(Продолжение. Начало – № 4–5/2001)

Микроконтроллеры? Это же просто!

Глава 2. Сопряжение микроконтроллера с программно управляемыми микросхемами

Подобно тому, как невозможно научиться плавать без практических уроков в бассейне, также невозможно освоить работу с микроконтроллерами без разбора практических примеров реализации тех или иных устройств. В настоящей главе мы с вами рассмотрим, как нужно аппаратно сопрягать МК с микросхемами, имеющими, как иногда принято говорить, интерфейс, ориентированный на микроконтроллеры. Также нам предстоит составить программы, которые "оживят" эти схемы. Параллельно с этим мы начнем знакомиться с командами, входящими в эти программы. Естественно, мы ближе познакомимся с конкретной версией программы ассемблер и с его помощью оттранслируем написанные программы. В результате этого вы будете располагать некоторыми конкретными программами, в которых будете понимать все, и базируясь на которых, сможете начать писать самостоятельные программки, пусть для начала и примитивные, то есть у вас появятся какие-то минимальные навыки в написании трансляции программ.

Сопряжение с параллельными АЦП

Одна из микросхем, наиболее часто используемых совместно с микроконтроллером – это АЦП. Ничего удивительного в этом нет – прежде чем как-то обработать и отобразить какой-то результат, его нужно ввести в МК в цифровой форме.

В настоящее время АЦП производят десятки фирм. АЦП различаются по принципу преобразования, быстродействию, разрядности, точностным параметрам, питающим напряжениям, диапазонам входных напряжений, количеству каналов – список этих параметров может быть продолжен еще на десяток строк. Однако из всего их многообразия в данный момент нас интересуют лишь те, которые имеют отношение к связи АЦП с МК. Вот их-то не так уж и много.

Во-первых, с точки зрения интерфейса АЦП делятся на параллельные и последовательные. Первые после преобразования передают микроконтроллеру все биты результата одновременно, каждый по своей индивидуальной линии. Это означает, что с 12-разрядным АЦП МК должен быть связан минимум двенадцатью проводниками (реально – на три–пять больше упомянутого числа за счет сигналов управления).

Последовательные АЦП связаны с микроконтроллером всего тремя–четырьмя проводниками, независимо от их разрядности. Биты результата оцифровки они передают последовательно по одному проводнику (один за другим). Управление передачей микроконтроллер осуществляет по второму проводнику. Третий проводник, как правило, дает АЦП команду на начало преобразования. Ясно, что последовательные АЦП работают медленнее параллельных, но достигнутые ими предельные скорости преобразования и передачи информации достаточно высоки (многим 12- и 14-разрядным АЦП требуется менее 10 мкс на весь цикл преобразования/передачи данных).

АЦП могут содержать некоторые внутренние регистры (ячейки памяти разрядностью от 4 до 24 бит), в которые микроконтроллер должен предварительно занести информацию. К таким АЦП относятся, к примеру, многоканальные преобразователи – микроконтроллер должен сообщить АЦП, какой из его каналов должен преобразовывать информацию. При работе с такими микросхемами МК не только читает информацию, но и записывает ее в АЦП при помощи соответствующих сигналов. Подобные преобразователи, естественно, сложнее простых, не требующих записи в них управляющих слов (заносимую микроконтроллером в подобные микросхемы информацию программисты обычно называют управляющими словами), поэтому на первом этапе мы исключаем подобные сложные микросхемы из нашего рассмотрения и вернемся к ним попозже.

Остальные характеристики – какими сигналами управляется АЦП, какова полярность этих сигналов, каковы привязки к их фронтам и спадам, задержки и т. д. – не столь принципиальны. Эту информацию, имеющуюся в datasheet'е на конкретную микросхему, вы должны держать перед собой при разработке аппаратного сопряжения и программы для МК.

Перейдем теперь к конкретному примеру, в качестве которого я предлагаю сопряжение с МК 12-разрядного параллельного АЦП AD7880 фирмы Analog Devices (рис. 5). Года четыре назад это был чуть ли не единственный АЦП, требовавший всего одно питающее напряжение 5 В и потреблявший при этом относительно небольшой ток (менее 10 мА). Почему я выбрал именно этот АЦП? Когда-то я с ним работал, и программы, которые мы с вами будем анализировать, были тогда отлажены и не содержат ошибок.

Коль скоро АЦП параллельный 12-разрядный, с микроконтроллером он должен быть соединен 12 линиями данных, по которым в последний будут одновременно переданы все 12 бит результата. Добавим, что для организации обмена АЦП AD7880 использует еще четыре сигнала управления – CONVST, CS, RD (входы АЦП) и выход BUSY (кстати, его можно и не использовать, в чем мы убедимся ниже). Кроме того, на вход CLKIN АЦП нужно подать тактовую последовательность с частотой до 2,5 МГц и отношением длительности единичного уровня к длительности нулевого в пределах от 0,4:0,6 до 0,6:0,4. К сожалению, хотя рассмотренный нами ранее сигнал ALE микроконтроллера имеет подходящую для данного случая частоту (в 6 раз меньшую, чем частота кварцевого резонатора, т. е. 2 МГц для 12-мегагерцового кварца), скважность этого сигнала составляет 33%. Поэтому на вход CLKIN АЦП нужно подать 1…2-мегагерцовую тактовую последовательность с отдельного генератора.

Как отмечалось, АЦП с МК должны связывать 15 или 16 проводников. Фактически мы будем вынуждены целиком задействовать два из 4- и 8-разрядных порта ввода/вывода МК. В принципе мы можем использовать для этого любые линии любых портов, но удобнее взять линии портов P1 и P2 или P1 и P3. Остановимся на втором варианте (рис. 5), поскольку в имеющихся у меня аппаратных средствах отладки порты P0 и P2 заняты – они используются для связи с компьютером.

Работает AD7880 следующим образом. В начальный момент времени на всех ножках, соединенных со входами управления АЦП, микроконтроллер должен установить единицы. Для запуска преобразования на CONVST необходимо подать отрицательный импульс. Перепад на нем из 0 в 1 при единичных уровнях сигналов на CS и RD запускает цикл преобразования. В течение примерно 20 мкс АЦП (при тактовой частоте 2 МГц) оцифровывает сигнал и заносит его в свой выходной регистр. В это время он удерживает нулевой уровень на своем выходе BUSY, и по состоянию этого сигнала МК может определить, завершил ли АЦП преобразование, или нет. Кстати, если по каким-либо причинам не возможно анализировать сигнал BUSY, можно просто отсчитать 20 мкс после подачи сигнала старта преобразования – за это время оно завершится.

После завершения, когда BUSY вернется в состояние логической 1, данные могут быть считаны из АЦП. С этой целью МК должен установить в 0 сначала сигнал на входе CS, а затем – на RD. Примерно через 75 нс после установки RD в 0 на выходах данных DB0–DB11 АЦП появится результат преобразования. МК должен считать его, после чего вернуть вначале RD, а затем CS в исходное состояние (логической 1), и на этом цикл завершается. Все вышесказанное иллюстрируется рис. 6.

Итак, схема соединения АЦП AD7880 с МК составлена и описан алгоритм работы. Теперь самое время перейти к рассмотрению программы, которая и реализует описанный алгоритм.

Программа работы с параллельными АЦП

```
;   НАЧАЛО ПРОГРАММЫ ЧТЕНИЯ АЦП AD7880,
;   РАБОТАЕМ С ПОРТАМИ P1 И P3, CS = P3.7,
;   RD = P3.6 , CONVST = P3.5.
;                  *****
CS        .EQU  P3.7
RD        .EQU  P3.6
CONVST    .EQU  P3.5
;
    MOV     P1,#11111111B      ;НАЧАЛЬНАЯ УС-
ТАНОВКА
  MOV     P3,#11111111B
  ;
  L7880:                    ;СОБСТВЕННО ЧТЕНИЕ
  ;
    CLR      CONVST ;ИМПУЛЬС СТАРТА ПРЕОБРАЗО-
ВАНИЯ
  SETB    CONVST
  ;
  NOP           ;ЗАДЕРЖКА НА ВРЕМЯ ПРЕОБРАЗОВА-
НИЯ
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  ;
  CLR     CS     ;CS=0
  CLR     RD     ;RD=0
  ;
  MOV     A,P1   ;ЧИТАЕМ ИЗ ПОР-
ТА P1 МЛ.И СР. ТЕТРАДЫ
  MOV     R4,A   ;СОХРАНЯЕМ ИХ В
R4
  ;
  MOV     A,P3   ;ЧИТАЕМ ИЗ ПОР-
ТА P3 СТ. ТЕТРАДУ
  MOV     R5,A   ;В R5R4 - РЕ-
ЗУЛЬТАТ
  ;
  SETB    RD     ;УСТАНОВКА RD В
1
  SETB    CS     ;УСТАНОВКА CS В 1
  ;
  SJMP    L7880       ;ЗАЦИКЛИВАНИЕ
  ;
```

Давайте разберемся с тем, что в ней присутствует. Первое, на что я рекомендую обратить внимание – это на то, что некоторые строки начинаются с точки с запятой (;). Запомните, во всех ассемблерах все, что записано после точки с запятой вплоть до конца текущей строки – это комментарии, которые ассемблер при трансляции вашей программы в контроллерные коды пропускает. Зачем же они тогда нужны? Для нас. В тот момент, когда мы пишем программу, мы знаем, зачем мы поставили ту или иную команду или последовательность команд. Но, отложив ее в сторону, спустя месяц мы с удивлением обнаруживаем, что не можем вспомнить, что и зачем мы написали в лежащей перед глазами программе. Именно для этого и нужны комментарии – понять, где начало программы, что она делает, что делает в ней та или иная команда. Настоятельно рекомендую не лениться и писать комментарии как можно подробнее – это сэкономит вам немало времени и нервов.

Иногда после точки с запятой ставят пустую строку, без комментариев – наличие пустых строк позволяет разбить программу на логически самостоятельные куски, что повышает читаемость текста программы. Кстати, обратите внимание, что точка с запятой (и с последующим комментарием) может стоять не только в начале строки, но и в середине ее, после той или иной команды – такой комментарий относится к конкретной команде, а не к последовательности команд (последнее называется фрагментом программы).

После такого вступления перейдем к описанию команд, входящих в анализируемую программу.

```
;
CS        .EQU  P3.7
RD        .EQU  P3.6
CONVST    .EQU  P3.5
;
```

Эти три строки сообщают ассемблеру, что везде, где он встретит имена CS, RD и CONVST, он должен подставлять вместо них линии портов P3.7, P3.6 и P3.5 соответственно. В принципе, мы могли бы этого не делать, и везде в программе писать имена выбранных линий (P3.7, P3.6, P3.5).

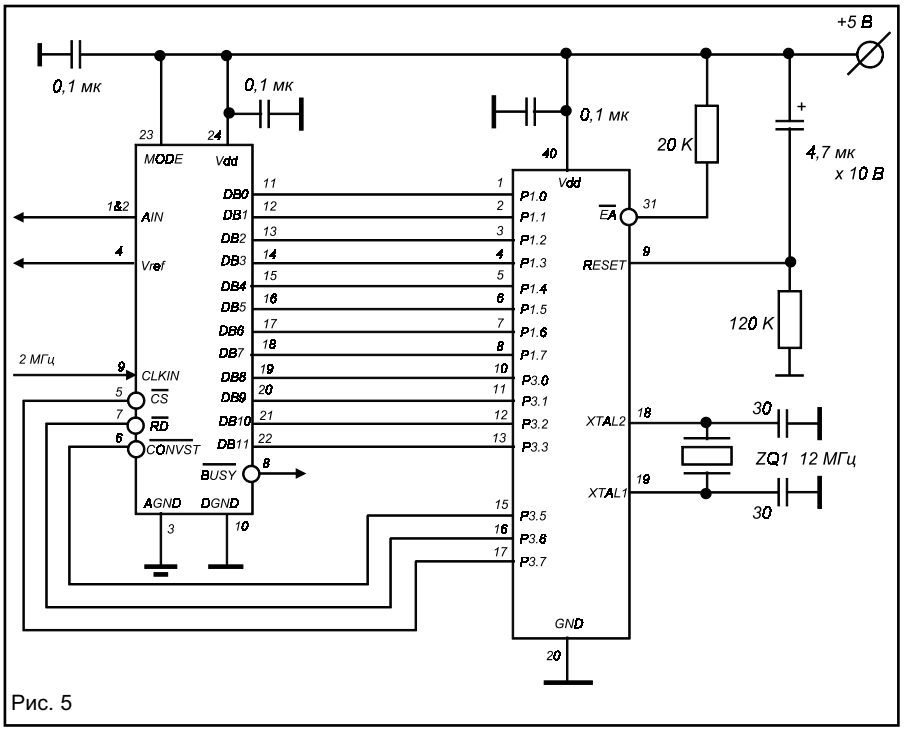

Рис. 5

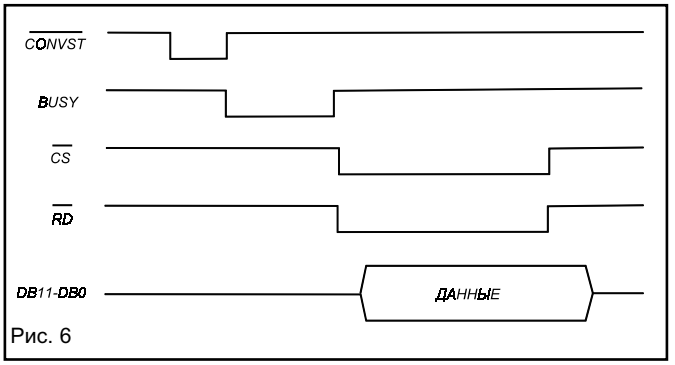

Рис. 6

Но я не рекомендую привыкать работать таким образом. Предположим, что завтра вам придется перенести этот фрагмент в другую программу, а там линия P3.7, будет уже занята другим устройством, и сигналом CS должна будет управлять другая линия порта, например P3.2. В таком случае вам придется в этой программе самостоятельно найти все места, где упомянута P3.7, и везде заменить ее на P3.2. Подобная операция – один из основных источников ошибок. В то же время, если в начале программы есть строки, подобные трем вышеупомянутым, то достаточно заменить P3.7 на P3.2 в первой из них, и дальше ассемблер осуществит требуемую замену самостоятельно, причем без ошибок. Есть еще аргументы в пользу работы с использованием подобных строк, но даже и без них, как мне кажется, ясна выгода использования в программе имен (CS, RD, CONVST) вместо явного указания закрепленных за ними линий.

```
;
MOV     P1,#11111111B
MOV     P3,#11111111B
;
```

Приведенные команды записывают в выходные буферы всех линий портов P1 и P3 единички. При этом, первая цифра, идущая после символа #, соответствует информации, заносимой в буфер линии (иногда говорят – разряда) P1.7 (или P3.7 соответственно), следующая за ней – информации, заносимой в буфер P1.6 (P3.6), и т. д. Естественно, последние цифры соответствуют буферам младших разрядов (P1.0 и P3.0). Если бы нам нужно было занести в P1.7, P1.5, P3.3 и P3.1 нули, а в остальные – единицы, то соответствующие команды выглядели бы так:

```
MOV     P1,#01011111B
 MOV     P3,#11110101B
```

Кстати, ассемблер вполне бы понял, если бы мы вместо #01011111B написали бы #5FH или #95 – все три записи соответствуют одному и тому же числу (равно как и #11110101B, #0F5H и #245). Но в данном случае шестнадцатеричное и десятичное представления числа менее наглядны, чем двоичное – в последнем случае сразу видно, какой бит установится в 0, а какой в 1.

Еще один момент, на который обязательно нужно обратить внимание – по правилам ассемблера для МК x51 перед числом обязательно должен стоять знак #. Запись

```
MOV     P1,#10101101B
```

означает, что в порт P1 нужно занести **число 00101101B** (45 дес.), в то время как

```
MOV     P1,10101101B
```

(то же, но без знака #) – что в P1 нужно занести **содержимое ячейки памяти с адресом 00101101B** (45 дес.). В ячейке же может храниться любое число, от 0 до 255, и поэтому в порту P1 после выполнения команды MOV P1,10101101B все линии, к примеру, могут оказаться зануленными (если в ячейке хранится нуль), хотя мы подразумевали, что в результате выполнения команды записи в порт числа 45 дес. занулятся только его первый, четвертый и шестой разряды, а остальные окажутся в единичном состоянии. Так что старайтесь не забывать ставить знак # перед числами, а если в какой-то момент обнаружите, что забыли, то не переживайте очень уж сильно, это одна из самых распространенных ошибок у программистов, пишущих на ассемблере для x51.

Зачем устанавливаются в 1 выходные буферы портов P1 и P3? Все линии порта P1 и младшие 4 порта P3 мы используем как входы для ввода в МК результата оцифровки, поэтому их, как я отмечал выше, нужно установить в единицы для предотвращения конфликтов на линиях. Линии P3.7, P3.6, P3.5 в соответствии с приведенным выше алгоритмом работы мы также до начала работы мы должны установить в 1. Ну а P3.4 мы устанавливаем в 1 за компанию с остальными.

Идем далее.

```
;
CLR     CONVST;ИМПУЛЬС СТАРТА ПРЕОБРАЗОВАНИЯ
SETB    CONVST
;
```

Напомню, что идущей чуть ранее строчкой (см. программу) CONVST.EQUP3.5 мы сообщили ассемблеру, что пятый разряд порта P3 мы назвали CONVST (поскольку он управляет одноименным входом АЦП). Команда CLR CONVST предписывает микроконтроллеру установить в 0 (сбросить, стереть) этот разряд. Соответственно SETB CONVST предписывает установить его в 1. Последовательно идущие друг за другом, эти две команды сформируют на выводе P3.5 импульс отрицательной полярности. Собственно, именно это нам и нужно было для того, чтобы запустить цикл преобразования АЦП.

Дальше идут одна за другой 20 команд NOP. Команда NOP – это так называемая пустая операция, которая предписывает процессору ничего не делать и переходить к выполнению следующей за ней команды. На первый взгляд может показаться, что она абсолютно бессмысленна. Однако надо принять во внимание, что она выполняется (если так можно сказать о команде, при исполнении которой, простите за каламбур, ничего не выполняется) в течение ровно 1 мкс при тактовой частоте МК 12 МГц. Двадцать идущих подряд команд NOP МК будет выполнять, как нетрудно сосчитать, ровно 20 мкс, в течение которых ни на одном из его портов и ни в одной из его ячеек памяти ничего не изменится. Другими словами, этот фрагмент – задержка на 20 мкс, в течение которых АЦП AD7880 гарантировано осуществит преобразование. Использование NOP для реализации задержек – одно из наиболее распространенных ее применений. Правда, много идущих одна за другой команд NOP – не самый красивый способ создания задержек, но он безотказно работает, и на первых порах, пока вы не научитесь более изящным способом формировать задержки нужной длительности, смело используйте NOP.

```
;
CLR     CS     ;CS=0
    CLR     RD ;RD=0
;
```

Этот фрагмент, надеюсь, очевиден: установка в 0 сигнала CS, а за ним и RD (соответственно линий P3.7 и P3.6).

Через 75 нс после того, как сигнал на входе RD АЦП установился в 0, на его выходах DB11–DB0 появится результат преобразования. Микроконтроллер считает результат следующим образом:

```
;
 MOV     A,P1  ;ЧИТАЕМ ИЗ ПОРТА P1 МЛ.И СР.
ТЕТРАДЫ
 MOV     R4,A  ;СОХРАНЯЕМ ИХ В R4
 ;
 MOV     A,P3  ;ЧИТАЕМ ИЗ ПОРТА P3 СТ. ТЕТРАДУ
 MOV     R5,A  ;В R5R4 - РЕЗУЛЬТАТ
 ;
```

Первая команда предписывает микроконтроллеру перенести данные из порта P1 в аккумулятор (мы уже упоминали о нем). Вторая переносит эти данные из аккумулятора в другой регистр, который называется R4. В результате этого младшие 8 бит результата “осядут” в регистре R4. Ну а после выполнения следующих двух команд старшие биты результата попадут, как вы, наверное, уже догадались, в регистр R5.

Остановимся на время для того, чтобы сделать лирическое отступление. Пока еще мы подробно не знакомились с внутренним устройством микроконтроллера, поэтому вы еще почти ничего не знаете о его регистрах. Это знакомство у нас еще впереди, хотя уже и не за горами. Сейчас же, чтобы не терять нить рассуждения, постарайтесь запомнить, что внутри МК есть 8-разрядный регистр-аккумулятор (или просто аккумулятор) и восемь 8-разрядных регистров общего назначения, которые называются R0, R1, …, R6, R7. 8-разрядные – это значит, что в них можно записывать числа длиной до 8 двоичных разрядов, т. е. от 00000000B до 11111111B,

или от 0 до 255 дес.. Команды МК позволяют осуществлять разнообразные действия с данными, находящимися в упомянутых регистрах. Какие конкретно – мы узнаем в ходе постепенного знакомства с системой команд МК.

Если вы обратили внимание, в вышеупомянутых командах перемещения данных после самой команды MOV записаны через запятую и тот регистр, куда нужно перенести данные, и тот, откуда их нужно взять. Так вот, во всех ассемблерах существуют свои правила, какой из регистров должен быть записан первым (приемник данных или их источник), а какой – вторым. Запомните, что **в ассемблере x51 после самой команды всегда первым записывается регистр, куда нужно поместить данные (регистр-приемник), а за ним через запятую – регистр, откуда их нужно извлечь (регистр-источник)**. Перенести данные из регистра R2 в аккумулятор – это MOV A,R2, но никак не MOVR2, A. Постарайтесь это хорошо запомнить.

Итак, вернемся к нашим баранам. Данные прочитаны из АЦП в регистры R4 и R5 микроконтроллера. Теперь с ними можно сделать все, что угодно – преобразовать, отобразить, вывести в ЦАП и т. д. Правда, сигналы CS и RD все еще сохраняют нулевые уровни. Для завершения операции чтения их нужно вернуть в единичное состояние. Делается это, как нетрудно догадаться, при помощи команд

```
;
SETB    RD     ;УСТАНОВКА RD В 1
SETB    CS     ;УСТАНОВКА CS В 1
;
Переходим к последней команде
;
        SJMP    L7880          ;ЗАЦИКЛИВАНИЕ
;
```

Она возвращает микроконтроллер к выполнению той части программы, которая идет после метки L7880. Меткой называется слово обычно не более чем из 8 символов, начинающееся с буквы и заканчивающееся двоеточием. Такая метка (L7880:) присутствует в программе перед командами, формирующими импульс старта преобразования. Следовательно, рассматриваемая команда вернет МК к формированию импульса старта преобразования, выполнению задержки 20 мкс и т. д., словом, к повторному исполнению оцифровки сигнала и считывания результата микроконтроллером до того момента, пока мы не отключим от схемы питание. Другими словами, мы таким образом зацикливаем программу, что и отражено в соответствующем комментарии.

Итак, мы разобрали первый пример – как связать микроконтроллер с АЦП параллельного типа, и какими командами заставить МК запустить преобразование в АЦП и считать во внутренние регистры микроконтроллера его результат. Как видите, в этом нет ничего недоступного. Сформировать на входе CONVST АЦП AD7880 отрицательный импульс, выждать 20 мкс и сформировать после этого еще 2 импульса на входах CS и RD оказалось при помощи микроконтроллера гораздо проще, чем сделать узел на дискретных элементах, формирующий все эти сигналы в нужной последовательности и с нужными длительностями. Для формирования импульсов на выводах МК есть замечательные команды CLR x.y и SETB x.y (x=0-3, y=0–7). Выполняя первую, мы устанавливаем соответствующую линию соответствующего порта в 0, а второй командой – устанавливаем ее в 1. Вставляя между ними некоторое количество команд NOP, мы можем изменять длительность этого импульса. И все, никаких мультивибраторов, времязадающих емкостей, инверторов для согласования фазировки сигналов. Словом, вы не зря захотели научиться работать с микроконтроллерами.

Александр Фрунзе,
alex.fru@mtu-net.ru

Продолжение следует