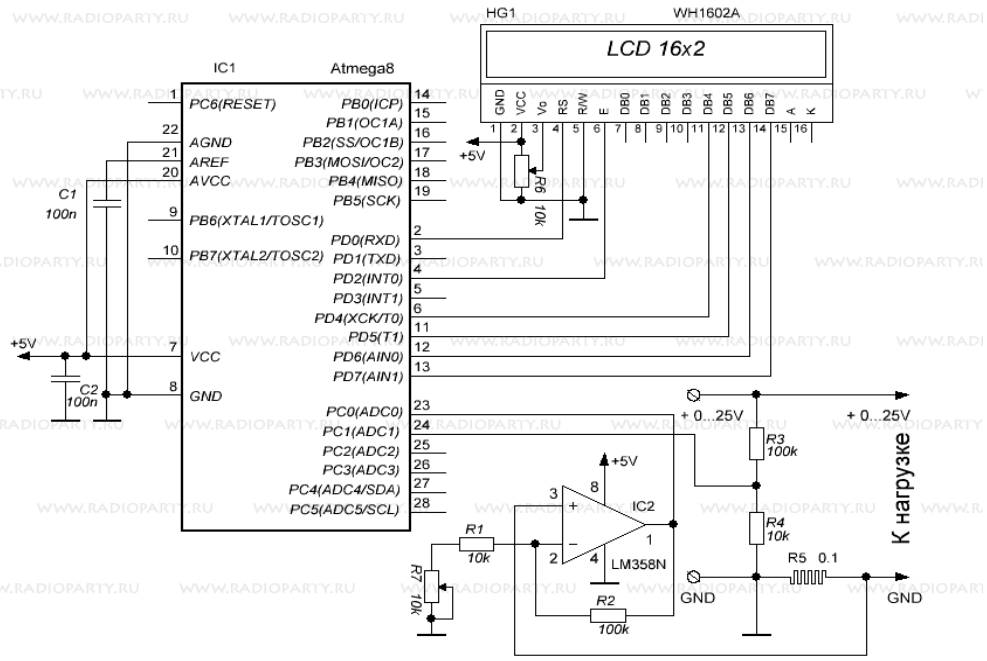
Измерение постоянного тока с помощью AVR. Простой вольтамперметр

Цель: Изучить метод реализации простого вольтамперметра на базе микроконтроллера ATmega8 со следующими характеристиками:

1. Величина измеряемого напряжения 0...25 V;
2. Величина измеряемого тока 0...2,5 A;
3. Вывод показаний на ЖК дисплей 1602;
4. Использование операционного усилителя.

Для измерения напряжения и тока потребуется 2 канала АЦП, используем каналы **ADC0** и **ADC1**, к которым соответственно будут подходить сигналы измеряемых тока и напряжения. Источник опорного напряжения внутренний на 2,56V, разрядность аналого-цифрового преобразователя 10 бит. Подопытный микроконтроллер **Atmega8**, тактируется от внутреннего генератора частотой 4MHz. Схема устройства представлена ниже:



Измерение напряжения

Измеряемое напряжение подается на делитель напряжения, и уже с делителя сигнал подается на вход ADC1. Номиналы сопротивления резисторов делителя 100 кОм и 10 кОм, значит соотношение входного и выходного сигналов 10:1.

Рассчитаем максимальное входное напряжение делителя, чтобы случайно не подать на вход большее напряжение и не повредить микроконтроллер, применим такую формулу:

Umax = Uin*(R1+R2)/R2

где: R1 = 100k, R2 = 10k, Uin = 5V (макс. напряжение порта контроллера),

Umax = 5*110k/10k = 55V

Из этого мы знаем, что больше 55V на вход делителя напряжения подавать нельзя.

Результат преобразования в Вольтах вычисляется по формуле:

U = ADC*Uref*K/1024

где:

ADC - результат преобразования;

Uref опорное напряжение(V);

K - коэффициент делителя напряжения;

1024 - Разрядность АЦП 10 бит.

Коэффициент делителя напряжения вычисляется по формуле:

K = (R1+R2)/R2

он равен: K = (100k + 10k)/10k = 11

Измерение тока

Измерение тока будем производить с помощью токового шунта, который включается в разрыв нагрузки. Падение напряжения на нем вычисляется при помощи закона Ома, эту величину будем измерять другим каналом АЦП(ADC0). Чем меньше сопротивление шунта, тем лучше, т.к.

меньше энергии рассеивается на нем. Возьмем шунт сопротивлением 0,1 Ом, используем обычный мощный резистор. Рассчитаем падение напряжения на нем при силе тока 1 А по формуле:

$$U = I * R$$

U = 1A * 0,1 Ом = 0,1 V

Для тока 2А падение напряжение на шунте будет 0,2V. Величина достаточно малая чтобы напрямую подавать ее на вход АЦП, но есть способ усилить ее с помощью операционного усилителя. Для нашего примера подойдет схема не инвертирующего усилителя, которая имеет бесконечно большое входное, и бесконечно малое выходное сопротивление, что является её несомненным достоинством. Коэффициент усиления ОУ рассчитывается по формуле:

$$Ky = 1 + (R2 / R1)$$

Этот коэффициент сделаем равным примерно 10, так чтобы измеряемый ток величиной 2 А соответствовал напряжению на выходе усилителя в 2 В. Так как ИОН на 2,56 V, больше этого значения на вход АЦП мы подать не можем, расчитаем разрядность измерителя тока:

2,56A / 1024 = 2,5 mA, что вполне достаточно.

Результат преобразования в Амперах вычисляется по формуле:

$$I = ADC*Uref*K/1024$$

где:

ADC - результат преобразования;

Uref опорное напряжение(V);

K - коэффициент усиления операционного усилителя;

1024 - Разрядность АЦП 10 бит.

Программа

Измерение напряжения и тока будем производить по прерыванию окончания преобразования АЦП. Если был выбран канал ADC1(напряжение) то снимаем показания с АЦП, суммируем с прошлыми показаниями и помещаем в буфер, затем выбираем канал ADC0(ток) и проделываем те же самые действия для измерения тока. Этот цикл повторяется 250 раз, затем вычисляем средние значения измеренных величин напряжения и тока, умножаем на 100, чтобы представить результат в милливольтах(миллиамперах) и выводим на экран. Исходный код нашей программы с подробными комментариями представлен ниже:

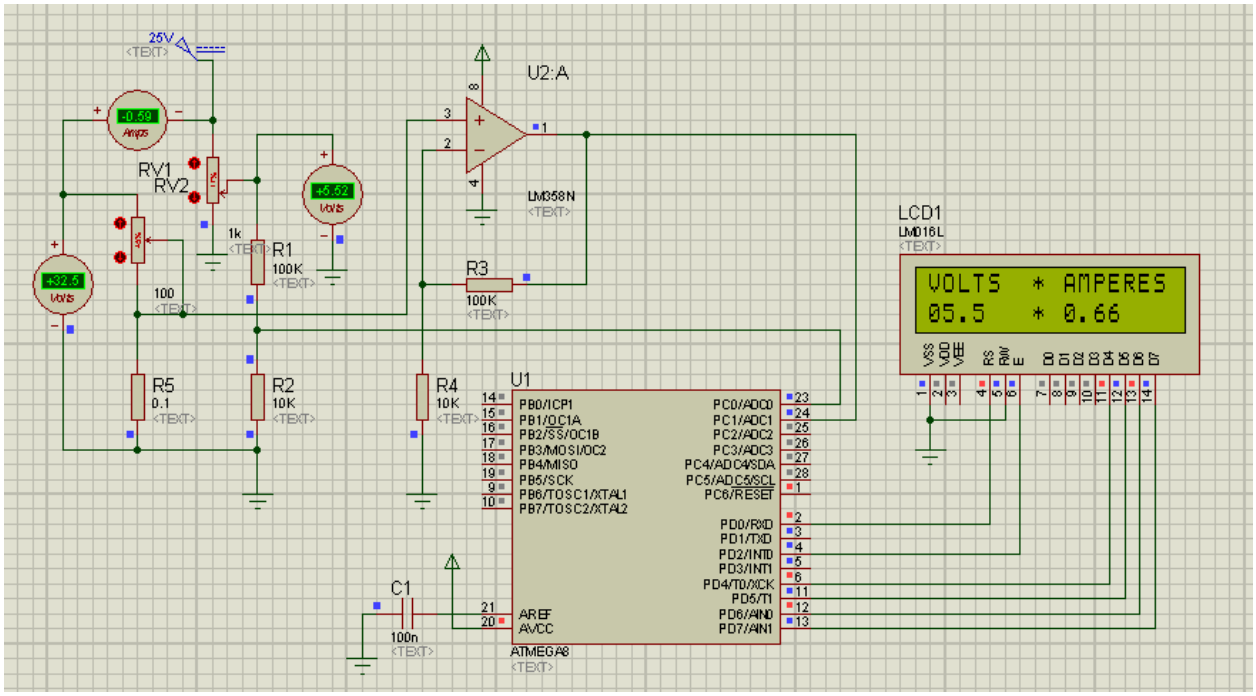
```
// Измерение постоянного тока с помощью AVR
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

unsigned char voltage_counter, current_counter;
unsigned int voltage, current;
volatile unsigned long voltage_buffer, current_buffer;

// Обработчик прерывания от АЦП
ISR(ADC_vect)
{
if(ADMUX & 0x0F) // Если был выбран канал ADC1
{
voltage_buffer += ADC; // Накапливаем в буфер значения напряжения
ADMUX = (ADMUX & 0xF0) | 0; // Выбираем канал ADC0
voltage_counter++; // Увеличиваем счетчик измерений
}
else
{
current_buffer += ADC; // Накапливаем в буфер значения тока
ADMUX = (ADMUX & 0xF0) | 1; // Выбираем канал ADC1
current_counter++; // Увеличиваем счетчик измерений
}
}

// Функции работы с LCD
#define RS PD0
#define EN PD2

// Функция передачи команды
```

```
void lcd_com(unsigned char p)
{
PORTD &= ~(1 << RS); // RS = 0 (запись команд)
PORTD |= (1 << EN); // EN = 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p & 0xF0); // Старший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
PORTD |= (1 << EN); // EN = 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p << 4); // Младший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
}

// Функция передачи данных
void lcd_data(unsigned char p)
{
PORTD |= (1 << RS)|(1 << EN); // RS = 1 (запись данных), EN - 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p & 0xF0); // Старший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
PORTD |= (1 << EN); // EN = 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p << 4); // Младший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
}

// Функция вывода строки на LCD
void lcd_string(unsigned char command, char *string)
{
lcd_com(0x0C); // Включение дисплея, курсор не видим
lcd_com(command); // Адрес знакоместа
// Выводим символы пока не будет конца строки
while(*string != '\0')
{
lcd_data(*string); // Выводим символ
string++; // Следующий символ строки
}
}

// Функция инициализации LCD
```

```
void lcd_init(void)
{
DDRD = 0xFF;
PORTD = 0x00;
_delay_ms(50); // Ожидание готовности ЖК-модуля
// Конфигурирование четырехразрядного режима
PORTD |= (1 << PD5);
PORTD &= ~(1 << PD4);
// Активизация четырехразрядного режима
PORTD |= (1 << EN);
PORTD &= ~(1 << EN);
_delay_ms(5);
lcd_com(0x28); // Шина 4 бит, LCD - 2 строки
lcd_com(0x08); // Полное выключение дисплея
lcd_com(0x01); // Очистка дисплея
_delay_us(100);
lcd_com(0x06); // Сдвиг курсора вправо
lcd_com(0x0C); // Включение дисплея, курсор не видим
}

int main(void)
{
// Настройка АЦП
ADMUX |= (1 << REFS1)|(1 << REFS0); // Внутренний ИОН 2,56V
ADCSRA |=     (1 << ADEN)  // Разрешение АЦП
          |(1 << ADSC)  // Запуск преобразования
          |(1 << ADFR)  // Непрерывный режим работы АЦП
|(1 << ADPS2)|(1 << ADPS1) // Предделитель на 64 (частота АЦП 125kHz)
          |(1 << ADIE); // Разрешение прерывания от АЦП

sei(); // Глобально разрешаем прерывания

lcd_init(); // Инициализация LCD

_delay_ms(25);
lcd_string(0x80 ,"VOLTS  * AMPERES");
lcd_string(0xC0 ,"  .    *  .     ");

while(1)
{
// Вычисляем среднее значение напряжения
if(voltage_counter == 250)
{
voltage = ((voltage_buffer*256*11)/1024)/voltage_counter;
```

```
voltage_counter = 0; // Обнуляем счетчик измерений
voltage_buffer = 0; // Обнуляем буфер значений напряжения
lcd_com(0xC0);
lcd_data((voltage/1000%10)+'0');
lcd_data((voltage/100%10)+'0');
lcd_com(0xC3);
lcd_data((voltage/10%10)+'0');
}

// Вычисляем среднее значение тока
if(current_counter == 250)
{
current = ((current_buffer*256*10)/1024)/current_counter;
current_counter = 0; // Обнуляем счетчик измерений
current_buffer = 0; // Обнуляем буфер значений тока
lcd_com(0xC9);
lcd_data((current/1000%10)+'0');
lcd_com(0xCB);
lcd_data((current/100%10)+'0');
lcd_data((current/10%10)+'0');
}

}
}
```

Реализация в среде Proteus:



Задания для выполнения:

1. Изучить метод реализации простого вольтамперметра на примере микроконтроллера ATmega8.
2. Реализовать приложенную программу в среде Proteus.

Результаты работы отправить на e-mail: rasov@rambler.ru с темой **Вольтамперметр_ФИО**