Лабораторная работа № 10

# Организация клавиатуры для микроконтроллера

Цель работы:

1. Изучение принципов организации клавиатуры для микроконтроллеров.
2. Программирование клавиатуры для МК ATmega8535.

Оборудование и программное обеспечение:

1. Лабораторный макет MK8535 в составе: модуль базовый МБ8535 +

модуль сменный МС03.

1. Кабель соединительный для программирования МК.
2. Персональный компьютер.
3. Интегрированная среда программирования CodeVisionAVR.

# Подготовка к работе

Структурная схема лабораторного макета приведена на рис. 66. Матричная клавиатура S1…S12 подключена к порту C. 12 кнопок сгруппированы в 3 строки по 4 кнопки (4 столбца).

К порту A подключен модуль жидкокристаллического индикатора. Он понадобится для отображения символов нажатых клавиш. Процедуры для обеспечения работы ЖКИ можно взять из лабораторной работы №7

«Подключение модуля ЖКИ».

Создать проект в **CodeWizardAVR**. Линии 0…6 порта A настро- ить на вывод. Линии 4, 5, 6 порта C также настроить на вывод. Линии 0…3 порта C настроить на ввод, причем включить внутренние «подтя- гивающие» резисторы МК для обеспечения уровня 1 на входах считыва- ния столбцов.

Ниже приведен пример программы для МК ATmega8535, где символы, соответствующие нажатым клавишам, выводятся на ЖКИ. В нем опущены те строки, автоматически сформированные CodeWiz- ardAVR, которые не существенны для данного примера.

Функция **ScanCode(void)** осуществляет построчный опрос клавиатуры и возвращает номер нажатой клавиши. Если возвращено значение 0, это означает, что нет нажатых клавиш. Значения 1…12 соот- ветствуют номерам (скан-кодам) нажатых кнопок S1…S12, как показано на рис. 66.

# void UnlockLCD(void) // Разблокировка шины

**{**

**PORTA.5 = 0; // ^WR2 = 0**

**PORTA.4 = 0; // WR1 = 0**

**PORTA.6 = 0; // A0 = 0**

**PORTA |= 0x0F; // адрес = 0x0F PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**PORTA.6 = 1; // A0 = 1**

**PORTA.0 = 1; // DB0 = 1**

**PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**}**

# void ClrLCD(void) // Очистка индикатора

**{**

Рис. 66. Структурная схема лабораторного макета

Процедура **ShowLCD()**обеспечивает передачу 10 символов в мо- дуль ЖКИ. Наборы бит, необходимые для изображения символов, содер- жатся в массиве **BitSet[12]**. В качестве индексов этого массива используются коды символов из **LCDStr[10]**.

Процедура **ClrLCD()**очищает табло индикатора путем заполне- ния его пробелами. Для разблокировки шины ЖКИ служит процедура **UnlockLCD()**.

**#include <mega8535.h> #include <delay.h>**

**unsigned char LCDStr[10];**

**unsigned char i;**

**UnlockLCD(); // Разблокировка шины**

**PORTA.5 = 0; // ^WR2 = 0**

**PORTA.4 = 0; // WR1 = 0**

**// Запись 10 пробелов с адреса 0 PORTA.6 = 0; // A0 = 0**

**PORTA &= 0xF0; // адрес = 0 PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**PORTA.6 = 1; // A0 = 1**

**PORTA &= 0xF0; // DB0..3 = 0**

**for (i=1; i<=20; i++)**

**{**

**PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**}**

**}**

# void ShowLCD(void)

**{**

**static unsigned char BitSet[12]= { 0xEE,0x60,0x2F,0x6D,0xE1, // 0..4**

**0xCD,0xCF,0x68,0xEF,0xED, // 5..9**

**0x00,0x01 }; // пробел, минус**

**unsigned char i;**

**UnlockLCD(); // Разблокировка шины**

**PORTA.5 = 0; // ^WR2 = 0**

**PORTA.4 = 0; // WR1 = 0**

**PORTA.6 = 0; // A0 = 0**

**PORTA &= 0xF0; // адрес = 0 PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**PORTA.6 = 1; // A0 = 1**

**for (i=0; i<10; i++)**

**{**

**PORTA &= 0xF0; // DB0..3 = 0**

**// DB0..3 = младшая тетрада**

**PORTA |= BitSet[LCDStr[i]] & 0x0F; PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**PORTA &= 0xF0; // DB0..3 = 0**

**// DB0..3 = старшая тетрада**

**PORTA |= BitSet[LCDStr[i]] >> 4;**

**PORTA.4 = 1; delay\_us(1);**

**PORTA.4 = 0; // Импульс WR1**

**delay\_us(1); // задержка 1 мкс**

**}**

**}**

# unsigned char ScanCode(void)

**{**

**if ((PINC & 0x0F) != 0x0F) // Если нажато -**

**{ofs = 4; c = PINC & 0x0F;} // зафиксировать PORTC.5 = 1; PORTC.4 = 0; // Выбор 3-й строки if ((PINC & 0x0F) != 0x0F) // Если нажато -**

**{ofs = 8; c = PINC & 0x0F;} // зафиксировать**

**switch (c)**

**{**

# case 0b1110 : return ofs + 4; // Вычислить case 0b1101 : return ofs + 3; // номер case 0b1011 : return ofs + 2; // нажатой case 0b0111 : return ofs + 1; // клавиши

**default : return 0; // Ничего не нажато**

**}**

**}**

# void main(void)

**{**

**unsigned char i, sc;**

**PORTA = 0x00; DDRA = 0x7F; // Port A init. PORTC = 0x0F; // Вкл. «подтягивающие» резисторы DDRC = 0x70; // Port C initialization**

**for (i=0; i<10; i++)**

**LCDStr[i] = 10; // 10 пробелов**

**ClrLCD();**

**while (1)**

**{**

# // Если нажато - отобразить

**if (sc = ScanCode())**

**{**

# LCDStr[9] = sc - 1; ShowLCD();

**}**

**}**

**}**

**unsigned char c = 0x0F, ofs; PORTC.6 = 0; PORTC.5 = 1;**

**PORTC.4 = 1; // Выбор 1-й строки**

**if ((PINC & 0x0F) != 0x0F) // Если нажато -**

**{ofs = 0; c = PINC & 0x0F;} // зафиксировать**

**PORTC.6 = 1; PORTC.5 = 0; // Выбор 2-й строки**

# Задание

1. Создать проект и реализовать рассмотренный выше пример программы.
2. Создать новый проект и организовать ввод с клавиатуры и отображе- ние на индикаторе многоразрядных чисел.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  | **EERIE** | **EEMWE** | **EEWE** | **EERE** |
| R | R | R | R | R/W | R/W | R/W | R/W |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# EEPROM-ПАМЯТЬ ДАННЫХ В МИКРОКОНТРОЛЛЕРАХ AVR

Современные микроконтроллеры имеют в своем составе энерго- независимую память данных (EEPROM-память). В такой памяти удобно сохранять константы, поправочные коэффициенты, данные текущей настройки и т.п. информацию, которую иногда нужно изменять. В МК ATmega8535 объем этой памяти составляет 512 байт. EEPROМ-память расположена в своем адресном пространстве и так же, как и ОЗУ, орга- низована линейно. Для работы с EEPROM-памятью используются три регистра ввода/вывода:

* **EEAR** – регистр адреса EEPROM-памяти (EEPROM Address Register)
* **EEDR** – регистр данных EEPROM-памяти (EEPROM Data Register)
* **EECR** – регистр управления EEPROM-памяти (EEPROM Control Register)

16-разрядный регистр адреса EEAR физически размещается в двух регистрах ввода/вывода EEARH:EEARL, расположенных по адре- сам $1F($3F) и $1Е($3Е) соответственно. В этот регистр загружается адрес ячейки EEPROM-памяти, к которой будет производиться обраще- ние. Регистр адреса доступен как для записи, так и для чтения. При этом в регистре EEARH задействуются только младшие разряды (количество задействованных разрядов зависит от объема EEPROM-памяти). Неза- действованные разряды регистра EEARH доступны только для чтения и содержат «0».

Регистр данных EEDR расположен по адресу $1D ($3D). В этот ре- гистр помещают данные, которые должны быть записаны в EEPROM, а при чтении в этом регистре находятся данные, считанные из EEPROM.

Регистр управления EECR расположен по адресу $1С ($ЗС). Этот регистр используется для управления доступом к EEPROM- памяти. Формат этого регистра показан на рис. 67, а его описание при- ведено в табл. 27.

Рис. 67. Формат регистра EECR микроконтроллера ATmega8535

*Таблица 27*

Разряды регистра EECR микроконтроллера ATmega8535

|  |  |  |
| --- | --- | --- |
| Разряд | Название | Описание |
| 7…4 | – | Не используются, читаются как «0» |
| 3 | EERIE | **Разрешение прерывания от EEPROM.** Этот разряд управляет генерацией прерывания, возникающего при за- вершении цикла записи в EEPROM. Если этот разряд уста- новлен в «1», прерывания разрешены (если флаг I реги- стра SREG также установлен в «1»). При сброшенном раз- ряде EEWE (см. далее в таблице) прерывание генерируется постоянно |
| 2 | EEMWE | **Управление разрешением записи в EEPROM.** Состоя- ние этого разряда определяет функционирование флага раз- решения записи EEWE. Если данный разряд установлен в «1», то при записи в разряд EEWE «1» происходит запись данных в ЕЕРRОМ. В противном случае установка ЕЕWЕ в «1» не про- изводит никакого эффекта. После программной установки разряд ЕЕMWЕ сбрасывается аппаратно через 4 машинных цикла |
| 1 | EEWE | **Разрешение записи в EEPROM.** При установке этого разряда в «1» происходит запись данных в ЕЕРRОМ (если EEMWE = 1) |
| 0 | EERE | **Разрешение чтения из EEPROM.** После установки этого разряда в «1» выполняется чтение данных из EEPROM. По окончании чтения этот разряд сбрасывается аппаратно |

# Запись в EEPROM-память

Процедура записи одного байта в EEPROM-память состоит из сле- дующих этапов:

1. Дождаться готовности EEPROM к записи данных (ждать пока не сбро- сится флаг EEWE регистра EECR).
2. Дождаться завершения записи во FLASH-память программ (ждать пока не сбросится флаг SPMEN регистра SPMCR).
3. Загрузить байт данных в регистр EEDR.
4. Загрузить адрес ячейки EEPROM в регистр EEAR.
5. Разрешить запись, установив в 1 флаг EEMWE регистра EECR.
6. Запустить запись байта, установив в 1 разряд EEWE регистра EECR не позже 4-х машинных циклов после установки флага EEMWE. После установки этого разряда процессор пропускает 2 машинных цикла перед выполнением следующей инструкции.

Второй пункт введен из-за того, что запись в EEPROM-память не может выполняться одновременно с записью в FLASH-память. Поэтому перед выполнением записи в EEPROM-память следует убедиться, что про- граммирование FLASH-памяти завершено. Если в программе отсутствует загрузчик, т.е. микроконтроллер никогда не изменяет содержимое памяти программ, шаг может быть пропущен.

Процесс обращения к EEPROM-памяти контролируется внутрен- ним RС-генератором. Соответственно, длительность цикла зависит от ча- стоты этого генератора, напряжения питания и составляет 8,4 мс для МК ATmega8535. По окончании цикла записи разряд EEWE аппаратно сбра- сывается, после чего программа может начать запись следующего байта.

При записи в EEPROM могут возникнуть некоторые проблемы, вызванные прерываниями:

* + При возникновении прерывания между 5-м и 6-м этапами описанной последовательности, запись в EEPROM будет сорвана**,** т.к. за время обработки прерывания флаг EEMWE успеет сброситься в 0.
	+ Если в подпрограмме обработки прерывания, возникшего во время за- писи в EEPROM-память, также происходит обращение к ней, то будет изменено содержимое регистров адреса и данных EEPROM. В резуль- тате первая запись (прерванная) будет сорвана.

Во избежание описанных проблем настоятельно рекомендуется запрещать все прерывания (сбрасывать бит I регистра SREG) на время выполнения пунктов 2...5 описанной выше последовательности.

Ниже приведен пример реализации функции записи в EEPROM-

память на языке С (управление прерываниями не производится):

# void EEPROM\_write(unsigned int Addr, unsigned char Data)

**{**

**// Ждать завершения предыдущей записи**

**while (EECR & 0x02);**

**EEAR = Addr; // Установить адрес**

**EEDR = Data; // Установить данные**

**EECR |= 0x04; // Разрешить запись в EEPROM EECR |= 0x02; // Начать запись в EEPROM**

**}**

# Чтение из EEPROM-памяти

Операция чтения из EEPROM-памяти проще операции записи. Нужно задать адрес ячейки в EEAR, произвести чтение, а затем забрать данные из EEDR.

Перед выполнением операции чтения также необходимо прокон- тролировать состояние флага EEWE. Дело в том, что пока выполняется операция записи в EEPROM-память (флаг EEWE установлен), нельзя выполнять ни чтения EEPROM-памяти, ни изменения регистра адреса.

После загрузки требуемого адреса в регистр EEAR необходимо установить в «1» разряд EERE регистра EECR. Когда запрошенные дан- ные будут помешены в регистр данных EEDR, произойдет аппаратный сброс этого разряда. Однако следить за состоянием разряда EERE для определения момента завершения операции чтения не требуется, т.к. операция чтения из EEPROM всегда выполняется за один машинный цикл. Кроме того, после установки разряда EERE в «1» процессор про- пускает 4 машинных цикла перед началом выполнения следующей инструкции.

С учетом сказанного функция чтения из EEPROM может быть реализована следующим образом:

# unsigned char EEPROM\_read(unsigned int Addr)

**{**

**// Ждать завершения предыдущей записи**

**while (EECR & 0x02);**

**EEAR = Addr; // Установить адрес**

**EECR |= 0x01; // Выполнить чтение**

**return EEDR; // Забрать данные из EEDR**

**}**

В приведенном ниже примере в EEPROM-память по адресу 0 будет записано число 0x55. Затем считанное из этой же ячейки EEPROM значение выведено в порт A (будут светиться нечетные светодиоды, подключенные к порту A):

# void main(void)

**{**

# DDRA = 0xFF; // Настр. порт A на вывод EEPROM\_write(0x0000, 0x55); // Записать в EEPROM PORTA = EEPROM\_read(0x0000); // Читать из EEPROM

**}**

Также при использовании EEPROM-памяти необходимо соблю- дать некоторые меры предосторожности, чтобы избежать повреждения данных, находящихся в ней. К сожалению, у EEPROM-памяти есть один недостаток: при снижении напряжения питания хранящиеся в ней дан- ные могут быть повреждены. Это может произойти по двум причинам:

* + Если напряжение питания ниже некоторой величины, запись в EEPROM будет произведена некорректно.
	+ Микроконтроллер сам может выполнять команды некорректно, если напряжение питания будет ниже нормы.

Чтобы избежать повреждения данных, хранящихся в EEPROM,

достаточно воспользоваться одним из трех следующих решений:

* + Удерживать микроконтроллер в состоянии сброса все время, пока напряжение питания находится ниже нормы. Для этого следует ис- пользовать встроенный детектор пониженного напряжения питания (Brown-out Detector, BOD).
	+ Удерживать микроконтроллер в «спящем» режиме (Power Down) пока напряжение питания находится ниже нормы. Поскольку в этом режи- ме микроконтроллер не может выполнять никаких команд, такое решение эффективно защищает служебные регистры EEPROM от непреднамеренной записи.
	+ Хранить константы во FLASH-памяти программ, если они не должны меняться во время работы программы. Микроконтроллер не может самостоятельно производить запись во FLASH-память. Соответствен- но, при понижении напряжения питания ее содержимое не будет повреждено.