movwf  FSR           ;Переносим адрес в регистр FSR
loop_SM_Data movf  INDF, w       ;Считываем косвенно значение для MCP2510
            SPI_OUT              ;Передаем данные
            incf   FSR, f
            decfsz index, f      ;Декрементируем индекс и выходим из
                                 ;  цикла, если index = 0
            goto loop_SM_Data
            CCS_CAN              ;Сбрасываем линию выборки кристалла
```

Листинг 2.1. Продолжение

```
            return                  ;Выходим из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Чтение сообщения
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; SPI_Read_Msg
; Задача: Макрос, загружает SPI_Buffer и SPI_Adress для Read_Message
; Параметры: RxBuffer(в MCP2510), MsgBuffer(в RAM)
;         => SPI_Address    => SPI_Buffer

SPI_Read_Msg   MACRO RxBuffer, MsgBuffer
               movlw RxBuffer
               movwf SPI_Address
               movlw MsgBuffer
               movwf SPI_Buffer
               call  Read_Message
               ENDM

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Read_Message
; Задача: Передача поступившего сообщения в SPI_Buffer
; Вход: SPI_Address, SPI_Buffer

Read_Message
               movlw  0x05          ;Количество регистров для чтения
                                    ; (5 ID-байтов)
               movwf  index
               SCS_CAN              ;Устанавливаем линию выбора кристалла
               movlw  CAN_Read      ;Загружаем команду "Чтение в регистр"
               SPI_OUT              ;Передаем команду
               movf SPI_Address,w   ;Загружаем начальный адрес буфера
                                    ; приема в MCP2510
               SPI_OUT              ;Передаем адрес
               movf SPI_Buffer,w    ;Загружаем адрес первого регистра
               movwf  FSR           ;Перемещаем адрес в регистр FSR
loop_RM_ID     movlw  0x00          ;Загружаем Dummy
               SPI_IN               ;Передаем Dummy, считываем данные в w
               movwf  INDF          ;Косвенная запись значения для MCP2510
               incf   FSR, f
               decfsz index, f      ;Декрементируем индекс и выходим из
                                    ;  цикла, если index = 0
               goto   loop_RM_ID
               andlw  0x0F          ;Маска кода длины данных в регистре W
               movwf  index         ;Перемещаем количество байт данных
                                    ; в index
               btfss  STATUS, Z     ;Выходим, если 0 байт данных
               goto loop_RM_Data    ;Переходим к чтению байтов данных
               CCS_CAN              ;Сбрасываем линию выбора кристалла
               return               ;Выходим из подпрограммы

;Чтение байтов данных в буфер в PIC

loop_RM_Data   SPI_IN               ;Передаем Dummy, считываем данные в w
               movwf  INDF          ;Косвенная запись значения для MCP2510
               incf   FSR, f
               decfsz index, f      ;Декрементируем индекс и выходим из
                                    ; цикла, если index = 0
```

Листинг 2.1. Продолжение

```
            goto loop_RM_Data
            CCS_CAN              ;Сбрасываем линию выбора кристалла
            return               ;Выходим из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;Запрос на передачу
;Передает команду 'Запрос на передачу' для передачи буфера TxBn
;TxB0 = CAN_RTS0, TxB1 = CAN_RTS1, TxB2 = CAN_RTS2,
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;SPI_RTS
;Задача: Макрос, предает в MCP2510 команду 'запрос на передачу'
;Параметры: TxBn [CAN_RTS0, CAN_RTS1, CAN_RTS2]

SPI_RTS     MACRO  TxBn
            movlw  TxBn
            call   Send_RTS
            ENDM

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;Send_RTS
;Задача: Передает в MCP2510 команду (RTS)
;Вход: W

;Send_RTS
;Команда должна быть помещена в регистр W

            SCS_CAN              ;Устанавливаем линию выбора кристалла
            SPI_OUT              ;Передаем команду
            CCS_CAN              ;Сбрасываем линию выбора кристалла
            return               ;Выходим из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;Переключение режимов MCP2510
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;Set_Normal_Mode, Do_Normal_Mode.
;Set_Config_Mode, Do_Config_Mode.
;Set_Sleep_Mode, Do_Sleep_Mode.
;Set_Loopback_Mode, Do_Loopback_Mode.
;Set_Listen_Mode, Do_Listen_Mode.
;Задача: "Set..." - подпрограмма переключает в режим с помощью
;        команды BitMod.
;        "Do..." - подпрограмма переключает в режим и ожидает выхода
;        из него.
;Входные/выходные параметры отсутсвуют.
;В этой программе будет применен только обычный режим.

Set_Normal_Mode
            SPI_BitMod @CANCTRL, 0x00, 0xE0
            return                           ;Выход из подпрограммы

Do_Normal_Mode
            SPI_BitMod @CANCTRL, 0x00, 0xE0
            call   Read_CANSTAT
            movlw  0xE0                      ;Загружаем маску для
                                             ;  разрядов OPMOD
            andwf  CANSTAT, w
            btfss  STATUS, Z                 ;Уже в обычном режиме?
```

Листинг 2.1. Продолжение

```
            goto   Do_Normal_Mode              ;Нет
            return                             ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Set_Config_Mode
            SPI_BitMod @CANCTRL, 0x80, 0xE0
            return                             ;Выход из подпрограммы

Do_Config_Mode
            SPI_BitMod @CANCTRL, 0x80, 0xE0
            call   Read_CANSTAT
            btfss  CANSTAT, 7                  ;Уже в режиме конфигурации?
            goto   Do_Config_Mode              ;Нет
            return                             ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Set_Sleep_Mode
            SPI_BitMod @CANCTRL, 0x20, 0xE0
            return                             ;Выход из подпрограммы

Do_Sleep_Mode
            SPI_BitMod @CANCTRL, 0x20, 0xE0
            call   Read_CANSTAT
            movlw  0xE0                        ;Загружаем маску для
                                               ; разрядов OPMOD
            andwf  CANSTAT, w
            sublw  0x20
            btfss  STATUS, Z                   ;Уже в "спящем" режиме?
            goto   Do_Sleep_Mode               ;Нет
            return                             ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Set_Loopback_Mode
            SPI_BitMod @CANCTRL, 0x40, 0xE0
            return                             ;Выход из подпрограммы

Do_Loopback_Mode
            SPI_BitMod @CANCTRL, 0x40, 0xE0
            call   Read_CANSTAT
            movlw  0xE0                        ;Загружаем маску для
                                               ; разрядов OPMOD
            andwf  CANSTAT, w
            sublw  0x40
            btfss  STATUS, Z                   ;Уже в режиме кольцевой
                                               ; проверки?
            goto   Do_Loopback_Mode            ;Нет
            return                             ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Set_Listen_Mode
            SPI_BitMod @CANCTRL, 0x60, 0xE0
            return                             ;Выход из подпрограммы

Do_Listen_Mode
            SPI_BitMod @CANCTRL, 0x60, 0xE0
            call   Read_CANSTAT
```

Листинг 2.1. Продолжение

```
            movlw  0xE0                    ;Загружаем маску для
                                           ;  разрядов OPMOD
            andwf  CANSTAT, w
            sublw  0x60
            btfss  STATUS, Z               ;Уже в режиме прослушивания?
            goto   Do_Listen_Mode          ;Нет
            return                         ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;Инициализация CAN-контроллера
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; INI_CAN
; Задача: Инициализация CAN-контроллера
;      Здесь используются таблицы TAB_CAN_INI, TAB_TX0_INI и TAB_TX1_INI.
;      Также инициализируется регистр RX-BufferControl.

INI_CAN
            movlw high TAB_CAN_INI
            movwf PCLATH           ;Инициализируем PCLAT High
            SCS_CAN                ;Устанавливаем линию выбора кристалла
            movlw  0x30            ;Количество регистров для инициализации
            movwf  index
            movlw  CAN_Write       ;Загружаем команду "Запись в регистр"
            SPI_OUT                ;Передаем команду
            movlw  0x00            ;Загружаем адрес первого регистра
            SPI_OUT                ;Передаем адрес
loop_IC_Data movf  index, w        ;Загружаем индекс
            call   TAB_CAN_INI     ;Считываем значение INI для MCP2510
            SPI_OUT                ;Передаем данные
            decfsz index, f
            goto   loop_IC_Data
            CCS_CAN                ;Сбрасываем линию выбора кристалла
            scs_can
            movlw high TAB_TX0_INI
            movwf PCLATH           ;Инициализируем PCLAT High

            movlw  0x0e            ;Количество регистров для инициализации
            movwf  index
            movlw  CAN_Write       ;Загружаем команду "Запись в регистр"
            SPI_OUT                ;Передаем команду
            movlw  0x30            ;Загружаем адрес первого регистра
            SPI_OUT                ;Передаем адрес
loop_TX0_D  movf   index, w        ;Загружаем индекс
            call   TAB_TX0_INI     ;Считываем значение INI для MCP2510
            SPI_OUT                ;Передаем данные
            decfsz index, f
            goto   loop_TX0_D      ;Таблица TAB_TX0_INI готова
            ccs_can
            scs_can
            movlw high TAB_TX1_INI
            movwf PCLATH           ;Инициализируем PCLAT High

            movlw  0x0e            ;Количество регистров для инициализации
            movwf  index
            movlw  CAN_Write       ;Загружаем команду "Запись в регистр"
            SPI_OUT                ;Передаем команду
            movlw  0x40            ;Загружаем адрес первого регистра
```

Листинг 2.1. Продолжение

```
            SPI_OUT                 ;Передаем адрес
loop_TX1_D  movf   index, w         ;Загружаем индекс
            call   TAB_TX1_INI      ;Считываем значение INI для MCP2510
            SPI_OUT                 ;Передаем данные
            decfsz index, f
            goto   loop_TX1_D       ;Таблица TAB_TX1_INI готова
            CCS_CAN                 ;Сбрасываем линию выбора кристалла
INI_Buffer
            SPI_Write 0x60, 0x04        ;Значение INI для RXB0CTRL
            SPI_Write 0x70, 0x00        ;Значение INI для RXB1CTRL
            return                  ;Выходим из подпрограммы

;*****************************************************************
; Таблица векторов и подпрограммы обслуживания прерываний от MCP2510
;*****************************************************************
; ISR_Table
; Задача: Ветвление на основании флагов прерываний в регистре CANINTF,
;         который считывается в READ_CANINTF и обрабатывается
;         в DATA_PROCCESSING. Целиком сбрасывается только после вызова
;         SERVCAN в основном цикле.
;         Также для отдельных флагов вызывается программа обслуживания.
; Вход: CANINTF

ISR_Table
            btfsc  CANINTF, 7
            call   ISR_Error
            btfsc  CANINTF, 6
            call   ISR_WakeUp
            btfsc  CANINTF, 5
            call   ISR_Multi

            btfsc  CANINTF, 0
            call   ISR_RxB0
            btfsc  CANINTF, 1
            call   ISR_RxB1

            btfsc  CANINTF, 2
            call   ISR_TxB0
            btfsc  CANINTF, 3
            call   ISR_TxB1
            btfsc  CANINTF, 4
            call   ISR_TxB2

            return                      ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;Подпрограммы обслуживания прерываний для флагов регистра CANINTF
;*****************************************************************
; ISR_RxB0
;      Буфер приема для выходных битов RxPDO1
; Выход: Загрузка в RB0SIDL и RB0D0, CANINTF.0 :=0 и SPI_Flag,3 :=1

ISR_RxB0
            movlw  0rb0sidl             ; Подстройка
            movwf  spi_address
            call   read_reg
            movf   spi_data,w
```

Листинг 2.1. Продолжение

```
            movwf  rb0sidl
            movlw  @rb0d0                ; Подстройка
            movwf  spi_address
            call   read_reg
            movf   spi_data,w
            movwf  rb0d0                 ; Теперь интересен выходной байт
            SPI_BitMod @CANINTF, 0x00, B'00000001'  ;Сбрасываем флаг
            bsf    SPI_Flag,3            ; Устанавливаем флаг
                                         ;  "Данные не обработаны"
            return

;*****************************************************************************
; ISR_RxB1
;     Буфер приема для запуска и останова узла
; Выход: Загрузка в RB1SIDL и RB1D0, CANINTF.1 :=0 и SPI_Flag,4 :=1

ISR_RxB1
            movlw  @rb1sidl              ; Подстройка
            movwf  spi_address
            call   read_reg
            movf   spi_data,w
            movwf  rb1sidl
            movlw  @rb1d0                ; Подстройка
            movwf  spi_address
            call   read_reg
            movf   spi_data,w
            movwf  rb1d0                 ; Узел запускает или останавливает
                                         ;  только этот байт
            SPI_BitMod @CANINTF, 0x00, B'00000010'  ;Сбрасываем флаг
            bsf    SPI_Flag,4            ; Устанавливаем флаг
                                         ;  "Данные не обработаны"
            return

; ISR_TxB2, ISR_TxB1, ISR_TxB0,
; ISR_Error, ISR_Multi и ISR_WakeUp
;           Срасываются только соответствующие разряды CANINTF

ISR_TxB2
            bcf    SPI_Flag,2            ;Сброс флага
                                         ; "Внутренний буфер полон/пуст"
            SPI_BitMod @CANINTF, 0x00, B'00010000'  ;Сбрасываем флаг
            return
ISR_TxB1
            bcf    SPI_Flag,1            ;Сброс флага
                                         ; "Внутренний буфер полон/пуст"
            SPI_BitMod @CANINTF, 0x00, B'00001000'  ;Сбрасываем флаг
            return                       ;Выходим из подпрограммы
ISR_TxB0
            bcf    SPI_Flag,0            ;Сброс флага
                                         ; "Внутренний буфер полон/пуст"
            SPI_BitMod @CANINTF, 0x00, B'00000100'  ;Сбрасываем флаг
            return                       ;Выходим из подпрограммы
ISR_Error
            SPI_BitMod @CANINTF, 0x00, B'10000000'  ;Сбрасываем флаг
            return                       ;Выходим из подпрограммы
ISR_Multi
            SPI_BitMod @CANINTF, 0x00, B'00100000'  ;Сбрасываем флаг
```

Листинг 2.1. Продолжение

```
            return                          ;Выходим из подпрограммы

ISR_WakeUp
            SPI_BitMod @CANINTF, 0x00, B'01000000'  ;Clear Flag
            return                          ;Return from Subroutine

;*****************************************************************
; Чтение регистра CANSTAT
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Read_CANSTAT
; Задача: Чтение регистра CANSTAT
; Выход: CANSTAT

Read_CANSTAT
            SPI_Move @CANSTAT, CANSTAT ;Чтение регистра CANSTAT
            return                      ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Чтение регистра CANINTF
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Read_CANINTF
; Задача: Чтение регистра CANINTF
; Выход: CANINTF

Read_CANINTF
            SPI_Move @CANINTF, CANINTF ;Чтение регистра CANINTF
            return                      ;Выход из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;Передача сообщения из Source в первый пустой буфер и вызов запроса на
; передачу
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; SPI_Move_Msg
;     Макрос, вызов Test_TxBn
; Test_TxBn
;     Оба не используются

SPI_Move_Msg MACRO  Source
            movlw  Source
            movwf  SPI_Buffer
            call   Test_TxBn            ;Передаем сообщение через первый
                                        ; свободный буфер
            ENDM
Test_TxBn
            NOP
Test_TxB0   btfsc  SPI_Flag, 0          ;Буфер передачи 1 пуст?
            goto   Test_TxB1            ;Нет - проверяем следующий буфер
            movlw  @TXB0                ;Инициализируем функцию
            movwf  SPI_Address
            call   Write_Message        ;Передаем в буфер сообщение
            bsf    SPI_Flag, 0          ;Устанавливаем флаг "Буфер полон"
            SPI_RTS CAN_RTS0            ;Инициализируем запрос на передачу
            return                      ;Выходим из подпрограммы
Test_TxB1
            btfsc  SPI_Flag, 1          ;Буфер передачи 2 пуст?
            goto   Test_TxB2            ;Нет - проверяем следующий буфер
            movlw  @TXB1                ;Инициализируем функцию
```

Листинг 2.1. Продолжение

```
            movwf  SPI_Address
            call   Write_Message        ;Передаем в буфер сообщение
            bsf    SPI_Flag, 1          ;Устанавливаем флаг "Буфер полон"
            SPI_RTS CAN_RTS1            ;Инициализируем запрос на передачу
            return                      ;Выходим из подпрограммы
Test_TxB2
            btfsc  SPI_Flag, 2          ;Буфер передачи 3 пуст?
            goto   Test_TxBn_Error      ;Нет - ошибка "Все буферы полны"
            movlw  @TXB2                ;Инициализируем функцию
            movwf  SPI_Address
            call   Write_Message        ;Передаем в буфер сообщение
            bsf    SPI_Flag, 2          ;Устанавливаем флаг "Буфер полон"
            SPI_RTS CAN_RTS2            ;Инициализируем запрос на передачу
            return                      ;Выходим из подпрограммы
Test_TxBn_Error
;Подпрограмма для случая, когда заполнены все буферы
            nop
            return                      ;Выходим из подпрограммы

;*****************************************************************
; Инициализация модуля SSP (режим SPI)
; BSSP - SPI
;*****************************************************************
; INI_SPI
; Задача: Подпрограмма инициализации для модуля SSP
;         Устанавливаются значения TRIS и SSPCON.
;         Прерывание от SSP запрещено и разряд SSPIF сброшен
; Вход: -
; Выход: -

INI_SPI     BANK1
            bsf    TRISC, 4             ;SDI - вход
            bcf    TRISC, 5             ;SDO - выход
            bcf    TRISC, 3             ;SPI-ведущий, выход SCK
            movlw  B'01000000'          ;0 - SMP - опрос по середине
                                        ;1 - CKE - передача по
                                        ;  нарастающему фронту
                                        ;000000 - не используются
            movwf  SSPSTAT
            bcf    PIE1, SSPIE          ;Запрещаем прерывания от SPI
            BANK0
            movlw  B'00100000'          ;0 - флаг "Clear Collision"
                                        ;0 - нет переполнения
                                        ;1 - разрешение выводов
                                        ;0 - CKP - полярность в режиме
                                        ;  простоя низкая
                                        ;0000 - тактирование FOSC/4
            movwf  SSPCON               ;Конфигурирование BSSP
            bcf    PIR1, SSPIF          ;Сбрасываем флаг SPI
            return                      ;Выходим из подпрограммы

;*****************************************************************
;Подпрограммы для обработки внутренних буферов данных
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Data_Proccessing
; Задача: Обработка принимаемых сообщений
```

Листинг 2.1. Продолжение

```
; Вход: SPI_Flag,3 - относится к буферу приема 0: (выходные биты)
;       SPI_Flag,4 - относится к буферу приема 1: запуск и останов узла

Data_Proccessing
            btfsc  SPI_Flag, 3      ;Опрос флага "Данные не обработаны"
            call   ISR_DP1
            btfsc  SPI_Flag, 4      ;Опрос флага "Данные не обработаны"
            call   ISR_DP2
            return                  ;Выходим из подпрограммы

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Здесь поступают команды шины CAN!
; ISR_DP1
; Обработка выходных бит RxPDO1;

ISR_DP1
dp1_0   ;   bcf    abit1               ; Если бит 0 = 0
            btfss  rb0d0,0
            goto   dp1_1
        ;   bsf    abit1
        ;   movlw  abitdau             ; Константа для
        ;   movwf  abittim             ;  песочных часов
;-------------------------------------------------------------------------
dp1_1       btfsc  rb0d0,1             ; Если в буфере приема 0 разряд 1
                                       ;  _байта данных 0 = 1,
        ;   bcf    abit2               ; то сбрасываем разряды
            nop
;
;-------------------------------------------------------------------------
dp1_2   ;   bcf    abit2
            btfss  rb0d0,2
            goto   dp1_3
        ;   bsf    abit2
;-------------------------------------------------------------------------
dp1_3       btfss  rb0d0,3
            goto   dp1_4
        ;   bsf    abit4

dp1_4       bsf    txpdo               ; Требование на передачу
                                       ;  кадра из 8 байт
dp1_41      bcf    SPI_Flag, 3         ; Сброс флага "Данные обработаны"
            return

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;ISR_DP2
; Обработка команд запуска и останова узла
; Соответственно изменяется 'состояние' в буфере тактирования.

ISR_DP2     movf   rb1d0,w             ; Запускаем и останавливаем узел
            xorlw  01
            skpz
            goto   knstop
knstart     SPI_WRITE 046H,05H         ; TXB1D0 = состояние тактирования
            bcf    txpdo
            bsf    knot_go
            goto   idp2end
knstop      SPI_WRITE 046H,04H         ; TXB1D0 = состояние тактирования
```

Листинг 2.1. Окончание

```
            bcf     knot_go

ep2end      bcf     SPI_Flag, 4         ; Сброс флага "Данные обработаны"
            return

;*****************************************************************
```

В файле can.inc интерес представляют подпрограммы ISR_RxB0 и ISR_RxB1, которые вызываются в ISR_TABLE. В этих подпрограммах выполняется выборка только что пришедших данных из буфера приема и перегрузка в собственные переменные.

В завершение следуют подпрограммы ISR_DP1 и ISR_DP2, которые вызываются в DATA_PROCESSING. Здесь обрабатываются принятые команды или данные.

BUCHCAN.ASM

Второй файл — главный файл .asm, содержащий определения для микроконтроллера, определения переменных и констант и главную программу (листинг 2.2).

Листинг 2.2. Файл buchcan.asm

```
      TITLE  "Buchprojekt: CAN-Knoten"
;*****************************************************************
;     Проект:       BuchCAN
;     Версия:       1
;
;     Микроконтроллер:    16 F 877
;     Компилятор:   MPLAB 6.60
;*****************************************************************
;     История обновлений:
;     N.      Автор         Дата
;     01      Kunig         10. 2004
;*****************************************************************

      INCLUDE "P16F877.INC"
      LIST   P=16F877,F=INHX8M
      __CONFIG  _CP_ALL & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF &
         _        LVP_OFF & _BODEN_ON & _PWRTE_ON & _WDT_OFF & _XT_OSC

      INCLUDE       "buchiop.iop"       ; Определение выводов
      INCLUDE       "buchmac.inc"       ; Макроопределения

;*****************************************************************
; Определения переменных

INDEX         EQU     0X22    ; Индекс для доступа к таблице
SPI_ADDRESS   EQU     0X23    ; Регистр адреса дял работы SPI
SPI_DATA      EQU     0X24    ; Регистр данных для работы SPI
SPI_MASK      EQU     0X25    ; Регистр маски для работы SPI (Bit Modify)
SPI_BUFFER    EQU     0X26    ; Регистр для SPI (адрес буфера сообщения)
CANINTF       EQU     0X27    ; Регистр флага прерывания
CANSTAT       EQU     0X28    ; Регистр состояния
SPI_FLAG      EQU     0X29    ; Регистр флагов
                              ; SPI-флаг_"TXB0_пуст" - разряд 0
                              ; SPI-флаг "TXB1_пуст" - разряд 1
                              ; SPI-флаг "TXB2_пуст" - разряд 2
                              ; SPI-флаг "RXB0_данные не обработаны"-разряд 3
```

Листинг 2.2. Продолжение

```
                                ; SPI-флаг "RXB1_данные не обработаны"-разряд 4

; Определения буферов
#DEFINE @RBUFFERS   0X43    ; Начальный адрес буфера сообщений CAN в RAM

       CBLOCK @RBUFFERS
       RB0SIDL              ; SIDL в RxPDO
       RB0D0                ; D0 в RxPDO
       RB1SIDL              ; SIDL в RxPD1
       RB1D0                ; D0 в RxPD1
       ENDC

;--------------------------Переменные
       CBLOCK 50H
       STEUER
       JMPS                 ; Общая информация о перемычках
       endc
;--------------------------Переменные времени
       cblock 60h
       EVENT, STEPEVE, STEP, vsekunde
       TSTAT
       hrdcount
       ENDC
;-------------------------- Разряды в steuer
;
#define       canda  steuer,7
;-------------------------- Разряды в tstat
#define       MSEROVER tstat,0
#define       knot_go      tstat,1
#define       free   tstat,2
#define       txpdo  tstat,3
#DEFINE       hrdBIT tstat,4
;-------------------------- Разряды в jmps
; Перемычки MCP2510.
#DEFINE       BAUD1  JMPS,0
#DEFINE       BAUD2  JMPS,1
;#define      frei   jmps,2
; Перемычки порта D микроконтроллера PIC
;#define      frei   jmps,5
;#define      frei   jmps,6
;#define      frei   jmps,7
;*****************************************************************
;
BANK0  MACRO                ; Макрос переключения банка 0
       BCF    STATUS, RP0
       BCF    STATUS, RP1
       ENDM
BANK1  MACRO                ; Макрос переключения банка 1
       BSF    STATUS, RP0
       BCF    STATUS, RP1
       ENDM

;*****************************************************************

       ORG    0
       CLRF   PCLATH
       GOTO   MAIN
```

Листинг 2.2. Продолжение

```
        NOP
        NOP
        RETFIE
        NOP

        INCLUDE     "can.inc"     ; Подпрограммы CAN от Клауса Кошински
                                  Некоторые из подпрограмм были нами модифицированы
  Подпрограммы
Delay       movlw  .10            ; Задержка в 35 мкс
            movwf  step
Delo        decfsz step
            goto   delo
            return

;*****************************************************************
        rbint
        Задача для случая, когда требуется линия прерываний
;*****************************************************************
SERVCAN     CALL   READ_CANINTF
            CALL   ISR_TABLE
            CALL   DATA_PROCCESSING
            RETURN

;*****************************************************************************
 watchtim :
 Общая организация отсчета времени:
 Коэффициент деления для TMR0 составляет 8

Воспользуемся следующими единицами времени:
        1 мс
        250 мс
        10 с
;*****************************************************************************
watchtim    bcf    msekover
            MOVF   EVENT,W
            SUBWF  TMR0,W         ; Делитель = 8
            ANDLW  0C0H           ; Временное окно 64*8 мкс = 512 команд
            SKPZ
            RETURN
;-----------------------------------  Каждую 1 мс:
            MOVLW  .125           ; Загружам EVENT для следующего такта
            ADDWF  EVENT          ;
            BSF    MSEKover

        Здесь вставляем действия на 1 мс

            DECFSZ STEP
            RETURN
;-----------------------------------  Каждую 1/4 с
            MOVLW  .250
            MOVWF  STEP
            bsf    free           ; Разрешение действий после 1/4 с
                                  ;  однократно после сброса
        DECF   hrdCOUNT           ; Время для подсчета тактов
        skpnz
        BSF    hrdBIT             ;
```

Листинг 2.2. Продолжение

```
;       Здесь вставляем действия на 1/4 с
;
            decfsz   vsekunde
            RETURN
;------------------------------------ Каждые 10 с -------
            movlw    .40        ; Перезагружаем vsekunde
            movwf    vsekunde
;
;       Здесь вставляем действия на 10 с
;
            return
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;Readjp / rdcanjp: чтение перемычек
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
READJP      MOVLW  1FH
            ANDWF  JMPS
            MOVF   PORTD,W
            ANDLW  0E0H
            IORWF  JMPS
            RETURN

RDCANJP     MOVLW  0E0H
            ANDWF  JMPS
            MOVLW  @TXRTSCTRL
            MOVWF  SPI_ADDRESS
            CALL   READ_REG
            RRF    SPI_DATA
            RRF    SPI_DATA
            RRF    SPI_DATA
            MOVF   SPI_DATA,W
            ANDLW  07
            IORWF  JMPS
            RETURN

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Проверяем, в наличии ли интерфейс CAN (если да, то canda=1)
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CHKCAN      NOP
            SPI_WRITE 033H,2AH
            NOP
            SPI_READ 033H
            MOVLW  2AH
            XORWF  SPI_DATA      ; zr = CAN
            bcf    canda
            skpnz
            bsf    canda
            RETURN
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Такты
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
hrdBEAT     SPI_RTS      CAN_RTS1     ;Инициализируем запрос на передачу
            BCF     hrdBIT
                     ; Устанавливаем счетчик для последующего такта
            RETURN
```

Листинг 2.2. Продолжение

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; servtx: Подготовка и отправка кадра из 8 байт данных
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
serv_tx     btfss  knot_go
            return
            movlw  @txb0D0
            movwf  spi_address
                                ; 1-й байт данных - в W
            movwf  spi_data
            call   write_reg
                                ; 2-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg
                                ; 3-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg
                                ; 4-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg
                                ; 5-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg
                                ; 6-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg
                                ; 7-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg
                                ; 8-й байт данных - в W
            movwf  spi_data
            incf   spi_address
            call   write_reg

            bcf    txpdo        ; Снимаем требование на передачу

            bsf    spi_flag,0
            SPI_RTS CAN_RTS0    ; Инициализируем 'Запрос на передачу'
            RETURN

;++++++++++++++++++++++++++++++++++++++++++++++++++Главная программа
MAIN
            MOVLW  RES_A        ; Установка значений сброса для выводов
            MOVWF  PORTA
            MOVLW  RES_B
            MOVWF  PORTB
            MOVLW  RES_C
            MOVWF  PORTC
            MOVLW  RES_D
            MOVWF  PORTD
            MOVLW  RES_E
            MOVWF  PORTE
```

Листинг 2.2. Продолжение

```
;           MOVLW  ICC_CH          ; Инициализируем АЦП
;           MOVWF  ADCON0
            BANK1                  ; Переключаемся на банк 1
            MOVLW  80H             ; 8/0 без Ref; A0,A1,A2,A5,E0,E1,E2;
                                   ;  A3 - свободен
            MOVWF  ADCON1          ; Конфигурируем АЦП
            MOVLW  TR_A            ; Устанавливаем направление передачи
                                   ;  данных через выводы портов
            MOVWF  TRISA
            MOVLW  TR_B
            MOVWF  TRISB
            MOVLW  TR_C
            MOVWF  TRISC
            MOVLW  TR_D
            MOVWF  TRISD
            MOVLW  TR_E
            MOVWF  TRISE
;
            BANK0
;
            MOVLW  20H             ;Обнуляем Ram
            MOVWF  FSR
CLO         CLRF   INDF
            INCF   FSR
            BTFSS  FSR,7
            GOTO   CLO
            MOVLW  0A0H
            MOVWF  FSR
CLO1        CLRF   INDF
            INCF   FSR
            BTFSC  FSR,7
            GOTO   CLO1
            CALL   READJP          ; Считываем состояние перемычки
            HARD_RESET             ; Аппаратный сброс MCP2510
            CALL   INI_SPI         ; Инициализируем модуль SPI
            CALL   CHKCAN          ; Есть контроллер CAN?
            CALL   RDCANJP         ; Считываем входы (перемычка) MCP2510
            CALL   INI_CAN         ; Инициализируем MCP2510 через SPI
            CALL   SET_NORMAL_MODE
            clrf   SPI_Flag        ; Инициализируем флаг SPI
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            BANK1
            MOVLW  82H             ; Конфигурируем Timer0
            MOVWF  OPTION_REG
            BANK0
            MOVLW  07              ; Конфигурируем Timer1
            MOVWF  T1CON
            CLRF   TMR1L           ; Обнуляем содержимое Timer1
            CLRF   TMR1H
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Инициализация переменных
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      MOVLW  .125                  ; Инициализация переменных времени
      MOVWF  EVENT
      CLRF   TMR0
      movlw  .40
      movwf  vsekunde
```

Листинг 2.2. Окончание

```
        movlw  .250
        movwf  step
        MOVLW   .40                    ; Такт
        movwf  hrdCOUNT

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Главный цикл
; WATCHTIM должна вызываться на протяжении минимум 512 мкс!

loop    call   watchtim          ; Наблюдение за всем процессом

        Учет аналоговых и цифровых входов
        и оценка результата

        btfss  canda             ; Если интерфейса CAN не существует,
        goto   loop1             ;  переходим к loop1

        btfss  spi_int           ; Разряд 0 порта B - поступила команда CAN!
        call   servcan           ; Обрабатываем принятую команду CAN
        call   watchtim          ; Для верности
        btfsc  txpdo             ; Команда CAN может установить этот разряд
        call   serv_tx           ; Отправляем 8-мибайтный кадр данных
        btfsc  hrdbit            ; hrdbit устанавливается в watchtim
        call   hrdbeat           ; Передаем тактовый сигнал, если заданный
                                 ;  период времени истек
loop1

        goto   loop
        FILL   (Goto 0),1FFEH-$
        END
```

В части программы MAIN вначале производится инициализация, а затем выполнение программы переходит прямо в главный цикл, называемый LOOP. При более близком рассмотрении здесь оказывается завершенная подпрограмму распределения контроля времени, задающая импульсы таймера для различных интервалов времени. Данные процесса должны считываться в начале цикла LOOP или же в определенные моменты времени.

Если распознается передача устройством CAN сообщения, то устанавливается флаг txpdo. В главном цикле этот флаг опрашивается в заданное время и потом вызывается подпрограмма serv_tx, в которой кадр должен быть сформирован и перенесен в соответствующий буфер, после чего выставляется запрос контроллеру CAN на разрешение отправки.

Если это устройство должно обращать на себя внимание через равномерные промежутки времени, то рационально использовать тактовый сигнал. Но для этого он должен быть разрешен. Содержимое телеграммы загружается в ISR_DP2.

Ключевое значение имеет подпрограмма WATCHTIM — модуль времени. В этой подпрограмме организован отсчет в режиме реального времени и решаются задачи других модулей, связанных со временем — прежде всего, "песочные часы". Предположим, некоторый модуль хочет получить уведомление, например, через 50 мс. Тогда этим модулем устанавливается необходимое значение счетчика, а WATCHTIM декрементирует это значение до тех пор, пока оно не станет равным нулю, после чего устанавливается определенный флаг.

BUCHIOP.IOP

В этом файле (листинг 2.3) определены выводы микроконтроллера как входы или выходы.

Листинг 2.3. Файл buchiop.iop

```
;=========================================================
;       Определения выводов микроконтроллера PIC
;=========================================================
;                       TRIS   Значение сброса
;========================= Порт A
;ana1           EQU     0       ;1      x       ; Аналоговый вход
;ana2           EQU     1       ;1      x       ; Аналоговый вход
;ana3           EQU     2       ;1      x       ; Аналоговый вход
;                       3       ;1      x       ; Требуется внешнее оп. напряжение
;                       4       ;1      x       ; TOCKI свободен
;ana4           EQU     5       ;1      x       ; Аналоговый вход

;========================= Порт E
;ana5           EQU     0       ;1      x       ; Аналоговый вход
;ana6           EQU     1       ;1      x       ; Аналоговый вход
;ana7           EQU     2       ;1      x       ; Аналоговый вход

ana1_CH         equ     041h                    ; an0
ana2_CH         equ     049h                    ; an1
ana3_CH         equ     051h                    ; an2
ana4_CH         equ     061h                    ; an4
ana5_CH         equ     069h                    ; an5
ana6_CH         equ     071h                    ; an6
ana7_ch         equ     079h                    ; an7

TR_A            EQU     3fH
RES_A           EQU     10h
TR_E            EQU     07h
RES_E           EQU     0

;======================  Порт B
;SPI_INT        EQU     0       ;1      x       ; Линия прерываний CAN-контроллера
;digi1          EQU     1       ;1      x       ; Цифровой вход
;digi2          EQU     2       ;1      x       ; Цифровой вход
;digi3          EQU     3       ;1      x       ; Цифровой вход
;digi4          EQU     4       ;1      x       ; Цифровой вход
;               EQU     5       ;0      0       ; свободен
;               EQU     6       ;0      0       ; Внутрисхемное программирование
;               EQU     7       ;0      0       ; Внутрисхемное программирование

#define         spi_int         PORTB,0
#define         digi1           PORTB,1
#define         digi2           portb,2
#define         digi3           PORTB,3
#define         digi4           PORTB,4

TR_B            EQU     1fh
RES_B           EQU     0

;======================   Порт C
;               EQU     0       ;0      0       ; Свободен
;CRST           EQU     1       ;0      0       ; Сброс CAN
;CCS            EQU     2       ;0      1       ; Выбор кристалла CAN
```

Листинг 2.3. Окончание

```
SCK         EQU   3       ;0      0       ; SPI-такт
SDI         EQU   4       ;1      0       ; SPI_DI
SDO         EQU   5       ;0      0       ; SPI_DO
            EQU   6       ;0      0       ; Свободен
            EQU   7       ;1      0       ; Свободен

#define     CRST  PORTc,1                 ; Сброс CAN
#define     CCS   PORTc,2                 ; Выбор кристалла CAN

TR_C        EQU   091H
RES_C       EQU   04H

;--------------------  Порт D
digo1       EQU   0       ;0      0       ; Цифровой выход
digo2       EQU   1       ;0      0       ; Цифровой выход
digo3       EQU   2       ;0      0       ; Цифровой выход
digo4       EQU   3       ;0      0       ; Цифровой выход
digo5       EQU   4       ;0      0       ; Цифровой выход
jp3         EQU   5       ;1      x       ; Вход перемычки
jp2         EQU   6       ;1      x       ; Вход перемычки
jp1         EQU   7       ;1      x       ; Вход перемычки

#define     digo1 portd,0
#define     digo2 PORTd,1
#define     digo3 PORTd,2
#define     digo4 PORTd,3
#define     digo5 PORTd,4
#define     jp3   PORTd,5
#define     jp2   PORTd,6
#define     jp1   PORTd,7

TR_D        EQU   0e0H
RES_D       EQU   0H
```

2.5. Шина LIN

LIN была задумана как "дешевая" шина, которая может передавать информацию на короткие дистанции. Нередко можно прочитать, что она пригодна лишь для очень простых, медленных применений, таких как, например, уведомление об изменении состояния переключателей. Однако, на наш взгляд, ее возможности значительно шире. Как-никак, она может быть задействована на расстояниях до 40 м и скоростях до 20 кБод.

Речь идет об однопроводной шине, которая может устанавливаться в состояние высокого уровня с помощью распределенных подтягивающих резисторов. Каждый участник имеет выход исключительно с открытым коллектором, так что каждый узел может изменять уровень только замыканием шины на "землю" (состояние низкого уровня — доминантное). Когда ни один из узлов не активирует свой выход, состояние шины соответствует высокому уровню (рецессивное).

низкий уровень = доминантный;

высокий уровень = рецессивный = холостой ход.

Шина LIN управляется одним ведущим устройством. Ведущий — единственный, кто может начать связь. Ведомый может передавать данные только тогда, когда это затребовано ведущим.

Когда ведомый выставляет в шину сообщение, все другие участники воспринимают его одновременно, так что возможны прямые связи между участниками.

Для того чтобы участники своевременно подготовились к обмену данными, ведущий отправляет перед байтом синхронизации "разрыв" протяженностью в 13 бит.

Поскольку отсутствует линия синхронизации, связь базируется на жесткой привязке к внутреннему тактированию. Во избежание использования дорогостоящего кварца, ведущий посылает в начале связи байт синхронизации, позволяющий ведомому подстроить свою ось времени. Разумеется, скорость передачи должна быть согласована (+/–15%).

Байту синхронизации, как и любому байту данных, предшествует при отправке стартовый бит (0). В конце следует стоп-бит (1). Байт синхронизации имеет значение $0AA_h$ (10101010_b). За байтом синхронизации следует идентификационное поле, состоящее (на данный момент) из одного байта. Последний содержит четыре разряда для адресации, два — для установки длины сообщения (2, 4 или 8) и два — для контроля четности.

2.5.1. Принцип действия LIN

Аппаратных модулей микроконтроллеров PIC для связи по шине LIN не существует. Управление передачей данных по LIN возлагается на ведущего, однако он не обеспокоен согласованием времени — критерием согласования ведомых является его скорость передачи.

По этой причине ведущий может применять в качестве модуля LIN обычный модуль UART. Проблемы возникают у ведомых, которые должны подстраивать свой отсчет времени под скорость передачи ведущего.

Если пользователь не желает возиться с побитной отправкой и приемом, тогда можно воспользоваться модулем EUSART, дополнительные свойства согласования которого предназначены специально для этой цели. Само собой разумеется, микроконтроллер PIC со встроенным модулем EUSART никак не относится к представителям низшей ценовой категории.

Если модуль EUSART также неприемлем, то простейшим вариантом становится организация обмена данными по LIN на основании обычного микропрограммного обеспечения. В случае использования прерываний существует один программно-технический недостаток: при каждом входе в подпрограмму обработки прерывания необходимо вспоминать, в каком месте протокола была совершена остановка, и заново выполнять соответствующее ветвление. Без прерываний протокол можно прорабатывать шаг за шагом.

Если применяют прерывания, тогда удобнее всего использовать модуль CCP. Для регистрации байта синхронизации применяют режим захвата, а для опроса — режим сравнения.

Использование прерываний INT и TMR0 очень трудоемкое и требует кропотливого труда. Кроме того, эти прерывания могут блокировать друг друга. Без крайней на то необходимости, мы бы такого решения не рекомендовали.

2.5.2. LIN на основе микропрограммного обеспечения

Недостаток микропрограммного управления заключается в том, что на время всего времени передачи микроконтроллеру приходится в значительной мере концентрироваться на задаче LIN. Побочные операции в течение этого периода очень ограничены и могут столкнуться с некоторыми затруднениями. Таким образом, передача по LIN длительностью в несколько миллисекунд, в зависимости от скорости

передачи и длины данных, в той или иной мере приводит к блокировке. Если конкретный случай применения это допускает, тогда обмен данными по LIN можно без проблем реализовать даже с помощью базовой серии микроконтроллеров PIC.

Между двумя протоколами LIN микропрограмма должна через достаточно короткие промежутки времени отслеживать возможное появление сигнала на входе Sync.

При скорости передачи 10 кБод на каждый бит приходится по 100 мкс. Разрыв синхронизации имеет длительность 1,3 мс, но это не означает, что проверка должна выполняться только каждые 1,3 мс, поскольку необходимо также проверять длину самого разрыва (хотя бы на достоверность). При этом точностью слишком увлекаться не следует, тем более, что протокол LIN поддерживает еще и другие проверки на достоверность.

Важно точно обнаружить байт синхронизации (10101010). Для того чтобы определить при передаче длительность бита, измеряют суммарное время, затраченное на прохождение восьми задних фронтов текущего байта, и делят его на 8. Рационально также выполнить дополнительную проверку на четность отдельных периодов между двумя фронтами.

Обнаружение байта синхронизации

Предположим, вход LIN называется LININPUT, а для учета времени мы используем программный суммирующий таймер TIMH:TIML. Счет реализован в макросе TIMUP. Кроме того, рационально предусмотреть выход по истечении тайм-аута.

Программа для контроля над общим временем могла бы выглядеть следующим образом:

```
SYNCH    CLRF   TIML        ; Программный таймер
         CLRF   TIMH
         MOVLW  4
         MOVWF  COUNT
         CLRF   TMR0
WAILO    BTFSC  IMPUT       ; Ожидаем низкий уровень
         TIMUP               ; См. ниже
         GOTO   WAILO
WAIHI    BTFSS  LINPUT      ; Ожидаем высокий уровень
         GOTO   WAIHI
WAINEXT  TIMUP              ; См. ниже
         DECFSZ COUNT
         GOTO   WAILO
```

В части программы SYNCH бросается в глаза то, что макрос TIMUP вызывается только внутри цикла WAILO, но не внутри цикла WAIHI. Это означает, что теоретически можно навсегда остаться в цикле WAIHI, но это невероятно, поскольку в состоянии холостого хода присутствует высокий уровень.

Если вызов макроса TIMUP поместить также и внутри цикла WAIHI, то точность, имеющая отношение к последнему положительному фронту, не была бы больше достаточной.

Видите, сколько деталей следует держать в голове при программировании с высокой временной точностью?!

Ради полноты, представляем нашу версию макроса TIMUP. После входа в этот макрос TIML представляет предыдущее значение TMR0, поэтому TMR0 всегда больше TIML, если только не произошло переполнение.

```
TIMUP   MACRO
        MOVF    TIML,W
        SUBWF   TMR0,W          ; W = TMR0-(5+TIML)
        SKPC
        INCF    TIMH            ; Если TMR0 < TIML
        ADDWF   TIML            ; TIML = прежнее значение TMR0
        BTFSC   TIMH,1          ; TIMH:TIML < 512!!
        GOTO    TIMEOUT         ; Обработка ошибки
        ENDM
```

Обратите внимание на то, что при хронометраже речь идет о моменте выполнения команды SUBWF. Момент появления последнего фронта — раньше на расстоянии от трех до шести командных циклов. Для того чтобы определить этот момент как можно точнее, вычитаем из TIMH:TIML еще четыре цикла таймера (если коэффициент деления частоты равен 1).

Выбор такого коэффициента деления TMR0, с точки зрения гарантии отсутствия переполнения во время прихода байта синхронизации, — не самый лучший. Во-первых, мы не обнаружим ошибочное переполнение, и, во-вторых, следует придерживаться высокой точности по отношению к длительности бита (неточности накапливаются!).

В точке, где вызывается TIMUP, достаточно времени для того, чтобы в целях диагностики сохранять промежуточные значения времени, которые можно просматривать позже в эмуляторе.

Переменную TIMH:TIML мы делим на восемь (при этом не забывайте об округлении!) и получаем длительность бита, которую называем BIT_ZEIT.

Когда длительность бита больше 255, то на практике ее делят на два, для чего увеличивают в два раза значение коэффициента деления TMR0. **Таким образом, в дальнейшем будем принимать, что длительность бита < 256.**

В следующем примере мы допускаем случай (неблагоприятный!), где скорость передачи составляет 20 кБод, а частота системной синхронизации — 4 МГц. Это означает, что длительность бита находится в пределах 50 (циклов таймера).

Опрос

После последнего нарастающего фронта поля синхронизации ожидается ниспадающий фронт стартового бита, после чего начинается отсчет времени для опроса последующих битов данных.

Рассмотрим пример, в котором, с целью упрощения, примем, что опрос выполняется только один раз на один бит (посередине бита). Зачастую, для повышения надежности опрос выполняют трижды. Позже мы покажем, что трехкратный опрос может только добавлять проблем.

У нас достаточно времени на то, чтобы спокойно вычислить момент первого опроса. Эта точка составляет 1,5 длительности бита после ниспадающего фронта. Последующие семь моментов опроса отстоят друг от друга через отрезки времени, равные длительности бита. Для установления моментов опроса можно воспользоваться таймером TMR0. Моменты опроса мы называем LIN_EVENT. Таймер Timer0 устанавливается сразу же после обнаружения стартового бита = 0.

Для первого момента опроса устанавливаем LIN_EVENT = 3/2 · BIT_ZEIT. Хотя мы заранее оговорили, что BIT_ZEIT < 256, это, безусловно, — необязательное условие для 3/2 · BIT_ZEIT. Таким образом, если потребуется, время ожидания первого момента опроса следует разделить на две части.

Для опроса восьми бит мы устанавливаем переменную-счетчик COUNT = 8. Переменную для хранения принятого байта назовем LINWERT:

```
LIN8    MOVF    LIN_EVENT,W     ; Ожидаем момент опроса
        SUBWF   TMR0,W          ; W := TMR0 - событие
        ANDLW   0F8H            ; Маскируем 3 разряда
        SKPZ                    ; Если 0, выходим из цикла
        GOTO    LIN8
        ABTASTEN                ; Макрос для опроса
        MOVF    BIT_ZEIT,W
        ADDWF   LIN_EVENT
        DECFSZ  COUNT
        GOTO    LIN8
```

Ради полноты, представим здесь также макрос ABTASTEN:

```
ABTASTEN  MACRO
          CLRC
          BTFSC  LINPUT
          SETC
          RRF    LINWERT
          ENDM
```

Цикл по опросу времени с использованием TMR0 длится шесть командных циклов, так что в самом неблагоприятном случае момент опроса запаздывает на шесть командных циклов. При длительности бита в 50 командных циклов — это довольно большая неточность.

В качестве альтернативы, воспользуемся помощью подпрограммы WARTE:

```
WARTE   MOVLW  4
WLO     SUBWF  WZEIT
        SKPNC
        GOTO   WLO
        COMF   WZEIT,W
        ADDWF  PCL          ; Команда перехода: 2 Цикла!!
        NOP
        NOP
        NOP
        RETURN
```

Вызов подпрограммы WARTE длится ровно WZEIT+11 командных циклов, включая CALL и RETURN (кто не верит, пусть проверит).

С этой подпрограммой цикл LIN8 выглядит следующим образом:

```
        MOVF    ZEIT1,W         ;3/2 BIT_ZEIT-11-X (см. ниже)
        MOVWF   WZEIT
LIN8    CALL    WARTE
        ABTASTEN                ; 4 командных цикла
        MOVLW   .21             ; 11 + остаток длины цикла
        SUBWF   BIT_ZEIT,W
        MOVWF   WZEIT
        DECFZ   COUNT
        GOTO    LIN8
```

При вычислении ZEIT1 необходимо также подсчитать количество команд (X), задействованных от появления отрицательного фронта стартового бита до метки LIN8.

Несимметричность

При опросе (начиная от отрицательного фронта стартового бита) мы исходим из того, что все биты имеют длительность BIT_ZEIT. Однако, по физическим причинам, такого никогда не бывает. Положительные и отрицательные фронты имеют разную крутизну.

В следующем примере мы представляем это несколько преувеличенно: теоретическая длительность бита — 50, фактическая (на входе LIN) длительность отдельного бита при высоком уровне — лишь 40, что для соседнего бита с низким уровнем дает 60.

Поскольку опрос всегда происходит посередине теоретического LIN-бита, начиная от ниспадающего фронта стартового бита, возникает ситуация, представленная на рис. 2.4.

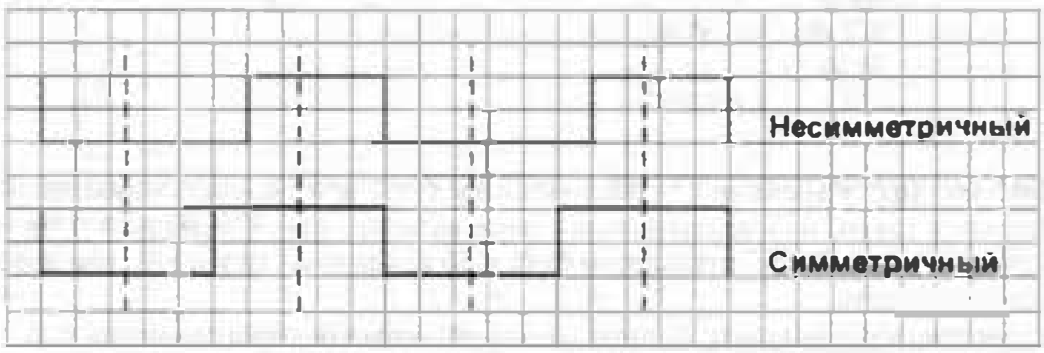


Рис. 2.4. Несимметричность сигналов LIN

Бит высокого уровня мы опрашиваем уже через 15 циклов таймера после положительного фронта. Само по себе, это — еще не проблема, особенно учитывая тот факт, что несимметричность, в действительности, вряд ли будет заметна. Однако, вместе с другими неточностями, это может приводить к ошибкам — прежде всего, при опросе в трех точках: один раз до средины и один раз после нее.

Точность!

Допустим, при с учетом байта синхронизации общее время составляет 396 циклов таймера. Деление на 8 дает 49,5. Округляем это значение до большего и получаем длительность бита 50. Погрешность составляет половину цикла таймера. Поскольку эта погрешность проявляется для каждого бита, то погрешность времени опроса на 9-м бите составляет уже 4,5 циклов таймера.

Ошибка округления длительности бита может или уменьшать погрешность, связанную с несимметричностью, или аддитивно с нею объединяться.

Если при опросе возникает погрешность длительности бита, тогда может произойти так, что при малом разрешении длительности бита (50 циклов таймера!) момент опроса может уже оказаться опасно близко к краю бита (или даже выйти за его пределы). Проблем становится еще больше, если опрос выполняют в трех точках с интервалом около 7 циклов таймера.

Разрешение в 50 циклов таймера проблематично! При увеличении частоты системной синхронизации можно также обдумать различные решения на микропро-

граммном уровне. Например, можно регистрировать и учитывать несимметричность ее или же учитывать дробную часть длительности бита.

Диагностика

Если присутствует погрешность обмена данными по шине LIN, и тесты на эмуляторе не внесли ясности, тогда на помощью приходит осциллограф. Впрочем, представленные ниже тесты интересны даже и в том случае, если никаких проблем не возникает.

Тест 1

Пишут маленький тестовый цикл, в котором считывается исключительно вход LIN и после того результат выдается на диагностический вывод. На осциллографе наблюдают (аналоговый) исходный сигнал и его эхо на диагностическом выводе. Таким образом обнаруживают несимметричность.

Тест 2

В готовой программе для LIN определяют несколько выходов для выдачи результатов. Для опытных образцов проекта часто используют микроконтроллер PIC с избыточностью выводов для диагностических целей.

Диагностический выход, изменяющий свой уровень на противоположный при каждом моменте опроса, дает хорошую картину, выходим ли мы при этом опросе за пределы бита или нет. Дополнительное эхо считанного значения опроса на диагностическом выходе показывает, в порядке ли сам опрос.

Внимание!

При вычислении времени не забывайте учитывать вставленные диагностические команды.

2.6. USB

Между тем, для подключения к ПК законченных устройств интерфейс RS232 почти полностью вытеснен шиной USB. По этой причине есть все причины предполагать, что в ряде микроконтроллеров PIC для этого интерфейса существует соответствующий аппаратный модуль. Таким модулем обладают некоторые представители семейств PIC16 и PIC18. Дополнительное аппаратное обеспечение не очень дорогостоящее. Разумеется, по такому интерфейсу невозможно передавать данные так же просто, как в случае с RS232 — необходима надлежащая регистрация на ПК. Дополнительные издержки потребуются также и со стороны ПК.

Из-за сложной организации управления, работать с шиной USB разработчику гораздо сложнее, чем с шиной CAN, однако, благодаря множеству статей и примеров от компании Microchip, эта задача, в том, что касается микроконтроллера PIC, вполне разрешима. Если же говорить о программном USB-обеспечении для ПК, то, по-видимому, эта сторона доступна только узкому кругу "посвященных".

2.6.1. Помощь начинающим от Microchip

Компания Microchip для освоения шины USB предлагает не только аппаратное, но также программное обеспечение и литературу.

Демонстрационная плата USB для PIC16C745/765

В ближайшее время также появится вариант этой платы для поддержки представителей семейства PIC18 с модулем USB. Демонстрационная плата (рис. 2.5) предоставляет множество разных возможностей.

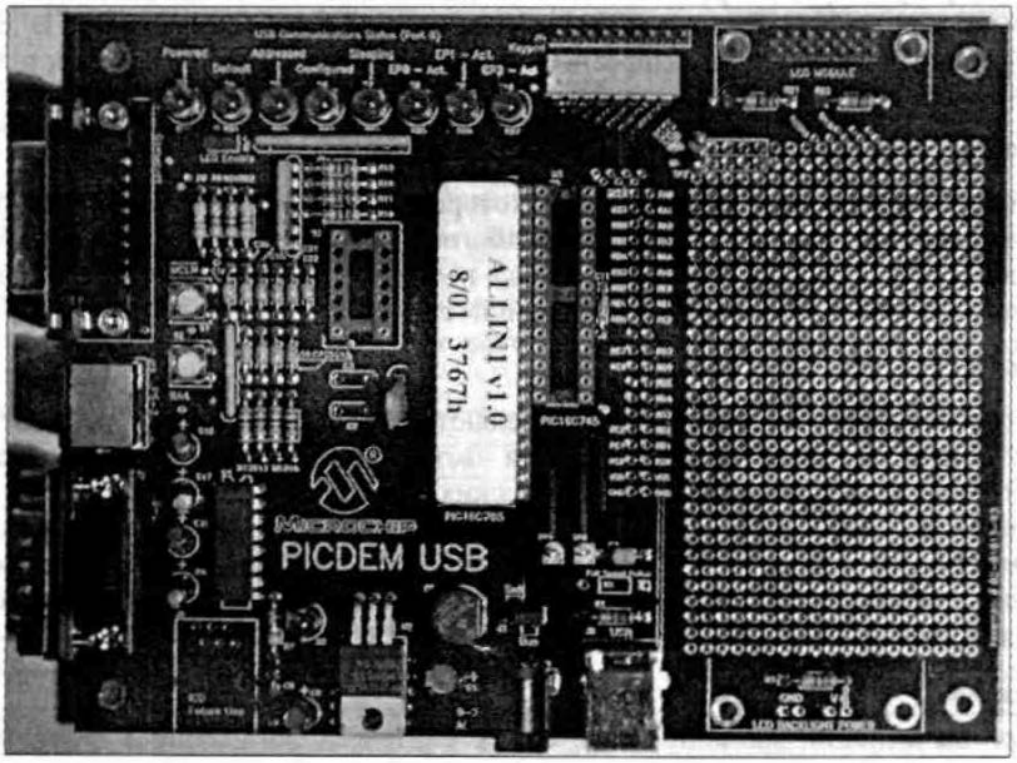


Рис. 2.5. Плата для демонстрации работы с шиной USB микроконтроллеров PIC16C745/765

Электропитание

Электропитание этой платы несколько отличается от стандарта. Гибкий вход AC/DC, рассчитанный на 9...20 В, подключен к стабилизатору напряжения 5 В и может отключаться, то есть, переключаться из режима "собственное питание" в режим "питание от шины". Таким образом, не должно подключаться никаких источников питания. Можно также питать плату от шины USB. Возможность подключения батарей отсутствует.

Блок V.24

В устройстве последовательного интерфейса нет ничего особенного. Он просто присутствует и ждет своего применения.

Светодиодная индикация

Светодиоды по традиции подключены к порту B, однако их обозначения полностью имеют отношение к USB. В результате при проверке программ можно наглядно увидеть то, что происходит в шине.

Кнопки

На этот раз в Microchip решили несколько сэкономить на кнопках: присутствует только кнопка сброса и кнопка на входе RA4.

Выбор осциллятора

Для контроллера USB RC-осциллятор не имеет смысла, поэтому его даже нет на плате, однако никто не мешает выпаять имеющийся резонатор и работать с кварцем или кварцевым осциллятором. Контроллеры USB были, пожалуй, первым применением микроконтроллеров PIC с фазовой автоподстройкой частоты (ФАПЧ) — задолго до появления PIC18. Благодаря ФАПЧ, PIC16C745/765 может обходиться внешним кварцем или резонатором на 6 МГц, внутренне же он работает на частоте 24 МГц. В том случае, если частота 24 МГц используется также и внешне для других задач, можно применить кварцевый осциллятор и отключить ФАПЧ.

Интерфейс USB

Последний элемент периферии на этой демонстрационной плате — самый важный. В конце концов, мы имеем дело с демонстрационной платой для USB. Гнездовой разъем USB имеет тип B.

Демонстрационная программа

В демонстрационном программном обеспечении так много всего, что в нем легко заблудиться. Демонстрационный пример уже "зашит" в поставляемом микроконтроллере PIC и отображает на экране монитора вращающийся курсор в те моменты, когда его не перемещают с помощью клавиатуры или мыши. Для этого достаточно подсоединить плату и выполнить автоматический поиск драйвера.

Литература от Microchip

Для получения навыков работы с USB компания Microchip предоставляет множество указаний по применению и других документов. Достаточно на домашней странице Microchip задать поиск по ключевому слову "USB" — и вы будете "засыпаны" различными вариантами. В ответ на то же ключевое слово в Google будет получено ошеломляющее изобилие статей. Кроме того, соответствующая документация находится на компакт-диске, прилагаемом к демонстрационной плате USB.

2.6.2. Подсказки начинающим

Мы уже давно заслужили титула чемпионов мира по подсоединению USB-устройств и установке их драйверов, и потому хотели бы предложить ряд подсказок в пользу начинающим.

- Аббревнатура HID (Human Interface Device) не означает, что ввод-вывод имеет отношение, главным образом, к клавиатуре, мыши и экрану.
- Низкоскоростные устройства работают на скорости 1,5 Мбит/с, однако она применима не для каждого устройства!! Длительность бита при малой скорости — 667 нс.
- Высокоскоростные устройства работают на скорости до 15 Мбит/с. В этом случае длительность бита составляет 83,3 нс.
- К шине USB могут подключаться до 127 устройств, включая концентратор.
- "Горячая" замена означает, что можно произвольно подключать и отключать почти любое устройство. Будьте внимательны с жестким диском! Вначале снимите его с регистрации!
- Интерфейс USB поддерживают операционные системы Windows 98SE и выше.

- USB-устройство поддерживает два способа питания: от шины и собственное (максимум 100 мА на одно устройство).
- Типы разъемов: A и B. Используется четыре линии: +5V, D–, D+, GND.
- Все свободные кабели — высокоскоростные. Односторонние низкоскоростные кабели всегда подключаются только к низкоскоростным устройствам.
- В полноскоростном устройстве линия D+ соединена через сопротивление 1,5 кОм с напряжением 3,3 В (в низкоскоростном приборе — линия D–).
- Есть только один ведущий и это — всегда ПК. Инициатором любых действий всегда является только он.
- Пакеты данных могут составлять от 8 до 256 байт.
- Кадр занимает 1 мс. В его пределах могут передаваться друг за другом различные пакеты данных для различных устройств.
- Поскольку из сигнала должно извлекаться тактирование, то устройство битовой подстановки следит за тем, чтобы после шести одинаковых битов вставлялся бит, изменяющий уровень.
- За работу на уровне линии отвечает передатчик, а затем — SEI (Serial Interface Engine). Оба вместе образуют аппаратный USB-модуль PIC.
- Функция концентратора: к неиспользуемому USB-подключению концентратора сигналы не посылаются. Как только подсоединяется какое-нибудь устройство, концентратор опознает его тип и может приступать к связи на соответствующей скорости. Сначала производится сброс шины (10 мс обе линии в состоянии низкого уровня), после чего устройство готово к регистрации. Система (хост) назначает устройству однозначно определяющий его номер.
- Высокоскоростные сигналы не могут поступать к низкоскоростным устройствам, за чем следит конечный концентратор. Концентратор — не бездумный распределитель: он знает, какие устройства к нему подключены.
- Типы передачи: управления, с прерываниями, массы данных и изохронная.
- Передача управления служит для управления аппаратным обеспечением. Она имеет высокий приоритет, использует протокол с отслеживанием ошибок и поддерживает имеет высокую скорость до 64 байт на запрос.
- Передача с прерываниями предназначена для повторяющихся малых порций данных (например, от мыши, клавиатуры). Передача 8 байтов за 10 мс.
- Передача массы данных предназначена для больших объемов данных, которые не критичны по времени. Реализует отслеживание ошибок. Типична для принтеров, сканеров. Имеет низкий приоритет.
- Изохронная передача предназначена для больших объемов, критичных по времени, без отслеживания ошибок (пример, — данные от звуковой карты).
- Регистрация — это опрос свойств устройства (дескрипторов) и назначение ему адреса. Только после этого прибор готов к работе.
- Существуют следующие дескрипторы: дескриптор устройства; дескриптор конфигурации; дескриптор интерфейса; дескриптор конечной точки; при известных случаях, дескриптор строки.
- Обязательные функции драйвера: `createfile()`; `closehandle()`; `readfile()`; `writefile()`; `deviceIOcontrole()`.

ГЛАВА

3 PIC18

В этой главе мы остановимся только на тех свойствах семейства PIC18, которые являются новыми по отношению к предыдущему поколению. Хотя мы исходили из того, что читатель обладает познаниями PIC12 или PIC16, все же наиболее важные факты в отношении этих семейств по тексту кратко излагаются.

Когда при описании новых свойств мы употребляем слова, наподобие, "раньше", то всегда подразумеваем под этим базовую серию или же среднее подсемейство микроконтроллеров PIC, то есть представителей с разрядностью памяти программ 12 или 14 бит. **Но это ни в коем случае не означает, что эти семейства ушли в прошлое.** В большинстве случае мы и по сей день выполняем разработки на микроконтроллерах типа PIC16, хотя зачастую это связано с явно выраженными сложностями в машинном управлении. Как производители услуг, мы должны всегда выбирать оптимальное и выгодное по цене решение.

Глава о PIC18 будет интересна также всем PIC-разработчикам, которые еще не планируют проекты для представителя нового поколения.

На опасливый вопрос: "Я должен опять приспосабливаться к совершенно новому типу микроконтроллеров?" — мы можем уверенно ответить: "Нет!". Несмотря на множество новых удобств и возможностей, с которыми нужно ознакомиться, основной принцип остался тем же. Это касается, в частности, аппаратных модулей, которые, хотя и заметно развились, тем не менее, не утратили своих хорошо знакомых принципов работы.

Компания Microchip немало потрудилась для сохранения совместимости с уже существующими PIC-структурами. Для этой цели каждое новое удобство можно отключать программно, если его нельзя просто проигнорировать. После сброса при этом, как правило, царствует "совместимое состояние".

Для тех, кто хорошо знаком с семейством PIC16, переход на PIC18 не составит особого труда. Аспекты, которые необходимо принять во внимание при переносе существующей программы для PIC16 в PIC18, описывается ниже в разделе 3.11.

Новые возможности семейства PIC18 не только расширяют область применения микроконтроллеров PIC, но и предоставляют много новых удобств при разработке проектов.

В частности, были учтены потребности компиляторов с языков высокого уровня, так что привычные жалобы на "издержки" при программировании на языке высокого уровня теперь будут звучать значительно реже. Но программист на ассемблере при чтении руководства по эксплуатации PIC18 также убедится, что теперь сможет исполнить многие из своих давних тайных желаний.

Поскольку появилось намного больше возможностей, необходимо знать намного больше об аппаратном обеспечении и обслуживании. Эта глава не задумывалась как перевод руководства по эксплуатации! Наши усилия направлены на то, чтобы

дать исчерпывающее описание тех аспектов, на которые следует обратить внимание. Что же касается подробностей, то мы настоятельно рекомендуем тщательно изучать руководства по эксплуатации и, прежде всего, технические паспорта.

Представители микроконтроллеров PIC первого поколения различались, главным образом, размерами памяти и числом выводов. Во втором поколении, главное внимание следовало обращать на аппаратные модули. В последнее время можно заметить также различия во внутренних свойствах (например, осциллятор, управление питанием).

При выборе представителя PIC18 в любом случае следует прочитать соответствующий паспорт. В этой главе мы не рассматриваем различия между представителями, а указываем исключительно на то, какие из свойств реализованы в том или ином ряде моделей. Но даже для нас это — не такая уж простая задача, поскольку изменения сейчас происходят быстрее, чем когда бы то ни было.

Совет

Приступая к новому проекту, всегда ищите самую свежую информацию!

Главная цель этой книги — дать понимание философии микроконтроллеров PIC, направленной на выполнение как можно большего количества сложных технических требований как можно более простым способом. По этой причине в дальнейшем речь пойдет не о множестве мелочей, а о путях их объединения в логически последовательное целое.

3.1. Архитектура и центральный процессор

Неизменным остался принцип архитектуры PIC: шина данных и шина команд разделены (Гарвардская архитектура). При этом две шины могут иметь различную разрядность. Кроме того, во время выполнения команды из памяти программ уже может выбираться следующая команда (конвейеризация).

На сегодняшний день разрядность команд PIC18 составляет 16 бит. Это позволяет не только адресовать больше памяти данных и программ, но и добавить некоторые новые команды. Существует около 40 новых команд, при этом "старые" команды по своим функциям не совсем идентичны командам среднего подсемейства микроконтроллеров PIC (см. раздел 3.11).

Проблема, связанная с ограниченностью адресного пространства, хотя и смягчена увеличением разрядности слов команд до 16, полностью не устранена. В виду этого, при разработке центрального процессора было принято компромиссное решение: в то время как длина большинства команд, как и раньше, составляет одно слово, для важных целей добавили четыре команды длиной в два слова.

Еще одно преимущество 16-тиразрядной памяти программ заключается в том, что теперь можно помещать на место слова программы два полных байта, к которым можно обращаться с помощью команд, предназначенных для работы с таблицами. И при этом, не только считывать, но и записывать!

Мы очень обрадованы дополнительными RAM-указателями (FSR), которых теперь стало три. С помощью операций, основанных на FSR (косвенная адресация), теперь возможно автоматическое инкрементирование или декрементирование соответствующих регистров FSR. Разрядность FSR составляет 12 бит, что позволяет адресовать до 4 кБайт.

Для того чтобы надлежаще использовать большие адресные пространства, некоторые представители PIC18 оснащены заметно большей памятью. Это касается не

только памяти данных, размер которой может достигать 4 кБайт, но и памяти программ, которая в настоящее время достигает 128 кСлов (хотя эти данные ко времени издания книги уже могли улучшиться). Теоретически, память программ допускает адресацию до 2 МБайт (1 МСлов программы).

Для того чтобы использовать эти 2 МБайт полностью, теперь существует возможность обработки кода программы, находящегося во внешней памяти.

Кроме того, нас очень порадовало новое аппаратное умножение, которое выполняется за один командный цикл.

Следующее новшество ядра PIC18 касается структуры прерываний. Сейчас есть две иерархии прерываний, к которым, соответственно, принадлежат два вектора прерываний. Кроме того, для прерывания с более высоким приоритетом появилось аппаратное помещение в стек и извлечение из стека содержимого наиболее важных регистров, что придает прерываниям гораздо больше значения.

3.2. Память программ

Поскольку 16-тиразрядная память программ может содержать не только слова кодов операций , но и байты данных, теперь она адресуется побайтно. Существуют команды, с помощью которых можно обращаться к любому байту памяти программ (рассматриваются ниже).

Указателем для доступа к памяти программ служит счетчик команд (PC). В семействе PIC18 его разрядность расширена до 21 бит. Разумеется, доступ на чтение и запись возможен только к младшему байту счетчика команд (PCL), а к старшему можно обращаться как и раньше — через соответствующий регистр-фиксатор. В дополнение к уже знакомому нам регистру PCLATH, в PIC18 существует также регистр PCLATU для старших адресов.

Если регистру PCL назначают нечетное значение (например, по команде `ADDWF PCL` или `GOTO`), то в этом случае игнорируется младший разряд, который устанавливается равным нулю. В результате таблицы, которые считываются по команде `RETLW`, могут иметь длину всего 128 байтов, если не манипулировать регистром `PCLATH` (при этом должна приниматься в расчет команда `ADDWF PCL`).

3.3. Доступ к памяти программ

Доныне чтение данных было возможным только с помощью команды `RETLW`. Поскольку слово команды имело разрядность только 14 или 12 бит, то, так или иначе, в нем могли помещаться только данные размером в один байт.

Само собой разумеется, доступ с помощью команды `RETLW` все так же возможен, однако с его помощью в слово программы можно поместить все тот же один байт.

3.3.1. Чтение памяти программ

Чтение байта данных происходит по команде **`TABLRD`**, которая имеет несколько разновидностей. В любом случае, эти команды используют два цикла.

Команда `TABLRD` не имеет аргументов. Результата выполнения команды чтения заносится в регистр **TABLAT**. Для адресации служит указатель размером в три байта под названием **TABLPTRU: TABLPTRH:TABLPTRL**.

Если читатель ожидает, что для команды `TABLRD` существуют варианты с автоматическим инкрементированием и декрементированием, то он не будет разочарован. Более того, можно выбрать момент увеличения/уменьшения указателя: перед

выполнением чтения (преинкремент) или же после него (постинкремент/постдекремент). Официальный язык ассемблера использует для этой цели следующие виды обозначений:

- TABLRD* — указатель не изменяется;
- TABLRD*+ — постинкремент;
- TABLRD*- — постдекремент;
- TABLRD+* — преинкремент.

Регистром назначения для команд TABLRD всегда является регистр TABLAT, однако на практике в качестве такого регистра было бы лучше использовать регистр W. Реализацию интерфейса между 16-тиразрядной памятью программ и 8-миразрядной памятью RAM никак не назовешь удобством. Как правило, приходится сразу же после команды TABLRD помещать команду MOVF TABLAT,W, то есть, для считывания одного байта из памяти программ в регистр W требуется две команды.

Кроме того, должны загружаться три байта регистра-указателя TABLPTR. При повторяющихся считываниях, как правило, заботиться об этом указателе нет необходимости, если грамотно воспользоваться возможностями инкрементирования и/или декрементирования.

3.3.2. Запись в память программ

Запись в память программ физически не может быть выполнена за время нескольких командных циклов. Таким образом, не позволяйте ввести себя в заблуждение тем, что по аналогии командам TABLRD, существуют также соответствующие команды TABLWT, которым в каждом случае требуется два цикла. Эти команды просто переносят содержимое регистра TABLAT в регистр HOLD (точнее сказать, — в один из регистров HOLD, которые при записи в память программ служат промежуточными буферами. Эти буферные ячейки не относятся к адресуемой области памяти).

Точная процедура инициализации процесса записи после загрузки регистра (или регистров) HOLD для различных представителей PIC18 отличается, и потому за дополнительной информацией следует обращаться к соответствующему техническому описанию. Сам процесс записи зависит от технологии изготовления памяти программ. В документации от Microchip доступ на запись обозначается как "long write". Запись всегда производится блоками, размер которых составляет минимум два байта.

Процесс записи одного блока занимает время в пределах миллисекунд. Более точные данные могут быть взяты из технических описаний. В течение этого времени центральный процессор находится в состоянии останова.

Для того чтобы дать представление о том, как в общем случае протекает процесс записи, рассмотрим пример записи в флэш-память программ блоками размером восемь байт.

При описанном здесь способе записи в память программ всегда записывается блок из восьми регистров HOLD. Таким образом, все эти регистры сначала должны загружаться с помощью команды TABLWT. Команда TABLWT записывает содержимое регистра TABLAT в один из регистров HOLD. Три младших разряда указателя TABLPTR определяют, в какой из восьми регистров HOLD записывается значение.

Когда все регистры HOLD загружены правильными значениями, начинается процесс записи. Запись одного отдельного байта не предусмотрена.

Перезапись одного отдельного блока в восемь байтов невозможна — перед записью соответствующая область памяти программ должны быть предварительно очищена. Однако процесс стирания всегда выполняется по 64 байта ("row erase" — стирание строки). Это значит, что при перезаписи уже запрограммированной области памяти должны заново записываться, по крайней мере, восемь блоков по восемь байт каждый.

Процесс стирания и записи инициируется одинаковыми последовательностями команд. Различие между обоими процессами заключается в разряде FREE регистра EECON1.

- EECON1,FREE = 1 — стирание блока из 64 байтов;
- EECON1,FREE = 0 — запись блока из 8 байтов.

Регистр-указатель TABLPTR задает адрес назначения для этих операций. При стирании игнорируются пять младших разрядов, а при записи — три.

Поскольку доступ на запись к внутренней памяти EEPROM и регистрам конфигурации также производятся с помощью одинаковых последовательностей команд, то вначале о необходимости доступа к флэш-памяти должен быть уведомлен регистр EECON1. Это реализуется с помощью двух команд:

```
BCF     EECON,CFGS
BSF     EECON,EEPGD
```

Затем применяется дополнительная мера предосторожности, призванная предотвратить непреднамеренную запись:

```
BSF     EECON,WREN
```

Если этот разряд не установлен, тогда последующая установка разряда WR в регистре EECON (начинает процесс записи) остается бездейственной.

Теперь, прежде чем начать стартовый ритуал, требуется сбросить разряд GIE в регистре INTCON, если он установлен.

Стартовый ритуал выглядит всегда одинаково:

```
MOVLW   55H
MOVWF   EECON2
MOVLW   0AAH
MOVWF   EECON2
BSF     EECON2,WR
```

Над регистром EECON2 ломать голову не нужно, поскольку физически его не существует.

С помощью представленной выше последовательности команд начинается доступ на запись. Не забывайте, что после этого центральный процессор некоторое время находится в состоянии останова. За вышеуказанной последовательностью команд должна быть помещена команда NOP (в некоторых версиях — три NOP). После этого не следует забывать снова установить, если необходимо, разряд GIE в регистре INTCON.

В обобщенном виде вся последовательность команд для обращения к флэш-памяти программ выглядит следующим образом (FLACC = "Flash-access"):

```
FLACC   BCF     EECON1,CFGS     ; Не область конфигурирования
        BSF     EECON1,EEPGD    ; Память программ, а не EEPROM
        BSF     EECON2,WREN     ; Для безопасности
        BCF     INTCON,GIE      ; Обязательно!
```

```
        MOVLW   55H             ; Эта последовательность не
        MOVWF   EECON2          ;   должна прерываться!!
        MOVLW   0AAH
        MOVWF   EECON2
        BSF     EECON2,WR       ; Начало процесса записи
        NOP
        NOP
        NOP
        BSF     INTCON,GIE      ; Опять разрешаем прерывания
```

3.4. Память данных

Когда говорят о памяти данных, то в первую очередь имеют в виду рабочие регистры, к которым можно обращаться посредством прямой или косвенной адресации.

Кроме рабочих, существует также ряд "фоновых" регистров данных, к которым невозможно получить доступ с помощью привычных команд напрямую ни для чтения, ни для записи. У микроконтроллеров PIC к ним относятся регистры стека для хранения адресов возврата. Сейчас к ним можно обращаться по указателю. Кроме того, существует также небольшой стек данных, в котором на аппаратном уровне сохраняются важные регистры при обработке прерываний или при выполнении команд типа CALL.

3.4.1. Адресация рабочих регистров

К 4096 рабочим регистрам можно обращаться косвенно с помощью регистров-указателей (FSR) или непосредственно. Косвенная адресация линейна, то есть, регистры-указатели содержат адрес рабочего регистра. Нововведение PIC18 заключается в том, что регистры-указатели теперь 12-тиразрядные. Эти регистры специального назначения расположены в самом конце адресного пространства.

Для прямой адресации память логически разделена на 16 банков по 256 рабочих регистра в каждом. Один из этих банков должен предварительно выбираться с помощью регистра выбора банка (BSR). Кроме того, при прямой адресации в любой момент, без смены выбора банка, можно получить доступ к особой области в 256 байт, которую называют "банком доступа" ("access bank"). В банке доступа содержатся 128 регистров специального назначения и 128 рабочих регистров произвольного применения. О том, как выглядят команды адресации, сказано в представленному ниже разделе, посвященном структуре команд. Сейчас же отметим одну важную особенность набора команд.

Важно!

С точки зрения ассемблера, команды с прямой и косвенной адресацией, как и прежде, идентичны по форме.

3.4.2. Стек возврата

Классической функция стека возврата: при вызове команд типа CALL в нем сохранялся адрес возврата (21 бит) — адрес, который следует за командой CALL

PC+2). Этот адрес располагается на "вершине" стека. По первой же команде типа RETURN этот адрес загружается в счетчик команд.

Размер стека ограничен, и у микроконтроллеров PIC первого поколения допускал хранение только двух адресов. Это означало, что вложенность команд CALL не могла превышать глубины двух уровней. У второго поколения PIC с 14-тиразрядным ядром размер стека возврата был расширен до семи адресов. **Архитектура PIC18 предоставляет в распоряжении стек глубиной 31.**

Со стеком связан особый указатель, который обычно обозначается как SP (от английского "stack pointer"). В классическом случае применения указатель стека указывает на последний помещенный адрес, и, таким образом, определяет вершину стека.

У многих процессоров и контроллеров стек вместе со своим указателем находится в адресуемой области RAM. До этого времени у микроконтроллеров PIC запись в стек производилась исключительно по прерываниям, а также командами типа CALL, а считывание — только по командам типа RETURN.

В семействе PIC18 появилось новшество, которого, наверное, ожидали многие пользователи микроконтроллеров PIC. Хотя сами регистры стека не находятся в адресуемой области RAM, зато там расположен указатель стека, доступный не только на чтение, но и на запись (в допустимых пределах)! Это открывает интересные возможности управления, а также фантастические возможности для появления ошибок, поэтому при использовании подобных приемов следует быть крайне осторожным.

Регистр STKPTR

Организационным центром стека является регистр специального назначения под названием **STKPTR**, который, кроме пятиразрядного указателя стека, содержит два важных флага:

7	6	5	4	3	2	1	0
stkful	stkunf	–	sp4	sp3	sp2	sp1	sp0

Флаг **stkful** устанавливается тогда, когда стек полностью заполнен или переполнен. Флаг **stkunf** устанавливается при попытке выбрать адрес из стека, несмотря на то, что тот пуст. Если наступает одно из этих событий, то, согласно установке, может быть выполнен сброс. **Для того чтобы разрешить этот сброс, необходимо установить в регистре конфигурации 4L разряд stvren.** При таком сбросе разряды stkful и stkunf, разумеется, не сбрасываются, чтобы в дальнейшем можно было определить, чем был вызван сброс.

Пять разрядов от sp0 до sp4 доступны на чтение и запись, а разряды stkful и stkunf — только на чтение и обнуление. Разряд 5 всегда считывается как 0.

Указатель стека (sp4...sp0) после каждого сброса имеет значение 0. Это означает, что стек пуст. При каждом выполнении команды типа CALL указатель стека сначала инкрементируется, а потом в ту позицию, на которую он указывает, помещается адрес возврата. Таким образом, в позицию, на которую указывает указатель стека со значением 0, адрес возврата помещать нельзя.

Итак, обобщим представленную информацию:

- стек состоит из 31 регистра с разрядностью 21;
- эти регистры — неадресуемые рабочие регистры.

Вершина стека

Вершиной стека мы обозначаем регистр стека, на который указывает указатель стека. Словом "вершина", собственно говоря, подразумевается, что речь идет о верхней ячейке хранения, — адресе возврата, который помещается в стек последним. Тем не менее, поскольку указатель стека может изменяться, то его вершиной можно сделать любой регистр стека.

С этим же "верхним" регистром ассоциируется регистр под названием TOSU:TOSH:TOSL, состоящий из трех рабочих регистров, доступных на чтение и запись (физически наличные регистры).

Push и Pop

Есть две связанные со стеком команды, не имеющие аргументов. Команда Push инкрементирует указатель стека и переносит расположенный за ней адрес (PC+2) на вершину стека. Команда Pop декрементирует указатель стека и тем самым выполняет операцию, наподобие DECF STKPTR.

3.4.3. "Быстрый" регистровый стек

"Быстрый" стек состоит из трех регистров, которые служат для сохранения важных регистров W, STATUS и BSR при входе в подпрограмму обработки прерывания и, по желанию, — также при выполнении команд типа CALL. Затем они могут восстанавливаться по командам типа RETURN.

Доступ к регистрам этого стека на запись и чтение — исключительно аппаратный. Они находятся вне адресуемого пространства RAM, и потому их названия WS, STATUSS и BSRS представляют лишь теоретический интерес.

"Быстрый" стек, собственно говоря, не заслуживает названия "стек" — здесь не может идти речь о складировании, поскольку сохраняется только строго определенный набор регистров.

Для того чтобы выразить пожелание на сохранение трех указанных выше регистров, команда CALL снабжена одноразрядным аргументом, который обозначается как "s" (вероятно, от слова "save") и для реализации новшества должен устанавливаться в "1". Кроме того, ассемблер вместо "1" распознает слово "fast".

Внимание!

В данном случае значение по умолчанию — "0".

"Теневые" регистры WS, STATUSS и BSRS находятся вне адресуемого пространства RAM, и доступны только на считывание их содержимого обратно в соответствующие регистры по команде типа RETURN.

Команды RETURN теперь также имеют аргумент s, который устанавливается в "1", если требуется восстановить содержимое регистров W, STATUS и BSR. С этим параметром следует обращаться очень осторожно в случае применения вложенных подпрограмм. Если же говорить более конкретно, то **необходимо быть осмотрительным при вызове команды Return fast**. Эта команда разрешается исключительно при возврате из вызова наиболее высокого приоритета. **Для обычных подпрограмм параметр fast неприменим при разрешенных прерываниях**. Это объясняется тем, что впоследствии, в случае разрешения прерываний, в программе могут возникнуть приводящие в негодование недоразумения.

Настоятельно рекомендуем лишний раз с "теневыми" регистрами не "играться".

3.5. Порты ввода-вывода

В портах ввода-вывода семейства PIC18 появилось два новшества:

- может присутствовать до 12 портов (хотя модели с таким количеством выводов сильно "бьют по карману");
- для каждого порта существует адресуемый фиксатор — регистр LAT (рис. 3.1).

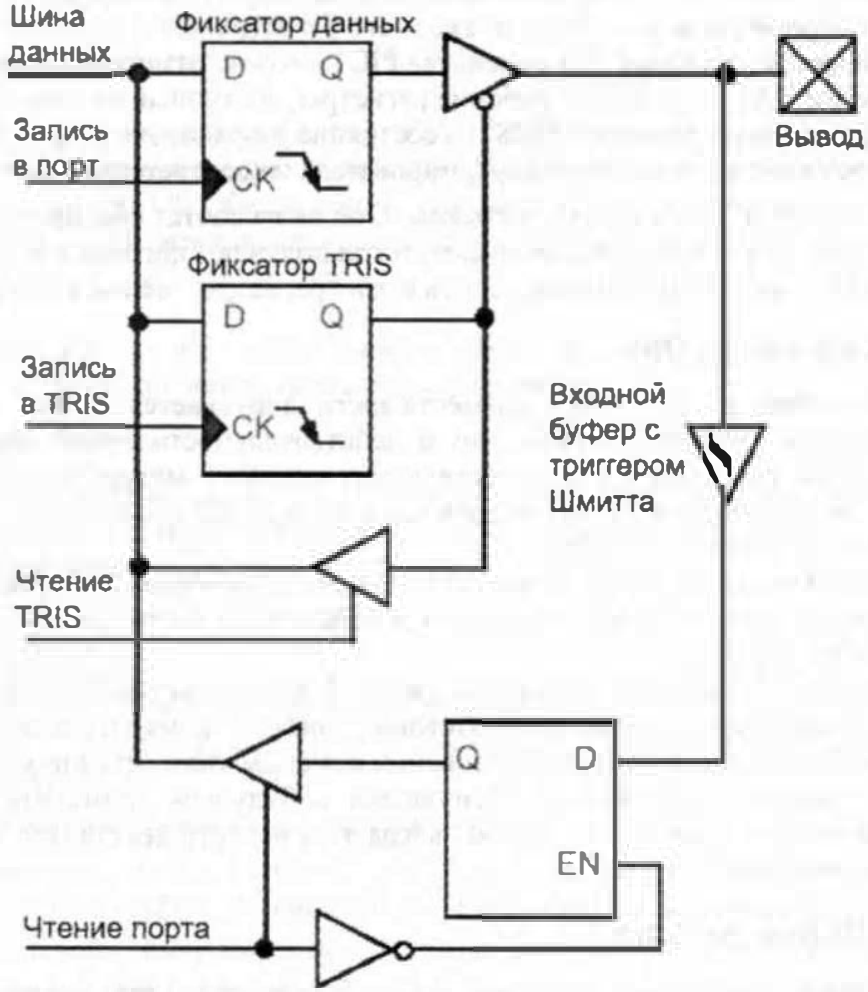


Рис. 3.1. Типичный порт

3.5.1. Регистр LAT

В том, что для каждого порта у PIC18 имеется регистр LAT, собственно говоря, нет ничего нового — эти регистры присутствовали и в базовой серии, и в среднем подсемействе микроконтроллеров PIC, однако раньше они не имели собственных адресов, а значит — и собственных названий. При записи в регистр PORT, фактически, происходит запись в регистр LAT, а при чтении из регистра PORT считывается состояние вывода.

В случае PIC18, регистр LAT порта — выходной и теперь адресуется. Значение из этого регистра выдается на соответствующие выводы, если соответствующий выходной усилитель-формирователь не заблокирован.

Как и прежде, должен ли выходной усилитель-формирователь быть заперт или разрешен, определяет регистр TRIS:

- 0 в некотором разряде означает открытое состояние усилителя формирователя соответствующего вывода (вывод — выход);
- 1 в некотором разряде означает закрытое состояние усилителя формирователя соответствующего вывода (вывод — вход).

Таким образом, можно сказать, что PORT — это "входной", а LAT — выходной регистр. Слово "входной" написано в кавычках, потому что PORT в обычном смысле не является регистром, в отличие от входного регистра.

Таким образом, обобщая, для семейства PIC18 можно отметить следующее.

- Регистры LAT — обычные рабочие регистры, доступные на чтение и запись. Если позволяют регистры TRIS, то состояние разрядов регистра LAT выдается через выходной усилитель-формирователь на соответствующие выводы.
- Регистры PORT — входные регистры. Они не являются обычными рабочими регистрами. Если из них считывают, тогда получают физическое состояние выводов. Собственно говоря, запись в эти "регистры" не имеет смысла.

Запись в регистр PORT

Тем не менее, по причинам совместимости допускается запись в регистры PORT. При этом следует помнить, что в действительности запись производится в регистры LAT, как в случае предшествующих семейств микроконтроллеров PIC. На рис. 3.1 видно, что там стробы записи для LAT и PORT объединяются посредством логического элемента "ИЛИ".

При чтении регистра PORT не всегда получают то значение, которое перед этим было записано в регистр PORT, поскольку в действительности запись была выполнена в регистр LAT.

Если по старой привычке применить вместо LATB название PORTB, то следует учитывать следующее: в случае использования "чистой" команды записи имя регистра значения не имеет, но при использовании команды типа "чтение-модификация-запись" — разница существенная. Приучитесь в будущем применять обращение к PORTB на запись только в том случае, когда этот регистр действительно подразумевается как входной!

3.5.2. Порты от A до L

При чтении технических описаний PIC18 бросается в глаза множество новых портов: от F до L. Теперь в общей сложности возможно наличие двенадцати портов, для которых в области регистров специального назначения зарезервировано до 36 адресов (на каждый порт приходится по три регистра).

Некоторые из выводов портов, кроме общих функций ввода-вывода, имеют также альтернативные функции, связанные с аппаратными модулями. Давно известные порты (от A до E) имеют, преимущественно, прежние свойства и дополнительные функции, связанные с аппаратными модулями.

Порт A

Порт A традиционно отвечает за входы АЦП, которые в PIC18 могут также относиться и к порту E. В качестве цифровых входов, выводы порта A имеют структуру TTL, за исключением вывода 4, который является счетным входом для TIMER0 и поэтому включает в себя триггер Шмитта.

Порт B

К компетенции порта B традиционно относятся прерывания INT0 и RBINT. Он имеет входы со структурой TTL, поскольку они не применяются для особых функций (прерываний, модуля CCP или программирования). TTL-входы могут, на выбор, снабжаться подтягивающими резисторами с небольшим сопротивлением.

Порт C

Порт C, как и у предшественников PIC18, отвечает за последовательный обмен данными и работу с модулями CCP.

Порт D

Порт D — это, как и прежде, порт PSP (Parallel Slave Port). Раньше PSP конфигурировался с помощью разрядов регистра TRISE, теперь же этой цели служит отдельный регистр PSCON. Когда порт E становится восьмиразрядным, старшие разряды регистра TRISE отвечают за направление передачи данных для этих выводов.

Порт E

К порту E все так же относя линии управления для порта PSP. Как только что упоминалось, он может иметь также и восемь выводов.

Порты от F до L

Для новых портов в руководстве по применению отсутствуют упоминания об их применении для аппаратных модулей. Порты от F до L описываются как порты с входами, имеющими характеристику триггера Шмитта.

3.6. Таймеры

В отношении таймеров в PIC18 есть ряд новшеств:

- Timer0 — на выбор, 8-ми- или 16-тиразрядный таймер/счетчик; старший байт буферизирован;
- Timer0 может отключаться программно;
- предварительный делитель Timer0 больше не используется совместно со сторожевым таймером, но также может выключаться программно;
- сторожевой таймер имеет постделитель, который настраивается только при программировании;
- сторожевой таймер может быть запрещен программно; получить разрешение программно он может только в том случае, если при программировании было установлено общее разрешение;
- давно известный регистр OPTION заменен регистром T0CON; при этом у него не только новое название, но также и новые разряды;
- старший байт буферизирован также и в Timer1 (это свойство можно включать программно);
- осциллятор Timer1 в некоторых моделях может при желании использоваться для системной синхронизации;
- появился еще один 16-тиразрядный таймер/счетчик (TMR3), который может применяться как альтернатива Timer1 для модулей CCP.

Следует уделять особое внимание состоянию таймеров и их регистров конфигурации после различных видов сброса. **Для того чтобы была гарантирована совместимость с прежними свойствами таймеров, после сброса новые свойства должны отключаться.** Таким образом, они всегда могут подключаться программно в конфигурационных регистрах.

3.6.1. Буферизированные регистры 16-тиразрядных таймеров

При операциях чтения и записи в таймере, состоящем из двух независимых регистров, всегда возможна небольшая проблема: регистр младшей части значения может переполняться в промежутке между двумя циклами обращения и, в итоге, младший и старший байты больше не будут относиться к одному и тому же значению. Это может происходить независимо от последовательности обращения.

Для того чтобы решить эту проблему программно, требуется несколько дополнительных команд: при чтении из таймера сначала считывают старший байт, а потом — младший. После этого старший байт считывают во второй раз. Если второй результат не совпадает с первым, тогда должно быть выполнено второе считывание и из младшего байта. Аналогичным образом поступают и при записи.

Во избежание подобных "маневров", в PIC18 в распоряжение пользователя предоставляется буферный регистр. При чтении из младшего байта таймера содержимое старшего байта записывается на аппаратном уровне в буферный регистр. При записи младшего байта таймера в его старший байт переносится значение из буферного регистра.

Таким образом, процесс чтения/записи для старшей части результата в таймере протекает через соответствующий буферный регистр. Для практической реализации чтения и записи для каждого буферизированного 16-тиразрядного таймера вытекает следующее:

- чтение — сначала считать из TMRL, потом — из буферного регистра TMRH;
- запись — сначала записать в буферный регистр TMRH, затем — в TMRL.

3.6.2. TMR0

Настал момент распрощаться со старым-добрым регистром OPTION. Отныне конфигурационный регистр для TMR0 называется T0CON и содержит только те разряды, которые отвечают за режим работы TMR0. "Чуждые" разряды INTEDG и /RBPU теперь находятся в регистре INTCON0.

Если для PIC18 необходимо использовать программу, составленную для PIC16, то следует уделить особое внимание конфигурационному регистру TMR0 — в нем изменилось не только название.

T0CON:

7	6	5	4	3	2	1	0
tmr0on	t08bit	tosc	tose	psa	tops2	tops1	tops0

Для работы TMR0 должен быть установлен в "1" новый разряд tmr0on. Таким образом, не забывайте: для использования TMR0 его следует включить!!!!

То, что теперь таймером TMR0 при желании можно управлять как 16-тиразрядным таймером/счетчиком, совместимости особо не мешает: старший байт можно просто игнорировать. Однако, если используют прерывание по переполнению в младшем байте, то должен выбираться режим работы с разрядностью 8.

Для того чтобы включить режим с разрядностью 8, разряд t08bit должен быть установлен в "1".

То, что предварительный делитель больше не используется совместно со сторожевым таймером, — большой плюс. Сторожевой таймер без предварительного делителя, как правило, слишком "прыткий", приходилось отказываться от предварительного делителя для TMR0. Однако зачастую от этого 8-миразрядного предварительного делителя отказываться очень не хочется, поскольку он позволяет получить коэффициент деления до 256, а с таким значением в режиме с 16-ю разрядами до переполнения проходит больше 16 секунд.

Для того чтобы разрешить использование предварительного делителя, разряд "PSA" должен быть установлен в "1". Когда разряд PSA имеет значение "1" делитель частоты не учитывается, то есть имеет место соотношение 1:1.

Выбор коэффициента деления частоты осуществляется с помощью трех младших разрядов регистра T0CON:

Коэффициент = 1 : $2^{(ps2:ps1:ps0+1)}$.

Другими словами, комбинации 000 соответствует делитель 2, а комбинации 111 — 256.

Разряды tocs и tose — наши старые знакомые: при tocs = 1 TMR0 переводится в режим счетчика (то есть, его значение увеличивается с появлением очередного фронта на выводе T0CKI), при этом вид фронта определяется состоянием разряда tose (tose = 1 — ниспадающий фронт).

3.6.3. Сторожевой таймер

Конфигурация сторожевого таймера больше не совпадает с конфигурацией таймера Timer0. Теперь у него есть постделитель, позволяющий выбирать коэффициент деления до 32000. Сторожевой таймер тактируется внутренним осциллятором INTRC и имеет период 4 мс. Таким образом, максимальное время до наступления переполнения сторожевого таймера может составлять до 2,18 минут.

Такой период до переполнения позволяет экономить потребление тока путем вывода микроконтроллера из "спящего" режима через удобные промежутки времени. Выбор происходит с помощью регистра CONFIG2H, принадлежащего к конфигурационным регистрам центрального процессора. Этот регистр доступен для программной записи. Постделитель сторожевого таймера устанавливается при программировании.

Регистр CONFIG2H содержит также разряд wdten, разрешающий работу сторожевого таймера, который, как всегда, задается при программировании. Теперь этот разряд может также программно считываться (TABLRD).

Сторожевой таймер можно включить двумя способами. Если в конфигурационном регистре CONFIG2H установить разряд WDTEN равным 1, тогда сторожевой таймер будет включен без всяких "а может" и "если". Этим сохраняется совместимость с PIC16. Еще один способ включения сторожевого таймера — программный (разряд SWDEN в регистре WDTCON). Для экономии потребляемого тока в "спящем" режиме он может временно выключаться.

3.6.4. Timer1 и Timer3

Мы будем рассматривать известный нам Timer1 и новый модуль Timer3 совместно, поскольку они — "братья" с очень похожими свойствами и почти одинаковыми функциями. Оба таймера могут применять T1OSC, а также совместно использу-

ют счетный вход и управление модулями CCP. В остальном, они независимы, располагают собственными предварительными делителями и режимами работы. Для того, чтобы при управлении модулями CCP эти два таймера не конфликтовали между собой, служат разряды T3CCP1 и T3CCP2 в конфигурационном регистре T3CON.

T1CON:

7	6	5	4	3	2	1	0
RD16	<T1RUN>	T1CKPS1	T1CKPS0	T1OSCEN	/T1SYNC	TMR1CS	TMR1ON

T3CON:

7	6	5	4	3	2	1	0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	/T3SYNC	TMR3CS	TMR3ON

Если мы рассмотрим конфигурационный регистр модуля Timer1, то обнаружим, что два старших разряда: RD16 и T1RUN — новые.

RD16

Новое свойство модуля Timer1 — чтение и запись 16-тиразрядного регистра через буфер — может отключаться с помощью установки RD16 = 0. Это служит для совместимости с прежним поколением.

В регистре T3CON тоже присутствует разряд RD16 с аналогичной функцией.

T1RUN

Функция, связанная с разрядом T1RUN, есть не у всех представителей PIC18. **Путем установки T1RUN = 1 допускается использование осциллятора Timer1 в качестве системного.** Предпосылкой для этого, разумеется, является то, что этот осциллятор вообще включен, благодаря установке t1oscen = 1 (см. раздел 1.10).

Разумеется, разряды T1RUN и T1OSCEN — уникальные в своем роде. На соответствующих позициях в регистре T3CON расположены разряды T3CCP2 и T3CCP1.

T3CCP2 и T3CCP1

Было бы естественно предположить, что разряд T3CCP2 отвечает за работу модуля CCP2, а T3CCP1 — модуля CCP1, однако на самом деле все не так просто. Пара разрядов T3CCP2:T3CCP1 устанавливает, какой таймер управляет каким модулем. У разных типов PIC18 эта пара имеет разное значение.

Функция счетчика

Модулями Timer1 и Timer3 можно управлять независимо как счетчиками. Для этого должен быть установлен разряд t1cs = 1 (соответственно, t3cs = 1). При этом, однако, имеется только один счетный вход.

В отличие от таймера TMR0, отсутствует возможность выбора типа активного фронта, и потому всегда подсчитываются нарастающие фронты (после первого ниспадающего фронта).

При этом можно при желании отключить синхронизацию счетных импульсов. Отключение синхронизации внешних счетных импульсов происходит при /T1SYNC = 1 или T3SYNC = 1.

Включение Timer1 и Timer3

Оба таймера/счетчика могут включаться или выключаться с помощью разрядов TMR1ON в регистре T1CON и, соответственно, — TMR3ON в регистре T3CON. Включению соответствует значение "1". На предварительные делители эти разряды не влияют.

Предварительные делители Timer1 и Timer3

Подобно паре разрядов T1CKPS1:T1CKPS0 в регистре T1CON, коэффициент деления для Timer3 определяют разряды T3CKPS1:T3CKPS0 в регистре T3CON:

- 00 — делитель 1:1;
- 01 — делитель 1:2;
- 10 — делитель 1:4;
- 11 — делитель 1:8.

3.6.5. Timer2 (и Timer4)

Некоторые представители PIC18 обладают также таймером Timer4, обладающим теми же свойствами, что и Timer2.

В модуле Timer2 все осталось по-старому. Он, как и прежде, — аутсайдер среди таймеров. Его свойства:

- 8-миразрядный таймер без внешнего счетного входа, а значит – не "счетчик";
- значение переполнения можно программировать (регистр PR2);
- используется предварительный делитель, позволяющий выбрать коэффициент деления 1, 4 или 16;
- используется программируемый постделитель, который играет важную роль для прерывания по переполнению;
- таймер, управляющий выходом ШИМ.

T2CON:

7	6	5	4	3	2	1	0
	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Таймер Timer2 ведет счет не обязательно до 255, а до значения, которое находится в регистре PR2. Это значение переполнения делает возможным вместе с предварительным делителем создавать прерывания через гибко задаваемые отрезки времени. Программируемое время переполнения особенно важно в случае модуля ШИМ, где оно определяет частоту.

Внимание!

При использовании Timer2 нельзя забывать о регистре PR2. Если PR2 = 0, то таймер всегда содержит значение 0. Если переполнение должно возникать обычным образом (после значения 255), тогда регистр PR2 должен иметь значение 255.

Постделитель может принимать не только значения, которые являются степенью двойки, но любые значения от 1 до 16.

Постделитель = (TOUTPS3:TOUTPS2:TOUTPS1:TOUTPS0+1)

Пример

При 1001 постделитель получает значение 10 (9+1).

При коэффициенте предварительного делителя 4 и постделителя 10 можно организовать прерывание через каждые 10 мс.

3.7. Прерывания

Хорошие новшества PIC18 касаются прерываний. Теперь используется два уровня прерываний, с двумя векторами:

- адрес 08_h — низкий приоритет;
- адрес 18_h — высокий приоритет.

Но это еще не все! Для прерывания с высоким приоритетом при входе в подпрограмму обработки прерывания аппаратно сохраняются важные регистры (W, STATUS и BSR). Они могут снова (при желании) восстанавливаться по команде `RETFIE` — и притом за один командный цикл. Для этой цели у команды `RETFIE` есть одноразрядный аргумент, который обозначается как "s":

```
RETFIE    0              ; W, STATUS и BSR неизменны
RETFIE    1              ; W, STATUS и BSR восстанавливаются
```

Задействованный стек имеет глубину только в один набор регистров, поэтому, хотя при возникновении прерывания с низким приоритетом регистры и сохраняются, нельзя полагаться на то, что они будут присутствовать в стеке к окончанию прерывания, если разрешено прерывание с высоким приоритетом. Последнее может возникнуть в любой момент и затереть стек. Исходя из этого, если разрешено прерывание с высоким приоритетом, то прерывание с низким приоритетом должно завершаться командой `RETFIE 0`.

С новой структурой прерываний в семействе PIC18 истинное назначение прерываний — быстрая реакция на события, приведшие к прерыванию, — проявляется гораздо эффективнее (по крайней мере, для прерываний с высоким приоритетом).

Напоминаем, что между наступлением события и началом первой команды в обработчике прерывания проходит определенное время (задержка реакции). В случае синхронного возникновении событий задержка реакции всегда составляет три командных цикла, даже когда прерываются двухсловные или двухцикловые команды. При внешних прерываниях задержка, как правило, на один командный цикл дольше, и, таким образом, составляет четыре цикла.

Если при этом еще необходимо предварительно сохранить важные регистры и затем, смотря по обстоятельствам, — выполнить ветвление в требуемую функцию обработки прерывания, то непосредственно до обработки проходит кое-какое время.

Раньше при разрешении нескольких источников прерываний обработчики могли заблокировать друг друга. В PIC18 эта проблема решена, и прерывание с высоким приоритетом не должно ожидать окончания обработки менее актуального прерывания.

Источников прерываний стало больше. Внешнее прерывание получило двух "братьев". Теперь выводы 1 и 2 порта также способны воспринимать сигналы на прерывание (INT1, INT2). Давно известный вывод INT в PIC18 обозначается как INT0. Конечно же появились прерывания, связанные с новыми аппаратными модулями и событиями сброса.

Новые удобства, естественно, требуют немного больше усилий при управлении прерываниями. Несмотря на новые структуры, внимание было уделено как можно более полной совместимости с управлением прерываниями в предшествующих семействах PIC.

Как и раньше, для каждого прерывания существует персональный разряд разрешения (IE). При значении "1" прерывание разрешено, а при "0" — блокируется (маскируется). Кроме того, для каждого источника прерывания есть индивидуальный флаг (IF), который сигнализирует о наступлении соответствующего события прерывания. **Флаг прерывания устанавливается независимо от того, разрешено это прерывание или нет.** В большинстве случаев флаг должен сбрасываться программно после того, как программное обеспечение приняло к сведению данное событие.

Новым является то, что теперь для каждого прерывания существует разряд приоритета (IP) в регистре IPR:

- 0 — низкий приоритет;
- 1 — высокий приоритет.

Внешнее прерывание INT0 не имеет разряда приоритета — оно всегда имеет высокий приоритет.

С точки зрения совместимости, регулирование приоритетами можно временно оставлять без эффекта. Для этого служит разряд IPEN в регистре RCON. В случае IPEN = 0 приоритеты игнорируются, то есть, все прерывания получают высокий приоритет.

Как и прежде, есть три привилегированных прерывания: TMR0, INT0 и RB. Разряды разрешения и флаги этих прерываний находятся в регистре INTCON, где расположены также оба разряда общего разрешения.

Все разряды и флаги новых внешних прерываний (INT1 и INT2) размещены в новых регистрах INTCON2 и INTCON3. В регистре INTCON2 для каждого внешнего прерывания находится также разряд, задающий вид активного фронта (1 — нарастающий фронт). Разряд выбора активного фронта для INT раньше находился в регистре OPTION, который под этим именем больше не существует.

Остальные прерывания называются "периферийными", их разряды разрешения (IE) находятся в регистре PIE1 или PIE2, а флаги прерываний (IF) — в регистрах PIR1 и PIR2. Разряды приоритетов (IP) находятся в регистрах IPR1 и IPR2. За информацией о точном расположении этих битов следует обращаться к соответствующим техническим описаниям. У каждого представителя PIC18 может быть максимум 16 периферийных прерываний, однако на данный момент уже существует 23 источника прерываний и, несомненно, будут быстро появляться новые.

Регистр INTCON:

7	6	5	4	3	2	1	0
GIE	PEIE	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

Структура регистра INTCON осталась неизмененной, однако разряды GIE и PEIE получили несколько другое значение, если разрешены приоритеты (IPEN = 1). Это объясняет, почему разряд 7 обозначается как GIE/GIEH, а разряд 6 — как PEIE/PIEL.

Переименование могло бы привести к ошибочному предположению, что разряд GIEH в случае с IPEN = 1 отвечает только за прерывания высокого приоритета. Самый старший разряд регистра INTCON (GIE/GIEH) в любом случае служит, как и

раньше, для общего разрешения прерываний. Если этот разряд сброшен, то никакое прерывание невозможно.

Типичная структура прерываний PIC18 показана на рис. 3.2.

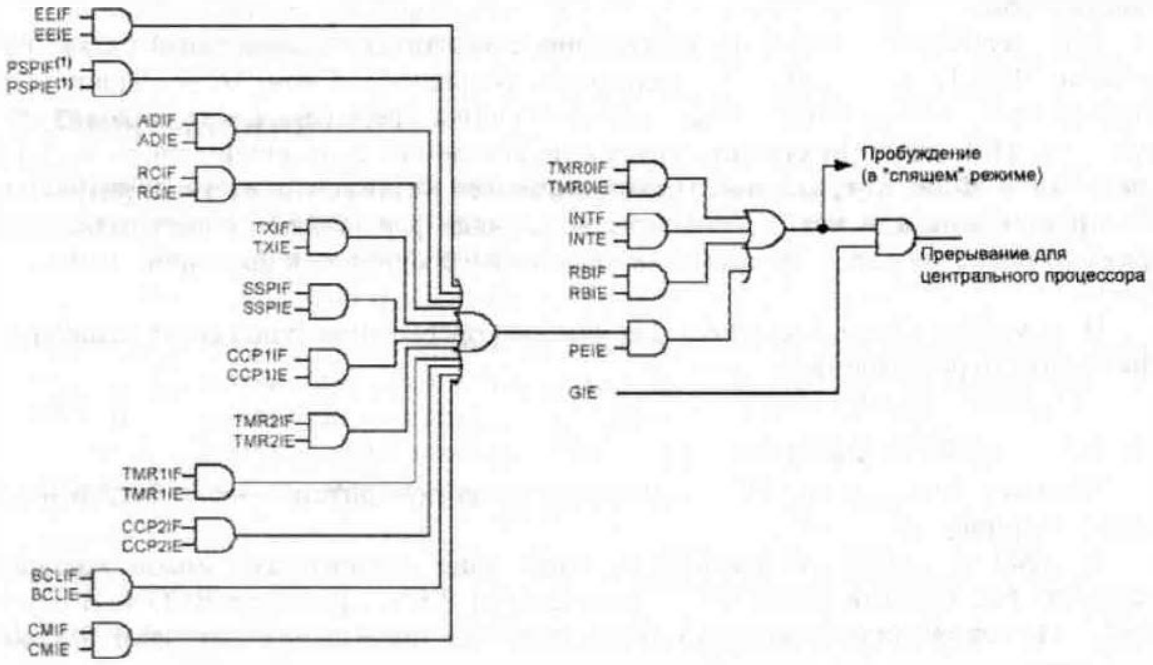


Рис. 3.2. Типичная структура прерываний PIC18

Таким образом, невозможно разрешить прерывания с низким приоритетом, а прерывания высокого приоритета запретить.

Раньше установкой разряда PEIE разрешались "периферийные" прерывания. Это актуально и сейчас, если выключено назначение приоритетов (IPEN = 0). Если же разряд IPEN = 1, то шестой разряд в регистре INTCON служит для разрешения или запрета всех индивидуально разрешенных прерывания с низким приоритетом. По этой причине этот разряд называется PEIE/GIEL.

Если обслуживается прерывание с низким приоритетом, то разряд GIEL сбрасывается, чтобы не могло быть допущено какое-нибудь другое прерывание с низким приоритетом. Но при этом допускается прерывание с высоким приоритетом, то есть, разряд GIE сбрасывается только в том случае, если обслуживается прерывание с высоким приоритетом. Соответствующие разряды опять устанавливаются по команде `RETFIE`.

Для тех, кто запутался в новшествах управления прерываниями в PIC18, представляем следующее обобщение.

- Установить индивидуальные разряды разрешения прерываний.
- Установить IPEN, если допускается приоритетность.
- Установить PEIE/GIEL. Если IPEN = 1, то этот разряд устанавливается для разрешения прерываний с низким приоритетом, если же IPEN = 0, то он устанавливается для разрешения "периферийных" прерываний.
- Если IPEN = 1, то установить разряды приоритета для прерываний с высоким приоритетом, для прерываний же с низким приоритетом — сбросить (INT0 всегда имеет высокий приоритет). Обратить внимание на то, чтобы все подпрограммы обработки прерываний размещались по правильным векторам прерываний!

- Не забывайте, что все параметры сопряженных аппаратных модулей должны правильно устанавливаться в соответствующих конфигурационных регистрах (например, выбор активного фронта, режим, разряды включения таймеров).
- Все задействованные флаги (IF) к началу инициализации прерываний должны быть сброшены, поскольку они могут быть установлены предшествующими событиями прерываний, и установка глобальных и индивидуальных разрядов разрешения сразу же приведет к возникновению соответствующего прерывания.
- Напоследок должен быть установлен разряд GIE/GIEH в регистре INTCON.

3.8. Сброс

У микроконтроллеров PIC18 появились две новые причины сброса: команда RESET и сброс в результате переполнения стека или обращения к пустому стеку. В этих случаях условия сброса такие же, как и в случае сигнала на выводе /MCLR или переполнения сторожевого таймера.

Разряды /TO, /PD, /POR и /BOR, которые раньше были собраны в регистрах STATUS и PCON, теперь находятся вместе в регистре RCON, который также дал пристанище одному разряду-гостю (IPEN).

Уведомление о том, что произошла ошибка, связанная со стеком, можно найти в регистре STKPTR (STKOVF, STKUNF).

3.9. Аппаратные модули

В принципе, обслуживание аппаратных модулей не изменилось. Поскольку в PIC18 присутствует больше регистров специального назначения, существует и больше возможностей для аппаратных модулей. Например, у среднего подсемейства микроконтроллеров PIC вряд ли были бы мыслим модуль CAN.

Также стал богаче сам набор модулей. Например, у некоторых представителей PIC18 есть до пяти таймеров и пяти модулей CCP. Соответственно, некоторые конфигурационные разряды должны интерпретироваться в зависимости от типа микроконтроллера (см., например, CMCON).

3.10. Новые команды

Нередко можно услышать, что во времена PIC18 на ассемблере уже никто не программирует, и потому знание отдельных команд для разработчика излишне.

Судя по изменениям в наборе команд, компания Microchip учла пожелания программистов на языках высокого уровня. Раньше при программировании на языке высокого уровня, в отличие от программ на ассемблере, возникала колоссальная нехватка памяти, однако теперь этого нет.

Если что-то полезно разработчику компилятора с языка высокого уровня, то это в равной мере полезно и инженеру, составляющему микропрограммы для микроконтроллеров PIC на языке ассемблера. Благодаря расширенному набору команд для PIC18 и увеличенному объему памяти, этот процесс стал намного проще.

Компания Microchip проделала немалую работу, подробно описав команды в свой документации, и у нее есть все основания гордиться собой. Конечно, можно программировать на ассемблере, и не зная структуры команд и тесно связанной с этим стратегии центрального процессора, однако мы считаем, что такие познания

при работе с микроконтроллерами PIC не только полезны, но и приводят в восторг при виде новых возможностей увеличенной разрядности команд.

3.10.1. Структура команд

Основная концепция структуры команд в PIC18 осталась такой же как и у поколения PIC16. В большинстве команд код операции занимает старшие шесть разрядов командного слова. К этому часто дополняется один разряд, задающий регистр результата выполнения операции. С помощью шести разрядов можно было бы скомпоновать диапазон из 64 команд, однако все не так просто: некоторые команды должны ограничиваться операционным кодом в четыре разряда, чтобы оставалось достаточно места для аргументов. Таких команд всего четыре. Другие команды, которые гребуют меньше места для аргументов или вообще их не имеют, для уплотнения могут разделять между собой место в пространстве команд. В общем, для PIC18 существует 77 команд — больше, чем удвоенное число.

Если взглянуть на все это с философской точки зрения, то можно было бы во многих случаях поспорить, какая часть команды является операционным кодом, а какая — аргументом. Центральный процессор зачастую расценивает машинный код не так, как это делает программист, глядя на хорошо знакомый ассемблерный код. К этому моменту мы еще вернемся позже, когда увидим, что поддерживаемый Microchip язык ассемблера упаковывает часть кода команды в аргумент (речь идет о косвенной адресации).

3.10.2. Регистр состояния

Регистр STATUS теперь содержит только разряды состояния центрального процессора. Разряды /TO и /PD, которые раньше находились в этом регистре, были перенесены в регистр RCON.

Кроме того, в регистре STATUS появились два новых разряда, которые имеют значение для арифметики со знаком (дополнение до двух):

- OV (разряд 3) — устанавливается, когда при выполнении арифметической операции происходит переполнение в разряде 6 (изменяется состояние разряда 7);
- N (разряд 4) — устанавливаются, если результат арифметической операции отрицательный.

3.10.3. Команды с рабочим регистром в качестве аргумента

Для начала рассмотрим команды, которые выполняют операцию с рабочим регистром, например, `ADDWF` или `IORWF`.

Как и прежде, для выбора операции используются шесть старших разрядов кода команды, после которых следует разряд "d", задающий регистр назначения результата. У 12-тиразрядного слова команды (16C5x) для адреса регистра остается, таким образом, только 5 разрядов, с помошью которых можно напрямую обращаться только к 32-м рабочим регистрам. При разрядности команд 14 для адреса остается 7 разрядов, так что можно обращаться к 128 адресам. **При разрядности команд 16 для адресации остается 9 разрядов.**

Пространство рабочих регистров, которые можно адресовать непосредственно, без предварительного выбора банка, имеет размер 512 байтов, однако память RAM по веским причинам теперь разделена на банки не по 512, а по 256 байтов. Сейчас можно производить доступ в любой момент к двум банкам, а именно: к предвари-

тельно выбранному банку и к однозначно определенному "банку доступа", который не требует выбора.

Предварительный выбор банка происходит с помощью регистра BSR. В наличии может быть до 16 банков.

Банк доступа — это, говоря с физической точки зрения, не банк. Он состоит из двух половинных банков, а именно, одна половина — банка 0, другая половина — банк 15. Первая область (в банке 0) предназначена для тех переменных, которые всегда должны быть в зоне досягаемости. В таких областях нуждаются языки высокого уровня, но они также удобны и для разработчиков на ассемблере.

Вторая область (в банке 15) содержит 128 регистров специального назначения. К слову, в настоящее время первая половина банка 15 не используется — вероятно, она будет позже выделена для новых регистров специального назначения.

Девять разрядов адреса разбивают на две части: восемь разрядов для адресации в пределах банка и один разряд "a", который отвечает за выбор одного из упомянутых банков. Если a = 1, то выбирается банк, заданный через регистр BSR, если же a = 0, то выбирается банк доступа.

Таким образом, группа команд, выполняющих некоторую операцию с рабочим регистром, имеет структуру `OOOOOO D A FFFF FFFF`.

Если, например, регистр BSR содержит значение 2, то могут адресоваться напрямую следующие 512 рабочих регистров (каждая ячейка отведена для 128 байтов):

000	100	200	300	400	500	600	700	800	900	a00	b00	c00	d00	e00	f00
080	180	280	380	480	580	680	780	880	980	a80	b80	c80	d80	e80	f80

Девять разрядов адреса `A FFFF FFFF` указывают однозначно на один из этих рабочих регистров, даже в случае "нелинейности".

Программиста на ассемблере адресация, в общем случае, не должна волновать. Если он в объявлениях присвоил переменной значение из диапазона $0...0FFF_h$, то ассемблер генерирует нелинейный девятиразрядный адрес автоматически.

В руководстве по применению и в технических описаниях для рассматриваемых команд используется следующая форма:

код операции	рабочий регистр,d,a	; a = 0 — банк доступа

При этом у нас поначалу сложилось впечатление, что при пропуске параметра "a" в качестве значения по умолчанию выбирается a = 1, однако это впечатление оказалось ошибочным. В действительности, ассемблер устанавливает параметр a в зависимости от объявления переменных.

Примечание

Как правило, параметр "a" можно опускать. Он необходим только в тех случаях, когда требуется перезаписать объявление.

Мы не нашли в описаниях соглашения об идентификаторе для a = 0 или a = 1, чего не скажешь о параметре "d", для которого заведены идентификаторы "W" и "F". Во всех примерах из технических описаний, в которых применяется параметр "a", используется явный способ написания: "F,0" или "F,1".

3.10.4. Новые арифметические команды

Как и ожидалось, в PIC18 появились новые арифметические команды, несомненно удобные для программирования. И конечно же, они опять "нагружают голову" — особенно когда идет речь о флагах состояния.

Следующие четыре команды изменяют все без исключения пять флагов регистра состояния. Команда COMF, которую раньше должны были использовать для операции NEG, устанавливает только флаги ZR и N.

```
ADDWFC   f,d<,a>      ; Результат := f+w+перенос
SUBWFB   f,d<,a>      ; Результат := f-w-заем (заем = /перенос)
SUBFWB   f,d<,a>      ; Результат := w-f-заем (заем = /перенос)
NEGF     f<,a>        ; Результат := -f
```

Как порой нам не доставало новых команд INCFSNZ и DECFSNZ! Как и старые команды INCFSZ и DECFSZ, они не воздействуют на флаги.

```
INCFSNZ  f,d<,a>      ; Результат := f+1. Пропустить следующую
                      ;  команду, если результат = 0
DECFSNZ  f,d<,a>      ; Результат := f-1, Пропустить следующую
                      ;  команду, если результат = 0
```

Хорошо знакомые команды RLF и RRF переименованы, чтобы отличать их по мнемонике от дополняющих новых команд сдвига. Оба новых сдвига не используют флаг переноса, то есть, они также оставляют неизмененным флаг переноса:

```
RLCF     f<,d,a>      ; Новое обозначение для RLF
                      ;  (сдвиг с учетом переноса)
RLCF     f<,d,a>      ; Новое обозначение для RRF
                      ;  (сдвиг с учетом переноса)
RLNCF    f<,d,a>      ; Сдвиг влево (разряд 7 → разряд 0)
RRNCF    f<,d,a>      ; Сдвиг вправо (разряд 0 → разряд 7)
```

3.10.5. Команды для работы с разрядами

Команды для работы с разрядами относятся к тем командам, в которых для кода операции отведено только четыре разряда. Опция выбора регистра назначения (аргумент d) у этих команд, как и прежде, невозможна, да и в ней нет необходимости.

По сравнению прежними микроконтроллерами PIC, команды по работе с разрядами не изменились (разве что за исключением аргумента a). Также появилась одна новая команда для работы с разрядами: долгожданная команда инвертирования разряда. Этой новой команде мы посвящаем отдельный подраздел, поскольку она нам кажется очень полезной.

3.10.6. Команда инвертирования разряда

Инвертирование разряда — часто используемая операция. Пример — формирование прямоугольного сигнала или переключение флага при изменении режима работы. Мы уже часто "сетовали" на то, что эта команда не была реализована раньше, и вот она появилась. Ее синтаксис:

```
BTG      F,D,A
```

Без такой команды есть два варианта инверсии разряда. Первый:

```
TOGLE   BTFSS file,bit
        GOTO  LOHI
        BCF   file, bit
        GOTO  WEITER
LOHI    BCF   file, bit
WEITER  ...
```

В данном случае эти пять команд не так уж и страшны, поскольку подобные операции встречаются в пределах одной программы не чаще, чем несколько раз.

Есть еще один вариант реализации инверсии, который, на первый взгляд, кажется более элегантным: для соответствующего разряда определяют маску, в которой единица содержится только в позиции подлежащего к инвертированию разряда, а остальные разряды равны нулю. С помощью такой маски инвертирование выполняется двумя командами:

```
TOGLE   MOVLW bitmask
        XORWF file
```

Однако у этого метода есть недостаток: он изменяет регистры W и STATUS. Если, например, в прерывании должен инвертироваться исключительно один разряд, тогда лучше выбрать первый способ с пятью командами, поскольку сохранение и восстановление W и STATUS требует явно больше затрат.

Новая команда BTG не изменяет никаких разрядов состояния.

3.10.7. Команды с косвенной адресацией

Напомним, что команды с косвенной адресацией имеют ту же форму, что и команды с непосредственной адресацией, — причем не только на уровне ассемблера, но и на уровне машинного языка.

Архитектура PIC18 допускает 16 банков регистров (4096 рабочих регистров). Для того чтобы иметь возможность указать на любой из этих рабочих регистров, требуется 12-тиразрядный регистр-указатель. Радостная новость: теперь есть три таких регистра-указателя (FSR0, FSR1, FSR2)!

Ранее команда, с помощью которой выполняли косвенную адресацию, записывалась, например, так:

```
MOVF    INDF,W
```

При этом INDF — регистр, на который указывает регистр FSR. Таким образом, регистр INDF — символический. Изобретение этого регистра — ловкий трюк, чтобы не требовать от пользователя исходного кода дополнительного кода операции для косвенной адресации.

Поскольку существует три регистра FSR, то, естественно, должны существовать и три сопряженных с ними регистра INDF: INDF0, INDF1, INDF2.

Формально, команды с косвенной адресацией выглядят точно так же как и команды с непосредственной адресацией (разумеется, аргумент a в данном случае не имеет смысла).

Но в PIC18 компания Microchip пошла еще дальше: для каждого регистра-указателя существует несколько возможностей косвенной адресации:

- с автоматическим инкрементом FSR;
- с автоматическим декрементом FSR;
- без изменения FSR.

Автоматический инкремент может происходить перед выполнением команды или после ее. В этом случае говорят о преинкременте и постинкременте. При желании, с помощью команды с косвенной адресацией можно даже прибавлять к указателю значение, хранимое в регистре W.

По большому счету, речь идет о пяти различных командах, но для того чтобы не требовать от пользователя для каждой из этих команд собственного ассемблерного кода, соответствующая информация упаковывается под именами символических регистров.

Такие символические регистры называются:

INDF0, POSTINC0, POSTDEC0, PREINC0, PLUSW0;
INDF1, POSTINC1, POSTDEC1, PREINC1, PLUSW1;
INDF2, POSTINC2, POSTDEC2, PREINC2, PLUSW2;

Префикс "POST" означает, что инкрементирование или декрементирование происходит после выполнения операции.

Если заглянуть в техническое описание одного из представителей PIC18 на страницу, где перечислены регистр специального назначения, то можно увидеть, что эти символические регистры как ни в чем ни бывало занесены в список как физические. Во всех соответствующих командах они занимают соответствующую позицию в поле адреса, в то время как код операции — тот же, что и при прямой адресации.

В связи с этим возникает несколько интересных вопросов, которые, хотя и не имеют особого практического значения, дают углубленное понимание рассматриваемого предмета.

Символические регистры действительно находятся в банке доступа и занимают там 15 адресов. Как в таком случае ассемблер представит, например, команду `INCF POSTINC0,F,1`? Адрес регистра POSTINC0 — FEE_h. На его основании ассемблер, не долго думая, получит адрес 1 1110 1110. Как видно на этом примере, ассемблер делает то, что ему говорят: обращается по адресу EE в выбранном в данный момент банке.

Теперь, если регистр BSR содержит, например, значение 5, то он адресует рабочий регистр 5EE. Но если BSR имеет значение F, тогда он адресует рабочий регистр FEE, а это — регистр POSTINC0. Что же делает центральный процессор, если он встречает команду, которая должна инкрементировать регистр POSTINC0 (постинкремент)? Распознает адрес и увеличивает содержимое регистра, на который указывает FSR.

В случае доступа к символическому регистру в результате прямой или косвенной команды с этим регистром просто ничего не происходит, поскольку его физически не существует.

3.10.8. Команды с разрядностью в два слова

Хотя у представителей PIC18, благодаря слову команды, объем памяти, к которой можно обращаться напрямую, в четыре раза увеличился, адресное пространство остается ограниченным. В одном слове команды можно также размещать только один адрес, так что операции с двумя рабочими регистрами всегда утомительны — особенно если оба регистра находятся в разных банках.

В результате, компания Microchip решилась на то, чтобы ввести четыре новых команды по два слова каждая. Но прежде чем мы представим их, сделаем важное замечание в отношении безопасности программы.

Мы довольно часто отмечали преимущества "однословных команд" микроконтроллеров PIC, одно из которых — возможность перепрыгивания через одну команду. В самом деле, кто же откажется от таких команд пропуска как INCFSZ или DCFSZ? Еще одно преимущество: невозможна ситуация, когда из-за недосмотра можно оказаться с помощью программного указателя во "втором слове", которое в любом случае содержит адрес или константу. Если бы это произошло, то возникла бы опасность, что адрес расценивается как команда, а программа начала бы выполнять хаотические переходы.

Для того чтобы сохранить эти преимущества, придумали одну хитрость: все "вторые слова" в своей старшей половине содержат значение "1111" и потому интерпретируются как NOP, если вдруг будут считаны как команда.

"Пропуск" команды с разрядностью в два слова длится, естественно, три командных цикла.

MOVFF

В первую очередь поговорим о команде MOVFF, поскольку мы уже упоминали об обычных командах, выполняющих операции с рабочими регистрами.

Ограничение, с которым приходится мириться и при 16-тиразрядных командных словах, заключается в том, что нельзя обращаться к двум аргументам одной командой. Все операции с двумя аргументами должны проходить через регистр W, и это не вызывает особых затруднений до тех пор, пока они находятся в одном банке. Но если потребуется выполнить две смены банка, тогда на выполнение операции уходит целых шесть команд.

Программист на ассемблере может максимально упростить эту процедуру с помощью множества хитростей и искусного "банкования", однако, те, кто не хотят или не могут позаботиться о вопросах организации переменных, должны всегда исходить из предположения, что переменные находятся в двух разных банках. В лице команда MOVFF мы получаем команду "межбанковской" пересылки двух регистров.

Команда MOVFF передает содержимое одного регистра в другой. При этом обращение к обоим регистрам осуществляется по их 12-тиразрядным адресам, то есть, пространство доступа этой команды — вся область 4096 рабочих регистров.

Первое слово команды содержит код операции 1100 и регистр-источник, а второе слово — регистр назначения и завершающий код 1111 (см. пояснение выше).

Внимание!

С помощью команды MOVFF запрещено загружать регистры PCL, TOSU, TOSH и TOSL (то есть, эти регистры не могут быть указаны в качестве регистра назначения). Также этой командой не могут загружаться регистры, управляющие прерываниями, пока разрешено хотя бы одно прерывание.

Команда MOVFF имеет следующий синтаксис:

```
MOVFF    sourcefile, destfile
```

Все флаги остаются неизменными.

LFSR

Это команда загрузки (Load) в FSR. Теперь регистры FSR 12-тиразрядные, и если бы не было команды LFSR, то загрузка обоих байтов регистра-указателя значениями констант требовала бы четыре команды.

Еще одно преимущество этой команда — отсутствие необходимости разбивать аргументы на старшую и младшую части.

Команда LFSR загружает постоянное значение [0...4095] в один из трех регистров FSR. Кроме 12-ти разрядов для константы, она требует еще два разряда для выбора FSR [0...2].

В ассемблерном коде команда выглядит следующим образом:

```
LFSR    fsri, const        ; const < 4095
```

Все флаги остаются неизмененными.

GOTO

Команда GOTO всегда состоит из двух слов и содержит 20 разрядов для адреса. При этом необходимо иметь в виду, что адреса программ всегда должны быть четными. Адрес, который задается в ассемблерном коде, может принимать значения до 2^{21} (около 2 Мбайт), однако машинный код не интересует младший разрядом и выбирает только разряды с 1 по 20.

Если эту команду в ассемблерном коде вызвать с нечетным адресом, то нечетный разряд будет проигнорирован. Ассемблер может выдавать предупреждение только в том случае, если во время ассемблирования уже известно, что адрес нечетный.

Обозначение adr20 означает, что адрес может иметь значение до 2^{21}, диапазон доступа составляет 2^{20} командных слов, в коде команды зарезервировано 20 разрядов для адреса (четного).

Команда GOTO в ассемблерном коде представлена следующим образом:

```
GOTO    adr20
```

CALL

Команда CALL также состоит из двух слов. Кроме того, есть еще одно новшество, касающееся самой команды. При вызове CALL по желанию в "быстром" регистровом стеке может сохраняться контекст (набор регистров W, STATUS и BSR).

Сохранение контекста выполняется, если указан параметр s = 1. Случай, когда параметр s опущен, равнозначен значению s = 0 (нет сохранения контекста). Восстановление регистров W, STATUS и BSR также может быть выполнено при желании по команде RETURN, поэтому команда RETURN в PIC18 аналогичным образом имеет параметр s. При s = 0 (или если этот параметр опущен) регистры остаются неизмененными, а при s = 1 контекст восстанавливается.

```
CALL    adr20,<s>      ; s=0 (по умолчанию), контекст не сохранен
RETURN  <s>            ; s=0 (по умолчанию), контекст не восстановлен
```

Еще раз напоминаем, что теперь можно выполнять доступ к стеку адресов возврата (до 31 адреса) по командам записи и чтения.

3.10.9. Команды относительного перехода

Совершенно новыми являются команды относительного перехода, у которых аргумент представляет собой не конечный адрес, а (половинную) разность между конечным адресом и адресом вызываемой команды. Программисты не обязаны, как правило, заботится о вычислении этой разницы, поскольку это берет на себя ассемблер или компилятор.

Все команды относительного перехода состоят из одного командного слова, поэтому у них гораздо меньше диапазон доступа, чем у команд абсолютного перехода. Существует восемь команд относительного перехода с восьмиразрядным аргументом. Кроме того, есть команда относительного перехода BRA с 11-тиразрядным аргументом и команда относительного вызова CALL RCALL, которая также обладает 11-тиразрядным аргументом:

```
RCALL    adr11        ; Относительный вызов,
                      ;  возможно сохранение контекста
BRA      adr11        ; Безусловный относительный переход
BZ       adr8
BNZ      adr8
BC       adr8
BNC      adr8
BN       adr8
BNN      adr8
BOV      adr8
BNOV     adr8
```

3.10.10. Новые команды пропуска по условию

Новые команды сравнения и пропуска многими уже были написаны в виде макросов, поскольку сравнение часто приводит к ошибкам.

В целом, существует шесть новых команд, с помощью которых можно перепрыгивать через строку:

```
CPFSEQ   f<,a>        ; Сравнить и пропустить, если f = w
CPFSGT   f<,a>        ; Сравнить и пропустить, если f > w
CPFSLT   f<,a>        ; Сравнить и пропустить, если f < w
TSTSZ    f<,a>        ; Проверить и пропустить, если f = 0
INCFSNZ  f<,d,a>      ; dest := f + 1; пропустить, если dest = 0
DECFSNZ  f<,d,a>      ; dest := f - 1; пропустить, если dest = 0
```

Эти новые команды в дальнейших объяснениях не нуждаются. Особенно полезно в них то, что они не изменяют флаги.

3.10.11. Умножение

Есть две новые команды, выполняющие умножение:

```
MULLW    k            ; PRODH:PRODL := W * k
MULWF    f<,a>        ; PRODH:PRODL := W * f<,a>
```

Обе команды составляют исключение другим командам, выполняющим некоторые операции. Обычно результат находится или в регистре W, или в указанном рабочем регистре. При умножении результат состоит из двух байтов, поэтому он загружается в два регистра произведения PRODH:PRODL.

В остальном эти две команды в особых объяснениях не нуждаются. Первая умножает W на константу k, а вторая — на содержимое регистра f, причем параметр a определяет выбор банка для рабочего регистра.

Ни один флаг в регистре состояния не изменяется. Сохраняет свое содержимое также и регистр W.

Для выполнения обеих команд умножения требуется в каждом случае лишь один командный цикл. В противоположность им, поразрядное программное умножение двух байтов занимает 68 командных циклов (без учета вызова CALL и возврата):

```
BMUL    CLRF    UCNT        ; UCNT := 8
        BSF     UCNT,3
        CLR2    ERGL        ; Переменная для результата
BLO     RRF     ZL
        SKPNC
        ADDWF   ERGH
        RR2     ERGL
        DECFSZ  UCNT
        GOTO    BLO
        RETURN
```

Но на практике операнды — это, зачастую, не только байты, но и значения, состоящие из нескольких байт. В качестве примера, рассмотрим умножение двух "слов" (напомним, что под "словом" мы подразумеваем переменную из двух байтов, если только не отмечается явно, из скольких байтов состоит это слово).

Мы исходим из того, что читатель знаком с основами математики шестнадцатеричных чисел и с соответствующими операциями вычислений. Здесь мы только указываем, что стиль записи ZH:ZL соответствует значению 256 * ZH + ZL.

Результатом умножения двух слов, в общем случае, является значение из четырех байтов, которое мы обозначаем как ZWERGH:ZWERGL:ERGH:ERGL.

ZWERGH:ZWERGL:ERGH:ERGL : =(ZH:ZL) * (NH:NL) =
(256 * ZH + ZL) * (256 * NH + NL)

Вычисление умножения выполняют по правилам действия скобок за четыре шага, об очередности которых можно спорить. Меньше всего команд требуется, если выполняют следующую последовательность.

1. Вычислить ZL* NL и загрузить результат в ERGH:ERGL.
2. Вычислить ZH * NH и загрузить результат в ZWERGH:ZWERGL.
3. Вычислить ZH * NL и прибавить результат к ZWERGH:ZWERGL:ERGH.
4. Вычислить ZL * NH и прибавить результат к ZWERGH:ZWERGL:ERGH.

Соответствующую программу обозначим как HWMUL, причем префикс "H" указывает на использование аппаратного умножения.

```
HWMUL   MOVF    ZL,W
        MULWF   NL
        MOV2    ERGL,PRODL      ; Макрос MOV2 (4 команды)
        MOVF    ZH,W
        MULWF   NH
        MOV2    ZWERGL,PRODL    ; Макрос MOV2 (4 команды)
        MOVF    ZH,W
        MULWF   NL
        ADD2    ERGH,PRODL      ; Макрос ADD2 (7 команд)
        MOVF    ZL,W
        MULWF   NH
```

```
        ADD2   ERGH,PRODL          ; Макрос ADD2 (7 команд)
        RETURN
```

Эта программа, все-таки, дает в результате 32 команды, не считая CALL и RETURN. Однако поразрядное умножение двух слов (WMNL) требует в девять раз больше командных циклов. Разумеется, при этом следует отметить, что поразрядное умножение требует меньше места в программе, поскольку выполняется в цикле.

```
WMUL    CLRF   UCNT          ; UCNT := 16
        BSF    UCNT,4
        CLR4   ERGL          ; Макрос CLR4 (4 команды)
WLO     RR2    ZL            ; Макрос RR2 (2 команды)
        SKPC
        GOTO   WLO
        ADD2   ZWERGL        ; Макрос ADD2 (7 команд)
        RR4    ERGL          ; Макрос RR4 (4 команды)
        DECFSZ UCNT
        GOTO   WLO
        RETURN
```

3.11. Совместимость

Слово "совместимость" — заклинание, вызывающее иллюзию, что сложную задачу можно описать с помощью абстракций таким образом, чтобы она была полностью независимой от архитектуры и других свойств аппаратного обеспечения выбранного микроконтроллера.

Хотя мы и находим интересной идею рассматривать PIC как подмножество виртуальных микроконтроллеров, но, полагаем, мы еще очень далеки от реализации подобного видения. Аппаратные элементы развиваются семимильными шагами (чего стоит, например, одно только управление питанием). Может быть однажды станет возможно описать все эти детали совместимым формальным языком, но только не в ближайшие годы.

Говоря о совместимости, мы, естественно, предполагаем, что новый микроконтроллер обладает необходимыми компонентами аппаратного обеспечения. В этой связи речь идет об аппаратной совместимости.

3.11.1. Аппаратная совместимость

Если нет желания переделывать аппаратную часть, тогда аппаратная совместимость означает, в первую очередь, совместимость по выводам. К счастью, на недостаток внимания к совместимости по выводам у разных типов микроконтроллеров PIC жаловаться не приходится, и PIC18 — не исключение.

Необходимой предпосылкой для перехода на другой микроконтроллер является то, что новый тип обладает требуемыми модулями аппаратного обеспечения. При этом следует обращать внимание также на свойства осциллятора, на процесс сброса и потребление тока. **Это справедливо также в том случае, когда переносят проект, разработанный для одного микроконтроллера PIC среднего подсемейства, на другой микроконтроллер PIC среднего подсемейства.**

3.11.2. Совместимость ассемблера

Мы говорим о полной совместимости ассемблера, если для идентичной функциональности в исходном файле должна быть изменена только "шапка", к которой также относится директива include, подключающая заголовочный файл.

Микроконтроллеры PIC базовой серии и среднего подсемейства совместимы между собой по ассемблеру в значительной мере. Могут быть только некоторые различия у аппаратных модулей (например, в разрядах конфигурации) или в случае использования десятиразрядного АЦП вместо восьмиразрядного.

Но как выглядит совместимость ассемблера при переходе от среднего подсемейства PIC к PIC18? Могут ли вообще два микроконтроллера с отличиями в наборах команд, организации памяти и аппаратных модулях выполнять одинаковые функции идентично?

Если речь идет только о результатах арифметических и логических операций, то, теоретически, это — не проблема. Однако на практике этого зачастую недостаточно. Для того чтобы говорить об одинаковом функционировании, должны также с достаточной точностью совпадать временные параметры процессов.

В дальнейшем попытаемся не затрагивать аспекты, касающиеся отличий в скорости работы центрального процессора.

Поставим перед собой следующий вопрос: "Что произойдет, если испытанную программу, написанную для среднего подсемейства PIC, отдать на трансляцию ассемблеру PIC18, то есть, изменить только "шапку" ?". Прежде чем ответить на этот вопрос, сделаем теоретическое отступление и выясним, какие преобразования потребует наша программа.

Адреса программы

Поскольку счетчик команд PIC18 считает байты, должен быть по-новому продуман прежний счетчик слов программы.

Обычно об адресах программы заботиться не за чем, поскольку ассемблер сам их вычисляет, если целевые адреса обозначают метками, но иногда используют и прямой доступ к счетчику команд (например в подпрограммах работы с таблицами). При этом необходимо учитывать, что адреса слов программ теперь четные. При вычисляемых переходах, кроме всего прочего, необходимо учитывать, что команда GOTO имеет длину в два программных слова.

Должны быть изменены строки с директивами **ORG**, которые сообщают ассемблеру, с какого адреса программы должны располагаться последующие команды.

Чтение табличных значений в процессе модернизации, разумеется, следует привести к форме TABLRD. При этом необходимо отыскать в исходном файле прежние прямые обращения к счетчику команд и выполнить соответствующую корректировку.

Внимание! Двухсловные команды!

Хотя хорошо знакомые команды GOTO и CALL в семействе PIC18 выполняют прежние функции, они имеют длину два слова. По этой причине изменяется длина программ, и существует вероятность выхода за границу доступной памяти, чего нельзя оставлять без внимания. Кроме того, для этих команд изменилось время выполнения, что также должно быть учтено при рассмотрении критических ко времени частей программы. В отдельных случаях команды GOTO и CALL лучше заменить командами относительного перехода BRA и RCALL.

Переключение банков

Почти каждый разработчик на ассемблере заложил в свои библиотеки макросы для переключения банков, прежде, чем была введен долгожданный макрос ассемблера BANKSEL.

Когда программу для PIC16 переносят на микроконтроллер семейства PIC18, то следует немало потрудиться на тем, чтобы по-новому организовать структуру переменных. Переключения банков для такой программы больше не будут играть особой роли. PIC18, кроме регистров специального назначения, предоставляет в распоряжение 128 байтов в банке доступа и 256 байтов в выбранном банке.

Таким образом, макросы для переключения банков избыточны, но они, как правило, к ошибкам не приводят, поскольку соответствующие разряды регистра состояния, которые применялись для этой цели у PIC16, теперь просто не задействованы (во всяком случае, в настоящее время). Если к этим разрядам обратиться по их символическим именам (RP0, RP1), то ассемблер для PIC18 выдаст сообщение об ошибке, поскольку он эти разряды не распознает.

Лучшее решение — удалить ненужные вызовы макросов для переключения банков или определить эти макросы как пустые.

Косвенная адресация

Косвенная адресация реализована с помощью регистра-указателя FSR. У семейства PIC18 новым является то, что теперь существует три таких регистра, а их разрядность — 12 бит. Если указать косвенно на адрес больше 256 (банк 2 и банк 3), то в случае PIC16 должен быть установлен девятый бит IRP в регистре STATUS.

При переносе на PIC18 в целях безопасности следует переработать все фрагменты, связанные с косвенной адресацией — **особенно, если была изменена структура переменных**. Это означает, что необходимо пересмотреть все доступы к FSR, IRP и INDF. Недостаточно заменить регистры FSR и INDF на FSR0L и INDF0 — следует также позаботиться о FSR0H. Команду `INCF FSR` обходят использованием автоматического инкремента.

Внимание! INCF и DECF!

Обращаем внимание на одну отличительную особенность PIC18, которая на первый взгляд совсем незаметна: раньше команды INCF и DECF изменяли только флаг ZR, но теперь изменяются все доступные флаги регистра состояния — даже CY и DC. Хотя в технических описаниях и попадаются примеры, отображающие новое положение вещей, тем не менее, четко эта перемена нигде не указана.

Большинство разработчиков используют флаги из регистра STATUS только после выполнения изменяющей их команды, однако в некоторых случаях полагаются на то, что флаг в регистре STATUS остается неизменным даже после выполнения нескольких команд.

Регистры специального назначения

Некоторые разряды регистров специального назначения были перемещены. В большинстве случаев распределение стало более последовательным и логичным. Например, флаги, которые после сброса содержат информацию об источнике сброса, подаче питания или завершении счета сторожевого таймера, перенесены из регистра STATUS в новый регистр RCON. Если бы макроассемблер MPASM не требовал, чтобы названия разрядов обозначались полным адресом (`регистр, разряд`), то никаких проблем, связанных с совместимостью ассемблера, не возникало бы. Ас-

семблер для PIC18 не может действовать самостоятельно, если он, например, встречает бессмысленное выражение STATUS, NOT_PD. Он даже не выдает никакого предупреждения, хотя в этой ситуации должен был бы это сделать.

Мы уже неоднократно сожалели о том, что ассемблером не поддерживаются простые (определенные с помощью #define) названия разрядов регистров специального назначения. Оставив в стороне вопрос совместимости, это заставляет при постоянно растущем числе разрядов искать в технических паспортах, в каком регистре специального назначения они находятся, не говоря уже об вытекающих из этого ошибках.

Особенно это касается флагов прерываний, которые в будущем, вероятно, не будут иметь определенной позиции в регистрах PIR, PIE и IRP. Разработчиков обычно не интересует, в каком из этих регистров находится, например, флаг CCPIF или TXIF, а работу по выискиванию подобной информации творческой никак не назовешь.

Названия **OPTION_REG** в PIC18 больше нет. Большинство разрядов этого регистра перенесено в регистр под названием **T0CON**. Разряды NOT_RBPU и INTEDG, которые в PIC16 находятся в регистре OPTION_REG, в PIC18 располагаются в регистре **INTCON2**. При этом, самой собой, разряд INTEDG переименован в INTEDG0.

Обращаем также внимание на то, что разряды PS0, PS1 и PS2 больше не используются для установки постделителя **сторожевого таймера** (это осуществляется с помощью регистра CONFIG0!). В результате эти разряды были переименованы в T0PS0, T0PS1 и T0PS2. Разряд PSA теперь служит просто для задания коэффициента деления для Timer0 равным 1.

В также PIC18 отсутствует название **PCON**. Теперь этот регистр называется **RCON**. По логике вещей, в регистре RCON находятся также разряды NOT_TO и NOT_PD, которые до этого находились в регистре **STATUS**.

Регистр таймера **TMR0** мы можем смело заменить на **TMR0L** (значение TMR0H не имеет смысла, поскольку прерывание по переполнению используется только в исключительных случаях). Кроме того, нет нужды беспокоиться о прерывании по переполненню, если не изменяется состояние разряда T08BIT в регистре T0CON.

Регистр **SSPCON** переименован на **SSPCON1**. Регистр SSPCON содержит те же разряды, что и SSPCON1, но был переименован, поскольку появился регистр SSPCON2. Однако регистр SSPCON2 есть также и у PIC16, поэтому его можно изменить директивой #define без дальнейшей проверки.

Прерывания

Если применяют прерывания, то необходимо обращать внимание только на то, чтобы подпрограммы обработки прерываний располагались по программному адресу 8.

Если разрешены одновременно несколько прерываний, тогда необходимо проверить, можно ли воспользоваться новым управлением приоритетами. Если управление приоритетами отключено (IPEN = 0), тогда все прерывания имеют высокий приоритет, то есть, можно экономить на процедурах "PUSH" и "POP".

В любом случае, следует удостоверяться, что флаги при прерываниях от периферии также находятся в привычных регистрах PIR и PIE.

3.11.3. Что слышно от MPASM18?

Проведем эксперимент с одним из наших последних файлов для PIC16. Проект состоит из 1655 командных слов. Поначалу мы притворимся совершенно несведущими и просто изменим "шапку" файла: для директив `include` и `list` мы запишем "18F452" вместо "16F877". Скопируем для нового проекта ассемблерный файл и дадим ему новое имя. Теперь выполняем ассемблирование и созерцаем сообщения об ошибках.

Свыше 600 сообщений об ошибках! Вот уж, не ожидали. Ассемблер не распознает символ `STATUS` — обычно, это говорит о том, что он не находит включаемый файл. Но подобное сообщение отсутствует. В чем же дело?

Представляем решение загадки. Ассемблер действительно не смог найти включаемый файл для PIC18, но сообщение об этом не появилось по той причине, что список ошибок ассемблирования состоял, например, из 1000 строк. Но максимальное число строк, которым в состоянии управлять окно результата, — всего лишь 600. Таким образом, начало списка ошибок было "обрезано". Такая вот неприятность. Теперь понятно, почему мы не узнали, что наши строки конфигурации не были поняты ассемблером для PIC18.

После того как была устранена ошибка во включаемом файле, мы получили, как и ожидалось, предполагаемые сообщения об ошибках. Ассемблер для PIC18 не знает имен OPTION_REG, PCON, SSPCON.

Кроме того, были сообщения об ошибках, которые мы не ожидали: переименованные команды **`RLF`** и **`RRF`**, которые теперь называются `RLCF` и `RRCF`, не воспринимаются.

Сокращенные формы **`SKPZ`** (вместо `BTFSS STATUS,Z`), **`SKPC`**, `CLRC` и т. д., к сожалению, ассемблером для PIC18 не поддерживаются. Мы предпочитали эти способы записи уже долгое время по причине из наглядности и думали, что имеем право на подобную привычку. Впрочем, такие сокращения в случае PIC18 используются нечасто, поскольку для условных переходов теперь есть собственные команды.

Обозначения `C` для переноса и `Z` для нулевого результата также не проходят, и мы их всегда переопределяем как `CY` и `ZR`.

Как и следовало ожидать, не распознаются имена **`FSR`** и **`INDF`**. Просто переименовать их в FSR0L и INDF0 — нерационально. В нашем примере использовалась косвенная адресация для обнуления всех свободных регистров RAM после сброса, то есть, по адресам от 20_h до $7F_h$ и от $0A0_h$ до $0FF_h$. Эти небольшие циклы переписываем по-новому. Обнуляем банк доступа и банк 1. При этом, разумеется, используем автоинкрементирование регистра FSR0.

Семейством PIC18 не была унаследована одна команда, поскольку она избыточна. Речь идет о команде **`CLRW`**, которая была заменена командой `CLRF WREG`. Но `CLRW` не преобразуется ассемблером в `CLRF WREG`. Этого мы, в общем-то, ожидали, поскольку подобное происходило и при упразднении команд `OPTION` и `TRIS`. Вместо этого, в поставляемом включаемом файле есть `define`-определение, которое преобразует текст `CLRW` в `CLRF WREG`. При этом существует один маленький подвох: присутствуют два таких определения, одно из которых состоит из прописных букв, а второе — из строчных. Если теперь отключить опцию чувствительности к регистру, как это мы привыкли делать, то получится двойное определение, и ассемблер выдаст сообщение об ошибке.

Еще возникла проблема с одним разрядом, который мы назвали **`FREE`**. Однако название `FREE` относится также к регистру EECON1. В этом случае мы, естественно, отступили и переименовали наш разряд в `FREI`.

Итак, вначале напишем небольшой файл (c16to18.inc), который позже будет еще дополнен. Этот файл мы подключаем с помощью директивы include, однако в него будут записаны только те определения, которые можно переименовать без дальнейшей проверки.

Пока что он содержит следующие строки:

```
#DEFINE  RLF                    RLCF
#DEFINE  RRF                    RRCF
#DEFINE  TMR0                   TMR0L
#DEFINE  SSPCON                 SSPCON1
#DEFINE  RCON, NOT_TO           PCON, NOT_TO
#DEFINE  RCON, NOT_PD           PCON, NOT_PD
#DEFINE  OPTION_REG, NOT_rbpu   INTCON2, NOT_RBPU
#DEFINE  OPTION_REG, INTEDG     INTCON2, INTEDG0
#DEFINE  OPTION_REG, T0CS       T0CON, T0CS
#DEFINE  OPTION_REG, T0SE       T0CON, T0SE
#DEFINE  OPTION_REG, PSA        T0CON, PSA
#DEFINE  OPTION_REG, PS0        T0CON, T0PS0
#DEFINE  OPTION_REG, PS1        T0CON, T0PS1
#DEFINE  OPTION_REG, PS2        T0CON, T0PS2
```

Кроме того, наш персональный стиль требует следующих определений:

```
#DEFINE  CY    C
#DEFINE  ZR    Z
#DEFINE  SETC  BSF   STATUS,CY
#DEFINE  CLRC  BCF   STATUS,CY
#DEFINE  SETZ  BSF   STATUS,ZR
#DEFINE  CLRZ  BCF   STATUS,ZR
#DEFINE  SKPC  BTFSS STATUS,CY
#DEFINE  SKPNC BTFSC STATUS,CY
#DEFINE  SKPZ  BTFSS STATUS,ZR
#DEFINE  SKPNZ BTFSC STATUS,ZR
```

Если кто-то раньше пользовался короткими, состоящими из двух строк макросами (BZ, BC и т.д.), то он должен их перепроверить. Теперь это — команды с ограниченным диапазоном доступа.

Затронем также макросы BANK0, BANK1, BANK2 и BANK3, с помощью которых мы выполняем переключение банков. В микроконтроллере PIC 16F877 присутствует четыре банка по 96 рабочих регистров в каждом. Это объем RAM в точности совпадает с тем, который PIC18 может адресовать в банке доступа и банке 1 без переключения банков. Таким образом, от вышеупомянутых макросов можно полностью отказаться, однако, поскольку нас интересует размер новой программы, мы пока что заменяем вызовы этих макросов командами NOP.

Кроме того, заменяем строки с CONFIG блоком CONFIG, который мы записали в нашу библиотеку в качестве заготовки конфигурации.

Теперь опять ассемблируем файл. На этот раз ассемблер об ошибках не сообщает.

Следующий этап — проверка длины транслированного файла (только из любопытства). Поначалу мы несколько удивлены, поскольку теперь у нас 1895 командных слов (длина программы для PIC16 составляла только 1655 слов). Это на 14,5% больше! В результате беглого просмотра исходного файла обнаруживаем, что этому

процентному приблизительно соответствует доля команд CALL и GOTO. Увеличение длины обусловлено тем, что CALL и GOTO состоят из двух слов. После удаления избыточных переключений банков остается всего 1775 слов.

Теперь необходимо просмотреть весь файл, строка за строкой. У нас есть хорошая привычка: при "большой работе" сначала нарисовать качественную картину реальных затрат труда. На работу над физической частью проекта у нас ушла неделя. Был тщательно спроектирован обмен данными по шине CAN. Большого количества тестов потребовали задачи измерения и управления, поскольку они должны быть реализованы с высокой временной точностью. Постановка задачи вновь и вновь модифицировалась и расширялась. В целом, накопилось несколько человеко-недель. Исходный файл содержит около 1200 командных строк (благодаря макросам, их стало явно меньше 1655!). Из них большая часть приходится на библиотечные функции и таблицы. Библиотечные функции уже были переработаны.

Сам процесс "написания программы", как обычно, потребовал наиболее незначительных затрат времени во всем проекте. По нашим оценкам, на просмотр программы, внесение необходимых изменений и реорганизацию переменных должно было уйти от трех до четырех часов — вполне реалистично.

Еще некоторое время мы потратили на оптимизацию нового исходного файла. Поскольку речь идет о важном и долгосрочном проекте, мы, например, ввели табличное чтение по команде TABLRD, а также функции автоинкрементирования при косвенной адресации.

Некоторые команды GOTO были заменены командами BRA, а от команд относительного перехода вообще можно отказаться. Кроме того, мы не нуждаемся в команде MOVFF, поскольку все переменные размещены или в выбранном банке, или в банке доступа.

Последний этап — тестирование проекта с PIC18. Поскольку мы были достаточно усердны и терпеливы, теперь отпадает необходимость поиска каких-либо ошибок.

3.11.4. Итог

Наш (предварительный) перечень пунктов для переноса программы с PIC16 на PIC18 выглядит следующим образом.

Общие подготовительные работы:

- составить файл `c16to18.inc`, как описано выше; содержимое этого файла отчасти зависит от привычек разработчика;
- составить стандартный блок CONFIG, который применяют как заготовку для будущих конфигураций;
- вставить и откорректировать файл `c16to18.inc` и конфигурацию;
- переработать определения переменных.

При обработке программы следует обращать внимание на следующее:

- адаптировать доступ к программному счетчику и директиве ORG;
- переработать косвенную адресацию (доступ к FSR, IRP и INDF);
- переработать, при необходимости, обслуживание прерываний;

Для оптимизации мы можем, при известных условиях (в критических по времени частях программы), заменить команды GOTO командами BRA. Например:

```
SKPZ
GOTO      LABLE
```

можно преобразовать в

```
BNZ     LABLE
```

Но при этом в любом случае следует проверять, находится ли адрес назначения в области досягаемости для команды BNZ!

3.11.5. Совместимость "сверху вниз"

Выражение "совместимость сверху вниз" звучит немного странно. Кто же захочет перейти с более нового микроконтроллера к более старому? Но такое очень часто происходит, когда, например, хотят вывести на рынок успешный проект в урезанном варианте. Кроме того, существуют даже микроконтроллеры PIC базовой серии, более новые, чем некоторые PIC18.

В таком случае ожидать совместимости, разумеется, не приходится. Однако зададимся вопросом: можно ли программу, написанную на ассемблере для PIC18, переписать без лишних затрат для PIC16?

При переходе от микроконтроллера типа PIC18 к микроконтроллеру типа PIC16 должны быть выполнены изменения, соответствующие рассмотренным выше, но только в обратном порядке.

Кроме того, необходимо позаботиться о новых командах PIC18, которых ассемблер для PIC16, разумеется, не поддерживает. Каждую из этих команд можно заменять макросом. Так, для команды MOVFF макрос может выглядеть следующим образом:

```
MOVFF   MACRO   SOURCE,DEST
        BANKSEL SOURCE          ; Макрос переключения банков
        MOVF    SOURCE,w
        BANKSEL DEST            ; Макрос переключения банков
        MOVWF   DEST
        ENDM
```

Внимание!

Фактически, макрос MOVFF срабатывает не совсем так же как команда MOVFF для PIC18. Макрос не оставляет неизменными регистр W и флаг ZR!

Нам бросилось в глаза, что во многих примерах от Microchip команда

```
MOVF    WERT, W
```

преобразуется в команду

```
MOVFF   WERT, WREG
```

Если исходный файл для PIC18 написан ясно и наглядно и в нем используется до тысячи команд, то его преобразование в файл для микроконтроллеров среднего подсемейства — не такая уж и сложная задача.

ГЛАВА

4 Управление питанием

С давних времен потери мощности привлекают самое пристальное внимание — и не только в тех случаях, когда речь идет о приборах с питанием от батарей. Дело в том, что напрасное рассеивание мощности создает дополнительные проблемы, наподобие чрезмерного тепловыделения.

Один только "древний" привод ворот нашего гаража в дежурном режиме потребляет 15 Вт. И спрашивается: ради чего? Поскольку на сегодняшний день такие потери недопустимы, то должна быть оптимизирована каждая часть электрической схемы: блок питания и аналоговые или высокочастотные части прибора.

Даже микроконтроллер вносит свою лепту в потери мощности, поэтому при разработке микроконтроллеров все больше уделяется внимание тому, максимально уменьшить потребление тока.

Компания Microchip называет такие стратегии **нановаттной технологией**, которая подразумевает, что физический процесс работы осцилляторов изменился настолько, что потребление тока уменьшилось на 80% (обычно от 1 до 2 мкА). Выражение "нановаттный" относится, скорее, к дежурному режиму, при котором потребляемая мощность может сокращаться до нановатт.

Но это еще не все... Новые микроконтроллеры PIC поддерживают управление питанием, с помощью которого могут быть выбраны экономичные режимы со сменой активного осциллятора "на лету".

Следующее описание мы взяли из спецификации микроконтроллера **PIC18F1220/1320**, в котором реализованы предельные (на момент написания книги) возможности управления питанием.

Информацию о том, в каком объеме управление питанием реализовано в других типах микроконтроллеров PIC, следует искать в соответствующих технических описаниях. В руководстве Product Selector Guide соответствующие представители помечены как "nW" (в колонке "other features" — "дополнительные свойства"). Впрочем, об объеме возможностей это ничего не говорит.

4.1. Режимы работы

В старых микроконтроллерах PIC применялись два режима работы: обычный и "спящий". Сейчас существует третий режим, в котором центральный процессор не тактируется, но периферийные устройства продолжают работать. Этот режим работы называется состоянием ожидания (idle).

Таким образом, новая стратегия управления питанием поддерживает три режима работы:

- RUN — обычный режим, в котором активны как центральный процессор, так и периферийные устройства;

- IDLE — состояние, в котором центральный процессор "спит", но аппаратные модули тактируются и работают, если только не выключены программно;
- SLEEP — знакомое нам состояние, в котором центральный процессор, а также большинство аппаратных модулей неактивны.

Если в режиме "SLEEP" включен сторожевой таймер, тогда работает внутренний RC-осциллятор. Осциллятор таймера Timer1 может оставаться включенным для того, чтобы, например, поддерживать подачу тактовых импульсов с частотой 32 кГц для функций реального времени. Если должен работать АЦП, то в качестве источника тактовых импульсов выбирается AD-RC-осциллятор, и тогда преобразование может происходить и в режиме "SLEEP". По окончанию преобразования устанавливается флаг прерывания, что выводит микроконтроллер PIC из "спящего" режима.

4.2. Классы осцилляторов

Для того чтобы еще больше уменьшить потребление мощности, схема управления питанием позволяет переключать источники тактовых импульсов прямо во время работы. При этом существует три класса осцилляторов (рис. 4.1):

- Первичный, который зарегистрирован в конфигурационном регистре 1H. Это или внешний осциллятор, который строится на базе выводов OSC1 и OSC2 (LP, XT, HS, HSPLL, RC), или внутренний RC-осциллятор (INTRC).
- Вторичный, который физически реализован на основе тактового сигнала Timer1.
- Внутренний блок RC-осциллятора (INTRC).

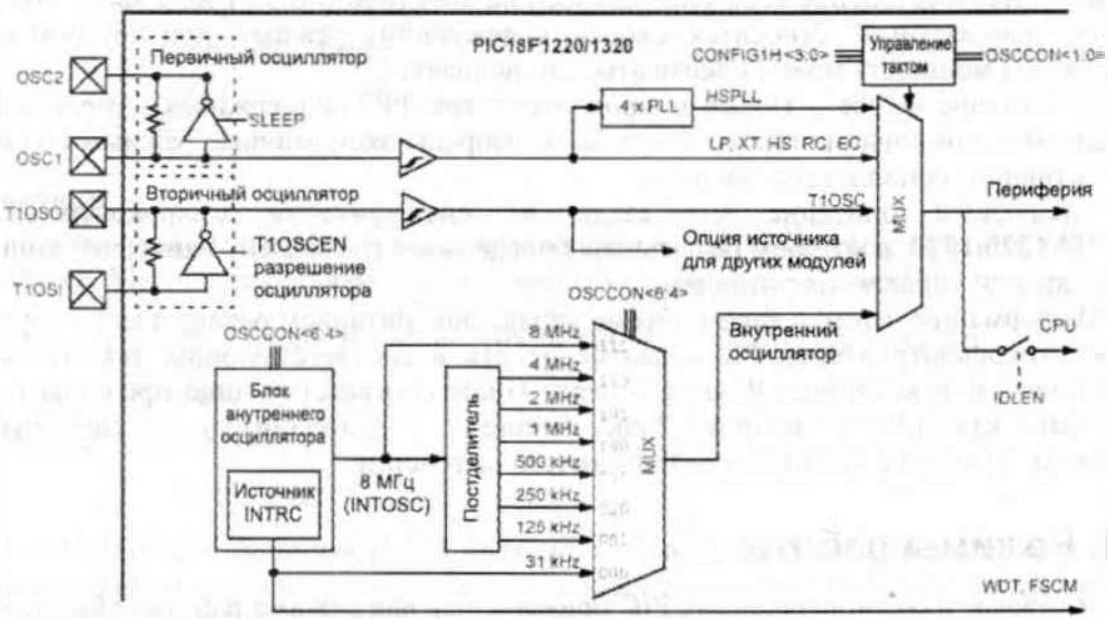


Рис. 4.1. Формирование тактовых импульсов в микроконтроллере 18F1220

Эти три класса осцилляторов, по сути, идентичны трем физическим типам осцилляторов: внешнему осциллятору между OSC1 и OSC2, осциллятору Timer1 и внутреннему блоку RC-осциллятора. Однако последний может служить или в роли основного, или только как дополнительный осциллятор.

При этом основной осциллятор играет привилегированную роль: он формирует импульсы системной синхронизации после каждого сброса. Таким образом, после сброса микроконтроллер PIC находится в обычном режиме "RUN" и тактируется от основного осциллятора.

Это основное состояние обозначается как "primary run mode" (сокращенно PRI_RUN). Все остальные возможные комбинации рабочих состояний и классов осцилляторов называются **режимами управления питанием**.

4.3. Регистр OSCCON

OSCCON — регистр управления питанием:

7	6	5	4	3	2	1	0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0

С помощью разрядов IDLEN:SCS1:SCS0 выбирают режим управления питанием в режиме "RUN" (см. следующий раздел).

Разряды IRCF2:IRCF1:IRCF0 определяют частоту внутреннего осциллятора, а разряды OSTS и IOFS служат для отображения состояния основного и, соответственно, внутреннего осциллятора. Точная функция, а также имена этих разрядов у других типов микроконтроллеров PIC могут отличаться от указанных.

Также обращаем внимание на разряд T1RUN в регистре T1CON, который показывает, работает ли осциллятор Timer1, формируя тактовые импульсы.

4.4. Режимы управления питанием

Всего существует шесть режимов управления питанием, которые в технических описаниях часто обозначаются с помощью сокращений:

- SEC_RUN — вспомогательный режим "RUN";
- RC_RUN — режим "RC-RUN";
- PRI_IDLE — основной режим "IDLE";
- SEC_IDLE — вспомогательный режим "IDLE";
- RC_IDLE — режим "RC-IDLE";
- SLEEP — режим "SLEEP".

К слову, вспомогательный режим "RUN" совместим с возможностью переключения такта в других микроконтроллерах PIC. Поскольку осциллятор Timer1 обычно работает на частоте 32,765 кГц, то существует возможность, как и прежде, работать с высокоточной синхронизацией, но при этом уменьшить потребление тока.

Разумеется, существует только один режим "SLEEP", поскольку в "спящем" состоянии класс осциллятора не играет никакой роли.

В этой связи перед пользователем возникают следующие вопросы.

- Что нужно сделать для того, чтобы перейти в режим управления питанием?
- Каким образом выйти из него?
- Что именно происходит при смене режима (особенно, если меняется осциллятор)?

С технической точки зрения, переход из одного режима в другой — очень хитроумный процесс. Пользователя при этом, главным образом, интересует, сколько

времени проходит, прежде чем новый режим будет стабилизирован, и что происходит во время перехода.

4.5. Смена режима в состоянии "RUN"

Режимы "RUN" — единственные, в которых переход можно выполнить с помощью команды. Для того чтобы запустить режим управления питанием, используется команда SLEEP, у которой была расширена ее базовая функция.

4.5.1. Команда SLEEP

Команда SLEEP дает ход не только "спящему" режиму, но и всем другим режимам управления питанием. При этом разряды IDLEN и SCS1:SCS0 регистра OSCCON определяют, какой именно режим управлением питанием должен быть запущен. Если все три разряда равны нулю, то команда SLEEP выполняет свою классическую функцию: переход в "спящее" состояние.

После сброса эти три разряда регистра OSCCON обнуляются. Если их никогда не изменять, то команда SLEEP совместима с микроконтроллерами PIC, не обладающими возможностью управления питанием.

Если установлен разряд IDLEN, тогда команда SLEEP переводит в один из трех режимов "IDLE" — в зависимости от того, какие из разрядов SCS1:SCS0 установлены.

В табл. 4.1 показано, как должны быть установлены разряды IDLEN:SCS1:SCS0 в регистре OSCCON, чтобы с помощью команды SLEEP перейти в определенный режим управления питанием.

Таблица 4.1. Установка разрядов регистра OSCCON для перехода в один из режимов управления питанием по команде SLEEP

IDLEN	SCS1	SCS0	Режим
0	0	0	SLEEP
0	0	1	SEC-RUN
0	1	x	RC-RUN
1	0	0	PRI-IDLE
1	0	1	SEC-IDLE
1	1	x	RC-IDLE

Внимание!

Изменение разрядов IDLEN:SCS1:SCS0 имеет смысл только в связи с командой SLEEP. В противоположность этому, изменение разрядов регистра OSCCON IRCF2:IRCF1:IRCF0, с помощью которых выбирается частота внутреннего RC-осциллятора, дает мгновенный эффект. Если переключать эти разряды по отдельности, когда питающее напряжение находится ниже уровня 3 В, то следует проследить, чтобы полученная временная частота поддерживалась напряжением питания.

Если разряд T1OSCEN (в регистре T1CON) ко времени поступления команды SLEEP не установлен, то запрос на включение вспомогательного режима "RUN" или режима "IDLE" отклоняется. Но даже если этот разряд установлен, но Timer1 работает нестабильно, то может произойти непредсказуемое! Для уведомления о том, что тактовые импульсы формируются осциллятором Timer1, устанавливается разряд T1RUN (в регистре T1CON).

4.5.2. Возвращение в основной режим "RUN"

После сброса микроконтроллер PIC находится в режиме "PRI-RUN" и может оказаться в одном из шести режимов управления питанием только по команде SLEEP. Если микроконтроллер находится в одном из режимов "RUN", то он может покинуть его также по команде SLEEP, хотя переходит при этом только в один из других режимов управления питанием.

Как видно из табл. 4.1, возврат в режим "PRI-RUN" с помощью команды SLEEP невозможен. Из режима управления питанием опять попадают в основной режим "RUN" или после сброса, или в результате прерывания. Для этого должен быть установлен соответствующий разряд разрешения прерывания в регистре INTCON или, соответственно, в регистре PIE. Однако разряд общего разрешения прерываний (GIE) должен быть установлен только тогда, когда должны фактически отработать соответствующие обработчики прерываний.

Переключение из вспомогательного режима "RUN" в основной инициируется обнулением разряда T1OSCEN. После этого осциллятор Timer1 работает до тех пор, пока не активизируется основное тактирование. Затем T1RUN сбрасывается и осциллятор Timer1 отключается.

В технических описаниях к возвращению в режим "PRI-RUN" применяют выражение "wake up" ("пробуждение") даже в том случае, если центральный процессор перед этим был активен. Но мы бы отнесли это выражение, скорее, к случаю "пробуждения" центрального процессора из "спящего" режима.

При переходе из режима "SEC-RUN" или "RC-RUN" обратно в режим "PRI-RUN" центральный процессор не нуждается в "пробуждении" — он работает с предыдущим осциллятором до тех пор, пока не стабилизируется сигнал главного осциллятора. При передаче командования центральный процессор "задерживает дыхание" на восемь импульсов (рис. 4.2).

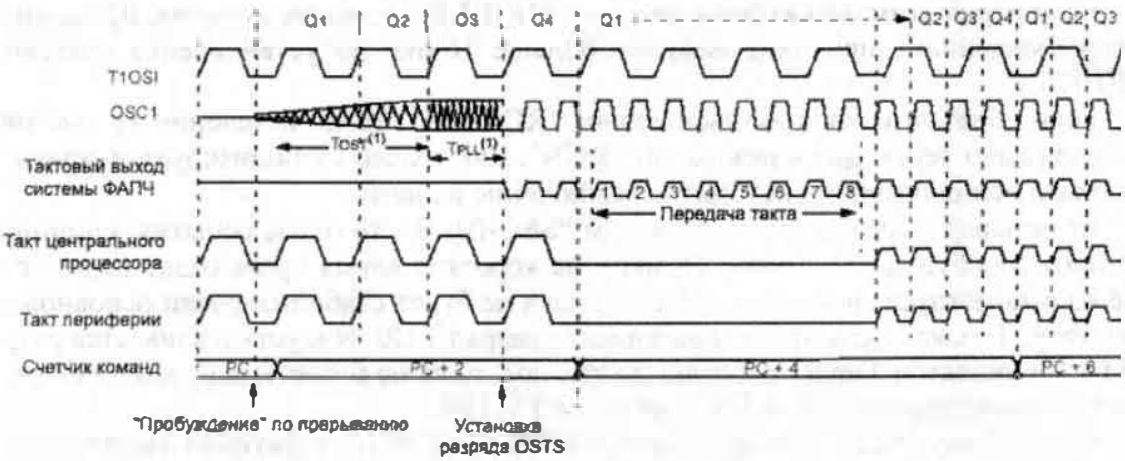


Рис. 4.2. "Пробуждение" из вспомогательного режима "RUN"

4.6. "Пробуждение" из режимов "IDLE" и "SLEEP"

"Пробуждение" из состояния "SLEEP" — знакомый нам процесс. Опять должен формироваться основной тактовый сигнал как после сброса, и центральный процессор возобновляет работу. Счетчик команд указывает на команду, переведшую в "спящее" состояние.

"Пробуждение" из режима "IDLE" происходит аналогичным образом, за исключением того, что на момент "пробуждения" обязательно присутствует активный сигнал синхронизации, который временно принимает на себя обработку команды.

"Пробуждение" без сброса происходит или по завершению счета сторожевого таймера или через прерывание. **В отличие от режимов "RUN", сторожевой таймер в состояниях "SLEEP" и "IDLE" сброс не вызывает.**

4.6.1. "Пробуждение" через прерывание

Возможности для "пробуждения" через прерывание в режиме "SLEEP" ограничены. Осциллятор Timer1 может работать и в режиме "SLEEP", и тем самым инициировать все без исключения прерывания, связанные с этим таймером. АЦП также может тактироваться от своего внутреннего RC-осциллятора, благодаря чему может формировать прерывание в режиме "SLEEP". Для "пробуждения" из режима "SLEEP" могут также применяться внешние прерывания, которые вообще не нуждаются в синхроимпульсах.

В режиме "IDLE" прерывание может быть вызвано любым источником прерывания. Все периферийные устройства обеспечиваются синхроимпульсами, если только они не были выключены программно.

Предпосылкой для "пробуждения" через прерывание является установка соответствующего разряда в регистре INTCON или PIE. Установка разряда общего разрешения прерываний необязательна — если он не установлен, происходит "пробуждение", но подпрограмма обработки прерывания не выполняется.

4.6.2. Процесс "пробуждения"

После события, послужившего причиной для "пробуждения", проходит около 10 мкс прежде, чем может возобновиться выполнение кода.

Если перед этим был активен режим "PRI-IDLE", то после события, приведшего к "пробуждению", проходит немного больше 10 мкс до установления состояния "PRI-RUN".

Если перед этим был активен режим "RC-IDLE", то по истечении 10 мкс микроконтроллер переходит в режим "RC-RUN". Как только стабилизируется основной тактовый осциллятор, происходит переключение на него.

Если перед этим был активен режим "SEC-IDLE", то после события, инициировавшего "пробуждение" (плюс 10 мкс), на короткое время происходит переключение во вспомогательный режим "RUN", пока не будет стабилизирован основной осциллятор. Только после этого сбрасывается разряд T1RUN и устанавливается разряд OSTS. Осциллятор Timer1 работает до тех пор, пока не будет выключен в результате обнуления разряда T1OSCEN в регистре T1CON.

Осциллятор INTRC включается в зависимости от того, активно ли отслеживание сбоев синхронизации и сторожевой таймер.

4.7. Смена осциллятора

Переключение между источниками тактовых импульсов реализовано настолько надежно, насколько это возможно (рис. 4.3). При переходе от основного к вспомогательному источнику это выглядит следующим образом: основной источник отключается, подсчитывается восемь периодов вспомогательного источника, и затем на устройство подаются синхроимпульсы. Прим этом, разумеется, предполагается, что осциллятор Timer1 вообще включен. Если это не так, то попытка переключиться

в режим "RUN" или "IDLE" будет проигнорирована. Если осциллятор Timer1 работает нестабильно, тогда могут произойти непредсказуемые вещи. По этой причине подобное не рекомендуется.

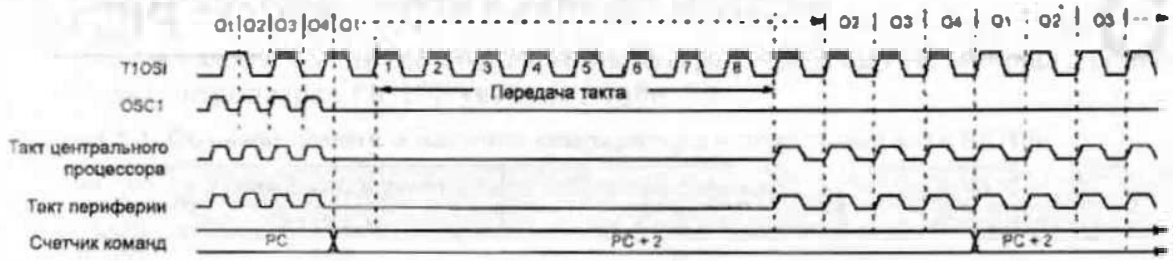


Рис. 4.3. Переход из основного режима "RUN" к вспомогательному

ГЛАВА

5 PIC10F

Основное назначение PIC10F (рис. 5.1) — замена нескольких логических элементов, поэтому этот тип микроконтроллеров оснащен памятью малого объема. Кроме того, благодаря встроенному компаратору, этот тип может заменять аналоговые компоненты. Сигнал с выхода этого компаратора может внутренне считываться или подаваться на вход таймера Timer0.

Тем не менее, скромные характеристики PIC10F ничуть не говорят о его слабости. С помощью 512 программных слов мы уже реализовали ряд серьезных многозадачных программ.

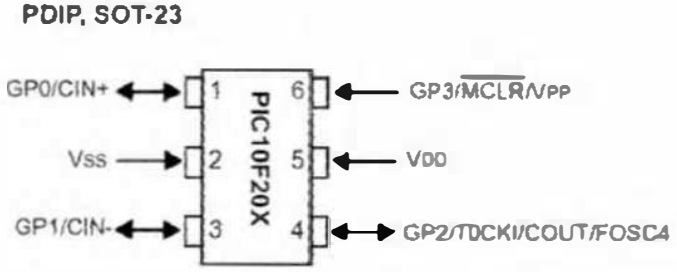


Рис. 5.1. Малейший из PIC

5.1. Беглый обзор характеристик

Перечислим основные характеристики PIC10F:

- память программ на 256/512 слов;
- память данных на 16/24 байт;
- ядро 5X; команды `SLEEP`, `TRIS` и `OPTION`;
- внутренний осциллятор частоты 4 МГц; возможность получить $F_{osc}/4$;
- диапазон рабочих напряжений — 2,0...5,5 В;
- корпус на 6/8 выводов;
- четыре вывода портов: три — входы/выходы; один — только вход;
- тип входов — TTL; тип выходов — КМОП; входы T0CKI и MCLR — с триггером Шмитта;
- три вывода с внутренними подтягивающими резисторами;
- компаратор с опорным напряжением;
- таймер TMR0;
- сторожевой таймер.

5.2. Аппаратные свойства

5.2.1. Структура памяти и наличие модулей

Объемы памяти и наличие модуля компаратора у представителей нового семейства микроконтроллеров PIC10F указаны в табл. 5.1.

Таблица 5.1. Объемы памяти и наличие компаратора у представителей PIC10F

Тип	*Память программ, в словах*	*Память данных, в байтах*	*Модуль компаратора*
PIC10F200	256	16	–
PIC10F202	512	24	–
PIC10F204	256	16	Есть
PIC10F206	512	24	Есть

5.2.2. Формы корпуса и число выводов

В качестве формы корпуса PIC10F используются PDIP8 и SOT-23. В случае корпуса SOT все выводы заняты, а в случае PDIP8 — два вывода не используются. Для того чтобы, несмотря на малое число выводов, реализовать еще четыре вывода, отказываются от использования внешней осцилляторной схемы.

5.2.3. Внутренний RC-осциллятор

Таким образом, микроконтроллер PIC10F может работать исключительно на своем внутреннем осцилляторе с частотой 4 МГц. При этом диапазон напряжения питания составляет 2,0...5,5 В. Точность внутреннего осциллятора вполне приемлема (табл. 5.2).

Таблица 5.2. Точность внутреннего осциллятора PIC10F

Допуск	*Диапазон напряжений*	*Диапазон температур*
+/– 1%	TBD	TBD
+/– 2%	2,5...5,5 В	TBD
+/– 5%	2,0...5,5 В	-40...+125°C

Сокращение "TBD" — производное от английского "to be defined", что в переводе на русский означает "еще должно быть специфицировано".

Разъясним суть табл. 5.2:

- если колебания напряжения и температуры не слишком велики, то для осциллятора можно принять допуск 1%;
- если напряжение может сильно изменяться, однако температура колеблется только в некотором узком диапазоне, то следует принять допуск 2%;
- если напряжение и температура в процессе работы микроконтроллера могут изменяться во всем допустимом диапазоне, то в расчетах берут допуск 5%.

Как и в случае любого другого мнкроконтроллера с внутренним RC-осциллятором, в серии PIC10F используется регистр OSCCAL, уже запрограммированный Microchip. Значение коррсктировки размещается в команде MOVLW в последней ячейке памяти программ. При стирании кристалла и новом программировании эта информация теряется, если, конечно, заранее не позаботиться о ее сохранении.

Поскольку в данном случае речь идет о флэш-памяти, мы получаем возможность самостоятельно устанавливать и корректировать значение регистра OSCCAL.

Таким образом, если в конкретном случае применения мы можем исходить из определенного напряжения питания и установить конечное значение температуры, то на последнем этапе производства изделия можем настроить осциллятор в точности в соответствии с преобладающими условиями эксплуатации.

Для этого в среде тестирования на вывод GP2 подают сигнал с частотой осциллятора (OSC/4) и замеряют его. В противоположность бытовавшей ранее традиции, в PIC10F частоту осциллятора получают с помощью регистра CONFIG, а не OSCCAL. Адрес регистра OSCCAL (табл. 5.3) — 05_h:

Таблица 5.3. Структура регистра OSCCAL

Разряд	7	6	5	4	3	2	1	0
Обозначение	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	FOSC4
Состояние после сброса	1	1	1	1	1	1	1	0

Разряд 0 выполняет роль переключателя для активизации/отключения подачи сигнала OSC/4 на вывод GP2. Если этот разряд содержит 0, то вывод GP2 работает в обычном режиме, иначе на него выдаются импульсы с частотой, равной одной четверти частоты осциллятора.

5.2.4. Внутрисхемное последовательное программирование

Возможность внутрисхемного последовательного программирования реализована в микроконтроллере PIC10F20X. Это означает, что предусмотрена обычная связь между программатором и микроконтроллером PIC (табл. 5.4).

Таблица 5.4. Распределение выводов PIC10F20X в режиме последовательного программирования

Линия программатора	*Описание*	*Вывод PIC10F20X*	*Номер вывода в корпусе PDIP8*	*Номер вывода в корпусе SOT-23*
VCC	Напряжение питания	VDD	2	5
GND	"Земля"	VSS	7	2
VPP	Напряжение программирования	GP3/MCLR/VPP	8	6
CLK	Тактовый сигнал	GP1	4	3
Ввод-вывод данных	Двунаправленная линия последовательной передачи данных	GP0	5	1

Хотя этот раздел и назван "Внутрисхемное последовательное программирование", само собой, никто не мешает выпаять микроконтроллер из схемы, поместить его непосредственно в программатор и таким образом запрограммировать. Из табл. 5.4 видно, что во впаянном состоянии назначение всех линий меняется.

Линия VDD

То, что эта линия связана в схеме именно таким образом, — вполне логично. Однако эта связь в случае внутрисхемного программирования порождает проблему для других конструктивных элементов схемы. В том случае, если для схемы не может быть обеспечен блок питания, следует позаботиться о том, чтобы напряжение было подано именно через программатор. Кроме того, емкостная нагрузка напряжения питания не должна быть слишком большой, поскольку иначе программатор не сможет реализовать крутизну фронта, требуемую спецификацией программирова-

ния. "Нормальная" нагрузка всего обслуживаемого узла, естественно, также должна покрываться программатором. Если она слишком велика, то можно подумать о том, чтобы изолировать остальную цепь VCC от программируемой микросхемы с помощью диода. Если контакт схемы не жесткий, а выполнен в виде специального разъема, то развязку можно альтернативно реализовать с помощью этого разъема. Когда программирование не выполняется, получается короткозамкнутый разъем, устанавливающий нормальные связи.

Линия /MCLR/VPP

На этот вывод, работающий только в режиме входа, в нормальном режиме работы может подаваться напряжение VDD или любое другое напряжение от формирователя сигнала сброса. В случае программирования на этот вывод подается напряжение программирования 13 В. Здесь также должна быть предусмотрена развязка, чтобы эти отличающиеся напряжения при программировании микросхемы "не провалились с треском". Для этого между выводом VDD (или выходом формирователя сигнала сброса) и входом MCLR следует включить резистор. Для вывода MCLR, как и для рассмотренного ранее вывода VDD, следует учитывать емкостную нагрузку, поскольку крутизна фронтов сигнала VPP также специфицирована и должна строго соблюдаться.

Линия CLK и линия данных

В нормальном режиме работы эти выводы могут служить как входами, так и выходами. В данном случае режим программирования также должен быть обособлен. Если сигнал поступает в микроконтроллер PIC как входной, то для развязки опять должен быть использован резистор. Если же микроконтроллер формирует выходной сигнал, то дело обстоит несколько сложнее. Последующая ступень схемы, работающая под управлением этого сигнала, должна быть каким-то образом отсоединена, иначе ей может быть причинен какой-либо вред. Если такой опасности наверняка не существует, то подобные меры предосторожности излишни.

Подводя итог всему сказанному, очевидно, что среда внутрисхемного программирования должна быть тщательно продумана! Следует быть внимательным даже на этапе прокладки линий к выводам. Слишком большая длина линий точно так же рискованна как и большие участки параллельно проложенных линий.

5.2.5. Внутренние слова и ядро 5X

5X — это простейшее процессорное ядро, уже известное нам по микроконтроллеру PIC16C54 и др. Оно поддерживает только 33 команды и не обрабатывает прерывания. Загрузка регистров TRIS и OPTION опять таки выполняется с помощью команд `OPTION` и `TRIS`. Непосредственный доступ невозможен.

Распределение разрядов в регистре OPTION нам хорошо знакомо (табл. 5.5).

Таблица 5.5. Структура регистра OPTION в случае ядра 5X

Разряд	7	6	5	4	3	2	1	0
Обозначение	/GPWU	/GPPU	T0CS	T0SE	PSA	PS2	PS1	PS0
Состояние после сброса	1	1	1	1	1	1	1	1

Что касается всего остального, то в случае ядра 5X используются внутренние подтягивающие резисторы, а также "пробуждение" по изменению уровня сигнала на некотором выводе. Хотя подобные микроконтроллеры PIC и не поддерживают

прерывания, существует возможность переводить их в энергосберегающий "спящий" режим с "пробуждением" по различным действиям.

Среди таких "пробуждающих" действий можно назвать:

- изменение уровня сигнала на некотором выводе;
- переключение вывода компаратора;
- переполнение сторожевого таймера.

Переполнение таймера Timer0 к этому списку, естественно, не относится. Поясним почему. Для того чтобы Timer0 мог формировать переполнение, он должен постоянно работать, однако, по определению, главный осциллятор в "спящем" режиме отключен. Таким образом, таймер должен быть обеспечен внешним тактовым сигналом, но поскольку мы уже знакомы со свойствами ядра 5X, то знаем, что внешний такт для Timer0 — тоже не помощник. Другими словами, переполнение Timer0 никак не может привести к выводу из "спящего" режима.

5.2.6. Выводы по одному

Блок-схема структуры одного вывода порта представлена на рис. 5.2.

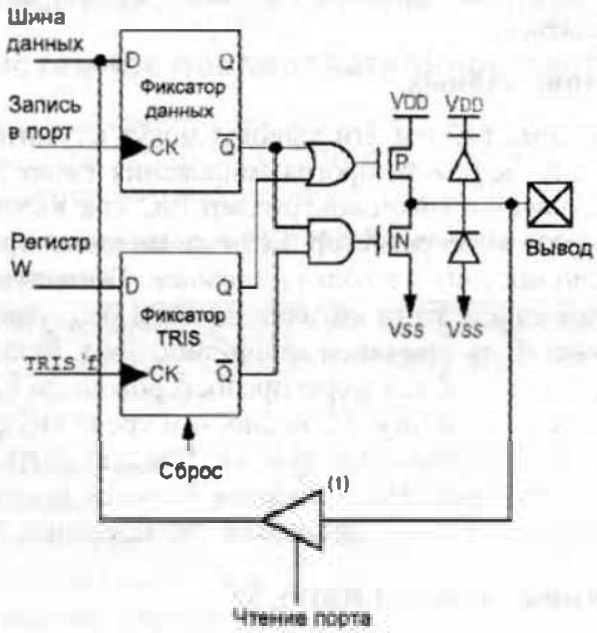


Рис. 5.2. Блок-схема вывода порта

GP0

Если этот вывод включен как обычный вход, то он TTL-совместим. Для питающего напряжения в диапазоне 4,5...5,5 В это означает, что напряжение ниже 0,8 В распознается как низкий уровень, а напряжение выше 2 В — как высокий, выходной каскад — типа КМОП. Это, опять-таки, означает, что на выходе, переключенном в состояние низкого уровня, никогда не превышается напряжение 0,6 В при нагрузке до 8,5 мА, а при состоянии высокого уровня напряжение никогда не опускается ниже значения VDD –0,7 В при нагрузке –3 мА. Теперь наступил момент еще раз поговорить о проблеме "чтения-модификации-записи" (см. раздел 1.7).

Корень всех зол — неверная интерпретация входного уровня, который в результате обработки (см. ниже) приводит к соответствующей ошибке. Таким образом, проблема "чтения-модификации-записи" возникает только в связи с портом ввода-

вывода. Подобные неверные распознавания входного уровня имеют место потому, что повышенный выходной ток фальсифицирует выходной уровень.

Если некоторый вывод (1) кратковременно переключить из состояния выхода в состояние входа и затем выполнить команду типа "чтение-модификация-запись", которая обрабатывает совершенно другой вывод (2), то на выводе (1) может измениться уровень в соответствии с содержимым его выходного регистра. Так, по вышеупомянутой команде в выходной регистр записывает неверное состояние, и при обратном переключении вывода (1) в режим выхода на нем появится некорректный уровень.

И еще несколько слов о фальсификации выходного уровня в результате перегрузки... Нагрузка более 10 мА приводит к повышению выходного напряжения, соответствующего низкому уровню, до значения выше 0,8 В. Поскольку такой уровень лежит в "запрещенном" диапазоне, низкий уровень на выводе, переключенном в состояние входа, больше не может считаться достоверным. В худшем случае выходной уровень повышается до значения свыше 2,0 В, что приводит к ложному распознаванию высокого уровня.

Под упомянутой выше "обработкой" могут подразумеваться логические или арифметические операции, реализуемые с помощью таких команд как `ADDWF`, `IORWF`, `BCF`, `BSF` и др. Во всех таких командах считывается байт или порт, значение изменяется и затем опять записывается.

Таковая проблематика команд типа "чтение-модификация-запись". Наконец, для вывода GP0 допускается программирование подтягивающего резистора, а также подача сигнала для вывода микроконтроллера из "спящего" состояния.

Если в случае внутрисхемного последовательного программирования вывод GP0 используется как линия ICSPDAT, то он принимает характеристики входного каскада с триггером Шмитта. Если же он используется в качестве входа компаратора CIN+, то является аналоговым входом.

GP1

Для этого вывода справедливо все, что было сказано выше для вывода GP0, за тем исключением, что в случае внутрисхемного последовательного программирования он исполняет функции линии ICSPCLK.

Вход компаратора, соответствующий выводу GP1, называется CIN–.

GP2

Вывод GP2 может работать как обычный вход/выход, однако без подтягивающего резистора и возможности "пробуждения" микроконтроллера.

Альтернативно, этот вывод может служить в качестве входа для таймера Timer0 (T0CKI). В этом случае он принимает характеристики входного каскада с триггером Шмитта. Еще один вариант конфигурации вывода GP2 — выход компаратора COUT (см. разряд COUTEN в регистре CMCON0).

И последний вариант использования — для выдачи импульсов с частотой, равной одной четвертой частоты системной синхронизации ($F_{OSC}/4$). См. разряд FOSC4 в регистре OSCCAL.

GP3

Вывод GP3 не имеет выходного каскада и может работать как обычный вход TTL. Подобно выводам GP0 и GP1, GP3 оснащен подтягивающим резистором и допускает "пробуждение" микроконтроллера из "спящего" состояния по изменению уровня сигнала (см. разряды /GPWU и /GPPU в регистре OPTION). Избирательное

включение подтягивающих резисторов невозможно: или все (GP0, GP1 и GP3), или ни один. Это же относится и к характеристикам "пробуждения".

Второй способ использования вывода GP3 — в качестве входа /MCLR, с помощью которого можно подать сигнал сброса, активный в случае низкого уровня. В такой конфигурации вход приобретает характеристики триггера Шмитта, и автоматически подключается подтягивающий резистор. Вывод GP3 функционирует полностью независимо от других выводов, а входное напряжение на нем не может превышать напряжения питания, иначе произойдет переход в режим программирования. Таким образом, мы подошли к третьему способу использования вывода GP3.

Если вывод GP3 используется как вход при программировании, на него подается напряжение программирования 13 В.

И еще одно общее замечание... Выше было сказано, что входное напряжение не должно превышать уровня напряжения питания. Это, конечно же, относится ко всем выводам, а не только к GP3. Если и не произойдет переключение в режим программирования, то в любом случае начнет протекать ток через защитные диоды, что в худшем случае может привести к знаменитому тиристорному эффекту (latch-up effect), а это означает смерть микроконтроллера.

5.2.7. Модуль компаратора

Модуль компаратора включает в себя аналоговый компаратор и внутренний источник опорного напряжения. Аналоговые входы мультиплексированы с выводами GP0 и GP1, а выход компаратора может быть подключен на вывод GP2. Впрочем, это — необязательное условие, поскольку результат работы компаратора можно опрашивать внутренне или подавать непосредственно на вход таймера Timer0.

Регистр CMCON0 (табл. 5.6), кроме разрядов настройки различных опций, содержит также разряд, имеющий отношение к выходу компаратора.

Таблица 5.6. Структура регистра CMCON0

Разряд	7	6	5	4	3	2	1	0
Обозначение	CMPOUT	/COUTEN	POL	/CMPT0CS	CMPON	CNREF	CPREF	/CWU
Состояние после сброса	1	1	1	1	1	1	1	1

Выходу компаратора соответствует разряд 7 (CMPOUT). Его можно опросить в любой момент. С помощью разряда 6 (/COUTEN) выход компаратора можно дополнительно соединить с выводом GP2, чтобы выдавать результат сравнения напряжений во внешние модули.

Полярность выхода компаратора переключается с помощью разряда 5 (POL).

В случае разряда 4 (/CMPT0CS) речь идет об источнике тактового сигнала для таймера Timer0. Если этот разряд содержит 0, то выход компаратора используется напрямую как тактовая линия для Timer0, если же он содержит 1, то источник тактирования таймера определяется по состоянию разряда T0CS в регистре OPTION.

Под номером 3 (CMPON) скрывается "главный рубильник" компаратора. Если этот разряд содержит 0, то модуль компаратора выключен.

С помощью разряда 2 (CNREF) входу отрицательного напряжения компаратора назначается опорное напряжение или сигнал на выводе CIN–.

С помощью разряда 1 (CPREF) входу положительного напряжения компаратора назначается сигнал на выводе CIN+ или CIN–. Если разряд 0 (/CWU) содержит 0, то компаратор может выводить микроконтроллер PIC из "спящего" состояния. В противном случае он такое возможности не имеет.

ГЛАВА

6 rfPIC

rfPIC (рис. 6.1) — это небольшое семейство микроконтроллеров PIC, оснащенное встроенным (точнее сказать, — размещенным в том же корпусе) высокочастотным (ВЧ) передатчиком. Эти два элемента полностью отделены друг от друга, что относится также и к электропитанию. Таким образом, между центральным процессором и ВЧ-блоком не предусмотрено никакого интерфейса в виде регистров специального назначения, как в случае аппаратных модулей.

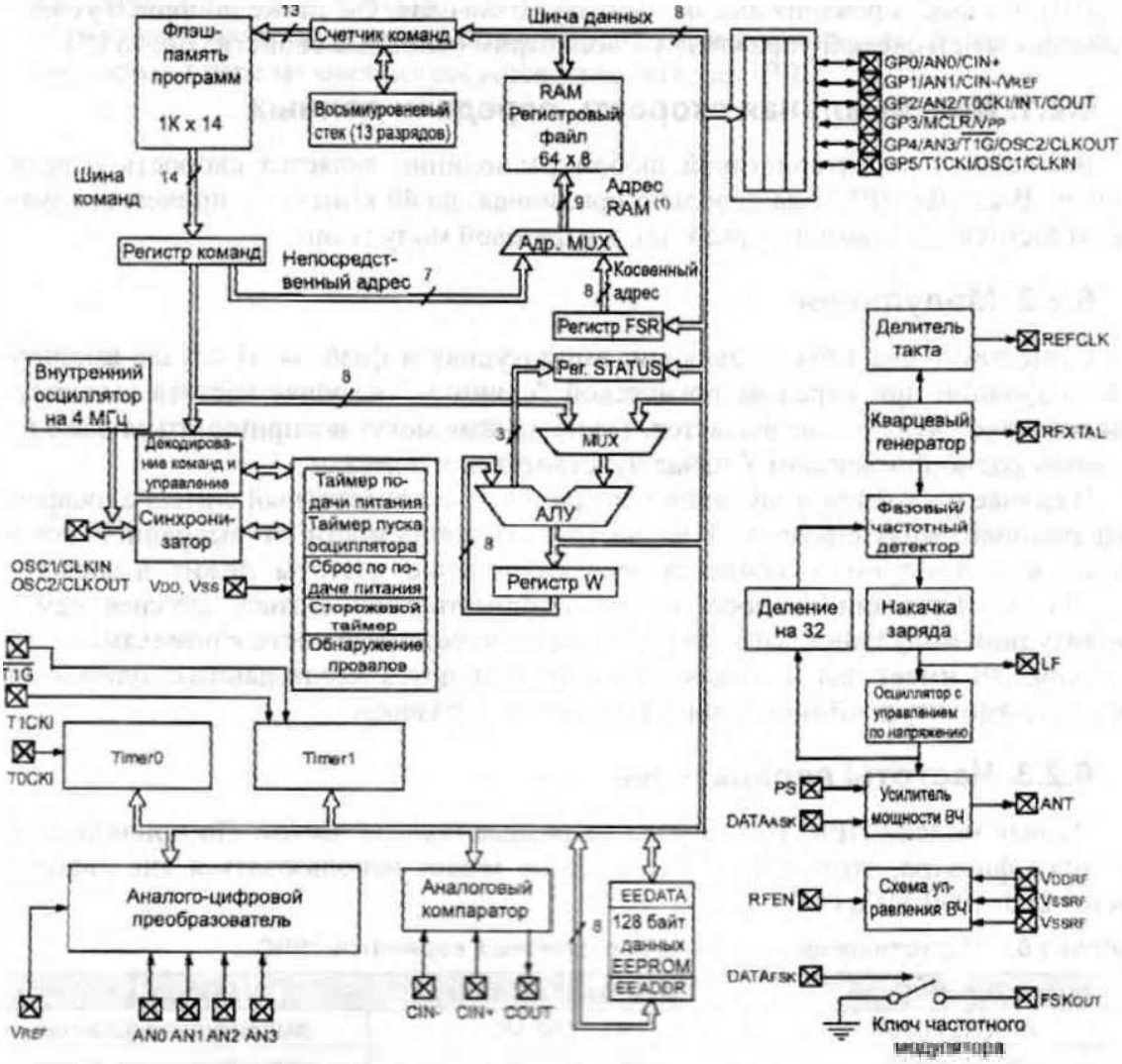


Рис. 6.1. Блок-схема микроконтроллера rfPIC

6.1. Блок микроконтроллера

Существует несколько вариантов rfPIC. В более старых типах был интегрирован микроконтроллер PIC12C509, а новые модули, выполненные с использованием флэш-технологии, содержат микроконтроллер PIC12F675. Все, что нам известно об "обычном" PIC12F675, относится также и к rfPIC, а именно: объем памяти программ и памяти данных, точность RC-осциллятора, глубина стека и т.д.

Самый значительным преимуществом такого микроконтроллера является наличие репрограммируемой флэш-памяти. Так, в типовых изделиях, наподобие пульта дистанционного управления телевизором, очень важна возможность "обучаемости", которая становится возможна, благодаря памяти EEPROM.

Как видно на рис. 6.1, выводы 1–4 и 17–20 относятся к блоку микроконтроллера. Распределение выводов — такое же, как в уже знакомом нам восьмиконтактном корпусе.

6.2. Блок ВЧ

Этот блок состоит из осциллятора и системы фазовой автоподстройки частоты (ФАПЧ) для формирования высокочастотного сигнала. Он также дополнен усилителем мощности и схемой управления. Рассмотрим основные свойства блока ВЧ.

6.2.1. Максимальная скорость передачи данных

Важнейшей характеристикой любой радиолинии является скорость передачи данных. В случае rfPIC она довольно приличная: до 40 кБит/с, — причем это значение относится как к амплитудной, так и к фазовой модуляции.

6.2.2. Модуляция

Существует два типа модуляции: амплитудная и фазовая. В случае амплитудной модуляции при передаче логической "единицы" несущая частота выдается, а при передаче "нуля" — не выдается. Такие данные могут восприниматься даже простейшим радиоприемником. Сигнал чувствителен к помехам.

В случае частотной модуляции передается высокочастотный сигнал с цифровой информацией, модулированной по частоте. В зависимости от выбранной полосы частот и используемых конденсаторов отклонение частоты лежит в диапазоне 5,5…34 кГц. Приемники, способные воспринимать такой сигнал, сложнее, чем для амплитудной модуляции, зато гораздо меньше чувствительность к помехам.

Блок ВЧ имеет два изолированных друг от друга входа данных: один — для сигнала с амплитудной модуляцией, а другой — с фазовой.

6.2.3. Частоты передатчика

Разные модели rfPIC рассчитаны на разные полосы частот. По причине обязательного фильтра, этот микроконтроллер не может использоваться вне диапазона частот 290…930 МГц (табл. 6.1).

Таблица 6.1. Частотные диапазоны для различных вариантов rfPIC

Тип rfPIC	*Диапазон частот*	*Модуляция*
rfPIC12F675K	290…350 МГц	амплитудная/фазовая
rfPIC12F675F	380…450 МГц	амплитудная/фазовая
rfPIC12F675H	850…930МГц	амплитудная/фазовая

Поскольку частота передатчика формируется с помощью системы ФАПЧ, достаточно только кварца с частотой в 32 раза меньше желаемой частоты передатчика.

Кварцы, применимые для формирования частоты передатчика:

- rfPIC12F675K — 9,06...10,9 МГц;
- rfPIC12F675F — 11,875...14,06 МГц;
- rfPIC12F675H — 26,5...29,06 МГц.

Таким образом, замена кварца передатчика приводнт к изменению используемого канала, а значит можно реализовать только передатчик с фиксированной частотой.

В отношении частотного диапазона, хотелось бы затронуть очень непопулярную тему регламентации. Причем для каждой страны правила отличаются! Используемые частоты, допустимая излучаемая мощность и разрешенная длительность передачи — все эти параметры в разных странах регулируются по-разному. В результате разработка продуктов, предназначенных для международного применения, может иметь далеко идущие последствия!

Однако, к сожалению, не достаточно просто выяснить нормы. Предварительно требуется также выяснить, какие ограничения планируют ввести. Чего стоят одни только наводящие ужас стандарты FCC!

Не каждое изделие является международным, и в некоторых случаях для изменения частоты передатчика потребуется изменить тип rfPIC.

ГЛАВА 7

Программирование микроконтроллеров PIC на языке ассемблера

Спрос на PIC-ассемблер неизменно высок, а слухи о его смерти — преувеличены. Авторы книги сами программируют на ассемблере, поскольку уже наработали удобную инфраструктуру, знакомы с компиляторами для PIC, начиная с самых первых версий, и разрабатывают исключительно машинно-ориентированные и очень критичные ко времени проекты. Мы повсеместно применяем макросы, без которых в большинстве случаев процесс программирования был бы долгим и проблематичным.

К нам поступает много вопросов по поводу ассемблерной арифметики, и потому несколько страниц этой главы будут посвящены обращению с числами.

Для работы с ассемблерной арифметикой не нужно познаний в высшей математике — достаточно только прикладных вычислений. Тем не менее, они не так уж просты и требуют большой точности, терпения и внимательного обращения с мелочами. По сравнению с программированием на языках высокого уровня, риск допустить ошибку гораздо выше. В случае языков высокого уровня исходят из того, что стандартные процедуры не содержат ошибок, хотя, конечно же, при разработке программ все равно допускается множество ошибок.

Во многих высокоскоростных случаях применения время расчета параметров регулирования или управления настолько ограничено, что для максимального сокращения времени вычислений приходится применять разнообразные математические трюки. И зачастую это возможно только в результате применения алгоритмов "ручной работы".

Увеличение частоты осциллятора в большинстве случаев проблему не решает, поскольку влечет за собой недостатки физического характера (например, повышение потребления мощности, что приводит к перегреву и электромагнитной несовместимости). В таких случаях более подходящее решение — потратить пару часов на изобретение какого-либо оригинального программного метода.

7.1. Форматы чисел

Мир микроконтроллеров по своей природе целочисленный. Если речь идет о результатах измерения, то эти целые числа получают с помощью АЦП или какого-либо вычислительного процесса. Также и время измеряется в целых долях элементарных единиц времени. Управление выходами ШИМ осуществляется с помощью целочисленного регистра ШИМ.

Восемь разрядов любого рабочего регистра можно рассматривать или просто как скопление битов, или же как число, которое в простейшем случае лежит в диапазоне от 0 до 255.

Для случаев практического применения целых числе в таком диапазоне, конечно же, недостаточно. Нам требуются значения больше 255, а также отрицательные и дробные числа.

7.1.1. Двухбайтные слова

Получить число больше 255 — не проблема. Достаточно просто использовать два байта точно так же как для представления десятичных числе выше девяти достаточно добавить еще один разряд. Если же и этого недостаточно, то можно взять три и больше байтов.

Примечание

В этой книге мы для краткости называем "словом" любое число, состоящее из нескольких байтов (в традиционном толковании это понятие относится к элементарной единице данных, обрабатываемых процессором). Если количество байтов явно не указано, то предполагается, что слово — двухбайтное.

В случае (двухбайтного) слова оба байта обычно обозначают одни и тем же названием. При этом иногда в обозначении старшего байта присутствует префикс "H" (от англ. "High"), а в обозначении младшего байта — префикс "L" (от англ. "Low"). В формулах для символического именования значения слова часто к общему названию прибавляют букву "X", например, ZX = 256 * ZH + ZL. Наибольшее значение, которое можно получить с помощью одного слова — 65535. Также часто используют запись вида ZX = ZH:ZL.

ZX — это имя рабочей переменной, которую мы чаще всего применяем в примерах в качестве счетчика при умножении, а NX=NH:NL — обобщенная переменная или второй счетчик. Переменную результата мы называем ERGX=ERGH:ERGL. Если двух байт недостаточно, то мы используем четыре: ZWERGH:ZWERGL:ERGH: ERHL.

Совет

Полезный совет программистам на ассемблере: байты больших слов должны размещаться в памяти RAM друг за другом. При этом первым (то есть, по меньшему адресу) следует младший байт. Таким же образом размещаются в RAM все двухбайтные регистры специального назначения.

Если для обработки слов используется макрос, то в качестве аргумента ему достаточно передать только единственный адрес. Это означает, что имена старших байтов зачастую вообще не нужны. — используются только имена младших байтов, которым соответствуют общие имена переменных. Таким образом, для обращения к адресам байтов в командах используют, например, запись ERGL, ERGL+1, ERGL+2, ERGL+3.

Если имена старших байтов явно не используются, то для объявления переменной можно, к примеру, использовать запись ERGL:2 или ERGL:4. В результате ассемблер автоматически зарезервирует под переменную два или четыре байта.

7.1.2. Отрицательные числа

В повседневной жизни отрицательные числа не вызывают у нас никаких затруднений. Мы просто берем привычное положительное число и ставим перед ним "минус". К тому же, каждый владеет парой простых правил вычислений с отрицательными числами, наподобие "минус на минус дает плюс".

Но как же обращаться с отрицательными числами, если речь идет о программировании на ассемблере для микроконтроллеров? По языкам высокого уровня нам известен числовой тип "Integer", который с помощью двух байтов дает диапазон значений –32700...+32700. Конечно, использование таких чисел при программировании на ассемблере было бы очень практично, однако в результате резко возрастает вероятность ошибки. С одной стороны, сложение и вычитание не вызывает особых проблем, но с другой стороны необходимо тщательно отслеживать переполнения. Также возможна ситуация, когда кто-то выполнил умножение с отрицательным целым числом и потом удивляется полученному результату.

Если требуется выполнить умножение или деление с участием отрицательного числа, то всегда вначале необходимо получить абсолютное значение. С практической точки зрения, это основано на организации чисел, при которой один разряд отводится для знака, а остальная часть — для абсолютного значения.

Зачастую бывает так, что программист отказывается от старшего разряда значения переменной и использует его в качестве флага знака. Если подобная переменная должна быть задействована в вычислениях, то вначале следует отделить знак.

Если позволяет объем свободной памяти RAM, то программист на ассемблере может выделить для знака отдельную переменную. Тем не менее, объем памяти в большинстве случаев все еще требует бережливого отношения, и потому мы зачастую знаки всех переменных собираем в один регистр. Это становится недостатком, если для выполнения операций с такой "знаковой" переменной, постоянно используется макрос или подпрограмма.

Нередко программисты вообще отказываются от знакового разряда и определяют знак промежуточного результата просто по изменениям в окружении.

Пример: X = A * (U – V).

Принимаем, что A, U и V — положительные числа (байты), а X — переменная-слово. Прежде всего мы загружаем A в рабочий регистр для умножения (ZL). Затем формируем разность W = (U – V). Если результат положительный или равен нулю, то мы это определяем по установке флага C (будьте внимательны, чтобы не привыкнуть к прямо противоположному).

Если результат отрицательный, то должны выполняться два обстоятельства:

- должен быть установлен разряд знака результата;
- должен быть сформирован модуль числа W (W := –W).

Последнее невозможно выполнить одной строкой (разве что в микроконтроллерах PIC18):

```
XORLW    0FFH          ; Дополнение W
ADDLW    1             ; W+1
```

Теперь можно умножить W на ZL, воспользовавшись обычным алгоритмом перемножения байтов. В случае PIC18 это реализуется с помощью одной команды, в остальных же случаях потребуется отдельная подпрограмма.

7.1.3. Действия с дробями

Если с помощью калькулятора выполнить действие "7:11", то будет получен результат 0,636363636. В том случае, если нам достаточно только две цифры после запятой, после округления получаем результат 0,64.

Что же нам дает число 0,64 вместо 7:11? Строго говоря, дробь 7:11 мы просто заменили еще одной дробью: 64:100. Польза только в том, что нам удобнее работать

с дробями, если в их знаменателе находится степень десяти. Если такое число разделить или умножить на десять, то в результате будет только смещена вправо или влево десятичная точка.

Таким способом мы получаем результат в форме, которая дает хорошо нам знакомые целые числа. Значение 0.64 — это 64, умноженное на 10^{-2}, а 0.636 — это 636, умноженное на 10^{-3}. Каждое число, с практической точки зрения, рассматривается как произведение некоторой элементарной величины самой на себя целое количество раз.

Отсюда следует, что, в общем случае, каждое число можно представить в так называемой экспоненциальной форме (с основой 10): $M*10^{EXP}$, где M — называют "мантиссой", а EXP — "порядком".

Когда мы делим в микроконтроллере два целых числа, то применяем точно такой же подход, что и при делении двух десятичных чисел, с той лишь разницей, что наши цифры могут принимать только значения 0 и 1. Таким образом, знаменатель представляет собой степень не десяти, а двух, а значит метод деления даже проще того, который мы учили в школе.

Сколько должно быть "знаков после запятой", — это выбор программиста. Обычно удобнее всего принять восемь или шестнадцать знаков. Тем не менее, во многих случаях количество десятичных знаков зависит от значения результата ("плавающая точка").

Проще всего приближенно представить число X в диапазоне от 0 до 1 в форме $X = M * 2^{-8} = M/256$.

При этом M — байт, и его значение всегда можно выбрать таким образом, чтобы (абсолютная) погрешность приближения была меньше или равна ±1/512.

Таким образом, для хранения подобного числа достаточно только одного байта для мантиссы. При этом, само собой разумеется, при дальнейших расчетах следует учитывать, что число должно еще быть поделено на 256.

Пример

Число 1/3 можно приближенно представить как 85/256. Таким образом, мантисса имеет значение 85. Разница (1/3 – 85/256) составляет примерно 0,0013.

Если представление с порядком –8 для конкретного случая применения слишком неточное, тогда его можно изменить на представление с порядком –16. В таком случае для хранения мантиссы потребуется слово, а абсолютная погрешность составит максимум ±0,0000076, чего для большинства случаев применения вполне достаточно (хотя и не всегда).

Если число слишком мало и зависит от относительной точности, то можно применить другое решение: выбрать порядок в зависимости от значения числа. В таком случае переменной величиной в уравнении $X = M * 2^{EXP}$ становится не только мантисса, но также и порядок.

В таком случае в памяти следует хранить и мантиссу, и порядок. Такой формат, конечно же, не ограничивается только отрицательными значениями порядка, и потому с его помощью можно выразить любое число. Особенно он полезен для переменных, которые могут принимать как очень малые, так и очень большие значения.

7.1.4. Вычисления с экспоненциальными форматами

Экспоненциальный формат с переменным порядком имеет одно большое преимущество: всегда получается поддающаяся оценке (относительная) точность и при этом не нужно беспокоиться о переполнении в переменных.

Однако есть и существенный недостаток: вычисления с подобными числами очень громоздки.

В языках высокого уровня экспоненциальный формат с переменным порядком обозначают как "действительный" или "с плавающей точкой". В большинстве случаев мантисса и порядок упакованы вместе со знаком с одном слове.

Решающее значение для точности играет количество разрядов, зарезервированное под мантиссу. При этом оптимальным считается вариант, при котором порядок постоянно изменяется таким образом, чтобы старший разряд мантиссы всегда содержал 1 (выравнивание мантиссы по левому краю). Такой формат является стандартом. Если результат некоторой операции (например, вычитания) не выровнен по левому краю, мантиссу сдвигают влево, а порядок уменьшают до тех пор, пока выравнивание не будет достигнуто.

Впрочем, разработчику на ассемблере совсем не обязательно подражать подобным трюкам. Например, мы охотно используем формат, в котором под абсолютное значение отведен один или два байта, плюс один дополнительный байт под порядок и знак. Несмотря на то, что при таком подходе чрезмерно расточается память, он позволяет затрачивать значительно меньше усилий при организации вычислений.

Кроме того, мы нередко отказываемся от сдвига промежуточного результата влево, если видим, что в дальнейшем его опять придется сдвигать вправо. Также, программист на ассемблере может без проблем перемножать или делить целые числа с экспоненциальными.

Тем не менее, такая свобода выбора стоит программисту многих умственных усилий и значительно повышает опасность допустить ошибку. В языках высокого уровня используются стандартные, гарантированно корректные подпрограммы, а ошибку может допустить только сам разработчик, если будет неправильно обращаться с числовыми форматами.

Экспоненциальный формат с основанием 10

Вместо основания 2 в ассемблере также иногда используют основание 10. Обычно это имеет место в тех случаях, когда ввод и вывод чисел организован в форме с десятичной точкой.

$$\text{Значение} = \text{Мантисса} * 10^{\text{Порядок}}.$$

Для умножения и деления основание не имеет никакого значения. Это становится важно только при сложении или вычитании, поскольку в этом случае вычисления с основанием 10 становятся несколько утомительными.

Правила для экспоненциальных форматов

Теоретически, вычисления с помощью экспоненциальных форматов очень просты. Правила гласят:

- в случае умножения мантиссы перемножаются, а порядки складываются;
- в случае деления частное образуют мантиссы, а порядки вычитаются;
- сложение и вычитание двух чисел возможно только в том случае, если их порядки равны, иначе число с меньшим порядком необходимо делить на 2 (или на 10) до тех пор, пока его порядок не станет равен порядку второго числа.

На практике, мнимая простота умножения и деления оказывается не такой уж и простой, если исходить из того, что результат необходимо постоянно сдвигать влево.

Сложение и вычитание чисел в экспоненциальном формате — также процесс утомительный. Например, в то время как для сложения двух целочисленных слов требуется только шесть команд, для сложения чисел в экспоненциальной форме необходима подпрограмма, которая запросто может состоять из двухсот-трехсот команд. Это же относится и к вычитанию.

При использовании основания 10 вычисления становятся еще более громоздкими, чем в случае основания 2, однако иногда основание 10 имеет преимущества — в частности, если обрабатываются числа, предназначенные для постоянного обмена данными с человеком. Зачастую их получают путем ввода в виде десятичных цифр с точкой или без нее, и в большинстве случаев они в такой же форме должны выводиться. Если при этом с подобными числами не требуется выполнять слишком много дополнительный вычислений, а их порядок почти всегда неизменен, то экспоненциальный формат с основанием 10 — лучший выбор.

7.1.5. Какой же формат избрать?

Выбор формата при работе с числами всегда зависит от конкретной ситуации. Для того чтобы ответить на вопрос о выборе подходящего формата, следует рассмотреть четыре пункта.

- Для чего предназначены эти значения? (Например, для вывода на индикатор или через интерфейс, для передачи в ЦАП, для модуля ШИМ).
- Откуда получают эти значения? (Например, ввод с клавиатуры или через интерфейс, в результате процесса измерения или вычислений).
- Какая требуется точность?
- Какие операции должны выполняться с этими значениями?

Результаты измерений, получаемые с помощью АЦП или расчетов, по своему происхождению — целые положительные числа. Физическое значение измеряемой величины (например, температуры с одним знаком после занятой) зачастую имеет смысл только в том случае, если оно отображается с помощью индикатора или выводится через какой-нибудь интерфейс.

С переменными, которые по своему происхождению — целочисленные, выполняют преимущественно суммирующие операции, например, сравнение с граничным значением, сложение для получения среднего значения, формирование разности для регулирования. В таких случаях использовать экспоненциальный формат — большая глупость, поскольку до выполнения операции сложения или вычитания пришлось бы вначале привести оба операнда к одинаковому порядку, а затем опять выровнять результат по левому краю. Все это подразумевает массу избыточных сдвигов вправо и влево. При этом зачастую нет никакого выигрыша в точности.

С другой стороны, вычисления с целочисленными значениями тоже вызывают множество проблем. Необходимо быть всегда начеку, чтобы не пропустить переполнение в промежуточном результате. Для этого приходится выполнять массу дополнительных проверок, что еще больше повышает риск сделать ошибку.

Особенно внимательным с целочисленными форматами следует быть в том случае, если используется какой-либо язык высокого уровня. Если программист не достаточно хорошо знаком с правилами языка или используемого компилятора, то запросто может допустить серьезный промах, сам того не заметив. Предположим, X и Y — это байты, а S — результат суммы X и Y. Даже если объявить S как слово, полученное значение суммы всегда будет соответствовать только младшему байту,

например, 130 + 130 = 4. Для того чтобы этого избежать, по крайней мере одно слагаемое должно быть загружено в переменную длиной в слово.

Из всего этого можно сделать вывод, что предпочтительнее всегда использовать тип переменных "real". Теперь понятна привычка многих программистов на языках высокого уровня для верности объявлять все переменные в экспоненциальном формате. Мы сами так часто поступаем, когда разрабатываем программу для ПК с использованием языка высокого уровня. Кому нужны лишние проблемы?

Однако в случае с микроконтроллерами ключевую роль играет объем памяти и, что еще важнее, — время вычислений. Излишние команды начинают проявляться в виде неожиданного переполнения памяти программ или некорректной работы программы из-за превышения лимитов времени.

Когда разрабатывается программа на ассемблере, следует просмотреть все операции по отдельности и определить, где следует выбрать другой формат для оптимизации процедур.

Пример

Предположим, нам требуется с помощью некоторого датчика измерить температуру TEMP. Датчик преобразует значение TEMP в напряжение, которое поступает для преобразования на вход 12-тиразрядного АЦП. Выходное значение АЦП обозначим как VOLT. Измеряемая температура лежит в диапазоне 0...60°C, при этом должна быть организована индикация с двумя знаками после запятой.

Значение VOLT, по своей природе, — целочисленное и лежит в диапазоне 0...4095. Значение же TEMP, на первый взгляд, кажется вещественным, поскольку должно быть представлено с двумя знаками после запятой. Тем не менее, если температуру выразить в сотых долях градуса, то также получим целое число в диапазоне 0...6000 (назовем его T).

Взаимосвязь между значениями измерений VOLT и TEMP часто представляют в виде характеристической кривой:

VOLT1 — T1;
VOLT2 — T2;
...
VOLTN — TN.

Примем, что измеренное значение VOLT лежит между значениями VOLT1 и VOLT2. Тогда соответствующее VOLT значение T находим с помощью линейной интерполяции:

T := T1 + Q * (T2 – T1), где Q = (VOLT – VOLT1) / (VOLT2 – VOLT1).

Характеристические значения VOLT1, VOLT2 и т.д. упорядочены по возрастанию. Мы также знаем, что результат выражений VOLT – VOLT1 и VOLT2 – VOLT1 — положительный, и что (VOLT – VOLT1) < (VOLT2 – VOLT1).

Теперь осталось только выяснить, в каком формате представить промежуточную переменную Q (всегда меньше 1). Решение зависит от разрядности максимального абсолютного значения разности T2 – T1. Произведение Q * (T2 – T1) должно иметь точность до одного разряда, то есть, погрешность не должна превышать ±0,5.

Если T2 – T1 всегда меньше 256, то для Q можно приближенно выбрать формат Q = MQ/256. В случае, если VOLT2 – VOLT1 также меньше 256, то Q можно получить путем простого деления байта на байт.

Произведение Q * (T2 – T1) получаем путем простого умножения байта на байт. Для этого перемножаем MQ и (T2 – T1). Из двухбайтного результата нам требуется

только старший байт, поскольку еще нужно выполнить деление на 256. Младший байт результата используем для округления.

В самом лучшем случае вся интерполяция займет менее 200 команд, в случае же применения стандартного экспоненциального формата это время могло запросто увеличиться в десять раз. Основная часть излишних команд приходится на ненужные сдвиги и сложения байтов, имеющих нулевое значение.

На планирование таблицы характеристических значений часто влияет величина VOLT2 – VOLT1 (или же T2 – T1). Если время интерполяции критично, то рационально поместить в такую таблицу побольше значений.

7.1.6. Точность

Совершенно ясно, что высоту колокольни выражать в миллиметрах — глупо. Однако, далеко не так очевидно, что значение измерения, полученное, к примеру, в виде байта на выходе АЦП, не должно получиться четырехразрядным. И здесь не поможет даже вычисление с помощью какого-нибудь сложного алгоритма.

В пользу повышения точности, конечно же, больше доводов, чем в пользу ее снижения, однако из-за мнимой точности также иногда возникают проблемы. Покажем на простом примере, что может произойти, если слишком увлечься повышением точности индикации при измерениях.

Пусть на вход восьмиразрядного АЦП подается напряжение в диапазоне 0...5 В, значение которого должно отображаться с двумя знаками после запятой. В результате получаем 256 возможных значений измерения при 501 возможном значении индикации (от 0,00 до 5,00). На каждое из 256 значений измерения приходится одно значение индикации, но не наоборот.

Это не представляет проблем, если согласиться на "прыжки" индикации (например, с 1,96 на 1,98 В) с пропуском промежуточных значений (в данном случае — 1,97 В), как бы пользователь ни крутил ручку входного потенциометра. Однако можно не сомневаться, что пользователь это заметит и расценит как дефект. Пользователь может остаться доволен только в том случае, если результат всегда округляется таким образом, что последняя цифра — 0 или 5.

Похожая проблема возникает, если пользователю разрешено задавать числа до 99,999, которые, однако, внутренне требуется представить только 16-ю разрядами, поскольку более высокая точность нерациональна. Если пользователь считает только что введенное им значение, то в результате округления может увидеть отличие в последнем разряде, каким бы точным ни был алгоритм обратного счета. Особенно нервирует пользователя, когда, например, вместо введенного значения 30,000 он видит 29,999.

Если речь идет об обработке одиночного значения, то, возможно, стоит подумать об использовании 24-х разрядов, но в случае табличных данных, вероятно, более рациональное решение — изменить формат ввода.

7.2. Функции

Пожалуй, чаще всего в работе используются такие функции как синус, косинус и логарифм. К этому списку можно еще добавить получение обратного значения.

Время обращения к функциям, используемым в таких областях как управление двигателями и робототехника, должно быть максимально малым, потому большие алгоритмы в расчет не принимаются.

До сих пор мы получали значения функций исключительно с помощью таблиц. Если микроконтроллер обладает достаточным объемом памяти для того, чтобы разместить 256 или более табличных значений, то, в зависимости от требований, предъявляемых к точности значений функции, во многих случаях от интерполяции можно отказаться. При вычислении синуса для управления двигателем иногда достаточно 64-х или даже меньше табличных значений.

В отсутствие интерполяции, для чтения значений функции требуется всего лишь несколько строк программного кода, однако даже с интерполяцией этот метод работает быстрее, чем любой математический алгоритм.

Иногда с помощью таблиц рационально выполнять даже умножение и деление.

7.3. Использование макросов

Макрос — это заранее подготовленный блок программных строк. Каждый макрос имеет определенное имя, по которому соответствующий блок вызывается в коде программы.

Таким образом, в случае макросов речь идет, по сути дела, только о способах написания, которые, тем не менее, играют большое значение для хорошо структурированного программирования.

Для использования макросов есть разные причины. В любом случае, они упрощают процесс программирования и уменьшают вероятность ошибки. Кроме того, исходный код становится более гибким и удобочитаемым.

В следующем примере представлен макрос с именем ZUENDEN:

```
ZUENDEN  MACRO
         BSF     PORTB,4
         NOP
         BCF     PORTB,4
         ENDM
```

Теперь в тексте программы на три представленные выше строки можно ссылаться по имени ZUENDEN. Таким образом, можно сказать о "вызове" макроса, хотя это понятие нельзя путать с вызовом подпрограмм (CALL).

Какие же преимущества дает нам использование макросов? Если бы представленные выше три строки кода были явно вставлены в исходный код программы, то в поясняющей документации пришлось бы пояснять, для чего они предназначены. В отличие от этого, благодаря имени ZUENDEN, назначение макросу сразу же становится очевидным (в переводе с немецкого "zuenden" означает "завершить" — прим. переводчика). Более того, хороший стиль программирования подразумевает отсутствие подробностей доступа к аппаратным средствам непосредственно в тексте программы — они относятся к части объявлений. В случае каких-либо изменений в аппаратной среде, будет достаточно модифицировать только определения.

Должен ли программист дополнительно с помощью директивы DEFINE явно задать имя для вывода PORTB,4 — дело вкуса и зависит от дальнейших обстоятельств.

Все удобство использования макросов покажем еще на одном примере:

```
ADD1     MACRO   DEST,SRC
         MOVF    SRC,W
         ADDWF   DEST
         ENDM
```

Примечание

Аргументы DEST и SRC — фиктивные. При вызове макроса вместо них можно подставить любые переменные.

На основании вызова ADD1 Y,X ассемблер выдаст две строки кода:

```
MOVF    X,W
ADDWF   Y
```

Настоящая польза этого макроса становится очевидна только тогда, когда он используется для сложения двухбайтных переменных. Предположим, первая из таких переменных называется SRC, а вторая — DEST. Мы всегда оговариваем, что старший байт двухбайтной переменной всегда следует сразу же после адреса младшего байта. Следовательно, старшие байты можно обозначить как SRC+1 и DEST+1.

```
ADD2    MACRO   DEST,SRC
        MOVF    SRC,W
        ADDWF   DEST
        SKPNC
        INCF    DEST+1
        MOVF    SRC+1,W
        ADDWF   DEST+1
        ENDM
```

Макрос ADD2 состоит из шести строк. Его вызов для сложения двух переменных выглядит следующим образом: ADD2 Y,X. Таким образом, мы не только экономим пять строк кода, но также делаем значительный вклад в наглядность программы. Макрос ADD2 находится в макробиблиотеке. Он выверен, и избавляет от опасности опечаток в тексте.

Мнение авторов: громоздкие операции с многобайтными переменными следует создавать без использования макросов только в том случае, если на это есть очень веские основания.

Допускается вложение макросов, как показано в следующем примере:

```
ADD2    MACRO   DEST,SRC
        ADD1    DEST,SRC
        SKPNC
        INCF    DEST+1
        ADD1    DEST+1,SRC+1
        ENDM
```

Здесь внутри определения макроса ADD2 используется вызов макроса ADD1.

Еще один распространенный класс макросов — макросы перехода. Например:

```
BGE     MACRO   X,Y,LABEL
        MOVF    Y,W
        SUBWF   X,W
        SKPNC
        GOTO    LABEL
        ENDM
```

С помощью четырех строк этого макроса запрограммирован условный переход (если X больше или равно Y, то выполняется команда GOTO LABEL).

Этот класс макросов снимает излишнее умственное напряжение и помогает избежать ошибок. Мы подобные макросы используем редко, поскольку их преимущества, на наш взгляд, выглядят не достаточно убедительно. Тем не менее, в своей макробиблиотеке у нас есть различные макросы сравнения, которые очень удобны при организации условных переходов.

Примем, что X и Y — двухбайтные переменные. В этом случае макрос BGE превращается в "двухбайтный" вариант BGE2. Его вызов BGE2 X,Y,Label в одном из случаев приведет к переходу к метке Label. В подобных случаях мы применяем макрос CMP2 собственной разработки. Впрочем, он не настолько краток в процессе ветвления как макрос, представленный выше:

```
CMP2  X,Y
SKPNC
GOTO  LABEL
```

Макросы, как правило, должны объявляться, однако существует ряд макросов, предлагаемых макроассемблером MPASM. К ним относятся такие важные макросы как BANKSEL и PAGESEL, которые избавляют программиста от необходимости постоянно контролировать размещение данных и программ в памяти. **В среде MPASM эти макросы определены как директивы ассемблера, однако они — предопределенные макросы с аргументом ADRESS.**

Между тем, от некоторых макросов, обозначенных как "специальные директивы", компания Microchip отказалась, о чем мы сожалеем, поскольку они позволяли значительно повысить наглядность программы и избежать многих ошибок.

В первую очередь, это относится к однокомандным макросам, наподобие тем, которые представлены в табл. 7.1

Таблица 7.1. Примеры однокомандных макросов

Команда	*Макрос (способ сокращенной записи)*
BTFSS STATUS,Z	SKPZ
BSF STATUS,C	SETC

Еще один макрос специальных функций состоит их двух строк (табл. 7.2).

Таблица 7.2. Макрос BZ

Команда	*Макрос (способ сокращенной записи)*
BTFSC STATUS,Z GOTO LABEL	BZ LABEL

В PIC18 условный переход BZ реализован в виде команды, что объясняет, почему этот макрос больше не может поддерживаться ассемблером для PIC18. Да мы и сами последние годы стараемся избегать макросов, состоящих из двухсловных команд.

Тем не менее, сокращенные способы записи SKPZ, SETC и пр. в ассемблере для PIC18 также больше не поддерживаются (см. раздел 3.11). Мы находим это досадным, поскольку больше не можем использовать сокращения, вошедшие за много лет применения в привычку (точнее сказать, их в PIC18 можно использовать, но только приходится определять их самому).

Смысл использования макросов

В предыдущем подразделе было отмечено, что для использования макросов существуют различные основания.

Так, макрос ZUENDEN служит в качестве оболочки для определенной пользователем функции, на которую наглядно указывает его имя. В результате определение функции находится не в самой программе, а в части объявлений. В случае изменения аппаратных средств потребуется внести лишь незначительные корректировки в макрос SUENDEN.

Совет

Оформляйте все аппаратные функции в виде макросов, даже если они занимают только одну строку кода.

Во всем остальном, макросы — это просто средство выделить часто повторяющиеся операции, и их цель — сделать программирование удобнее, а программы — более наглядными.

При активном использовании макросов можно получить своеобразный "доморощенный" язык высокого уровня. В этом есть как свои преимущества, так и недостатки.

К сожалению, в ассемблере, поддерживаемым MPLAB, не предусмотрено стандартных полезных макросов. И проблема не в том, что приходится их писать вручную, а в несовместимости с остальным миром. Именно поэтому мы отказались от примеров применения макросов перехода.

С другой стороны, макросы, наподобие ADD2, мы считаем практически незаменимыми. Временные затраты на явное описание подобных повторяющихся операций были бы неоправданно большими, а тексты программ — длинными и неудобочитаемыми. Кроме того, непосвященному пользователю строку ADD2 X,Y понять гораздо проще, чем шесть строк непосредственной реализации операции.

Единственный недостаток заключается в том, что реальную длительность программы больше нельзя определить простым подсчетом количества строк — следует подсчитывать и число строк, соответствующее макросам.

Обобщим преимущества использования макросов:

- с помощью макросов создается оболочка для важных небольших модулей;
- с помощью макросов программа становится более гибкой;
- благодаря макросам, исходный код становится более наглядным;
- макросы упрощают процесс программирования;
- при использовании макросов уменьшается вероятность ошибок.

Разумеется, у макросов есть и недостатки:

- сопровождение макросов требует затрат времени;
- слишком большое число макросов может затруднить удобочитаемость программы.

Когда проект создается группой разработчиков, соглашения по использованию макросов должны быть оговорены особо тщательно.

Макрос по сравнению с подпрограммой

Мы бы хотели еще раз провести различие между макросом и подпрограммой, поскольку нам часто задают вопрос о том, какой из этих двух методов лучше использовать. На этот вопрос нельзя дать однозначный ответ.

Как макросы, так и подпрограммы представляют собой некоторое количество заранее подготовленных строк кода, которые программист может при желании "вызывать". Однако слово "вызывать" в этих двух случаях имеет разный смысл.

При вызове макроса на этапе ассемблирования все его строки явно записываются в HEX-файл. Однако разным вызовам в программе могут соответствовать разные последовательности команд, поскольку ассемблер подставляет вместо фиктивных фактические аргументы. Также локальные метки внутри макроса содержат фактические адреса.

При вызове подпрограммы в HEX-файл помещается только команда CALL. Таким образом, благодаря использованию подпрограмм, зачастую экономится много места в памяти программ, поскольку строки подпрограммы записываются в память только один раз.

Тем не менее, недостаток подпрограмм заключается в том, что в них могут использоваться только четко определенные переменные, подставляемые при каждом вызове, и никаких фиктивных.

Если бы мы записали макрос ADD2 в виде подпрограммы, то предварительно должны были бы загрузить в рабочие переменные подпрограммы те значения, которые хотим сложить. Затем, после вызова подпрограммы результат сложения должен также быть загружен в соответствующую переменную. На все эти "загрузки" требуется больше команд, чем экономится в результате использования подпрограммы.

Если в программе часто вызывается какой-либо длинный макрос, то имеет смысл подумать, не лучше ли оформить соответствующие строки кода в виде подпрограммы (при этом, конечно же, предполагается, что выигрыш в количестве команд будет превышать потери в результате необходимых загрузок переменных).

В программах, критичных ко времени, следует принять во внимание еще одно обстоятельство: команды CALL и RETURN требуют в сумме четыре командных цикла, что особенно может сказаться при вызове подпрограмм внутри циклов.

Внимание!

Хотим предупредить еще об одном аспекте. Вызов макроса выглядит как одна команда, и потому может возникнуть искушение пропустить его с помощью команды "SKIP". Если макрос состоит из нескольких строк, то по команде "SKIP" будет "перепрыгнута" только первая макрокоманда, и текущей станет вторая. Даже если программист и знает об этой "западне", он все равно может по оплошности в нее угодить. Также не следует "перепрыгивать" с помощью команды "SKIP" и однострочные макросы. Если когда-нибудь макрос будет преобразован в многострочный, то это в последствии может привести к странным и труднообъяснимым проявлениям программы.

7.4. Структура программы

Программы для микроконтроллеров PIC обычно имеют назначение, отличное от назначения программ для ПК, и потому имеют другую структуру. Тем не менее, основные принципы остаются неизменными: модульность и объектно-ориентированное программирование, при соответствующих модификациях, присущи также и программам для PIC, а хорошее планирование структуры программы — обязательное условие для успеха проекта.

В процессе разработки проекта список требований, предъявляемых к программе, почти всегда расширяется. Постановку задачи приходится дополнять или изменять, а некоторые части — вообще удалять.

Каждый разработчик придерживается собственного подхода к программированию. У каждого, подобно художнику, есть собственный стиль и излюбленные методы. Не существует какого-либо стандартного метода построения программ, точно

так же как не существует стандартного метода рисования картин. Тем не менее, существует ряд принципов, которых следует придерживаться в любом случае!

Мы хотим здесь изложить сугубо наш личный взгляд, сформированный на основании большого опыта разработки самых разнообразных проектов для микроконтроллеров PIC. Хотим отметить, что представленные ниже рассуждения несколько упрощены и обобщены и в отдельных случаях, конечно же, могут не соответствовать действительности.

Для нас всегда важную роль при разработке программ играют временные затраты, потому первый аспект, который следует рассматривать при планировании программы, — решаемые задачи. С чего начинать в первую очередь? Каких временных рамок придерживаться? При множестве задач необходимо также ответить на вопрос: какова их длительность и не должны ли некоторые из них быть разбиты на подзадачи?

Время — не только важный параметр практически во всех физических процессах, но и потребляемый программой ресурс.

7.5. Модульное программирование

Если сделать общий обзор всего множества задач, то, в большинстве случаев, можно сразу же увидеть распределение всей программы по модулям. Мы на своем богатом опыте убедились, что модули формируются на основании не какой-то формальной точки зрения, а поставленных задач. Например, программа может состоять из модуля управления двигателем, интерфейсного модуля, модуля контроля за питанием, модуля оповещений и модуля формирования временных интервалов.

Понятие модуля в нашем случае применения не обязательно совпадает с понятием "подключаемого модуля", ассемблируемого отдельно. В самом подключаемом модуле вполне может быть заключено множество логических модулей.

Большие модули иногда могут быть разбиты еще на несколько модулей. Если, к примеру, разрабатывается устройство, обслуживающее меню, то функцию обслуживания меню рационально оформить в виде отдельного модуля, который должен быть обособлен от работы устройства или интерфейсов. Обслуживание сложных меню, опять таки, разбивается на модули. Если необходимо обслуживать больше сотни подменю, то их целесообразно разбить на смысловые категории.

Не каждая часть программы является модулем. Модуль всегда должен содержать только изолированные программные элементы. Каждый, кто имеет опыт оформления программ, хорошо понимает, что прямые переходы изнутри одного модуля внутрь другого крайне нежелательны. Это означает, что каждый раз после передачи модулем управления должен происходить возврат в главный цикл.

Если блок-схема программы выглядит причудливо, то имеет смысл подумать о том, не лучше ли разбить ее на модули.

7.5.1. Полномочия модулей

У каждого модуля есть строго определенные полномочия. К примеру, модуль контроля силы тока отвечает исключительно за опрос и оценку силы тока. Если он обнаруживает значительную перегрузку, то ни в коем случае не должен отключать двигатель или включать сигнализацию — это не его задача. Он просто передает сообщения модулю управления двигателем и модулю уведомлений о том, что они должны соответствующим образом отреагировать. Существует совсем немного си-

туаций, в которых недопустимо запаздывание даже в несколько миллисекунд (например, исчезновение напряжения питания).

Для модуля управления двигателем могут быть различные основания для отключения двигателя, одним из которых является и перегрузка по току. Однако это может произойти и по нажатию кнопки или по сигналу от концевого переключателя. Модуль управления собирает всю информацию, которая ему передается из других модулей, и затем принимает решение о дальнейших действиях. В свою очередь, он "заведует" флагами уведомлений (например, "двигатель включен"), с помощью которых извещает другие модули о том, работает ли двигатель. Иногда даже целесообразно применить еще несколько дополнительных флагов для уведомления причин отключения двигателя. Например, на всякий случай, всегда следует проверять, что двигатель полностью открыл двери гаража, а не отключился из-за перегрузки по току из-за того, что двери заклинило.

Тем не менее, определять слишком много флагов уведомления "про запас" (так сказать, "на всякий случай") не целесообразно. Мы добавляем такие флаги в систему уведомлений только в том случае, если в этом есть реальная необходимость, и затем сразу же документируем их назначение.

К компетенции модулей также относится доступ к аппаратной части и доступ к переменным на запись. Таким образом, каждая переменная, как правило, "привязана" к какому-то конкретному модулю.

Существует множество практических причин, по которым в программах для микроконтроллеров PIC иногда отклоняются от строгих правил модульного программирования. Зачастую просто неизбежно или целесообразно, чтобы какой-либо модуль выполнял функции, которые, по сути, относятся к компетенции других модулей.

Одним из важных оснований для исключения из правил могут стать ограниченные ресурсы. В случае микроконтроллеров PIC первого поколения приходится идти на многие компромиссы из-за скудной памяти и недостаточной глубины стека.

Но даже в развитых микроконтроллерах PIC мы иногда отклоняемся от строгого курса, если этого требует какая-либо затруднительная процедура или ухудшается удобочитаемость программы, или же в крайних случаях, наподобие исчезновения напряжения питания.

Подобные методики исключений следует применять только тогда, когда это приносит явную пользу, а не просто из-за лени или стресса. И вот наш совет: как следует их документируйте.

Существуют модули, которые предоставляют услуги для других модулей. Типичный пример — модуль организации временных интервалов. Его задача — организовать глобальную временную структуру. Этот модуль вполне может взять на себя отсчет тайм-аутов для других модулей.

В случае подобного "привлечения внешних ресурсов" очень важно точно сформулировать задачу. Например, иногда от временного модуля требуется только декремент счетной переменной (если она не равна нулю). Иногда бывает так, что такой модуль уполномочивают устанавливать некоторый флаг, как только переменная достигает значения нуля. Это, по-прежнему, — в его компетенции. Однако порой временной модуль даже считают вполне пригодным выполнить после отсчета времени какую-нибудь небольшую задачу, которая, по сути дела, не имеет к нему никакого отношения. Это, конечно же, не отвечает правилам и может иметь место только в хорошо обоснованных случаях.

Если допускается слишком много исключений из правил, то можно запросто лишиться преимуществ, достигаемых при хорошей организации программы.

Если программа небольшая, то на исключения можно согласиться с большей смелостью. Многие большие программы можно сделать поменьше. Вначале можно немного видоизменить разросшуюся структуру, затем "урезать" время, пока, наконец, внести изменения без значительных затрат не станет невыполнимо.

7.5.2. Переменные

Планирование структуры переменных в больших проектах играет не менее важную роль, чем планирование структуры программы. Лучше сказать, планирование переменных шагает с планированием программы рука об руку.

Переменные могут содержать различные категории информации, как то: состояние программы, результаты аналоговых измерений или показания времени. С их помощью можно передавать уведомления из одного модуля в другой или использовать их только внутри одного определенного модуля.

Если разработчик программ хочет упростить себе жизнь, то он должен хорошенько обдумать структуру переменных, а результат этих размышлений прилежно задокументировать.

Подобное документирование мы считаем важнейшей частью в описании программы. Усилия, затраченные на комментирование распределения переменных по модулям, в дальнейшем окупятся сторицей. Для каждой переменной должно быть дано понятное описание ее назначения и указано, из каких модулей к ней обращаются на чтение и запись.

Большинство переменных привязаны к какому-то конкретному модулю в том смысле, что этот модуль отвечает за изменение "своих" переменных. Но из этого правила есть два исключения.

Рабочие и вспомогательные переменные (например, счетчики или операнды арифметических подпрограмм) предназначены для использования всеми модулями. В случае применения языка высокого уровня программист о существовании таких переменных даже не догадывается, но в программировании на ассемблере они доступны для общего применения. С определенной мерой осторожности, их можно также применять и за пределами предусмотренных операций.

Еще одно исключение составляют переменные уведомлений, которые обычно в одном модуле инициализируются, а в другом — обнуляются. Типичный пример такой переменной — хранение "состояния ошибок", наподобие уведомлений о перегреве, падении напряжения или сбое при обмене данными. Разряды переменных уведомлений могут устанавливаться и сбрасываться в различных модулях. Иногда состояние ошибок считывается многими модулями и управляет всем ходом выполнения программы, поэтому такие переменные не могут быть явно прикреплены к какому-то конкретному модулю.

7.5.3. Флаги

Понятие "флаг" мы относим к разрядам рабочего регистра, представляющего собой особую переменную, которая служит для указания некоторого двоичного состояния или для передачи уведомлений между модулями.

Все множество уведомлений программы должно всегда составлять логически последовательную систему. Уведомления, которые недостаточно документированы, зачастую оказываются непродуманными, а иногда — даже излишними. Ненужные уведомления снижают удобочитаемость программы и заставляют делать лишнюю работу.

Построить наглядную систему уведомлений — не такое уж и простое дело. Однако, как показывает рассмотренный ниже небольшой пример, если применить последовательный системный подход, то никакой "каши" в уведомлениях не возникнет.

Предположим, некоторое устройство должно выключаться, если измеренная температура превышает пороговое значение T1. Тем не менее, обычно устройство выключают только тогда, когда превышение температуры удерживается на протяжении какого-то определенного времени. Иногда постановка задачи разрешает опять включить устройство, если через некоторое время температура опять придет в норму. Зачастую для этого определяют гистерезис. Это означает, что устройство может быть опять включено, если температура меньше T0, причем T0 < T1.

Опыт показывает, что неопытные разработчики для подобных процессов используют много переменных и флагов, мы же опишем состояние температуры с помощью двух флагов и одной переменной-счетчика.

Для того чтобы распознать текущее состояние "перегрева", обязательно необходим флаг, который мы назовем `temp_ov`. Если возникает изменение состояния, мы будем загружать в переменную (`tempcnt`) некоторое заданное значение. Прежде всего, будем исходить из того, что это значение при изменении температуры с нормальной на повышенную имеет величину, отличную от той, которая должна применяться при изменении в обратную сторону, однако в обоих случаях речь идет о величине одного порядка.

Так, с помощью переменных `temp_ov` и `tempcnt` мы уже полностью описали состояние температуры. До тех пор, пока `tempcnt` <> 0, состояние характеризуется с помощью `temp_ov` — еще без релевантности. Зачастую бывает удобно ввести дополнительный флаг `temp_over`, уведомляющий о том, является ли превышение температуры релевантным. Этот разряд избыточен, поскольку его значение в любой момент можно узнать по значению `temp_ov` и `tempcnt` (табл. 7.3)

Таблица 7.3. Зависимость значения `temp_over` от `temp_ov` и `tempcnt`

temp_ov	tempcnt	temp_over
0	0	0
0	<>0	1
1	0	1
1	<>0	0

Иногда требуется или целесообразно ввести не только флаги для **состояний**, но также и для **событий**. События — это изменения состояний. В качестве примера можно привести ситуацию, когда после длительного перегрева температура опять на достаточно долгий период становится нормальной (в таком случае флаг будет установлен в момент, когда `temp_ov=1`, а для `temp_cnt` начинается обратный отсчет).

Зачастую, вводят дополнительный разряд уведомления, который извещает модуль формирования временных интервалов о начале отсчета задержки. Если выполняется обратный отсчет времени, то этот флаг почти всегда избыточен. Пока счетная переменная не равна нулю, модуль всегда выполняет подсчет.

Одна из наиболее распространенных ошибок, допускаемых новичками, заключается в том, что флаги не достаточно хорошо продуманы. В большинстве случаев определяют слишком много разрядов, что в случае большой и плохо документированной системы уведомлений доставляет массу хлопот.

К документированию флагов, в первую очередь, относится, в каком модуле они устанавливаются, в каких модулях используются, и где сбрасываются. Иногда также

необходимо задокументировать, для чего требуется тот или иной флаг! Часто также бывает полезно описать логическую взаимосвязь переменных, как, например, это было сделано в табл. 7.3.

Выше мы определили два флага различных категорий. Флаг `temp_ov` — внутренний разряд состояния, который в нашей постановке задачи используется только внутри модуля температуры для хранения предыдущего состояния. В отличие от него, флаг `temp_over` предназначен для различных модулей, например, для модуля ввода-вывода.

Еще один важный аспект флагов — срок их действия. Так, флаг некоторого состояния "живет" до тех пор, пока это состояние не изменилось. Иначе обстоит дело с флагом события, который устанавливается только в момент возникновения события и сбрасывается сразу же, как только уведомление больше не нужно (например, если все модули уже получили сообщение). Если уведомления ожидают несколько модулей, то существует вероятность возникновения ошибок. С одной стороны, все модули должны получить сообщение до сброса флага, но с другой — ни один модуль не может получить уведомление дважды. Кроме того, флаг должен быть сброшен до того, как событие повторится.

7.6. Регистрация событий

Когда речь идет о событиях применительно к микроконтроллерам PIC, то подразумеваются следующие основные категории:

- цифровые события ввода-вывода (фронты, импульсы);
- аналоговые события (АЦП, компаратор);
- события последовательной передачи данных (USART, SSP);
- события времени (таймер);
- события счета (счетчик).

Таким образом, регистрация события — это опрос на предмет того, **возникло** ли уже это событие, или, в некоторых случаях, — **ожидание** его возникновения. Для каждого события существует подпрограмма обслуживания, которая должна быть выполнена как можно скорее после его возникновения.

Услышав слово "событие", многие автоматически вспоминают о прерываниях, однако, в большинстве случаев, не всегда возможно препоручить все множество событий прерываниям.

В зависимости от случая применения, существуют различные требования к регистрации событий, например:

- события должны быть зарегистрированы до того как исчезнут (пример — импульс);
- события должны быть обслужены достаточно быстро.

"Достаточно быстро" в каждом конкретном случае означает разное время, что зависит как от самого события, так и от требуемого для него обслуживания. Например, нажатие кнопки должно быть зарегистрировано до того как кнопка будет отпущена, чему соответствует время в несколько миллисекунд. С другой стороны, для каждого шага в случае шагового двигателя необходимо уложиться в микросекунды после выдержки приличного промежутка времени.

Существует три методики регистрации событий:

- использование прерываний, с помощью которых одновременно реализуют и обслуживание;

- постоянные опросы флагов прерываний;
- постоянные опросы событий.

Постоянные опросы не могут похвастаться большой популярностью — особенно опросы событий, даже если это — самое удобное решение. Дело в том, что он требует обзора всего хода выполнения программы, и при этом между опросами должно быть достаточно времени для выполнения остальной программы.

Иногда требуется точно знать, сколько центральному процессору требуется времени на выполнение различных частей программы, и через какие интервалы будут достигнуты те или иные программные фрагменты. Вот почему полностью асинхронной структуре программы мы предпочитаем использование циклов ожидания или тактирования (см. ниже).

Если в нашем распоряжении есть прерывания, но вполне можно воспользоваться их флагами без разрешения самих прерываний. Флаги уведомляют о наступлении некоторого события. Особенно они популярны в модуле захвата.

И все же, иногда без прерываний не обойтись. Использование подпрограмм обработки прерываний рационально в тех случаях, когда без них был бы нанесен значительный ущерб точности или удобству.

7.6.1. Постоянные опросы

Опрос событий возможен только тогда, когда промежуток времени между двумя опросами достаточно велик для решения остальных задач. Для этого большие задачи зачастую приходится разбивать на меньшие подзадачи.

Само собой, постоянные опросы событий накладывают ограничения. Например, если короткие импульсы возникают без какой-либо регулярности в промежутках между ними, то программа будет состоять почти из одних только опросов.

Опрос флагов прерываний допускает гораздо больше времени между опросами, поскольку флаги хранят информацию до тех, пока она не будет использована, и только потом сбрасываются. При этом следует только обращать внимание на то, чтобы флаг был опрошен до того, как событие возникнет еще раз.

В былые времена такой подход использовался для реализации последовательных протоколов (USART, I²C), однако теперь аппаратные модули, по большей части, избавляют нас от подобной работы. Тем не менее, существуют важные практические случаи, когда последовательные протоколы должны обрабатываться программно (например, при реализации узлов LIN с помощью микроконтроллеров PIC базовой серии).

7.6.2. События времени

События времени, в основном, заключаются в том, что таймер или принимает определенное значение, или находится в некотором заданном интервале. В связи с этим, существуют две задачи: ожидание наступления события времени и проверка на предмет наступления такого события.

Переменную, задающую момент времени, мы часто называем EVENT. Представленный ниже цикл WAITIM ожидает появления события TMR0=EVENT:

```
WAITIM   MOVF  EVENT,W
         SUBWF TMR0,W        ; W := TMR0 - EVENT
         SKPZ                ; Выходим из цикла ожидания, если 0
         GOTO  WAITIM
WEITER   ...
```

Коэффициент деления для TMR0 должен составлять как минимум 8, иначе в результате опроса момент времени TMR0 = EVENT распознается случайным образом, и позиция WEITER достигается за три-восемь командных циклов до этого момента.

Вместо того, чтобы требовать точного равенства выражения TMR0 – EVENT нулю, мы великодушно разрешаем, чтобы три младших разряда разности отличались от нуля. Тогда цикл ожидания принимает следующий вид:

```
WAITIM   MOVF  EVENT,W
         SUBWF TMR0,W       ; W = TMR0 - EVENT
         ANDLW 0F8H         ; Маскируем три разряда
         SKPZ               ; Если (TMR0 - EVENT) = 0...7
         GOTO  WAITIM
WEITER   MOVLW .125         ; Настраиваем EVENT
         ADDWF EVENT        ;  на следующий момент времени
```

Этот опрос работает даже тогда, когда, например, EVENT = 255, а к моменту выполнения команды SUBWF таймер уже переполнен и, возможно, имеет значение 3.

В связи с этим, коэффициент деления 1 имеет мало практического смысла. В конкретных случаях применения мы часто используем делитель 32. Кроме того, мы маскируем не только три, но, например, шесть разрядов и в результате с помощью представленного выше опроса получаем период не от 0...7, а от 0...63. Этот период занимает 32*64 командных цикла, что при частоте осциллятора в 4 МГц составляет более 2 мс.

Преимущество такого подхода заключается в том, что мы распознаем событие времени еще и в том случае, когда опрос однократно (как исключение) происходит слишком поздно. Для распознания временного интервала после события TMR0 = EVENT есть промежуток в более, чем 2000 команд. Это важно в случае тактированных циклов (см. ниже).

Представленная ниже подпрограмма WATCHTIM служит для того, чтобы подпрограмма времени ZEITDA вызывалась в точности один раз за каждые 4 мс. Для этого в подпрограмме ZEITDA значение переменной EVENT увеличивается на 125.

```
WATCHTIM  MOVF  EVENT,W
          SUBWF TMR0,W      ; W = TMR0 - EVENT
          ANDLW 0C0H        ; Маскируем 6 разрядов
          SKPNZ             ; Если (TMR0 - EVENT) = 0...63
          CALL  ZEITDA
          RETURN
```

В данном случае, для того чтобы не пропустить интервал TMR0 – EVENT = 0...63, подпрограмма WATCHTIM должна вызываться как минимум каждые 2 мс.

То, что подпрограмма ZEITDA не вызывается в точности с одинаковым промежутком в 4 мс, во многих случаях применения не играет роли. Важно только то, чтобы она наверняка вызывалась **один раз в пределах каждого интервала 4 мс**.

Несмотря на свою видимую неточность, подобные опросы применимы даже в очень точных часах, поскольку неточность не накапливается.

Предположим, у нас есть цифровые часы, отображающие часы, минуты и секунды. Внутренне в них используется счетчик, который за одну секунду увеличивается 250 раз (то есть, каждые 4 мс). Для стороннего наблюдателя важно только то, чтобы этот счетчик увеличивался за секунду 250 раз — точный же момент времени, когда это произойдет, его не интересует. Такие часы и после года работы будут выдавать очень точные показания (соответствуют точности осциллятора). То, что счет-

чик секунд в отдельных случаях запаздывает на 2 мс, наблюдатель гарантированно не заметит.

Совершенно другая постановка задачи возникает в том случае, если необходимо регистрировать точные временные промежутки. Тогда подпрограмма WAITIM зачастую оказывается недостаточно точной. Само собой, было бы удобно воспользоваться модулем сравнения, однако, если это невозможно по причине недостаточного финансирования, существует такой вариант как цикл ожидания. Например, следующий цикл ожидания длится 3*WERT+1 командных цикла.

```
WAIT    MOVLW  WERT
        MOVWF  COUNT
WLOOP   DECFSZ COUNT
        GOTO   WLOOP
```

Если этот цикл оформить в виде подпрограммы, то его длительность увеличится на четыре командных цикла и будет составлять 3*WERT+4, поскольку для команд CALL и RETURN требуется по два командных цикла.

Один из важных случаев применения точной временной регистрации без прерываний — это последовательные интерфейсы, которые, из соображений экономии, приходится программировать без применения аппаратных модулей. В таком случае между моментами опроса или передачи бита должны быть точные промежутки времени.

Тем не менее, если длительность цикла задержки — величина переменная, которая на момент разработки программы еще неизвестна, то приходится прибегать ко множеству уловок. Типичный случай, когда может возникнуть подобная ситуация, — прием по протоколу LIN (пример соответствующей подпрограммы WARTE см. в разделе 2.5).

7.6.3. Ожидание фронта

Если мы, к примеру, ожидаем появления отрицательного фронта сигнала на некотором выводе INPU, то цикл ожидания может выглядеть следующим образом:

```
WAILO   BTFSC  INPU
        GOTO   WAILO
WEITER
```

Поскольку длительность цикла составляет три командных цикла, до позиции WEITER после появления фронта проходит от двух до пяти командных циклов. Проблема представленного выше цикла заключается в том, что в случае пропуска фронта он становится бесконечным (зацикливание). В качестве альтернативы можно воспользоваться счетной переменной TIMO:

```
WAILO   BTFSS  INPU
        GOTO   WEITER
        DECFSZ TIMO
        GOTO   WAILO
        GOTO   TIMEOUT      ; Подпрограмма обработки ошибки
WEITER
```

Между двумя опросами состояния входа INPU теперь расположено пять циклов, и после появления фронта до позиции WEITER проходит от трех до восьми циклов. Если это неприемлемо, то можно увеличить частоту осциллятора или выбрать мик-

роконтроллер PIC, поддерживающий прерывания. Если для реализации точного (короткого) цикла ожидания требуется использовать именно недорогого представителя базовой серии микроконтроллеров PIC, то, конечно же, возможен и трюк, подобный представленному ниже:

```
        BTFSS INPU
        GOTO  WEITER
        BTFSS INPU
        GOTO  WEITER
        BTFSS INPU
        GOTO  WEITER
        ; и т.д.
TIMEOUT
```

7.6.4. Регистрация по прерыванию

Регистрация событий по прерыванию, на первый взгляд, выглядит удобной — не нужно беспокоиться о том, что можно что-то пропустить, — тем не менее, у этого метода также есть как свои преимущества, так и недостатки. Однако прежде, чем обсудить их, рассмотрим три типичных примера.

Последовательный обмен данными

Если для обмена данными с интерфейсами USART или SPI применяется аппаратный модуль, то используют удобный метод, согласно которому соответствующая подпрограмма обработки прерывания вызывается только тогда, когда в буфер записывается передаваемый байт. В случае обнаружения знака "конец сообщения" (например, символа $0d_h$), следующего в конце принятого байта, главная программа с помощью программного флага извещается о том, что в буфере находится готовое сообщение. В дальнейшем главная программа может опрашивать и использовать этот программный флаг, сколько ей будет угодно.

Иногда сообщение может также полностью обрабатываться в подпрограмме обслуживания прерывания, однако это означает, что прерывание будет дольше блокировано, а это может стать преградой для использования очереди из двух прерываний.

Подпрограмма обработки прерывания также применима и в случае передачи данных. Отправляемый символ просто записывают в буфер, в который каждый раз по завершении передачи помещается очередной символ. Когда буфер освобождается, прерывания следует запретить.

Отказ от использования прерываний может быть обоснован только в том случае, если прерывание требуется для другого очень критичного ко времени процесса.

Управление двигателем

В случае управления двигателями (шаговыми или трехфазными) идеальный вариант — прерывание по сравнению. При этом значение для ввода/вывода, соответствующее текущему шагу, извлекается из таблицы. Затем указатель устанавливается на следующий шаг, а регистр CCPR — на соответствующую временную позицию. Таким способом также очень удобно реализовать грузовую платформу. Главная программа в управлении подобными процессами особого участия принимать не должна.

Регистрация импульсов, формируемых таймером

Для организации процесса, который должен обслуживаться в определенных временных рамках, также прекрасно подходит прерывание по сравнению. Переполнения таймеров Timer0 и Timer1 имеют, как правило, тот недостаток, что таймер всегда должен устанавливаться в исходное значение, чтобы запрограммировать период до следующего переполнения. Однако мы еще раз напоминаем о том, что постоянная запись в таймер — не самый лучший метод, поскольку с его помощью вряд ли можно достичь высокой точности. Даже если приближение составляет всего лишь несколько командных циклов, следует принимать во внимание тот факт, что ошибка накапливается. Для создания часов реального времени подобный метод не рекомендуется, поскольку просчет даже на тысячную долю секунды приводит к запаздыванию часов на пятнадцать минут за десять дней.

В случае с таймером Timer2 регулярная запись в счетный регистр с помощью программируемого момента переполнения (регистр PR2) не требуется.

Когда речь идет об импульсах, формируемых таймером, то имеется в виду регистрация равноотстоящих друг от друга событий времени, на основании которых строится программный таймер. Многие пользователи сейчас подумали о прерываниях, однако, если импульсы используются только для организации счета программного таймера, то использование прерываний, как показывает опыт, не только излишне, но даже портит структуру программы.

Если импульсы, формируемые таймером, регистрировать по прерыванию, то и все необходимые программные таймеры также должны быть реализованы с помощью прерываний. Уведомление о поступлении импульса, как правило, используется нечасто, поскольку в этих целях можно также опрашивать разряд переполнения.

Впрочем, программный таймер не приносит главной программе никакой пользы, если он не опрашивается через достаточно короткие промежутки времени. В таком случае главная программа не полностью освобождается от отслеживания времени, а только удлиняются интервалы, в которых она должна позаботиться о времени.

Вследствие того, что подпрограмма обработки прерывания берет на себя приращение программного таймера, создается впечатление, что с главной программы снимаются все заботы. Однако мы знаем множество программных примеров, где для запуска или останова программного таймера между подпрограммой обслуживания прерывания и главной программой должно передаваться столько сообщений, что об удобочитаемости не может идти даже речи.

Кроме того, подпрограммы формирования импульсов с помощью таймера могут быть очень длинными, что может привести к блокировке прерываний. Отсюда вывод: для формирования импульсов с помощью таймера прерывания, как правило, не используются, поскольку это почти всегда излишне, а контроль за временем лучше реализовать в главной программе.

7.7. Организация программного таймера

Время представляют как некоторую непрерывную величину. Однако наш таймер — как аппаратный, так и программный — ведет отсчет величин времени, кратных более или менее точным единицам.

Для построения времязадающей структуры почти всегда используют аппаратный таймер. Отсчитываемые им единицы времени зависят от такта системной синхронизации и выбранного коэффициента деления частоты. Рассмотрим на конкрет-

ных примерах, каким образом используется аппаратный таймер для построения сложных систем с программным таймером и часами реального времени.

В случае программного таймера выделим базисный и прикладные таймеры. **Базисный таймер**, как правило, работает непрерывно параллельно программному процессу. Его еще называют "автономным". Такие таймеры или ведут счет до переполнения их счетной переменной, или образуют иерархическую структуру. Самым известным иерархическим базисным таймером являются часы, отсчитывающие время в секундах, минутах, часах, месяцах и годах.

Прикладной таймер запускается с началом процесса и обычно отсчитывает определенный интервал времени (по убыванию). Зачастую отсчет служит для того, чтобы организовать останов процесса по достижении нуля. Иногда подобные таймеры применяют в качестве счетчиков тайм-аута. Одним из распространенных примеров использования прикладного таймера является контроль задержек.

В некоторых случаях применения требуется множество прикладных таймеров, которые работают одновременно, однако независимо друг от друга и асинхронно по отношению друг к другу. В подобных случаях рационально использовать элементарный таймер, который служил бы в качестве источника тактовых сигналов для различных прикладных таймеров.

7.7.1. Пример программного таймера

Следующий пример предназначен только для того, чтобы продемонстрировать типичный случай построения счетчика. В конкретном варианте применения многие детали могут быть реализованы по-другому. Предположим, в этом примере нам необходимо организовать как часы реального времени, так и множество специальных прикладных таймеров. При этом будем исходить из того, что прикладных таймеров с разрешением больше 4 мс не требуется.

Прежде всего, организуем **элементарный счетчик**, с помощью которого будем выполнять отсчеты, кратные 4 мс. Для реализации счетной подпрограммы можно воспользоваться модулем сравнения или подпрограммой ZEITDA из рассмотренной ранее подпрограммы WATCHTIM. Подпрограмма должна вызываться через равномерные промежутки времени (как минимум, через каждые 4 мс).

Таким образом, подпрограмма ZEITDA используется внутренне только один раз за 4 мс. Впрочем, в большинстве случаев это не означает, что она вызывается в точности через каждые 4 мс.

Переменную, используемую для отсчета четырех миллисекунд, назовем **STEP**. Как правило, достаточно, если размерность STEP составляет байт. Значение переменной STEP увеличивается без сброса (модуль 256). Мы опять выбираем частоту 4 МГц и коэффициент деления 32. В простейшем случае программа имеет следующий вид:

```
ZEITDA   MOVLW .125
         ADDWF EVENT        ; 125 * 32 мкс = 4 мс !
         INCF  STEP
         CALL  TIMUP        ; Организация каждого
                            ;  программного таймера
         RETURN
```

Во многих случаях рационально каждый программный таймер (как часы, так и прикладной таймер) размещать в подпрограмме TIMUP. В результате счетная переменная STEP будет в распоряжении всех программных таймеров.

Внутри подпрограммы TIMUP будут заключены все таймеры, ведущие счет в одной и той же иерархии времени. Например, подпрограмма MILLITIMER относится ко всем прикладным таймерам, которые считают с тактом STEP, а подпрограмма SEKTIMER — ко всем прикладным таймерам, отсчитывающим секунды. На основании посекундного такта вызывается подпрограмма UHRZEIT.

Результат отсчета секунд будем хранить в переменной SEKEVENT. Всякий раз, когда STEP = SEKEVENT, прошла одна секунда, и значение SEKEVENT увеличивается на 250.

```
TIMUP   CALL   MILLITIMER        ; Прикладной таймер
        MOVF   SEKEVENT,W
        SUBWF  STEP,W
        SKPZ
        RETURN
        MOVLW  .250              ; Прошла одна секунда
        ADDWF  SEKEVENT
        CALL   UHRZEIT
        CALL   SEKTIMER          ; Таймер секунд
        RETURN
```

Иногда в таких подпрограммах велико искушение, кроме отсчета времени, реализовать и другие задачи, которые завершались бы до истечения четырех миллисекунд ("миллизадачи") или секунды ("секундные задачи"). Хотя в редких случаях это и может иметь практический смысл, подобные решения превышают полномочия модуля TIMUP. Если строго придерживаться модульного принципа, то модулю формирования временных отрезков разрешено только передавать сообщения, информирующие о том, что прикладной таймер выполнил отсчет или истекла секунда. Программа опрашивает соответствующие флаги и затем сама выполняет "миллизадачу" и "секундную задачу".

Нам часто встречаются программы, в которых между программой и модулем времени передается множество сообщений, вносящих путаницу. В большинстве случаев мы решаем эту проблему (если это возможно) следующим образом. Прикладной таймер в начале счета инициализируется из программы некоторым исходным значением. В подпрограмме TIMUP предполагается, что таймер всегда ведет счет с убыванием, если значение не равно нулю. Как только счетное значение становится равным нулю, прикладной таймер считается отключенным.

Программа сама должна распознавать, что прикладной таймер активизирован. Сообщение о том, что таймер ведет счет с убыванием, из чего следует, что счетная переменная равна нулю.

В таком случае, счетчик на вычитание прикладного таймера XTIME имеет следующий незамысловатый вид:

```
MOVF  XTIME
SKPZ
DECF  XTIME
```

В большинстве случаев прикладной таймер состоит только из одного байта. Тем не менее, иногда требуется применить двухбайтный таймер. Если XTIME состоит из двух байтов, то счетчик на вычитание принимает следующий вид:

```
        MOVF  XTIME,W
        IORWF XTIME+1,W
```

```
        SKPNZ
        GOTO  WEITER
        DEC2  XTIME
WEITER
```

Подпрограмма реализации часов

Хотя подпрограмма реализации часов (назовем ее UHRZEIT) не является чем-то оригинальным, мы все же рассмотрим один из ее вариантов, поскольку это будет полезно начинающим программистам. Эта подпрограмма существует в различных вариантах. Иногда время подсчитывают в BCD-формате, но в представленном ниже примере мы использовали обычный формат чисел.

В позиции UHRZEIT мы оказываемся каждый раз по истечении секунды. используются переменные SEKUNDE (секунды), MINUTE (минуты), STUNDE (часы), TAG (дни), MONAT (месяцы) и JAHR (годы).

```
UHRZEIT  INCF   SEKUNDE
         MOVLW  .60
         SUBWF  SEKUNDE,W
         SKPZ
         RETURN
         CLRF   SEKUNDE      ; Следующая минута
         INCF   MINUTE
         MOVLW  .60
         SUBWF  MINUTE,W
         SKPZ
         RETURN
         CLRF   MINUTE       ; Следующий час
         INCF   STUNDE
         MOVLW  .24
         SUBWF  STUNDE,W
         SKPZ
         RETURN
         CLRF   STUNDE       ; Полночь, следующий день
         INCF   TAG
         CALL   GETTAGE      ; Таблица: 28, 29, 30 или 31
         SUBWF  TAG,W
         SKPZ
         RETURN
         MOVLW  1            ; Первый день следующего месяца
         MOVWF  TAG
```

Конечно же, мы могли бы продолжить реализацию подсчета времени до следующего тысячелетия, однако решили остановиться, чтобы не рассердить читателя.

7.7.2. Точность таймера

В отношении точности следует ответить на два вопроса.

- Какие требования к точности предъявляет данная задача?
- Какую точность может обеспечить выбранная организация времени?

Часто встречаются большие программы, в которых на эти вопросы нельзя ответить однозначно. С одной стороны, требуется микросекундная точность регистрации формируемых таймером импульсов, с другой — организованный подобным образом таймер рассматривается только в ракурсе пятидесятиста миллисекунд. Нередко временной процесс отсчитывается с точностью в несколько тысячных, в то время как в другом месте программы не играет роли даже неточность в несколько процентов.

Иногда должны выполняться требования, предъявляемые к точности, которые, на самом деле не имеют смысла. Например, по сути, не существенно, сколько времени длится подсветка ЖК-дисплея после последнего нажатия кнопки: 59 или в точности 60 секунд — поскольку такая разница практически не воспринимается. И все же, можно поспорить, что кто-то с помощью секундомера зарегистрирует эту неточность и сделает вывод о том, что вся система недостаточно точная.

Для организации надежного управления временем с помощью простых средств следует соблюдать ряд важных правил. При этом, естественно, предполагается, что такт системной синхронизации соответствует требованиям, предъявляемым к точности программы.

- Регистрация формируемых таймером импульсов не обязательно должна выполняться с микросекундной точностью, **однако следует помнить, что погрешность постоянно накапливается.**
- **По возможности, следует воздерживаться от доступа к таймеру на запись.** Если он, все-таки, необходим, то необходимо учитывать, что в таком случае таймер останавливается на два тактовых цикла, а делитель частоты сбрасывается.
- Прикладные таймеры должны быть достаточно точными. Если, к примеру, активизировать какой-либо процесс, асинхронный по отношению к базовым тактам, и затем установить счетчик минут на стартовое значение 5, то может случиться так, что минутный импульс отстанет от базового времени на секунду, и значение сразу же изменяется на 4. Это дает погрешность 20%. Лучше выбрать секундный таймер, в который должно записываться исходное значение 300. Это означает, что для такого счета потребуется два байта.

Естественно, лучше реализовать систему базовых импульсов таким образом, чтобы двухбайтный счет не понадобился, а результат подсчета не был слишком грубым. Это означает, что система базовый импульсов — это комплекс, который должен быть реализован с учетом всех без исключения вариантов применения.

В качестве альтернативы, можно также попытаться избежать запуска процесса, асинхронного по отношению к базовым импульсам. Для этого мы часто используем описанные ниже "тактированные циклы".

7.8. Главные циклы

Подобные программы почти всегда состоят из небольшого главного цикла (или нескольких циклов), представляющего собой модуль, который, как правило, только опрашивает события и вызывает подпрограммы.

Задача главной программы — взаимодействие с отдельными модулями по хорошо продуманной системе. Таким образом, она выполняет функцию своеобразной операционной системы, хотя, в отличие от нее, спроектирована согласно конкретной постановке задачи.

Главные циклы стараются сделать короткими, поскольку это улучшает наглядность программы. Для достижения этой цели применяют ряд уловок. Если главный цикл большой, то следует проверить, не слишком ли детализировано распределение в модуле, и не лучше ли несколько модулей объединить в одно логическое целое.

В случае микроконтроллеров PIC первого поколения это зачастую было невозможно, поскольку допускалось только два уровня подпрограмм, которые нельзя было расходовать без лишней надобности.

Одним из хороших вариантов базовой структуры является организация нескольких главных циклов. Типичный пример — один цикл для управления двигателем, и второй — для его отключения. Такой подход избавляет от множества опросов и флагов уведомления. Часть программы, объединяющая оба цикла, выполняется как подпрограмма. При таком способе организации переходы между циклами гораздо нагляднее, чем в случае простого включения/отключения двигателя и установки/сброса флага "двигатель включен".

Использование нескольких главных циклов — хороший вариант также в тех случаях, когда требуется реализовать несколько рабочих режимов. Преимущество при этом заключается в том, что приходится четко описывать переход из одного режима в другой. С одной стороны, это означает, что условия перехода будут точно определены, а с другой стороны — что процесс смены режима более нагляден.

7.8.1. Асинхронные циклы

Простейшей формой главного цикла является случай, когда для выполнения всех законченных задач предоставляется заданное время центрального процессора. Для контроля за временем в промежутках постоянно вызывается подпрограмма `WATCHTIM` (см. выше). При этом внутри каждого цикла задача может быть без проблем обслужена несколько раз.

Длительность асинхронных циклов определяется исключительно временем решения конкретной задачи — ни больше, ни меньше. В таких случаях полезно приблизительно знать минимальное, максимальное и среднее затрачиваемое время. Хотя подобные циклы и не отличаются особой элегантностью, у них есть большое преимущество: простота и наглядность.

7.8.2. Ждущие главные циклы

Когда на передний план выходит управление или регулирование каким-либо периодическим процессом, то обычно организуют главные циклы, которые выполняются в точности один раз за время периода.

Если все задачи завершаются в пределах одного периода, то программа ожидает начала следующего периода. Мы называем подобные программные структуры **"ждущими циклами"**.

Типичным случаем применения таких циклов является управление двигателями, а также любые другие периодические процессы, основанные на применении переменного напряжения. Для событий, которые должны регистрироваться асинхронно к таким процессам (например, события интерфейсов или нажатия кнопки), требуется некоторая изобретательность. Выбор способа регистрации (через равномерные отрезки времени или с помощью подпрограмм обработки прерываний) всегда зависит от конкретной ситуации.

7.8.3. Тактированные главные циклы

Когда речь идет о событии времени, то соответствующую программную структуру мы называем "тактированным циклом" (рис. 7.1). Такой цикл ожидает возникновения события, как это было описано ранее на примере подпрограммы WAITIM.

Эта стратегия основана на том факте, что в большинстве случаев центральный процессор использует только очень незначительную часть своего времени на выполнение относительно важных задач, а все остальное время находится в состоянии ожидания и опроса.

В большинстве случаев применения достаточно промежутка от 1 до 10 мс. Мы активно используем циклы длительностью 4 мс, с помощью которых удобно формировать отсчет времени.

Применение тактированных циклов, с одной стороны, устраняет проблему регистрации формируемых таймером импульсов, а с другой — помогает создать наглядную структуру программы, дающую наилучшее представление о временных соотношениях.

То, что отдельные задачи выполняются только за фиксированные промежутки времени, — это, как правило, преимущество, поскольку в результате обычно отпадает необходимость в опросах. Таким образом, например, легко организовать учет дребезга контактов для точной регистрации одинаковых состояний кнопки.

Рис. 7.1. Тактированный цикл

Преимущество подобной программной структуры заключается в том, что многие процессы могут выполняться синхронно с базовым тактовым сигналом. Это означает, что прикладные таймеры могут без проблем работать синхронно с базисными таймерами.

Под "миллизадачей" подразумевается любая задача, которая обслуживается в течение четырех миллисекунд. Типичные примеры — учет дребезга контакта и контроль за током. Если же некоторая задача может быть обслужена только с промежутком в 20 мс, то используют специальный счетчик. Для попеременного выполнения функции также можно воспользоваться описанной выше переменной STEP.

Если частота системной синхронизации составляет 4 МГц, то за четыре миллисекунды можно обработать 4000 команд. А с помощью такого количества ассемблерных команд можно реализовать много чего!

Обычно требуется намного меньше, чем 4000 команд, поэтому программа большую часть времени проводит в цикле ожидания. В таком случае, в цикл можно включить еще несколько асинхронных фрагментов, наподобие опроса интерфейсов. Кроме того, тактированные циклы имеют смысл только тогда, когда большая часть программы тактированная.

Типичные примеры "секундной" или "минутной" задачи — управление отоплением: "минутная" задача — включение насоса, а "секундная" — контроль за ним. Таким образом, модуль управления отоплением разбивают на несколько частичных процессов.

С секундным или минутным тактом также нередко работают прикладные таймеры, за опрос и обслуживание которых отвечает подпрограмма TIMUP. Кроме того, эта подпрограмма по истечении секунды или минуты передает уведомления, которые после обработки "секундной" или "минутной" задачи обязательно должны быть сброшены.

ГЛАВА

8 Система разработки MPLAB

Эта глава разбита на несколько разделов: вначале рассматривается установка MPLAB, а затем представлены первые шаги работы в этой среде на примере использования мастера проектов. Самые объемные разделы посвящены описанию команд меню MPLAB и особенностей ассемблера MPASM.

8.1. Установка

В этой книге рассматривается версия MPLAB 6.60. Процесс установки очень прост, и, традиционно для программных продуктов компании Microchip, не вызывает никаких проблем.

На прилагаемом к книге компакт-диске в папке MPLAB, кроме пакета установки MPLAB 6.60, присутствуют также пакеты для версий MPLAB 5.7 и 7.0.

Среда, в которой автор установил свою версию MPLAB:

- процессор AMD Athlon XP 2200+ 1,8 ГГц;
- операционная система Windows 98SE;
- объем оперативной памяти — 512 МБайт;
- достаточно свободного места на жестком диске.

Для установки версии MPLAB 6.60 потребуется архивный файл `MPLAB660.zip`, содержащий программу установки `MPLAB v6.60.exe` (или просто сама программа). После распаковки этой программы ее следует запустить на выполнение двойным щелчком мыши.

Как обычно, перед установкой новой программы следует закрыть все другие приложения, которые могут мешать в процессе установки. Компания Microchip рекомендует закрыть даже калькулятор.

Итак, мы запустили на выполнение файл установки `MPLAB v6.60.exe`. Как и для большинства других программ, в первую очередь потребуется подтвердить свое согласие с некоторым "лицензионным соглашением", иначе установку продолжить не удастся.

В следующем окне выбирается папка назначения для установки MPLAB. Как показывает опыт, изменять путь размещения программы следует только в том случае, если без этого никак не обойтись. Выбор по умолчанию, как правило, — самый надежный.

В наш век ГИГАнтских дисков сделать резервную копию установленной программы — не проблема, потому рекомендуем создать ее в отдельной директории.

В дальнейшем, после проверки корректности установки MPLAB, страховочную копию можно будет удалить.

Далее в мастере установки осталось только три раза нажать кнопку **Next**, чтобы задать создание ярлыков в меню **Пуск** и на рабочем столе, и запустить собственно процесс установки.

По завершению установки будет предложено прочитать файл Readme, что, в общем-то, делать совсем не обязательно. Тем не менее, в случае возникновения каких-либо проблем или неясностей этот файл станет хорошим помощником. Компания Microchip хорошо потрудилась над файлом Readme, поэтому не стоит о нем полностью забывать — там можно найти ответы на многие вопросы.

Что же касается HTML-файла с описанием установки драйвера, то его прочтение — обязательно. USB-драйвер для устройств Microchip (ICD2, ICE2000 и др.) должен быть установлен до того, как соответствующее устройство будет подключено. В противном случае может загрузиться какой-либо из драйверов Microsoft, который ни на что не годится.

8.2. Первые шаги

Если при запуске MPLAB 6.60 компьютер немного "задумался", то выждите несколько секунд. В версиях до 5.62 запуск выполнялся быстро, но теперь потребуется немного терпения.

Центральными понятиями среды MPLAB являются "проект" и "рабочая среда". Проект содержит информацию о приложении: различные файлы, средства преобразований и их настройки. Рабочая среда определяет тип микросхемы, конфигурацию среды разработки, программатор, открытые окна и их параметры, а также все другие настройки, имеющие отношение к среде программирования.

Для того чтобы использовать привычные методы работы, характерные для "старых" версий MPLAB, следует в первую очередь установить опцию **Use one-to-one project-workspace model** в диалоговом окне **Settings** (рис.8.1). Это означает, что в рабочей среде может быть открыт только один проект.

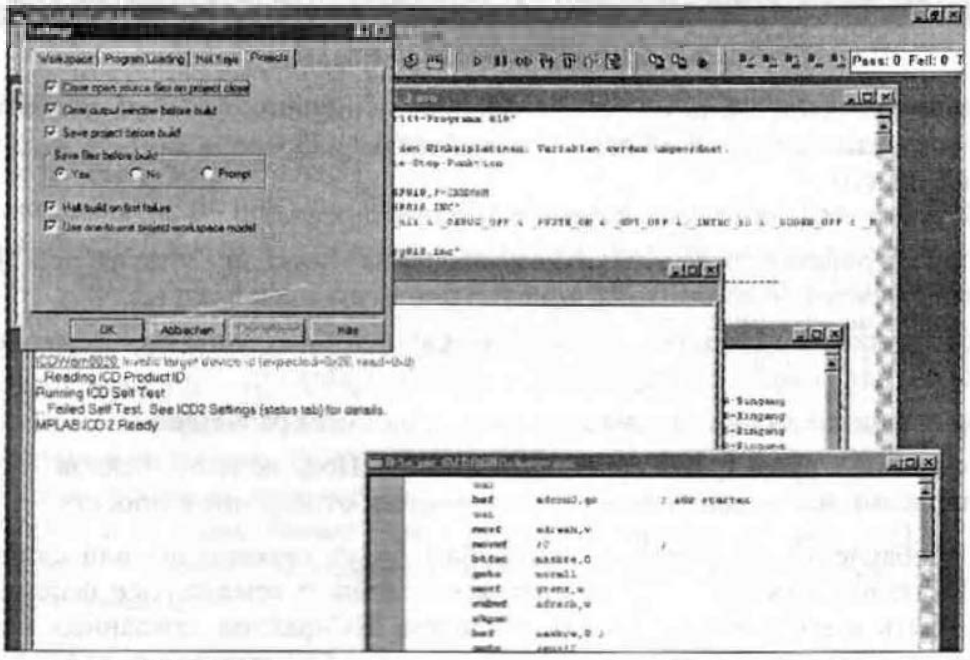


Рис. 8.1. Диалоговое окно **Settings**, открываемое по соответствующей команде меню **Configure**

Если в дальнейшем потребуется открывать одновременно несколько проектов, это можно легко организовать, сбросив флажок **Use one-to-one project-workspace model** в диалоговом окне **Settings**.

Этим новая система несколько непривычна для пользователей предыдущих версий MPLAB, однако для создания проектов можно воспользоваться удобным средством под названием Project Wizard, который вызывается по соответствующей команде меню **Project** (рис. 8.2).

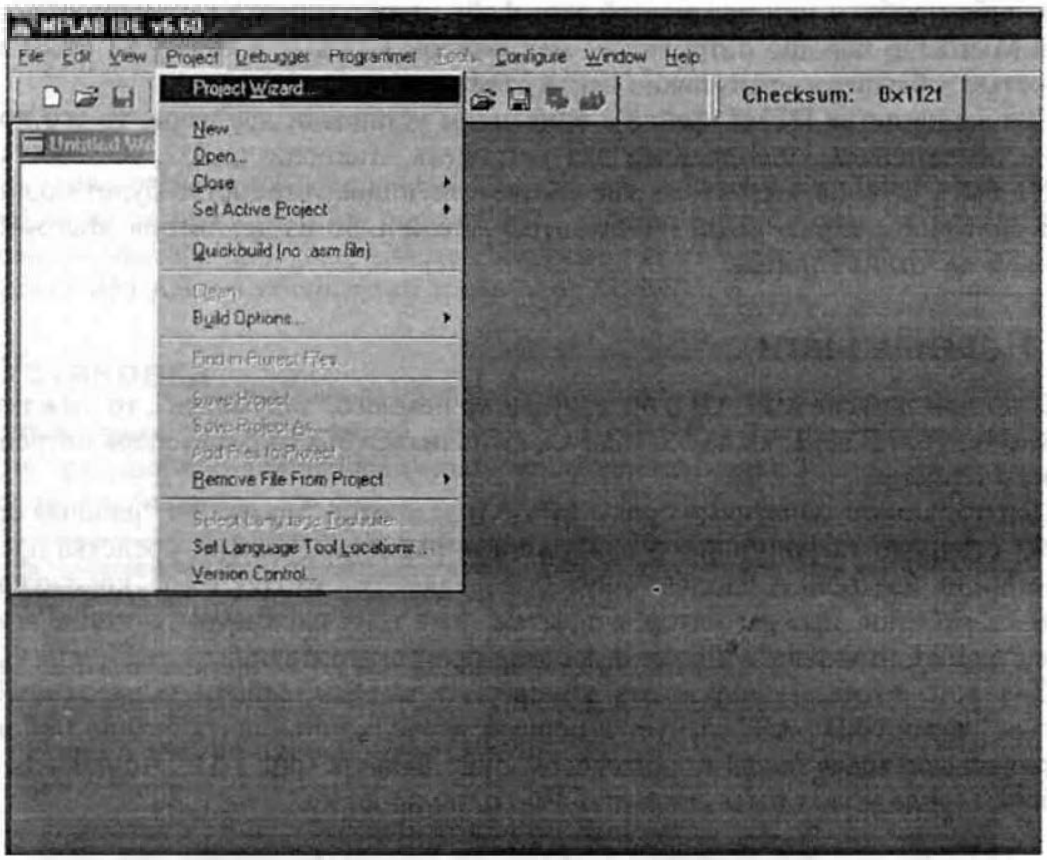


Рис. 8.2. Вызов мастера проектов MPLAB

Мастер проектов очень удобен для начинающих пользователей, поскольку подсказывает каждый следующий шаг. В дальнейшем, набравшись опыта, можно обходиться и без него.

Работа Project Wizard состоит из следующих этапов.

1. Выбор устройства (типа микроконтроллера) из списка представителей PIC, который с каждой новой версией MPLAB становится все длиннее.
2. Выбор языкового средства: ассемблера или одного из поддерживаемых компиляторов, для которого следует задать корневой каталог.
3. Ввод имени проекта и каталога, в котором он будет размещен.
4. Добавление к проекту существующих файлов. Подключение файлов с макросами или подпрограммами, имеющими отношение к проекту.

Это добавление не означает, что файлы будут перенесены или скопированы в новый каталог проекта, — речь идет всего лишь о ссылке. Все файлы проекта должны быть в его каталоге (за исключением INC-файлов, созданных компанией Microchip для различных типов микроконтроллеров PIC, например, `P16F877.inc`).

После отображения окна с общими итогами, мастер завершает свою работу, после чего можно открывать окна для различных файлов проекта (например, файлов с исходным кодом программы).

8.3. Обзор команд меню MPLAB

В этом разделе дан беглый обзор команд меню среды MPLAB 6.XX. При этом команды, традиционные для всех приложений Windows, не рассматриваются, дабы не утомлять читателя.

8.3.1. Меню File

Назначение меню **File** (рис. 8.3) — традиционно, а все его пункты интуитивно понятны.

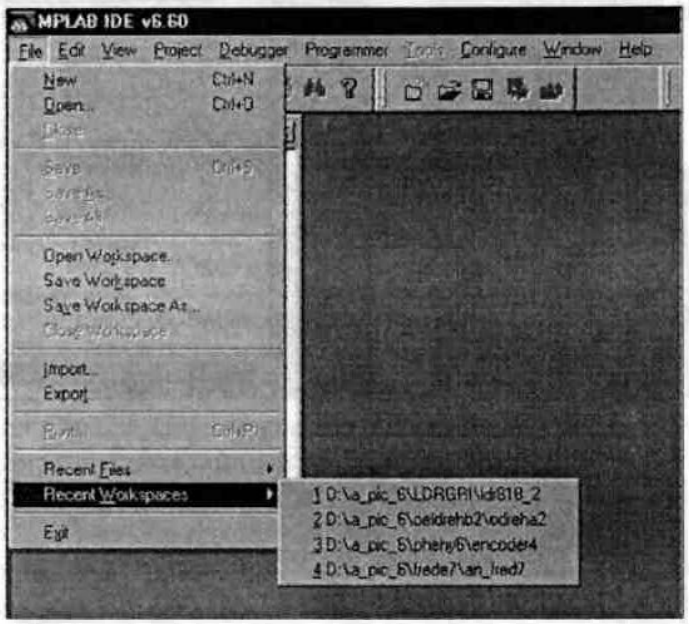


Рис. 8.3. Меню File

Первые шесть пунктов меню **File** хорошо знакомы любому пользователю Windows, а следующие четыре в рассматриваемом контексте имеют отношение к проекту (в режиме "one-to-one"!!).

С помощью команд **Import** и **Export** обрабатываются исключительно HEX-файлы. Таким образом, к примеру, даже без создания проекта можно прочитать любой HEX-файл, и затем просмотреть его команды в окне программы. Однако, если вы забыли правильный тип микроконтроллера, то HEX-код распознан не будет, и появится сообщение об ошибке. Среда MPLAB не может определить, к какому типу микроконтроллеров PIC относится данная программа, по одному только HEX-коду!

С помощью команды **Export** содержимое памяти программ можно сохранить в двух различных форматах: INHX32 и INHX8S.

Последние файлы и рабочие среды, которые открывал пользователь, также можно выбрать в подменю **Recent Files** и **Recent Workspaces** (см. рис. 8.3). В "традиционной" среде "one-to-one" в подменю **Recent Workspaces** перечислены недавно открывавшиеся проекты.

8.3.2. Меню Edit

В меню Edit (рис. 8.4) есть все, что душа пожелает. Первые одиннадцать команд — стандартные для любого редактора. Дополнительные удобства предоставляют команды подменю Advanced. С их помощью, к примеру, можно одним нажатием клавиши перевести символы выделенного фрагмента текста в верхний регистр (команда Uppercase). Для упрощения поиска определенных строк текста предлагаются закладки (подменю Bookmarks). Последняя команда Properties вызывает диалоговое окно настройки параметров редактора (рис. 8.5 и рис. 8.6).

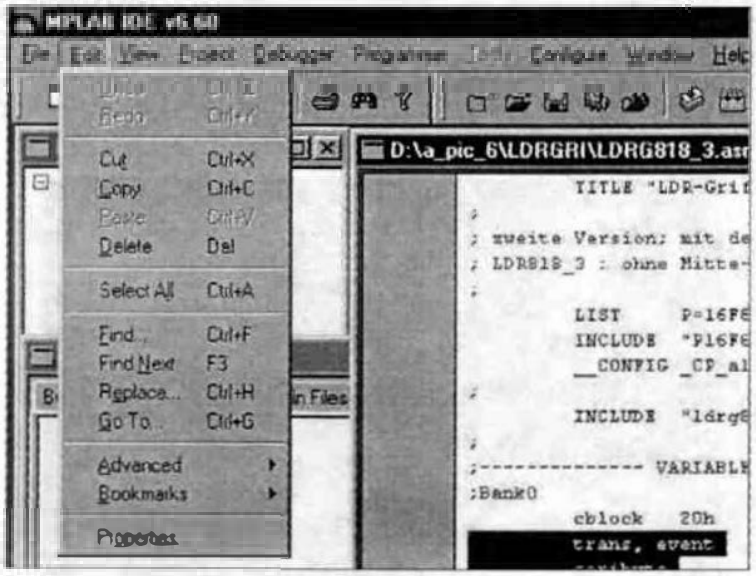


Рис. 8.4. Меню Edit

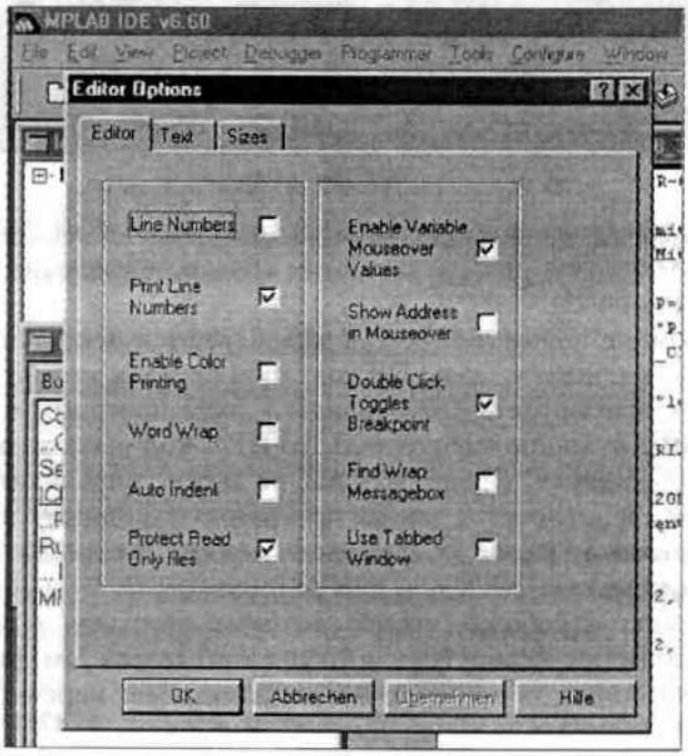


Рис. 8.5. Набор параметров редактора

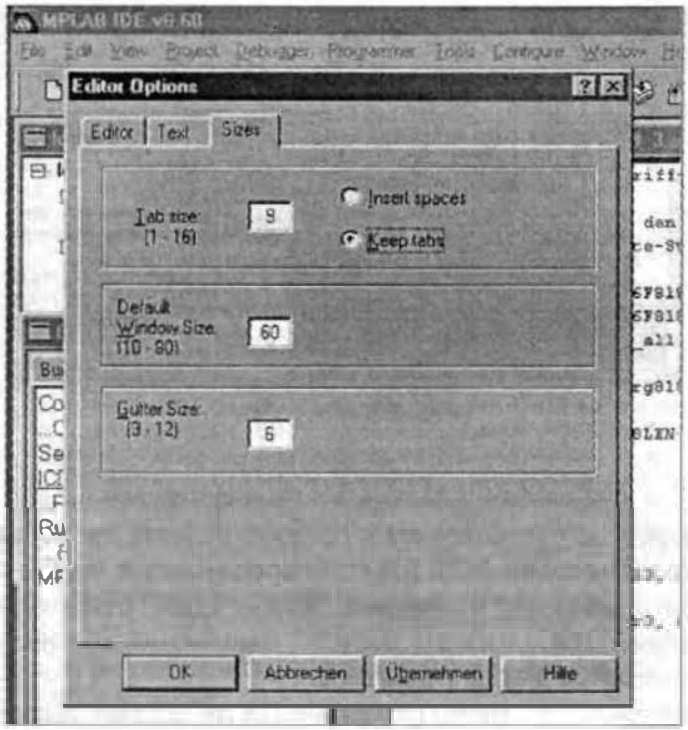


Рис. 8.6. И еще несколько параметров редактора

По сравнению с предыдущими версиями, редактор MPLAB 6.XX предоставляет гораздо больше возможностей. Одним из самых заметных новшеств является "цветное кодирование". Это означает, что команды ассемблера отображаются полужирным шрифтом синего цвета, комментарии — светло-зеленым, а директивы, наподобие code, equ и org, — светло-синим цветом. И эти настройки не ограничены жесткими установками цвета. В представленном выше диалоговом окне **Editor Options** можно выбрать любой цвет для различных элементов программы — для этого следует перейти на вкладку **Text** и нажать кнопку **Choose Color**. Рационально ли менять предустановленные компанией Microchip цветовые настройки редактора, пусть каждый решает сам.

Не знаем, как для других, но для нас большое значение имеет установка отступа по нажатию клавиши <Tab>, и стандартное значение 8 нас не устраивает. Его можно уменьшить на вкладке **Sizes** в поле **Tab size** (см. рис. 8.6). Ограничение ширины окна определенным числом знаков очень полезно при форматировании текста программы, поскольку для подобных документов очень важно, чтобы их содержимое всегда было полностью перед глазами. Для подобной настройки предназначено поле **Default Window size** (см. рис. 8.6).

Ширине серого поля, расположенного вдоль левого края окна редактора, соответствует параметр **Gutter size**. Для нас значение 6 кажется слишком большим, и потому мы уменьшаем его до минимума.

8.3.3. Меню View

В меню **View** (рис. 8.7) находятся команды отображения/сокрытия окон и панелей инструментов. Для того чтобы не загромождать рабочую область MPLAB, целесообразно временно скрывать те окна, которые в данный момент не используются. Некоторые пункты меню **View** — фиксированные, другие же зависят от используемых средств отладки и программирования.

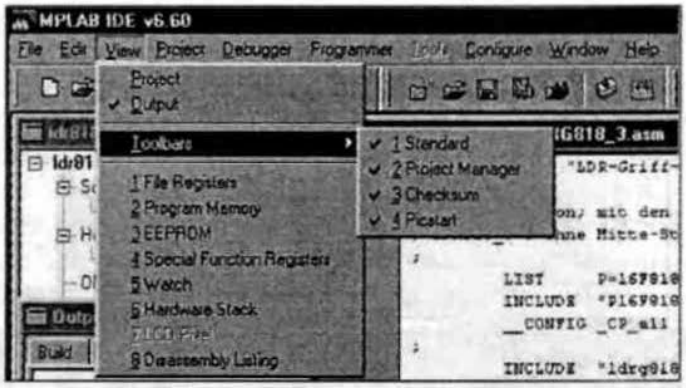


Рис. 8.7. Меню View

Окно проекта

В этом окне перечень относящихся к проекту файлов представлен в виде иерархической структуры. В версии MPLAB 6.60 предлагается шесть категорий файлов. В самой верхней категории под названием "Source Files" отображается имя главного ассемблерного файла. При двойном щелчке мышью на имени некоторого файла в иерархии он открывается в окне редактора.

С помощью контекстного меню (открывается по щелчку правой кнопкой мыши) с файлом можно выполнять различные операции (рис. 8.8). Так, команда **Assemble** выполняет трансляцию соответствующего файла с исходным кодом. Для вызова диалогового окна настройки параметров трансляции используется команда **Build Options**. Рассмотрим кратко процесс ассемблирования по команде **Assemble**.

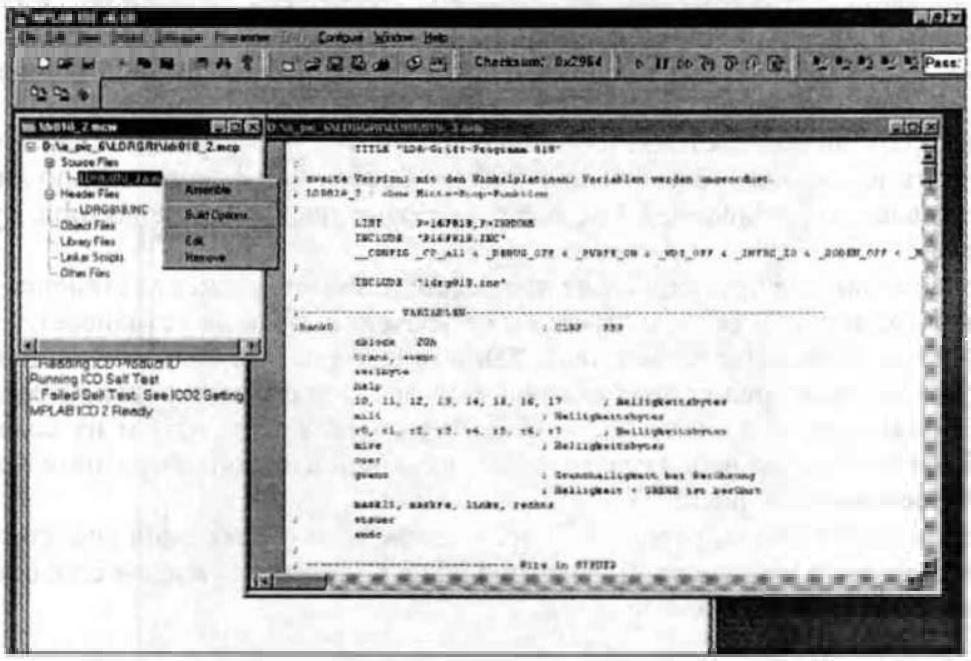


Рис. 8.8. Окно проекта с контекстным меню

Окно Output

В окне **Output** отображается ход ассемблирования (рис. 8.9) или обмен данными с ICD2 (рис. 8.10) или PICStart. Здесь же выводятся сообщения о результатах поиска заданного текстового фрагмента в файлах проекта (вкладка **Find in Files**).

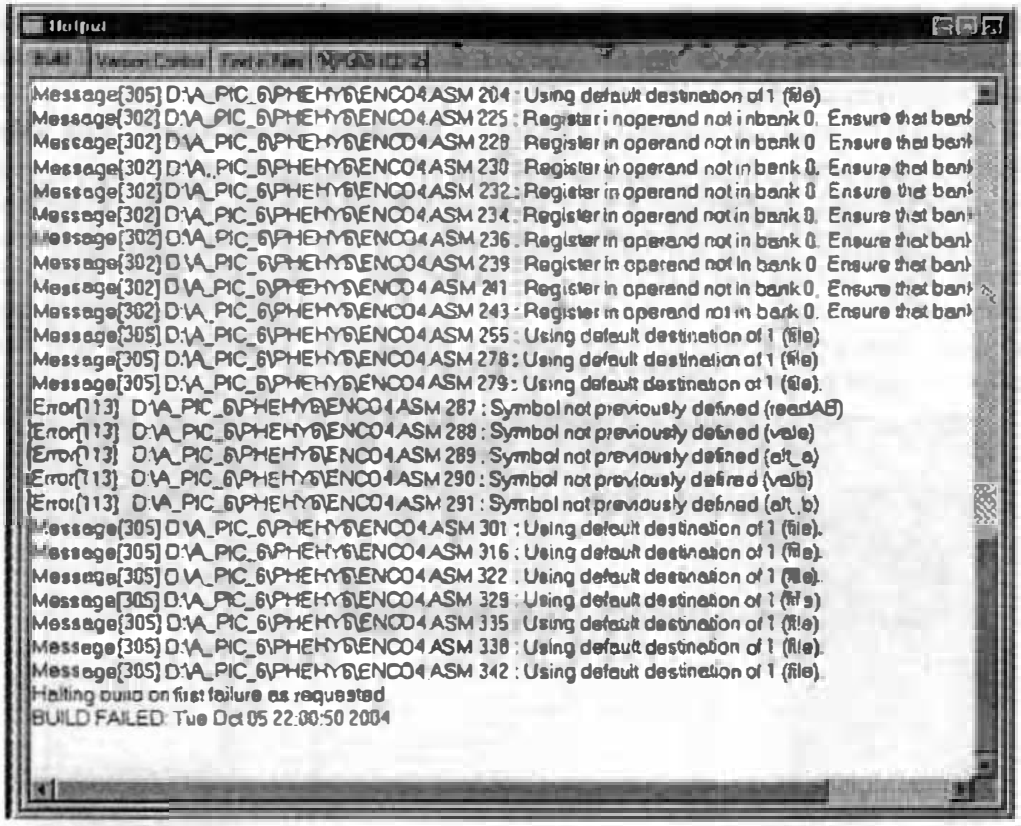


Рис. 8.9. Окно Output — результат трансляции

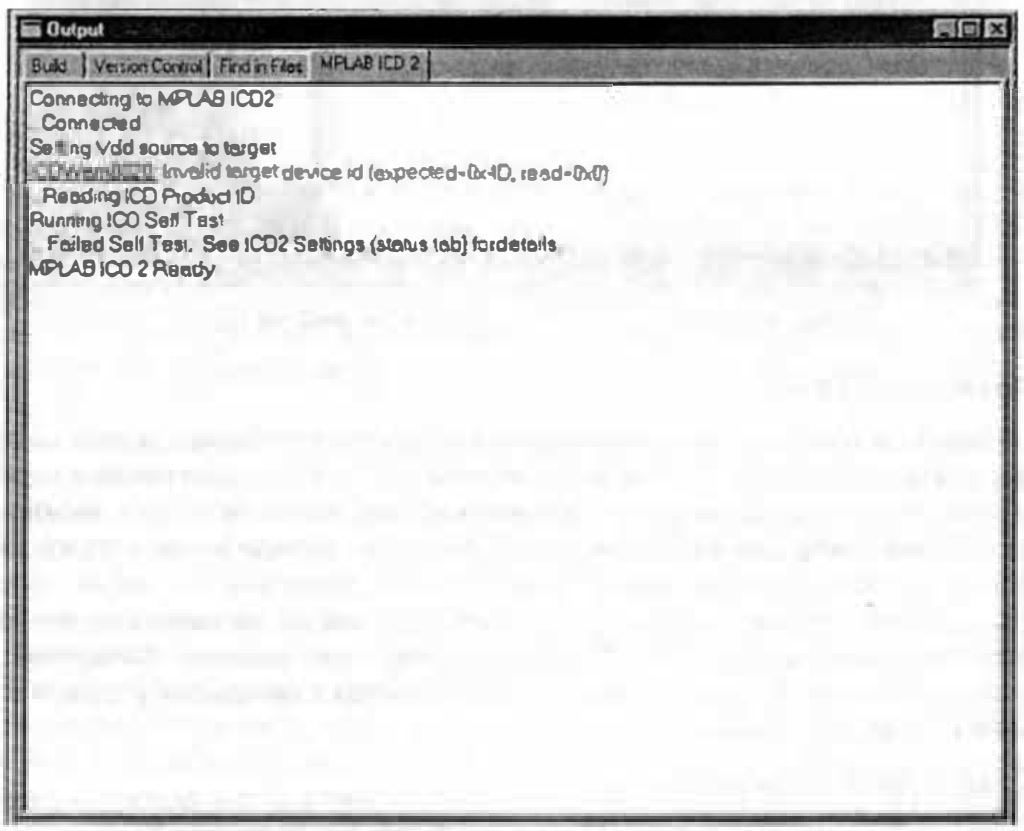


Рис. 8.10. Окно Output — обмен данными с ICD2

Если в этом списке щелкнуть мышью на некотором элементе, то курсор автоматически переместится в ту позицию файла с исходным кодом, в которой была обнаружена соответствующая ошибка.

На рис. 8.10 показан пример обмена данными с эмулятором MPLAB ICD2. В случае возникновения каких-либо проблем можно дважды щелкнуть мышью на номере ошибки ("ICD1000" или "ICDWarn0073") для автоматического вызова окна контекстной справки с описанием возможных причин возникновения проблемы.

На рис. 8.11 представлен результат выполнения запроса на поиск в других файлах проекта (команда меню **Project ▸ Find in Project Files**). В этом случае, кроме имени файла, в окне Output указывается номер строки, в которой был найден искомый фрагмент, а также сама эта строка.

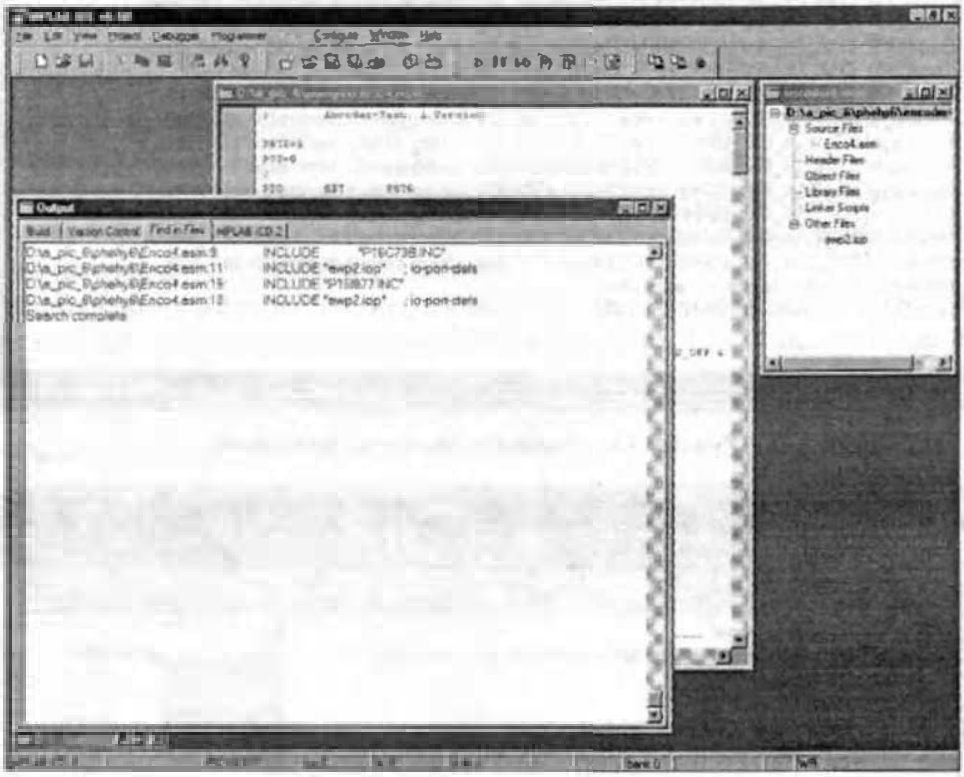


Рис. 8.11. Результат поиска по ссылкам на внешние файлы

Подменю Toolbar

С помощью пунктов подменю **Toolbar** пользователь выбирает, какие панели инструментов должны быть отображены, а какие — нет. Аналогичное контекстное меню также открывается по щелчку правой кнопкой мыши непосредственно на любой из панелей инструментов. Если возле некоторого пункта меню установлена "галочка", то соответствующая панель инструментов отображается, иначе — скрыта. Для определения набора кнопок панели инструментов пользователь может выбрать в соответствующем контекстном меню самую нижнюю команду: **Customize**. Выбор отображаемых элементов, опять таки, осуществляется с помощью флажков в диалоговом окне **Customize Toolbar**.

Стандартные окна отладчика

Список восьми команд меню **View**, следующих после пункта **Toolbar**, — фиксированный. По щелчку мышью на одной из них открывается соответствующее окно

(рис. 8.12). В распоряжение пользователя предоставляется окно, отображающее содержимое памяти программ (команда **Program Memory**), окно регистров специального назначения (команда **Special Function Registers**), аппаратного стека (команда **Hardware Stack**) и др.

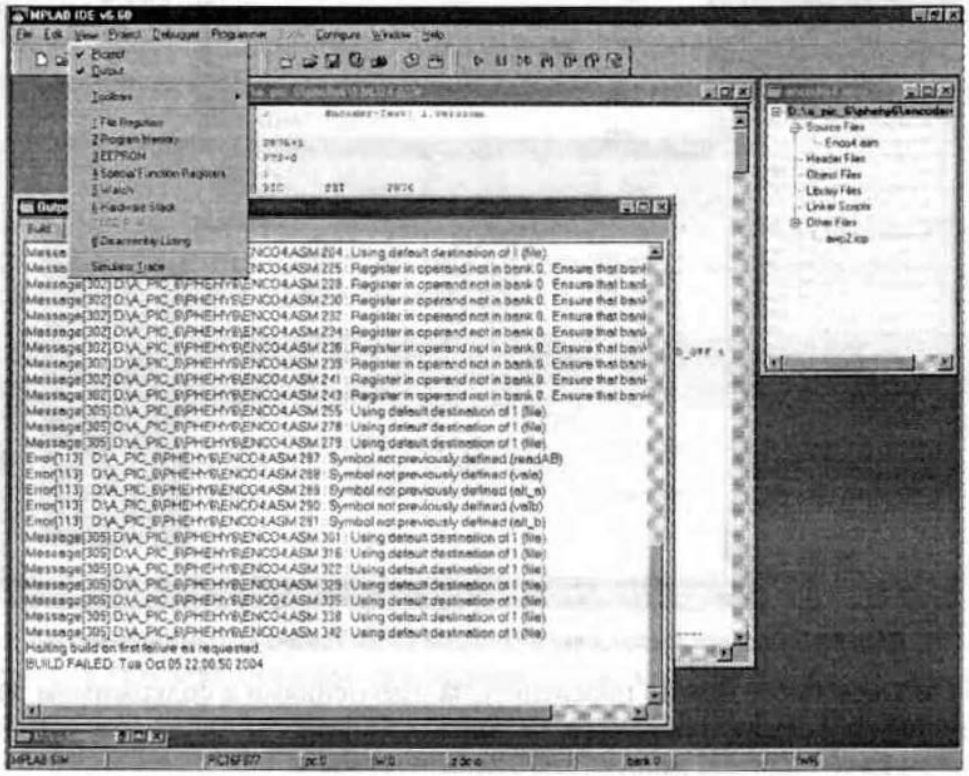


Рис. 8.12. Меню View, содержащее команду эмулятора Simulator Trace

Процесс эмуляции

Последняя команда меню **View** (**Simulator Trace**) отображается только в том случае, если в подменю **Debugger ▸ Select Tool** выбрана опция **MPLAB SIM**. Она представляет интерес исключительно в режиме эмуляции программы, поскольку открывает окно, в котором отображается запись для каждой выполняемой программы. Это окно называется **Trace**. Если в свободной области этого окна щелкнуть правой кнопкой мыши, то появится меню с различными командами режима трассировки (рис. 8.13).

Эти команды определяют информацию, отображаемую в окне **Trace**, а также ширину отдельных столбцов. Кроме того, для быстрого задания набора столбцов в окне **Trace** можно воспользоваться специальным контекстным меню, которое открывается по щелчку правой кнопкой мыши на заголовке списка (выделен серым цветом).

Раньше функции поиска в окне эмулятора **Trace** отсутствовали, и его содержимое приходилось копировать в окно редактора и только там уже осуществлять поиск. Теперь это затруднение устранено (и не только оно одно).

В каждой строке окна **Trace** указывается неизменяемое показание времени, которое может быть выражено в количестве циклов или в секундах (столбец **Time**). Это важно, прежде всего, в тех случаях, когда для кварца выбирается неопределенное значение. С помощью столбца **Time** удобно определять время выполнения подпрограмм (время завершения минус время начала).

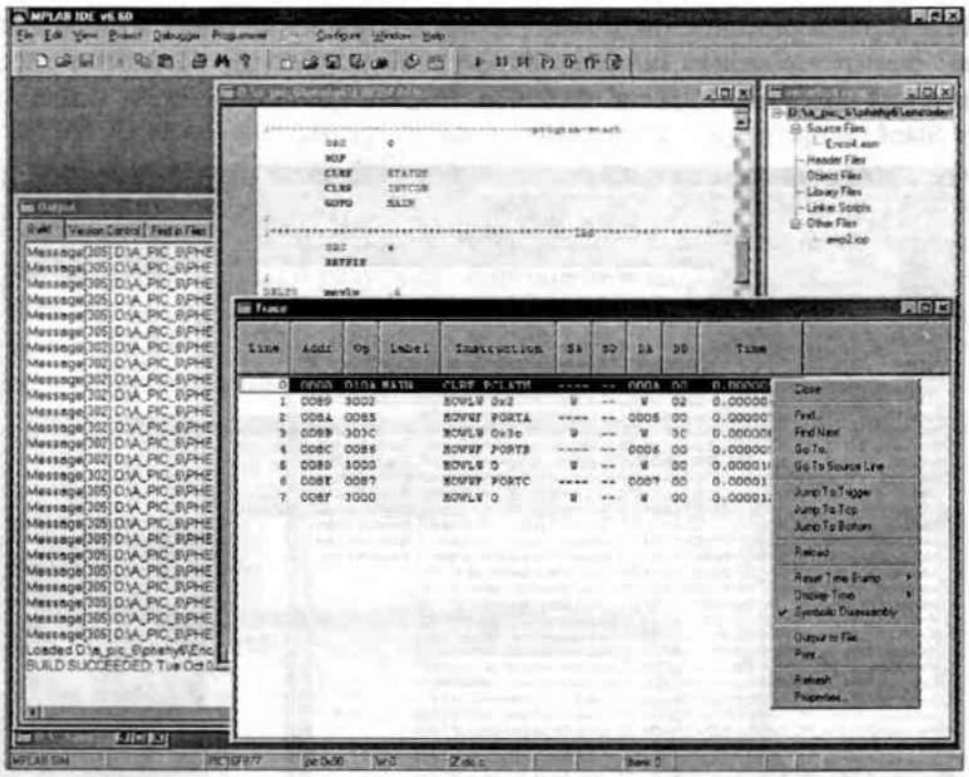


Рис. 8.13. Процесс трассировки и большой набор команд контекстного меню

В окне Trace также можно проследить за изменениями в содержимом регистров, используемых в качестве операндов текущей команды.

На этом рассмотрение режима эмуляции завершим.

8.3.4. Меню Project

В версии MPLAB 6.60 пункты меню **Project** (рис. 8.14), в основном, такие же как и в предыдущей версии, однако самая верхняя команда **(Project Wizard)** — новая. Она используется для запуска мастера создания проектов.

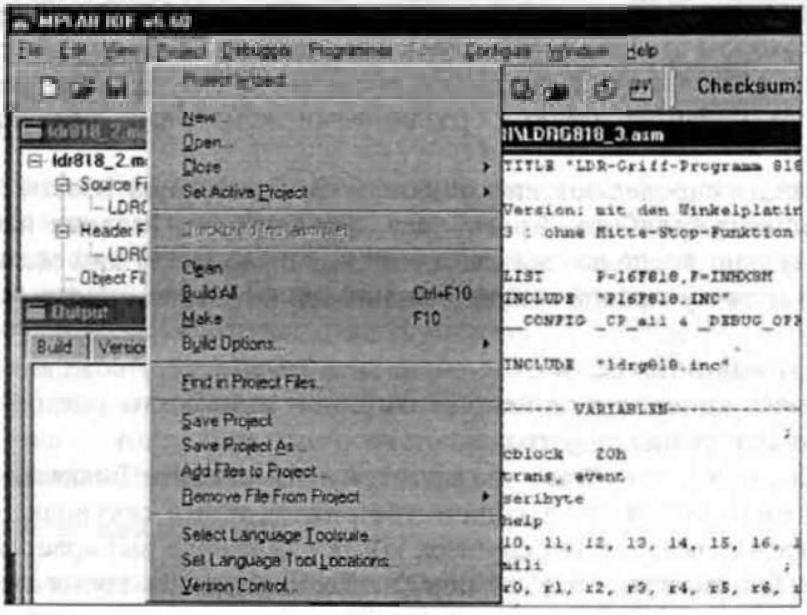


Рис. 8.14. Меню Project

Этот мастер задает пару простых вопросов и на основании ответов создает новый проект (см подраздел 8.3.2).

Следующие пункты меню **Project** служат для ручной обработки проекта, его компоновки и настройки параметров компиляции. Далее следуют команды ручного сохранения проекта, а также пункты выбора компилятора/ассемблера. Эти элементы меню **Project** уже должны быть знакомы пользователям предыдущих версий MPLAB. Последние команды необходимы только в том случае, если требуется добавить новое средство компиляции.

Удобное средство поиска

В меню **Project** появился один очень полезный пункт: **Find in Project Files**. С его помощью одним нажатием клавиши можно выполнить поиск всех файлов, ссылки на которые присутствуют в проекте (рис. 8.15).

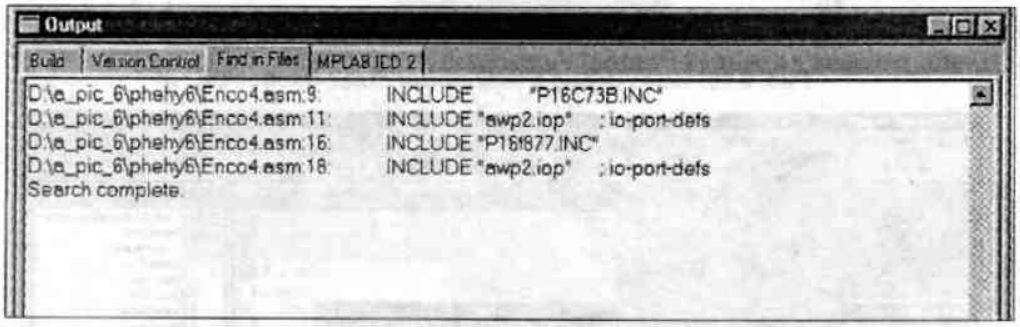


Рис. 8.15. Результат поиска файлов отображается на вкладке **Find in Files** окна **Output**

Раньше, когда включаемые файлы использовались в качестве "турбазы" для некоторых подпрограмм, они в проекте не отображались. Начиная с версии MPLAB 6.50, включаемые файлы можно соотнести с программным кодом с помощью ветки **Other Files** (Другие файлы) в окне проекта. Теперь, для того чтобы открыть такие файлы в окне редактора, достаточно дважды щелкнуть мышью на их имени. Раньше для просмотра всего исходного текста с целью поиска некоторого фрагмента, который мог быть разбросан по множеству файлов, приходилось загружать каждый файл в редактор и там его просматривать. Теперь этот процесс реализован намного проще: с помощью диалогового окна **Find in Project Files**.

8.3.5. Меню Debugger

Первый пункт меню **Debugger** представляет собой подменю выбора средства отладки (рис. 8.16). Варианты отладчиков для MPLAB 6.60 перечислены в табл. 8.1.

Таблица 8.1. Варианты средств отладки для MPLAB 6.60

Вариант	*Описание*
None	Отладчик не используется; только редактор
MPLAB SIM	Эмулятор для микроконтроллеров PIC за исключением dsPIC
MPLAB SIM30	Эмулятор для dsPIC
MPLAB ICD2	Небольшой внутрисхемный отладчик
MPLAB ICE2000	Два больших эмулятора с памятью трассировки и множеством других полезных свойств
MPLAB ICE4000	

После выбора некоторого отладчика в меню Debugger появляется ряд пунктов, соответствующих выбранному средству, а также отображается новая панель инструментов (рис. 8.17).

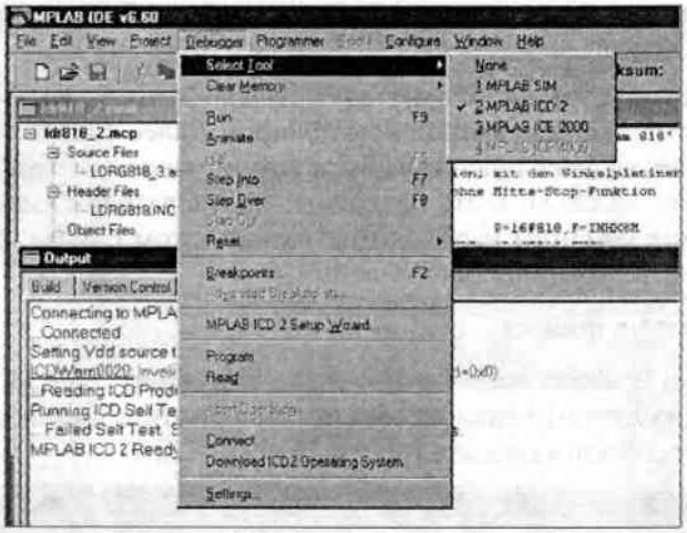


Рис. 8.16. Меню Debug с выбранным отладчиком ICD2

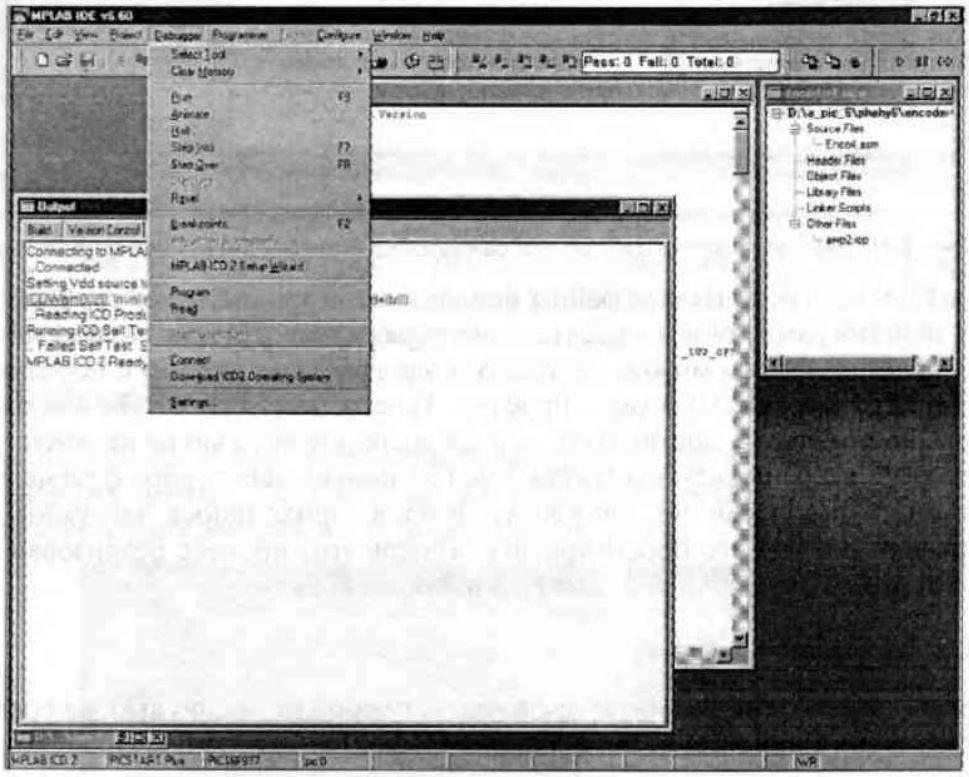


Рис. 8.17. Пункты меню Debug в случае, когда выбран отладчик ICD2

Без отладчика программу в MPLAB невозможно запустить, выполнить пошагово, эмулировать сброс и т.п.

Пункты с **Run** по **Reset** на рис. 8.17 — основные команды для управления ходом выполнения программы. Особо стоит упомянуть о команде **Step Out**. Раньше, если в ходе выполнения программы встречался вызов подпрограммы (команда CALL), то в случае непреднамеренного входа в эту подпрограмму по команде **Step Into** возникали проблемы. Теперь в такой ситуации можно воспользоваться командой меню **Step Out** (или соответствующей кнопкой панели инструментов), которая автоматически выполняет программу до команды RETURN, и "внешний мир" опять становится доступным. Тем не менее, к сожалению, для отладчика MPLAB ICD2 эта команда

неактивна (см. рис. 8.17). Другими словами, команда **Step Out** реализована не во всех эмуляторах.

И еще один инструмент отладки, имеющий отношение к ICD2... После каждого ассемблирования новую программу следует записывать в микроконтроллер PIC, однако в данном случае для этого используется не меню **Programmer**, а дополнительная команда в меню **Debugger**! Процесс записи не может быть автоматизирован, поскольку пользовательский код программируется в микроконтроллер только после успешного завершения отладки. Однако для этого ICD2 необходимо переключить из режима отладки в режим программирования, и только после этого записывать программу в PIC.

8.3.6. Меню Programmer

Если программатор еще не выбран, то следует установить флажок возле соответствующего пункта в подменю **Select Programmer** (рис. 8.18). В среде MPLAB 6.60 возможны следующие варианты выбора: "None" (программатор не выбран), PICSTART Plus, Pro Mate 2, MPLAB PM3 и ICD2.

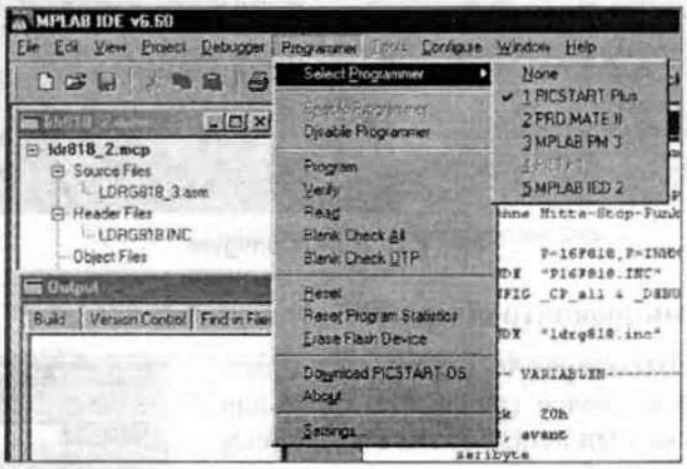


Рис. 8.18. В меню Programmer выбран программатор PICSTART Plus

Примечание

Количество доступных программаторов может оказаться меньше в зависимости от того, поддерживается ли то или иное средство выбранным типом микроконтроллера. Это также относится и к средствам отладки.

И еще одно замечание относительно ICD2... Данное средство не может быть одновременно определено как отладчик и программатор. Это обусловлено тем, что при отладке с помощью ICD2 внутренне включаются определенные регистры и в микроконтроллер PIC программируется специальный фрагмент кода (так называемый "debug executive"). Само собой, при "обычном" программировании ни того, ни другого не происходит. Таким образом, возникает конфликт одновременного использования ICD2 для двух целей.

После того как программатор выбран, в меню **Programmer** появляется соответствующий набор пунктов. Среди стандартных пунктов можно назвать **Blank check**, **Read**, **Program** и **Verify**. В нижней части меню **Programmer** находятся команды для загрузки в программатор нового операционного обеспечения и настройки параметров связи (порт и скорость передачи).

Загрузка программного обеспечения, которая раньше применялась для Promate и ICD2, теперь также доступна и для Picstart. При этом, разумеется, должно быть выполнено аппаратное расширение программатора путем включения контроллера флэш-памяти, чтобы можно было использовать микроконтроллеры PIC18 (или же приобрести новые PICStart Plus на базе технологии Flash).

8.3.7. Меню Tools

Это меню в версии MPLAB 6.60, по сути, не реализовано. Посмотрим, что мы здесь найдем в будущем.

8.3.8. Меню Configure

Меню **Configure** (рис. 8.19) состоит из пяти пунктов, первый из которых используется для выбора типа микроконтроллера PIC.

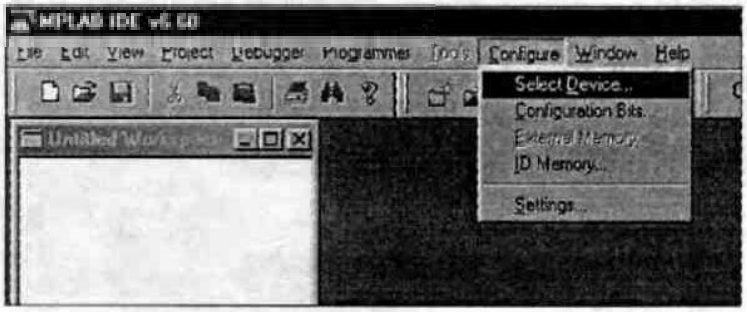


Рис. 8.19. Меню Configure

Выбор типа микроконтроллера PIC

Тип микроконтроллера PIC выбирается в диалоговом окне **Select Device** (рис. 8.20). Если для создания проекта не был использован Project Wizard, то к этому окну обращаются в первую очередь.

Здесь также показано, какое средство программирования выбрано для данного устройства, и какой может быть использован отладчик. Для больших эмуляторов в окне **Select Device**, кроме всего прочего, указывается, какой требуется модуль процессора.

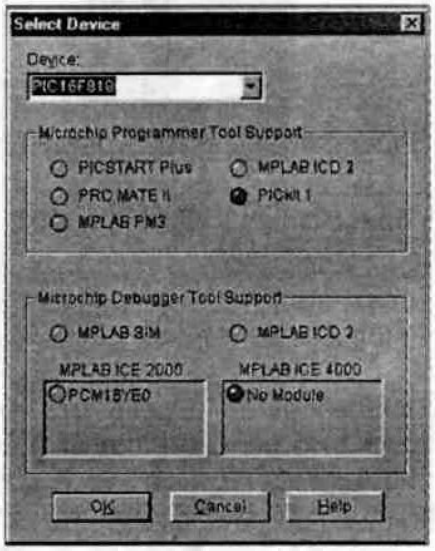


Рис. 8.20. Диалоговое окно выбора типа микроконтроллера PIC

Установка разрядов конфигурации

Вторая команда мню **Configure** служит для вызова окна (рис. 8.21), в котором можно установить разряды конфигурации микроконтроллера PIC (в файле с исходным кодом задаются с помощью директивы `__CONFIG`). Временное изменение этих разрядов имеет смысл, к примеру, в том случае, когда микроконтроллер передается заказчику на испытания.

Так, с помощью разряда "Code Protect" можно защитить от чтения программный код. Согласитесь: ни один разработчик не захочет, чтобы его программа была прочитана конкурентом.

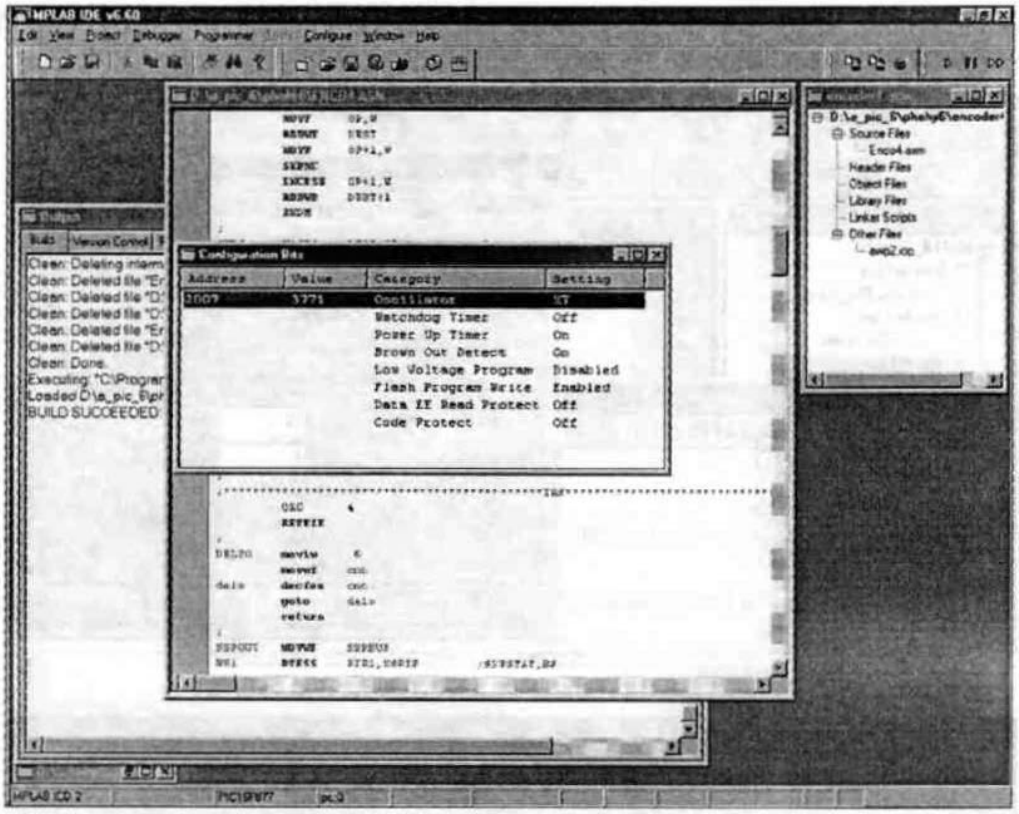


Рис. 8.21. Диалоговое окно Configuration Bits

Внешняя память

Команда **External Memory** служит для подключения внешней памяти к микроконтроллерам, наподобие PIC 18C601.

Редактирование ID-областей памяти

Четвертая команда меню **Configure** дает пользователю возможность задать любую числовую ID-строку, которую затем можно считать даже из защищенного от чтения микроконтроллера PIC. Альтернативный способ задания ID-строк — с помощью директивы `__idlocs` в исходном тексте программы.

При желании, здесь можно также задать автоматическое создание контрольной суммы для программы, что в дальнейшем можно использоваться для идентификации версий.

Настройка параметров

Последняя команда меню **Configure** открывает диалоговое окно настройки параметров **Settings**. Параметры, размещенные на вкладке **Workspace** (рис. 8.22), совершенно безобидны: они отвечают только за автоматическую загрузку и сохранение, а также за количество отображаемых в меню **File** файлов и рабочих областей, использованных последними.

На вкладке **Program Loading** указывают, когда и какая память должна быть очищена.

Последние важные параметры настраиваются на вкладке **Projects** (рис. 8.23). В частности, здесь включают/отключают модель "one-to-one". Если флажок **Use one-to-one project-workspace model** установлен, то понятия "проект" и "рабочая область"

становятся равнозначными, в противном же случае для одной рабочей области можно определить несколько проектов.

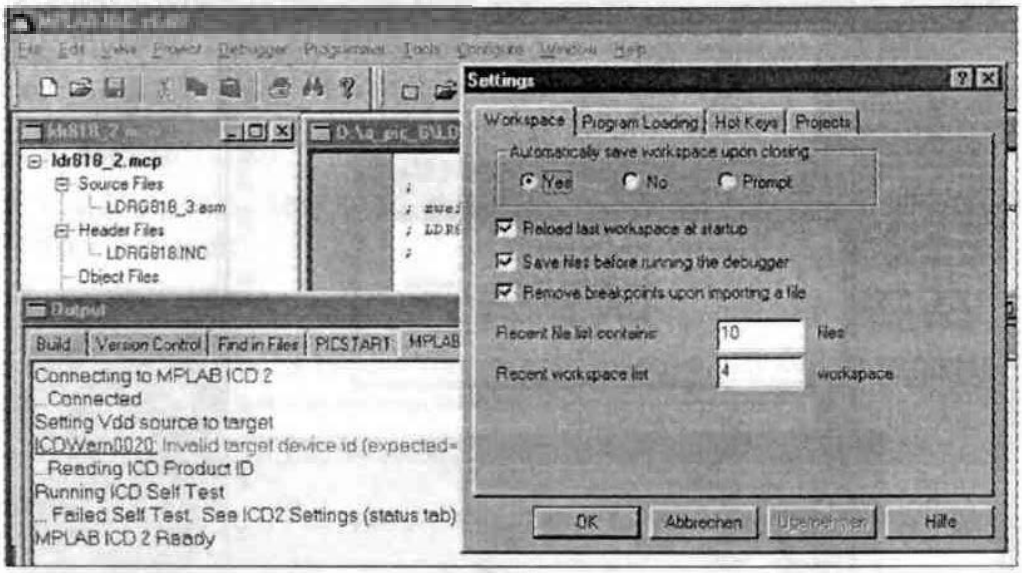


Рис. 8.22. Вкладка Workspace диалогового окна Settings

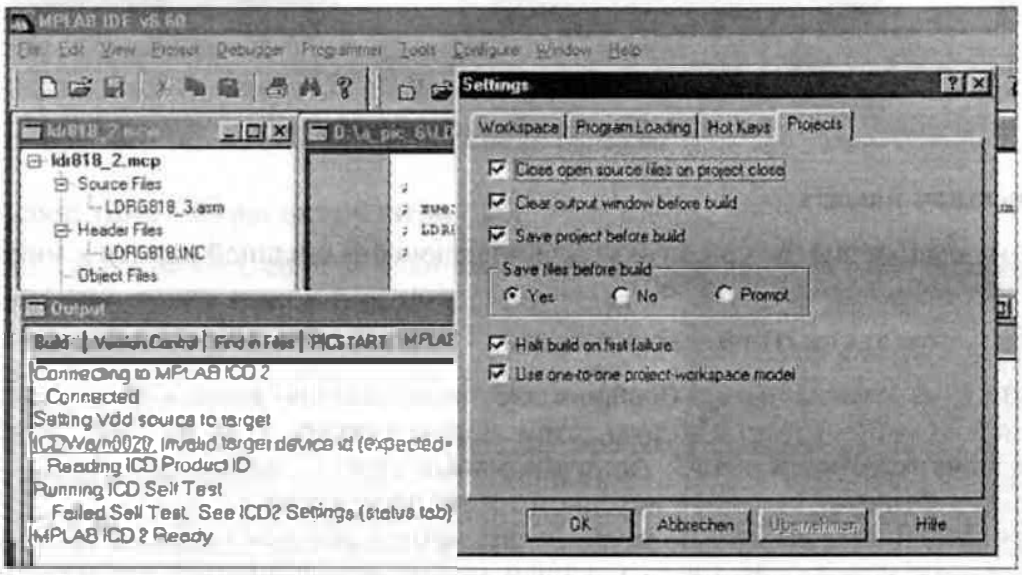


Рис. 8.23. Вкладка Projects диалогового окна Settings

8.3.9. Меню Window

Команды, расположенные в верхней части меню **Window**, отвечают за работу с отдельными окнами, например, команда **Close All** закрывает все открытые окна. **Tile Horizontally** — распределяет их на окне равномерно по вертикали, а **Cascade** — создает каскадное размещение.

В нижней части меню **Window** перечислены открытые в данный момент окна (даже те из них, которые не видны на экране). Выбрав один из таких пунктов, можно быстро перейти к соответствующему окну.

Больше об этом меню сказать нечего, поскольку оно — стандартно для всех Windows-приложений.

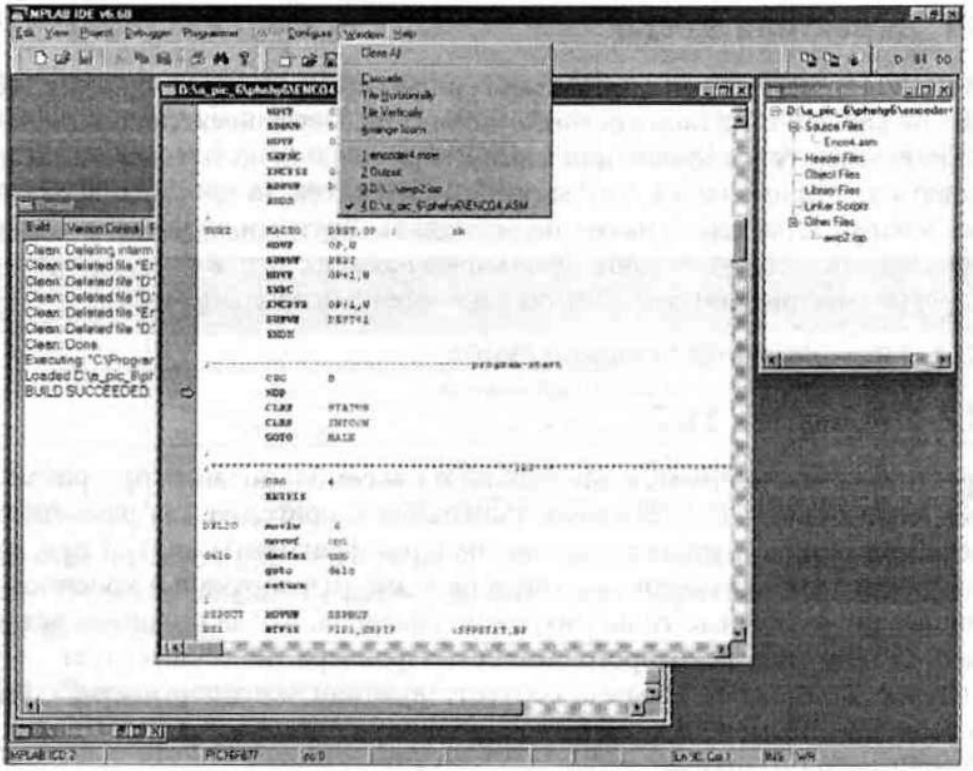


Рис. 8.24. Стандартное меню Window

8.3.10. Меню Help

Меню **Help** — общепринятое для Windows-программ. Справочная система MPLAB 6.60 разбита на категории. Например, в первом разделе в общих чертах описаны понятия проекта и рабочей области, а также различия между ними. Также рассматривается переход с версии MPLAB 5.XX на версию 6.XX. При этом затрагиваются различия в компонентах, интегрированной среде разработки, редакторе, ассемблере, эмуляторе и программаторах. Здесь вы также найдете обширный предметный указатель, а также поиск по ключевому слову.

В завершение отметим, что компания Microchip предлагает услуги информационного обслуживания, извещая о выходе новых версий MPLAB или отчетов об ошибках для того или иного микроконтроллера.

8.4. Ассемблер MPASM

Ассемблер MPASM — основная составная часть среды MPLAB. Он поддерживается большинством микроконтроллеров PIC и даже языком серии PIC18. При этом в нем используются не только новые команды, но также и новые директивы.

Классические директивы ассемблера не порождают кода, а только сообщают о том, каким образом должен транслироваться код, или для какого микроконтроллера предназначена данная программа. К тому же, применяются соглашения об именах и адресах.

В отличие от этого, в MPSAM существует ряд порождающих код директив, например, `DB`, `FILL`, `BANKSEL`, `PAGESEL`. Ниже кратко рассматриваются некоторые из основных директив ассемблера с примерами их применения. При этом будем придерживаться последовательности, характерной для их следования в программах.

8.4.1. Директива TITLE

С помощью этой директивы задают выражение, которое будет отображено в "шапке" на каждой странице листинга. Если существует несколько версий одной и той же программы, то заголовок или строки под ним можно (точнее сказать, нужно) использовать для примечаний, чтобы при открытии текста программы сразу же было видно, какими характеристиками она обладает. Другими словами, в таких примечаниях указывается статус версии. Очень полезно включать в них строку с номером версии программы, именем разработчика и внесенными изменениями.

```
TITLE "Programm 460frede5-REV00"
```

8.4.2. Директива IF

Директива IF используется для условного ассемблирования программ. Она необходима, если некоторая программа должна быть пригодна для различных ситуаций, когда при одних условиях применимы одни фрагменты кода, а при других — другие. Пример такой ситуации, — когда на этапе разработке и в конечном устройстве используют различные типы микроконтроллеров. Когда подобное встречается? Например, когда для некоторого микроконтроллера не существует эмулятора. В этом случае, применяют образец, соответствующий заменяемому набором проверяемых выводов. Возможно также, что заказчику поставляется "эрзац" для обеспечения своевременного производства.

В представленном ниже примере в первых двух строках находится определение двух возможных типов. В третей строке "переменной" PIC назначается значение выбранного типа.

Далее следуют три блока, которые ассемблируются только по выполнению условия. В первом из них осуществляется привязка соответствующей части объявлений, а во втором и третьем — выбор аппаратного модуля в зависимости от конкретной ситуации. С помощью такой методики можно легко включать и отключать подпрограмму тестирования, реализованную в микроконтроллерах PIC нового поколения. Аналогичным же способом может быть выполнена задача диагностирования.

```
P628=1
P715=0
PIC SET P628                    ; Задаем тип микроконтроллера
IF PIC == P715
LIST P=16C715,F=INHX8M
INCLUDE "P16C715.INC"
__CONFIG +CP+OFF&...&_XT_OSC
ENDIF

IF PIC == P715
MOVLW 0
MOVWF ADCON0
ENDIF

IF PIC == P628
MOVLW 07H
MOVWF CMCON
ENDIF
```

8.4.3. Директива LIST

С помощью директивы LIST ассемблер формирует различные сообщения. Самая важная опция этой директивы — объявление используемого процессора (хотя, это можно сделать и с помощью директивы PROCESSOR). Перечень некоторых опций директивы LIST представлен в табл. 8.2.

Таблица 8.2. Некоторые опции директивы LIST

Опция	Значение по умолчанию	Описание
c=nnn	132	Ширина колонки в LST-файле
n=nnn	59	Количество строк на страницу
f=<формат>	inhx8m	Формат HEX-файла
p=<тип>	-	Все типы PIC16 и PIC17
r=<основание>	hex	Шестнадцатеричное, восьмеричное, десятичное

Описание всех опций директивы LIST можно найти в справочной системе в разделе "Directive language", посвященном языку директив MPASM. Пример использования см. выше.

8.4.4. Директива INCLUDE

С помощью этой директивы загружаются внешние файлы, которые совместно транслируются во время ассемблирования. Директива INCLUDE помещается в тексте программы в точности в том месте, где должен быть вставлен внешний текст. Ее размещение особенно важно, если таким образом программный код связывается с подпрограммами и таблицами.

Таким способом, как правило, подключаются стандартные определения в "шапке" программы. Они находятся в так называемых "заголовочных файлах", которые компания Microchip предоставляет для каждого типа микроконтроллеров PIC. Их можно найти в папке \MPLAB IDE\MCHIP_Tools.

Кроме того, с помощью директивы INCLUDE можно подключать специальные файлы подпрограмм или файлы с макроопределениями (пример см. выше).

8.4.5. Директива __CONFIG

С возрастанием числа разрядов конфигурации конфигурацию рационально определять сразу же в "шапке" программы. В результате она "вольется" прямо в HEX-файл, который, следовательно, будет содержать всю необходимую информацию. Как сказано выше, в разделе, посвященном интегрированной среде разработки MPLAB, разряды конфигурации можно, при желании, изменить до начала программирования микроконтроллера.

Запомнить, какие разряды конфигурации присутствуют в том или ином микроконтроллере PIC, и как они называются, вряд ли возможно — да и не нужно. Перечень всех таких разрядов находится в конце вышеупомянутых заголовочных файлов. В них также можно найти примеры использования директивы __CONFIG.

8.4.6. Директива _IDLOCS

В микроконтроллерах PIC, защищенных от чтения, кроме разрядов конфигурации, доступны также области памяти, обозначенные с помощью идентификаторов. В эти ячейки памяти можно запрограммировать программный код, данные, имена

или контрольные суммы. В дальнейшем они используются для идентификации кристалла, и потому называются ID-строками.

```
__IDLOCS 0x1234
```

8.4.7. Директива EQU

С помощью директивы EQU некоторому выражению назначают значение. При этом могут быть использованы адреса, константы и адреса переменных. Тем не менее, авторы книги предпочитают определять переменные с помощью директивы CBLOCK.

8.4.8. Директива CBLOCK

Директива CBLOCK предоставляет возможность назначать адреса для нескольких переменных. При этом могут обрабатываться многобайтные переменные.

```
CBLOCK 20H
LAB1, LAB2, LAB3        ; Комментарий
LAB4:3                  ; Переменная LAB4 состоит из 3-х байт
LAB5, LAB6              ; Комментарий
ENDC
```

Если раньше могло случиться так, что двум переменным ошибочно сопоставлялся один и тот же адрес, то в случае с директивой CBLOCK это больше невозможно.

Если требуется двоякий способ обращения к какой-либо многобайтной переменной (например, к LAB4), то на сцену выходит директива EQU. Разумеется, к переменной LAB4 можно обратиться просто по ее имени как к целому слову. Но также существует возможность обращения к каждому байту в отдельности. Для этого после представленной выше директивы CBLOCK помещают следующие строки:

```
LAB4L EQU LAB4
LAB4M EQU LAB4+1
LAB4H EQU LAB4+2
```

Таким образом, адресу переменной LAB4 просто сопоставляется еще одно имя: LAB4L.

8.4.9. Директива #DEFINE

С помощью директивы #DEFINE некоторой строке можно сопоставить любую другую строку. Эта директива очень часто используется для обозначения разрядов. Несмотря на то, что поразрядные команды, наподобие BSF и BCF, имеют два аргумента (рабочий регистр и номер разряда), благодаря директиве #DEFINE все выражение можно определить под одним именем.

Например, если сигнальный выход назначен выводу 2 порта B, то его можно обозначить именем SIGNAL, а в части определений указать следующую директиву:

```
#DEFINE SIGNAL PORTB,2
```

Вопрос в том, какие преимущества и недостатки имеет подобное именование. В случае использования выводов портов в качестве сигнальных линий их имена дают пользователю четкое представление об их назначении. Теперь программист может не задумываться, на какой порт и вывод подается тот или иной сигнал. При

этом следует учитывать, что распределение выводов в процесс разработке проекта вполне может измениться. По этой причине выводы портов, как правило, определяют с помощью директивы #DEFINE.

Если же разряды находятся в переменных, предназначенных для управления или регистрации ошибок, дело обстоит иначе. В этом случае программисту вполне может потребовать знать, в какой переменной расположен тот или иной разряд, каким образом его распознать, какое значение он имеет. Также нередко возникает ситуация, когда при разработке программы организуются циклы поиска по переменным, которые не сработают в случае именования, подобного тому, которое используется при обработке сигналов портов. Кроме того, иногда разные переменные называют одними и теми же именами. Учитывая все вышесказанное, для именования разрядов в обычных регистрах директиву #DEFINE не используют, а только применяют директиву EQU для номера разряда.

Пример:

```
BSF INTCON,GIE
```

Смысл этого примера очевиден: здесь просто выполняется разрешение прерываний. Другими словами, в программе используются имена, хорошо знакомые каждому пользователю. Изменять такие имена не рационально, поскольку это приведет к путанице.

С увеличением числа "блуждающих разрядов" (то есть, таких, которые в разных типах микроконтроллеров PIC могут оказаться в разных регистрах), компания Microchip для надежности определяет разряды в заголовочных файлах. При этом используются два способа записи. Пример для разряда GIE:

```
INTCON  EQU 0BH
GIE     EQU 07H
#DEFINE _GIE 0BH.07H
```

В результате рассмотренный выше вызов команды BSF можно представить в следующем виде:

```
BSF _GIE
```

8.4.10. Директива ORG

Эта директива сообщает ассемблеру, в какую ячейку памяти программ должно быть записано следующее слово программного кода. Пример:

8.4.11. Директивы BANKSEL и PAGESEL

С помощью этих двух инструкций ассемблеру сообщается, какие должны быть установлены банк и страница. Поскольку в данном случае речь идет о макросах, мы можем сослаться на соответствующую главу книги.

```
BANKSEL PORTB
```

Эта команда приводит к переключению на банк 0. В зависимости от того, сколько банков присутствует в микроконтроллере PIC, могут быть изменены два разряда или один разряд (или вообще ни одного).

8.4.12. Директива FILL

Эта директива ассемблера дает программисту возможность заполнить все неиспользуемые ячейки памяти программ (а значит — и программный код) некоторым определенным значением.

Основное применение этой директивы — заполнение участков незадействованной памяти командами GOTO. Это бывает полезно на этапе разработки, когда в результате ошибки программиста программа оказывается в таком положении, в котором никогда не должна была бы оказаться. Если неиспользуемую область памяти заполнить кодом перехода, наподобие GOTO FORREST, то перед ней должна существовать метка FORREST с командой NOP, за которой следует точка прерывания и код, извещающий об ошибке.

Из соображений безопасности, избыток команд GOTO рационально использовать даже в окончательной версии программы, предназначенной для тиражирования. Это обосновано тем, что теоретически к сбою может привести любой дефектный бит или сильная электромагнитная помеха. В этом случае, благодаря дополнительным командам GOTO, программа сможет быстро вернуться в корректное состояние.

Еще одна типичная область применения директивы FILL — формирование данных для внешней памяти. В этом случае директива имеет два параметра: первый — символ, используемый в качестве заполнителя, а второй обозначает число повторений.

Пример двух последних строк программы:

```
FILL (Goto 0), 1FFEH-$
END
```

8.4.13. Директива END

Как и в любом другом ассемблере, в MPASM существует директива END, завершающая программу. Все символы, расположенные после этой директивы, игнорируются. Кроме того, если вы забудете указать ее в конце программы, то ничего страшного не произойдет — ассемблер это распознает.

8.4.14. Формирование с помощью MPASM данных для памяти EEPROM

Для этой цели существует еще несколько директив. Перечислим их перед тем как рассмотреть соответствующий пример.

Директива DATA

С помощью этой директивы байты, слова, символы и текст последовательно размещаются в памяти программ. Если установлено 16-тиразрядное слово, а количество байт явно не задано, то половина последнего слова остается пустой (то есть, содержит нули).

Директива DATA очень полезна для тех типов микроконтроллеров, которые могут обращать к памяти программ напрямую (не только с помощью команды RETLW). В противном случае, данные в памяти программ имеют только документированное значение.

Директива RES

Для резервирования ячеек памяти можно воспользоваться директивой RES. При этом всегда оперируют словами. Если ширина ячеек установлена на восемь разрядов, то даже в случае директивы RES 8 будет зарезервировано 16 байт. Для таких ячеек не предусматривается запись в HEX-файле в формате Intel.

Директива DB

Назначение этой директивы — такое же как у директивы DATA. Символы или слова последовательно размещаются в памяти. Если их количество — нечетное, и определено 16-тиразрядное слово, то половина последнего слова, опять таки, остается нулевой.

Директива DE

Название этой директивы происходит от "Define EEPROM-data". Она предназначена для внутренней памяти EEPROM таких микроконтроллеров, как PIC 16C84, однако обращается с данными несколько иначе, чем предыдущие рассмотренные директивы. В случае восьмиразрядной ячейки памяти символ каждый раз определяется как байт. В случае же 16-тиразрядной ячейки первый байт слова остается пустым. Значения больше 255 не допускаются, что, как уже было сказано ранее, соответствует одному символу или байту.

Директива DT

В случае восьмиразрядных ячеек, директива DT служит для извещения об ошибках. Для формирования EEPROM-данных с помощью ассемблера MPASM эту директиву не применяют. Ее назначение — получить команду RETLW. Образцы синтаксиса директивы DT представлены ниже на примере фрагмента, взятого из одного файла листинга.

```
0010    3400 3417   DT    BMUL,WMUL
0012    3441        DT    'A'
0013    3441 346E   DT    "Anne"
        346E 3465
```

Директива DW

Эта директива также служит для извещения об ошибках в случае восьмиразрядных ячеек, однако размещает все заданные значения в словах.

Пример использования

В представленном ниже листинге показано, каким образом с помощью ассемблера можно формировать восьми- и шестнадцатиразрядные данные. Используемые команды уже были рассмотрены выше. Для какое памяти EEPROM предназначены эти данные: для последовательной или для параллельной — не играет никакой роли. Ассемблер MPASM может создавать выходные HEX-файлы в трех форматах Intel.

```
;EEPROM-данные
        LIST    P=EEPROM8       ; Альтернатива - EEPROM16
        INCLUDE "MEMORY.INC"
        LIST    M=_24C08B
        ORG     0
ANFTXT  DATA    "abcdABCD"
```

```
        FILL    0X06,20
RSVRT   RES     8                   ; Зарезервировано 8x2 байт
        VARIABLE I
I = 0
LOOPANF WHILE   I < 0X0A
        DATA    I
I += 1
LOOPEND ENDW
BLOCKD  DB      1,2,4,5,'a'
        ORG     100
BLOCKT  DE      "my program",5
        END
```

С помощью ассемблерной переменной I и цикла WHILE мы сформировали байты данных, содержащие значения от 00 до 09. Для этой цели можно также применить и другие конструкции.

Более подробно все директивы MPASM рассмотрены в справочной системе, где можно найти поясняющие примеры.

ГЛАВА

9 ICD2 — внутрисхемный отладчик и программатор

ICD2 обладает свойствами и характеристиками, которые отсутствуют в обычном эмуляторе. Представляем сравнительную характеристику ICD2 и эмулятора (ICE).

ICE — относительно большой и дорогостоящий инструмент со множеством сменных процессорных модулей, формирующих настоящий полнофункциональный PIC.

ICD2 — небольшой, удобный в обращении и дешевый. Он представляет собой инструмент, работающий с реальным микроконтроллером PIC в схеме. Для этого схема должны быть завершенной и корректной, включая электропитание и тактовый генератор.

С точки зрения управления и удобства работы, также можно отметить различие: с помошью ICE программу можно написать именно так, как она задумана, а в случае ICD2 приходится идти на некоторые компромиссы.

Это обусловлено тем, что некоторые ресурсы микроконтроллера должны использоваться совместно с ICD2:

- память программ;
- память данных;
- выводы (RB6 и RB7);
- от одного до двух уровней стека.

Кроме того, требуется подготовка в виде некоторых схемных изменений. Различиями в удобстве также не стоит пренебрегать. В этом смысле можно привести следующие весомые аргументы:

- используются только точки прерывания;
- отсутствует память трассировки;
- считывание переменных происходит гораздо дольше, в результате чего в окне **Watch** приходится использовать как можно меньше переменных; окно **EEPROM** закрыто.

А теперь перечислим свойства ICD2, превосходящие свойства эмулятора:

- ICD2 может подключаться к готовым образцам всего лишь с помощью пяти линий; при этом извлекать микроконтроллер PIC не нужно.
- С помощью этого интерфейса ICD2 может в любой момент заново перепрограммировать впаянный микроконтроллер.

И что же из этого следует?

- На этапе разработки ICE имеет явное преимущество.

- С помощью ICD2 систему можно без проблем протестировать с помощью диагностического программного обеспечения (не в режиме отладки) или обновить микропрограмму до последней версии без необходимости выпаивать микроконтроллер.

Единственным условием является уже упомянутая подготовка. Проект должен включать в себя интерфейс I^2C, что для некоторых линий подразумевает определенные дополнительные затраты. Кроме того, при отладке следует соблюдать некоторые условия, имеющие отношение к программному обеспечению (ресурсы микроконтроллера). Но на этом мы остановимся подробнее чуть позже.

ICD2 — инструмент двойного назначения, однако в среде MPLAB его нельзя одновременно использовать как отладчик и программатор.

Для всех микроконтроллеров PIC, поддерживающих последовательное внутрисхемное программирование, ICD2 может применяться в качестве программатора. Внутрисхемное последовательное программирование (английская аббревиатура ICSP — In-Circuit Serial Programming) подразумевает, что микроконтроллер не выпаивается.

Для небольшого ряда микроконтроллеров PIC ICD2 может также использоваться как отладчик. При этом также различают возможность отладки с "обычным" PIC или с так называемым "пробником" (табл. 9.1).

Таблица 9.1. Применение монтажных колодок для различных типов PIC

Тип PIC	*Вспомогательные средства*
PIC10F20X	С пробником AC162059
PIC12Cxxx и 12Cexxx	Без вспомогательных средств
PIC12F508/509	С пробником AC162059
PIC12F629/675	С пробником AC162050
PIC12F635	С пробником AC162057
PIC12F683	С пробником AC162058
PIC14000	Без вспомогательных средств
PIC16C5x	Без вспомогательных средств
PIC16C6x	Без вспомогательных средств
PIC16C7x	Без вспомогательных средств
PIC16C9x	Без вспомогательных средств
PIC18CE6xx	Без вспомогательных средств
PIC16F5x	Без вспомогательных средств
PIC16F505	С пробником AC162059
PIC16F7x	Без вспомогательных средств
PIC16F84A	Без вспомогательных средств
PIC16F87/88	Без пробника
PIC16F627/628	Без вспомогательных средств
PIC16F684	С пробником AC162055
PIC16F627A/628A/648A	С пробником AC162053
PIC16F630/676	С пробником AC162052
PIC16F688	С пробником AC162056
PIC16F716	С пробником AC162054
PIC16F7x7	Без пробника
PIC16F81x/87x	Без пробника
PIC17Cxxx	Без вспомогательных средств

Таблица 9.1. Окончание

Тип PIC	Вспомогательные средства
PIC18Cxxx	Без вспомогательных средств
Исключение — PIC18C601/801	Без пробника
PIC18Fxxx/xxxx	Без пробника
rfPICxxxx	Без вспомогательных средств
dsPIC30Fxxxx	Без пробника

Для того чтобы сделать возможной отладку, соответствующий микроконтроллер PIC устанавливается в пробник. Судя по внешнему виду колодки AC162050, в нее устанавливается микроконтроллер, у которого количество выводов больше реально существующего для PIC данной серии. Дело в том, что здесь используется специальная ICD-микросхема, а в конечном устройстве задействованы только восемь выводов (рис. 9.1).

В случае 14-тивыводных микроконтроллеров PIC16F630 и 676 поступают подобным же образом: для реализации отладки без потери выводов здесь также применяют особую ICD-микросхему (рис. 9.2).

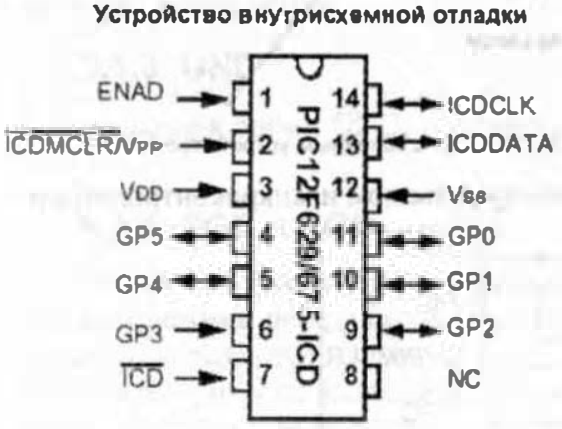


Рис. 9.1. Пробник ICD2 для восьмивыводных микроконтроллеров PIC

Рис. 9.2. Микросхема эмуляции для 14-тивыводных микроконтроллеров PIC

Для того чтобы определить, какие микроконтроллеры PIC поддерживают непосредственный режим отладки, обращайтесь к разделу "Development Systems Products" руководства "Product Selector Guide", ранее названного "linecard". Там перечислены все типы PIC и соответствующие им инструменты разработки. То, какой требуется пробник (и требуется ли он вообще), можно увидеть в столбце "ICD2".

Если теперь в конечной системе необходимо реализовать интерфейс ICD, то это будет зависеть от того, поддерживает ли используемый микроконтроллер PIC непосредственную отладку или нет. Если поддерживает, то аппаратную часть системы можно выполнить завершенной, если же требуется пробник, то оригинальный микроконтроллер необходимо удалить и заменить его пробником. После разработки микропрограммы пробник больше не нужен.

Затем можно перепрограммировать и запустить любое тестовое программное обеспечение или новейшую версию конечной программы. Однако отладка теперь будет недоступна — только просмотр регистров или установка точек прерывания.

А теперь поговорим подробнее об интерфейсе ICD...

9.1. Интерфейс ICD2

Штекерный разъем между ICD2 и конечным устройством может быть любым, однако предпочтительно использовать оговоренный компанией Microchip соединитель RJ45, поскольку в этом случае можно организовать непосредственное подключение стандартного кабеля. На наш взгляд, лучше всего это проиллюстрировано в руководстве пользователя ICD2 (рис. 9.3).

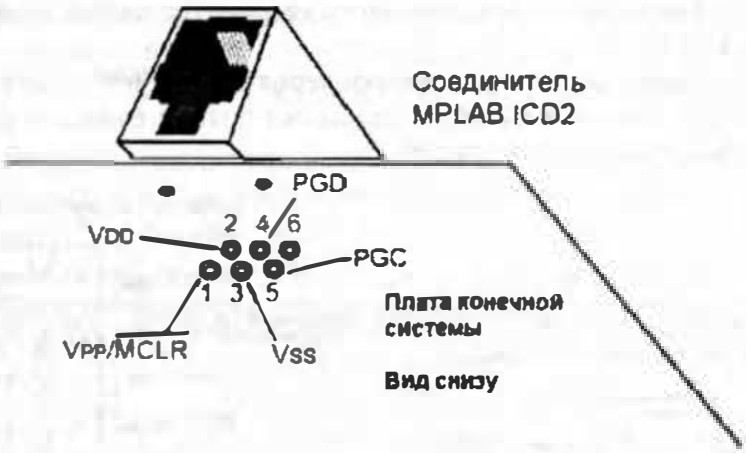


Рис. 9.3. Шестиконтактный RJ-разъем для соединения с конечным устройством

На рис. 9.4 показана схема соединений между разъемом и микроконтроллером.

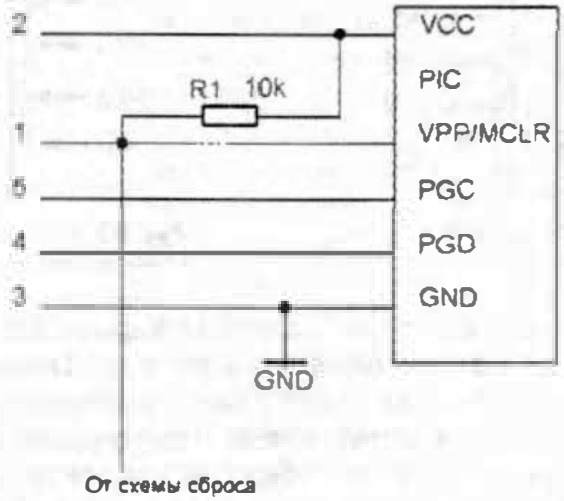


Рис. 9.4. Соединения в RJ-разъеме

Рассмотрим каждую линию подробнее.

9.1.1. MCLR

Линия MCLR обычно соединена с линией VCC через резистор сопротивлением 10 кОм. При этом исходят из того, что схема сброса за заданное время переводит

линию в состояние низкого уровня. Обычно выход схемы сброса — высокоомный. Поскольку через этот вывод микроконтроллер PIC переводится в режим программирования путем подачи напряжения V_{PP}, выход схемы сброса должен выдерживать напряжение около 13 В. Помехоподавляющий конденсатор на выводе MCLR может привести к проблемам! Если скорость увеличения V_{PP} окажется некорректной, возникнут неприятности.

9.1.2. VCC

Через эту линию ICD2 во время отладки может обеспечивать целевое устройство рабочим напряжением 5 В. Тем не менее, целевое устройство может использовать и собственное напряжение от 2,5 до 5,5 В. Важно отметить, что на вывод VCC всегда должно подаваться напряжение питания, иначе ICD2 не будет работать. С одной стороны, ICD2 следит за тем, есть ли напряжение на этом выводе, а с другой стороны напряжение V_{CC} питает драйвер в ICD2.

При программировании микроконтроллера напряжение питания всегда обеспечивается ICD2. Это происходит, к примеру, с помощью универсального модуля программирования. Следует обращать внимание на то, чтобы вывод электропитания микроконтроллера PIC был изолирован от остальных линий питания. Обычно это реализуют с помощью диода Шоттки небольшого номинала. Конденсатор, если его емкость превышает значение 0,1 мкФ, размещают по другую сторону диода. Излишняя нагрузка VCC может повлиять на скорость повышения напряжения, что приведет к проблемам.

9.1.3. GND

Благодаря линии GND, обеспечивается одинаковый для всех базовый потенциал.

9.1.4. PGC и PGD

Все взаимодействие ICD2 с микроконтроллером PIC происходит в последовательном режиме передачи через выводы PGC и PGD. Сигналы (синхроимпульсы и данные) передаются очень быстро, поэтому конденсаторы относительно линии GND категорически запрещены. Кроме того, можно не применять подтягивающих резисторов, поскольку сформированный резисторами ICD2 делитель искажает уровень напряжения.

Вышеупомянутые сигналы должны передаваться от ICD к микроконтроллеру PIC через непосредственное соединение. В большинстве типов PIC сигналы PGD и PGC выдаются на выводы RB6 и RB7. В режиме отладки ICD-интерфейс необходимо зарезервировать. Если ICD используется только для внутрисхемного программирования, то эти выводы можно применять в обычном режиме.

9.2. Режим отладки

Для того чтобы некоторую программу можно было проверить в PIC с помощью отладчика, в PIC вначале программируют прикладную программу и "ядро отладки". При этом необходимо учитывать ряд факторов:

- в системе разработки MPLAB инструментарий ICD2 должен быть настроен исключительно на роль отладчика (не может быть одновременно и программатором);

- вполне логично, что схема ICD2 должна быть связана с микроконтроллером PIC и источником рабочего напряжения;
- между ICD и вмонтированным микроконтроллером PIC должны быть реализованы пять знаменитых линий: VCC, GND, VPP, PGC и PGD;
- на целевой микроконтроллер должно быть подано напряжение питания; кроме того, к нему должен быть подключен работающий осциллятор;
- в целевой микроконтроллер должны быть запрограммированы корректные конфигурационные значения. В частности, в соответствии со схемой должна быть запрограммирована опция осциллятора. В дальнейшем и включение сторожевого таймера, и защита кода будут недоступны. Опция отладки должна быть включена (само собой, если в качестве отладчика выбран ICD2, это выполняется автоматически)!

9.3. Резервирование и ограничение режима отладки

- Как уже поминалось выше, выводы PGC и PGD зарезервированы исключительно для ICD. Из этого следует неизбежность использования пробника для микроконтроллеров PIC с малым числом выводов.
- "Ядром отладки" используется один или два уровня стека.
- Некоторые RAM-переменные (для PIC16F877A): 70H, (0F0H, 170H, 1F0H), 1EBH...1EFH.
- Первая ячейка памяти программ (адрес 0) должна содержать код операции 00H (команда NOP). Последние 100H программных слов отведены под "ядро отладки".
- Программирование по низкому уровню напряжения должно быть отключено.
- В случае микроконтроллеров PIC18Fxxxx стек для регистров STATUS, WREG и BSR отсутствует. Он перезаписывается по срабатыванию точки прерывания ICD.

Поскольку эти ограничения для разных типов PIC различны, всегда следует выяснять, каким образом они выглядят для конкретного используемого микроконтроллера.

9.4. Режим программатора

В противоположность представленным выше рассуждениям, теперь речь пойдет об ICD2, используемом исключительно в роли программатора. Всех описанных ранее ограничений можно придерживаться, но не обязательно. Теперь опция отладки в слове конфигурации будет автоматически отключена.

Как вы уже, наверное, поняли, микроконтроллер PIC можно программировать непосредственно в схеме с помощью ICD2 через рассмотренные выше пять линий, но при этом не обязательно использовать тактовый генератор. Микроконтроллер использует только ток, который обеспечивается схемой ICD2. Вывод VCC микроконтроллера должен быть изолирован от остальной сети VCC с помощью диода, чтобы схема ICD2 могла получать питание от PIC.

Тот, кто не хочет программировать микроконтроллер PIC во впаянном состоянии, может воспользоваться универсальным модулем программирования — достаточно только правильно подсоединить пять проводников (рис. 9.5).

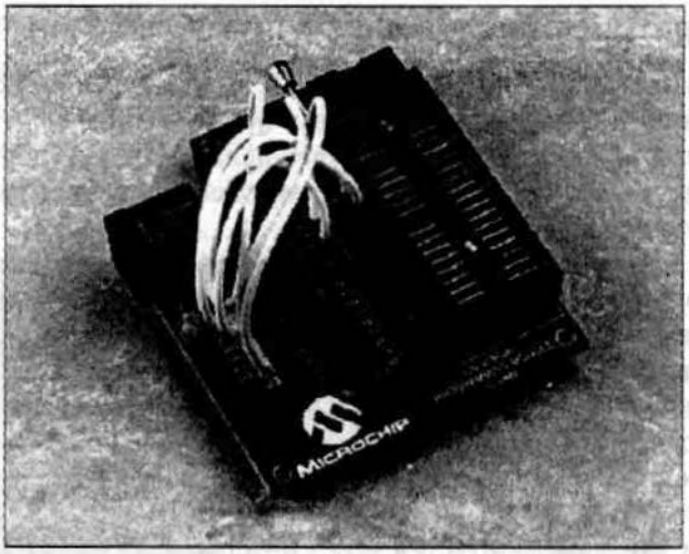


Рис. 9.5. Универсальный модуль программирования

Это средство программирования оснащено цоколем ZIP, предназначенным для программируемых микроконтроллеров PIC. Вход ICD2 — шестиконтактный разъем RJ45. Слева от него расположены семь линий с гнездами. Посередине платы проходит однорядный столбиковый разъем, связанный с цоколем ZIP один к одному. Для того чтобы теперь запрограммировать некоторый PIC, требуется создать необходимые линии. Линии VCC и GND присутствуют дважды, поскольку некоторые микроконтроллеры PIC имеют более одного вывода VCC и GND.

И не забудьте подключить ICD2 к источнику питания, иначе он ничего не сможет запрограммировать!

9.5. Управление ICD2

Для управления ICD2 как в режиме отладчика, так и в режиме программатора отведены целых два меню в среде разработки MPLAB 6.X. Подробности относительно тех или иных команд можно узнать в главе 8.

И еще одно... Если в программу необходимо внести какие-то изменения и затем по-новому выполнить ассемблирование, то, в отличие от эмулятора, новый код не оказывается автоматически там, где он должен быть для запуска обновленной программы. Программирование следует выполнить явным образом.

9.6. Ввод ICD2 в эксплуатацию

9.6.1. Программное обеспечение

Перед тем как подключить ICD2 в разъем USB, должна быть установлена среда разработки MPLAB. Причина: в одном из подкаталогов MPLAB находится корректный драйвер для USB. Например, путь может выглядеть следующим образом: `C:\Program Files\MPLAB IDE\Drivers98\ICD2_USB`.

Если при первом подключении USB-кабеля к ICD2 Windows захочет отыскать "подходящий" драйвер, то можно указать этот или другой подобный путь (точное размещение MPLAB определяют при установке среды). Если драйвер установить неправильно, то это приведет ко множеству проблем, так что будьте внимательны!

9.6.2. Аппаратное обеспечение

Тому, кто раньше имел дело с отладчиками, изложенные ниже рассуждения будут хорошо знакомы. Во избежание будущих проблем при работе с ICD2, настоятельно рекомендуем воспринять то, что будет дальше сказано, очень серьезно. Последнее звено в цепи (то есть, целевая система) обеспечивается током в последнюю очередь, и в первую очередь отключается!

Пару слов об электропитании ICD2... Несмотря на то, что, по сути, оно может быть реализовано через шину USB, мы используем отдельный источник питания. Это обосновано тем, что ICD2, оснащенный собственной системой электропитания, в состоянии обеспечить тестовую схему током силой до 200 мА. В буднях разработчиков часто бывает так, что используется тест-схема без собственного источника питания. Для того чтобы ничего не менять в среде разработки и ничего не забыть в решающий момент, источник питания постоянно находится в рабочем режиме и первым отключается вместе с ПК.

Предположим, ПК уже работает, а через интерфейс USB установлена связь. Целевая система еще не подключена. Если теперь выбрать ICD2 в качестве средства отладки, то будет выдано сообщение об ошибке, поскольку неизвестен правильный идентификатор микроконтроллера.

Для начала, простейший вариант — выбрать, к примеру, демо-плату PICDEM2 plus, и организовать соответствующие соединения. С помощью подключенного микроконтроллера PIC и правильной настройки перемычек (особенно перемычки J7, отвечающей за осциллятор), результата можно добиться очень быстро.

Для теста создадим конфигурацию с вышеупомянутой платой PICDEM2 plus. В сорокаконтактный цоколь подключен микроконтроллер PIC15F877 с опцией осциллятора XT. Однако на демо-плате кварцевый осциллятор отсутствует, а перемычка J7 замкнута, поскольку к выводу OSC1 подключена RC-схема, рассчитанная на частоту 2 МГц.

Теперь необходимо установить в MPLAB следующие разряды конфигурации:

- **Oscillator — XT;**
- **Watchdog Timer — Off;**
- **Power Up Timer — On;**
- **Brown Out Detect — Off;**
- **Low Voltage Program — Disabled;**
- **Flash Program Write — Enabled;**
- **Data EE Read Protect — Off;**
- **Code Protect — Off.**

С помощью кнопки **Program target device** (рис. 9.6) эта конфигурация вместе с памятью программ, заполненной командами NOP, программируется в микроконтроллер PIC. По завершению процесса программирования сразу же появляется сообщение об ошибке номер ICD0083: "Target not in debug mode, unable to perform operation...". Ее суть в следующем.

MPLABICD2 не может выполнить операции отладки (например, пошаговое выполнение, установка точек прерывания и т.д.). Причины этому могут быть самыми разными:

- целевой процессор не был запрограммирован с "debug executive" (то есть, MPLAB ICD2 был выбран как программатор, а не отладчик);
- целевая прикладная система не обеспечена питанием;

- в целевой прикладной системе отсутствует осциллятор;
- в течение низковольтного применение активизирован разряд BOD;
- установка разряда конфигурации для осциллятора (команда меню **Configure ▸ Configuration Bits**) некорректна для целевого осциллятора;

После корректировки опции осциллятора могут быть использованы функции отладки.

Рис. 9.6. Кнопка Program target device на панели инструментов ICD2

Другими словами, наличие напряжения питания и рабочее состояние осциллятора в целевой системе подтверждается в конце процесса программирования, и только после этого можно выполнить отладку.

ГЛАВА

10 Демо-платы и наборы разработчика

Компания Microchip предлагает целый ряд демо-плат и наборов разработчика для микроконтроллеров и периферийных устройств. Назначение этих плат — демонстрация текущего состояния системы и отображение режима работы устройств, чтобы разработчик мог сразу же применить полученные сведения на деле.

Однако демо-платы не следует выбрасывать, как только с их помощью была получена вся необходимая информация. Будучи "интеллектуальными" лабораторными инструментами, они находят самое разнообразное применение при тестировании, проведении экспериментов и организации обмена данными.

10.1. Базовые модули

Все демо-платы оснащены определенным базовыми модулями. Рассмотрим кратко каждый из этих компонентов.

10.1.1. Схема электропитания

Эта схема демо-платы (рис. 10.1) предназначена для обеспечения запитывания от блока питания. В случае плат PICDEM2, PICDEM2 plus и PICDEM4 можно также в качестве альтернативы подключить аккумуляторный блок на 9 В. Такая возможность очень удобна в портативных устройствах.

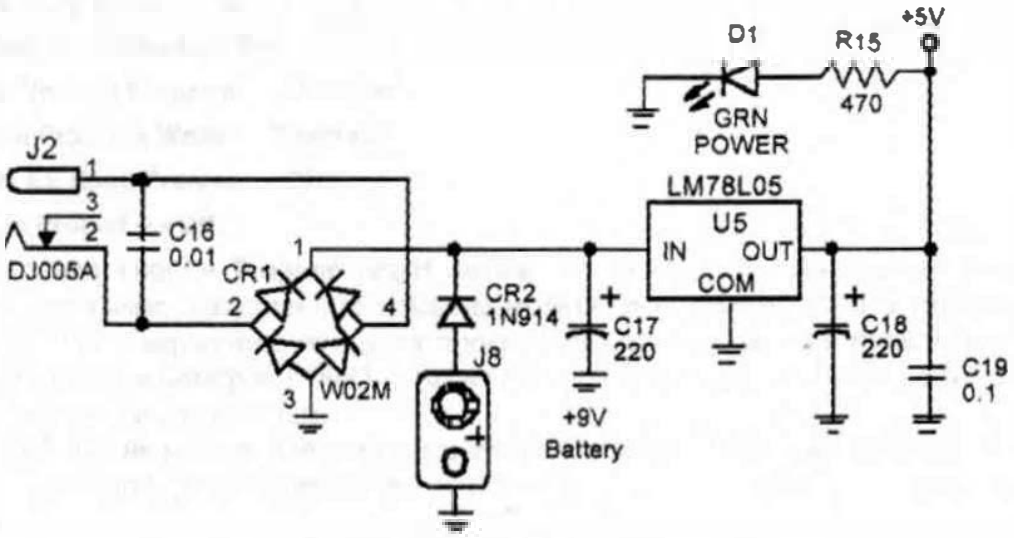


Рис. 10.1. Типичная схема питания демо-платы

10.1.2. Схема осциллятора

Базовая схема осциллятора (рис. 10.2) ограничивается подключением главного осциллятора к выводам OSC1 и OSC2, тем не менее, этого вполне достаточно. Можно реализовать как RC-осциллятор, так и XT с кварцем и HS (например, с помощью кварцевого осциллятора в 14-тиконтактом корпусе DIL). RC-звено подключается к выводу OSC1 через перемычку, а кварцевый осциллятор устанавливают в разъем. На плату впаян только кварц с его балластным конденсатором.

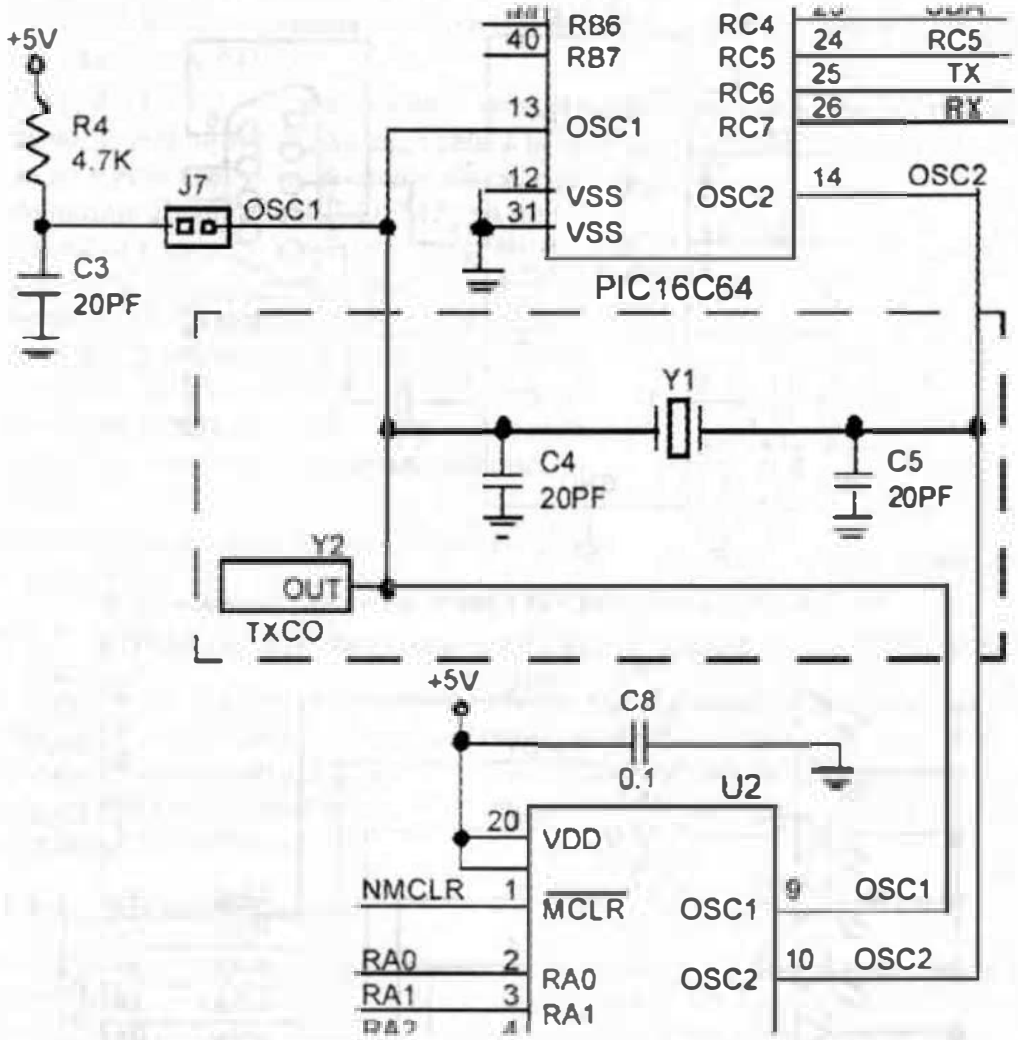


Рис. 10.2. Возможность подключения внешней схемы осциллятора

10.1.3. Схема управления V.24

Этот интерфейс относится к давним стандартам, используемым компанией Microchip в демо-платах. С помощью обычного кабеля 1:1–V.24 можно без труда организовать соединение с ПК. В качестве схемы управления можно использовать микросхему MAX232A (рис. 10.3) или LT1280A, что равнозначно.

10.1.4. Ряд светодиодов

Для быстрой индикации цифровых состояний и содержимого байтов с давних времен используется ряд светодиодов (рис. 10.4). Возможность отслеживания хода

некоторого процесса с помощью светодиодов, очень практична, и потому реализована в демо-платах. Наличие на плате PICDEM2 plus только четырех светодиодов уравновешивается присутствием ЖК-дисплея.

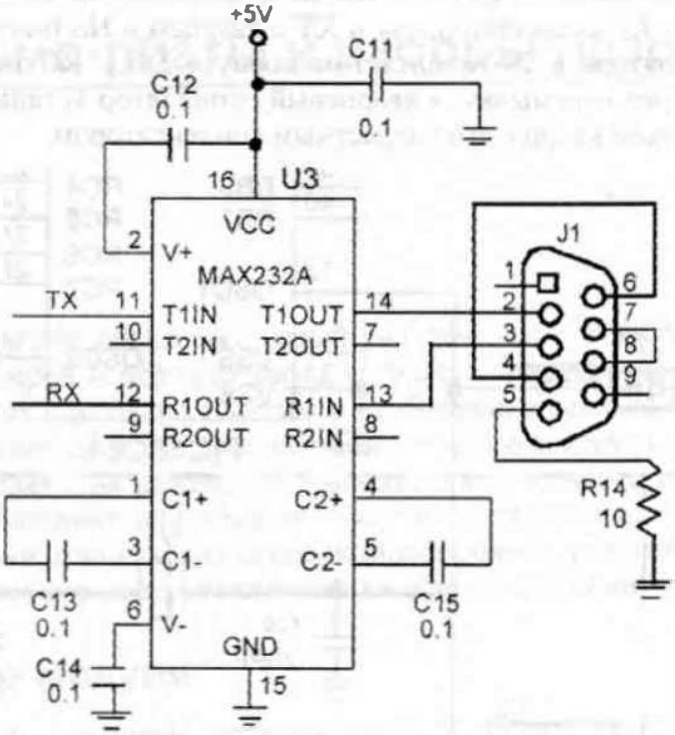


Рис. 10.3. Схема управления V.24 с девятиконтактным разъемом DSUB

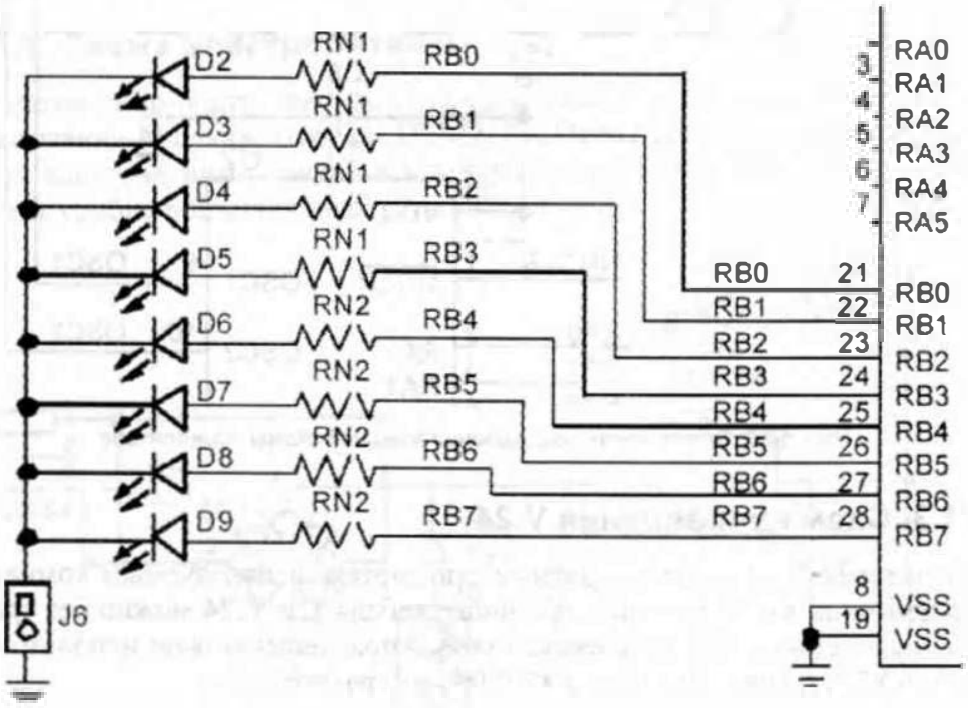


Рис. 10.4. Светодиоды с добавочными резисторами

Выводы, к которым подключаются светодиоды, не всегда одни и те же, и их число также не всегда совпадает. Когда-то (на плате PICDEM1) ряд светодиодов нельзя было отключить с помощью перемычки, но теперь это стало стандартом.

10.1.5. Кнопки

Присутствие пары кнопок на тестовой плате просто неоценимо. Всегда существуют функции, которые нужно запустить на выполнение, или команды меню, которые нужно выбрать. Для таких задач и используются кнопки. В стандартном наборе их три:

- MCLR — самой собой, эту кнопку нельзя опросить напрямую, однако ее наличие обязательно;
- RTCC, T0CKI — связана с выводом 4 порта A — вход таймера Timer0 (раньше назывался RTCC);
- RA1, RB0, RC2 — третья кнопка не всегда подключена к одному и тому же выводу. Например, в случае платы PICDEM1 это — вывод 1 порта A, и кнопка не имеет какой-либо специальной функции. Для платы PICDEM2 она подключена к выводу 2 порта C, то есть, — к выводу CCP, через который можно косвенно вызвать прерывание. В случае плат PICDEM2 plus и PICDEM4 третья кнопка связана с выводом 0 порта B. Это — вход INT, для которого можно запрограммировать фронт сигнала, вызывающий прерывание.

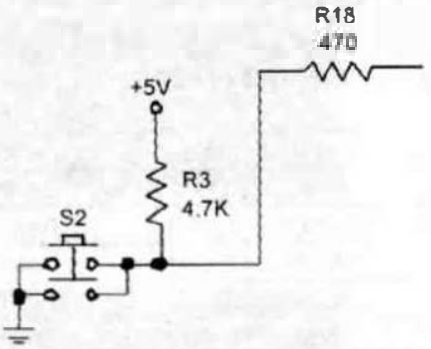


Рис. 10.5. Типичная кнопка демо-платы

Типичная схема подключения кнопки показана на рис. 10.5

10.1.6. Большая контактная матрица с выводами GND и +5V

На каждой демо-плате от компании Microchip предусмотрена матрица для подключения дополнительных элементов. Это хорошо, поскольку каждому хотя бы раз приходилось использовать особые узлы, которые, например, формируют сигналы. С помощью подобных дополнительных элементов можно создавать и специализированные опытные образцы.

10.1.7. Потенциометр

Что же за демо-плата без потенциометра? Кроме всего прочего, она должна воспринимать аналоговые сигналы, а большинство микроконтроллеров PIC оснащены встроенным АЦП. Именно для того чтобы продемонстрировать возможности АЦП, каждая демо-плата и оснащена потенциометром. На плате PICDEM4 их даже четыре. Типичное сопротивление — 10 кОм.

10.2. Обзор

Каждая разновидность демо-плат рассчитана на микроконтроллеры с определенным типом корпуса (табл. 10.1).

Таблица 10.1. Назначение различных демо-плат

Демо-плата	*Назначение*
PICDEM 14A	Для микроконтроллеров PIC14000
PICDEM 1	Для 18-, 28- и 40-выводных микроконтроллеров семейств C5X, CE62X, C71, 84 и 17C4X

Таблица 10.1. Окончание

Демо-плата	Назначение
PICDEM 2 plus	Последователь платы PICDEM 2. Предназначен для 28- и 40-выводных корпусов 16C6X, 16C7X, 16F87X, 18CXX2 и 18 FXXX
PICDEM 3	Для микроконтроллера 16C92X
PICDEM 4	Для 8-, 14- и 18-выводных микроконтроллеров 12F629/675, 16F62X, 16F81X, 18F1220 и т.д.
PICKIT 1	Для 8- и 14-выводных микроконтроллеров 12F629/675 и 16F639/676
PICDEM 17	Для микроконтроллера PIC17CXX

Другие категории демо-плат предназначены для узко специализированных применений (табл. 10.2).

Таблица 10.2. Назначение специализированных демо-плат

Демо-плата	Назначение
dPIC Development Kit	Демонстрация ВЧ-связи в диапазоне 315–433 МГц. Используются отдельные модули для приемника и для передатчика
PICDEM.net	Набор для демонстрации протокола TCP/IP
PICDEM LIN	Демо-плата для LIN-микроконтроллера 16C43X
PICDEM USB	Демо-плата для USB-микроконтроллера 16C7X5
MCP251X CAN	Плата для CAN-разработчика
MCP250XX	Плата для CAN-разработчика
PICDEM MSC	Для микроконтроллера 16C781/2. Существует несколько дополнительных плат

Кроме того, существует ряд демо-плат для микроконтроллеров dsPIC, которые в этой книге не рассматриваются. И конечно же, представленные выше перечни далеко не исчерпывающие.

10.3. Краткое описание некоторых плат

10.3.1. PICDEM1

В эту демо-плату (рис. 10.6) можно установить следующие типы микроконтроллеров PIC:

- PIC16C5X, 18 выводов;
- PIC16C5X, 28 выводов;
- PIC17C4X, 40 выводов;
- PIC16CXX, 18 выводов, с ограничениями.

Кроме вышеупомянутых стандартных периферийных элементов, PICDEM1 не предлагает никаких дополнительных свойств. Это — единственная демо-плата для микроконтроллеров 5X.

10.3.2. PICDEM2 plus

Разносторонняя демо-плата со множеством "потомков" (рис. 10.7). Поддерживает следующие микроконтроллеры:

- PIC16CXX, 28 и 40 выводов;
- PIC16FXXX, 28 и 40 выводов.

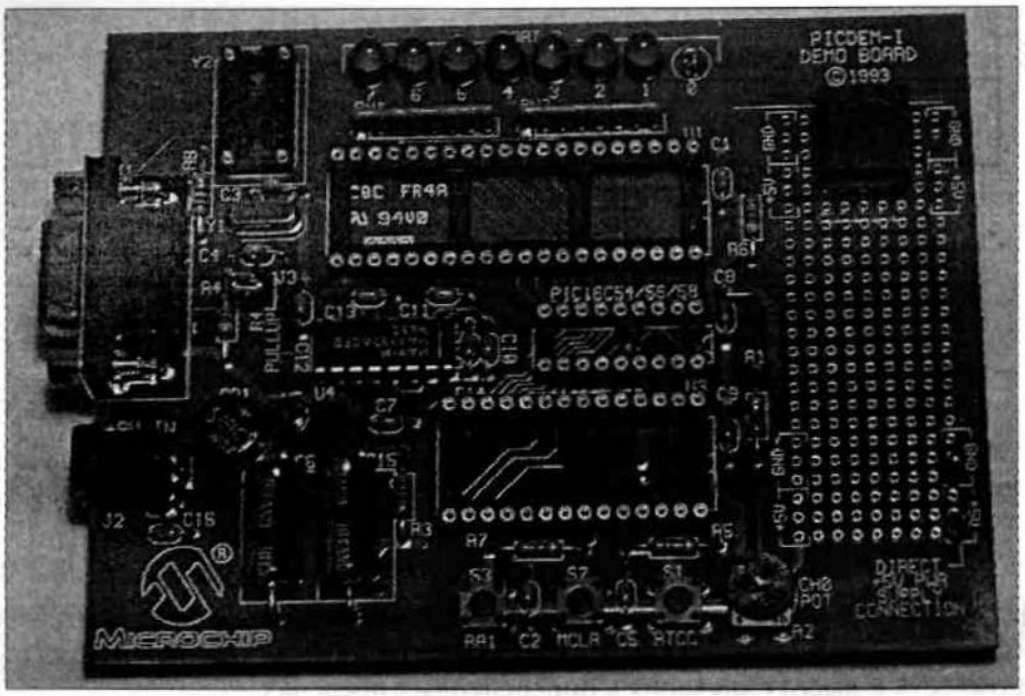


Рис. 10.6. Плата PICDEM1 для семейства 5X и 40-выводных представителей семейства 17

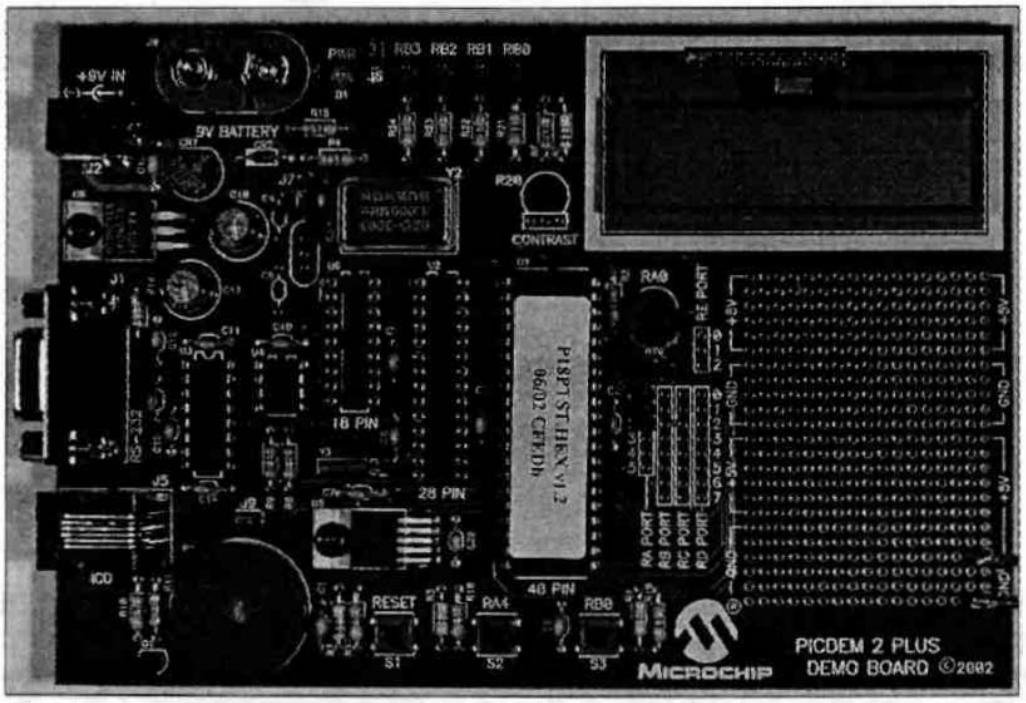


Рис. 10.7. Плата PICDEM2 plus с ЖК-дисплеем

Кроме стандартного набора модулей, на этой демо-плате дополнительно размещен осциллятор таймера Timer1, оснащенный часовым кварцем и двумя балластными конденсаторами.

Кроме того, плата PICDEM2 plus предлагает двухстрочный ЖК-дисплей по 16 символов в каждой строке, а также соответствующий интегрированный контроллер, к которому из тестового микроконтроллера PIC можно обращаться с помощью четырехразрядной шины данных и трех управляющих линий (рис. 10.8).

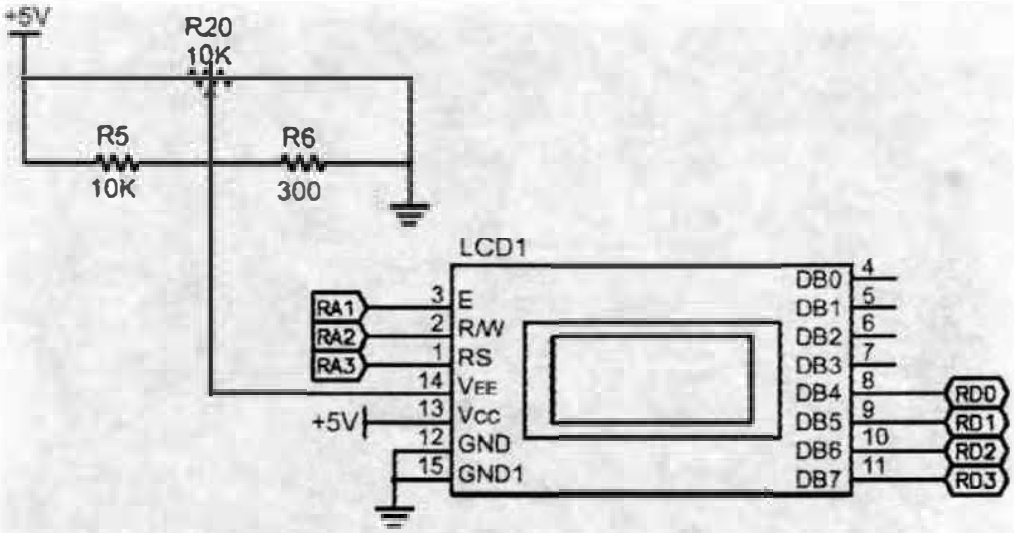


Рис. 10.8. ЖК-дисплей с контроллером и четырехразрядной шиной данных

PICDEM2 plus — единственная демо-плата, которая может выдавать звуковые сигналы с помощью зуммера (рис. 10.9). Демонстрационная программа может изменять громкость и высоту звучания в определенных пределах.

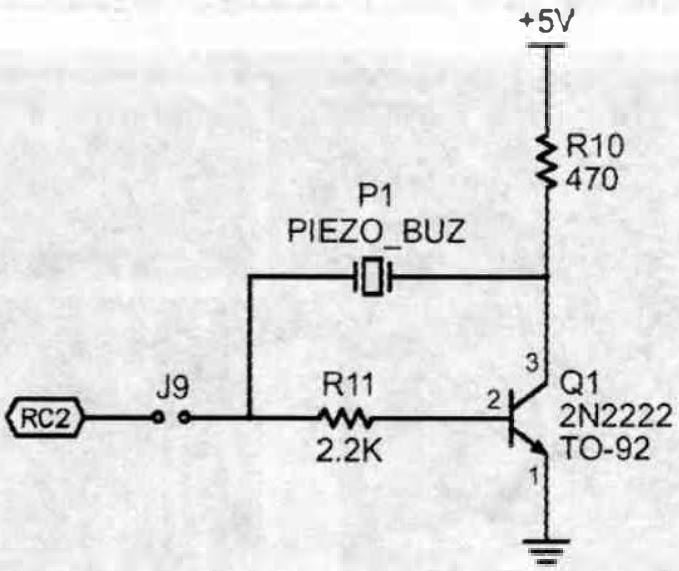


Рис. 10.9. Схема включения зуммера платы PICDEM2 plus

Кроме того, на этой плате присутствует последовательная память EEPROM размером 256 кБит с интерфейсом I^2C. Таким образом, с помощью PICDEM2 plus можно тестировать подпрограммы для работы с памятью EEPROM по интерфейсу I^2C.

Предпоследняя особенность PICDEM2 plus — датчик температуры TC74 (рис. 10.10). Вышеупомянутая демонстрационная программа, кроме всего прочего, также отображается температуру в градусах Цельсия.

Последнее полезное дополнение — разъем, к которому можно подключить модуль ICD2 для типов микроконтроллеров, поддерживающих отладку (рис. 10.11). К таким типам, которые подходят к используемому цоколю, например, относятся PIC16F876A и PIC16F877A. Достаточно только установить соответствующий микроконтроллер в цоколь и выбрать правильную схему осциллятора.

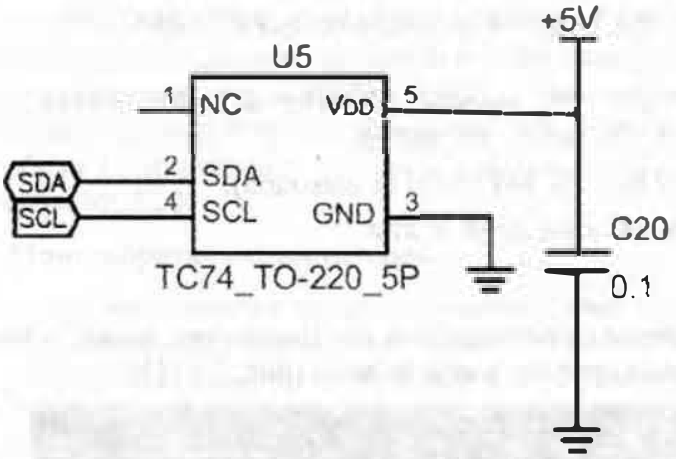


Рис. 10.10. Датчик температуры TC74

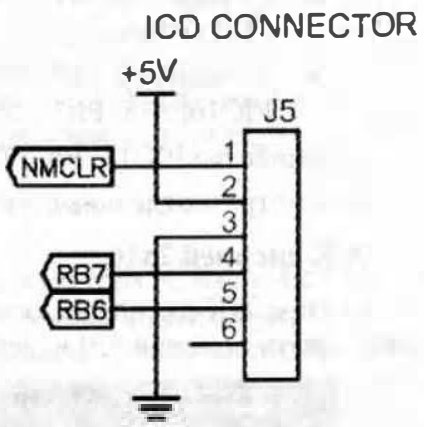


Рис. 10.11. ICD2-соединение платы PICDEM2 plus

10.3.3. PICDEM3

Эта плата предназначена специально для микроконтроллера 16C92X с интерфейсом с ЖК-дисплеем, и только для него.

10.3.4. PICDEM4

Как и PICDEM2 plus, PICDEM4 (рис. 10.12) — очень универсальная демо-плата со множеством возможностей и с поддержкой самых разнообразных микроконтроллеров PIC.

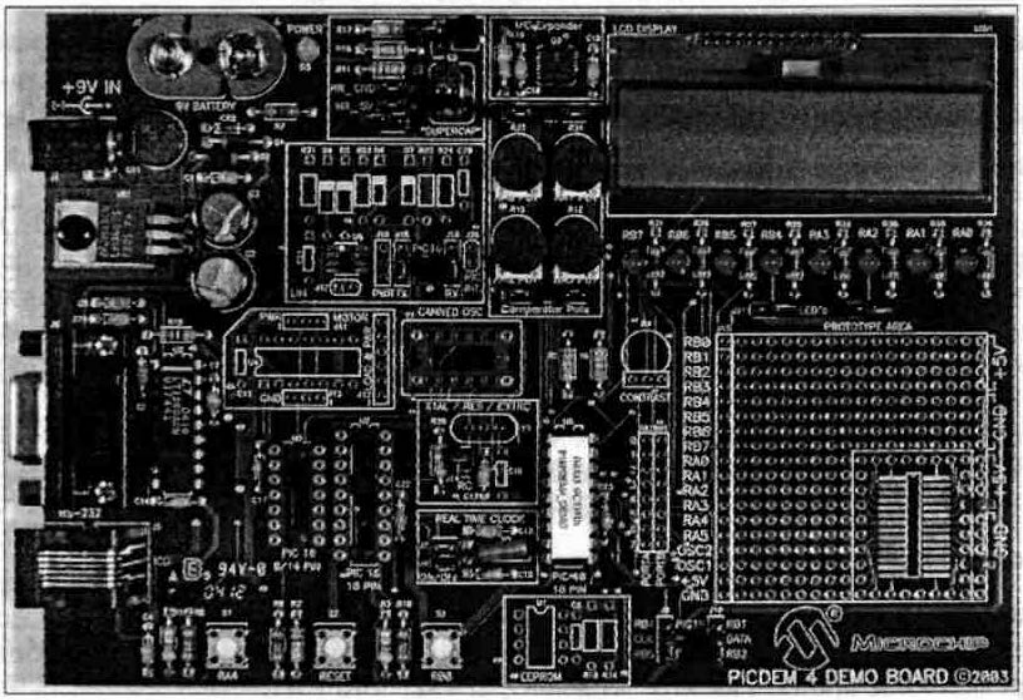


Рис. 10.12. Плата PICDEM4

Эта плата — превосходная тестовая платформа для следующих микроконтроллеров PIC:

- семейство PIC16:
  - 8 выводов — PIC12F629, PIC12F635, PIC12F675, PIC12F683;

- 14 выводов — PIC16F630, PIC16F636, PIC16F676, PIC16F684, PIC16F688;
- 18 выводов — PIC16F627A, PIC16F628A, PIC16F648A, PIC16F684, PIC16F818, PIC16F819, PIC16F87, PIC16F88;

- семейство PIC18: PIC18F1220 и PIC18F1320 (18 выводов).

Рассмотрим отдельные элементы этой демо-платы.

ЖК-дисплей 2x16

С этим ЖК-дисплеем, оснащенным встроенным контроллером, можно взаимодействовать по шине SPI через расширитель ввода-вывода (рис. 10.13).

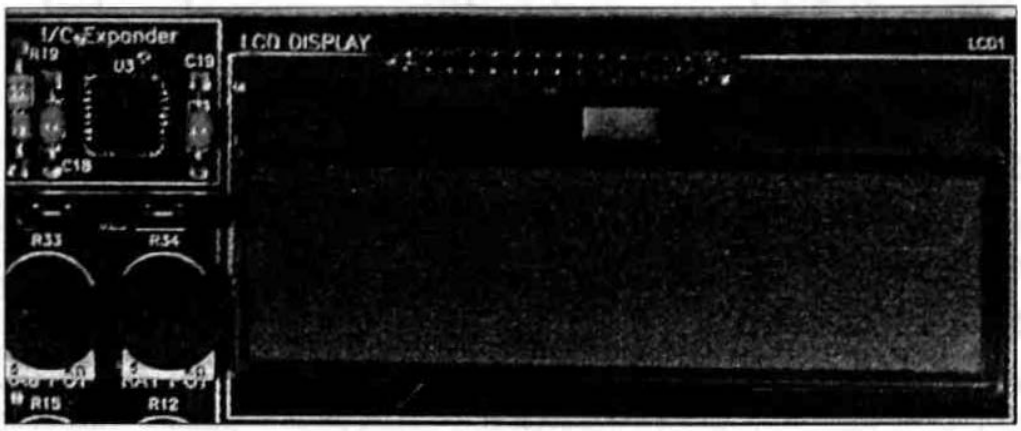


Рис. 10.13. ЖК-дисплей на расширителе ввода-вывода

Набор команд расширителя PIC16LF72 объясняется во множестве программных файлов, входящих в состав прилагаемого к демо-плате компакт-диска. К стандартным расширителям ввода-вывода относится, к примеру, микросхема MCP34016.

Внутрисхемное последовательное соединение с ICD2

В зависимости от типа микроконтроллера PIC, через соединение с ICD2 может осуществлять программирование или отладка. Как уже было отмечено в предыдущей главе, отладку через ICD2 поддерживают не все типы PIC. В вышеупомянутом перечне микроконтроллеров, поддерживаемых демо-платой PICDEM4, подобная отладка возможна только для семейства PIC18, а также для моделей PIC16F81X и PIC16F8X.

Поскольку все целевые микроконтроллеры PIC устанавливаются в цоколь, можно без проблем использовать соответствующий пробник. В таком случае ICD2 должен быть подключен к этому пробнику.

Восемь светодиодов, четыре потенциометра и три кнопки

На первый взгляд, эти простые элементы ввода-вывода не производят особого впечатления, однако для того, чтобы реализовать их самостоятельно, потребовалось бы определенное время. Так что, "спасибо, что вы есть у нас".

Плата поддерживает память EEPROM, H-мост (схема управления двигателем) и приемопередатчик LIN

Еще один элемент, который нередко требуется включать в состав схемы, — внешняя память EEPROM с последовательным доступом. Для такого случая на пла-

те PICDEM4 зарезервирован участок, который можно укомплектовать микросхемой EEPROM, конденсатором для организации электропитания и двумя подтягивающими резисторами для линий SDA и SCL. С помощью перемычек устанавливается соединение с микроконтроллером одного из двух типов: PIC16XXX или PIC18XXX. Подобная комплектация используется для схемы управления двигателем и LIN-драйвера MCP201.

Многообразие осцилляторов

Стандартом считается подключение главного осциллятора к выводам OSC1 и OSC2. Вместо него, разумеется, можно выбрать часовой кварц для таймера Timer1, который удобно включать/отключать с помощью перемычки. Этот кварц уже оснащен балластным конденсатором, однако не такой надежный как "большой" кварц. Как показывает опыт, при использовании с микроконтроллерами PIC он иногда работает некорректно, поэтому перед тем как заказать партию таких элементов, рекомендуем проверить работу этого кварца на одном опытном образце.

Гибкая система питания

В случае платы PICDEM4 система электропитания выходит за пределы стандартных возможностей. Благодаря технологии "Gold-Cap" становится доступно выполнение опытов в реалистичной обстановке:

- с внешним источником питания или без него;
- уменьшение потребляемой мощности во время использования "Gold-Cap".

Можно сразу же, без особых затрат протестировать переключение в режим управления питанием.

Расширение возможностей с помощью контактной матрицы

Контактная матрица присутствует также и на плате PICDEM4, однако имеет одну особенность: наличие площадок для корпусов SO, которые всегда доставляют разработчикам немало хлопот. Впрочем, SO — не такой уж и плохой вариант, и сопротивление 1208 — тоже не трагедия. Хорошо же то, что теперь мы можем работать, по сути дела, только с пинцетом в руке. И это не мало!

Как видим, плата PICDEM4 — очень удачная. Кто хочет сэкономить деньги и время, тот обязательно должен обзавестись PICDEM2 plus и PICDEM4.

10.3.5. PICKIT1

PICKIT1 (рис. 10.14) — это первая демо-плата с возможностью USB-подключения к ПК. Она предоставляет множество возможностей и поставляется с учебными материалами по самым разным вопросам. Рассмотрим области применения PICKIT1.

Демо-плата для небольших микроконтроллеров PIC

Эта очень маленькая плата содержит только одну кнопку и один потенциометр, однако — до двенадцати светодиодов. Она не укомплектована полем расширения с драйвером V.24 и контактной матрицей, однако этот вопрос быстро решается.

Хотя PICKIT1 и не содержит много "железа", вместе с этой платой поставляется несколько учебных пособий, в которых изложены основные методики программирования (например, учет дребезга контакта кнопки и уменьшение числа выводов при подключении нескольких светодиодов).

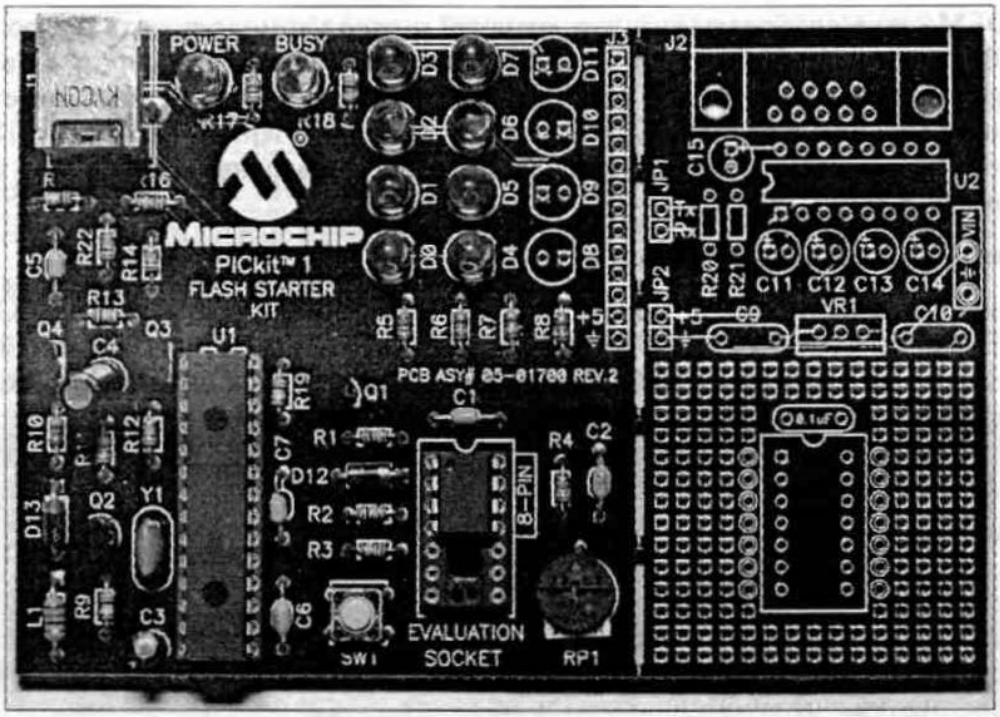


Рис. 10.14. Плата PICKIT1 с интерфейсом USB

Демонстрация работы с интерфейсом USB

Третья цель набора PICKIT1 — демонстрация взаимодействия с ПК через интерфейс USB. Для этого поставляются все исходные коды программ с пояснениями.

Микроконтроллеры PIC можно программировать с помощью специального графического интерфейса (рис. 10.15), который также используется для восстановления калибровочных значений.

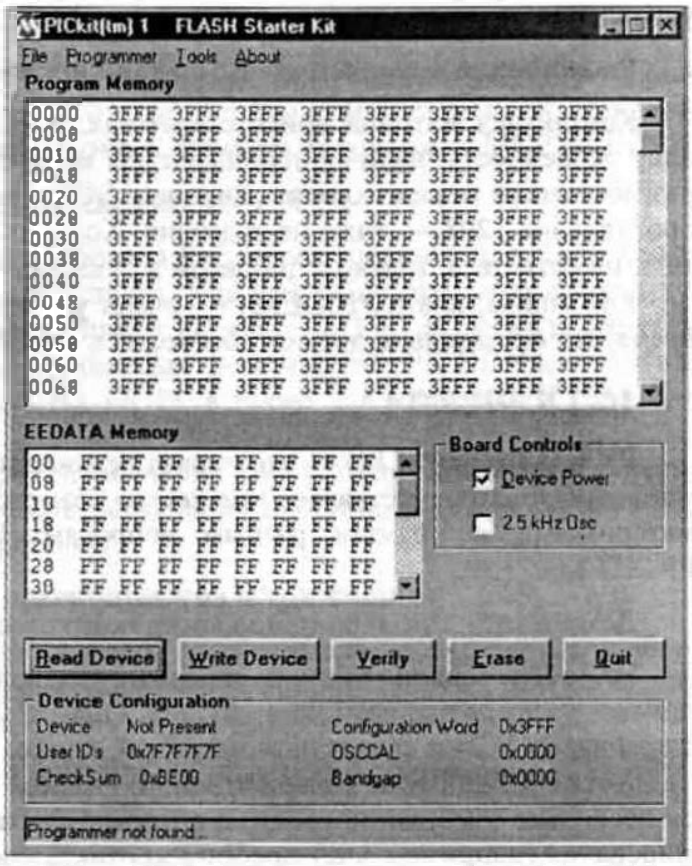


Рис. 10.15. Графический интерфейс для работы с PICKIT1

10.3.6. PICDEM MSC

Эта демо-плата (рис. 10.16) была разработана специально для микроконтроллера 16C781/782, поэтому разработчик должен быть знаком с этим типом устройств. И опять таки, в распоряжение пользователя предоставляется графический интерфейс (рис. 10.17), с помощью которого можно конфигурировать отдельные модули.

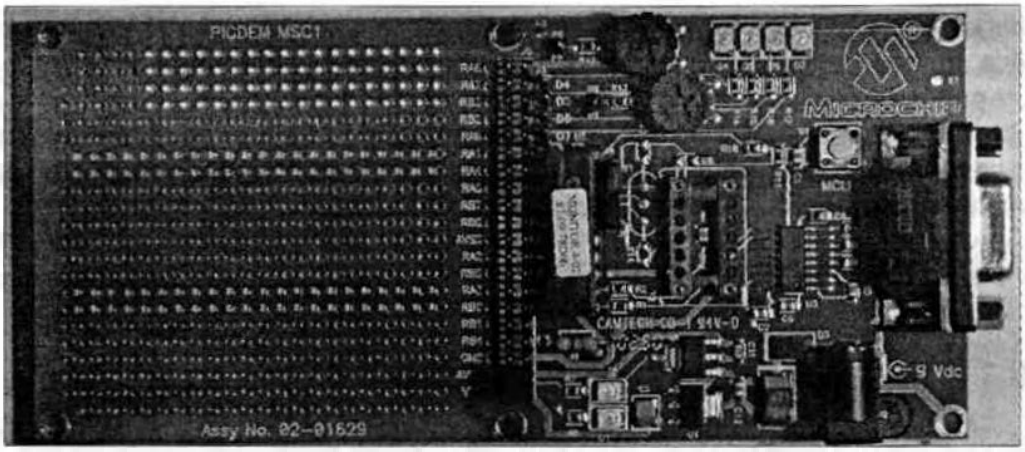


Рис. 10.16. Демо-плата PICDEM MSC

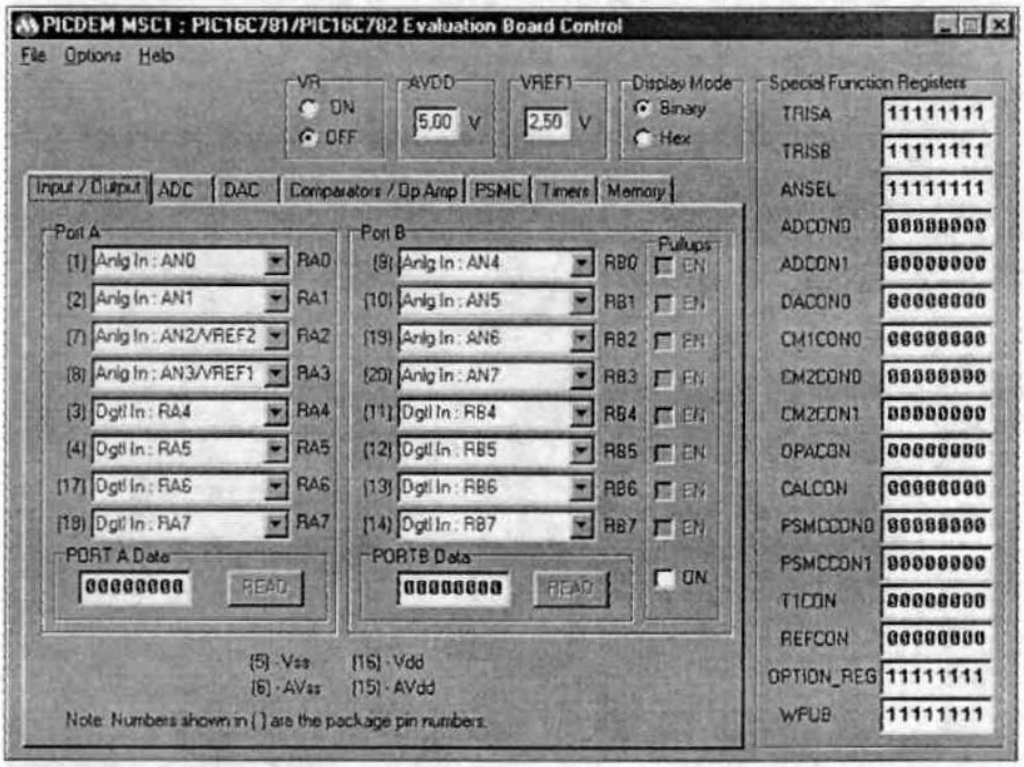


Рис. 10.17. Графический интерфейс набора PIC-MSC

В этой плате присутствуют следующие дополнительные модули:

- модуль таймера Timer0;
- модуль таймера Timer1;
- модуль опорного напряжения;
- программируемый модуль обнаружения низкого напряжения (PLVD);
- АЦП (8 разрядов);
- ЦАП (8 разрядов);
- операционный усилитель;
- модуль компаратора;
- программируемый контроллер режима переключения (PSMC).

Для каждого из этих модулей существует отдельная вкладка графического интерфейса, на которой с помощью регистров управления можно определить все свойства (см рис. 10.17).

В случае модуля АЦП существует даже возможность задать пользовательскую форму кривой. Контроллер PSMC служит для построения импульсных источников питания, работающих под управлением микроконтроллера PIC.

Таким образом, плата PICDEM MSC — завершенная платформа для PIC16C781/782, которые как нельзя лучше подходят для решения задач измерения, управления и регулирования.

10.3.7. PICDEM CAN

На момент издания книги, для шины CAN существовало две демо-платы: MCP2510/2515 и MCP250XX. Плата MCP2510 CAN Development Board (рис. 10.18) предоставляет в распоряжение разработчика два узла CAN.

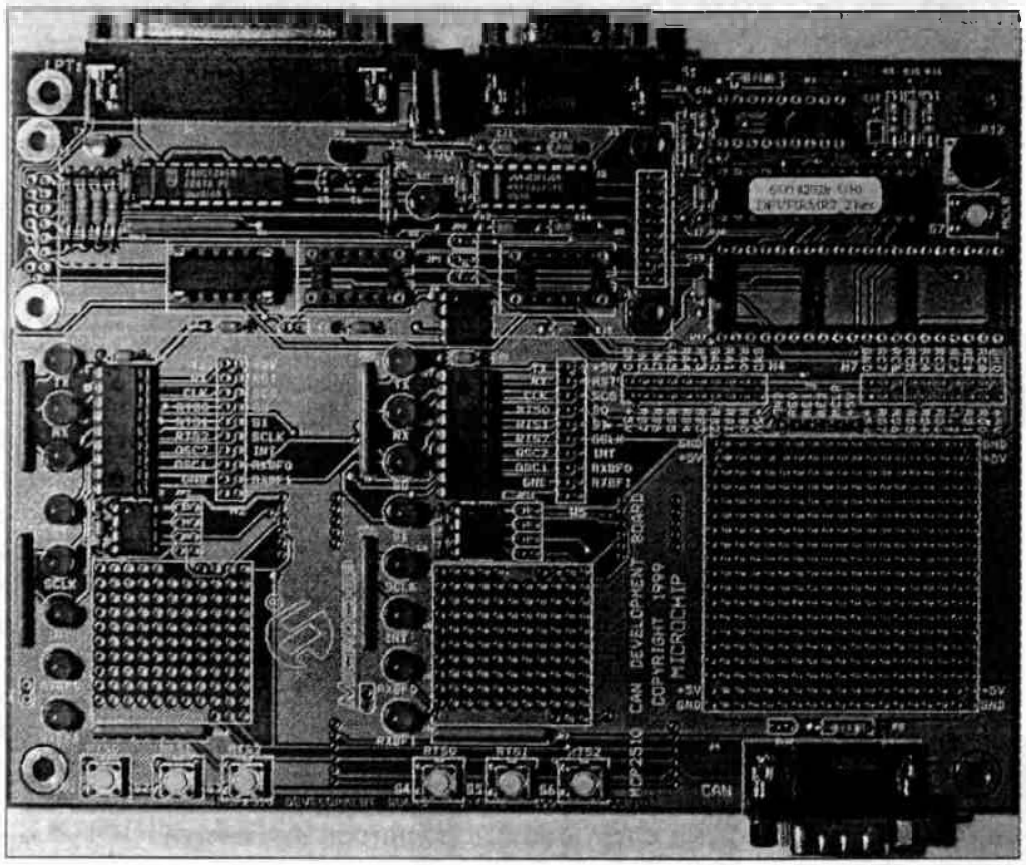


Рис. 10.18. Демо-плата MCP2510

Узел, расположенный слева, напрямую связан с ПК через параллельный интерфейс. Для его обслуживания устанавливается Windows-программа, которая предоставляет все возможности программирования MCP2510/2515. Присутствуют специальные средства для разработки фильтров и масок.

Узел, расположенный справа, взаимосвязан с предварительно запрограммированным микроконтроллером PIC и поддерживает аналоговые значения. Для работы с ПК-узлом также предусмотрены три кнопки и три светодиода. Через шину CAN от ПК можно даже установить параметры времени для микроконтроллера PIC.

Шина CAN подключается с помощью девятиконтактного разъема DSUB. Так, самостоятельно разработанный узел CAN подключается к узлу ПК и с его помощью тестируется. В этом плане демо-плата MCP2510 — просто неоценимое подспорье.

Вторая демо-плата: MCP250XX CAN Development Board (рис. 10.19) — предоставляет в распоряжение три узла CAN и один разъем для программирования.

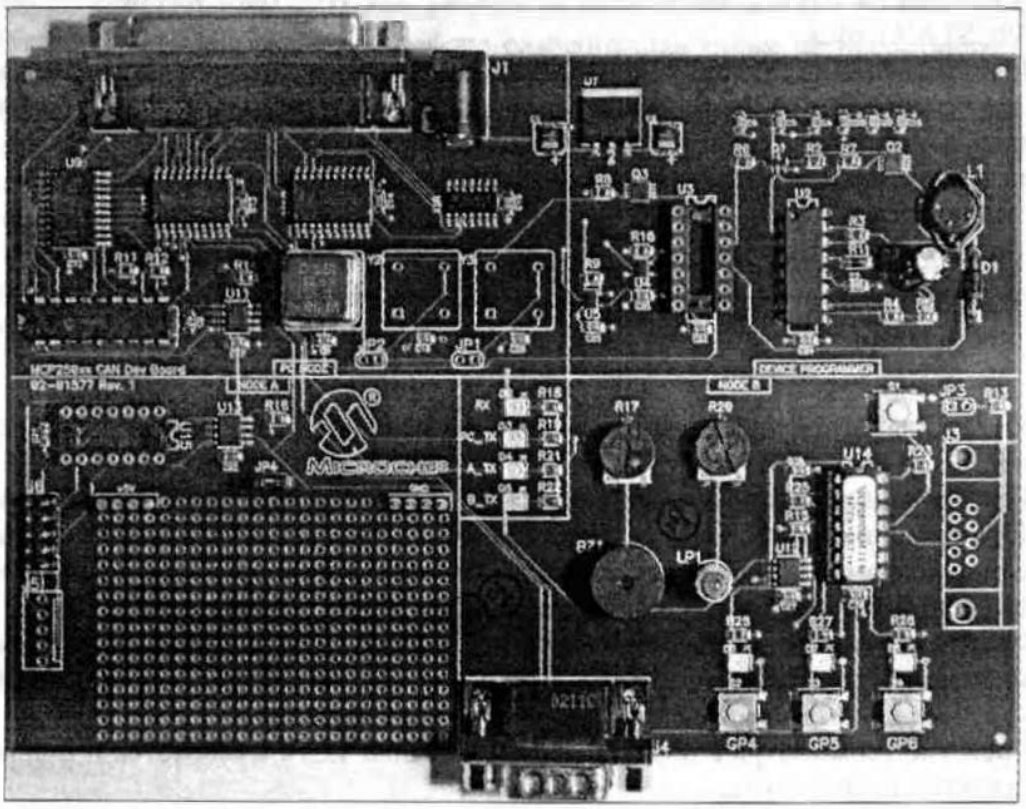


Рис. 10.19. Демо-плата MCP250XX

Один узел соединен с ПК по параллельному интерфейсу, а остальные два — с автономным CAN-контроллером MCP250XX. Правый из этих двух узлов для демонстрации возможностей оборудован кнопкой, светодиодами, потенциометрами и зуммером. Узел, расположенный слева, оснащен контактной матрицей для реализации расширений, и тем самым предоставляет разработчику максимум свободы для воплощения его идей.

Так же как и MCP2510, плата MCP250XX оборудована девятиконтактным разъемом DSUB, к которому можно без проблем подключать собственные схемы.

Справа, возле ПК-узла, находится разъем программирования. За формирование требуемого напряжения программирования отвечает расположенный тут же импульсный стабилизатор.

С помощью этого разъема демо-плата готовится к фактическому применению. При этом необходимо знать, как часто и какая информация с какими идентификаторами кадра должна передаваться по шине. Кроме того, для корректного приема информации должны быть заранее запрограммированы фильтры и маски входных каналов.

10.4. Практическое применение

Для объяснения преимуществ этих наборов, рассмотрим два примера.

10.4.1. PICDEM2 plus

С помощью этой платы мы уже провели множество тестов. Как и все другие демо-платы, PICDEM2 plus содержит завершенную систему электропитания со всеми соответствующими компонентами. Для работы нам требуется только аккумуляторная батарея на 9 В или питание от источника 9 В. Наш блок питания — такой же как в PICSTART plus.

Присутствуют полнофункциональные схемы контроллеров, которые можно сразу же перевести в рабочий режим с помощью поставляемых микроконтроллеров. В данном конкретном случае используются два устройства: PIC16 и PIC18. Программа в этих заранее запрограммированных микроконтроллерах состоит из четырех компонентов:

- вольтметр;
- зуммер;
- термометр;
- часы.

Обслуживание осуществляется с помощью кнопок, подключенных к выводам RA4 и RB0.

В режиме вольтметра постоянно отображается напряжение на потенциометре R16. Это значение колеблется в диапазоне 0...5 В. Вывод 0 порта A, связанный с потенциометром, может запросто использоваться для какой-либо другой цели, и при измерении принимать другое значение напряжения в диапазоне 0...5 В.

В режиме зуммера на пьезозуммер выдаются импульсы определенной частоты, сформированной с помощью широтно-импульсной модуляции. Частота и длительность пауз между импульсами изменяется с помощью кнопок.

В режиме термометра с помощью термодатчика TC74 измеряется и циклически отображается температура.

В режиме часов реализован отсчет реального времени на основании таймера Timer1 с кварцем 32 кГц. Установка часов осуществляется с помощью кнопок.

Как и многие другие демо-платы, PICDEM2 plus оснащен драйвером V.24: микросхемой MAX232.

Данные, которые в перечисленных выше режимах выдаются на ЖК-дисплей, одновременно подаются через V.24 на девятиконтактный штекерный разъем. С помощью терминальной программы соответствующие меню можно также без проблем отобразить и на экране компьютера.

Само собой разумеется, PICDEM2 plus позволяет тестировать и программы собственной разработки. Для этого необходим только программатор ICD2, подключенный к демо-плате через соответствующий разъем. С его помощью микроконтроллер можно очистить, запрограммировать и отладить. Все это, конечно же, удобнее выполнить в программной среде разработки MPLAB. Вам остается позаботиться только о напряжении питания и правильном подключении тактового генератора.

10.4.2. MCP251X CAN Development Kit

С помощью этой платы мы можем без проблем протестировать CAN-устройства собственной разработки. Но "без проблем" не значит "без труда"! Впрочем, это относится не столько к самой демо-плате или CAN-модулю, сколько к самой технологии CAN. Вместе с платой поставляются хорошие программные примеры, поработав над которыми вы увидите, что работать с CAN не так уж и сложно, как кажется вначале.

Когда речь идет о фильтрах и масках, то новичку не мудрено запутаться. Установку временных параметров мы также рассматривать не будем — об этом достаточно сказано в прилагаемых к плате программах.

Как же узнать, что происходит в шине CAN? Очень просто. В отдельном окне поставляемой с MCP251X программы показано, какие кадры передаются через шину, и какие данные в них содержатся.

Аппаратная установка этого набора разработчика также не представляет собой никаких проблем — требуется только параллельный порт.

После установки программного обеспечения придется поработать с литературой, что, впрочем, не займет много времени. При этом в вашем распоряжении не только руководства, имеющие отношение к самой демо-плате, но и ряд инструкций к применению от компании Microchip (например, AN733).

Ознакомления с вышеупомянутой литературой никак не избежать, поскольку, как и в случае любой другой стандартной шины, необходимо ответить на три основных вопроса:

- Что я должен?
- Что я могу?
- Что мне доступно?

Если в распоряжении имеется структура команд (кадров), то ответить на эти вопросы намного проще. Если же требуется использовать "открытую" архитектуру CAN, то все значительно усложняется. В таком случае вопросы несколько изменяются:

- Какие команды мне требуются?
- Какими командами я могу воспользоваться, а о каких могу забыть?
- Какие команды я должен поддерживать, даже если совсем не понимаю их смысла?

ГЛАВА

11 Периферийные модули

Компания Microchip издавна выпускает периферийные модули для микроконтроллеров PIC, наподобие АЦП с прямым управлением ЖК-дисплеем. Хотя представленные далее разделы и не претендуют на исчерпывающую полноту, они, тем не менее, дают хорошее представление о производимых Microchip компонентах.

11.1. Интерфейсные преобразователи

В этом разделе рассматриваются микросхемы, имеющие отношение к таким развитым шинам как CAN, IRDA, LIN и I^2C.

11.1.1. CAN

Для шины CAN компания Microchip предоставляет три различные группы устройств (табл. 11.1):

- интерфейсные преобразователи с интерфейсом SPI для подключения к микроконтроллерам PIC;
- активные расширители ввода-вывода с АЦП или без него;
- приемопередатчики CAN для сопряжения шины и преобразования уровня сигнала.

Таблица 11.1. Три группы устройств Microchip для шины CAN

Интерфейсные преобразователи	*Расширители ввода-вывода*	*Приемопередатчики*
MCP2510	MCP25020	MCP2551
MCP2515	MCP25025	
	MCP25050	
	MCP25055	

В новых разработка микросхема MCP2510 к использованию не рекомендуется. Ее преемником является микросхема MCP2515 (рис. 11.1).

Краткое описание микросхемы MCP2515

Характеристики MCP2515:

- активный CAN-контроллер в трех различных вариантах корпуса: PDIP (18 выводов), SOIC (18 выводов) и TSSOP (20 выводов);
- высокоскоростной интерфейс SPI (10 МГЦ) и SPI-режимы "0,0" и "1,1" для обмена данными с микроконтроллером;
- CAN-спецификация V2.0B со скоростью передачи 1 Мб/с;
- длина поля данных: от 0 до 8;

- стандартные и расширенные кадры;
- два буфера приема с шестью 29-тиразрядными фильтрами и двумя 29-тиразрядными масками;
- три буфера передачи;
- выход для сигнала индикации о переполнении буфера;
- входы RTS для передачи CAN-сообщений с помощью внешних триггерных сигналов.

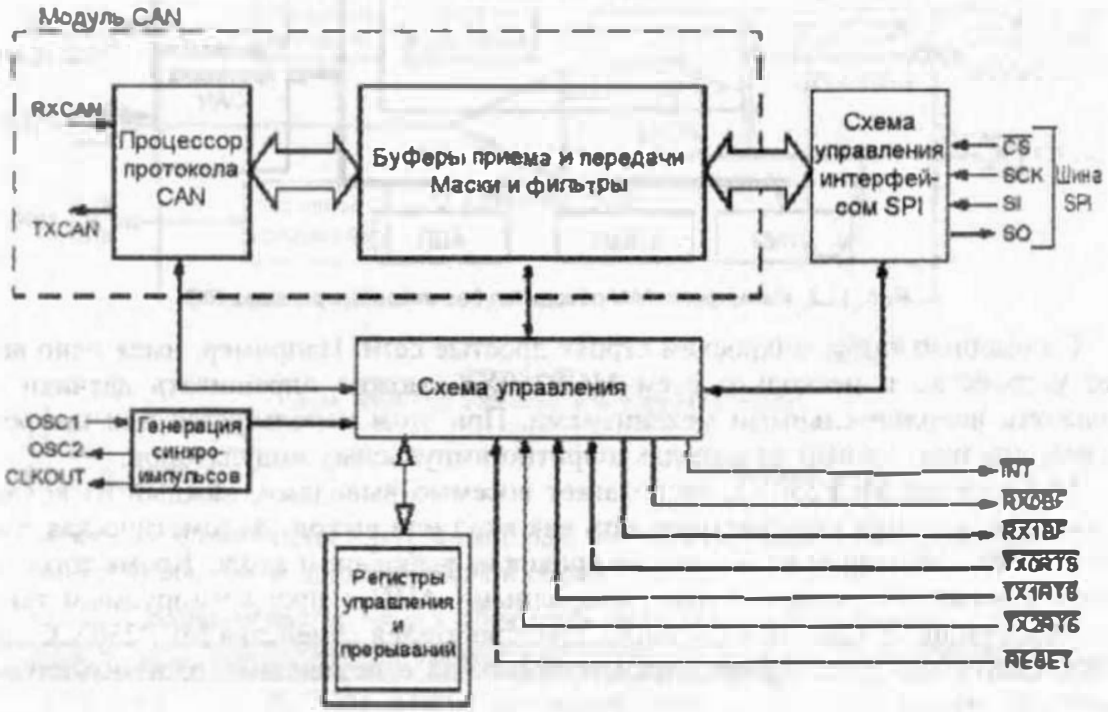


Рис. 11.1. Завершенный CAN-контроллер с шиной SPI

Справа на рис. 11.1 показаны сигнальные и управляющие линии интерфейса SPI, которые являются обязательными и обслуживаются микроконтроллером. Линии, расположенные на рисунке внизу справа, используются на усмотрение разработчика. Так, линия /INT предназначена для передачи уведомлений, а по линиям /RX0BF и /RX1BF подается информация для буферов приема. Если на одной из них установлен низкий уровень сигнала, то это означает, что в соответствующий буфер только что было записано сообщение. По линиям управления /TXxRTS выдаются сигналы буфера передачи. На линию /RESET подается сигнал сброса, формируемый RC-звеном при включении устройства.

Краткое описание микросхем семейства MCP250XX

Перечислим основные особенности MCP250XX (рис. 11.2):

- спецификация CAN V2.0B;
- программируемая маска;
- два программируемых фильтра;
- три автоматических буфера передачи;
- два буфера приема данных.

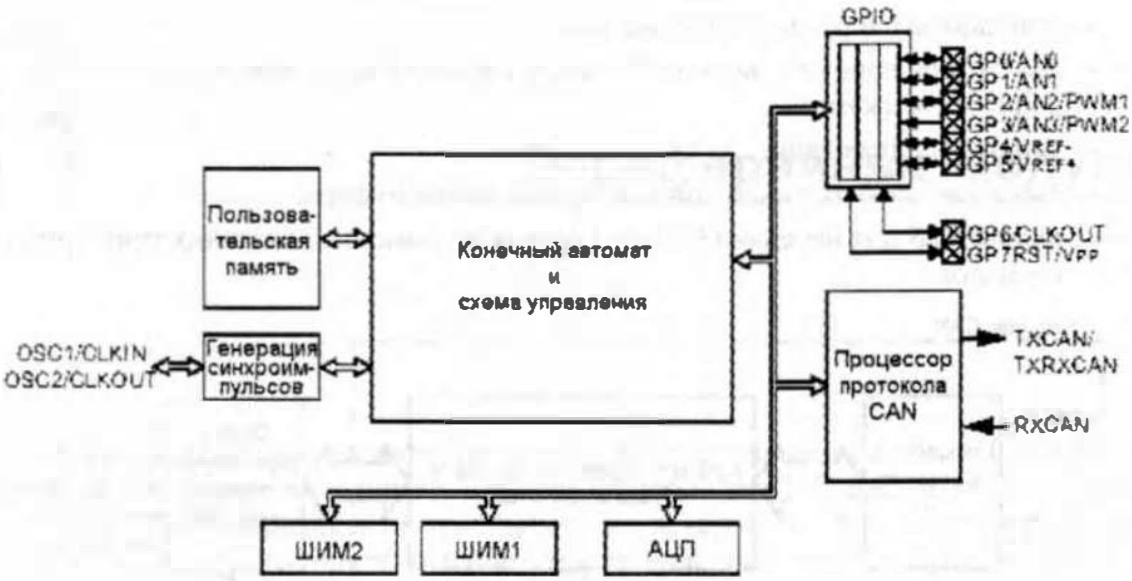


Рис. 11.2. Интерфейс CAN обходится без микроконтроллера PIC

С помощью таких микросхем строят простые сети. Например, имея одно ведущее устройство и несколько схем MCP250XX, можно опрашивать датчики или управлять исполнительными механизмами. При этом используется или цифровой сигнал, или полученный на выходах широтно-импульсных модуляторов.

Микросхема MCP250XX располагает восемью выводами, каждый из которых можно при желании сконфигурировать как вход или выход. Автоматическая функция передачи реагирует на изменение уровня на выбранном входе. Кроме того, микросхема оснащена четырьмя десятиразрядными АЦП с программируемым тактом преобразования. В каждом из четырех представителей семейства MCP250XX также присутствует два десятиразрядных ШИМ-выхода с независимо программируемой частотой сигнала.

Стандартная память конфигурации программируется внутрисхемно в последовательном режиме, однако конфигурацию можно изменять по шине CAN. Пользовательская конфигурация может храниться в ПЗУ и загружаться автоматически при включении.

Как схема MCP2502X без АЦП, так и схема MCP2505X с АЦП существуют в корпусах типа PDIP (14 выводов) и SOIC (14 выводов).

Для этого семейства устройств компания Microchip разработала специальный набор разработчика: DV 250501.

Краткое описание микросхемы MCP2551

Преобразователь уровня MCP2551 (рис. 11.3) выполняет роль моста между сигналами на шине и TTL-совместимыми сигналами интерфейса для микросхемы MCP2515. Это — высокоскоростной приемопередатчик, характеризующийся максимальной скоростью, доступной для шины CAN: 1 Мбит/с. Через вход RS можно управлять крутизной фронта.

Микросхема MCP2551 исполняется в восьмивыводном корпусе типа PDIP или SOIC-SMD.

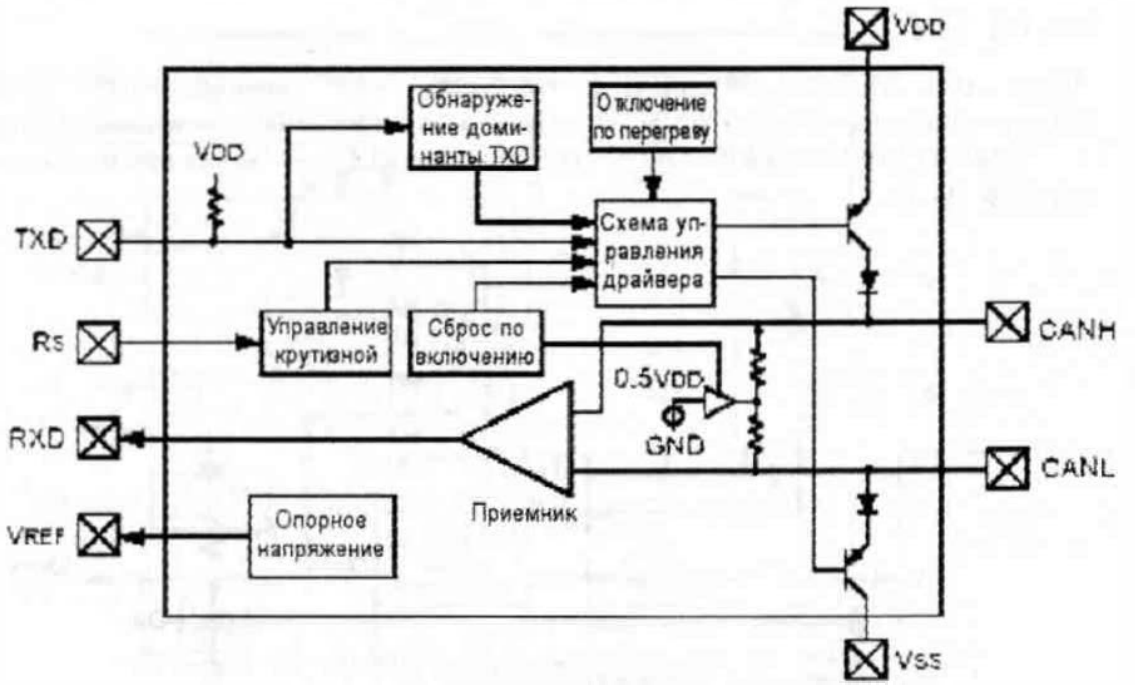


Рис. 11.3. Преобразователь уровня MCP2551

11.1.2. IRDA

Спектр изделий Microchip для работы с инфракрасными сигналами (табл. 11.2), охватывает шифраторы/дешифраторы и обработчики протоколов.

Таблица 11.2. Изделия Microchip для обработки инфракрасных сигналов

Устройство	*Описание*
MCP2140	9600 бод, шифратор и дешифратор
MCP2120	325 Кбод, шифратор и дешифратор
MCP2150	115,2 Кбод, DTE, шифратор, дешифратор и обработчик протокола
MCP2155	115,2 Кбод, DCE, шифратор, дешифратор и обработчик протокола

Недорогая микросхема MCP2140 работает с жестко заданной скоростью передачи 9600 бод и выпускается в корпусе PDIP (14 или 18 выводов), SOIC (18 выводов) или SSOP (20 выводов).

Микросхема MCP2120 или MCP2140 — это исключительно шифратор или дешифратор. Она преобразовывает только последовательный байт по стандарту V.24, передаваемый в виде инфракрасного сигнала от модуля UART или обратно. Тем не менее, когда упоминается аббревиатура IRDA, обычно подразумевают стандарт соединения, обрабатываемый по специальному протоколу. Более того, микросхемы MCP2150 и MCP2155 поддерживают не один, а сразу несколько таких протоколов! К ним, в частности, относятся Tiny TP и IRComm.

В случае применения устройств MCP2120/40 требуемый протокол должен быть реализован программно. Функции шифрации и дешифрации, характерные для микросхем MCP2120/40, присутствуют также и в "старших" устройствах: MCP2150/55.

Если вместо приемопередатчика MCP2551 использовать микросхему MCP2120, то шина CAN может быть реализована с помощью инфракрасной среды передачи.

11.1.3. LIN

Микросхема MCP201 (рис. 11.4) — это "всего лишь" преобразователь уровня с интегрированным стабилизатором напряжения для обеспечения питания подключенного микроконтроллера PIC. Обработка протокола LIN — задача исключительно микроконтроллера.

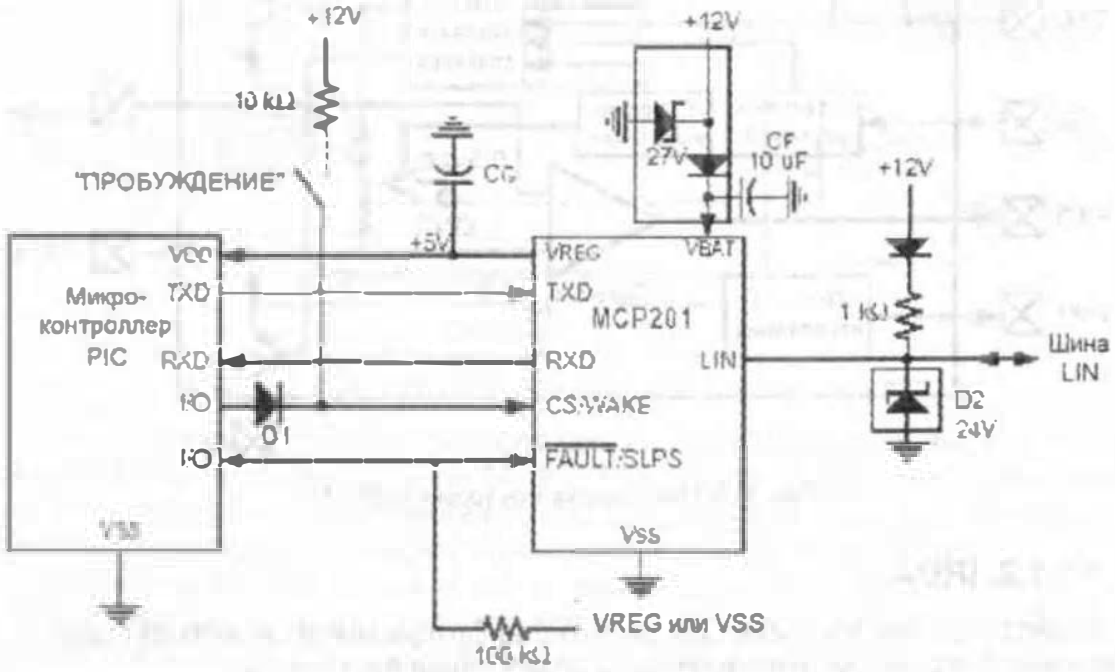


Рис. 11.4. Типичное включение преобразователя уровня LIN MCP201 (ведущее устройство)

Приемопередатчик LIN MCP201 рассчитан только на скорость 20 Кбод. Он используется в промышленности и автомобилестроении, где решающую роль играют не скорость и надежность, характерные для шины CAN, а стоимость.

Компания Microchip выпускает демо-платы также и для шины LIN. Благодаря трем таким платам обеспечивается оптимальная поддержка разработчиков при сопряжении интерфейсов CAN и LIN. В комплект поставки также входят несколько программных решений. Реализация протокола LIN может поддерживаться различными аппаратными модулями. Например, возможен очень элегантный вариант с применением модуля EUSART.

Более подробную информацию о протоколе LIN можно найти в разделе 2.5.

11.1.4. Расширитель ввода-вывода I^2C

С помощью микросхемы MCP23016 можно получить на основе шины I^2C до шестнадцати линий ввода-вывода при силе управляющего тока 25 мА (рис. 11.5). Корпус на 28 выводов существует в вариантах исполнения PDIP, SOIC, SSOP и QFN.

Эта очень полезная микросхема содержит регистр прерывания, с помощью которого сохраняется состояние восьми выводов соответствующего порта в момент изменения этого состояния. Отдельные входы могут быть индивидуально инвертированы. С помощью регистра I/ODIR0/1 все шестнадцать выводов могут одновременно и независимо друг от друга быть определены как входы или выходы.

В табл. 11.3 представлены отдельные команды, а также указано к каким из четырех внутренних регистров эти команды обращаются.

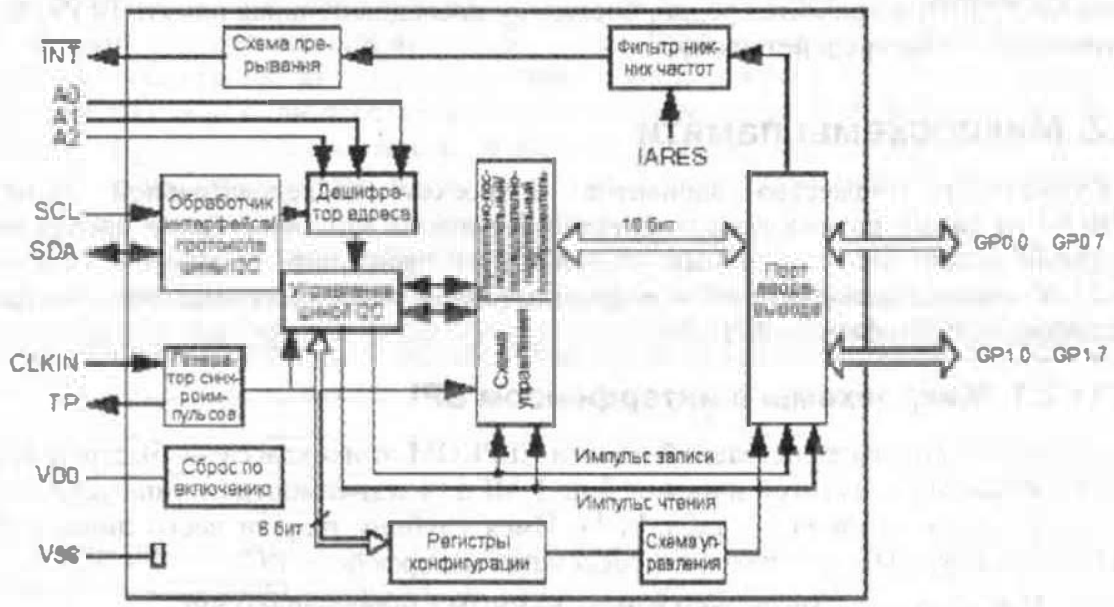


Рис. 11.5. Шестнадцать линий ввода-вывода подключены через шину I^2C

Таблица 11.3. Команды расширителя MCP23016

Байт команды	*Обращение*	*Комментарий*
0	GP0	Прямое чтение порта. Доступ к этому регистру на запись изменяет содержимое регистра OLAT0/1
1	GP1	
2	OLAT0	Чтение/запись фиксатора вывода
3	OLAT1	
4	IPOL0	Чтение/запись регистра полярности входа
5	IPOL1	
6	IODIR0	Чтение/запись регистра направления передачи через вывод
7	IODIR1	
8	INTCAP0	Чтение регистра прерывания
9	INTCAP1	
0Ah	IOCON0	Чтение/запись регистра разрешения прерывания. Этот регистр только один!
0Bh	IOCON0	

С помощью последнего регистра выбирают разрешение, соответствующее частоте возникновения прерывания. Разряд 0 этого регистра содержит 0, что означает минимальное разрешение в 32 мс. Если требуется получить более точную регистрацию изменений, то разряд 0 можно установить в 1, что даст разрешение 200 мкс (однако это, конечно же, приведет к повышению силы тока, потребляемого микросхемой).

Если разряд 0 регистра IOCON0 равен 1, то на выводе TP (Testpoint) замеряют частоту следования тактов, получаемых с помощью RC-звена на выводе CLK.

Как следует из названия микросхемы MCP23016, для доступа к этому семейству устройств используют шину I^2C. Обе линии шины, как правило, реализованы с открытым коллектором и подтягивающим резистором. Согласно протоколу, приемник по завершению приема байта выдает бит подтверждения. В том случае, когда на обработку принятого байта ему требуется некоторое время, тактовая линия может быть заблокирована (поддерживается сигнал низкого уровня) до тех пор, пока не будет опять готова к приему следующего байта. Впрочем, такой вариант для микро-

схемы MCP23016 встречается редко, поскольку последовательная память EEPROM, как правило, — быстродействующая.

11.2. Микросхемы памяти

Существует множество вариантов микросхем последовательной памяти EEPROM от самых разных производителей. Компания Microchip также предлагает обширный ассортимент подобных устройств: от небольших модулей в корпусе SOT23 до микросхем значительной информационной емкости, оснащенных быстродействующим интерфейсом SPI.

11.2.1. Микросхемы с интерфейсом SPI

Микросхемы последовательной памяти EEPROM этого класса — быстродействующие, с тактовой частотой шины от 2 до 3 МГц (в зависимости от типа устройства и напряжения питания — табл. 11.4). Имея глубину памяти всего лишь 1–16 Кбит, они значительно уступают микросхемам с интерфейсом I²C.

Таблица 11.4. Параметры микросхем памяти EEPROM с интерфейсом SPI

Тип	*Диапазон напряжений*	*Максимальная тактовая частота*
C	4,5–5,5 В	2–3 МГц
LC	2,5–5,5 В	2–3 МГц
AA	1,8–5,5 В	2–3 МГц

11.2.2. Микросхемы с интерфейсом I²C

Маркировка ряда таких устройств (рис. 11.6) начинается с "24C16". Обозначение "24C", как правило, соответствует структуре EE на 5 В, а "16" — объеме памяти 16 Кбит.

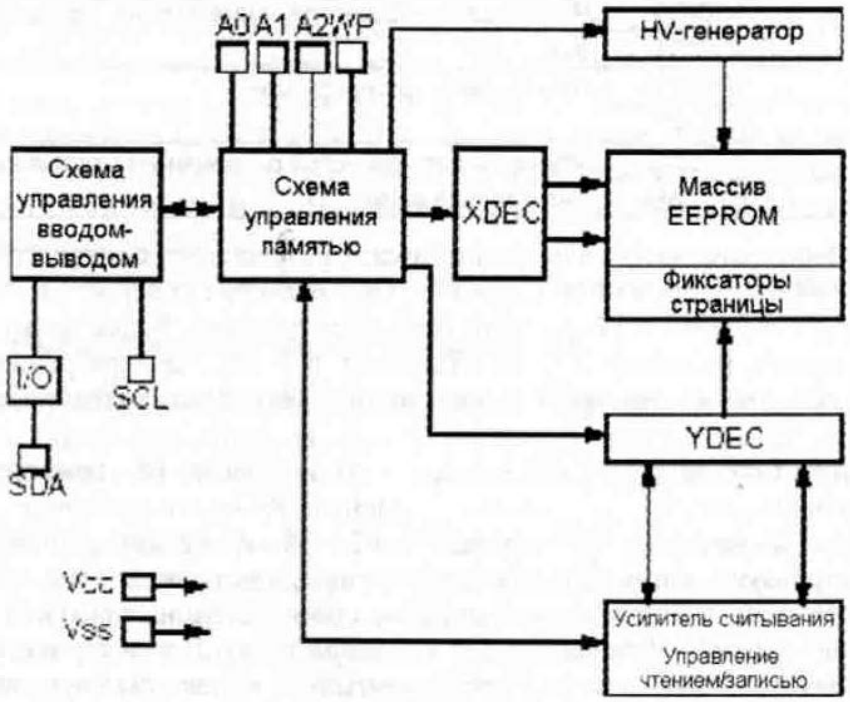


Рис. 11.6. Внутренняя структура микросхемы памяти EEPROM с интерфейсом I²C

Если маркировка начинается с "24LC", то это означает, что микросхема функционирует при напряжении вплоть до 2,5 В. Этому нижнему порогу соответствует тактовая частота 100 кГц. При напряжении 4,5 В память типа "24LC" работает на частоте 400 кГц. Если требуется частота повыше, то следует выбрать тип памяти "24FC". При этом, опять таки, если выбрать нижний предел рабочего напряжения (4,5 В), то частота составит все те же 400 кГц. Устройства типа "AA" предназначены специально для малых напряжений питания.

Перечисленные выше данные сведены в табл. 11.5.

Таблица 11.5. Тактовые частоты микросхем памяти EEPROM с интерфейсом I²C

Тип	1,8–2,5 В	2,5–4,5 В	4,5–5,5 В
C	–	–	400 кГц
LC	–	400 кГц	400 кГц
FC	–	400 кГц	1 МГц
AA	100 кГц	400 кГц	400 кГц

Информационный объем микросхем памяти EEPROM, выпущенных компанией Microchip, составляет от 16 байт до 64 Кбайт. Кроме того, они могут быть выполнены в различных корпусах (рис. 11.7).

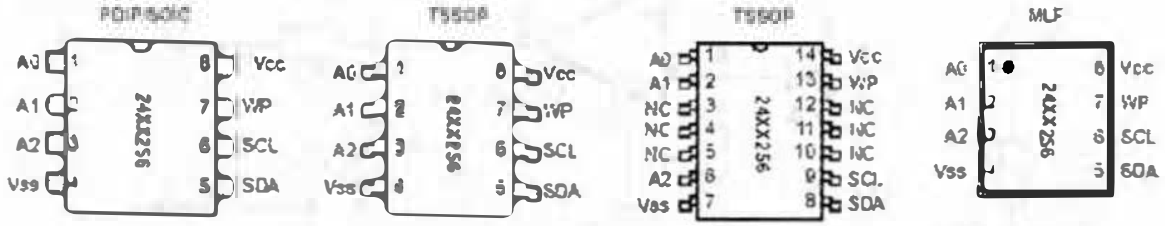


Рис. 11.7. Различные типы корпусов для микросхем EEPROM с интерфейсом I²C

Однако существуют корпуса еще меньше. Шестнадцатиразрядная память 24LC00 может быть выполнена даже в корпусе SOT (рис. 11.8).

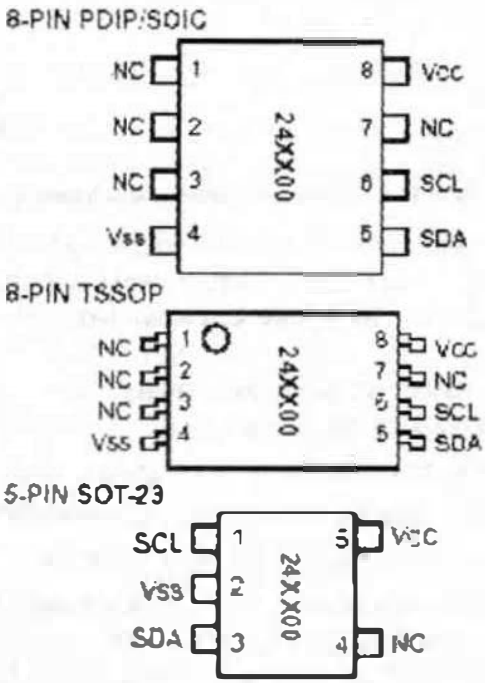


Рис. 11.8. Миникорпуса микросхемы 24LC00

11.3. Операционные усилители и компараторы

В этой области существует такое разнообразие представителей, что мы ограничимся только рассмотрением нескольких новых типов устройств с малым энергопотреблением.

11.3.1. MCP654X

Семейство компараторов MCP654X обладает выдающимися характеристиками. Как видно из представленного ниже примера, наилучшая из таких характеристик — минимальное потребление тока. Благодаря скромному рабочему напряжению от 1,6 до 5,5 В, эти устройства как нельзя лучше подходят для работы от аккумуляторных батарей.

На рис. 11.9 представлен пример схемы, в которой для увеличения срока службы батареи светодиод включается только с наступлением темноты.

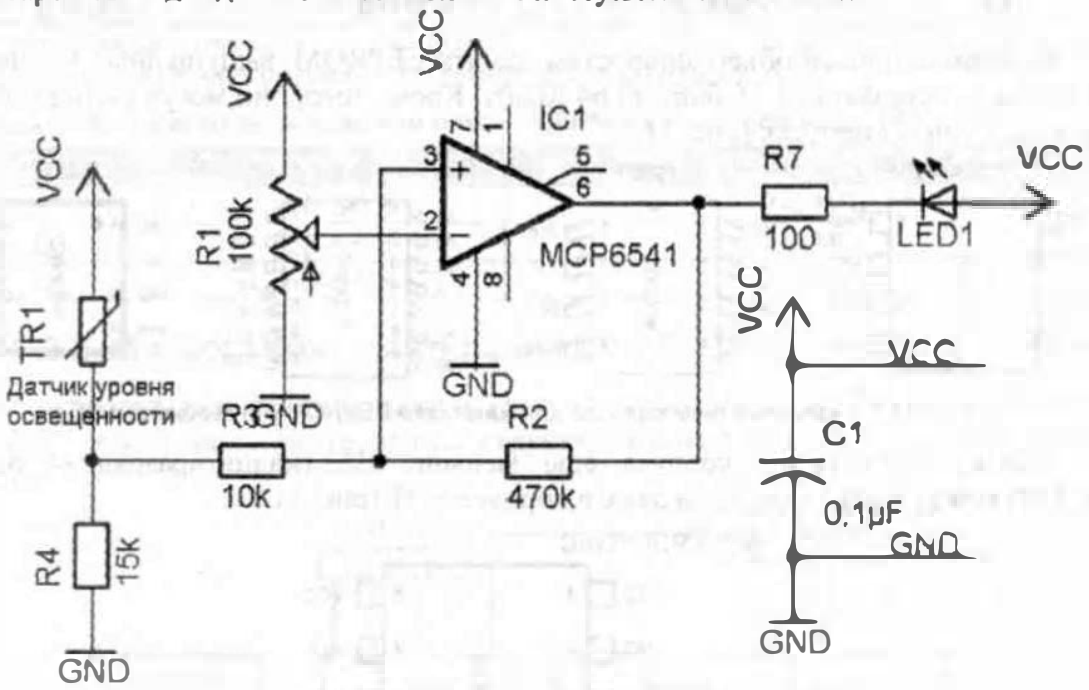


Рис. 11.9. Схема автоматической "свечки"

Сила тока через светодиод составляет 14 мА, но для остальной схемы — всего лишь 131 мкА. Схема потребляет лишь малую часть электроэнергии, потребляемой светодиодом, и таким образом на сроке службы батареи в основном сказывается только темное время суток.

Семейство MCP654X имеет четыре представителя:

- MCP6541 — один компаратор в корпусе;
- MCP6542 — два компаратора в одном корпусе;
- MCP6543 — один компаратор в корпусе с возможностью выбора кристалла;
- MCP6544 — четыре компаратора в одном корпусе.

Многообразие корпусов превосходит все ожидания. Здесь есть все, начиная корпусов PDIP, удобных для быстрого построения опытных образцов, и обычных вариантов SOIC, MSOP и TSSOP, до миниатюрных SOT-23-5 и SC-70.

Например, более микроскопический корпус, чем SC-70 (рис. 11.10), размеры которого составляют каких-то 1x2x2 мм, и придумать трудно.

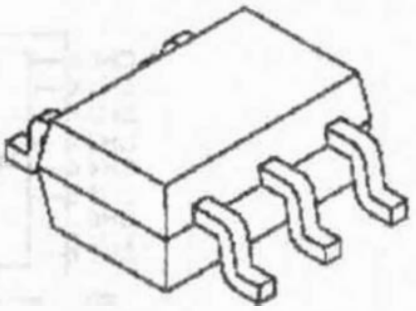


Рис. 11.10. Таким большим корпус SC-70 выглядит, разве что, под лупой

11.3.2. MCP604X

Семейство операционных усилителей (ОУ) MCP604X также имеет четыре представителя:

- MCP6041 — один ОУ в корпусе;
- MCP6042 — два ОУ в одном корпусе;
- MCP6043 — один ОУ в корпусе с возможностью выбора кристалла;
- MCP6044 — четыре ОУ в одном корпусе.

Имея типичное потребление тока всего лишь 0,6 мкА и диапазон рабочих напряжений 1,4–5,5 В (один источник питания), эти микросхемы также прекрасно подходят для работы от аккумуляторной батареи. Пример применения этого операционного усилителя представлен на рис. 11.11.

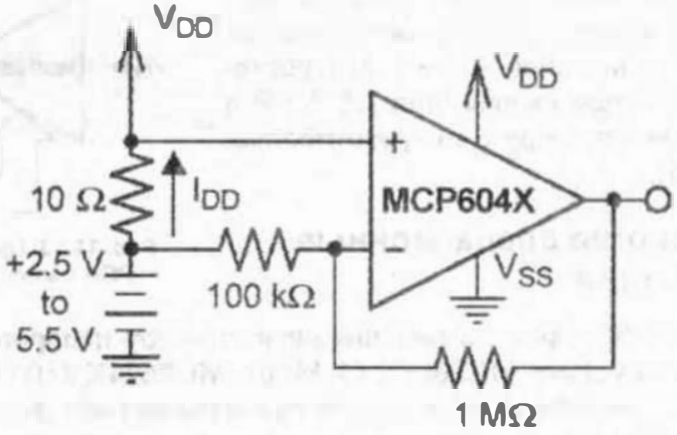


Рис. 11.11. Пример применения ОУ MCP604X

Здесь неинвертирующий вход ОУ соединен с уровнем положительного напряжения питания. Такая схема применяется для опроса тока в цепи положительного напряжения. Критерием для протекающего тока является падение напряжения на резисторе сопротивлением 10 Ом. Все потребители подключены между VDD и VSS.

11.3.3. MCP6S2X

Микросхемы MCP6S21, MCP6S22, MCP6S26 и MCP6S28 — это так называемые усилители с программируемым усилением (PGA — Programmable Gain Amplifier), число входных каналов которых составляет от 1 до 8 (рис. 11.12):

- MCP6S21 — один канал;
- MCP6S22 — два канала;
- MCP6S26 — шесть каналов;
- MCP6S28 — восемь каналов.

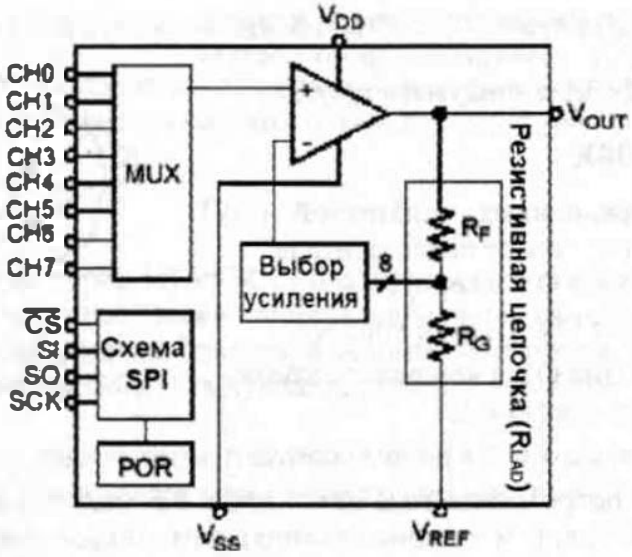


Рис. 11.12. PGA с восемью входными каналами

Входы и выходы имеют размах сигнала, равный напряжению питания. Полоса частот составляет 2–12 МГц, а ток питания — 1 мА. Микросхема работает на питающем напряжении 2,5–5,5 В и составляет очень гибкую пару с микроконтроллером PIC (рис. 11.13).

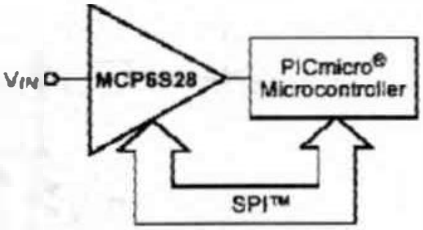


Рис. 11.13. Пример использования PGA с микроконтроллером PIC

11.3.4. Недорогие операционные усилители

Компания Microchip представляет широкую палитру недорогих операционных усилителей в лице семейств MCP600X (1 МГц), MCP624X (500 кГц) и MCP623X (200 кГц). Напряжение питания этих устройств составляет от 1,8 или 2,7 до 5,5 В.

Примеры схем и расчетные формулы можно найти в специализированной литературе, а также в фирменных изданиях компании Microchip. Кроме того, на Web-сайте Microchip есть технические паспорта, советы по применению и так называемые "technical briefs". В этих источниках рассматриваются различные технические нюансы для разнообразных сфер применения — причем касательно не только операционных усилителей, но и других цифровых и аналоговых устройств.

Дополнительно, компания Microchip предлагает программу FilterLab, которая позволяет очень быстро создавать типы фильтров. Этот инструмент также можно бесплатно загрузить с Web-сайта `www.microchip.com`.

11.3.5. Линейные компоновочные блоки

Эти модули обрадуют тех разработчиков, которые используют несколько различных линейных компонентов, однако имеют в своем распоряжении не много свободного пространства. В данном случае в одном корпусе объединены операционный усилитель, компаратор и источник опорного напряжения, которые, по существу, также выполнены в собственных корпусах. Пример типичного обозначения подобных микросхем — TC1026C (существуют в корпусах трех различных типов).

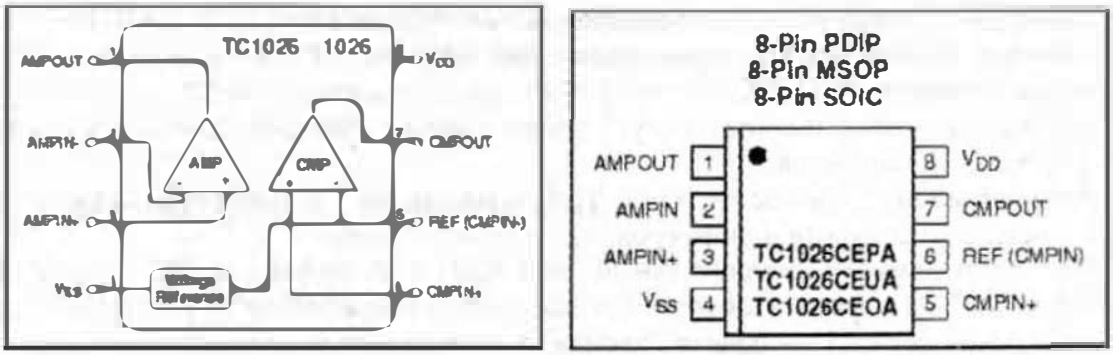


Рис. 11.14. Небольшой линейный блок

Микросхема TC1043C, содержащая два операционных усилителя, два компаратора и источник опорного напряжения, существует только в корпусе одного типа: QSOP (16 выводов) (рис. 11.15).

Оба вышеупомянутых устройства могут работать от источника питания в диапазоне напряжений 1,8–5,5 В. При небольшом потреблении тока в 12/16 мкА они также могут питаться от аккумуляторных батарей.

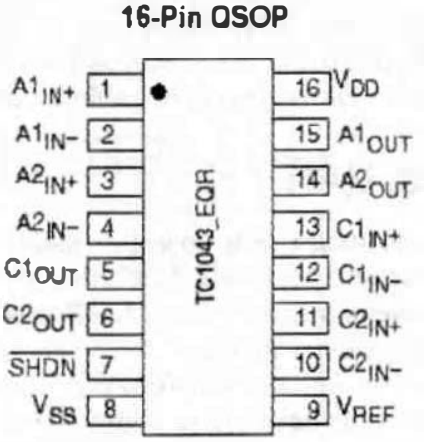


Рис. 11.15. Линейный блок большего размера

11.4. АЦП

Благодаря большому выбору аналого-цифровых преобразователей, в цифровых мультиметрах и регистраторах данных можно применять устройства с самыми разнообразными характеристиками, включая интерфейс с ЖК-дисплеем (табл. 11.6).

Таблица 11.6. Характеристики различных АЦП

Тип АЦП	*Максимальная частота выборки*	*Разрешение*
SAR (последовательное приближение)	22–200 тыс. в секунду	10–13 разрядов (в зависимости от типа)
Сигма-дельта	> 400 тыс. в секунду	10–16 разрядов
С двойным интегрированием	2–10 в секунду	12–17 разрядов
Двоичные и двоично-десятичные	До 40 в секунду	До 4,5 цифр

Очень медленнодействующий АЦП с двойным интегрированием дает разрешение до 17 разрядов. Такие микросхемы снабжены разнообразными интерфейсами, наподобие трехпроводного, последовательного и/или параллельного. Как следствие, они выполнены в корпусах с большим разнообразием в числе выводов: от 14 до 44.

Сигма-дельта АЦП работает значительно быстрее и при этом демонстрирует разрешение от 10 до 16 разрядов. Входные каналы (от 1 до 4) могут быть сконфигурированы как несимметричные или дифференциальные. Двухпроводной интерфейс не является стандартом, и для новых разработок такие микросхемы не рекомендуются.

Новейшее поколение аналого-цифровых преобразователей — это АЦП с последовательным приближением, оснащенные интерфейсом SPI. К таким устройствам относится семейство MCP3XYY, где X обозначает разрешение, а YY — количество входов. Значению X = 0 соответствует разрешение 10 разрядов; X = 2 — 12 разрядов, а X = 3 — 13 разрядов.

Количество выводов составляет 1, 2 , 4–8. Кроме того, в этой серии существуют типы с несимметричными и дифференциальными входами.

Еще один немного "медлительный" тип АЦП с интерфейсом I^2C — MCP3221 в корпусе SOT-23A (пять выводов). Он предлагает разрешение 12 разрядов и один входной канал (а не 21 — правая "двойка" в маркировке указывает на наличие интерфейса I^2C).

Если используется микроконтроллер, то на время аналого-цифрового преобразования его, возможно, лучше перевести в "спящий" режим. Кроме того, следует позаботиться о раздельном питании цифровой и аналоговой частей схемы, а также о пространственном разделении обеих частей на печатной плате.

11.5. ЦАП

До недавнего времени выбор ЦАП компании Microchip был не особо велик: только микросхемы TC1320 и TC1321. Оба устройства выполнены в корпусе SMD, оснащены интерфейсом SMBus и предлагают разрешение 8 или 10 разрядов. Время установления выхода 10 мкс не назовешь выдающимся, однако в большинстве случаев оно вполне приемлемо. Большим недостатком этих двух микросхем является отсутствие источника опорного напряжения.

Относительно новым является ЦАП типа MCP4921/22 (рис. 11.16).

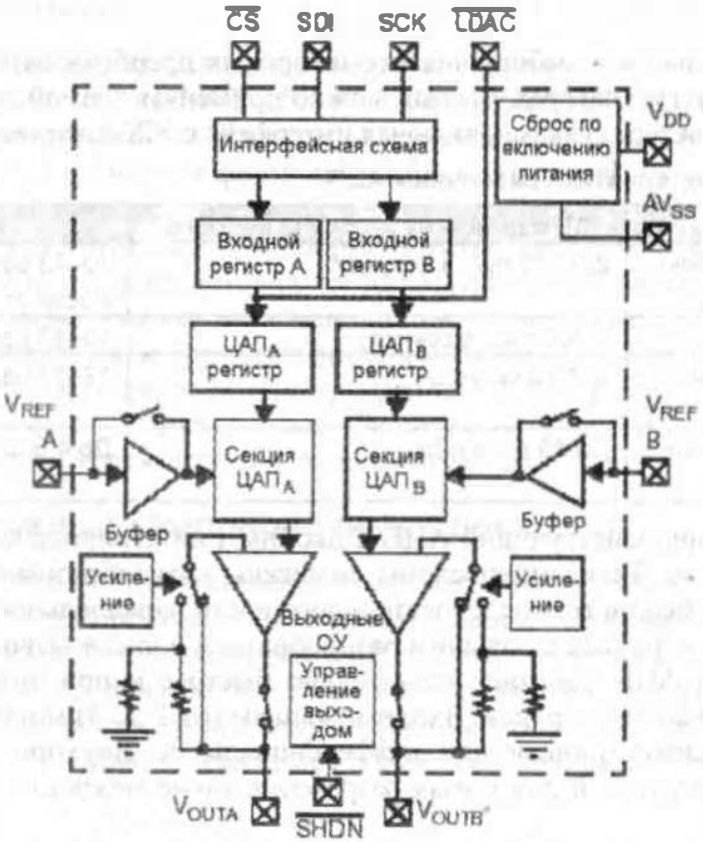


Рис. 11.16. ЦАП с двумя силовыми выходами (в MCP4921 — только один выход)

Хотя этот ЦАП также не отличается особым быстродействием, он оснащен выходными усилителями, которые обычно обеспечивают ток силой 15 мА и даже ток короткого замыкания силой 25 мА. Таким образом, MCP4921/22 способен справиться с большой емкостной нагрузкой.

11.6. Цифровой потенциометр

11.6.1. Взгляд изнутри

Цифровой потенциометр (рис. 11.17) от 10 до 100 k оснащен высокоскоростным интерфейсом SPI.

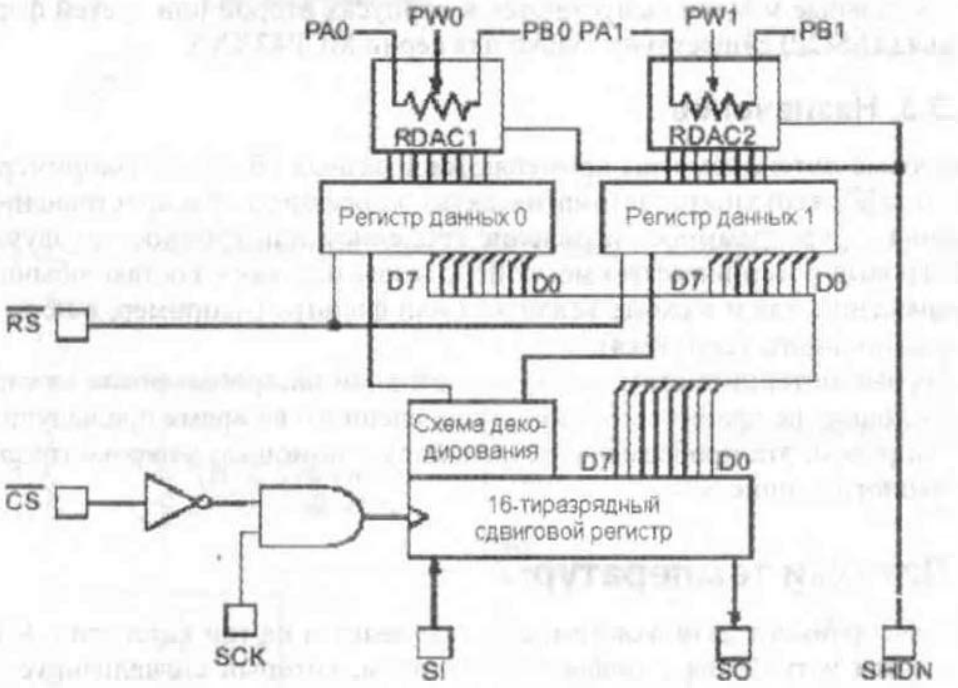


Рис. 11.17. Схема цифрового потенциометра MCP42XXX

Но шина SPI отличается не только высокой скоростью передачи данных. Еще одна важная ее особенность заключается в том, что несколько модулей можно подключить последовательно — так называемое гирляндное подключение (рис. 11.18).

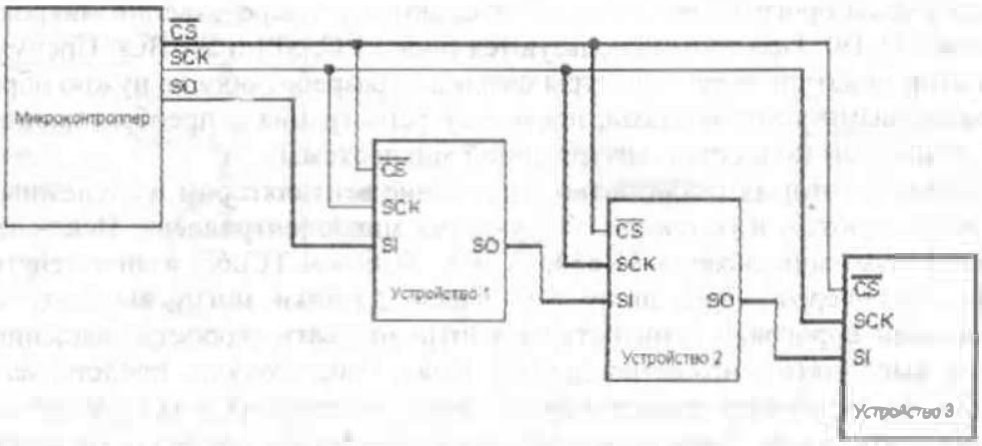


Рис. 11.18. Гирляндное подключение нескольких цифровых потенциометров

Единственное ограничение состоит в том, что внутри цепочки не может использоваться модуль MCP41XXX, поскольку в нем отсутствует вывод SO (Serial Output). Впрочем, представители этой серии могут выполнять роль замыкающего звена.

Если последовательно по всей цепочке потенциометров проводится некоторое установочное значение, то на вход CS всех микросхем на определенное время подается сигнал высокого уровня.

11.6.2. Многообразие моделей

Маркировка потенциометров очень проста в расшифровке и соответствует шаблону "MCP4XYYY", где X равно 1 или 2 (количество потенциометров в одном корпусе), а YYY представляет значение сопротивления (010 k, 050 k и 100 k).

Все остальные модели выпускаются в корпусах второй или третей формы, причем третья (TSSOP) существует только для серии MCP42XXX.

11.6.3. Назначение

Цифровые потенциометры применяются в разных областях. Например, с их помощью можно выполнять автоматическую коррекцию. Распространенная сфера применения — программное управление усилением или громкостью звучания. Для этого цифровые потенциометры могут использоваться как в составе обычного делителя напряжений, так и в схеме усилителя или фильтра (например, в цепи обратной связи операционного усилителя).

Цифровые потенциометры требуют повторной настройки после каждого включения, поскольку не хранят значения, установленного во время предыдущего сеанса работы. Впрочем, эта проблема легко решается с помощью микроконтроллера PIC, подключенного к шине SPI.

11.7. Датчики температуры

Весь ассортимент датчиков температуры делится на три категории. К первой из них относятся устройства с цифровым выходом, который сигнализирует только о том, находится ли температура ниже или выше определенного порога. Второй класс датчиков оснащен аналоговым выходом. Одному градусу Цельсия соответствует изменение напряжения на 6,25 или 10 мВ. Это значение считывается АЦП, с помощью которого и получают фактическую температуру.

Последнее поколение датчиков оснащено последовательным интерфейсом. Такие микросхемы программируются и считываются непосредственно микроконтроллером (рис. 11.19). При этом используются шины I^2C, SPI и SMBus. Преимущество этой категории датчиков температуры очевидно: разработчику не нужно обременять себя аналоговыми компонентами, поскольку регистрация и преобразование напряжения происходит полностью внутри самой микросхемы.

Датчики, в которых реализовано управление вентилятором и отслеживание отказов, используются, в основном, без участия микроконтроллера. Исключение составляют только микросхемы TC654, TC655, TC664 и TC665, взаимодействующие с микроконтроллером через шину SPI. Такие датчики могут выдавать сигналы о превышении пороговой температуры, контролировать скорость вращения вентилятора и выполнять множество других задач. Так, модуль, представленный на рис. 11.20, на основании измеренной датчиком температуры регулирует скорость вращения вентилятора. Сила тока, протекающего через вентилятор, регистрируется, позволяя легко установить эффективность охлаждения.

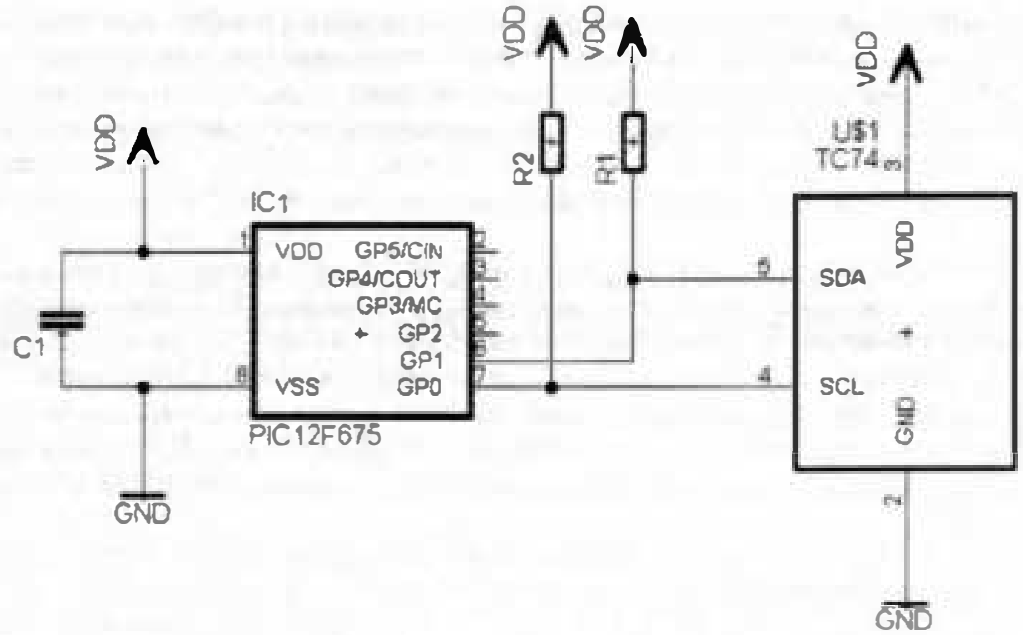


Рис. 11.19. Датчик температуры TC74, подключенный к шине I²C микроконтроллера PIC12Fxxx

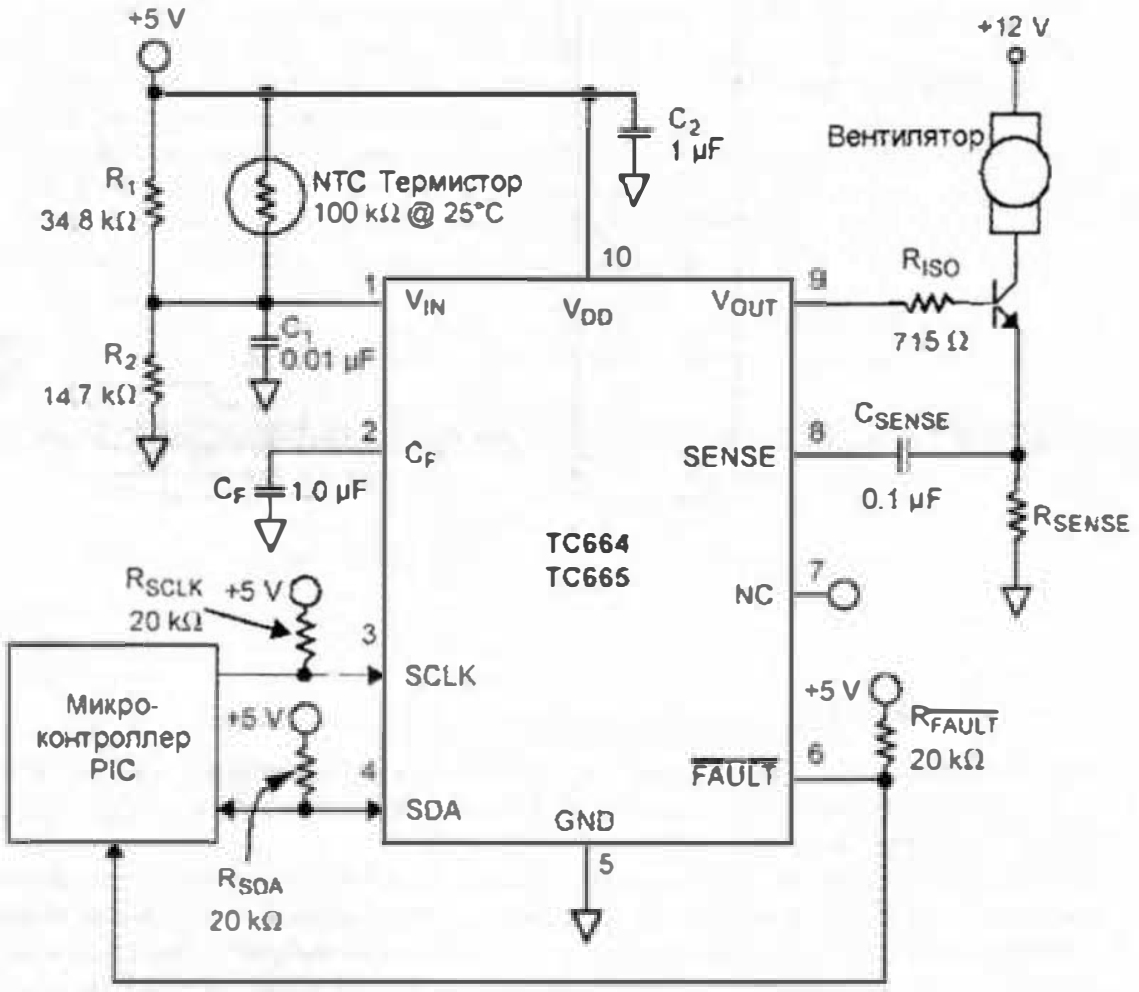


Рис. 11.20. Типичное применение микросхемы TC664

Альтернативой TC74 является цифровой термометр DS1620 производства Dallas, которому для регистрации температуры не требуется никаких внешних компонентов. Он может измерять температуры в диапазоне –55...+125°C. Значение измерения 0 соответствует температуре 0°C, а разрешение составляет 0,5°C, чтобы измеряемые значения можно было представить девятью разрядами. Таким образом, измеренному значению 1 соответствует 0,5°C, а значению 1FFH — температура -0,5°C.

Отсюда следует, что девятый разряд не следует рассматривать как знаковый, а остальные восемь разрядов — как положительное число. При установке девятого разряда для вычисления абсолютного значения температуры остальной байт должен быть инвертирован и разделен на два, поскольку разрешение составляет 0,5°. Другими словами, термометр DS1620 выдает значение в виде конегативного числа.

Линия выбора, которая в случае DS1620 обозначается как /RST, разрешает несколько таких же компонентов, подключенных к линиям тактирования и данных (рис. 11.21).

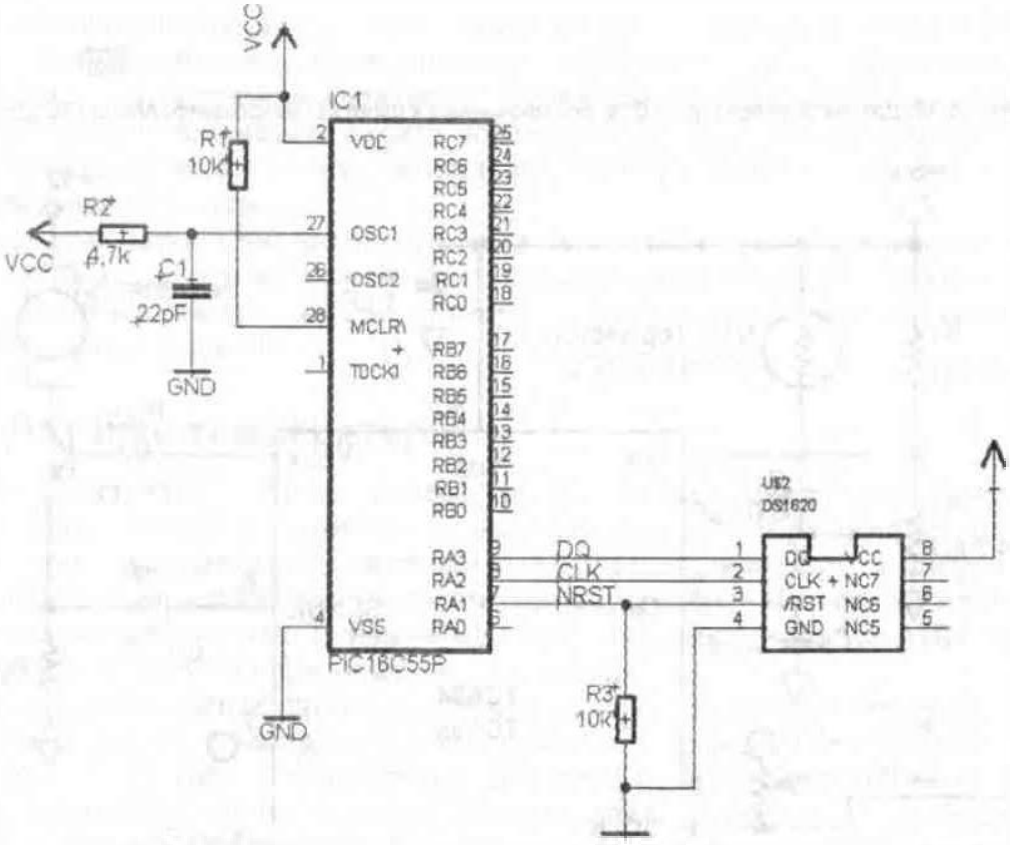


Рис. 11.21. Пример включения цифрового термометра DS1620

Протокол передачи, к сожалению, не совместим ни с интерфейсом I^2C, ни с SPI. По этой причине обмен данными с DS1620 осуществляется без каких-либо последовательных периферийных модулей.

С помощью регистра конфигурации термометр DS1620 может быть настроен для работы в различных режимах. В частности, можно выбрать автономный режим, в котором не требуется участие микроконтроллера. Если выбран режим одиночных измерений, то после получения стартовой команды всегда выполняется только один замер. Разряды выбора режима хранятся в регистре типа EEPROM.

В результате чтения регистра конфигурации получают также и информацию о состоянии устройства (например, "Done" — "преобразование выполнено"). Кроме того, можно воспользоваться флагами, уведомляющими о том, в каком положении относительно некоторого запрограммированного значения находится текущая температура.

Взаимодействие между DS1620 и микроконтроллером реализовано с помощью набора команд (табл. 11.7).

Таблица 11.7. Команды для работы с цифровым термометром DS1620

Команда	Описание
AAH	Считывание температуры
01H	Запись TH
02H	Запись TL
A1H	Чтение TH
A2H	Чтение TL
EEH	Начать преобразование (преобразование длится 1 с)
22H	Остановить преобразование (в режиме одиночных измерений смысла не имеет)
0CH	Запись конфигурации
ACH	Считывание конфигурации

Все команды, кроме команд запуска и останова преобразования, считывают или записывают некоторый параметр. Обращаем внимание на то, что этот параметр — девятиразрядный, поскольку мы имеем дело со значением температуры. Разрядность регистра конфигурации — байт.

Процедуры записи/чтения — типичные, и потому останавливаться на них мы не будем. Напомним только, что по низкому уровню сигнала на тактовой линии выполняется как чтение, так и запись, а первым передается младший разряд данных.

Пример программы для взаимодействия с термометром DS1620 представлен в листинге 11.1.

Файл DS1620.am находится также на прилагаемом к книге компакт-диске в папке Prog_bsp.

Листинг 11.1. Программа DS1620.am

```
;=============================================================
;      Ассемблерный модуль для цифрового термометра
;=============================================================

#DEFINE  DQ        PORT?.?
#DEFINE  CLK       PORT?.?
#DEFINE  RST       PORT?.?
;
V_DS     EQU   ??H           ; Корректировка основания
EVENT    EQU   V_DS
UTIM     EQU   V_DS+1
PSEUDO   EQU   V_DS+2
COUNT    EQU   V_DS+3
FEHLER   EQU   V_DS+4
TRANS    EQU   V_DS+5
CONF     EQU   V_DS+6
TEMP     EQU   V_DS+7
UCOUNT   EQU   V_DS+8
BCD      EQU   V_DS+9
;
```

Листинг 11.1. Продолжение

```
; Макрос для PIC16CXX
SETRIS   MACRO TRISC,PIN
         BANK_1
         BSF  TRISC,PIN
         BANK_0
         ENDM
CLTRIS   MACRO TRISC,PIN
         BANK_1
         BCF  TRISC,PIN
         BANK_0
         ENDM
;
; Макрос для PIC16C5X
;
SETRIS MACRO  PORTC,PIN
       BSF    CTRIS,PIN
       MOVF   CTRIS,W
       TRIS   PORTC
       ENDM
CLTRIS MACRO  PORTC,PIN
       BCF    CTRIS,PIN
       MOVF   CTRIS,W
       TRIS   PORTC
       ENDM
;-------------------------------------------------------------------------
;       MKBCD: получает из шестнадцатеричного ZAHL число в BCD-формате
;       Вход: ZAHL
;       Выход: ZAHL
;       Вспомогательная переменная: ZEHNER
;-------------------------------------------------------------------------
MKBCD  CLRF    BCD
SUZ    MOVLW   .10
       SUBWF   ZAHL,W          ; W:=ZAHL-10
       BNC     ENDE            ; ENDE, если ZAHL<10
       MOVWF   ZAHL            ; ZAHL:=ZAHL-10
       INCF    BCD             ; BCD содержит число, кратное 10
       GOTO    SUZ
ENDE   SWAPF   BCD,W           ; Старший полубайт(BCD):=ZAHL DIV 10
       IORWF   ZAHL            ; ZAHL:= (ZAHL DIV 10)*16 + (ZAHL MOD 10)
       RETLW  0
;
;=========================Подпрограмма для DS1620
; Обмен данными с DS1620 происходит по низкому уровню тактового сигнала,
; первым передается младший разряд данных
;-------------------------------------------------------------------------
;       WRLO: запись COUNT бит из TRANS, начиная с младшего
;       Вход: COUNT бит из TRANS
;-------------------------------------------------------------------------
WRLO     CLTRIS  DQ
WRLO1    BCF     CLK             ;Низкий уровень тактового сигнала
         RRF     TRANS           ;Бит TRANS - в DQ
         SKPC
         BCF     DQ
         SKPNC
         BSF     DQ
         NOP
         BSF     CLK             ;Высокий уровень тактового сигнала
```

Листинг 11.1. Продолжение

```
        DECFSZ  COUNT
        GOTO    WRLO1
        SETRIS  DQ
        RETLW   0
;
;-----------------------------------------------------------------------
;       RDLO: чтение COUNT бит в TRANS
;       Вход: COUNT
;       Выход: TRANS
;-----------------------------------------------------------------------
RDLO    BCF     CLK         ; Низкий уровень тактового сигнала
        NOP
        BTFSS   DQ
        BCF     STATUS,CY
        BTFSC   DQ          ; DQ в CY
        BSF     STATUS,CY
        RRF     TRANS       ; CY в TRANS
        BSF     CLK         ; Высокий уровень тактового сигнала
        DECFSZ  COUNT
        GOTO    RDLO
        RETURN
;
;-----------------------------------------------------------------------
;       RDCONF: чтение байта конфигурации/состояния в TRANS
;       Выход: TRANS = DONE xxx xx CPU 1SHOT
;-----------------------------------------------------------------------
RDCONF  MOVLW   0ACH
        MOVWF   TRANS
        MOVLW   8
        MOVWF   COUNT
        BSF     RST             ;Интерфейс включен
        CALL    WRLO
        MOVLW   8
        MOVWF   COUNT
        CALL    RDLO
        BCF     RST             ;Интерфейс отключен
        RETURN
;
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
;       RDTEMP: чтение 9 бит, в результате чего младший разряд - в CY
;       Выход: TEMP=TRANS = VZ + 7 бит температуры в градусах Цельсия
;              CY = 1: + 0.5
;-----------------------------------------------------------------------
RDTEMP  MOVLW   0AAH            ; Команда для считывания температуры
        MOVWF   TRANS
        MOVLW   8
        MOVWF   COUNT
        BSF     RST             ;Интерфейс включен
        CALL    WRLO
        MOVLW   9
        MOVWF   COUNT
        CALL    RDLO
        MOVF    TRANS,W
        MOVWF   TEMP
        BCF     RST             ;Интерфейс отключен
        RETLW 0
```

Листинг 11.1. Окончание

```
;
;Команды
;
CSTOP   MOVLW   022H            ; Команда: Остановить обмен
        GOTO    CST             ; Не используется, поскольку режим
                                ;  одиночных измерений
CSTART  MOVLW   0EEH            ; Команда: Начать обмен
CST     MOVWF   TRANS
        MOVLW   8
        MOVWF   COUNT
        BSF     RST             ; Интерфейс включен
        CALL    WRLO
        BCF     RST             ; Интерфейс отключен
        RETURN
;
;-----------------------------------------------------------------------
;       CONFIG: запись 3 в конфигурацию (DONE xxx xx CPU 1SHOT)
; В случае PIC16C5X следует учитывать следующее:
; CONFIG не может быть подпрограммой из-за неглубокого стека
;-----------------------------------------------------------------------
CONFIG  CALL    RDCONF
        MOVF    TRANS,W
        ANDLW   3
        XORLW   3
        SKPNZ                   ; Если 2 разряда уже = 3, не записываем
        GOTO    READI           ; Поскольку это - EEPROM, не лишне записать
        MOVLW   3
        MOVWF   CONF
        MOVLW   0CH             ; Команда: Запись конфигурации
        MOVWF   TRANS
        MOVLW   8
        MOVWF   COUNT
        BSF     RST             ; Интерфейс включен
        NOP
        CALL    WRLO
        MOVF    CONF,W
        MOVWF   TRANS
        MOVLW   8
        MOVWF   COUNT
        CALL    WRLO
        BCF     RST             ; Интерфейс отключен
READI   RETURN
;-------------------------------------
; Конец модуля
```

11.8. Генераторы сигнала сброса

Сброс микроконтроллера — важное событие, поскольку служит для запуска программы в определенном порядке. Если напряжение питания нарастает быстро, то никаких проблем не возникает, поскольку внутренняя схема сброса функционирует корректно. Если же электропитание нестабильное, то необходимо позаботиться о том, чтобы микроконтроллер находился в состоянии сброса до тех пор, пока не будет получено определенно номинальное значение питающего напряжения.

Генератор сигнала сброса, оснащенный встроенным компаратором и источником опорного напряжения, следит за уровнем напряжения питания микроконтрол-

лера. Выход сброса освобождает только тогда, когда компаратор устанавливает превышение некоторого заданного порога.

Преимущество таких генераторов заключается в том, что они надежно работают и корректно управляют выходом сброса даже при малых значениях напряжения.

11.8.1. Взгляд изнутри

Перечень генераторов сигнала сброса (их также называют системными супервизорами) на данный момент состоит из более, чем двадцати наименований. За некоторыми исключениями, все они работают на напряжении от 1 до 5,5 В. Очень часто приходится оперировать таким значением, как установившийся ток, поэтому имеет смысл остановиться на нем подробнее. Его сила лежит в пределах 6–900 мкА, потому можно подобрать вполне приемлемый вариант генератора для приборов, работающих от аккумуляторных батарей. Ассортимент пороговых напряжений (от 3 до 7) достаточен для систем, рассчитанных как на 5 В, так и на 3,3 В. В качестве выходного каскада, наряду с пушпульными моделями, используются также и типы с открытым стоком.

11.8.2. Виды корпусов

На данный момент наблюдается тенденция перехода к корпусам SMD, однако еще нередко используются и транзисторные корпуса TO-92.

11.8.3. Альтернативные способы применения

"Обычный" способ применения генератора сигнала сброса вполне очевиден, однако его можно использовать и не по прямому назначению.

В некоторых случаях напряжение питания колеблется в таком широком диапазоне (от 2,5 до 5 В, причем микроконтроллер при этом работает корректно), что частота осциллятора в результате воздействия этих колебаний перестает быть достоверной — частота RC-осциллятора модулируется значением сопротивления сенсора. Тем не менее, поскольку время от времени оптимальное напряжение надежно, для получения достоверной частоты мы могли бы ожидать только соответствующий период. Задача определения таких периодов возлагается на генератор сигнала сброса, поскольку он надежно работает при напряжениях питания от 1 В. Тем самым, проблема решена. Генератор, для которого установлен порог 4,7 В, гарантирует, что напряжение питания будет находиться в диапазоне 4,7–5 В, а колебания в таком диапазоне для осциллятора вполне приемлемы. Информацию от генератора сигнала сброса можно получать по обычной линии ввода-вывода.

11.8.4. Модельный ряд

А теперь представим характеристики различных генераторов сигнала сброса *в виде таблицы (табл. 11.8)*.

Таблица 11.8. Характеристики различных генераторов сигнала сброса

Тип	*Диапазон VCC*	*Активный уровень сигнала сброса*	*Тип выхода*	*Типичное потребление тока*	*Варианты корпуса*
TCM811	1,0...5,5 В	Низкий	Пушпульный	6 мкА	SOT-143
MCP100	1,0...5,5 В	Низкий	Пушпульный	45 мкА	TO-92 SOT23B

Таблица 11.8. Окончание

Тип	Диапазон VCC	Активный уровень сигнала сброса	Тип выхода	Типичное потребление тока	Варианты корпуса
TCM810	1.2...5,5 В	Высокий	Пушпульный	12 мкА	SOT23B SC-70
MCP120	1.0...5,5 В	Низкий	Открытый сток	45 мкА	TO-92 SOT23 SOIC (8 выв.)
MCP130	1,0...5,5 В	Низкий	Открытый сток с подтягивающим резистором на 5к	45 мкА	TO-92 SOT23 SOIC (8 выв.)

11.9. LDO-стабилизаторы напряжения

Ввиду существования устройств, работающих от аккумуляторных батарей, имеет смысл упомянуть также и об LDO-стабилизаторах напряжения MCP1700 и MCP1701. В том случае, когда речь идет о портативных приборах с питанием от батарей 9 В, оптимальный вариант — MCP1701. Некоторые технические характеристики этого стабилизатора представлены в табл. 11.9.

Таблица 11.9. Технические характеристики стабилизатора напряжения MCP1701

Характеристика	Значение
Максимальное входное напряжение	10 В
Выходное напряжение	1,8 В; 2.5 В; 3,0 В; 3.3 В; 5,0 В
Максимальный выходной ток	250 мА
Температурный диапазон	–40...+85°C
Варианты корпуса (все — с тремя выводами)	SOT-23A, SOT-89, TO-92 (рис. 11.22)

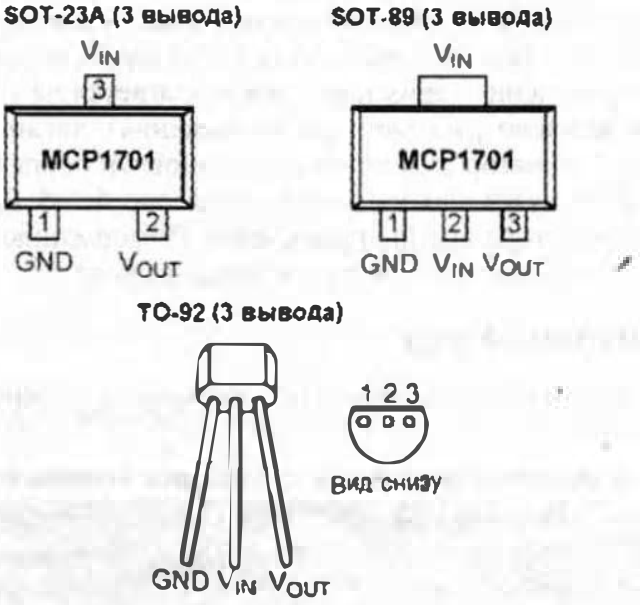


Рис. 11.22. Различные варианты корпуса для MCP1701

Схема включения этого стабилизатора — проще не придумаешь (рис. 11.23).

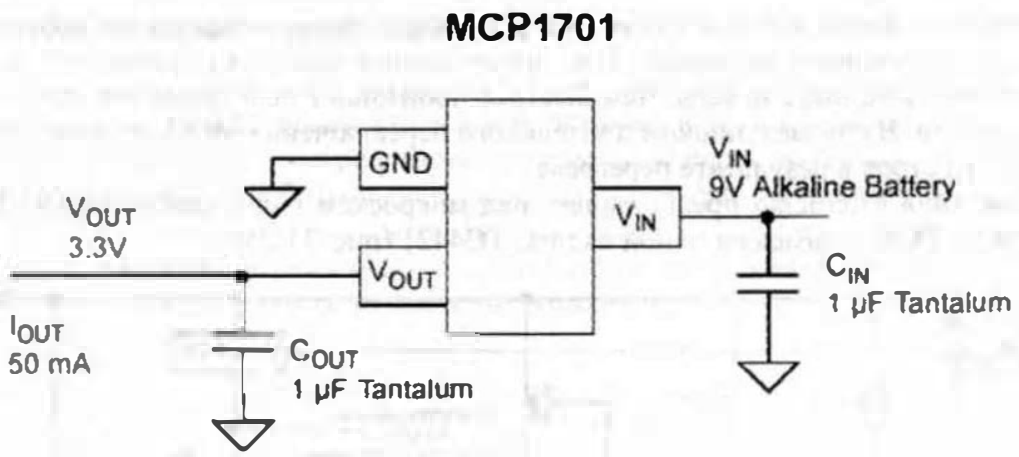


Рис. 11.23. Питание 3,3 В в устройстве на 9 В

В том случае, если прибор долгое время находится в отключенном состоянии и включается только по нажатию кнопки, можно сэкономить также и на токе покоя стабилизатора. Для этого, в случае необходимости, источник 9 В включают через МОП-транзистор (рис. 11.24).

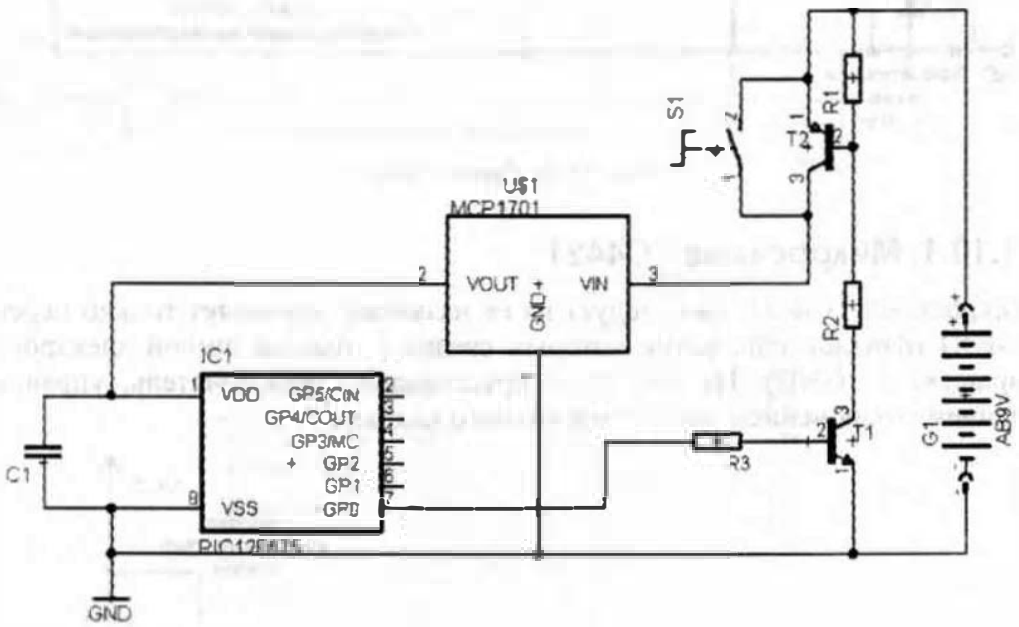


Рис. 11.24. Отключение батареи 9 В

Если в результате бездействия пользователя устройство длительное время остается неактивным, микроконтроллер может разомкнуть транзистор. В качестве элементов T1 и T2, изображенных на рис. 11.24, могут быть использованы как МОП-, так и биполярные транзисторы.

11.10. Схемы управления МОП-транзисторами

МОП-транзисторы характеризуются высокой скоростью переключения, однако управление ими — задача нетривиальная. Корень всех зол — большая емкость затвора. В результате, для того чтобы переключить МОП-транзистор, требуется не только приложить напряжение 10 В между затвором и истоком, но и сделать эти 10 В относительно низкоомными. Это необходимо для ускорения процесса пере-

ключения, во время которого транзистор проходит фазу, когда он выполняет роль резистора переменной величины. Ток, протекающий через этот "резистор", дает определенную мощность потерь. Чем быстрее произойдет переключение, тем меньше будут потери. В случае слишком длительного переключения МОП-транзистор будет выведен из строя в результате перегрева.

Компания Microchip предоставляет ряд микросхем для управления МОП-транзисторами. Остановимся на одной из них: TC4421 (рис. 11.25).

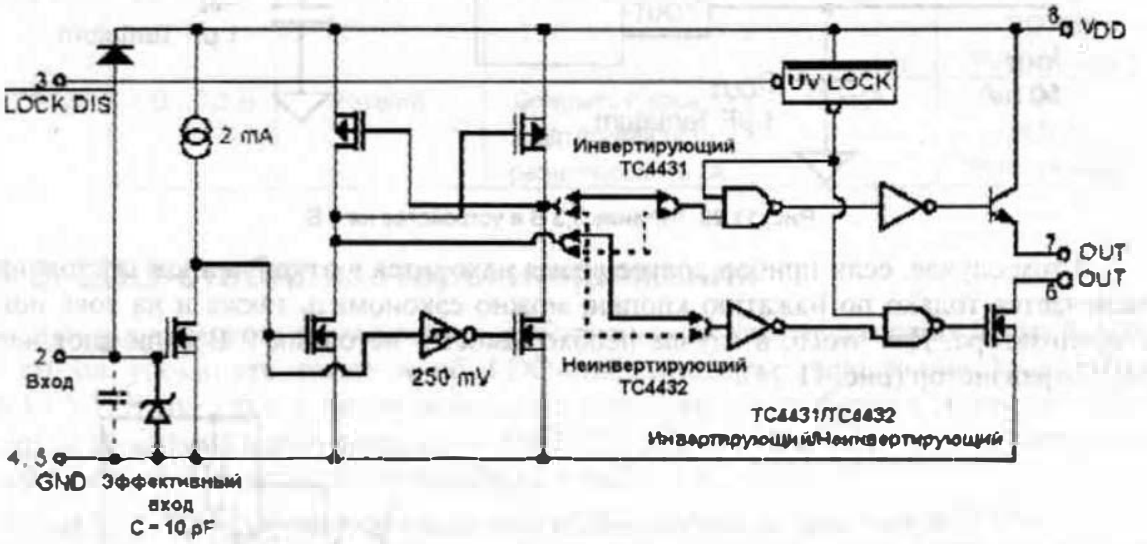


Рис. 11.25. Схема TC4421

11.10.1. Микросхема TC4421

Микросхема TC4421, как следует из ее названия, управляет только переключением МОП-транзисторов, исток которых связан с нижней шиной электропитания (как правило — GND). На рис. 11.26 представлен переключатель, управляющий включением/отключением электромагнитного клапана.

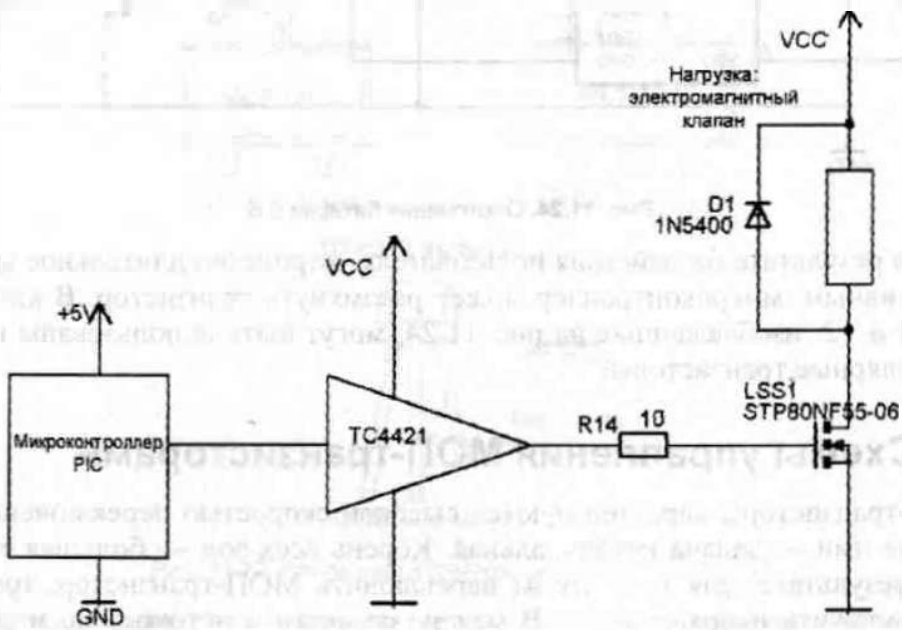


Рис. 11.26. Пример применения микросхемы TC4421

11.10.2. Микросхема MIC5016

В автомобиле, благодаря наличию шасси, повсюду доступно подключение к массе (GND). Положительное напряжение питания подается по кабелю к соответствующему потребителю, что реализовано с помощью переключателя.

Для управления подобными переключателями служит, например, микросхема MIC5016 от компании Micrel. Вывод истока верхнего МОП-транзистора (сток — напряжение питания) играет роль основания для напряжения, вырабатываемого в микросхеме. Таким образом происходит управление затвором МОП-транзистора (рис. 11.27).

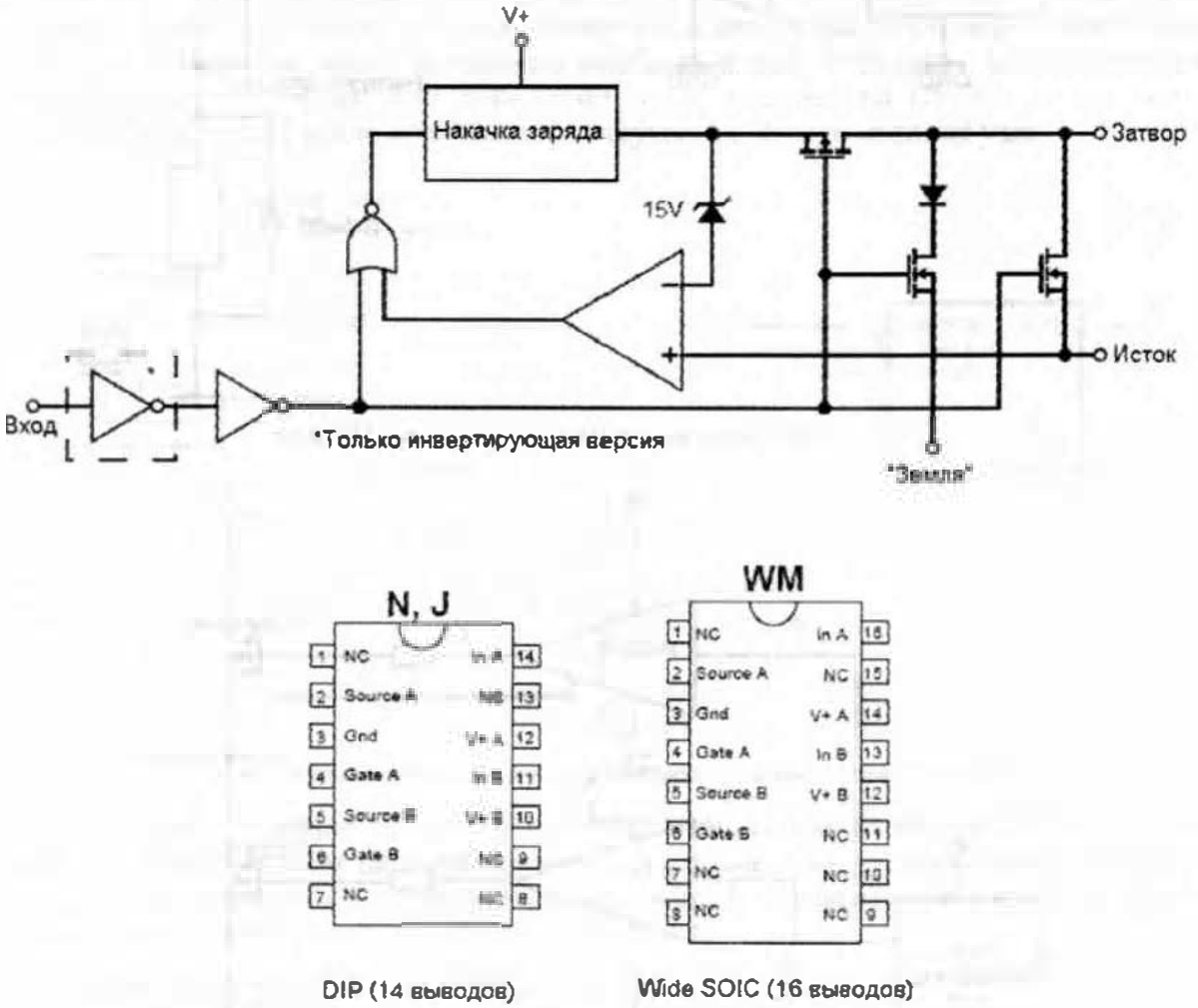


Рис. 11.27. Схема и типы корпуса для MIC5016

Пример использование микросхемы MIC5016 представлен на рис. 11.28. Для построения Н-моста, состоящего из двух полумостовых схем, две части объединяются в одну (рис. 11.29).

В случае управления скоростью вращения двигателя, на вход управляющей схемы обычно подается ШИМ-сигнал, полученный, например, с помощью модуля CPP микроконтроллера PIC. Если теперь необходимо реализовать еще и изменение направления вращения, то двигатель включается в состав Н-мостовой схемы. Для того чтобы обеспечить обязательными сигналами все четыре микросхемы управления, требуются или внешние компоненты, или несколько портов ввода-вывода. Можно также использовать модуль ECCP, работающий в соответствующем режиме. В та-

ком случае, для управления всеми четырьмя микросхемами можно использовать четыре выхода.

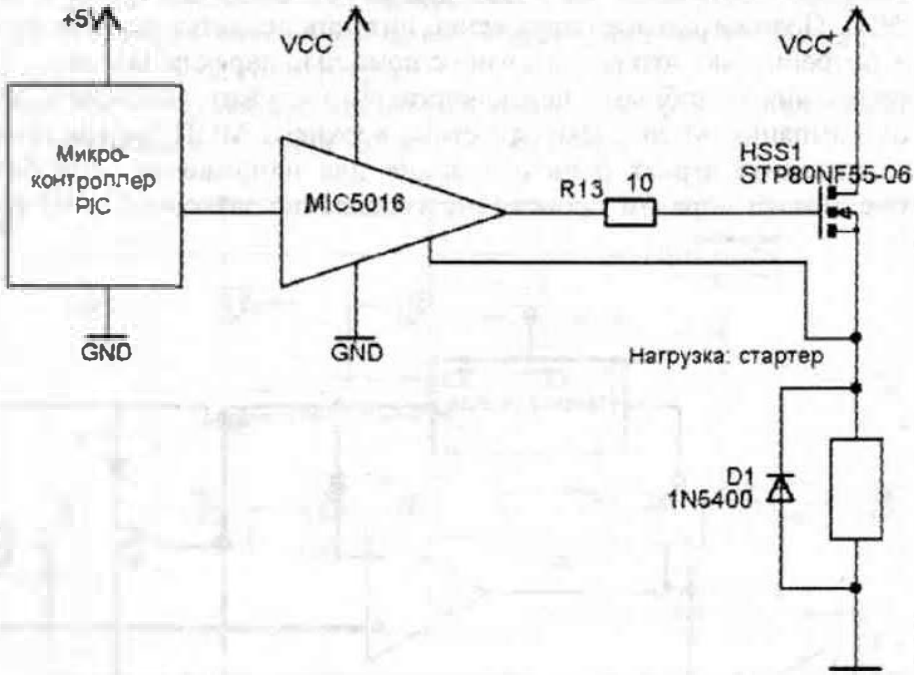


Рис. 11.28. Пример использования микросхемы MIC5016

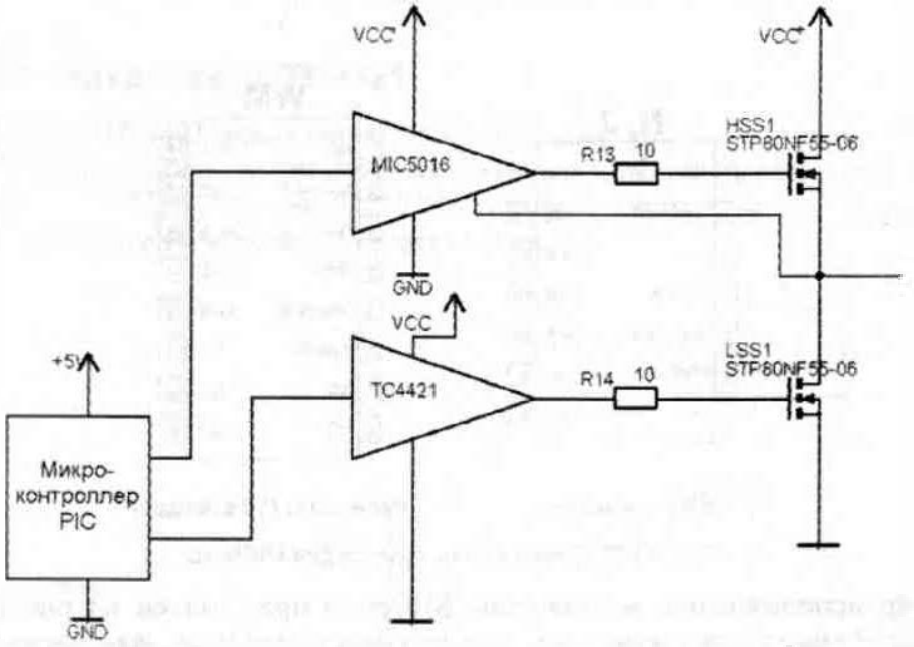


Рис. 11.29. Полумостовая схема с переключателями MIC5016 и TC4421

Переключение направления вращения — еще одно щекотливое дело. Необходимо гарантировать, что ключ одной полумостовой схемы размыкается до того, как другой ключ той же схемы получит сигнал на замыкание. Более подробно информацию на этот счет можно найти в описании модуля ECCP по ключевой фразе "dead-band delay".

11.11. Модули часов

Хорошие микросхемы выпускают не только в Microchip. К примеру, мы уже многие годы успешно применяем такие модули часов производства Dallas как DS1302 и DS1307. Обе микросхемы для получения реального времени используют кварц на 32 кГц.

11.11.1. DS1302

Особенность микросхемы DS1302 (рис. 11.30) — так называемый "капельный заряжатель". Он представляет собой резервную систему электропитания, которая одновременно с основной системой, может быть нагружена током определенной величины. В качестве такого резервного варианта может, к примеру, использоваться аккумуляторная батарея. Как показывает опыт, микросхема DS1302 выдает корректное время даже после получасового отсутствия общего питания узла.

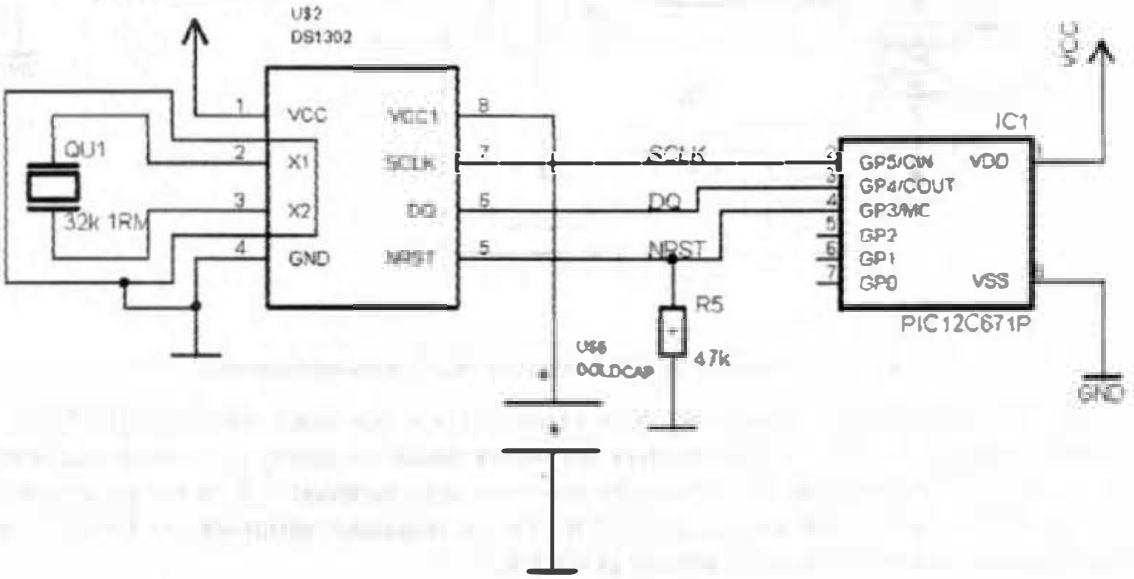


Рис. 11.30. Сопряжение микроконтроллера PIC и микросхемы DS1302

Например, в случае системы управления отоплением, из модуля часов каждый час считывает значение реального времени. В дальнейшем, с целью документирования, это значение вместе с информацией о текущей температуре и состоянии системы отопления записывается в память EEPROM.

11.11.2. DS1307

В отличие от DS1302, эта микросхема оснащена интерфейсом I^2C. Благодаря этой особенности, ее можно, к примеру, подключить к одной шине с несколькими модулями памяти EEPROM.

Резервная система электропитания для DS1307 должна быть основана на литиевом аккумуляторном элементе. Согласно данным производителя, один такой элемент емкостью 35 мА/ч поддерживает микросхему часов в рабочем состоянии более десяти часов.

В схеме, представленной на рис. 11.31, модуль DS1307 со своим кварцем соединен по шине I^2C с микросхемой последовательной памяти EEPROM и микроконтроллером PIC12C671. Микроконтроллер, который находится в "спящем" режиме, "пробуждается" по сигналу от модуля часов, выполняет определенную работу (на-

пример, измеряет температуру и записывает ее вместе с меткой времени в память EEPROM) и опять "засыпает". Частота выдачи сигнала модулем часов настраивается и может принимать значение 1 Гц, 4 кГц, 8 кГц или 32 кГц.

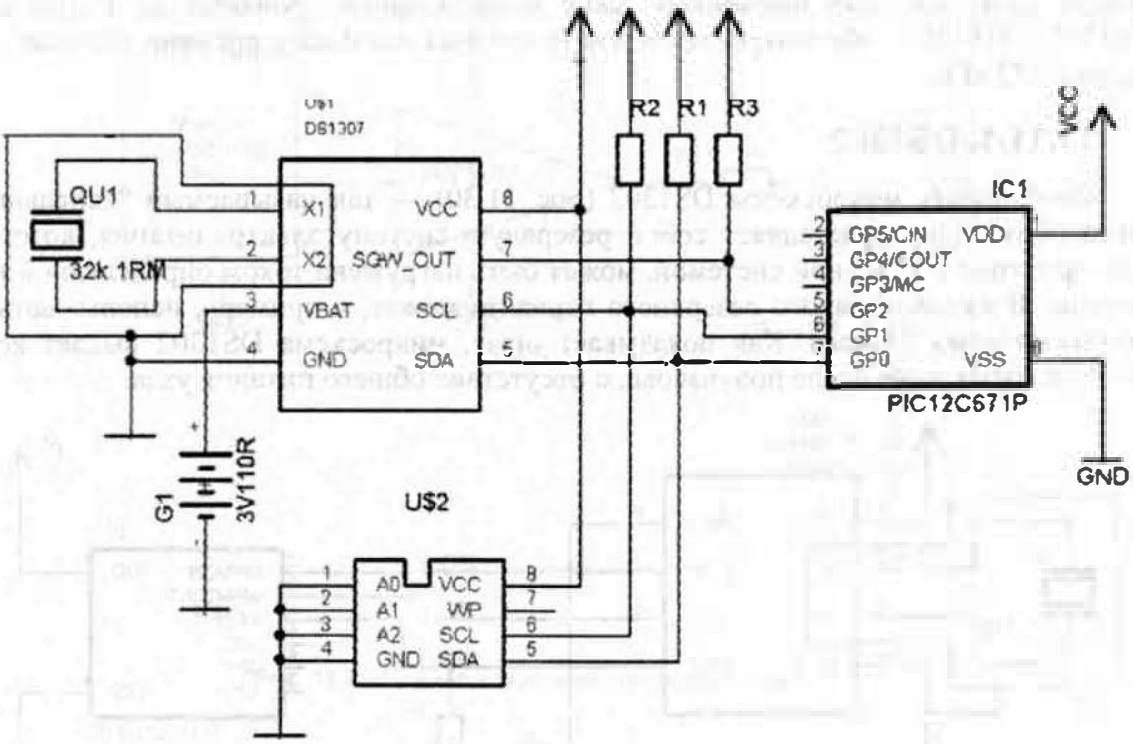


Рис. 11.31. Модуль DS1307 в качестве часов реального времени

Когда напряжение питания модуля часов опускается ниже значения U_{bat}*1,25, то остается активной только внутренняя функция часов, а доступ ко всем регистрам невозможен. Напряжение V_{CC} специфицировано на значение 5 В, а напряжение V_{bat} должно находиться в диапазоне 2,5–3,5 В. Таким образом, микросхема DS1307 пригодна только для систем, рассчитанных на 5 В.

11.12. Структура источника питания

Не каждый прибор, благодаря LDO-стабилизатору, обеспечен аккумуляторной батарсей, и не каждый требует наличие сложного блока питания для обеспечения необходимого тока. Множество устройств требуют очень незначительного тока и при этом питаются от сети.

В подобных случаях используют источники питания с высоким КПД и малой мощностью. Ближайшим приближением к оптимуму можно назвать подходящий штекерный блок питания с трансформатором и включенным последовательно стабилизирующим каскадом.

В последние годы на рынке появились стабилизаторы напряжения, которые питаются от постоянного напряжения в диапазоне электросети и своим выходом управляют трансформатором, способным выдать почти любое значение напряжения.

Подобные стабилизаторы выпускаются разными производителями, одним из которых является компания PI (Power Integrations). Выпускаемые ею семейства LinkSwitch, TopSwitch, EcoSmart и TinySwitch покрывают большой диапазон мощ-

ностей. В качестве примера можно назвать модуль LinkSwitch LNK500, рассчитанный на малую мощность (рис. 11.32).

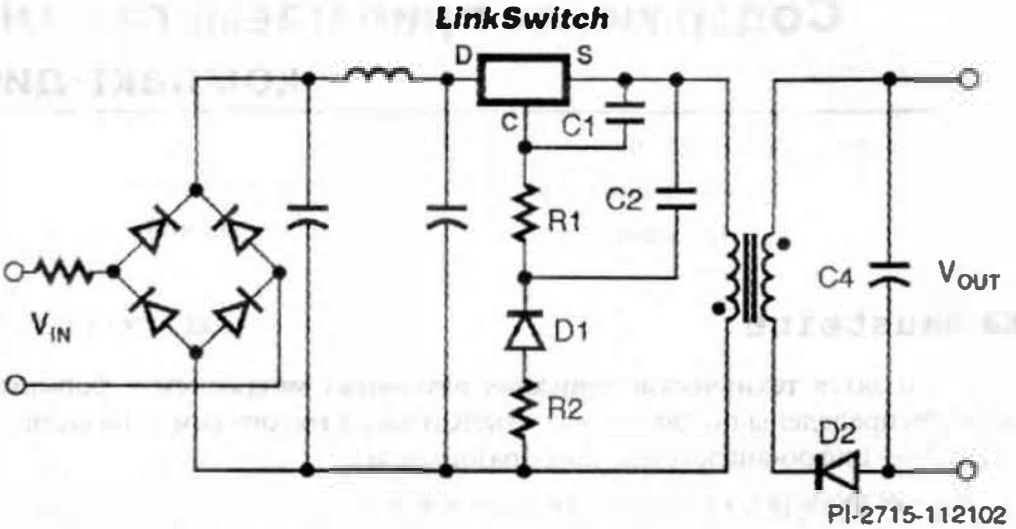


Рис. 11.32. Пример схемы с применением LNK500

Содержимое прилагаемого к книге компакт-диска

Папка Bausteine

Здесь находятся технические описания различных микросхем в формате PDF. Все файлы распределены по папкам в соответствии с категориями устройств:

- AD_DA — цифро-аналоговые преобразователи;
- CAN — устройства, имеющие отношение к шине CAN;
- contoller — некоторые микроконтроллеры PIC;
- dig_pot — цифровые потенциометры семейств MCP41XXX/MCP42XXX;
- io_exp — расширитель портов ввода-вывода MCP23016;
- ir — устройства с поддержкой технологии IrDA;
- lin — LIN-приемопередатчик MCP201;
- MOS_DRV — схемы управления МОП-транзисторами;
- op_comp — операционные усилители и компараторы;
- PI — изделия компании Power Integrations;
- Resetgen — генераторы сигнала сброса;
- speicher — микросхемы последовательной памяти EEPROM;
- spg_rgl — стабилизатор напряжения MCP1700;
- Tempsens — датчики температуры;
- timing — модули часов.

Папка CAN

Файлы примеров и документация по использованию шины CAN (см. раздел 2.4).

Папка MPLAB

Программы установки среды разработки MPLAB версий 5.7, 6.60 и 7.00, а также руководство пользователя в формате PDF.

Папка PDF

Документы в формате PDF, посвященные различным аспектам применения микроконтроллеров PIC, а также руководство "Product Selector Guide" от Microchip. Все файлы, кроме prodselguide.pdf, распределены по папкам в соответствии с тематикой:

- `i2c` — вопросы, имеющие отношение к шине I^2C;
- `icsp` — вопросы, имеющие отношение к внутрисхемному программированию;
- `nanowatt` — каталог устройств, использующих технологию nanoWatt, а также описание их особенностей;
- `Oszillator` — вопросы, имеющие отношение к осцилляторам;
- `spi` — вопросы, имеющие отношение к шине SPI;
- `usb` — вопросы, имеющие отношение к шине USB.

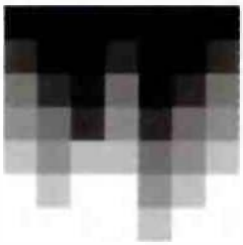
Папка `Prog_bsp`

Примеры модулей и программ на ассемблере:

- `DAC.AM` — модуль для ЦАП AD7249;
- `DS1620.AM` — модуль для цифрового термометра;
- `EE93.AM` и `EE93H.AM` — подпрограммы для микросхемы последовательной памяти EEPROM 93LCXX;
- `I2C.AM` — модуль для работы с интерфейсом I^2C для микроконтроллера 24C01;
- `ADDARAHM.ASM` — тестовая программа для ЦАП и АЦП;
- `ADDASPI.ASM` — тестовая программа для ЦАП и АЦП с шиной SPI;
- `DAC.ASM`, `DAC2.ASM`, `DACTEST1.ASM` — тесты ЦАП;
- `DIGTH.ASM` и `DIGTH55.ASM` — демонстрационные программы для цифрового термометра;
- `EE9356.ASM` и `EE9356H.ASM` — демонстрационные программы для микросхемы EE93LC56;
- `IMPERF.ASM` — программа регистрации импульсов с помощью микроконтроллера PIC16C55;
- `MUSI.ASM` — генератор тона;
- `MUSTUHR.ASM` — часы со светодиодной индикацией;
- `PWM1.ASM` — использование широтно-импульсной модуляции.

Анна
и Манфред Кёниг

Полное руководство по PIC микроконтроллерам



На основании своего многолетнего опыта разработки проектов с использованием микроконтроллеров PIC, авторы рассматривают данную тему с точки зрения конструкторов-практиков. Затрагиваются все вопросы, связанные с периферийными устройствами, для взаимодействия с которыми, собственно, и предназначены микроконтроллеры. В частности, описаны аналоговые периферийные модули, инструментальная среда MPLAB, а также отладчик и демонстрационные платы.

Эта книга рассчитана на тех, кто уже имеет опыт работы с микроконтроллерами PIC и делает основное ударение на новых разработках последних лет. К ним, в первую очередь, относится серия микроконтроллеров PIC18, которая, благодаря своему 16-разрядному ядру, не только расширяет возможности программирования, но и открывает множество новых технических возможностей. Кроме того, в книге рассказано о многих нововведениях в микроконтроллерах PIC с 14- и 12-разрядным ядром.

Рассмотрены следующие темы:

- Последовательный обмен данными
- PIC18
- Управление питанием
- Новые модели PIC10F и nPIC
- Программирование PIC на ассемблере
- Система разработки MPLAB 6.XX
- Внутрисхемный отладчик и программатор ICD2
- Демонстрационные платы и периферийные модули

На компакт-диске:

- MPLAB
- Программы для PIC
- Исходный код примеров

ISBN 966-8806-21-2
9 789668 806216

МК-ПРЕСС

WWW.MK-PRESS.COM