Статья опубликована в журнале Радиолюбитель. 2005. № 11, с. 37 – 41.
Журнал издается в Минске. http://www.radioliga.com/

Александр Черномырдин
http://chav1961.narod.ru

АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ ДЛЯ МИКРОКОНТРОЛЛЕРОВ. ЧАСТЬ 2: ПРЕРЫВАНИЯ И ВИРТУАЛЬНЫЕ ТАЙМЕРЫ

Разговор в зоопарке:
Мам, мам, это уже обезьяна?
Нет, это еще кассир!

В данной статье будут рассмотрены вопросы обработки сигналов прерывания и еще одна специальная техника программирования – виртуальные таймеры. Знатоков просят не сетовать на терминологию и упрощения.

В предыдущей статье [1] цикла были рассмотрены вопросы автоматного программирования, и было отмечено, что для реализации автоматов применяется один из двух способов – либо switch-технология, либо программа-интерпретатор конечного автомата. В обоих случаях работа автомата заключается в том, что он непрерывно проверяет, не произошло ли интересующее нас событие (терминальный символ), и при его появлении выполняет те или иные действия. Поскольку проверка терминальных символов происходит непрерывно, то следовательно ничем другим микроконтроллер в это время заниматься не может. Во многих случаях от микроконтроллера в общем-то ничего большего и не требуют, но такой способ использования аппаратуры микроконтроллера выглядит по крайней мере расточительным. Между тем в микроконтроллере есть средства, которые позволяют узнать об интересующем нас событии без непрерывного опроса сигналов. Эти средства представлены в микроконтроллере механизмом **прерываний**.

Очень упрощенно механизм прерываний работает следующим образом. В микроконтроллере есть специальные схемы, предназначенные для отслеживания состояния некоторых сигналов. К этим сигналам относятся сигналы на линиях портов, сигнал от регистра интервала интервального таймера, сигнал переполнения сторожевого таймера, сигнал окончания операции записи в ЭСППЗУ и сигнал переполнения (переопустошения) стеков данных и адресов возврата. После выполнения любой команды программы аппаратура микроконтроллера проверяет, не изменились ли значения этих сигналов. Если значение **любого** из этих сигналов изменилось, то нормальный ход выполнения программы прерывается, и управление передается на один из заранее предопределенных адресов в программе. Именно за эту особенность – прерывать нормальный ход программы, – этот механизм и назван механизмом прерываний. В микроконтроллере КР1878ВЕ1 заранее предопределенные адреса приведены в таблице:

Адрес	Когда передается управление	Маскируе

программы		мое
0001h	Отработал сторожевой таймер	Нет
0002h	Переполнение (переопустошение) стеков процессора	Нет
0003h	“Звонок” от регистра интервала интервального таймера	Да
0006h	Изменился сигнал на одной из линий порта A	Да
0007h	Изменился сигнал на одной из линий порта B	Да
000Fh	Закончена операция записи байта в ЭСППЗУ	Да

Как следует из таблицы, все заранее предопределенные адреса находятся в пределах первых 16 адресов программы. Именно поэтому они в программе никогда не используются без крайней необходимости.

Что же расположено в этих адресах? В них в большинстве случаев располагается команда перехода **jmp** на подпрограмму, которая обрабатывает соответствующий сигнал прерывания (такие подпрограммы называются **обработчиками прерываний**). Для примера с «торшером» [1] такая команда **jmp** должна располагаться по адресу 0006h, так как у нас подсоединена к порту A. При изменении сигнала на линии порта («Потягивании» или «Отпускании» веревочки), микроконтроллер автоматически передаст управление этой подпрограмме, которая определенным образом должна обработать полученный сигнал. После того, как обработчик прерываний выполнит все необходимые действия по обработке сигнала прерывания, микроконтроллер может вновь вернуться к выполнению прерванной программы. Для того, чтобы это было возможно, при прерывании программы процессор микроконтроллера запоминает в стеке адресов возврата адрес возврата в прерванную программу – внешне вся обработка прерываний выглядит так, как будто в программе вдруг «ниоткуда» возникла команда **jsr** на обработчик соответствующего прерывания. Соответственно, программа-обработчик прерывания, должна для продолжения работы прерванной программы просто выполнить команду возврата **rts**, но на самом деле для возврата из программ-обработчиков вместо команды **rts** используется ее специальный вариант – команда **rti**. Что это за команда и чем она отличается от команды **rts**?

Вернемся немного назад – туда, где был описана работа механизма прерываний. Обратите внимание, что проверка сигналов аппаратурой микроконтроллера выполняется после **каждой** команды программы. Это значит, что прерывание может произойти в **абсолютно любом** месте программы. Например, если в нашей программе есть фрагмент кода:

```
cmpl   %a0,10        ; Проверить величину переменной
jz     $1            ; Величина=10 – переход на обработку этого случая.
```

то, в принципе, ничто не мешает прерыванию возникнуть после команды **cmpl.** Тогда этот фрагмент кода выполнится процессором так:

```
cmpl   %a0,10        ; Проверить величину переменной
jsr    interrupt     ; Здесь возникло прерывание!
jz     $1            ; Величина=10 – переход на обработку этого случая.
```

Если же в обработчике прерывания будут выполнены какие-либо команды, изменяющие флаги регистра состояния процессора – а такие команды **почти наверняка** в обработчике прерываний окажутся! – то после возврата из обработчика прерываний содержимое регистра состояний будет описывать уже не результат сравнения ячейки **%a0** и числа 10, а все, что угодно! В результате из-за возникновения неожиданного прерывания вся логика работы программы окажется полностью разрушенной. Вот для

того, чтобы этого не случилось, при обработке прерываний процессор не только записывает в стек адресов возврата адрес команды, на которую необходимо вернуться (в нашем примере это – адрес команды **jz** $1), но и сохраняет в **стеке данных** содержимое **регистра состояний. Н**ебольшое отступление: такой механизм реализован далеко не во всех выпускаемых в мире микроконтроллерах, в некоторых программисту приходится в явном виде заботиться о том, чтобы сохранять регистр состояния. В нашем примере при выполнении команды **rti**, она не только возвращает управление на команду **jz** $1, но и **восстанавливает** содержимое регистра состояний из стека данных. При этом после выполнении команды **rti** регистр состояния вернется точно в то состояние, в котором он был после выполнения команды **cmpl**, и программа вообще «не заметит», что ее выполнение прерывалось! Итак, возвращаться из обработчика прерываний можно **только командой rti**. Еще один вывод: при использовании в программе механизма прерываний глубина вложенности подпрограмм **не должна превышать семи** (а не восьми), а глубина стека данных – **15** (а не 16), так как по одному (как минимум!) элементу из стека данных и адресов возврата могут в любой момент использовать обработчики прерываний.

Можно ли управлять моментом обработки прерываний, ведь прерывание, как было сказано, может возникнуть в любом месте программы? Да, можно, и вот каким образом.

Начать необходимо с того, что в микроконтроллере в регистре состояний существует специальный бит разрешения работы механизма прерываний (флаг **IE**). При включении или сбросе микроконтроллера этот бит автоматически сбрасывается, запрещая тем самым работу механизма прерываний. Сделано это для того, чтобы в момент включения, когда в программе еще не выполнены необходимые настройки портов, таймеров и т.д., посторонние сигналы не мешали работе. Если механизм прерываний необходимо включить, следует установить этот бит в логическую единицу. Делается это командой **sst** (она очень похожа по принципу работы на команду **bisl**). Если же сбросить этот бит в логический ноль (для этого существует команда **cst**), то работа механизма прерывания вновь будет заблокирована. Указанный бит разрешения действует, однако, не на все прерывания: прерывания от сторожевого таймера и от сигнала переполнения стеков будут возникать независимо от того, установлен ли этот бит или сброшен. Эти два прерывания называются **немаскируемыми** (поскольку на их возникновение не действует бит разрешения прерываний), а все остальные прерывания – **маскируемыми**. Такое исключение для немаскируемых прерываний сделано ввиду того, что они возникают при серьезных ошибках в программе (например, при «зацикливании» программы или попытке сохранить в стеке данных больше, чем он может вместить), и программа обязательно должна получить сигналы о таких серьезных ошибках! Итак, бит разрешения прерываний разрешает или запрещает работу механизма прерываний для всех маскируемых прерываний.

Для того, чтобы разрешить или запретить возникновение того или иного сигнала прерывания **индивидуально**, используются соответствующие биты в управляющих регистрах внешних устройств микроконтроллера. Такой бит есть в управляющем регистре интервального таймера (разрешает «звонок» от интервального таймера, если содержимое счетного регистра интервального таймера совпало с содержимым регистра интервала). Такой бит имеется и в управляющем регистре ЭСППЗУ (разрешает прерывание по окончанию операции записи в ЭСППЗУ), а также в управляющих регистрах портов A и B (разрешает прерывания при изменении сигналов на линиях соответствующих портов). Помимо битов в управляющих регистрах портов, для управления прерываниями от портов используются также подрегистры шесть и семь управляющих регистров портов A и B. Это делается для того, чтобы указать, от каких линий порта должно возникать прерывание (логическая единица), а от каких – нет (логический ноль).

Что произойдет, если при обработке одного прерывания возникнет другое? Все зависит от того, какое прерывание возникнет. Если немаскируемое – то программа-обработчик будет прервана и начнется обработка немаскируемого прерывания. Если же новое прерывание маскируемое – программа-обработчик первого прерывания продолжит свою работу: дело в том, что при возникновении прерывания микроконтроллер не только сохраняет адрес возврата и регистр состояния, но еще и **автоматически сбрасывает бит разрешения прерывания** в регистре состояния – как раз для того, чтобы обработчики прерывания не прерывали бесконечно друг друга. Кстати, именно поэтому бит IE и расположен в регистре состояния процессора, хотя, по большому счету, к состоянию процессора он имеет весьма и весьма отдаленное отношение. Новое прерывание при этом не пропадет, а дождется своей очереди: когда обработчик прерывания выполнит команду **rti**, завершая обработку своего прерывания, тогда ожидающее своей очереди новое прерывание и вызовет свой обработчик. Особо следует отметить, что при восстановлении регистра состояний командой **rti**, бит IE регистра состояний вернется в то состояние, в котором он был в момент возникновения только что обработанного прерывания – обработчики прерываний не отключают механизм прерываний «навсегда»!

А если случится так, что при работе микроконтроллера возникнут сразу два или более прерываний? В этом случае микроконтроллер обработает их в определенном порядке, причем в первую очередь будет обработано прерывание с меньшим предопределенным адресом. Если, например, возникнут сигналы прерываний сразу от ЭСППЗУ, интервального таймера и порта B, то первым будет обработан сигнал интервального таймера (адрес 0003h), затем – порта B (адрес 0007h), и лишь после этого – сигнал от ЭСППЗУ (адрес 000Fh). В таком случае обычно говорят о **приоритете** сигналов прерываний. Отметим, что не во всех микроконтроллерах использован столь прямолинейный способ выбора, а в некоторых достаточно мощных микроконтроллерах его можно даже настраивать программным путем.

Попытаемся переписать наш «торшер» так, чтобы вместо постоянной проверки терминальных символов воспользоваться механизмом прерываний. Рассмотрим это на примере switch-технологии (там это делается более «прозрачным» способом). Для упрощения возьмем «неправильный» торшер, в котором не учитывается дребезг контактов. В этом случае наша программа будет выглядеть следущобразм:

```
Start: jmp    begin            ; Переход к началу программы.
       nop
       nop
       nop
       nop
       nop
       jmp    int6             ; Переход на обработчик прерываний порта A.
       nop
       nop
       nop
       nop
       nop
       nop
       nop
       nop
       nop
       nop
begin: ldr    #D,18h           ; Сегмент D – управляющие регистры портов.
       movl   %d1,00011011b    ; Доступ к подрегистру 3 в режиме автоинкремента.
       movl   %d1,00000000b    ; Подрегистр 3: все линии - вводные.
```

```
        movl    %d1,00000000b       ; Подрегистр 4: режим вывода безразличен.
        movl    %d1,00000001b       ; Подрегистр 5: резистор A[0] подключен.
        movl    %d1,00000001b       ; Подрегистр 6: прерывания от A[0].
        movl    %d1,00000001b       ; Подрегистр 7: прерывания от A[0].
        movl    %d2,00011011b       ; Доступ к подрегистру 3 в режиме автоинкремента
        movl    %d2,00000001b       ; Подрегистр 3: включить линию B[0] на вывод.
        movl    %d2,00000001b       ; Подрегистр 4: режим вывода – «полноценный».
        movl    %d2,00000000b       ; Подрегистр 5: резисторы нагрузки нам не нужны.
        movl    %d2,00000000b       ; Подрегистр 6: прерывания нас не интересуют.
        movl    %d2,00000000b       ; Подрегистр 7: прерывания нас не интересуют.
        ldr     #A,40h              ; Сегмент A – ячейки ОЗУ для работы с автоматом.
        ldr     #B,00h              ; Сегмент B – адреса рабочих регистров портов и
                                    ; таймера.
        movl    %a0,0               ; В ячейке %a0 будем хранить номер текущего
                                    ;состояния автомата. В начале работы он равен
                                    ; нулю.
        movl    %b2,0               ; Выключить лампочку (исходное состояние).
        movl    %d1,00100000b       ; <1> Разрешить прерывания от порта A.
        sst     00001000b           ; <2> Включить механизм работы.
                                    ; маскируемых прерываний.
$1:     wait                        ; <3> Переход в состояние ожидания.
        jmp     $1                  ; <4> Снова переход в состояние ожидания.
```

Обратим внимание, что в начале программы появились некоторые отличия по сравнению с прежней программой, приведенной в работе [1] – в подрегистры шесть и семь порта A занесена логическая единица в бит, соответствующий линии, к которой подключена кнопка. Теперь нас будут интересовать сигналы прерываний при изменения сигнала на этой (и только на этой!) линии. Кроме того, по адресу 0006h теперь вместо команды **nop** расположена команда перехода на обработчик прерываний от порта A (именно по этому адресу она и должна находиться).

По сравнению с прежним вариантом программы, в новый добавились четыре последние команды:

1. Команда **movl %d1** устанавливает бит разрешения прерываний от порта A. Именно этим битом включается работа механизма прерываний для порта A – без него никакие изменения сигнала на линии A[0] никакого прерывания не вызовут.
2. Командой **sst** разрешается работа механизма прерываний в целом, устанавливая бит разрешения прерываний в регистре состояния процессора (формат регистра состояний был описан в статье «Первая программа для микроконтроллера»).
3. Новая, ранее не использовавшаяся команда – **wait**. Она вызывает останов работы микроконтроллера – прекращение выполнения программы. В таком состоянии микроконтроллер будет находиться до тех пор, пока не возникнет какое-нибудь прерывание. При появлении любого сигнала прерывания выполнение команды **wait** заканчивается, и начинается работа обработчика прерываний, а после него – выполнение команды, следующей за **wait**. Эту команду обычно используют тогда, когда нечем занять процессор микроконтроллера. Кроме команды **wait** в системе команд процессора есть другая, похожая на нее команда – **slp** (или **stop**). Отличие команды **slp** от команды **wait** в том, что при ее выполнении, кроме останова работы, останавливается также тактовый генератор микроконтроллера – микроконтроллер «засыпает». В таком «спящем» режиме микроконтроллер потребляет ничтожно малый ток, что бывает важно и полезно для устройств с маломощным батарейным питанием. Недостаток команды **slp** – сравнительно медленный выход микроконтроллера из

«спячки»: если для микроконтроллера был задан бит задержки запуска в регистре конфигурации, то эта задержка происходит не только при включении микроконтроллера, но и при завершении команды **slp**.

4. И, наконец, команда перехода вновь на команду **jmp**, образующая бесконечный цикл ожидания.

А вот как выглядит обработчик прерывания от порта А:

```
int6:   mial   %a1,labels      ; В рабочую ячейку %a1 загружаем адрес таблицы
        miah   %a2,labels      ; переходов  (младший байт), в %a2 – старший байт.
        add    %a1,%a0         ; Складываем адрес перехода с номером состояния.
        adc    %a2             ; Распространяем перенос/
        mtpr   #6,%a1          ; Загружаем вычисленный адрес в IR1/
        mtpr   #7,%a2
        ijmp                   ; Переходим по вычисленному адресу.
labels: jmp    cond_0          ; Таблица переходов на участки, соответствующие
        jmp    cond_1          ; состояниям автомата.
        jmp    cond_2
        jmp    cond_3
cond_0:bttl    %b1,00000001b   ; Нажата ли кнопка «торшера» (лог.0).
        jnz    exit            ; Пока нет – завершить обработчик.
        movl   %a0,1           ; Теперь новое состояние автомата 1.
        btsl   %b2,00000001b   ; Включить лампочку.
        jmp    exit
cond_1:bttl    %b1,00000001b   ; Отпущена ли кнопка «торшера» (лог.1).
        jnz    exit            ; Пока нет – завершить обработчик.
        movl   %a0,2           ; Теперь новое состояние автомата 2.
        jmp    exit
cond_2:bttl    %b1,00000001b   ; Нажата ли кнопка «торшера» (лог.0).
        jnz    exit            ; Пока нет – завершить обработчик.
        movl   %a0,3           ; Теперь новое состояние автомата 3.
        btcl   %b2,00000001b   ; Отключить лампочку.
        jmp    exit
cond_3:bttl    %b1,00000001b   ; Отпущена ли кнопка «торшера» (лог.1).
        jnz    exit            ; Пока нет – завершить обработчик.
        movl   %a0,0           ; Теперь новое состояние автомата 0.
exit:   movl   %d1,00100000b   ; <1> Снова разрешить прерывания от порта А.
        rti                    ; <2> Выход из обработчика прерываний.
        .end
```

По большому счету, новый вариант автомата отличается только тем, что вместо бесконечного цикла на метку again, как было в предыдущем варианте, выполняется переход на метку **exit**. Этой меткой отмечен участок завершения обработки прерывания, в котором следует обязательно выполнить следующие действия:

1. Сбросить сигнал прерывания, установленный процессором. Для этого, как правило, достаточно считать или записать какую-либо информацию в управляющий (или – для портов, – рабочий) регистр внешнего устройства. Подобным же образом необходимо поступать при обработке прерываний от интервального таймера и ЭСППЗУ (на этом автор остановится отдельно).
2. Возвратиться из обработчика прерываний. Напомним читателю, что возврат из обработчика прерывания выполняется **только** специальной командой **rti**.

Итак, чего же мы добились? Да, теперь микроконтроллер, как тот нетерпеливый малыш из эпиграфа, не теребит внешние устройства на предмет проверки терминального

символа – когда терминальный символ появится, ему об этом сообщат. Но ведь зато теперь процессор микроконтроллера большую часть времени просто стоит, а раньше он занимался хоть какой-то работой (хотя правильнее было бы сказать – имитацией «бурной деятельности»).

Вместо команды **wait** можно занести в программу **все, что угодно**. Например, написать некий кусок программы, наигрывающий новогоднюю песенку, которая будет непрерывно «крутиться» в том месте, где раньше была команда **wait**. Когда вы нажмете (или отпустите) кнопку «торшера», микроконтроллер на краткий миг отвлечется от наигрывания мелодии, включит или отключит лампу «торшера» и тут же продолжит мелодию. На слух вы, скорее всего, даже ничего и не заметите! В программистской литературе для такой постоянно выполняющейся в «свободное время» программы принят специальный термин – **фоновый процесс**, так как эта программа выполняется только тогда, когда микроконтроллер ничем срочным не занят (например, обработкой прерываний). Фоновый процесс не обязательно должен быть таким примитивным – в нем, например, может выполняться цифровая фильтрация сигнала, которая является одной из наиболее «затратных» задач, решаемых с помощью микроконтроллера. При этом микроконтроллер сохранит свою способность оперативно отреагировать на внешние «раздражители», каковыми в нашем примере является кнопка «торшера».

Подведем итог. Наличие механизма прерываний в микроконтроллере позволяет разгрузить аппаратуру микроконтроллера от непрерывных опросов различных сигналов, и позволяет создавать в микроконтроллере **фоновый процесс**, который может выполнять длинные неспешные вычисления, заставляя микроконтроллер, таким образом, заниматься несколькими делами одновременно. **Механизм прерываний идеальным образом ложится на технологию автоматного программирования**, так как терминальные символы автомата, как правило, имеют однозначное соответствие тем или иным сигналам прерывания микроконтроллера. Все это позволяет микроконтроллеру **управлять несколькими устройствами** или **решать несколько задач одновременно**! Еще одно немаловажное достоинство программирования «от прерываний», о котором мы уже вскользь упоминали – **экономичность в плане энергопотребления**: применив вместо команды **wait** команду **slp**, переходим в режим со сверхмалым потреблением энергии, которая будет тратить ее только на реальное управление объектом, а не на бесконечный опрос переменных. Одна маленькая батарейка на годы работы устройства – это ли не плюс!

Литература

1. *Черномырдин А.* Автоматное программирование для микроконтроллеров. Switch-технология и интерпретаторы. http://is.ifmo.ru/progeny/_autmicroc.pdf