(Продолжение. Начало — № 4—7/2001)

Микроконтроллеры? Это же просто!

Сопряжение с последовательными АЦП

Не знаю, утомились ли вы читать про ассемблер, а я писать о нем слегка утомился. Поэтому предлагаю сделать некоторую передышку и на время вернуться к рассмотрению "железа". В этом разделе мы познакомимся с одним из типичнейших представителей микросхем, обменивающихся информацией в последовательном формате – АЦП ADS7816 фирмы Burr-Brown.

Микросхема выпускается в 8-выводном корпусе. Схема ее соединения с МК, а также ее цоколевка приведены на рис. 8.

Две ножки микросхемы являются аналоговыми входами (вход –IN рекомендуется соединить с общим проводом), на вход Vref подается опорное напряжение, Vcc и GND — соответственно питание и "земля". Для обмена с МК используются три оставшиеся ножки АЦП. Сигнал старта преобразования МК подает на вход CS, по входу DCLOCK он тактирует АЦП, а с выхода Dout принимает результат преобразования, бит за битом. Временные диаграммы сигналов на ножках CS, DCLOCK и Dout приведены на рис. 9.

Как видите, алгоритм работы с ADS7816 несложен. МК при включении должен установить на выводах CS и Dout единичные уровни сигналов, а на выводе DCLOCK – нулевой. Запуск преобразования осуществляется установкой нулевого уровня на ножке CS. После этого МК должен сформировать на DCLOCK три положительных импульса. По спаду последнего из них на выходе Dout появится старший бит результата преобразования (DB11). Считав его, МК должен сформировать на DCLOCK следующий положительный импульс. По спаду его на выходе Dout появится следующий бит результата преобразования (DB10). Считав его, МК снова формирует импульс на DCLOCK, по спаду которого на Dout появится бит DB9, и т. д., вплоть до DB0. Считав последний, МК должен оставить DCLOCK нулевым, а CS вернуть в единицу. На этом цикл запуска преобразования и считывания информации завершается.

Программа, реализующая этот алгоритм, содержится в файле ser_adc.a51 и приведена ниже.

```
;       ПРОГРАММА ЧТЕНИЯ АЦП ADS7816,
;       РАБОТАЕМ С ПОРТАМИ P1 И P3, CS = P3.7,
;       DCLOCK = P3.6 , DOUT = P3.5.
;                                       *****
;
R7              .EQU   7 ;АДРЕСА РЕГИСТРОВ R0-R7
R6              .EQU   6
R5              .EQU   5
R4              .EQU   4
R3              .EQU   3
R2              .EQU   2
R1              .EQU   1
R0              .EQU   0
ACC             .EQU   0E0H     ;АДРЕС АККУМУЛЯТОРА
B               .EQU   0F0H     ;АДРЕС РЕГИСТРА B
PSW             .EQU   0D0H     ;АДРЕС РЕГИСТРА (СЛОВА) СОСТОЯНИЯ
SP              .EQU   81H      ;АДРЕС УКАЗАТЕЛЯ СТЕКА
DPL             .EQU   82H      ;АДРЕС МЛАДШЕЙ ПОЛОВИНЫ DPTR
DPH             .EQU   83H      ;АДРЕС СТАРШЕЙ ПОЛОВИНЫ DPTR
P0              .EQU   80H      ;АДРЕС РЕГИСТРА ПОРТА P0
P1              .EQU   90H      ;АДРЕС РЕГИСТРА ПОРТА P1
P2              .EQU   0A0H     ;АДРЕС РЕГИСТРА ПОРТА P2
P3              .EQU   0B0H     ;АДРЕС РЕГИСТРА ПОРТА P3
B.0             .EQU   0F0H     ;АДРЕСА ОТДЕЛЬНЫХ БИТОВ РЕГИСТ-
РА B
B.1             .EQU   0F1H
B.2             .EQU   0F2H
B.3             .EQU   0F3H
B.4             .EQU   0F4H
B.5             .EQU   0F5H
B.6             .EQU   0F6H
B.7             .EQU   0F7H
ACC.0           .EQU   0E0H     ;АДРЕСА ОТДЕЛЬНЫХ БИТОВ
АККУМУЛЯТОРА
ACC.1           .EQU   0E1H
ACC.2           .EQU   0E2H
ACC.3           .EQU   0E3H
ACC.4           .EQU   0E4H
ACC.5           .EQU   0E5H
ACC.6           .EQU   0E6H
ACC.7           .EQU   0E7H
PSW.0           .EQU   0D0H     ;АДРЕСА ОТДЕЛЬНЫХ БИТОВ
РЕГИСТРА PSW
PSW.1           .EQU   0D1H
PSW.2           .EQU   0D2H
PSW.3           .EQU   0D3H
PSW.4           .EQU   0D4H
PSW.5           .EQU   0D5H
PSW.6           .EQU   0D6H
PSW.7           .EQU   0D7H
P0.0        .EQU   080H     ;АДРЕСА ОТДЕЛЬНЫХ ЛИНИЙ ПОРТА P0
P0.1        .EQU   081H
P0.2        .EQU   082H
P0.3        .EQU   083H
P0.4        .EQU   084H
P0.5        .EQU   085H
P0.6        .EQU   086H
P0.7        .EQU   087H
P1.0        .EQU   090H     ;АДРЕСА ОТДЕЛЬНЫХ ЛИНИЙ ПОРТА P1
P1.1        .EQU   091H
P1.2        .EQU   092H
P1.3        .EQU   093H
P1.4        .EQU   094H
P1.5        .EQU   095H
P1.6        .EQU   096H
P1.7        .EQU   097H
P2.0        .EQU   0A0H     ;АДРЕСА ОТДЕЛЬНЫХ ЛИНИЙ ПОРТА P2
P2.1        .EQU   0A1H
P2.2        .EQU   0A2H
P2.3        .EQU   0A3H
P2.4        .EQU   0A4H
P2.5        .EQU   0A5H
P2.6        .EQU   0A6H
P2.7        .EQU   0A7H
P3.0        .EQU   0B0H     ;АДРЕСА ОТДЕЛЬНЫХ ЛИНИЙ ПОРТА P3
P3.1        .EQU   0B1H
P3.2        .EQU   0B2H
P3.3        .EQU   0B3H
P3.4        .EQU   0B4H
P3.5        .EQU   0B5H
P3.6        .EQU   0B6H
P3.7        .EQU   0B7H
;
CS          .EQU    P3.7
DCLOCK      .EQU    P3.6
DOUT                .EQU    P3.5
;
.ORG 0              ;НИЖЕСЛЕДУЮЩАЯ КОМАНДА С АДРЕСА 0
;
LJMP START          ;НА КОМАНДУ ПОСЛЕ МЕТКИ START
;
.ORG 100H           ;НИЖЕСЛЕДУЮЩАЯ КОМАНДА С АДРЕСА 100H
;
START:
MOV P0,#11111111B           ;НАЧАЛЬНАЯ УСТАНОВКА
MOV P1,#11111111B
      MOV   P2,#11111111B
MOV   P3,#11111111B
      CLR     DCLOCK                ;УСТАНОВКА DCLOCK В 0
;
L7816:                              ;СОБСТВЕННО ЧТЕНИЕ
;
      CLR   CS                      ;ИМПУЛЬС СТАРТА ПРЕОБРА-
ЗОВАНИЯ
;
      SETB   DCLOCK ;1-Й ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
      SETB   DCLOCK ;2-Й ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
```

```
;
      SETB   DCLOCK ;3–Й ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   B.3,C                    ;DB11 В B.3
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   B.2,C                    ;DB10 В B.2
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   B.1,C                    ;DB9 В B.1
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   B.0,C                    ;DB8 В B.0
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.7,C                  ;DB7 В ACC.7
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.6,C                  ;DB6 В ACC.6
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.5,C                  ;DB5 В ACC.5
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.4,C                  ;DB4 В ACC.4
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.3,C                  ;DB3 В ACC.3
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.2,C                  ;DB2 В ACC.2
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.1,C                  ;DB1 В ACC.1
      SETB   DCLOCK ;ТАКТОВЫЙ ИМПУЛЬС
CLR   DCLOCK
;
MOV   C,DOUT
MOV   ACC.0,C                  ;DB0 В ACC.0
;
      SETB   CS                        ;ЗАВЕРШЕНИЕ СЧИТЫВАНИЯ
;
MOV    R4,A   ;СОХРАНЯЕМ МЛ.И СР. ТЕТРАДЫ В R4
;
MOV    A,B    ;ЧИТАЕМ ИЗ РЕГИСТРА В СТ. ТЕТРАДУ
ANL   A,#00001111B     ;ЗАНУЛЯЕМ СТАРШИЕ 4 БИТА
MOV    R5,A   ;В R5R4 – РЕЗУЛЬТАТ
;
SJMP   L7816              ;ЗАЦИКЛИВАНИЕ
;
.END
```

Почти все, что в программе написано, вам уже знакомо, а именно фрагмент присвоения директивой .EQU адресов основным регистрам МК, их битам и линиям портов, метки, оканчивающиеся двоеточиями, директивы .ORG, команды пересылок, установки и сброса линий портов и т. д. Новым для вас является регистр В, команды пересылки битов типа MOV C,DOUT и команда ANL. Рассмотрим их более подробно. Но перед этим замечу, что тем, кому не по душе наше крайне медленное знакомство с системой команд МК, советую обратиться к литературе. Помимо перечисленной в первой части цикла (“Схемотехника”, № 4, 2001), рекомендую также справочник “Однокристальные микроЭВМ” А. В. Боборыкина и др.

Регистр В иначе называют расширителем аккумулятора. В основном его используют при выполнении команды умножения — перед операцией в нем хранится один из сомножителей (второй — в аккумуляторе), а после выполнения — старший байт произведения. Главное достоинство этого регистра в сравнении с регистрами R0–R7 заключается в том, что мы можем осуществлять обмен между каждым битом этого регистра и уже упоминавшимся битом переноса. Кстати, такое возможно также и с битами аккумулятора, поэтому в программе ser_adc.a51 результат измерения первоначально помещается именно в регистр В и в аккумулятор.

Теперь о командах MOV C,DOUT. При рассмотрении предыдущей программы мы уже знакомились с командами пересылки данных (помните, MOV A,P1 и MOV R4,A). Но те команды пересылали содержимое того или иного порта или регистра целиком, т. е. 8-битные числа (от 00000000B до 11111111B). Одной из замечательных особенностей микроконтроллеров x51 является то, что они допускают также пересылку одного бита из какого-либо разряда порта P0–P3 или регистра ACC либо В в бит переноса и наоборот. Бит переноса обозначают как CY, и он является одним из битов регистра состояния PSW, о котором мы будем говорить при подробном знакомстве с системой команд. Так, уже упомянутая команда MOV C,DOUT перенесет нолик или единичку с линии P3.5 (помните, у нас DOUT .EQU P3.5) в бит переноса. Далее команда MOV B.x, C или MOV A.x,C перенесет этот ноль или единицу в бит B.x (x=0–3) или A.x (x=0–7) соответственно. Таким образом, в рассматриваемой выше программе в процессе чтения на входе Dout (P3.5) МК последовательно возникают биты результата DB11, DB10, DB9, …, DB0, а он считывает их и размещает соответственно в третьем, втором, первом, нулевом битах регистра В, далее в седьмом, шестом и т. д. битах аккумулятора.

Замечу, что описанная реализация программы чтения ADS7816 крайне неизящна с точки зрения правил написания программ — она длинна, в ней много повторяющихся кусков и т. д. Но у нее есть два преимущества в сравнении с тем, что обычно считается изящно написанной программой. Во-первых, она доступна для понимания самых малоподготовленных пользователей, и пока вы не научитесь писать программы лучше, пишите их таким образом, они тоже будут работать. А во-вторых, программы, написанные подобным образом, чаще всего работают быстрее других — микроконтроллер не тратит времени на вызов подпрограмм и т. п. Но об этом как-нибудь позже.

И последнее, с чем мы познакомимся в этом разделе – команда ANL A,#00001111B. Она осуществляет функцию логического И между битами аккумулятора и числа 00001111B. Для тех, кто не помнит — логическое И нуля с нулем или единицей дает ноль, и только логическое И двух единиц дает в результате единицу. Следовательно, в результате выполнения этой команды старшие 4 бита аккумулятора занулятся (какими бы они ни были, в результате операции логического И между ними и нулями в старших четырех битах числа 00001111B выйдут нули), а младшие — останутся без изменений. Подобное использование команды ANL для зануления тех или иных битов аккумулятора очень распространено, поскольку это гораздо проще, чем ставить несколько команд CLR ACC.x (x=0–7). Отмечу, что этой командой мы занулили с седьмого по четвертый биты аккумулятора, третий, второй, первый и нулевой биты которого в этот момент содержали старшую тетраду результата, считанного с АЦП. Сделано это для того, чтобы после завершения чтения АЦП в тех битах регистров, хранящих результат, которые по забывчивости можно было бы трактовать как двенадцатый, тринадцатый, четырнадцатый и пятнадцатый, были бы нули.

Вот и все, что можно сказать о работе с последовательным АЦП. В качестве упражнения рекомендую познакомиться с 14-разрядным последовательным АЦП AD7894 от Analog Devices и с 10-разрядным MAX1243 от Maxim и попытаться адаптиро-

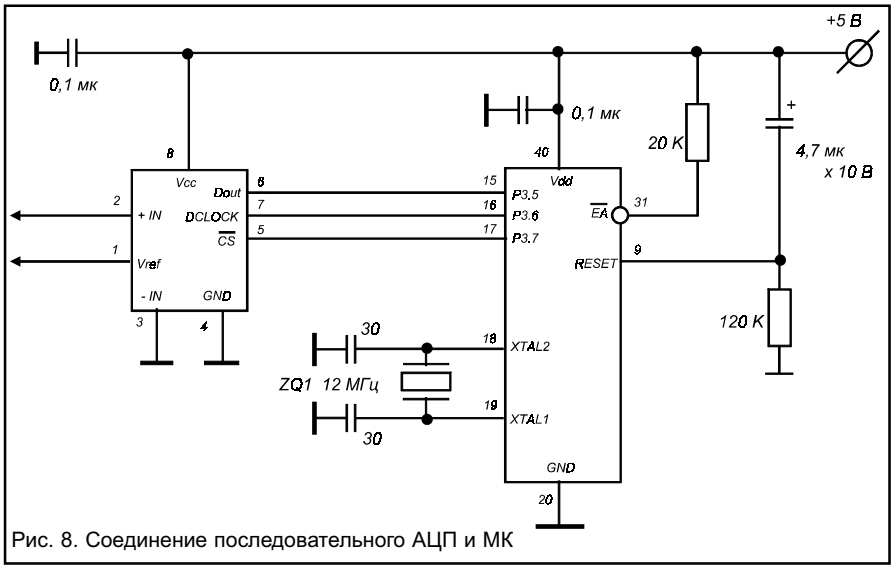

Рис. 8. Соединение последовательного АЦП и МК

вать рассмотренную программу под них — это вам уже вполне по силам.

Краткие выводы

Итак, уважаемые читатели, вы узнали, как сопрягать МК с микросхемами, имеющими интерфейс, ориентированный на микроконтроллеры. Как видите, ничего сложного в этом нет. Вы соединяете ножки требуемого числа линий портов ввода/вывода МК с управляющими и информационными входами и выходами этих микросхем и далее пишите программу, которая реализует заданный в описании на микросхему алгоритм работы с ней. При этом на начальном этапе достаточно крайне ограниченного числа команд из тех 255, которые понимает рассматриваемый нами МК семейства x51. Установка в 0 или в 1 линий ввода/вывода МК осуществляется командами CLR и SETB. Эти же команды используются для формирования импульсов, отдельных фронтов и спадов, запускающих обмен и преобразование данных. Для пересылки же 8-битных данных между портами и регистрами применяют разновидности команды MOV. Также используют эту команду и для пересылки отдельных битов (преимущественно между какой-либо линией ввода/вывода МК и битом переноса CY, а также между битом переноса и битом одного из регистров, в частности аккумулятора или регистра B).

Мы рассмотрели варианты программ для работы с конкретными микросхемами – параллельным АЦП AD7880 и последовательным ADS7816. Они просты для понимания и абсолютно работоспособны, хотя не очень рациональны с точки зрения продвинутых программистов. Пока вы не наберетесь нужного опыта, программы ваши будут с их точки зрения столь же примитивными, но пусть это вас не смущает. На первом этапе важно, чтобы они работали. А изящество в написании программ со временем само придет, опыт – дело наживное.

Также состоялось более близкое знакомство с конкретной версией ассемблера. Теперь вы знаете, что ассемблер переводит (транслирует) текстовый ассемблерный файл, содержащий строки с командами микроконтроллера и директивами ассемблера, в obj- или hex-файлы. Файлы в этих форматах нужны для занесения написанной вами программы в МК. Как правило, программаторы понимают оба этих формата, хотя бывают и исключения. Транслируйте вашу программу в тот формат, который доступен используемому вами программатору. Если же ему доступны оба формата, рекомендую работать с объектным obj-.

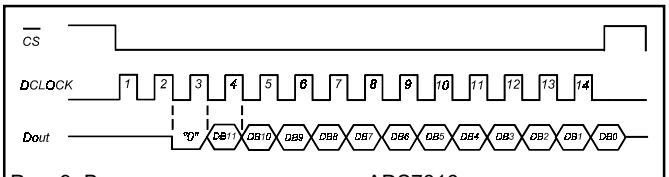

Рис. 9. Временные диаграммы для ADS7816

Управление процессом трансляции осуществляется при помощи директив ассемблера и ключей (указаний), предписывающих ассемблеру при трансляции выполнять определенные действия – формировать файл листинга, включать в него таблицу меток и т. д. Подробнее о том, что он может сделать, и как заставить его выполнить те или функции, можно узнать, прочитав содержимое файла документации (в нашем случае tasm_rus.doc), а также проанализировав командные файлы tasm51b.bat и tasm51h.bat — ключи в них записаны должным образом. На первых порах можете просто ограничиться использованием этих командных файлов без вникания в их содержание.

Наиболее часто используемые директивы ассемблера — .ORG, .EQU и .END. Первая из них предписывает ассемблеру транслировать идущий вслед за ней фрагмент программы (или всю программу) с того адреса, который указан после слова .ORG (.ORG 100H значит — с адреса 100H или 256D).

Следующая директива — .END. Она информирует ассемблер о том, что команда, стоящая перед ней – последняя в этой программе, и что на этом месте нужно завершить трансляцию написанных команд в понятные микроконтроллеру коды. Перед .END, кстати, также, как и перед .ORG, нужно поставить один или несколько пробелов (или нажать Tab).

Очень важной является третья из рассматриваемых директив – .EQU. Она используется для сообщения ассемблеру, что то или иное имя переменной, либо та или иная константа при трансляции должны иметь значения, записанные в строке с именем этой переменной или константы и директивой .EQU (например, CONVST .EQU P3.5). Отметим, что в этой директиве имя переменной (в данном случае CONVST) записывается с самого начала строки, без предшествующих ему пробелов.

При написании программ рекомендуется использовать подобные символические имена везде, где это возможно. Помимо легкости понимания, это дает нам ряд дополнительных удобств. В самом деле, если вы, к примеру, по причине упрощения разводки платы вынуждены будете соединить вход АЦП CONVST не с линией P3.5 МК, а с P3.2, вам всего-навсего лишь будет нужно заменить P3.5 на P3.2 в строке CONVST .EQU P3.5.

При трансляции ассемблер формирует также листинговый файл. Он представляет из себя исходный ассемблерный файл, дополненный следующей информацией: перед каждой командой стоит номер ее строки в ассемблерном тексте, адрес ячейки памяти программ, в которой расположен код операции команды, а после этого адреса — один, два или три байта самой команды. При обнаружении ошибок ассемблер сообщает о них в листинговом файле, причем сообщение об ошибке находится перед или после строки, где эта ошибка была найдена. Далее, в конце программы находится таблица имен и меток, а также содержимое полученного в результате трансляции obj- или hex–файла.

Завершив трансляцию, ассемблер выводит на экран отчет о результате трансляции. Если в этом отчете присутствуют слова tasm: Number of errors = 0 (или аналогичные, если вы используете другой ассемблер), то можете использовать полученный obj- или hex-файл для программирования МК. В противном случае вам необходимо с помощью листингового файла найти ошибку (ошибки), исправить и произвести трансляцию по новой, до тех пор, пока ассемблер не сочтет, что все в порядке.

Александр Фрунзе
alex.fru@mtu-net.ru

Продолжение следует