Лабораторная работа 3

Применение микроконтроллеров для создания сложных систем (Операции ввода-вывода на МК ATmega16)

Цель работы

Изучить основные приёмы построения сложных систем, управляемых внешними сигналами и выдающих заданные сигналы согласно заложенному алгоритму.

Освоить методику проектирования сложной системы, создания на языке ассемблера программы управления системой и реализации системы на микроконтроллере ATmega16. Научиться программировать операции ввода-вывода на ATmega16.

Сложные системы, управляющиеся внешними сигналами и выдающие заданные сигналы согласно заложенному алгоритму, наиболее удобно реализовывать с помощью использования микроконтроллеров. Причина такого удобства состоит в универсальности физической реализации сложной системы – все системы строятся фактически по одной электронной схеме, различие между ними состоит только в разном программном обеспечении. Модификация алгоритма работы системы не требует радикального физического изменения физической структуры системы, необходима только смена программы, на основе которой система функционирует.

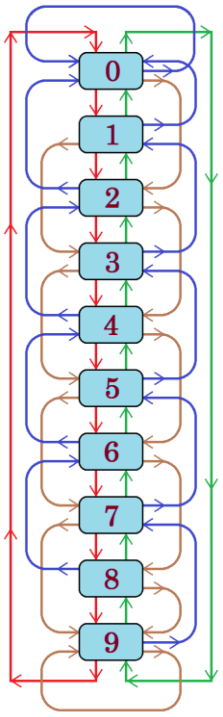


Рис. 1. Пример графа сложной системы

Алгоритм работы любой сложной системы проще всего описать графом. Система имеет несколько состояний (узлы графа), переходы от одного состояния к другому вызываются теми или иными внешними сигналами (ветви графа). Если предполагается несколько внешних сигналов (импульс таймера, сигнал датчика, нажатие кнопки и т.д.), текущее состояние системы может сменяться тем или иным состоянием, то есть ветви графа должны иметь разные цвета.

Рассмотрим в качестве примера сложную систему, граф которой приведён на рис. 1. Система имеет 10 состояний – 0, 1, …, 8, 9. При нажатии кнопки 1 (красные ветви графа) состояние системы увеличивается на 1, причём из состояния 9 система переходит в состояние 0. При нажатии кнопки 2 (зелёные ветви

графа) состояние системы уменьшается на 1, причём из состояния 0 система переходит в состояние 9. При нажатии кнопки 3 (коричневые ветви графа) состояние системы увеличивается на 2, причём из состояния 8 система переходит в состояние 9, а состояние 9 не изменяется. При нажатии кнопки 4 (синие ветви графа) состояние системы уменьшается на 2, причём из состояния 1 система переходит в состояние 0, а состояние 0 не изменяется.

Проектирование любой сложной системы следует начинать с построения схемы алгоритма его работы.

Схема алгоритма работы (блок-схема) рассматриваемой сис-темы приведена на рис. 2. Перио-дически производится считывание двоичного четырёхразрядного чис-ла **K**, которое задаётся четырьмя независимыми ключами. Если счи-танное значение не отличается от предыдущего (**K = K_{old}**), никаких действий не производится. Если оно отличается от предыдущего, но равно 0 (ни один ключ не замкнут), то также не производится никаких действий, кроме запоминания зна-чения 0. Если же при предыдущем значении **K**, равном 0, считывается новое значение, отличное от 0 (это может быть только один замкнутый ключ, а все остальные – разомкну-ты), начинается изменение состоя-ния системы. При нажатии первого ключа (**K = 0001**) текущее состояние системы увеличивается на 1, за исключением перехода 9→0 (крас-ные стрелки на рис. 1), при нажатии второго ключа (**K = 0010**) текущее

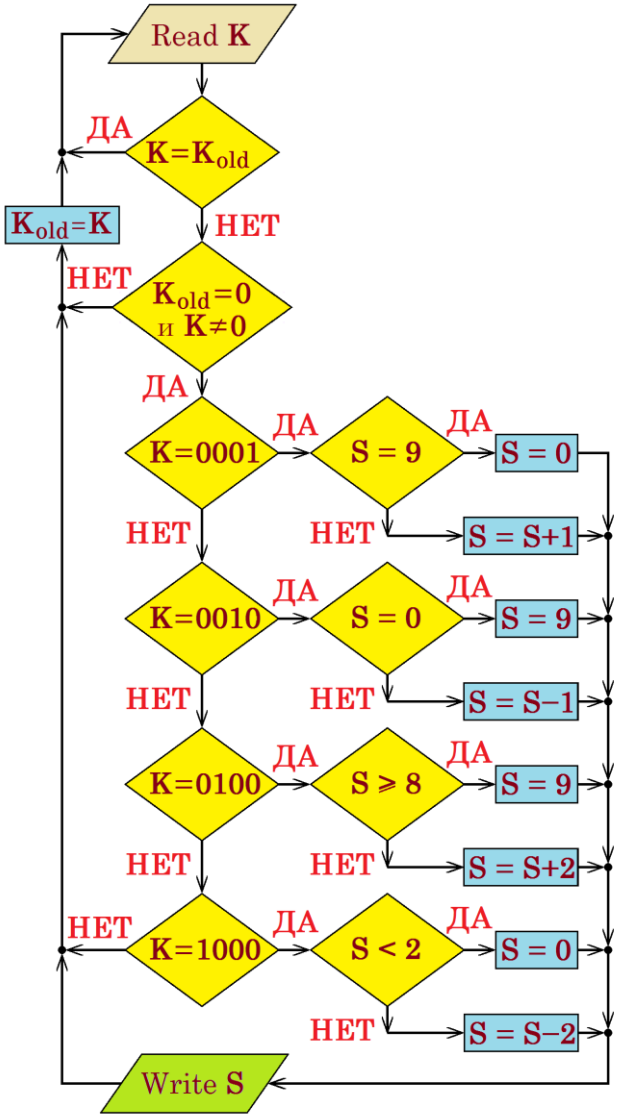


Рис. 2. Схема алгоритма работы системы

состояние системы уменьшается на 1, за исключением перехода 0→9 (зелёные стрелки на рис. 1), при нажатии третьего ключа (**K = 0100**) текущее состояние системы увеличивается на 2, за исключением переходов 8→9 и 9→9 (коричневые стрелки на рис. 1), при нажатии четвёртого ключа (**K = 1000**) текущее состояние системы уменьшается на 2, за исключением переходов 1→0 и 0→0 (синие стрелки на рис. 1).

Таким образом, алгоритм, реализованный по схеме рис. 2, должен обеспечивать функционирование системы, заданной графом рис. 1.

После определения схемы алгоритма следует перейти к созданию программы на языке ассемблера микроконтроллера ATmega16.

При создании сложной системы необходимо каким-либо образом организовать ввод информации в систему и вывод информации из системы. Для этой цели все микроконтроллеры снабжаются портами ввода-вывода. Микроконтроллер ATmega16 имеет 4 8-разрядных порта ввода-вывода: A, B, C и D. Каждый разряд каждого порта программируется независимо (в т.ч. некоторые разряды одного порта могут работать на ввод, а некоторые – на вывод информации), если содержимое порта используется как 8-разрядное двоичное число (например, для арифметических операций), то самым младшим является разряд Pn0 (n = A, B, C, D) а самым старшим – разряд Pn7.

Для управления разрядами каждого из регистров микроконтроллер ATmega16 имеет три регистра: PINn, PORTn и DDRn (n = A, B, C, D) разряды которых соответствуют разрядам портов.

Если $DDRn_k = 1$, то k-ый разряд порта n работает на вывод, разряд $PORTn_k$ может принимать значения 0 либо 1, которые можно считывать или использовать для управления внешними устройствами.

Если $DDRn_k = 0$, то k-ый разряд порта n работает на ввод, и в него можно с помощью внешних устройств записывать 0 либо 1. Для ATmega16 это означает – соединить $DDRn_k$ с 0 В либо с 5 В. Если сигнал 0 В или 5 В подаётся с какого-либо источника, ввод информации не вызывает никаких трудностей. Если же 0 В либо 5 В подаются на $DDRn_k$ посредством коммутации $DDRn_k$ с источниками постоянного напряжения, то при этом возникает обычная проблема – как ATmega16 будет воспринимать $DDRn_k$, «висящий в воздухе» (Z-состояние). В этом случае, кроме того, возникает сильная зависимость потенциала входа $DDRn_k$ от внешних помех.

Для устранения возможности возникновения Z-состояния в микроконтроллере ATmega16 используются подтягивающие резисторы (pull-up resictors), через которые рассматриваемый вход имеет постоянное соединение с постоянным потенциалом (обычно это 0 В или 5 В). В ATmega16 при $DDRn_k = 0$ k-ый разряд порта n в случае $PORTn_k = 0$ имеет Z-состояние, а в случае $PORTn_k = 1$ – соединяется через подтягивающий резистор (10 кОм) с источником питания. Поэтому вводить информацию в разряд $PORTn_k$ посредством коммутирующих устройств следует одним из способов, показанных на рис. 3.

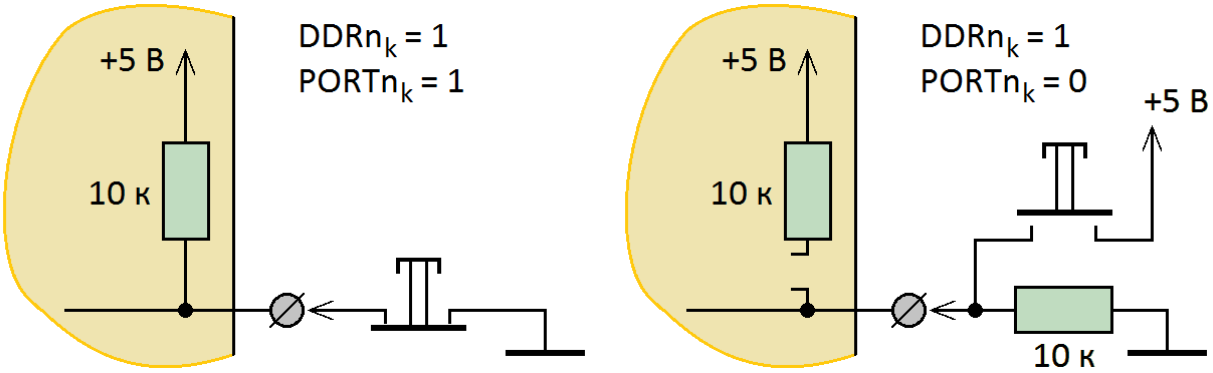


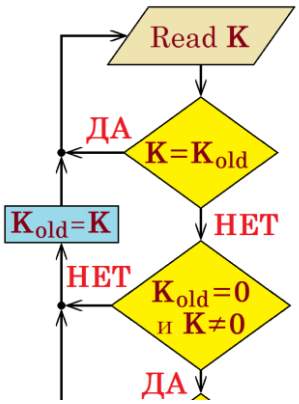
Рис. 3. Варианты кнопочного ввода информации в ATmega16

Таким образом, прежде всего необходимо назначить один из портов микроконтроллера на ввод информации и один из портов – на вывод. Это реализуется посредством команд:

```
ldi   temp,0x00     ; 0 --> temp
out   ddrd,temp     ; Назначаем порт rd на ввод (00000000 --> ddrd)
ldi   temp,0xFF     ; 0xff --> temp
out   ddrb,temp     ; Назначаем порт rb на вывод (11111111 --> ddrb)
out   portd,temp    ; Подключаем подтягивающие резисторы (11111111 --> portd)
```

Затем можно приступать непосредственно к программированию схемы алгоритма рис. 2:

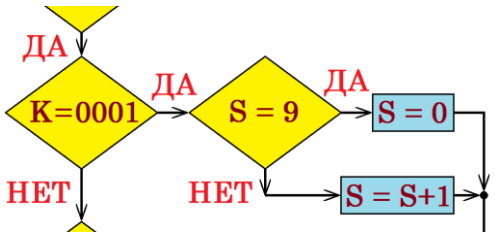
```
ldi     kold, 0x00   ; 0--->kold
ldi     s___, 0x00   ; 0--->s___
out     portb, s___  ; Вывели s___(=0)
read:                ;
in      k___, pind   ; Считали содержимое порта pd (--->k___)
cp      k___, kold   ; Сравнили k___ и kold
breq    read         ; Если k___=kold, read
tst     kold         ; Проверили kold
brne    remem        ; Если kold!=0, remen
tst     k___         ; Проверили k___
breq    remem        ; Если k___=0, remem
```



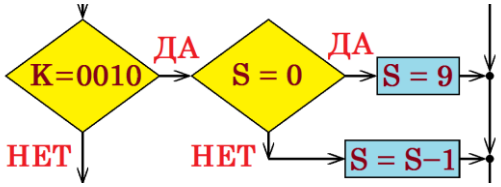
По завершению этого набора команд переменная **K** (k___) имеет одно из значений: 1 (0001_2), 2 (0010_2), 4 (0100_2) или 6 (1000_2). Кроме этого, в переменной $\mathbf{K_{old}}$ (kold) хранится предыдущее значение **K**.

Различные значения переменной **K** обрабатываются в различных блоках программы:

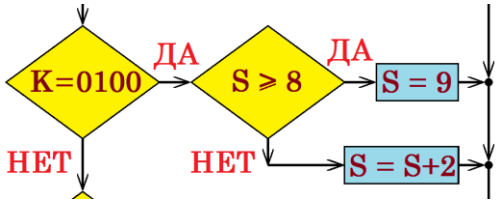
```
lbl1:                ;
cpi     k___, 0x01   ; Сравнили k___ и 1
brne    lbl2         ; Если k___!=1, lbl2
cpi     s___, 0x09   ; Сравнили s___ и 9
brne    lbl_1        ; Если s___!=9, lbl_1
clr     s___         ; 0--->s___
jmp     print        ; Перешли на Вывод
lbl_1:               ;
inc     s___         ; s___+1--->s___
jmp     print        ; Перешли на Вывод
```



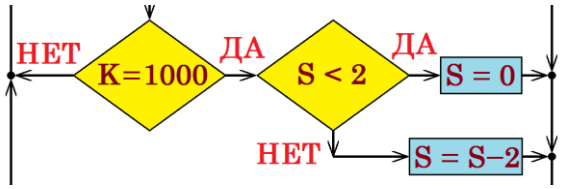
```
lbl2:                ;
cpi     k___, 0x02   ; Сравнили k___ и 2
brne    lbl4         ; Если k___!=2, lbl4
cpi     s___, 0x00   ; Сравнили s___ и 0
brne    lbl_2        ; Если s___!=0, lbl_2
ldi     s___, 0x09   ; 9--->s___
jmp     print        ; Перешли на Вывод
lbl_2:               ;
dec     s___         ; s___-1--->s___
jmp     print        ; Перешли на Вывод
```




```
lbl4:                ;
cpi     k___, 0x04   ; Сравнили k___ и 4
brne    lbl8         ; Если k___!=4, lbl8
cpi     s___, 0x08   ; Сравнили s___ и 8
brlo    lbl_4        ; Если s___<8, lbl_4
ldi     s___, 0x09   ; 9--->s___
jmp     print        ; Перешли на Вывод
lbl_4:               ;
inc     s___         ; ...
inc     s___         ; s___+2--->s___
jmp     print        ; Перешли на Вывод
```




```
lbl8:                ;
cpi     k___, 0x08   ; Сравнили k___ и 8
brne    remem        ; Если k___!=8, remem
cpi     s___, 0x02   ; Сравнили s___ и 2
brsh    lbl_8        ; Если s___>=2, lbl_8
ldi     s___, 0x00   ; 0--->s___
jmp     print        ; Перешли на Вывод
lbl_8:               ;
dec     s___         ; ...
dec     s___         ; s___-2--->s___
```

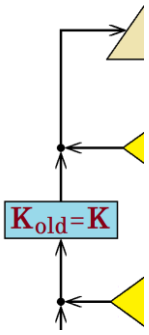


После завершения одного из блоков программы сформированное состояние системы **S** (s___) выводится на порт вывода:

```
print:
out     portb,s___ ; Вывели s___ в порт pb
jmp     remem
```

Кроме того, текущее состояние переменной **K** запоминается в переменной $\mathbf{K_{old}}$:

```
remem:               ;
mov     kold, k___   ; Записали k___ в kold
jmp     read         ; Вернулись в read
```



Таким образом, полный текст программы на языке ассемблера ATMEGA16 выглядит следующим образом:

```
.include "m16def.inc" ; подключение библиотеки для работы с ATmega16
.list ; включение листинга
.def temp=r16 ; определение главного рабочего регистра
.def kold=r17
.def k___=r18
.def s___=r19
;-----------------------------------------
.cseg ; выбор сегмента программного кода
.org 0 ; установка текущего адреса на ноль
;-----------------------------------------
ldi temp,0x80 ; выключение компаратора
out acsr,temp
;-----------------------------------------
ldi temp,0x00       ; 0 --> temp
out ddrd,temp       ; Назначаем порт rd на ввод (00000000 --> ddrd)
ldi temp,0xFF       ; 0xff --> temp
out ddrb,temp       ; Назначаем порт rb на вывод (11111111 --> ddrb)
out portd,temp      ; Подключаем подтягивающие резисторы (11111111 --> portd)
;-----------------------------------------
ldi         kold, 0x00      ; 0--->kold
ldi         s___, 0x00      ; 0--->s___
out         portb, s___     ; Вывели s___(=0)
;-----------------------------------------
read:                       ;
in          k___, pind      ; Считали содержимое порта pd (--->k___)
cp          k___, kold      ; Сравнили k___ и kold
breq        read            ; Если k___=kold, read
tst         kold            ; Проверили kold
brne        remem           ; Если kold!=0, remen
tst         k___            ; Проверили k___
breq        remem           ; Если k___=0, remem
jmp         lbl1            ;
;-----------------------------------------
remem:                      ;
mov         kold, k___      ; Записали k___ в kold
jmp         read            ; Вернулись в read
;-----------------------------------------
lbl1:                       ;
cpi         k___, 0x01      ; Сравнили k___ и 1
brne        lbl2            ; Если k___!=1, lbl2
cpi         s___, 0x09      ; Сравнили s___ и 9
brne        lbl_1           ; Если s___!=9, lbl_1
clr         s___            ; 0--->s___
jmp         print           ; Перешли на Вывод
lbl_1:                      ;
inc         s___            ; s___+1--->s___
jmp         print           ; Перешли на Вывод
;-----------------------------------------
lbl2:                       ;
cpi         k___, 0x02      ; Сравнили k___ и 2
brne        lbl4            ; Если k___!=2, lbl4
cpi         s___, 0x00      ; Сравнили s___ и 0
brne        lbl_2           ; Если s___!=0, lbl_2
ldi         s___, 0x09      ; 9--->s___
jmp         print           ; Перешли на Вывод
lbl_2:                      ;
dec         s___            ; s___-1--->s___
jmp         print           ; Перешли на Вывод
;-----------------------------------------
```

```
lbl4:                           ;
cpi         k___, 0x04          ; Сравнили k___ и 4
brne        lbl8                ; Если k___!=4, lbl8
cpi         s___, 0x08          ; Сравнили s___ и 8
brlo        lbl_4               ; Если s___<8, lbl_4
ldi         s___, 0x09          ; 9--->s___
jmp         print               ; Перешли на Вывод
lbl_4:                          ;
inc         s___                ; ...
inc         s___                ; s___+2--->s___
jmp         print               ; Перешли на Вывод
;-----------------------------------------
lbl8:                           ;
cpi         k___, 0x08          ; Сравнили k___ и 8
brne        remem               ; Если k___!=8, remem
cpi         s___, 0x02          ; Сравнили s___ и 2
brsh        lbl_8               ; Если s___>=2, lbl_8
ldi         s___, 0x00          ; 0--->s___
jmp         print               ; Перешли на Вывод
lbl_8:                          ;
dec         s___                ; ...
dec         s___                ; s___-2--->s___
;-----------------------------------------
print:                          ;
out         portb, s___         ; Вывели s___ в порт pb
jmp         remem
```

Написанную программу следует оттранслировать с помощью программы AVR_Studio. Созданный программой hex-файл необходимо связать с микроконтроллером ATmega16 в системе PROTEUS. При сборке разработанной системы в программе PROTEUS удобно использовать для отображения состояния системы семисегментный индикатор типа 7SEG-BCD, а в качестве ключей для подачи информации на вход микроконтроллера – кнопки BUTTON, как это показано на рис. 4.

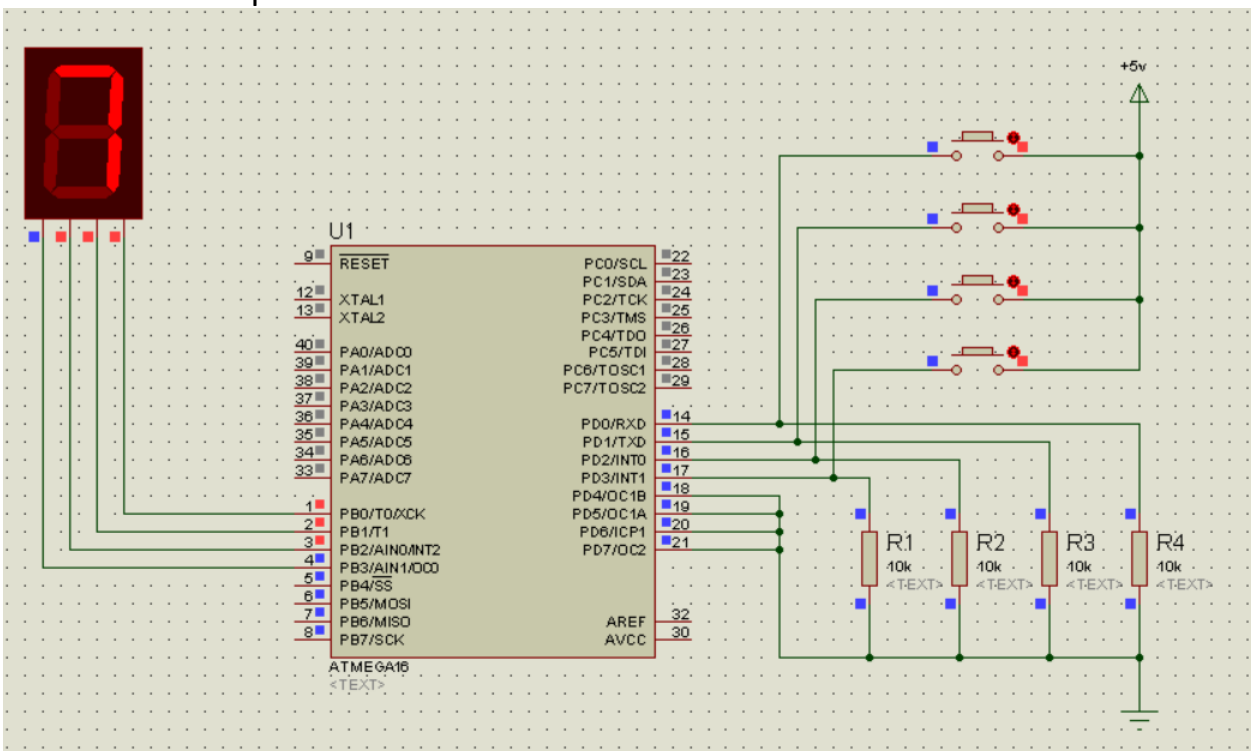


Рис. 4. Реализация разработанной системы в программе PROTEUS

Следует отметить, что кнопка BUTTON имеет по умолчанию следующие параметры: сопротивление в нажатом состоянии – 100 мОм, сопротивление в ненажатом состоянии – 100 МОм, то есть при нажатии на неё, действительно наступает замыкание, как это и требуется в правой схеме рис. 3. В левой же схеме того же рисунка необходимо, чтобы нажатие на кнопку вызывало размыкание цепи. Это можно легко обеспечить, переназначив значения этих сопротивлений – например, 100 МОм в нажатом состоянии и 100 мОм – в ненажатом.

Порядок выполнения работы

1. Получить у преподавателя задание – граф переходов системы

2. Составить схему алгоритма работы системы.

3. Написать на языке ассемблера ATmega16 программу, реализующую алгоритм п.2.

4. С помощью программы AVR_Studio осуществить трансляцию программы (получить hex-файл).

5. Собрать в системе PROTEUS схему на основе микроконтроллера ATmega16, реализующую разрабатываемую систему.

6. Убедиться в правильном функционировании разработанной системы.

Содержание отчёта

Отчёт должен содержать:

1. Задание лабораторной работы – граф переходов разрабатываемой системы.

2. Алгоритм работы системы

3. Программу для микроконтроллера ATmega16, реализующую разработанный алгоритм.

4. Функционирование разработанной системы должно быть продемонстрировано в программе PROTEUS.