

**Министерство образования Республики Беларусь**

**Учреждение образования  
«Гомельский государственный технический  
университет имени П. О. Сухого»**

**Кафедра «Автоматизированный электропривод»**

**В. А. Савельев, М. Н. Погуляев**

**МИКРОПРОЦЕССОРНЫЕ СРЕДСТВА  
В АВТОМАТИЗИРОВАННОМ  
ЭЛЕКТРОПРИВОДЕ**

**ПРАКТИКУМ**

**по одноименной дисциплине  
для студентов специальности 1-53 01 05  
«Автоматизированные электроприводы»  
дневной формы обучения**

**Гомель 2019**

УДК 62-83-52:004.315(075.8)  
ББК 31.291+32.965.7я73  
С12

*Рекомендовано научно-методическим советом  
факультета автоматизированных и информационных систем  
ГГТУ им. П. О. Сухого  
(протокол № 4 от 03.12.2018 г.)*

Рецензент: доц. каф. «Физика и электротехника» ГГТУ им. П. О. Сухого  
канд. техн. наук, доц. *А. В. Козлов*

**Савельев, В. А.**  
С12 Микропроцессорные средства в автоматизированном электроприводе : практикум по одноим. дисциплине для студентов специальности 1-53 01 05 «Автоматизированные электроприводы» днев. формы обучения / В. А. Савельев, М. Н. Погуляев. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 29 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: <https://elib.gstu.by>. – Загл. с титул. экрана.

Практикум содержит примеры решения типовых задач программирования микроконтроллеров по дисциплине «Микропроцессорные средства в автоматизированном электроприводе», а также задания для самостоятельной работы.

Для студентов специальности 1-53 01 05 «Автоматизированные электроприводы» дневной формы обучения.

**УДК 62-83-52:004.315(075.8)  
ББК 31.291+32.965.7я73**

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2019

# Практическая работа №1

## Подсчёт и вывод на 7-сегментные индикаторы количества внешних событий

### 1.1. Цель работы

1. Работа по внешним прерываниям.
2. Работа с динамической индикацией.
3. Работа с массивом данных.
4. Работа с макрокомандами.

### 1.2. Постановка задачи

Для схемы, представленной на рис.1, необходимо написать программу, производящую подсчёт числа нажатий кнопки SB1, подключенной ко входу внешнего прерывания INT0 (линия 2 порта D), в пределах от 0 до 99, с последующим отображением на светодиодных индикаторах методом динамической индикации.

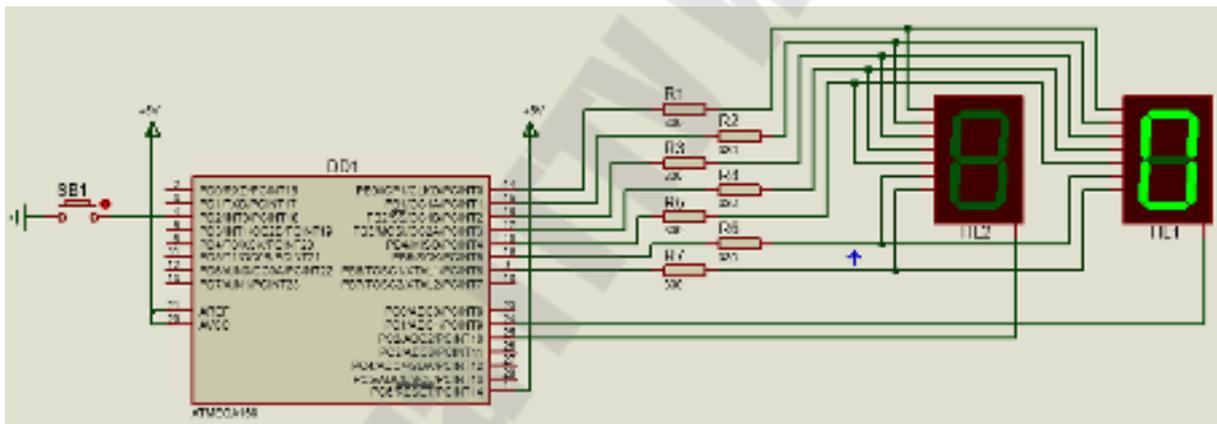


Рис.1. Схема включения индикаторов при динамической индикации

### 1.3. Пример решения задачи

Для решения поставленной задачи будем использовать внешнее прерывание по входу INT0, к которому и подключен нормально разомкнутый контакт SB1. Когда контакт разомкнут, на вход INT0 будет поступать логическая «1», а когда замкнут – логический «0». Прерывание настроим на срабатывание по переходу от логической «1» к логическому «0», то есть на момент замыкания контакта.

Для правильной работы линии PD2 (INT0) нужно настроить ее на ввод. Для того, чтобы при разомкнутом состоянии контакта на входе INT0 присутствовала стабильная логическая «1», этот вход

необходимо «подтянуть» к шине питания +5В путем включения подтягивающего резистора.

При работе с механическим контактом возникает такая проблема, как дребезг контактов. При нажатии кнопку, перед тем, как контакты плотно соприкоснутся, они будут колебаться (т.е. «дребезжать»), порождая множество срабатываний вместо одного. Аналогичная ситуация возникнет и при размыкании контакта.

Соответственно, микроконтроллер «поймает» все эти нажатия, потому что для микроконтроллера дребезг не отличим от настоящего нажатия на кнопку. Пример осциллограммы, отображающей реальный процесс переключения с логического «0» на логическую «1» приведен на рис.2.

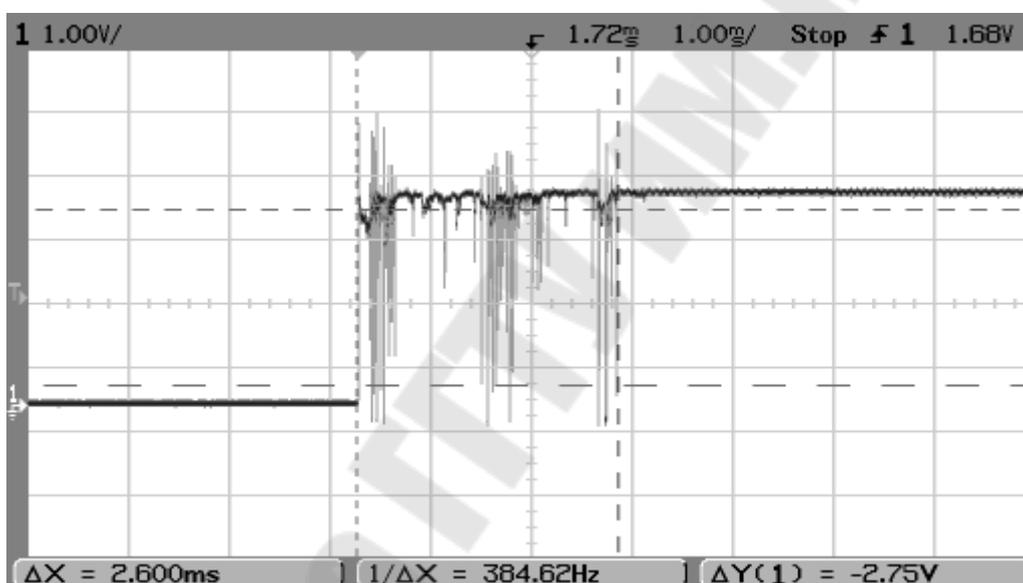


Рис.2. Осциллограмма переключения с «0» на «1»

Суть программного способа, позволяющего избежать дребезга, сводится к следующему. После нажатия кнопки переходим к обработке прерывания, внутри которого сразу же локально запрещаем данный вид прерывания и начинаем проверку состояний линии порта, куда подключена кнопка. Делаем 3...5 проверок с интервалом 0,5...1 мс. Если при каждой проверке будет получен идентичный результат, а именно логический «0», то можно считать, что контакт замкнулся.

Далее выполняем необходимые в прерывании действия.

Перед выходом из прерывания снова производим проверку, но в данном случае уже на логическую «1». Только после этого разрешаем данное прерывание и выходим из прерывания.

Внутри обработчика прерывания необходимо произвести увеличение числа нажатий кнопки на 1. Проверить, не произошло ли превышение значения 99. Произвести перевод двоичного кода числа нажатий кнопки в двоично-десятичный код.

В основной программе будем выводить информацию (BCD-код) на индикаторы. Как видно из схемы, переключение 7-сегментных индикаторов HL1, HL2 производится через порт С по линиям РС2 и РС1.

Рассмотрим текст программы.

В начале программы выполняем директивы присоединения файла описания выбранной модели контроллера (.include), присвоения удобных имен регистрам общего назначения (.def), присвоения символному имени числового значения (.set).

```
***** ЗАДАНИЕ ПАРАМЕТРОВ *****
.include "m168def.inc"           ; присоединение файла описания ATmega168

.def nul = r12                  ; присвоили имя регистру r12
.def del1 = r13                 ; присвоили имя регистру r13
.def del2 = r14                 ; присвоили имя регистру r14
.def del3 = r15                 ; присвоили имя регистру r15
.def temp = r16                 ; присвоили имя регистру r16
.def cnt = r17                  ; присвоили имя регистру r17
.def st_raz = r18               ; присвоили имя регистру r18
.def ml_raz = r19               ; присвоили имя регистру r19
.def schet = r20                ; присвоили имя регистру r20

.set INT0_pin = 2               ; номер линии INT0
;=====
```

Далее следует секция макрокоманд.

Макрокоманда «uout A, Rd» из РОН в любой РВВ не зависимо от адреса последнего (вместо команд OUT и STS).

Макрокоманда «uin Rd, A» позволяет выполнять пересылку из любого РВВ в РОН, не зависимо от адреса РВВ (вместо команд IN и LDS).

Макрокоманда «usbis A, b» - пропуск следующей команды, если бит I/O(A).b = 1. Макрос работает со всеми РВВ, в то время, как команда sbis работает только с первыми 31-м РВВ.

Макрокоманда «sbic A, b» - пропуск следующей команды, если бит I/O(A).b = 0. Макрос работает со всеми РВВ, в то время, как команда sbic работает только с первыми 31-м РВВ.

И, наконец, макрокоманда «ldil Rd, k» непосредственной записи в регистры R0-R15. Как известно, младшие 16 регистров изначально не поддерживают непосредственную адресацию.

```
***** МАКРОКОМАНДЫ *****  
.macro uout  
    .if @0 < 0x40  
        out @0, @1  
    .else  
        sts @0, @1  
    .endif  
.endm  
;=====  
.macro uin  
    .if @1 < 0x40  
        in @0, @1  
    .else  
        lds @0, @1  
    .endif  
.endm  
;=====  
.macro usbis  
    .if @0 < 0x20  
        sbis @0, @1  
    .else  
        push temp  
        uin temp, @0  
        sbrs temp, @1  
        pop temp  
    .endif  
.endm  
;=====  
.macro usbic  
    .if @0 < 0x20  
        sbic @0, @1  
    .else  
        push temp  
        uin temp, @0  
        sbrc temp, @1  
        pop temp  
    .endif  
.endm  
;=====  
.macro ldil  
    push temp  
    ldi temp, @1  
    mov @0, temp  
    pop temp  
.endm  
;=====
```

Теперь указываем сегмент памяти для хранения программы – в нашем случае это сегмент C (flash-память).

```
; FLASH =====  
.cseg          ; выбор сегмента памяти для хранения программы  
;=====
```

Далее следует секция векторов прерываний. Напомню, что таблица прерываний индивидуальна для каждого контроллера и ее можно найти в инструкции к контроллеру. По сути, нам необходимо только внешнее прерывание INT0, вектор прерывания которого имеет адрес 0x0002 в таблице прерываний. Однако, для нормальной работы программы практически всегда должно быть указано прерывание по сбросу (адрес 0x0000). Оно имеет наивысший приоритет и выполняется сразу после включения контроллера или после нажатия кнопки сброса.

В векторе прерывания прописывается команда безусловного перехода (rjmp) на метку, связанную с расположением обработчика прерывания. В нашем случае, для прерывания по сбросу это метка «reset», а для внешнего прерывания INT0 это метка «int\_0». Кстати, назвать метку «int0» не получится, потому что «int0» - это внутренне имя одного из битов регистра EIMSK.

В прерываниях, которые не используются, ставят «заглушки» в виде команд reti.

```
;***** ВЕКТОРЫ ПРЕРЫВАНИЙ *****  
.org 0  
    rjmp reset      ; сброс  
.org 2  
    rjmp int_0      ; внешний запрос на прерывание 0  
.org 4  
    reti            ; внешний запрос на прерывание 1  
.org 6  
    reti            ; прерывание 0 по изменению состояния выводов  
.org 8  
    reti            ; прерывание 1 по изменению состояния выводов  
.org 10  
    reti            ; прерывание 2 по изменению состояния выводов  
.org 12  
    reti            ; тайм-аут сторожевого таймера  
.org 14  
    reti            ; совпадение в канале А таймера/счётчика T2  
.org 16  
    reti            ; совпадение в канале В таймера/счётчика T2  
.org 18  
    reti            ; переполнение таймера/счётчика T2
```

```

.org 20      reti          ; захват фронта таймером T1
.org 22      reti          ; совпадение в канале A таймера/счётчика T1
.org 24      reti          ; совпадение в канале B таймера/счётчика T1
.org 26      reti          ; переполнение таймера/счётчика T1
.org 28      reti          ; совпадение в канале A таймера/счётчика T0
.org 30      reti          ; совпадение в канале B таймера/счётчика T0
.org 32      reti          ; переполнение таймера/счётчика T0
.org 34      reti          ; передача по SPI завершена
.org 36      reti          ; приём по USART завершён
.org 38      reti          ; регистр данных USART пуст
.org 40      reti          ; передача по USART завершена
.org 42      reti          ; окончание преобразования АЦП
.org 44      reti          ; готовность EEPROM
.org 46      reti          ; аналоговый компаратор
.org 48      reti          ; двухпроводный последовательный интерфейс
.org 50      reti          ; готовность SPM
;=====

```

Далее следует секция инициализации. Секция начинается меткой «reset:». В начале данной секции производим обнуление всех регистров общего назначения (РОН) и всех ячеек оперативной памяти (ОЗУ).

После этого выполняем инициализацию стека и портов ввода/вывода.

При настройке портов линию PD2, к которой присоединена кнопка, настраиваем на ввод и подключаем подтягивающий резистор.

Линии порта В, к которым подключены аноды индикаторов, настраиваем на вывод и подаем в порт В начальное значение, соответствующее отображению на индикаторах «00».

Линии PC2 и PC1, подключенные к общим катодам индикаторов, также настраиваем на вывод.

В этой же секции программы производим настройку внешнего прерывания. Вначале задаем режим прерывания – по спадающему фронту, а затем разрешаем прерывание INT0 локально.

И, наконец, производим глобальное разрешение прерываний.

```

;***** ИНИЦИАЛИЗАЦИЯ *****
reset:
; чистка ОЗУ
    ldi z1, low(sram_start)      ; адрес начала ОЗУ в индекс
    ldi zh, high(sram_start)
    clr temp                    ; очищаем temp
clr_ram:
    st z+, temp                 ; пишем «0» в ячейку памяти
    cpi zh, high(ramend+1)      ; достиг ли СБ индекса конца ОЗУ?
    brne clr_ram                ; продолжаем чистку
    cpi z1, low(ramend+1)       ; достиг ли МБ индекса конца ОЗУ?
    brne clr_ram                ; продолжаем чистку
    clr z1                      ; чистим индекс
    clr zh

;=====
; чистка ПОН
    ldi z1, 30
    clr zh
clr_ron:
    dec z1
    st z, zh
    brne clr_ron

;=====
; инициализация стека
    ldi temp, high(ramend)
    uout sph, temp
    ldi temp, low(ramend)
    uout spl, temp

;=====
; инициализация портов
    cbi ddrd, 2                  ; настройка линии PD2 на ввод
    sbi portd, 2                 ; подключение подтягивающего резистора
    ldi temp, 0b11111111         ; настройка порта В на вывод
    uout ddrb, temp
    ldi temp, 1<<PC2|1<<PC1      ; настройка линий PC2 и PC1 на вывод
    uout ddrc, temp
    ldi st_raz, 0b00111111       ; начальный вывод «00» на индикаторы
    ldi ml_raz, 0b00111111

;=====
; инициализация прерывания INT0
    ldi temp, 1<<ISC01|0<<ISC00 ; настройка срабатывания по спадающему
                                ; фронту (по замыканию контакта)
    uout eicra, temp
    ldi temp, 1<<INT0            ; локальное разрешение прерывания INT0
    uout eimsk, temp
    sei                          ; глобальное разрешение прерываний
;=====

```

Теперь следует секция основной программы (метка «main:»).

Не смотря на название секции, в ней может вообще ничего не происходить. Но в нашем случае, в основной программе будет производиться вывод информации на индикаторы. Поскольку в программе используется динамическая индикация, алгоритм вывода будет следующий.

1. Выводим в порт В код для старшего разряда.
2. Включаем старший разряд (PC2 = 0).
3. Задаем задержку времени.
4. Выключаем старший разряд (PC2 = 1).
5. Повторяем п.1 - п.4 для младшего разряда.

Контроллер будет всё время работать в основной программе, до того момента, пока не сработает прерывание.

```
;***** ОСНОВНАЯ ПРОГРАММА *****
main:
; индикатор ст. разряда
  out portb, st_raz          ; выводим значение ст. разряда
  ldi temp, 0<<PC2|1<<PC1   ; включаем индикатор ст. разряда
  uout portc, temp
  rcall del_100ms          ; задержка времени
  ldi temp, 1<<PC2|1<<PC1   ; отключаем индикатор ст. разряда
  uout portc, temp
;=====
; индикатор мл. разряда
  out portb, ml_raz          ; выводим значение мл. разряда
  ldi temp, 1<<PC2|0<<PC1   ; включаем индикатор мл. разряда
  uout portc, temp
  rcall del_100ms          ; задержка времени
  ldi temp, 1<<PC2|1<<PC1   ; отключаем индикатор мл. разряда
  uout portc, temp

  rjmp main                 ; завершаем цикл
;=====
```

Теперь следует секция обработчиков прерываний. В данной секции присутствует всего один обработчик (метка «int\_0:»).

Обработчик вначале сохраняет важную информацию (состояние регистров *temp* и *sreg*) в стек (*push*). Это необходимо в связи с тем, что прерывание может произойти в любой точке основной программы, а внутри прерывания состояние указанных регистров может меняться и меняется. После окончания прерывания в основную программу вернуться совершенно другие значения регистров *temp* и *sreg*, чем были до прерывания, и ход основной программы будет нарушен.

Затем происходит локальный запрет прерываний INT0, чтобы дребезг контактов не влиял на работу программы. Далее следует блок защиты от дребезга. Пять раз с интервалом 500 мкс проверяем состояние линии INT0 на равенство нулю. Если дребезга нет, переходим к основной процедуре.

В основной процедуре происходит приращение счётного регистра *cnt*, сравнение его содержимого с максимальным значением и перевод в BCD-код.

Процедура перевода состоит в том, что из значения счётного регистра, которое не превышает 99, вычитают число 10 и подсчитывают количество удавшихся вычитаний. Например, если в счётном регистре будет значение  $72_{10}$ , то число удачных вычитаний будет равно 7. Это количество и составляет значение старшего разряда, и временно (до окончания преобразования) хранится в регистре *cnt*. Остаток же составляет значение младшего разряда.

Однако, вывести BCD-коды разрядов непосредственно на индикаторы не получится. Вначале необходимо используя BCD-коды взять из таблицы соответствующие коды для 7-сегментного индикатора.

Выбор кода из массива производится путем задания адреса в массиве, равного значению метки массива («*massiv:*») плюс значение BCD-кода соответствующего разряда.

После преобразования возвращаем из стека значение регистра *cnt*.

Завершаем процедуру прерывания проверкой отпущения кнопки, разрешением прерывания INT0 и восстановлением значений регистров *temp* и *sreg*.

```

;***** ОБРАБОТЧИКИ ПРЕРЫВАНИЙ *****
int_0:
    push temp                ; сохраняем temp и SREG в стек
    uin temp, sreg
    push temp
    clr temp                 ; запрещаем прерывание INT0
    uout eimsk, temp
;=====
; защита от дребезга контактов
    ldi schet, 0x05          ; число проверок нажатия
del_int0:
    rcall del_500us
    usbic pind, INT0_pin    ; если INT0 = 0, продолжаем проверку
    rjmp end_int_0         ; иначе выходим из прерывания
    dec schet               ; уменьшаем счётчик проверок на 1
    brne del_int0          ; если счётчик не обнулится,
                           ; продолжаем проверки
;=====

```

```

; основная процедура
inc cnt ; приращение счётчика нажатий на 1
cpi cnt, 100 ; сравнение счётчика с числом 100
brne conv ; если счётчик меньше 100, то переходим
; к переводу счётчика в BCD
ldi cnt, 0 ; если счётчик переполнен,
; то его обнуляем
;=====
; преобразование двоичного кода в BCD-код
conv:
push cnt ; сохраняем cnt в стек
clr temp ; обнуляем temp
loop:
subi cnt, 10 ; вычитаем 10 из числа нажатий
brcs back ; если меньше 10, то переходим дальше
inc temp ; переносим 1 в разряд десятков
rjmp loop ; переходим к очередному вычитанию 10-и
back:
subi cnt, -10 ; возвращаем назад значение счёта,
; если остаток меньше 10
;=====
; получаем код младшего разряда ml_raz для LED
clr nul ; сброс nul
ldi z1, low(massiv*2) ; инициализация массива BCD-кодов
ldi zh, high(massiv*2)
add z1, cnt ; сложение начального адреса
; массива с числом cnt
adc zh, nul
lpm ml_raz, z ; загрузка в ml_raz значения из таблицы
;=====
; получаем код старшего разряда st_raz для LED
clr nul ; сброс nul
ldi z1, low(massiv*2) ; инициализация массива BCD-кодов
ldi zh, high(massiv*2)
add z1, temp ; сложение начального адреса массива
; с числом temp
adc zh, nul
lpm st_raz, z ; загрузка в st_raz значения из таблицы
pop cnt ; восстанавливаем cnt из стека
;=====
; перед выходом из прерывания INT0 ждем, когда установится "1"
end_int_0:
usbis pind, INT0_pin ; если INT0 = 1, продолжаем проверку
rjmp end_int_0 ; иначе, продолжаем ждать
ldi schet, 0x0a ; задаем число проверок размыкания
; контакта
del2_int0:
rcall del_500us ; длительность одной проверки
usbis pind, INT0_pin ; если INT0 = 1, продолжаем проверку
rjmp end_int_0 ; иначе, продолжаем ждать
dec schet ; уменьшаем счётчик проверок на 1
brne del2_int0 ; если счётчик не обнулился,
; продолжаем проверки

```

```

    ldi temp, 1<<INT0           ; разрешение прерывания INT0
    uout eimsk, temp
    pop temp                    ; восстанавливаем temp и SREG из стека
    uout sreg, temp
    pop temp
    reti                        ; возврат из прерывания
;=====

```

При размещении элементов в массиве необходимо помнить, что в строке должно быть чётное количество элементов (кроме последней строки), либо все элементы должны быть в одной строке.

```

;***** МАССИВЫ *****
; массив кодов десятичных чисел
massiv:
.db 0b00111111, 0b00000110    ; код 0,1
.db 0b01011011, 0b01001111    ; код 2,3
.db 0b01100110, 0b01101101    ; код 4,5
.db 0b01111101, 0b00000111    ; код 6,7
.db 0b01111111, 0b01101111    ; код 8,9
;=====

```

Секция подпрограмм задержки содержит две подпрограммы, построенные на принципе циклического уменьшения константы на 1. При этом константа задается в регистрах del1, del2, del3. Ее величина определяет длительность задержки.

```

;***** ЗАДЕРЖКИ *****
; подпрограмма задержки на 500 мкс
del_500us:
    ldil del1, 51
    ldil del2, 6
loop_1:
    dec del1
    brne loop_1
    dec del2
    brne loop_1
    ret
;=====
; подпрограмма задержки на 100 мс
del_100ms:
    ldil del1, 245
    ldil del2, 15
    ldil del3, 5
loop_2:
    dec del1
    brne loop_2
    dec del2
    brne loop_2

```

```
dec del3  
brne loop_2  
ret
```

;

Расчёт значений констант указанных регистров основан на величине необходимой задержки, значении тактовой частоты и числа тактов, необходимых для выполнения каждой команды подпрограммы. Кроме того, расчёт точных значений может быть выполнен при помощи утилиты «AVR Delay» для Android.

#### ***1.4. Задание для самостоятельной работы***

1. Модифицируйте рассмотренную выше программу для вывода на индикаторы трёхразрядного числа.
2. Дополните схему вторым кнопочным выключателем, подключенным ко входу INT1 и выполняющим функцию сброса информации, отображаемой на индикаторах.

### **Практическая работа №2**

#### **Передача байта данных по интерфейсу SPI**

##### ***2.1. Цель работы***

1. Работа с интерфейсом SPI.
2. Программирование таймера/счётчика.
3. Программирование аналого-цифрового преобразователя.
4. Работа с таблицей данных.

##### ***2.2. Постановка задачи***

Для схемы соединения двух микроконтроллеров, приведенной на рис.3, написать управляющую программу, которая позволит передавать информацию о состоянии переключателей SB1 на светодиодный индикатор VD1.

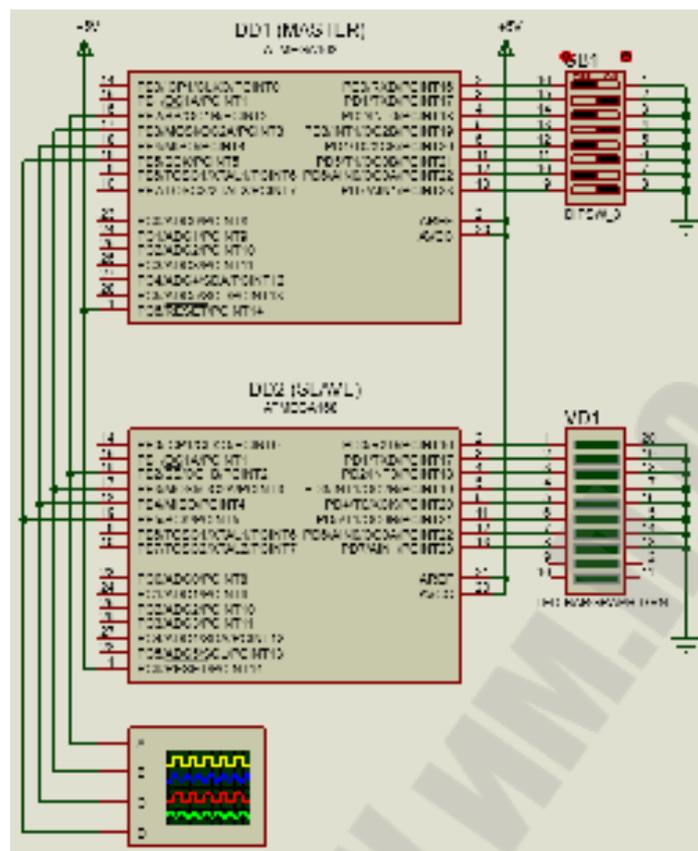


Рис.3. Схема соединения контроллеров по интерфейсу SPI

### 2.3. Пример решения задачи

Для решения поставленной задачи необходимо настроить один микроконтроллер в режим передатчика (master), а второй – в режим приемника (slave). Передатчик будет осуществлять считывание состояния подключенных к порту D переключателей, и формировать последовательный двоичный код, который будет передан приемнику.

Приемник, в свою очередь, будет выполнять приём и дешифрацию полученного кода, и формировать параллельный двоичный код, который будучи отправлен в порт D приемника, приведет к включению светодиодов индикатора VD1.

Важно помнить, что передачу информации можно контролировать двумя способами. Во-первых, через прерывания по завершению передачи через SPI (адрес 34), а во-вторых, путем контроля флага SPIF окончания передачи через SPI, который расположен в регистре SPRS. В данном примере воспользуемся вторым способом.

Рассмотрим текст программы для передающего устройства.

В начале программы выполняем директивы присоединения файла описания выбранной модели контроллера (.include) и присвоения удобных имен регистрам общего назначения (.def).

```

;***** ЗАДАНИЕ ПАРАМЕТРОВ *****
.include "m168def.inc"      ; присоединение файла описания ATmega168
.def temp = r16            ; временный регистр
.def del = r17             ; регистр задержки
;=====

```

Далее следует секция макрокоманд.

```

;***** МАКРОКОМАНДЫ *****
; Макрокоманда "uout A,Rd" пересылки из POH в любой PVB
; не зависимо от адреса PVB (вместо команд OUT и STS)
.macro uout
    .if @0 < 0x40
        out @0, @1
    .else
        sts @0, @1
    .endif
.endm
;=====

```

```

; Макрокоманда "uin Rd,A" пересылки из любого PVB в POH
; не зависимо от адреса PVB (вместо команд IN и LDS)
.macro uin
    .if @1 < 0x40
        in @0, @1
    .else
        lds @0, @1
    .endif
.endm
;=====

```

Указываем сегмент памяти для хранения программы.

```

; FLASH =====
.cseg      ; выбор сегмента памяти для хранения программы
;=====

```

Далее следует секция векторов прерываний. В данной секции активен только один вектор (reset), однако при необходимости контролировать передачу информации через прерывания можно активировать ещё и вектор прерывания по завершению передачи SPI.

```

;***** ВЕКТОРЫ ПРЕРЫВАНИЙ *****
.org 0
    rjmp reset ; переход на обработку сброса
.org 2

```

```

    reti          ; внешнее прерывание IRQ0 (INT0)
.org 4
    reti          ; внешнее прерывание IRQ1 (INT1)
.org 6
    reti          ; прерывание 0 по изменению состояния выводов (PCINT0)
.org 8
    reti          ; прерывание 1 по изменению состояния выводов (PCINT1)
.org 10
    reti          ; прерывание 2 по изменению состояния выводов (PCINT2)
.org 12
    reti          ; тайм-аут сторожевого таймера (WatchDog)
.org 14
    reti          ; срабатывание компаратора А таймера T2
.org 16
    reti          ; срабатывание компаратора В таймера T2
.org 18
    reti          ; переполнение таймера T2
.org 20
    reti          ; захват фронта таймера T1
.org 22
    reti          ; срабатывание компаратора А таймера T1
.org 24
    reti          ; срабатывание компаратора В таймера T1
.org 26
    reti          ; переполнение таймера T1
.org 28
    reti          ; срабатывание компаратора А таймера T0
.org 30
    reti          ; срабатывание компаратора В таймера T0
.org 32
    reti          ; переполнение таймера T0
.org 34
    reti          ; rjmp SPI_int      ; передача по SPI завершена
.org 36
    reti          ; приём по USART завершен
.org 38
    reti          ; регистр данных USART пуст
.org 40
    reti          ; передача по USART завершена
.org 42
    reti          ; окончание преобразования АЦП
.org 44
    reti          ; готовность EEPROM
.org 46
    reti          ; аналоговый компаратор
.org 48
    reti          ; 2-х проводной последовательный интерфейс (TWI)
.org 50
    reti          ; готовность SPM
;=====

```

Далее следует секция инициализации. Кроме обнуления регистров и памяти здесь выполняется настройка порта D передатчика на прием сигналов и к нему подключаются подтягивающие резисторы. Далее конфигурируются четыре линии порта B, задействованные в работе интерфейса SPI.

И, наконец, настраиваем сам интерфейс SPI: включаем SPI, запрещаем прерывания по SPI, устанавливаем порядок передачи данных, делаем контроллер ведущим, задаем полярность и фазу тактового сигнала, а также скорость передачи данных.

При этом порядок передачи, полярность и фаза тактового сигнала особого значения не имеют. Необходимо лишь, чтобы эти настройки совпадали у передатчика и приемника.

```

;***** ИНИЦИАЛИЗАЦИЯ *****
reset:
; чистка ОЗУ
    ldi z1,low(sram_start)    ; адрес начала ОЗУ в индекс
    ldi zh, high(sram_start)
    clr temp                  ; очищаем temp
clr_ram:
    st z+, temp              ; пишем 0 в ячейку памяти
    cpi zh, high(ramend+1)   ; достиг ли СБ индекса конца ОЗУ?
    brne clr_ram            ; продолжаем чистку
    cpi z1,low(ramend+1)    ; достиг ли МБ индекса конца ОЗУ?
    brne clr_ram            ; продолжаем чистку
    clr z1                   ; чистим индекс
    clr zh
; чистка PОН
    ldi z1, 30
    clr zh
clr_ron:
    dec z1
    st z, zh
    brne clr_ron
;=====
; инициализация стека
    ldi temp, high(ramend)
    uout sph, temp
    ldi temp, low(ramend)
    uout spl, temp
;=====
; инициализация портов
; установка порта D на ввод с включенными подтягивающими резисторами
    ldi temp, 0b11111111
    uout portd, temp
    ldi temp, 1<<PB5|1<<PB3|1<<PB2
    uout ddrb, temp          ; MOSI, SCK и SS на вывод
    sbi portb, 2             ; устанавливаем высокий уровень на SS

```

```

    sbi portb, 4          ; включаем подтяг. резистор на MISO
;=====
; инициализация SPI
    ldi temp, 0b01010011
; SPIE    0b[0]xxxxxxx - разрешение прерывания от SPI
; SPE     0bx[1]xxxxxx - включение SPI
; DORD    0bxx[0]xxxxx - порядок передачи данных
; MSTR    0bxxx[1]xxxx - ведущий или ведомый (ведущий)
; CPOL    0bxxxx[0]xxx - полярность тактового сигнала (положительная)
; CPHA    0bxxxxx[0]xx - фаза тактового сигнала (по нараст. фронту)
; SPR1:0  0bxxxxxxx[11] - скорость передачи (fclk/4)
    uout spcr, temp
;sei          ; разрешение прерываний

```

Основная программа посредством порта D осуществляет приём информации в параллельном двоичном коде с блока микропереключателей SB1. Далее следует отправка полученного байта информации по последовательному каналу, начинающаяся установкой низкого уровня на линии SS контроллера. При этом процесс передачи контролируется не по прерываниям, а путём проверки флага окончания отправки – 7-го бита регистра SPSR. Как только флаг окончания передачи будет установлен, на линии SS необходимо установить высокий уровень. Далее процедура отправки пакета может быть повторена.

```

;***** ОСНОВНАЯ ПРОГРАММА *****
main:
    uin temp, pind          ; чтение порта D
    cbi portb, 2           ; устанавливаем низкий уровень на SS
    uout spdr, temp        ; отправляем 8 бит
; ожидаем окончания отправки
wait:
    uin temp, spsr         ; проверяем флаг окончания отправки
    sbrs temp, 7
    rjmp wait
    sbi portb, 2           ; устанавливаем высокий уровень на SS
; пауза между пакетами
    rcall del_50us        ; пауза между пакетами
    rjmp main             ; переходим к отправке нового пакета

;***** ЗАДЕРЖКИ *****
; подпрограмма задержки на 50 мкс
del_50us:
    ldi del, 134
loop:
    dec del
    brne loop
    nop
    ret

```

### ***2.4. Задание для самостоятельной работы***

1. Напишите программу для контролера–приёмника.
2. Напишите эту же программу, используя систему прерываний для контроля передачи информации.
3. Используйте АЦП для задания передаваемой величины и ЦАП для вывода полученного результата на принимающей стороне.

## **Практическая работа №3**

### **Управление униполярным шаговым двигателем**

#### ***3.1. Цель работы***

1. Управление шаговым двигателем.
2. Программирование таймера/счётчика.
3. Программирование аналого-цифрового преобразователя.
4. Работа с таблицей данных.

#### ***3.2. Постановка задачи***

Для схемы управления униполярным шаговым двигателем (ШД), приведенной на рис.4, написать управляющую программу, которая позволит регулировать частоту «шагания» двигателя с помощью потенциометра, подключенного ко входу аналого-цифрового преобразователя (АЦП) микроконтроллера.

#### ***3.3. Пример решения задачи***

Анализируя схему, приведенную на рис.4, можно отметить, что каждая полуобмотка ШД подключена к общему проводу схемы через один из транзисторов VT1...VT4, а средние точки обеих обмоток соединены с источником питания +24В. Таким образом, для того, чтобы ток протекал по полуобмотке ШД необходимо открыть соответствующий транзистор путём подачи на его базу от микроконтроллера логической «1».

За счёт определённой последовательности отпираания транзисторов можно задать различные режимы работы ШД: полношаговые «1-1» или «2-2» и полушаговый «1-2».

Таким образом, для управления ШД необходимо на каждом его шаге выставлять на линиях PD0, PD1, PD5, PD6 порта D определенный двоичный код.

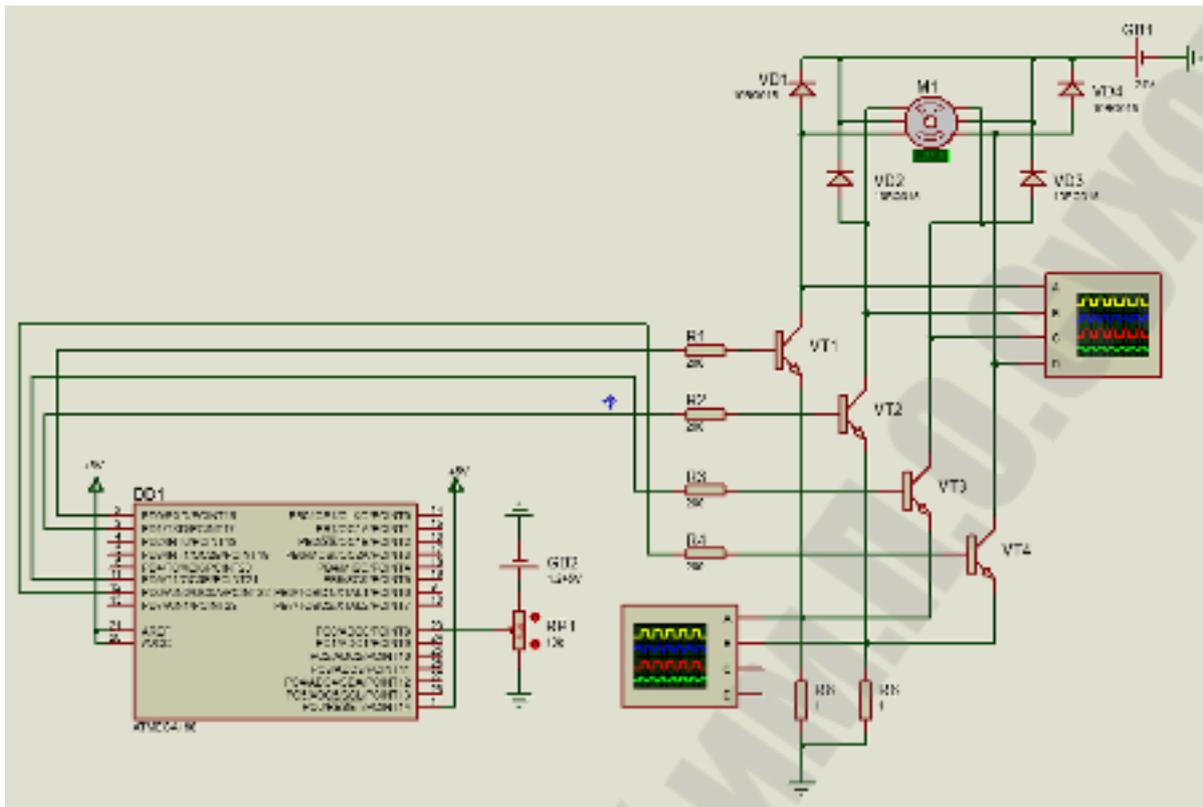


Рис.4. Схема управления униполярным шаговым двигателем

Частота «шагания» ШД будет определяться частотой смены двоичных кодов на линиях порта D. Эту частоту необходимо задавать с помощью потенциометра, подключенного ко входу АЦП микроконтроллера.

Рассмотрим текст программы.

В начале следует стандартный набор процедур: присоединение файла описания контроллера, присвоение имен регистрам, макрокоманды, определение сегмента памяти для хранения программы.

```

;***** ЗАДАНИЕ ПАРАМЕТРОВ *****
#include "m168def.inc"           ; присоединение файла описания
ATmega168

.def temp = r16                 ; временный регистр
.def temp2 = r17                ; временный регистр
.def cnt = r18                  ; счётчик состояний

;***** МАКРОКОМАНДЫ *****
.macro uout
    .if @0 < 0x40
        out @0, @1
    .else

```

```

        sts @0, @1
    .endif
.endm
;=====
.macro uin
    .if @1 < 0x40
        in @0, @1
    .else
        lds @0, @1
    .endif
.endm
;=====
.macro usbis
    .if @0 < 0x20
        sbis @0, @1
    .else
        push temp
        uin temp, @0
        sbrs temp, @1
        pop temp
    .endif
.endm
;=====
.macro usbic
    .if @0 < 0x20
        sbic @0, @1
    .else
        push temp
        uin temp, @0
        sbrc temp, @1
        pop temp
    .endif
.endm
;=====
.macro ldil
    push temp
    ldi temp, @1
    mov @0, temp
    pop temp
.endm

; FLASH =====
.cseg          ; выбор сегмента памяти для хранения программы

```

В секции векторов прерывания указываем метки перехода для прерываний по сбросу «reset», по срабатыванию компаратора канала А таймера/счётчика Т2 «timer2\_int» и по окончании преобразования АЦП «adc\_int».

```

;***** ВЕКТОРЫ ПРЕРЫВАНИЙ *****
.org 0

```

```

.rjmp reset          ; переход на обработку сброса
.org 2
reti                ; внешнее прерывание IRQ0 (INT0)
.org 4
reti                ; внешнее прерывание IRQ1 (INT1)
.org 6
reti                ; прерывание 0 по изменению состояния выводов
.org 8
reti                ; прерывание 1 по изменению состояния выводов
.org 10
reti                ; прерывание 2 по изменению состояния выводов
.org 12
reti                ; тайм-аут сторожевого таймера (WatchDog)
.org 14
.rjmp timer2_int    ; срабатывание компаратора А таймера Т2
.org 16
reti                ; срабатывание компаратора В таймера Т2
.org 18
reti                ; переполнение таймера Т2
.org 20
reti                ; захват фронта таймера Т1
.org 22
reti                ; срабатывание компаратора А таймера Т1
.org 24
reti                ; срабатывание компаратора В таймера Т1
.org 26
reti                ; переполнение таймера Т1
.org 28
reti                ; срабатывание компаратора А таймера Т0
.org 30
reti                ; срабатывание компаратора В таймера Т0
.org 32
reti                ; переполнение таймера Т0
.org 34
reti                ; передача по SPI завершена
.org 36
reti                ; приём по USART завершен (USART Rx)
.org 38
reti                ; регистр данных USART пуст
.org 40
reti                ; передача по USART завершена (USART Tx)
.org 42
.rjmp adc_int       ; окончание преобразования АЦП
.org 44
reti                ; готовность EEPROM
.org 46
reti                ; аналоговый компаратор
.org 48
reti                ; 2-х проводной последовательный интерфейс
.org 50
reti                ; готовность SPM
;=====

```

Секция инициализации начинается с процедур обнуления РОН и ОЗУ, затем следует стандартная процедура загрузки указателя стека и настройка портов ввода/вывода. Кроме того, в данном примере, происходит отключение цифрового буфера входа ADC0 (хотя это не обязательно), а также загружается начальный адрес таблицы состояний ШД в индексный регистр Z.

```

;***** ИНИЦИАЛИЗАЦИЯ *****
reset:
; чистка ОЗУ
    ldi z1,low(sram_start)      ; адрес начала ОЗУ в индекс
    ldi zh, high(sram_start)
    clr temp                    ; очищаем temp
clr_ram:
    st z+, temp                ; пишем 0 в ячейку памяти
    cpi zh, high(ramend+1)     ; достиг ли СБ индекса конца ОЗУ?
    brne clr_ram              ; продолжаем чистку
    cpi z1,low(ramend+1)       ; достиг ли МБ индекса конца ОЗУ?
    brne clr_ram              ; продолжаем чистку
    clr z1                     ; чистим индекс
    clr zh
;=====
; чистка РОН
    ldi z1, 30
    clr zh
clr_ron:
    dec z1
    st z, zh
    brne clr_ron
;=====
; инициализация стека
    ldi temp, high(ramend)
    uout sph, temp
    ldi temp, low(ramend)
    uout spl, temp
;=====
; инициализация портов
    usbis didr0, ADC0D        ; отключение цифрового буфера линии ADC0
    ldi temp, 0b01100011     ; установка PD0, PD1, PD5, PD6 на вывод
    uout ddrd, temp

    ldi z1, low(full_2*2)    ; загрузка адреса таблицы состояний
    ldi zh, high(full_2*2)
;=====

```

Секция инициализации продолжается настройкой режима работы таймера/счётчика T2. Таймер настраивается на режим «сброс при совпадении», то есть частота прерываний будет зависеть от константы, записанной в регистр сравнения таймера. Здесь также задается де-

литель тактовой частоты, и разрешаются прерывания по совпадению в канале А.

Далее следует инициализация АЦП. Выбираем аналоговый вход ADC0, смещение результата вправо, разрешаем работу АЦП, однократное преобразование без автоматического перезапуска. Обязательно разрешаем прерывания от АЦП и выбираем делитель, такой, чтобы частота работы АЦП составляла от 50 до 200 кГц.

После всего разрешаем прерывания глобально.

```
=====
; инициализация таймера Timer_2
    ldi temp, 0b01111111    ; начальная установка скважности
    uout ocr2a, temp
    ldi temp, 0b00000010
; COM2A1:0 0b[00]xxxxxx - режим формирования выходного сигнала OC2A
; COM2B1:0 0bxx[00]xxxx - режим формирования выходного сигнала OC2B
; -        0bxxxx[00]xx - зарезервирован
; WGM21:0  0bxxxxxx[10] - режим работы таймера (СТС)
    uout tccr2a, temp
    ldi temp, 0b00000111
; FOC2A    0b[0]xxxxxxxx - принудительное изменение состояния OC2A
; FOC2B    0bx[0]xxxxxxxx - принудительное изменение состояния OC2B
; -        0bxx[00]xxxx - зарезервированы
; WGM22    0bxxxx[0]xxx - режим работы таймера (быстрая ШИМ)
; CS22:0   0bxxxxxx[111] - выбор тактового источника с предделителем
;                                     частоты 8МГц/8=1МГц
; При тактовой частоте 8МГц с включенным делителем на 8 имеем частоту
1МГц/1024/256=3,8 Гц
    uout tccr2b, temp
    ldi temp, 0b00000010
; -        0b[00000]xxx - зарезервировано
; OCIE2B   0bxxxxxx[0]xx - разрешение прерывания по совпадению в канале В
; OCIE2A   0bxxxxxx[1]x  - разрешение прерывания по совпадению в канале А
; CS22:0   0bxxxxxxxx[0] - разрешение прерывания по переполнению
    uout tmsk2, temp

=====
; инициализация АЦП
    ldi temp, 0b00000000
; REFS1:0  0b[00]xxxxxx - выбор ИОН (напряжение AREF)
; ADLAR    0bxx[0]xxxxxx - смещение 10-битного результата вправо
; -        0bxxx[0]xxxx - зарезервирован
; MUX3:0   0bxxxx[0000] - выбор аналогового входа (ADC0)
    uout admux, temp
    ldi temp, 0b11001011
; EDEN     0b[1]xxxxxxxx - разрешение работы АЦП
; ADSC     0bx[1]xxxxxx - однократное преобразование
; ADFR     0bxx[0]xxxxxx - разрешение автоматического запуска АЦП
; ADIF     0bxxx[0]xxxx - сброс флага прерывания АЦП
; ADIE     0bxxxx[1]xxx - разрешение прерывания от АЦП
```

```
; ADPS2:0 0bxxxxx[011] - выбор делителя частоты 8МГц/8/8=125 кГц
uout adcsra, temp
```

```
sei ; глобальное разрешение прерываний
```

```
=====
```

Теперь следует секция основной программы (метка «main:»), которая представляет собой пустой цикл.

```
;***** ОСНОВНАЯ ПРОГРАММА *****
main:
```

```
rjmp main
```

```
=====
```

Основные действия происходят в обработчиках прерываний. В прерывании от АЦП происходит чтение результата преобразования АЦП. Не смотря на то, что результат в данном примере не превышает 8 бит, необходимо читать оба байта регистра ADC (сначала младший, затем старший). Младший байт результата копируется в регистр сравнения таймера.

В прерывании от таймера/счётчика T2 производим считывание байта, определяющего состояние ШД на текущем шаге, из массива и отправляем его в порт D. При этом ведём подсчёт считанных из массива байт (состояний). Если все состояния из массива выбраны, сбрасываем адрес элементов массива в индексном регистре Z и счётчик числа элементов массива *cnt*. В конце этого прерывания запускаем новое преобразование АЦП.

```
;***** ОБРАБОТЧИКИ ПРЕРЫВАНИЙ *****
```

```
; обработка прерывания от АЦП
```

```
adc_int:
```

```
uin temp, adcl ; вывод младшего байта АЦП в регистр temp
```

```
uin temp2, adch ; вывод старшего байта АЦП в регистр temp2
```

```
uout ocr2a, temp ; запись результата из АЦП в регистр
```

```
; сравнения OCR2A Timer_2
```

```
reti
```

```
=====
```

```
; обработка прерывания от Timer_2
```

```
timer2_int:
```

```
lpm temp, z+ ; чтение очередного элемента таблицы
```

```
uout portd, temp ; состояний и отправка в порт D
```

```
inc cnt ; увеличение счётчика
```

```
cpi cnt, 4 ; сравнение счётчика с числом состояний
```

```
; двигателя
```

```
brne end_t2_int ; окончание прерывания, если все
```

```
; состояния пройдены
```

```

    ldi z1, low(full_2*2) ; загрузка адреса таблицы состояний
    ldi zh, high(full_2*2)
    clr cnt ; сброс счётчика
end_t2_int:
    ldi temp, 0b11001011 ; запускаем новое преобразование от АЦП
    uout adcsra, temp
    reti
;=====

```

Массив состояний ШД для полношагового режима «2-2» (режим так называется, потому что на каждом шаге включены две полуобмотки) представляет собой четыре байта, в каждом из которых присутствуют только 2 логические «1». Эти «единицы» соответствуют полуобмоткам ШД, которые необходимо подключить на текущем шаге.

```

;***** МАССИВЫ *****
; таблица состояний ШД
full_2:
.db 0b00000011, 0b00100010, 0b01100000, 0b01000001

```

### ***3.4. Задание для самостоятельной работы***

1. Напишите программу для управления биполярным шаговым двигателем с использованием драйвера L298.
2. Напишите программу, позволяющую сделать двигателю заданное число шагов в одном из направлений, а затем столько же в противоположном.
3. Напишите программу, в которой шаговый двигатель будет совершать один шаг при каждом замыкании кнопочного выключателя, соединенного со входом INT0.

## Литература

1. Евстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя – М.: Издательский дом «Додэка-XXI», 2007. – 592 с.
2. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы – М.: Издательский дом «Додэка-XXI», 2004. – 288с.
3. Бродин В.Б. Микроконтроллеры. Архитектура, программирование, интерфейс / В.Б. Бродин, М.И. Шагурин. - М.: ЭКОМ, 1999. - 400 с.
4. Водовозов А.М. Микроконтроллеры для систем автоматики: учебное пособие / А.М. Водовозов. - Вологда: ВоГТУ, 2002. - 123 с.
5. Рюмик С.М. 1000 и одна микроконтроллерная схема. Книга 1 / С.М. Рюмик. - М.: Додэка-XXI, 2012. - 356 с.
6. Хартов В.Я. Микроконтроллеры AVR. Практикум для начинающих: Учебное пособие / В.Я. Хартов. - М.: МГТУ им. Баумана, 2012. - 280 с.

## Содержание

Практическая работа №1	
Подсчёт и вывод на 7-сегментные индикаторы количества.....	3
Практическая работа №2	
Передача байта данных по интерфейсу SPI .....	14
Практическая работа №3	
Управление униполярным шаговым двигателем.....	20
Литература .....	28

**Савельев Вадим Алексеевич  
Погуляев Михаил Николаевич**

**МИКРОПРОЦЕССОРНЫЕ СРЕДСТВА  
В АВТОМАТИЗИРОВАННОМ  
ЭЛЕКТРОПРИВОДЕ**

**Практикум  
по одноименной дисциплине  
для студентов специальности 1-53 01 05  
«Автоматизированные электроприводы»  
дневной формы обучения**

Подписано к размещению в электронную библиотеку  
ГГТУ им. П. О. Сухого в качестве электронного  
учебно-методического документа 10.09.19.

Пер. № 4Е.  
<http://www.gstu.by>