ОГЛАВЛЕНИЕ

Благодарности 9
Предисловие 10
Глава 1. **Введение** 11
Краткое замечание для пользователей PIC 13
Системы счисления 14
Сложение в двоичной системе 17
Отрицательные числа 17
8-битный RISC FLASH-микроконтроллер? 19
Первые шаги 20
Выбор модели 20
Блок-схема алгоритма 23
Написание программы 24
Ассемблирование 25
Регистры 25
Команды 29
Шаблон программы 30

Глава 2. **Основные операции в AT90S1200 и TINY12** 37
Программа A. Светодиод (LEDon) 37
AVR Studio — трансляция с языка ассемблера 39
Проверка 40
AVR Studio — симуляция 40
Эмуляция 41
Аппаратное обеспечение 42
AVR Studio — программирование 45
Конфигурационные ячейки 46
Программы B и C. Кнопка 47
Семисегментные индикаторы и косвенная адресация 49
Программы D и E. Счетчик 55
Формирование временных интервалов 60
Программа F. Бегущий огонек 63
Формирование временных интервалов без таймера? 69
Счетчик команд и подпрограммы 71
Программа G. Счетчик (версия 3.0) 75
Программа H. Светофор 77
Логические элементы 83
Программа I. Симулятор логических элементов 85
SREG — регистр состояния 91
Сторожевой таймер 91
Спящий режим 93
Остальные команды 94
Программа J. Частотомер 95
Глава 3. **Знакомство с остальными моделями семейства** 111
Глава 4. **Дополнительные возможности** 118
Прерывания 118
Программа K. Измеритель скорости реакции 120
Случайное распределение 123
Аналоговый компаратор 128
Программа L. 4-битный аналого-цифровой преобразователь 129
Аналого-цифровой преобразователь (АЦП) 132
Программа M. Инвертор напряжения 136
EEPROM 140
Таймер/счетчик 1 (16-битный) 142
Функция захвата 143
Функция сравнения 146
Главная программа N. Музыкальный автомат 146
Глава 5. **Продвинутые возможности** 152
ШИМ — широтно-импульсная модуляция 152
UART 154
Программа O. Конвертер клавиатуры 160

Последовательный интерфейс SPI 163
Нестандартный Таймер 1 модели Tiny15 167
Сокращение объема кода 170
Обзор семейства Mega 171
Заключительная программа P. Робот, управляемый компьютером 172
Заключение 178
Приложение A. **Основные параметры некоторых моделей AVR** 180
Приложение B. **Цоколевка некоторых моделей AVR** 181
Приложение C. **Обзор системы команд** 182
Приложение D. **Справочник команд** 186
Приложение E. **Таблица векторов прерываний** 195
Приложение F. **Преобразование шестнадцатеричных чисел** 197
Приложение G. **Таблица кодов символов ASCII** 198
Приложение H. **Если ничего не получается, прочтите это** 199
Приложение I. **Контактная информация и дополнительная литература** 200
Приложение J. **Полные тексты учебных программ** 201
Ответы к упражнениям 244
Предметный указатель 265
Предметный указатель 1001

Посвящается Таре

БЛАГОДАРНОСТИ

Когда Роберт Жарнек познакомил меня с микроконтроллерами AVR, я очень быстро осознал их преимущества перед прочими микроконтроллерами. Единственным недостатком, впрочем, весьма относительным, была их неизвестность по сравнению, например, с микроконтроллерами PIC фирмы Microchip. Я прекрасно понимал, что быстрое распространение микроконтроллеров AVR всего лишь вопрос времени, и поэтому написал книгу, которую можно рассматривать как базовое руководство по их использованию. Эта книга предназначена для тех, кто совершенно незнаком с микроконтроллерами или имеет о них только смутное представление.

Я хотел бы воспользоваться возможностью и поблагодарить всех, кто помогал мне в создании этой книги. Английское отделение компании Atmel любезно предоставило мне образцы своего оборудования, однако я вас уверяю — при написании книги я оставался беспристрастным и объективным! Я очень хочу поблагодарить Мэта Вебба за его квалифицированную и тщательную вычитку, в результате которой на страницах появлялась целая куча надписей «Что это?». Несмотря на то что у него было множество других, более полезных дел, вроде сдачи выпускных экзаменов, он умудрялся найти время для тщательного просмотра моей рукописи. Также я хочу выразить свою благодарность Ричарду Джорджу за предложенные им примеры программ и общие советы. Я благодарю Мэта Гаррисона за помощь в подготовке иллюстраций — впоследствии он начал учиться по этому направлению в Королевском художественном колледже. В заключение я должен поблагодарить Макса Хоси за его огромное великодушие, поддержку и консультации, а также руководство кафедры электронной техники колледжа Рэдли, Абингдон за предоставленную возможность работать в их великолепно оборудованной лаборатории.

Джон Мортон

ПРЕДИСЛОВИЕ

Примите мои поздравления! Раз вы читаете эту книгу, значит, вас заинтересовало одно из наиболее производительных и универсальных семейств 8-битных микроконтроллеров в мире — семейство AVR. Прочитав книгу, вы получите общее представление обо всех микроконтроллерах семейства и узнаете, каким образом с их помощью можно упростить разработку своих устройств, а также создавать более сложные изделия.

Микроконтроллеры AVR, как и все другие, позволяют создавать нестандартные и вместе с тем достаточно гибкие решения. Однако микроконтроллеры AVR являются при этом эффективными, быстродействующими и простыми в использовании, благодаря чему идеально подходят для разработчиков электронных устройств. Сначала мы познакомимся с основными принципами программирования микроконтроллеров (в частности, с различными системами счисления) и подробно рассмотрим основные этапы создания программ. После этого вы приступите к изучению собственно микроконтроллеров AVR, причем все рассматриваемые вопросы будут сопровождаться примерами в виде реально работающих программ. Среди этих программ, в частности, имитатор светофора, музыкальный автомат, частотомер и даже робот, управляемый персональным компьютером.

На первых порах мы в основном будем рассматривать готовые учебные программы. Однако по мере прочтения книги объем кода, самостоятельно написанного вами при выполнении упражнений, будет постоянно увеличиваться. Эти упражнения встречаются на протяжении всей книги, а ответы к ним приведены в самом конце. В приложениях собраны основные данные, относящиеся к наиболее популярным микроконтроллерам AVR, что позволяет быстро найти нужную информацию, не перерывая кучу документации.

Короче говоря, в этой книге используется активная методика обучения программированию микроконтроллеров AVR. Кроме того, книга будет полезным источником информации для всех программистов, работающих с этими микроконтроллерами.

Джон Мортон

Глава 1. **ВВЕДЕНИЕ**

Микроконтроллеры AVR — это одни из самых быстродействующих микроконтроллеров в мире. Лично я представляю себе микроконтроллер в виде бесполезного куска кремния, обладающего тем не менее огромным потенциалом. Пока в нем нет программы, он ничего не будет делать, однако при ее наличии он сможет выполнять практически любые функции. Достаточно большая принципиальная схема в ваших руках может превратиться в обычную программу, уменьшив таким образом целое устройство до одной единственной микросхемы. Микроконтроллеры ликвидируют разрыв между аппаратным и программным обеспечением — они выполняют программу как обычный компьютер, являясь в то же время дискретными элементами, которые могут взаимодействовать с другими компонентами схемы. За несколько лет микроконтроллеры стали неотъемлемой частью инструментария радиоинженеров и огромного числа радиолюбителей, поскольку они великолепно подходят для экспериментирования, мелкосерийного производства и реализации проектов, требующих определенной гибкости выполняемых функций.

Этапы разработки программного обеспечения микроконтроллеров AVR приведены на **Рис. 1.1**.

Среди микроконтроллеров AVR имеется огромное количество различных моделей, начиная от небольших устройств в 8-выводных корпусах (семейство Tiny) и заканчивая микросхемами в 40-выводных корпусах (семейство Mega) [1]. Однако самое потрясающее заключается в том, что можно спокойно писать программу для одной модели, а затем передумать и переделать эту программу под другую модель микроконтроллера, внеся всего лишь незначительные изменения. Более того, изучая один из микроконтроллеров AVR, вы научитесь работать со всеми моделями семейства. Разумеется, каждый из микроконтроллеров имеет свои особенности, однако в основе всех моделей лежит общее ядро.

---

[1] В настоящее время наиболее развитые микроконтроллеры AVR выпускаются в 64-выводных корпусах. — *Примеч. пер.*

Рис. 1.1. Этапы разработки программного обеспечения микроконтроллеров AVR

Вообще говоря, программирование микроконтроллеров AVR заключается в различных манипуляциях числами. Соответственно, задача программирования состоит в том, чтобы заставить микроконтроллер выполнять поставленную задачу путем простых перемещений чисел и осуществления операций над ними. Существует ограниченный набор операций, которые можно выполнять над числами, — эти операции называются *командами*. В программах используются как простые команды (общего назначения), так и более сложные, выполняющие различные специфические функции. Микроконтроллер будет последовательно перебирать эти команды, выполняя миллионы их каждую секунду (это зависит от частоты подключенного к нему генератора), и, таким образом, выполнять поставленную задачу. В микроконтроллерах AVR числа можно:

1. **Принимать** с входов (например, используя входной «порт»).
2. **Сохранять** в определенных ячейках микросхемы.
3. **Обрабатывать** (например, складывать, вычитать, умножать и т.п.).
4. **Передавать** через выходы (например, используя выходной «порт»).

Вот, в принципе, и все, что касается программирования — вы, наверное, уже думаете: «Класс!» К счастью, в микроконтроллерах AVR имеется много других полезных функций, которые сильно облегчают нам жизнь. Сюда относятся различные модули, такие как встроенные таймеры, последовательные интерфейсы, аналоговые компараторы, а также куча так на-

зываемых «флагов», благодаря которым мы можем определить, произошло ли какое-либо определенное событие или нет.

Мы начнем обучение с рассмотрения основных концепций, общих для всех микроконтроллеров, и сразу же после этого приступим к изучению нескольких учебных проектов на микроконтроллерах AT90S1200 (которые будем для краткости называть 1200) и Tiny. Затем мы познакомимся с более сложными операциями, воспользовавшись для этого более развитыми моделями (такими, как AT90S2313). В заключение мы изучим наиболее продвинутые возможности микроконтроллеров AVR и выполним заключительный проект на базе микроконтроллера 2313. Большинство рассматриваемых нами проектов можно легко адаптировать под любую модель AVR, поэтому вам совершенно не требуется бежать в магазин и скупать все имеющиеся там микроконтроллеры.

Краткое замечание для пользователей PIC

Я полагаю, что многие читатели уже знакомы с популярными микроконтроллерами PIC фирмы Microchip. Поэтому я вкратце упомяну о преимуществах микроконтроллеров AVR по сравнению с PIC. Тем, кто не имеет о микроконтроллерах PIC никакого понятия, не стоит особо беспокоиться, если что-то окажется непонятным, — чуть позже вы все поймете!

Прежде всего, микроконтроллеры AVR имеют более совершенную архитектуру и могут выполнять команды в каждом такте (в отличие от PIC, которым для выполнения команды требуется четыре такта). Поэтому при той же тактовой частоте микроконтроллеры AVR работают в 4 раза быстрее. Кроме того, они имеют 32 рабочих регистра (в отличие от одного единственного, имеющегося в PIC) и почти в 3 раза больше команд. Благодаря этому программы для AVR практически всегда будут короче аналогичных программ для PIC. Однако, несмотря на то что в документации указывается от 90 до 120 команд (в зависимости от модели), многие из них дублируют друг друга, и, по моим подсчетам, из всех команд действительно уникальными является не более 50.

А вот к так называемым регистрам специальных функций микроконтроллеров PIC (которые в AVR называются регистрами ввода/вывода) разрешен прямой доступ (можно писать непосредственно в порты), что в микроконтроллерах AVR не допускается. Однако это не такой уж большой недостаток, и в целом программы для AVR являются более эффективными. Все микроконтроллеры AVR имеют FLASH-память программ, что позволяет осуществлять их многократное перепрограммирование. И наконец, в связи с тем, что различные модели микроконтроллеров PIC разрабатывались на протяжении многих лет, у них имеется ряд досадных проблем с совместимостью, которых в микроконтроллерах AVR до сих пор удавалось избегать.

Системы счисления

Теперь пришло время познакомиться с различными системами счисления, используемыми при программировании микроконтроллеров AVR: двоичной, десятичной и шестнадцатеричной. Двоичные числа являются числами с *основанием 2* (т.е. каждая цифра может принимать только два значения: 0 или 1) в отличие от десятичных чисел, имеющих *основание 10*, с десятью различными цифрами (от 0 до 9). Соответственно, числа в шестнадцатеричной системе имеют *основание 16* и представлены 16 различными цифрами (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E и F). В **Табл. 1.1** приведен пример счета в различных системах счисления.

Таблица 1.1. **Пример счета в различных системах счисления**

Двоичная (8 разрядов)	Десятичная (3 разряда)	Шестнадцатеричная (2 разряда)
00000000	000	00
00000001	001	01
00000010	002	02
00000011	003	03
00000100	004	04
00000101	005	05
00000110	006	06
00000111	007	07
00001000	008	08
00001001	009	09
00001010	010	0A
00001011	011	0B
00001100	012	0C
00001101	013	0D
00001110	014	0E
00001111	015	0F
00010000	016	10
00010001	017	11
и т.д.		

Двоичный разряд (или *бит*), расположенный справа, называется младшим значащим разрядом (*МЗР*) или младшим значащим битом (Least Significant Bit — *LSB*), а также битом 0 (почему нумерация начинается с 0, а не с 1, станет ясно немного позже). Нулевой бит показывает количество

«единиц» в числе. Единица равна 2^0. Бит, расположенный левее (бит 1), показывает количество «двоек», следующий бит (бит 2) показывает количество «четверок» и т.д. Отметим, что $2 = 2^1$, а $4 = 2^2$, т.е. номер бита соответствует степени двойки, представляемой этим битом. Помните, что нумерация битов ведется справа налево (об этом очень часто забывают!). Совокупность 8 битов называется байтом. Самый старший бит двоичного слова (например, 7-й бит байта) называется старшим значащим разрядом (СЗР) или старшим значащим битом (Most Significant Bit — *MSB*).

Таким образом, чтобы преобразовать десятичное число в двоичное, необходимо найти наибольшую степень двойки, которая будет меньше этого числа, вычесть и многократно повторить указанные вычисления.

ПРИМЕР 1.1. Найдем двоичный эквивалент десятичного числа 83.

Наибольшая степень двойки, меньше 83, равна $64 = 2^6$. Бит 6 = **1**.

Разность 83 – 64 = 19.	32 > 19,	поэтому бит 5 = **0**,
	16 < 19,	поэтому бит 4 = **1**.
Разность 19 – 16 = 3.	8 > 3,	поэтому бит 3 = **0**,
	4 > 3,	поэтому бит 2 = **0**,
	2 < 3,	поэтому бит 1 = **1**.
Разность 3 – 2 = 1.	1 = 1,	поэтому бит 0 = **1**.

Таким образом, двоичный эквивалент равен **1010011**.

В то же время существует другой (и более изящный) метод, который, вероятно, покажется вам более легким. Возьмите число, которое вы хотите преобразовать, и разделите его на два. Если остаток равен единице (т.е. число было нечетным), запишите единицу. Затем снова разделите результат на два и так далее, записывая остаток *слева* от предыдущего значения, до тех пор, пока делимое (и остаток) не станет равным единице.

ПРИМЕР 1.2. Найдем двоичный эквивалент десятичного числа 83.

Делим 83 на 2.	Частное 41, остаток **1**.
Делим 41 на 2.	Частное 20, остаток **1**.
Делим 20 на 2.	Частное 10, остаток **0**.
Делим 10 на 2.	Частное 5, остаток **0**.
Делим 5 на 2.	Частное 2, остаток **1**.
Делим 2 на 2.	Частное 1, остаток **0**.
Делим 1 на 2.	Частное 0, остаток **1**.

Таким образом, двоичный эквивалент равен **1010011**.

УПРАЖНЕНИЕ 1.1. Найдите двоичный эквивалент десятичного числа 199.

УПРАЖНЕНИЕ 1.2. Найдите двоичный эквивалент десятичного числа 170.

Аналогично двоичным числам разряд 0 шестнадцатеричного числа показывает количество единиц ($16^0 = 1$), разряд 1 — количество чисел 16 ($16^1 = 16$) и т.д. Чтобы преобразовать десятичное число в шестнадцатеричное (вместо этого слова часто используют сокращение «hex»), следует определить, сколько в числе содержится единиц и сколько чисел 16.

ПРИМЕР 1.3. Преобразуем десятичное число 59 в шестнадцатеричное. В числе 59 содержится три числа 16, поэтому 1-й разряд равен 3. Разность 59 – 48 = 11; число 11 соответствует шестнадцатеричному B, поэтому 0-й разряд равен B. Следовательно, искомое число равно **3B**.

УПРАЖНЕНИЕ 1.3. Найдите шестнадцатеричный эквивалент десятичного числа 199.

УПРАЖНЕНИЕ 1.4. Найдите шестнадцатеричный эквивалент десятичного числа 170.

Одной из полезных особенностей шестнадцатеричной системы, которую вы могли заметить при выполнении Упражнения 1.4, является очень простое преобразование двоичных чисел в шестнадцатеричные. Если разбить двоичное число на 4-битные группы (называемые *полубайтами*, или *тетрадами*), то каждая такая группа будет соответствовать одному шестнадцатеричному разряду.

ПРИМЕР 1.4. Преобразуем число 01101001 в шестнадцатеричную систему счисления. Делим число на полубайты: 0110 и 1001. Нетрудно заметить, что 0110 соответствует 4 + 2 = 6, а 1001 соответствует 8 + 1 = 9. Таким образом, указанное 8-битное число равно 69 в шестнадцатеричной системе. Очевидно, что это преобразование выполнить гораздо проще, чем в случае десятичных чисел, поэтому при программировании шестнадцатеричные числа используются намного чаще.

УПРАЖНЕНИЕ 1.5. Преобразуйте 11100111 в шестнадцатеричное число.

Сложение в двоичной системе

Сложение двоичных чисел выполняется абсолютно по тем же правилам, что и десятичных. Посмотрим различные комбинации битов.

0 + 0 = 0	нет переноса
1 + 0 = 1	нет переноса
1 + 1 = 0	перенос 1
1 + 0 + 0 = 1	нет переноса
1 + 1 + 0 = 0	перенос 1
1 + 1 + 1 = 1	перенос 1

ПРИМЕР 1.5. 4 + 7 = 11

```
  1
 1100
+0111
 1011 = 11 в десятичной системе
```

УПРАЖНЕНИЕ 1.6. Вычислите 01011010 + 00001111, используя двоичное сложение.

Отрицательные числа

Мы разобрались с вами, как преобразовывать положительные десятичные числа в двоичные, но как сделать то же самое с отрицательными? Прежде всего, необходимо выделить один бит для хранения знака, в результате чего 4-битное число сможет принимать значения от –7 до +8. Вообще говоря, имеется несколько способов представления отрицательных чисел, однако наиболее распространенным является представление отрицательных чисел в *дополнительном коде* (two's complement). Чтобы из положительного числа получить отрицательное число в дополнительном коде, необходимо инвертировать все биты исходного числа, а затем прибавить к получившемуся числу единицу.

ПРИМЕР 1.6. 0111 = 7
Инвертируем все биты: 1000
Прибавляем единицу: 1001
1001 = –7

ПРИМЕР 1.7. 1000 = 8
Инвертируем: 0111
Прибавляем единицу: 1000
1000 = –8 = +8 НЕОДНОЗНАЧНОСТЬ!

Из Примера 1.7 видно, что мы не можем использовать число –8, поскольку его двоичное представление совпадает с представлением числа +8. Эта асимметрия является прискорбным недостатком представления чисел в дополнительном коде, однако с ним приходится мириться, поскольку этот недостаток является наименьшим по сравнению с недостатками других способов представления двоичных чисел. Давайте попробуем сложить –2 и + 7.

ПРИМЕР 1.8. 2 = 0010, соответственно –2 = 1110

$$\begin{array}{rl} & 1110 = -2 \\ + & \underline{0111 = 7} \\ & 0101 = 5 \end{array}$$

Что и ожидалось!

УПРАЖНЕНИЕ 1.7. Представьте число –40 в 8-битном дополнительном коде и докажите, что результат операции –40 + 50 соответствует ожидаемому (–10).

Благодаря такому представлению чисел нам достаточно просто проверить старший значащий бит (MSB), чтобы определить, отрицательное перед нами число или положительное. Единица в старшем бите соответствует отрицательному числу, нуль — положительному. Однако применительно к результатам сложения или вычитания больших положительных или отрицательных чисел это утверждение может быть неверным.

ПРИМЕР 1.9. 69 + 120 = ...

$$\begin{array}{rl} & \textit{1} \\ & 11000101 = +69 \\ + & \underline{01111000} = +120 \\ & 10111101 = +189 \text{ или } -67 \end{array}$$

Другими словами, при использовании чисел в дополнительном коде мы должны интерпретировать результат как отрицательный (имеющий 1 в старшем бите). Поэтому существует проверка на переполнение дополнительного кода, которую мы можем использовать для определения *действительного* знака результата. Переполнение дополнительного кода происходит, когда:

- MSB обоих слагаемых равны 0, а MSB результата равен 1.
- MSB обоих слагаемых равны 1, а MSB результата равен 0.

Соответственно, действительный знак числа определяется результатом проверки на переполнение дополнительного кода и значением MSB результата операции (см. **Табл. 1.2**).

Таблица 1.2. **Определение действительного знака результата**

Переполнение дополнительного кода	MSB результата	Знак
Нет	0	+
Нет	1	–
Есть	0	+
Есть	1	–

При сложении чисел в Примере 1.10 произошло переполнение дополнительного кода, а MSB результата равен 1, поэтому результат положительный (+189), как и ожидалось. Думаю, вы будете рады узнать, что большинство описанных действий поддерживается микроконтроллерами AVR автоматически.

Другим способом представления отрицательных чисел является *обратный код числа* (one's complement), который получается в результате простого инвертирования всех его битов и занесения единицы в знаковый бит.

8-битный RISC FLASH-микроконтроллер?

Мы называем AVR *8-битным микроконтроллером*. Это означает, что он оперирует 8-битными числами. Двоичное число 11111111 является наибольшим 8-битным числом и равно десятичному 255 и шестнадцатеричному FF (проверьте!). Для указания конкретной системы счисления в программах используются различные способы записи (ведь десятичное число 11111111 очень сильно отличается от двоичного числа 11111111!). Двоичные числа записываются в виде 0b00101000 (т.е. **0b**...). Десятичная система счисления используется по умолчанию, а шестнадцатеричные числа начинаются с символов **0x** или знака доллара (0x3A, или $3A). Следовательно,

0b00101011 равно 43, которое равно 0x2B.

При работе с входами и выходами микроконтроллеров AVR обычно используют двоичную систему счисления, при этом каждый входной или выходной контакт соответствует конкретному биту. Бит, установленный в 1, соответствует состоянию, называемому *логическая единица*. Это означает, что напряжение на выводе микроконтроллера равно напряжению питания (например, +5 В). Бит, сброшенный в 0, соответствует состоянию *логического нуля*, или 0 В. Для входных сигналов порогом между состояниями логического 0 и логической 1 является половина напряжения питания (например, +2.5 В).

Также вы не раз услышите, что микроконтроллеры AVR называют *RISC-микроконтроллерами*. Это означает, что они принадлежат к классу

микроконтроллеров с сокращенным набором команд (**R**educed **I**nstruction **S**et **C**omputer). Такая архитектура немного усложняет жизнь программистам (нам с вами), однако микросхема при этом получается более простой и более быстродействующей.

Иногда микроконтроллеры AVR называют *FLASH-микроконтроллерами* (читается как «флэш»). Это название отражает тот факт, что написанная вами программа хранится в *FLASH-памяти* — памяти, которую можно перезаписывать снова и снова. Соответственно, вы сможете многократно программировать один и тот же кристалл AVR — для радиолюбителей это означает возможность длительного использования одной и той же микросхемы.

Первые шаги

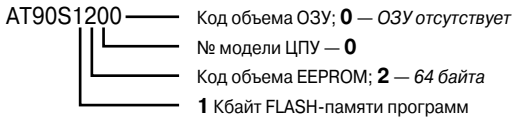
Процесс разработки программ состоит из пяти основных этапов:

1. **Выбор** конкретного микроконтроллера и составление блок-схемы программы.
2. **Написание** программы (с помощью Блокнота, AVR Studio или любой другой подходящей программы).
3. **Ассемблирование** программы (преобразует написанный вами текст в форму, понятную микроконтроллеру).
4. **Симуляция** или **Эмуляция** программы, чтобы убедиться в ее работоспособности (или неработоспособности).
5. **Программирование** AVR. На этом этапе написанное вами заносится в реальный микроконтроллер.

А теперь рассмотрим некоторые из этих этапов более подробно.

Выбор модели

Поскольку в семейство AVR входит большое число различных моделей микроконтроллеров, необходимо хорошенько подумать о том, какая из них лучше всего подойдет для вашего устройства. Некоторую информацию о микроконтроллере можно получить из его обозначения:



Коды объема памяти:

Коды	0	1	2	3	4	5	6	7	8	9	A	B
Объем [байт]	0	32	64	128	256	512	1K	2K	4K	8K	16K	32K

Значения использованных терминов могут быть вам незнакомы, однако не волнуйтесь — мы скоро их рассмотрим. Следует заметить, что микроконтроллеры семейств Tiny и Mega имеют немного другую систему обозначений. Краткие сведения о характеристиках некоторых микроконтроллеров AVR приведены в Приложении А.

УПРАЖНЕНИЕ 1.8. Определите объем различных областей памяти микроконтроллера AT90S8515.

Одним из наиболее важных параметров микроконтроллеров, не нашедший, к сожалению, отражения в обозначении модели, является число входов и выходов. Модель 1200 имеет 15 контактов ввода/вывода (т.е. 15 выводов, которые могут использоваться как входы *или* выходы), а модель 8515 — целых 32 контакта ввода/вывода!

ПРИМЕР 1.10. Необходимо разработать устройство, которое будет считать количество нажатий на кнопку и отображать это число на одном семисегментном индикаторе (при достижении значения 9 он будет сбрасываться).

1. Для управления семисегментным индикатором требуется **семь** выходов.
2. Для кнопки требуется **один** вход.

Таким образом, для такого устройства потребуется в общей сложности 8 контактов ввода/вывода. В данном случае вполне можно использовать 1200, поскольку это одна из самых простых моделей, имеющая достаточное количество выводов.

При работе с большим числом входов и выходов часто используется полезный прием, называемый *стробированием*. Он особенно удобен при управлении несколькими семисегментными индикаторами или при необходимости контролировать большое количество кнопок. Лучше всего продемонстрировать этот прием на примере.

ПРИМЕР 1.11. Необходимо разработать счетчик, который прибавляет число от 1 до 9 к текущему двухзначному значению. Соответственно, в устройстве будет 9 кнопок и 2 семисегментных индикатора. На первый взгляд, для решения поставленной задачи потребуется достаточно много входов и выходов:

1. Для каждого семисегментного индикатора требуется семь выходов, итого **14**.
2. Для каждой кнопки требуется один вход, итого **9**.

Таким образом, в общей сложности требуется 23 вывода, что влечет за собой необходимость использования «большого» микроконтроллера, такого как 8515 (имеющего 32 контакта ввода/вывода); однако на самом деле использовать такой «большой» микроконтроллер нет никакой необходимости, поскольку требуемое число выводов можно значительно уменьшить.

При использовании стробирования состояния всех кнопок можно будет прочитать с помощью шести выводов, а для управления двумя семисегментными индикаторами потребуется всего девять выводов. Итого получается 15 контактов ввода/вывода, имеющихся в микроконтроллере 1200. Соответствующая схема приведена на **Рис. 1.2**.

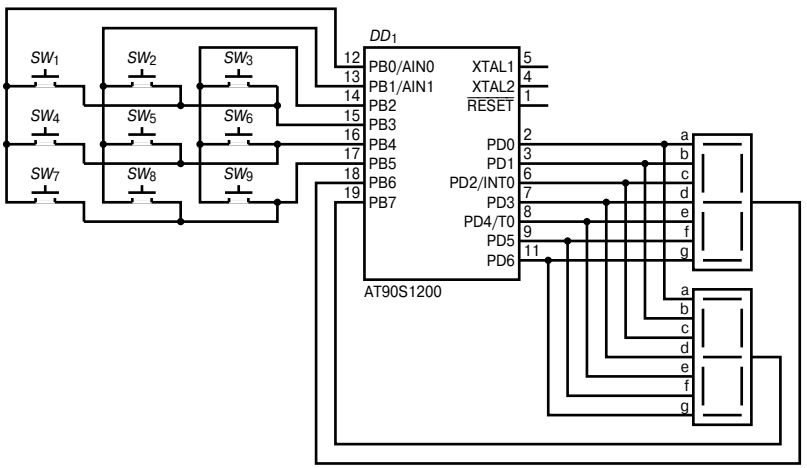


Рис. 1.2. Схема стробирования

При подаче на вывод PB0 лог. 1 (+5 В), а на выводы PB1 и PB2 — лог. 1 (0 В) разрешается обработка кнопок 1, 4 и 7. После этого состояние каждой из них можно узнать, проверив напряжение на одном из выводов PB3...PB5. Таким образом, подавая последовательно на выводы PB0...PB2 лог. 1, можно проверить состояние всех кнопок. Чтобы определить, какое количество выводов потребуется для обслуживания массива из X кнопок, найдите пару сомножителей числа X, имеющих наименьшую сумму (например, для числа 24 сомножителями с наименьшей суммой являются числа 6 и 4, поэтому для контроля 24 кнопок потребуется 6 + 4 = 10 контактов ввода/вывода). Лучше сделать меньшее число выводов (конечно, если эти числа не равны) выходами, а большее число — входами. В этом случае опрос всех строк матрицы кнопок займет меньше времени.

Стробирование семисегментных индикаторов заключается в кратковременном отображении числа на одном индикаторе и последующем выключении этого индикатора на время отображения другого числа на другом индикаторе. На выводы PD0...PD6 выдается код числа для обоих индикаторов, а подавая лог. 1 на вывод PB6 или PB7, вы можете включать соответствующий индикатор. Хотя в действительности индикаторы мерцают с большой частотой, кажется, что они светятся непрерывно. Требования к программированию подобных узлов мы рассмотрим позже.

УПРАЖНЕНИЕ 1.9. С помощью Приложения А определите, какую модель микроконтроллера AVR можно использовать для реализации 4-разрядного калькулятора с кнопками для цифр от 0 до 9 и пяти операций: +, –, ×, ÷ и =.

Блок-схема алгоритма

После того как вы определили требуемое количество контактов ввода/вывода и, таким образом, выбрали конкретный микроконтроллер, можно приступать к следующему этапу, который заключается в создании блок-схемы программы. В принципе, на этом этапе формируется основа программы, а написать программу, имея перед собой блок-схему, гораздо легче, чем с нуля.

Блок-схема должна отображать основные этапы функционирования микроконтроллера, а также прояснять структуру программы. Представьте, что ваша программа является растительным лабиринтом. В этом случае блок-схема будет представлять собой грубую карту, обозначающую основные участки лабиринта. При создании блок-схемы вы должны иметь в виду, что лабиринт не может выходить к обрыву (т.е. программа не может просто взять и закончиться), так как в противном случае AVR перешагнет через край и разобьется. Вместо этого AVR вынужден постоянно бродить по лабиринту (хотя вы можете усыпить его!). Простой пример блок-схемы программы приведен на **Рис. 1.3**.

ПРИМЕР 1.12. Блок-схема программы, которая включает светодиод (СИД), если нажата кнопка.

Блок инициализации представляет некоторые действия, которые необходимо выполнять в начале каждой программы для настройки различных функций. Эти действия мы рассмотрим чуть позже.

Прямоугольники со скругленными углами используются для обозначения начального и завершающего блоков программы, а ромбы используются для обозначения условий. Условные переходы (ромбы на блок-схеме) означают: «*если* что-то произошло, *то* переходим туда-то».