УДК 004.383.3:004.312.466

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА НА ЯЗЫКЕ СИ

Е.М. Баранова, В.М. Глаголев, К.М. Щепакин

Рассмотрена разработка программного обеспечения специализированного вычислительного устройства на основе микроконтроллера AVR ATmega8. Особое внимание уделяется особенностям программирования для микроконтроллеров на Си и разрабатывается алгоритм управления, позволяющий принимать и выводить аналоговые и цифровые данные, производя их простейшую обработку.

Ключевые слова: микроконтроллер, программирование на языке Си, алгоритм управления, побитовые операции, логические операции, цифровой сигнал, аналоговый сигнал.

Разрабатываемое устройство вычисляет две независимые величины E и F по следующим формулам: $E = \frac{1}{n}\sum_{i=1}^{n} A_i$; $F = \frac{1}{n}\sum_{i=1}^{n} B_i$, где A и F – аналоговое сигналы, B и E – цифровые сигналы, n – количество введенных на данный момент значений. Принципиальная схема устройства, для которого разрабатывается алгоритм управления представлена на рисунке 1.

Для программирования микроконтроллера можно использовать язык Си, либо, ассемблер. Программирование производилось на языке *ANSI C* ввиду его наглядности и неизменности стандартов.

Программирование микроконтроллеров на Си имеет ряд особенностей, связанных со спецификой управления реальным объектом. Во-первых, программа для микроконтроллера никогда не должна заканчиваться, а значит помимо главной программы *main()*, обязательным является наличие главного бесконечного цикла *while(1)* внутри *main()*. Таким образом, код будет выполняться пока на микроконтроллер подается питание.

Второй особенностью работы с микроконтроллерами на языке Си являются побитовые операции, которые встречаются очень редко при классическом программировании. Все «переключатели» и переменные микроконтроллера находятся внутри 8-битных регистров и очень часто бывает необходимо взаимодействовать только с одним битом регистра, а все остальные оставить, как было и не учитывать.

Для выполнения побитовых операций в языке Си существуют следующие операторы:

- **&** (Побитовое И).
- | (Побитовое ИЛИ).
- ~ (Побитовое НЕ, инверсия каждого бита в двоичном числе).
- << (Побитовый сдвиг влево, «дописывание» нулей справа).

– |= (Выполнение побитового ИЛИ над значениями переменных справа и слева с присваиванием результата в переменную слева).
– **&=** (Выполнение побитового И над значениями переменных справа и слева с присваиванием результата в переменную слева).
– ^= (Выполнение побитового исключающего ИЛИ над значениями переменных справа и слева с присваиванием результата в переменную слева).

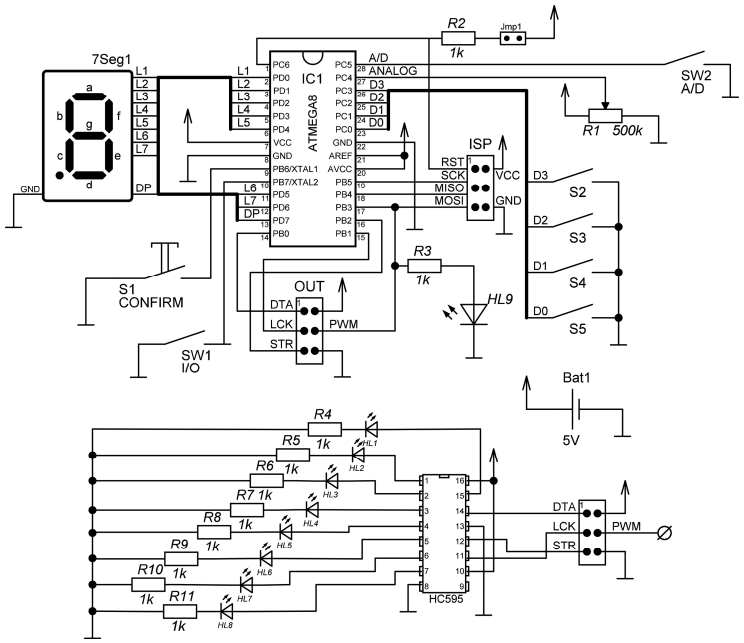

Рис. 1. Принципиальная схема устройства

Из этих операций составляются шаблонные конструкции для управления отдельными битами регистра, представленные в таблице 1 (из них многие начинающие программисты пишут интуитивно понятные директивы **#define** и функции).

Таблица 1

Шаблонные конструкции для управления битами

`REG |= 1<<N`	Установить бит N в регистре REG (сделать бит единицей)
`REG &= ~(1<<N)`	Очистить бит N в регистре REG (сделать бит нулем)
`REG ^= 1<<N`	Инвертировать бит N в регистре REG (1→0, 0→1)
`REG & (1<<N)`	Проверить установлен ли бит N в регистре REG

Стандартные порты ввода-вывода организованы таким образом, что каждому выводу микроконтроллера соответствует один бит в трех 8-

битных регистрах. В микроконтроллерах AVR, порты (регистры, связанные с физическими контактами) именуются буквами (A,B,C,…), а контакты цифрами от 0 до 7. Каждому порту соответствуют следующие регистры:

– **DDRx** – регистр направления передачи сигнала (0 – Выход, 1 – Вход);
– **PORTx** – регистр значения (для выхода) и подтяжки (для входа);
– **PINx** – регистр состояния (реальный логический уровень на контакте).

Подтяжка (*Pull-up*) – Подключение контакта к питанию или земле через высоко-омный (около 40 КОм) резистор.

Таблица 2 иллюстрирует вышесказанное в наглядной форме.

Таблица 2

Управление контактами ввода-вывода

DDRx.n		**PORTx.n**		**PINx.n**
Чтение / Запись		Чтение / Запись		Только чтение
0	Вход	0	Высоко-омный вход	0 + Влияние схемы
		1	Подтяжка к **VCC**	1 + Влияние схемы
1	Выход	**PORTx.n** = **PINx.n**		

Для примера предположим следующее содержимое регистров порта **B**:

```
(DDRB  == 0b11111000 &&
 PORTB == 0b01010011 &&
 PINB  == 0b01010110)
```

Биты нумеруются начиная с нуля справа налево. Из представленного состояния битов можно сделать следующие выводы:

– Контакты 0, 1 и 2 являются входами, а остальные выходами.
– На контакте 0 низкий логический уровень не смотря на подтяжку к **VCC**, вероятно он физически подключен к земле.
– На контакте 1 высокий логический уровень при учете подтяжки. Он точно НЕ подключен к земле.
– На контакте 2 низкий логический уровень с выключенной подтяжкой. – Он точно не подключен к земле.
– Контакты с 3 по 7 исправно выводят заданные значения.

Весь дальнейший код будет написан с использованием принципов описанных выше и привычной логики языка Си.

В начале любой программы необходимо инициализировать порты. Ниже приведен соответствующий фрагмент кода:

```
void ports_init(){
   DDRB = 0b00111111;
/*PB2..0–Сдвиговый регистр | PB6–Кнопка CONFIRM | PB7-Переключатель I/O*/
   PORTB = 0b11000000;
   DDRC = 0;      // C – Полностью вход
/*PC0..3–Цифровой вход | PC4-АЦП(Аналоговый вход) | PC5-Переключатель A/D*/
   PORTC = 0b101111;
   DDRD=0xFF;  // D - дисплей
   PORTD=0xFF;            // Выключаем дисплей
}
```

Далее, необходимо настроить АЦП, фрагмент кода представлен ниже:

```
void ADC_init(){
   ADMUX = 0<<REFS1 | 0<<REFS0;        //Источник опорного напряжения - AREF
   ADMUX |= 1<<ADLAR;          // 10 бит в 16-битном регистре смещены влево
   ADMUX |= 0<<MUX3 | 1<<MUX2 | 0<<MUX1 | 0<<MUX0; //работаем с ADC4
   ADCSRA = 1<<ADEN;                    //Включение АЦП
   ADCSRA |= 1<<ADIE; //Включение прерывания от окончанию преобразования
   ADCSRA |= 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0; //Частота АЦП = XTAL/128
   sei();  //Разрешить все прерывания
}
```

Изменять сразу несколько бит одной строчкой можно с помощью операции побитового ИЛИ. Сначала устанавливаются в 1 все необходимые биты (у каждого бита есть свое имя), потом полученное число претерпевает побитовое ИЛИ с имеющимся значением регистра и результат записывается в регистр. По сути, предыдущий фрагмент кода можно было описать тремя строчками, но читаемость снизилась бы существенно. Далее, необходимо настроить ШИМ. Используем второй 8-битный счетчик, который выводит ШИМ на контакт **OC2**. формируем следующую строчку:

```
TCCR2 = 1<<CS20 | 1<<COM21 | 1<<WGM21 | 1<<WGM20;
```

Для работы со сдвиговым регистром необходимо обратиться к data-sheet'у на него, откуда можно получить следующий алгоритм записи числа в регистр:

– Установить на контакте **DTA** нужное значение.
– Поднять и опустить логический уровень на контакте **STR**.
– Если записаны все 8 битов, перейти к пункту 4. Иначе, перейти к пункту 1.
– Поднять и опустить логический уровень на контакте **LCK**.

На основании алгоритма, можно составить функцию для записи числа в регистр:

```
void SHR_Write(uint8_t n){
        for (uint8_t i=0; i<8; i++) {
                        if (n & 0x80)       PORTB |=  1<< PB0;                  // DTA = 1
                        else                        PORTB &= ~(1<< PB0);  // DTA = 0
                PORTB |=  1<< PB2;                  // STR = 1
                PORTB &= ~(1<< PB2);  // STR = 0
                n = n<<1;
        }
PORTB |=  1<<PB1;                   // LCK = 1
PORTB &= ~(1<<PB1);   // LCK = 0
}
```

Функция для вывода числа на 7-сегментный индикатор представляет собой один оператор *switch()* с *case*'ами для чисел от 0x0 до 0xF и одним оператором присваивания в **PORTD** числа в каждом *case*'е. Число, отображающее нужную цифру определяется для каждой цифры исходя из способа подключения дисплея и стандартного вида цифр на 7-сегментных индикаторах.

Далее следует описание логической структуры алгоритма. В блок-схеме на рисунке 2 отсутствует конечный терминатор, поскольку алгоритм содержит в бесконечный главный цикл.

Ниже представлено описание функции *main()*:

```
#define NUM ~PINC&0b00001111  //Значение на цифровом входе.
uint8_t a = 0;     // В библиотеках для микроконтроллеров AVR существуют
uint8_t A[100];    // собственные сокращенные обозначения типов данных.
uint8_t B[100];    // uint8_t – это беззнаковое 8и-битное целое число.
uint8_t n = 0;     // Аналог такого типа на обычном Си – unsigned char.
uint16_t E, F;     // Аналог uint16_t – unsigned int.
int main(void){
  ports_init();    //
  ADC_init();      // Вызываем описанные ранее функции инициализации
  pwm_init();      //
  while (1){       //Главный цикл
    if(PINB & (1<<PB7)){     //Если I/O в положении “Ввод”
      Store_chk();     //Проверка нажатия кнопки подтверждения
      if(PINC&(1<<PC5)){     //Если A/D а положении “Аналоговый”
        ADCSRA |= 1<<ADSC;     //Начать аналогово-цифровое преобразование
      }else{         //Если A/D а положении “Цифровой”
        dn(NUM);     //Отобразить введенное число на 7-сег. инд.
      }
    }else{     //Если I/O в положении “Вывод”
      if (!(PINB&(1<<PB6)))          //Если нажата кнопка “CONFIRM”
        SHR_Write(F);    //Отобразить на цифр. выводе значение F
      else          //Если кнопка “CONFIRM” не нажата
        SHR_Write(E);    //Отобразить на цифровом выводе значение E
      OCR2 = F*0x10;     //Установить ШИМ на уровень F
    }
  }
}
```

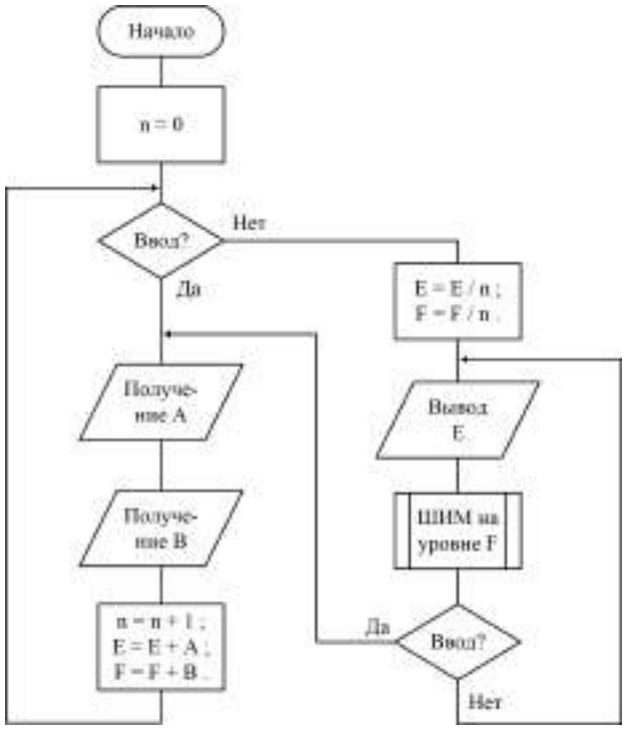

Рис. 2. Блок-схема алгоритма работы устройства

В данном фрагменте кода отсутствует описание функции `Store_chk()`, которая, судя по комментарию должна проверять нажатие на кнопку подтверждения и формировать величины E и F. Функция выглядит следующим образом:

```
void Store_chk(){
      if (CONFIRM){
            n++;        //Прибавить единицу к количеству введенных чисел
```

```
            A[n] = a;   //Сохранить аналоговое значение
            B[n] = NUM; //Сохранить цифровое значение
            E = 0;      //
            F = 0;      //Обнулить
            for(uint8_t i=1; i<=n; i++){  //Просуммировать
                  E += A[i];
                  F += B[i];
            }
            E /= n;     //Поделить на количество
            F /= n;     //
      }
}
```

Теперь не описанным осталось только получение значения аналоговой переменной `uint8_t a`, которая, хранит значение, полученное от АЦП. Однако, в главном цикле присутствует строка:

```
ADCSRA |= 1<<ADSC;     //Начать аналогово-цифровое преобразование
```

Преобразование происходит какое-то время и необходимо получить его результаты как только преобразование закончится. Для этих целей существуют прерывания. При инициализации АЦП было включено прерывание по окончанию преобразования и разрешены общие прерывания, что позволяет быть уверенным в том что прерывание возникает и его можно обработать. Ниже представлен фрагмент кода обработчика прерывания.

```
ISR(ADC_vect){
  a = ADCH;   //Сохранить результат преобразования
  dn(a/0x10); //Отобразить примерный результат преобразования на 7-сег. инд.
}
```

Алгоритм управления был написан на одном из самых распространенных языков, следовательно модификация этого устройства под любые нужды не составит никакого труда. Устройство можно настроить на работу с полноценным знаковым дисплеем, а так же, благодаря аппаратному UART, легко организовать связь с ПК через RS-232 или USB. С помощью микроконтроллера можно управлять шаговыми двигателями и сервоприводами. Столь широкие возможности по обработке данных и управлению физическими объектами нашли применение во многих отраслях науки и промышленности.

Список литературы

1. Прокопенко В.С. Программирование микроконтроллеров ATMEL на языке С. СПб.: КОРОНА-ВЕК, 2012. 307 с.

2. Дхананья Гадре, Нигуал Мэлхотра. Занимательные проекты на базе микроконтроллеров tinyAVR. СПб.: БХВ-Петербург, 2012. 330 с.

3. Википедия – свободная энциклопедия [Электронный ресурс] // Wikimedia Foundation, Inc.: [сайт]. [2001]. URL: http://ru.wikipedia.org/ (дата обращения: 30.08.2013).

4. Краткий Курс – Самоучитель AVR, ATmega и ATtiny [Электронный ресурс]: [сайт]. [2007]. URL: http://123avr.com/ (дата обращения: 30.08.2013).

Баранова Елизавета Михайловна, канд. техн. наук, доц., stroymaster@tula.net, Россия, Тула, Тульский государственный университет,

Глаголев Владислав Максимович, студент, glagol15@gmail.com, Россия, Тула, Тульский государственный университет,

Щепакин Константин Михайлович, д-р техн. наук, проф., Россия, Тула, Тульский институт экономики и информатики

PROGRAMMING OF MICROCONTROLLER USING C LANGUAGE

E.M. Baranova, V.M. Glagolev, K.M. Schepakin

Here examined the process of developing software part to the specific computing device, based on the AVR ATmega8 microcontroller unit. Special attention is paid to peculiarity of programming for microcontrollers using C. The aim is to develop the control algorithm, that allows to receive, simply process and transfer analog and digital signals.

Key words: microcontroller unit, C programming, control algorithm, bitwise operations, logical operations, analog signal, digital signal.

Glagolev Vladislav Maximovich, student, glagol15@gmail.com, Russia, Tula, Tula State University,

Baranova Elizaveta Mihailovna, candidate of technical science, docent, stroymaster@tula.net, Russia, Tula, Tula State University,

Schepakin Konstantin Michailovich, doctor of technical science, professor, Russia, Tula Institute of Economics and Informatics

УДК 004.652.4

АЛГОРИТМ ПОИСКА ОШИБОК И ИЗБЫТОЧНОСТИ В СТРУКТУРЕ БАЗ ДАННЫХ

А.И. Баранчиков, И.В. Дрожжин, А.Ю. Громов

Рассмотрена реализация поиска ошибок, связанных с неадекватным представлением схемой РБД (реляционной базы данных) предметной области, формально описанной с помощью аппарата семантических зависимостей (функциональных, многозначных и соединения) и избыточности в структуре баз данных с использованием табло. Целевой аудиторией разработанного алгоритма являются компании, работающие с базами данных, консалтинговые фирмы, обслуживающие базы данных, ВУЗы для изучения основ теории РБД. Ввиду отсутствия подобных программных средств на рынке, использующих алгоритмы проверки с помощью табло в качестве основного инструмента проверки структуры баз данных, алгоритм является принципиально новым.

Ключевые слова: база данных, зависимости соединения, функциональные зависимости, табло, метод прогонки.

В современных реляционных системах управления базами данных (СУБД) необходимостью становится проверка баз данных на правильность