(Продолжение. Начало — №4/001)

Микроконтроллеры? Это же просто!

Сопряжение с четырехразрядными светодиодными матричными (5*7) индикаторами типа HCMS-2xxx

Индикаторы HCMS-2xxx (отечественные аналоги — КИПВ70А-4/5*7К, КИПВ71А-4/5*7К) являются малогабаритными 4-разрядными светодиодными матричными индикаторами. Они позволяют отображать практически любые символы и требуют для сопряжения относительно небольших аппаратных ресурсов МК, даже меньших, чем рассмотренный ранее АЛС318. Они хороши для встраивания в переносную аппаратуру, работающую в условиях недостаточного освещения и/или низкой температуры, где применение жидкокристаллических индикаторов сопряжено с некоторыми проблемами.

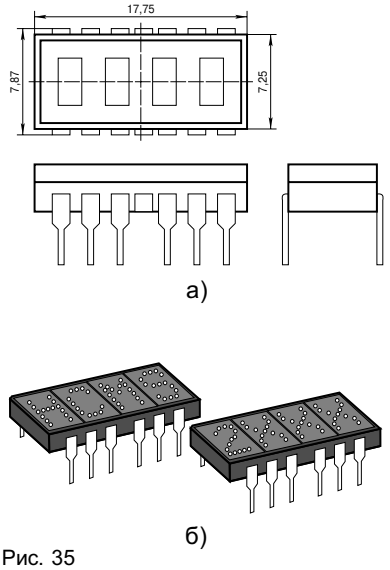

Рис. 35

Внешний вид и габаритные размеры индикаторов приведены на рис. 35.

Далее по тексту я эти индикаторы буду называть HCMS-2xxx, но все сказанное будет справедливо и для упомянутых выше отечественных аналогов.

Индикаторы совместимы с TTL-уровнями, имеют четыре знаковых разряда и представляют собой гибридную сборку в керамическом корпусе из четырех матриц по 7*5 элементов (140 светоизлучающих диодов) и регистра сдвига с формирователями (драйверами) постоянного втекающего тока. Каждая матрица светоизлучающих диодов имеет 7 строк (у диодов в строке соединяются катоды) и 5 столбцов (у диодов в столбце соединяются аноды). Соответствующие выводы столбцов всех матриц соединены между собой и с 1-5 выводами индикатора (рис. 36). Строки матриц наружу не выводятся, и внутри индикатора соединены каждая со своим драйвером втекающего тока. На рис. 36 эти драйверы символически изображены в виде транзисторов обратной проводимости.

Схема управления содержит 28-разрядный внутренний сдвиговый регистр с последовательным входом и последовательным и параллельными выходами, причем последние, как упоминалось, управляют драйверами втекающего постоянного тока. Драйверы, в свою очередь, подключены к общим катодным выводам строк светоизлучающих матриц. Последовательный выход сдвигового регистра связан с последним его разрядом и может быть подключен к входу следующего такого же индикатора, что позволяет создавать 8-, 12-, 16- и т. д. разрядные индикаторы.

Сдвиговый регистр снабжен также входом разрешения, нулевой сигнал на котором отключает отображение независимо от состояния сигналов на входах столбцов индикатора и содержимого регистра.

Схема сопряжения HCMS-2xxx с микроконтроллером приведена на рис. 37.

Как видите, схема довольно проста. Пять линий порта P1 (с P1.0 по P1.4) управляют транзисторами прямой проводимости, формирующими токи через столбцы матриц светоизлучающих диодов. Шестая линия (P1.5) передает информацию в сдвиговый регистр индикатора. Запись информации в сдвиговый регистр осуществляется тактовыми импульсами, формируемыми на P1.6. Для отключения индикатора можно использовать еще одну линию порта (P1.7), сигнал на которой при разрешении отображения должен быть единичным, а при запрещении – нулевым. Если мы не собираемся гасить отображение нулевым сигналом на входе Vb индикатора, его нужно оторвать от P1.7 и подать на него уровень логической единицы.

Работает наш индикатор в режиме динамической индикации. Вначале надо загрузить в него 28 бит, которые определят состояние каждого из светодиодов выбранного столбца у всех четырех матриц (включен или выключен). Пусть для определенности вначале мы зажжем светодиоды первых (крайних слева) столбцов индикаторных матриц, открыв для этого транзистор VT1. Первый из пересланных в сдвиговый регистр бит определит состояние нижнего светодиода левого столбца крайней справа индикаторной матрицы, второй бит — второго снизу светодиода этого же столбца этой же матрицы, и т. д., до 7-го бита, задающего состояние верхнего светодиода левого столбца правой матрицы (рис. 38). Кстати, если занесенный в сдвиговый регистр соответствующий светодиоду бит равен

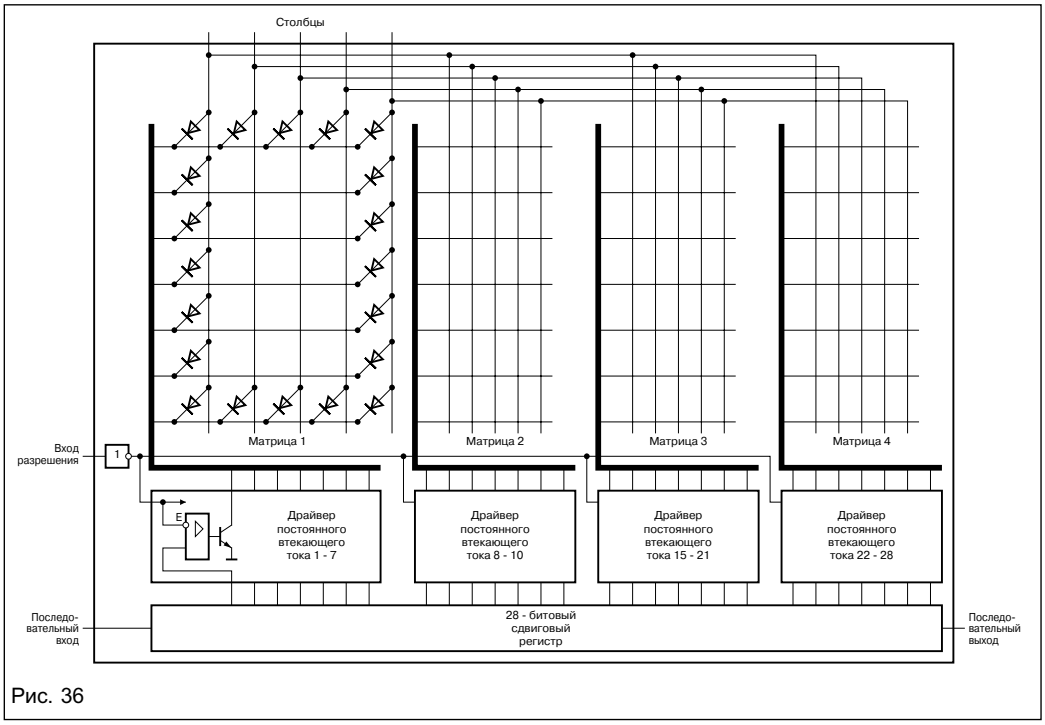

Рис. 36

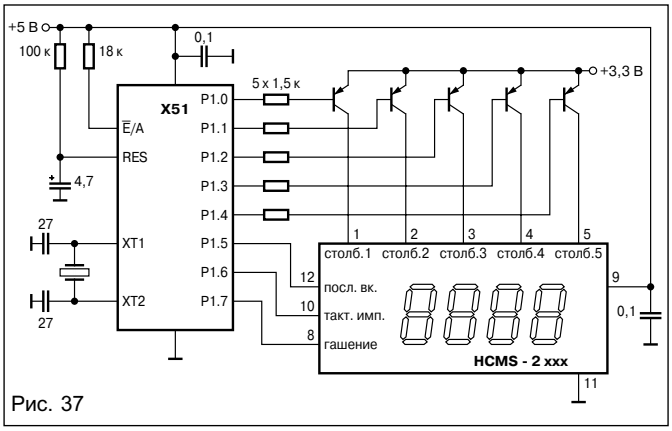

Рис. 37

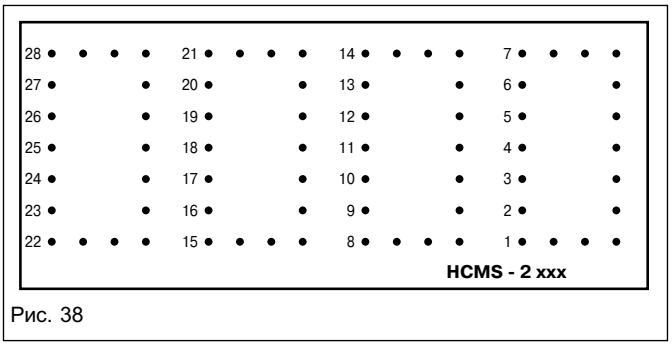

Рис. 38

1, то светодиод при активировании столбца будет гореть, если он равен 0 – соответственно будет погашен.

Далее, 8-й из переданных в сдвиговый регистр бит определит состояние нижнего светодиода левого столбца второй справа индикаторной матрицы, 9-й бит — второго снизу светодиода этого же столбца этой же матрицы, и т. д., до 14-го бита, задающего состояние верхнего светодиода левого столбца второй справа матрицы. Аналогичным образом биты с 15-го по 21-й зададут состояние светодиодов левого столбца третьей справа матрицы, а с 22-го по 28-й — четвертой матрицы, крайней слева, причем 28-й бит будет управлять ее левым верхним светодиодом.

Итак, мы разобрались, какой из битов сдвигового регистра отвечает за каждый светодиод. В соответствии с тем, какие из них для отображения требуемой информации должны быть зажжены, а какие погашены, мы заносим информацию в упомянутый регистр и открываем VT1. При этом те светодиоды левого столбца всех четырех матриц, которым соответствуют единичные биты, загораются, а все остальные светодиоды этих матриц оказываются погашенными.

Дав им погореть несколько миллисекунд, мы должны вывести в сдвиговый регистр информацию для светодиодов второго слева столбца. Также, как и в предыдущем случае, первый из пересланных в сдвиговый регистр бит определит состояние нижнего светодиода второго слева столбца крайней правой из индикаторных матриц, второй бит — второго снизу светодиода этого же столбца этой же матрицы, и т. д., до 7-го бита, задающего состояние верхнего светодиода второго слева столбца правой матрицы. Точно также, как и в предыдущем случае, биты с 8-го по 14-й отвечают за светодиоды второй справа матрицы, с 15-го по 21-й — третьей справа, и с 22-го по 28-й — левой матрицы. Но, еще раз подчеркну, что поскольку мы собираемся пропускать ток через второй слева столбец, названные биты отвечают теперь за светодиоды именно этого, второго столбца.

Занеся информацию в упомянутый регистр, мы на несколько миллисекунд открываем VT2. При этом, как и в предыдущем случае, те светодиоды второго слева столбца всех четырех матриц, которым соответствуют единичные биты, загораются, а все остальные светодиоды этих матриц оказываются погашенными.

Далее по описанному алгоритму мы выводим в сдвиговый регистр информацию для третьего слева столбца и на несколько миллисекунд открываем VT3. После этого, как нетрудно догадаться, нужно вывести в него информацию для четвертого слева столбца и открыть VT4, а затем — и для пятого, сопровождая это открыванием VT5. По завершении отображения пятого столбца цикл необходимо повторить, и повторять его до того момента, пока мы не прервем процесс отображения.

Для тех, кто делает первые шаги в микроконтроллерной технике, это выглядит крайне непросто. Откуда взять информацию, какие светодиоды надо зажигать в первом столбце, какие во втором, и т. д.? Как выводить информацию в сдвиговый регистр, управлять транзисторами, формировать десятичную запятую? На фоне этих вопросов работа рассмотренных выше индикаторов, особенно HT1610, кажется вам, наверное, ох какой простой!

Но не отчаивайтесь, с теми знаниями, которые у вас уже имеются, вам и этот индикатор вполне по зубам. Вам это может показаться удивительным, но программа, которая оживит этот индикатор, содержит лишь те команды, которые нам с вами уже хорошо знакомы. Вся сложность работы заключается в том, чтобы разбить поставленную задачу на ряд более простых, и написать для них подпрограммы. Пока у вас еще просто очень мало навыков, и описанные в этой главе примеры, равно как и те, которые вы встретите в последующих главах, направлены в первую очередь на то, чтобы эти навыки у вас сформировать, по крайней мере некоторое минимальное их количество, которое позволит вам, опираясь на них, двигаться дальше самостоятельно.

Мы чуть-чуть отвлеклись, но сделал это я умышленно. Я хочу, чтобы вы обратили особое внимание на то, каким образом мы разобьем задачу на части и решим каждую из этих малых задач, причем даже не сразу, а, как говорится, в два-три приема. Я покажу вам, что получается при первой попытке решения задачи, и как результат дальше оптимизируется. Те, кто имеет большой опыт, проводит эти предварительные стадии разработки в уме. Но у нас с вами такого опыта нет, поэтому мы будем решать нашу задачу постепенно, не всегда сразу оптимально, но в конце получим как некоторые новые навыки, так и правильно работающую подпрограмму, а именно это и есть наша цель.

Итак, поехали. По аналогии с рассмотренными выше индикаторами сформулируем задачу следующим образом. Пусть в ячейках микроконтроллера AD00+0 - AD00+3 хранятся в обычном двоичном представлении четыре цифры, которые нам нужно отобразить на экране индикатора следующим образом: цифру из AD00+0 — в крайнем справа разряде индикатора, из AD00+1 — во втором справа разряде, из AD00+2 — в третьем справа, и из AD00+3 — в крайнем слева. Вот и вся задачка, естественно, с учетом правил работы с HCMS-2xxx.

Первый шаг, который мы сделаем — введем некоторое упрощение в упомянутые только что правила работы с индикатором. Как вы помните, мы говорили о том, что нужно вывести в сдвиговый регистр индикатора вначале информацию, касающуюся первых столбцов *всех четырех отображаемых символов* и открыть VT1, затем касающуюся вторых столбцов *всех четырех отображаемых символов* и открыть VT2, и т. д. Это означает, что перед выводом в сдвиговый регистр информации мы должны так или иначе обрабатывать все четыре отображаемых символа одновременно. Для начинающих это достаточно сложная задача. Чтобы ее упростить, давайте поступим следующим образом. Мы будем отображать символы по одному. Как? Сначала занесем в сдвиговый регистр в биты с 1 по 7 информацию, касающуюся первого столбца того символа, который должен отобразиться в крайнем справа разряде индикатора, а в биты с 8 по 28, ответственные за соответствующие столбцы остальных символов, занесем нули. Откроем VT1 и отобразим первый столбец этого правого символа. Затем, по-прежнему занулив биты 8-28, в биты с 1 по 7 занесем информацию относительно второго столбца этого правого символа, выведем полученные 28 бит в сдвиговый регистр и, открыв VT2, отобразим второй его

столбец. Далее, занеся опять-таки в 1-7 биты информацию о третьей строке отображаемого символа, а в 8-28 биты — нули, выведем их в регистр, откроем VT3 и отобразим третий столбец, а затем аналогично ему четвертый и пятый. Таким образом, мы отобразим весь символ, при этом остальные три символа отображены не будут, т. к. в биты, их формирующие, мы все время заносили нули.

Думаю, что вы уже догадались, что для отображения второго символа соответствующую его столбцам информацию мы будем заносить в биты с 8 по 14-й, а в 1-7 и в 15-28 биты опять будем помещать нули. Занося в 8-14 биты информацию, управляющую светодиодами первой, второй и т. д., вплоть до пятой колонки и открывая последовательно все те же VT1, VT2, …, VT5, мы отобразим второй символ при погашенных первом, третьем и четвертом. Естественно, для отображения третьего символа соответствующую его столбцам информацию мы будем заносить в 15-21 биты, а помещать нули будем в 1-14 и в 22-28 биты, а для отображения четвертого символа занулять будем 1-21 биты, а информацию, соответствующую его столбцам, станем выводить в 22-28 биты.

Итак, задача слегка упростилась. Мы собираемся в соответствии с описанным в двух предыдущих абзацах упрощенным алгоритмом выводить в индикатор символы последовательно — сначала из AD00+0 в крайний справа знаковый разряд, затем из AD00+1 во второй справа, и т. д. Что нам для этого нужно?

Нам нужно написать подпрограмму, которая отображала бы цифру, помещенную в один из регистров МК (пусть для определенности в R2) в разряд индикатора, номер которого размещен в другом регистре (к примеру, в R3). Также для определенности будем считать, что номер крайнего справа отображаемого индикатором символа — 0, второго справа — 1, третьего справа — 2, и четвертого справа, т. е. левого — 3. Лучше эти номера выбрать именно таким образом, и тогда из AD00+0 символ выводится в нулевой разряд индикатора, из AD00+1 — в первый разряд, из AD00+2 — во второй разряд, а из AD00+3 — в третий разряд.

Еще раз повторюсь, нам нужна подпрограмма (назовем ее VIVHCMS), выводящая символ, размещенный в регистре R2, в разряд индикатора, номер которого помещен в регистр R3. Если она будет в нашем распоряжении, то поставленная задача (вывод информации в индикатор из ячеек микроконтроллера с адресами AD00+0 - AD00+3) легко решится при помощи следующей программки:

```
;
OTBHCMS:
;
    MOV   R1,#255
;
OTBHC1:
    MOV   R2,AD00+0
    MOV   R3,#0
    LCALL  VIVHCMS
;
    MOV   R2,AD00+1
    MOV   R3,#1
    LCALL  VIVHCMS
;
    MOV   R2,AD00+2
    MOV   R3,#2
    LCALL  VIVHCMS
;
    MOV   R2,AD00+3
    MOV   R3,#3
    LCALL  VIVHCMS
;
    DJNZ  R1,OTBHC1
    RET
```

Перед вами простенькая подпрограмма, которая 255 раз выполняет следующие действия: вначале помещает в R2 данные из ячейки с адресом AD00+0, а в R3 — ноль, т. е. адрес ячейки индикатора, где нужно отобразить эти данные, и вызывает задуманную, но еще не написанную VIVHCMS, выводящую символ, размещенный в регистре R2, в разряд индикатора, номер которого помещен в регистр R3. Затем помещает в R2 данные из ячейки с адресом AD00+1, а в R3 — единицу, и вновь вызывает VIVHCMS. То же самое повторяется с данными из AD00+2 и AD00+3, которые попадают соответственно во второй и третий разряды индикатора. И все! Как только мы сформулировали требования к тому, что же должна делать VIVHCMS, написать подпрограмму, решающую основную задачу, оказалось довольно просто. Все сложности теперь именно в подпрограмме VIVHCMS, мы сконцентрируем свои усилия на ней, и скоро вы убедитесь, что она не столь уж сложна.

А теперь еще одно небольшое отвлечение. Почему я решил выделить VIVHCMS в отдельную подпрограмму, которая должна выводить символ, размещенный в регистре R2, в разряд индикатора, номер которого помещен в регистр R3? Как определить, когда нужно пытаться те или иные действия вынести в определенную подпрограмму, и как сформулировать, что же эта подпрограмма должна делать?

Ответ простой — любые действия, которые вам придется делать в вашей программе более, чем один или два раза, удобно оформить в виде подпрограммы со сразу понятным для вас именем. Например, вы не раз будете в своих программах осуществлять операции умножения одного 16-разрядного числа на другое или делить 32-разрядное число на 16-разрядное, следовательно, эти действия нужно оформить в виде соответствующих подпрограмм (в одной из следующих глав мы тщательно их разберем). По той же причине мы выделили в самостоятельную подпрограмму процесс преобразования двоичного числа в двоично-десятичное (вспомните главу 3) — эта подпрограмма всегда нужна, если вы собираетесь отобразить на знакосинтезирующем индикаторе результаты тех или иных измерений или вычислений.

Да и зачем далеко ходить? Вспомните, как мы организовывали вывод информации на экран НТ1610. Мы занесли в аккумулятор число из AD00+7 и вызвали подпрограмму SIMBOL1, затем — число из AD00+6 и снова вызвали SIMBOL1, и т. д. Вам это ничего не напоминает? А подпрограмма DISPLEY из раздела, посвященного АЛС318? При ее вызове в R0 мы помещали число от 0 до 7, и она далее извлекала цифру из ячейки памяти МК с адресом AD00+n (n=0…7), и выводила его в порт P1 для отображения в n-й разряд АЛС318. Как видите, по крайней мере в том случае, когда вам нужно вывести на тот или иной дисплей цифровую информацию (а это всегда не менее 3-4 цифр), очень полезно выделить в самостоятельную подпрограмму все действия по пересылке символа на дисплей, при этом сам пересылаемый символ (а если надо, то и его место отображения на индикаторе) должны находится в каком-нибудь регистре (регистрах) нашего МК. Если вы привыкните это делать, то написанные вами программы будут проще, стройнее и безошибочнее.

Кстати, при написании подпрограммы вывода цифр в DV-16100 (см. главу 3) я поленился выделить совокупность команд, организующих вывод цифры в индикатор, в отдельную подпрограммку. Результат не замедлил сказаться — посмотрите, сколь большой и сложной оказалась программа IZOBR1! А если бы не поленился, то получилось бы нечто похожее на подпрограмму INDVIV (см. главу 2). Кстати, можете в качестве примера попробовать совершить такую оптимизацию подпрограммы IZOBR1 самостоятельно, это будет полезной практикой.

Итак, вернемся к нашим баранам. Подпрограмма вывода информации на индикатор HCMS-2xxx нами уже написана, и теперь нужно только составить отдельную подпрограмму VIVHCMS, которая должна выводить символ, размещенный в регистре R2, в разряд индикатора, номер которого помещен в регистр R3. Займемся этим.

Вспомним, что при отображении того или иного символа мы должны в семь выбранных ячеек сдвигового регистра HCMS-2xxx вывести вначале информацию о том, какие из

светодиодов первого столбца формирующей изображение матрицы должны быть зажжены, а какие погашены, затем туда же вывести аналогичную информацию для второго, третьего, четвертого и пятого столбцов. Вот вам и вопрос — откуда взять эту информацию?

Обратимся к рис. 39. На нем схематически изображены несколько вариантов нашей знакосинтезирующей матрицы формата 7*5 со сформированными изображениями символов “0”, “3” и “3,” (в последнем случае — цифра 3 с идущей сразу после нее десятичной запятой). Для того чтобы эти цифры были сформированы реальным индикатором, нужно, чтобы те светодиоды матрицы, которые соответствуют квадратикам, содержащим цифру 1, были зажжены, а те, которые соответствуют пустым квадратикам — погашены. Это, думаю, понятно всем, кто читает эти строки.

Далее вспомним, что при отображении первого столбца (открывании VT1) первый пересланный в сдвиговый регистр

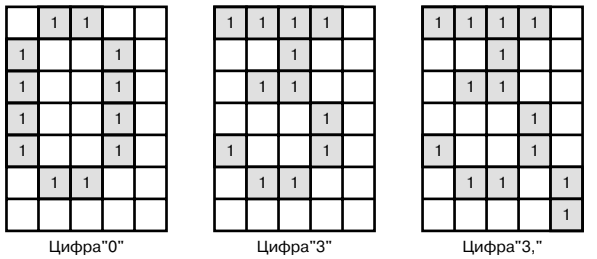

Рис. 39

бит отвечает за состояние левого нижнего светодиода знакосинтезирующей матрицы (бит1 = 1 — светодиод загорится, бит1 = 0 — погаснет). Второй бит, как тоже нетрудно вспомнить, отвечает за второй снизу светодиод этого же столбца, и т. д., вплоть до седьмого, который отвечает за седьмой снизу, т. е. за верхний светодиод все того же первого столбца. А теперь внимание! С учетом сказанного, для того, чтобы отобразить первую колонку цифры 0, биты с 1 по 7 в сдвиговом регистре должны иметь следующие значения: бит1 = 0, бит2 = 0, бит3 = 1, бит4 = 1, бит5 = 1, бит6 = 1, бит7 = 0. Для отображения второй и третьей колонок цифры 0, биты с 1 по 7 в сдвиговом регистре должны иметь следующие значения: бит1 = 0, бит2 = 1, бит3 = 0, бит4 = 0, бит5 = 0, бит6 = 0, бит7 = 1. Чтобы отобразить четвертую колонку цифры 0, биты с 1 по 7 в сдвиговом регистре должны иметь следующие значения: бит1 = 0, бит2 = 0, бит3 = 1, бит4 = 1, бит5 = 1, бит6 = 1, бит7 = 0, а для пятой колонки все семь бит, пересылаемые в сдвиговый регистр, должны быть нулевыми.

Я специально так подробно расписал состояние всех семи бит для каждой из пяти колонок — в описываемом здесь принципе формирования цифр на экране матричного индикатора обязательно нужно разобраться так, чтобы была исключена любая неясность или двусмысленность, в противном случае вы очень скоро потеряете нить рассуждений. Если вам что-то оказалось непонятным – еще раз внимательно прочитайте содержимое последних двух абзацев и вникните в приводимую в них информацию.

Идем далее. Давайте разберемся с битами, формирующими цифру 3. Для того, чтобы отобразить первую колонку цифры 3, биты с 1 по 7 в сдвиговом регистре должны иметь следующие значения: бит1 = 0, бит2 = 0, бит3 = 1, бит4 = 0, бит5 = 0, бит6 = 0, бит7 = 1. Для отображения второй колонки цифры 3, биты с 1 по 7 в сдвиговом регистре должны иметь следующие значения: бит1 = 0, бит2 = 1, бит3 = 0, бит4 = 0, бит5 = 1, бит6 = 0, бит7 = 1. Третья колонка предполагает, что бит1 = 0, бит2 = 1, бит3 = 0, бит4 = 0, бит5 = 1, бит6 = 1, бит7 = 1. Чтобы отобразить четвертую колонку тройки, биты с 1 по 7 в сдвиговом регистре должны иметь следующие значения: бит1 = 0, бит2 = 0, бит3 = 1, бит4 = 1, бит5 = 0, бит6 = 0, бит7 = 1, а для пятой колонки, как и в предыдущем случае, все семь бит, пересылаемые в сдвиговый регистр, должны быть нулевыми.

Третий пример — цифра 3 со стоящей после нее десятичной запятой. Формирующие изображение этого символа (точнее, двух символов, ведь десятичная запятая самостоятельный символ) биты для колонок с первой по четвертую совершенно идентичны битам для этих же колонок, формирующих цифру 3. В этом нет ничего удивительного — и там, и там формируется цифра 3. А вот пятая колонка для цифры со стоящей после нее десятичной запятой (причем любой, не только тройки) должна выглядеть следующим образом: бит1 = 1, бит2 = 1, бит3 = 0, бит4 = 0, бит5 = 0, бит6 = 0, бит7 = 0.

Я надеюсь, рассмотренные примеры дают ясное, четкое и однозначное объяснение принципа формирования цифр на экране матричного индикатора. Кстати, совершенно аналогично они формируются и в индикаторах типа DV-16100, но входящий в состав этих индикаторов контроллер выполняет такое формирование самостоятельно, благодаря чему процесс вывода в него информации намного проще, чем аналогичный в HCMS-2xxx.

Пойдем дальше. Как мы только что убедились, каждому из отображаемых на HCMS-2xxx символу должны быть поставлены в соответствие пять 7-битовых чисел: первое — с информацией для первой колонки, второе — с информацией для второй колонки, и т. д. Вы, наверное, не забыли, что ячейки как памяти программ нашего МК, так и памяти данных, 8-разрядные. Следовательно, информацию о каждом символе нам придется хранить в пяти ячейках памяти программ. Если нам предстоит отображать 32 символа, информация о них займет 32·5=160 байт памяти программ, если 128 символов — то 128·5=640 байт. Это нужно учитывать при анализе того, хватит ли у применяемого микроконтроллера объема ПЗУ и для программы, и для этих данных, или придется применять МК с большим объемом ПЗУ.

Согласитесь, естественно, что первыми в этом массиве информации, необходимой для формирования изображения цифр (этот массив называют таблицей знакогенератора) должны быть 5 байт, требуемые для формирования цифры 0, за ними 5 байт с информацией для цифры 1, и т. д., по крайней мере до цифры 9 включительно. Логично также и то, что внутри каждых пяти байт, несущих информацию о той или иной цифре, первый байт содержит информацию, необходимую для формирования первой колонки этой цифры, второй байт — информацию для второй колонки, третий байт — для третьей колонки, четвертый — для четвертой и пятый — пятой. По крайней мере, если соблюсти именно такой порядок, интересующая нас подпрограмма VIVHCMS напишется относительно просто. Любой же другой порядок следования информации в таблице знакогенератора заметно усложнит нашу подпрограмму.

Кстати, а как вы думаете, почему таблица знакогенератора должна храниться именно в памяти программ? Не догадались? Ну а где же еще? Ведь и память данных, и регистры теряют свое содержимое при выключении питания.

У вас может возникнуть еще один вопрос. Информация для вывода в сдвиговый регистр 7-битная, а ячейки памяти — 8-битные. Разрядность одного не соответствует разрядности другого. Как быть?

Ну, на самом деле, хранить 7 бит информации в 8-битной ячейке можно (наоборот нельзя). Давайте договоримся, что эта 7-битная информация будет у нас храниться в младших семи битах отведенного под нее байта. В старшем, восьмом его бите, который мы никак не будем использовать, может с равным успехом быть и 0 и 1. Пусть для определенности это всегда будет 1. Это абсолютно несущественно, важно то, что мы должны исключить любые неопределенности — это правило, которого всегда полезно придерживаться.

Итак, мы договорились, что в памяти программ начиная с некоторого определенного места (пометим его какой-либо меткой, например TABZNAK:) хранятся 5 байт с информацией для отображения цифры 0, за ними — 5 байт с информацией для отображения цифры 1, затем по пять байт с информацией для двойки, тройки, …, восьмерки, девятки. Далее, после 9 в шестнадцатиричной системе идут еще

```
;
;ЗНАКОГЕНЕРАТОР НА 32 СИМВОЛА
;
TABZNAK:
;
;ЦИФРА 0 [00H]
.DB     10011110B,10100001B,10100001B,10011110B,10000000B
;ЦИФРА 1 [01H]
.DB     10000000B,10000000B,10000010B,10111111B,10000000B
;ЦИФРА 2 [02H]
.DB     10100010B,10110001B,10101001B,10100110B,10000000B
;ЦИФРА 3 [03H]
.DB     10010001B,10100101B,10100111B,10011001B,10000000B
;ЦИФРА 4 [04H]
.DB     10001111B,10001000B,10001000B,10111111B,10000000B
;ЦИФРА 5 [05H]
.DB     10010111B,10100101B,10100101B,10011001B,10000000B
;ЦИФРА 6 [06H]
.DB     10011110B,10100101B,10100101B,10011001B,10000000B
;ЦИФРА 7 [07H]
.DB     10100001B,10010001B,10001001B,10000111B,10000000B
;ЦИФРА 8 [08H]
.DB     10011010B,10100101B,10100101B,10011010B,10000000B
;ЦИФРА 9 [09H]
.DB     10100110B,10101001B,10101001B,10011110B,10000000B
;ЦИФРА A [0AH]
.DB     10011110B,10100001B,10100001B,10011110B,10000000B
;ЦИФРА B [0BH]
.DB     10011110B,10100001B,10100001B,10011110B,10000000B
;ЦИФРА C [0CH]
.DB     10011110B,10100001B,10100001B,10011110B,10000000B
;ЦИФРА D [0DH]
.DB     10011110B,10100001B,10100001B,10011110B,10000000B
;ЦИФРА E [0EH]
.DB     10011110B,10100001B,10100001B,10011110B,10000000B
;ЦИФРА F [0FH]
.DB     10000000B,10000000B,10000000B,10000000B,10000000B
;
;ЦИФРА 0,[10H]
.DB     10011110B,10100001B,10100001B,10011110B,11100000B
;ЦИФРА 1,[11H]
.DB     10000000B,10000000B,10000010B,10111111B,11100000B
;ЦИФРА 2,[12H]
.DB     10100010B,10110001B,10101001B,10100110B,11100000B
;ЦИФРА 3,[13H]
.DB     10010001B,10100101B,10100111B,10011001B,11100000B
;ЦИФРА 4,[14H]
.DB     10001111B,10001000B,10001000B,10111111B,11100000B
;ЦИФРА 5,[15H]
.DB     10010111B,10100101B,10100101B,10011001B,11100000B
;ЦИФРА 6,[16H]
.DB     10011110B,10100101B,10100101B,10011001B,11100000B
;ЦИФРА 7,[17H]
.DB     10100001B,10010001B,10001001B,10000111B,11100000B
;ЦИФРА 8,[18H]
.DB     10011010B,10100101B,10100101B,10011010B,11100000B
;ЦИФРА 9,[19H]
.DB     10100110B,10101001B,10101001B,10011110B,11100000B
;ЦИФРА A,[1AH]
.DB     10011110B,10100001B,10100001B,10011110B,11100000B
;ЦИФРА B,[1BH]
.DB     10011110B,10100001B,10100001B,10011110B,11100000B
;ЦИФРА C,[1CH]
.DB     10011110B,10100001B,10100001B,10011110B,11100000B
;ЦИФРА D,[1DH]
.DB     10011110B,10100001B,10100001B,10011110B,11100000B
;ЦИФРА E,[1EH]
.DB     10011110B,10100001B,10100001B,10011110B,11100000B
;ЦИФРА F,[1FH]
.DB     10000000B,10000000B,10000000B,10000000B,10000000B
;
```

Рис. 40

цифры A, B, C, D, E и F (надеюсь, не забыли). Отведем еще по 5 байт под каждую из них. А какие цифры идут в этой HEX-системе дальше, после F? Правильно, 10, 11, 12, …, 19, 1A, 1B, … 1F . Если вы помните, при анализе АЛС318 мы договаривались, что хранящиеся в AD00+n для отображения числа 00000011B (03H) или 00000111B (07H) должны отображаться в виде упомянутых в скобках тройки или семерки, а вот 00010011B (13H) или 00010111B (17H) — в виде тройки с десятичной запятой после нее или семерки с той же запятой. Давайте сохраним эту договоренность. При этом в позициях с 10H по 19H мы разместим 5-байтовые фрагменты, формирующие изображения символов “0,”, “1,”, …, “9,”. В итоге мы должны получить таблицу знакогенератора, приведенную на рис. 40. Не пугайтесь, встретив в ней новую для вас директиву ассемблера .DB — без нее никак не сформировать эту таблицу, т. к. именно она предписывает ассемблеру разместить один за другим в памяти программ те байты, которые записаны друг за другом через запятую после этой директивы.

Теперь можно сказать, что мы преодолели самую большую трудность на пути написания подпрограммы VIVHCMS — сформировали массив данных, где по известному нам принципу размещена вся информация, необходимая для отображения на HCMS-2xxx любой из цифр.

Далее задачу написания подпрограммы VIVHCMS можно условно разбить на две: первая — формирование массива данных, который надо выводить в сдвиговый регистр индикатора, и второй — вывод этого массива в индикатор. Начнем с первой. Но перед тем, как это сделать, нужно еще отвести в ОЗУ МК область из четырех байт, куда мы будем заносить информацию для вышеупомянутого формируемого массива данных. Пусть это будут ячейки с символическими именами EKRAN, EKRAN+1, EKRAN+2, и EKRAN+3. В младших 7 битах ячейки EKRAN будут располагаться первые 7 выводимых в сдвиговый регистр бит, в младших 7 битах EKRAN+1 — вторые 7 бит, и т. д. Следовательно, первая из поставленных задач сводится к занесению информации в ячейки EKRAN-EKRAN+3, а вторая — вывод ее из них в регистр индикатора.

Ниже приведен фрагмент подпрограммы, осуществляющий заполнение ячеек EKRAN - EKRAN+3.

```
;
;
    MOV   A,#0
    MOV   COUNSK,A  ;COUNSK М.Б. РАВЕН 0,1,2,3,4,
                                          ;ЭТО НОМЕР
ВЫВОДИМОГО СТОЛБЦА
    MOV   EKRAN,A
    MOV   EKRAN+1,A
    MOV   EKRAN+2,A
    MOV   EKRAN+3,A ;ПРЕДВАРИТЕЛЬНО ЗАНУЛИЛИ
;
    MOV   A,R3
    ANL   A,#00000011B ;R3 НЕ М.Б. БОЛЕЕ 3
    MOV   R3,A
    MOV   A,#EKRAN
    ADD   A,R3
    MOV   R0,A    ;В R0 АДРЕС ТОЙ ИЗ
                                          ;EKRAN...EKRAN+3,
                                          ;КУДА ВЫВОД
    MOV   DPTR,#TABZNAK
;
    MOV   A,R2
    RL    A
    RL    A
    ADD   A,R2    ;A = 5*R2
    ADD   A,COUNSK ;A = 5*R2+COUNSK
    MOVC  A,@A+DPTR
    MOV   @R0,A   ;В EKRAN...EKRAN+3
                                          ;ВЫВЕЛИ БАЙТ,
                                    СООТВ.
                                          ;ПЕРВОЙ
                                    КОЛОНКЕ
              ;ОТОБРАЖАЕМОГО СИМВОЛА
;
```

Теперь давайте разберемся с тем, что делается в этом фрагменте подпрограммы. Вначале мы зануляем некую переменную COUNSK (в ней должен храниться номер отображаемого столбца, т. е. она может принимать значения лишь

0, 1, 2, 3, 4) и ячейки EKRAN – EKRAN+3. Как вы должны помнить, только в одну из этих ячеек мы выведем информацию для отображения: в EKRAN, если будем отображать символ в правом разряде индикатора, в EKRAN+1, если во второй справа, и т. д. Остальные же, чтобы светодиоды неотображаемых символов не горели, должны содержать нули. Далее, переносим из R3 в аккумулятор номер разряда индикатора, в который наша подпрограммы должна вывести информацию, и для страховки ограничим его числом 3 при помощи команды ANL A,#00000011B. Затем складываем это число с адресом ячейки, которой мы дали символическое имя EKRAN (именно с адресом, а не с содержимым, на что указывает знак # перед именем EKRAN). Что при этом получилось? Если в R3 перед сложением был 0, т. е. предполагался вывод в крайний справа разряд индикатора, то после сложения в аккумуляторе будет храниться не изменившийся адрес ячейки с именем EKRAN. Если в R3 перед сложением была 1, т. е. предполагался вывод во второй справа разряд индикатора, то после сложения в аккумуляторе будет храниться адрес следующей ячейки (EKRAN+1). Двойка в R3 даст в аккумуляторе после сложения адрес EKRAN+2, тройка, как нетрудно догадаться, — адрес EKRAN+3. А ведь именно это нам и было нужно!

Далее мы сохраняем этот найденный адрес в R0, а в регистр DPTR занесем адрес начальной ячейки таблицы знакогенератора (MOV DPTR,#TABZNAK). После этого в аккумулятор из R2 помещаем число, которое нам предстоит отобразить. Идущая затем команда RL A осуществит, как видно из рис. 41, сдвиг каждого бита нашего двоичного числа на одну позицию влево, что эквивалентно удвоению числа (на рис. 41 в аккумуляторе до сдвига было число 00000111B=7дес., а после сдвига — 00001110B=14дес.).

Зачем это нужно? А вот зачем. Положим, отображаемая цифра — двойка. Где в таблице знакогенератора мы должны найти информацию для формирования ее первого, второго, …, пятого столбцов? Вспомним, что в первых пяти байтах таблицы (TABZNAK+0… TABZNAK+4) хранятся байты для формирования цифры 0, в следующих пяти (TABZNAK+5… TABZNAK+9) — цифры 1. Информация для первого столбца

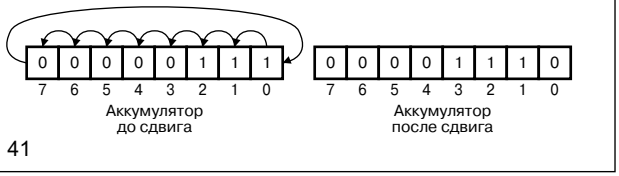

Рис. 41

двойки хранится в ячейке TABZNAK+10, для второго столбца — в TABZNAK+11, …, для пятого столбца — в TABZNAK+14. В общем виде, при помощи формулы можно записать, что информация для n-го столбца цифры 2 хранится в ячейке памяти программ с адресом TABZNAK + 2·5 + n, где для первого столбца n=0, для второго — n=1, …, для пятого — n=4. Если мы хотим отобразить цифру 3, то информация для ее n-го столбца хранится в ячейке памяти программ с адресом TABZNAK + 3·5 + n, для цифры 7 — TABZNAK + 7·5 + n, и т. д.

Теперь вам понятно, что для нахождения байта с соответствующей нужному столбцу отображаемой цифры информацией мы должны взять адрес начала таблицы знакогенератора (#TABZNAK), прибавить к нему умноженную на пять отображаемую цифру, и к сумме прибавить номер n отображаемого столбца. Теперь вернемся чуть назад, к команде RL A, которая, как мы говорили, увеличивает вдвое выводимое на индикатор число. Еще одно выполнение этой команды учетверит это число, а прибавление к нему содержимого регистра R2, где по-прежнему хранится предназначенная для отображения цифра, приведет к тому, что в аккумуляторе у нас окажется число, ровно в пять раз большее, чем отображаемая цифра из R2. То, что нам и нужно! Теперь осталось лишь прибавить к аккумулятору командой ADD A,COUNSK содержимое ячейки COUNSK (в ней должен храниться номер отображаемого столбца, причем именно в нужном нам виде — 0, 1, 2, 3, 4), и мы располагаем всем, что надо для нахождения информации о первом столбце отображаемой цифры.

Стоящая далее команда MOVC A,@A+DPTR выполняет следующее действие: она складывает содержимое аккумулятора и DPTR, находит ячейку памяти программ с адресом, равным полученной сумме, извлекает из нее информацию и помещает извлеченное в аккумулятор. Но вспомним, в DPTR у нас хранился адрес начала таблицы знакогенератора (#TABZNAK), а в аккумуляторе — умноженная на пять отображаемая цифра с прибавленным к ней номером n отображаемого столбца! Следовательно, после выполнения команды MOVC A,@A+DPTR мы получим в аккумуляторе байт с информацией о первом столбце отображаемой цифры. И дальше командой MOV @R0,A перешлем его в одну из ячеек EKRAN – EKRAN+3, ту, адрес которой мы сохранили в R0, и которая соответствует содержавшемуся в R3 номеру разряда индикатора, где мы хотели отобразить наш символ.

Александр Фрунзе
alex.fru@dian.ru