Подключение LCD HD44780 к AVR через регистр сдвига 74HC595. Делаем двухканальный вольтметр на Attiny13

Цель: Изучить работу контроллера **HD44780.**

Как известно ЖК дисплей на базе контроллера **HD44780** требует для управления минимум 6 линий ввода/вывода микроконтроллера, поэтому подключить его к микроконтроллеру с малым числом портов, например, **Attiny13**, в стандартном 8/4-битном режиме невозможно.

Мы рассмотрим технику управления ЖК дисплеем с использованием всего лишь трех линий ввода/вывода микроконтроллера. Команды управления и данные будут пересылаться последовательно в сдвиговый регистр **74HC595** (8-разрядный сдвиговый регистр с защелкой на выходе), а параллельные выходные данные с регистра поступают на LCD.

Символьные ЖК дисплеи на базе контроллера **HD44780** требуют 14 выводов для управления: 8 линий данных (D0…D7), 3 линии управления (RS, E, R/W), 3 линии питания (Vdd, Vss, Vo). Кроме того, многие модели оснащены подсветкой.

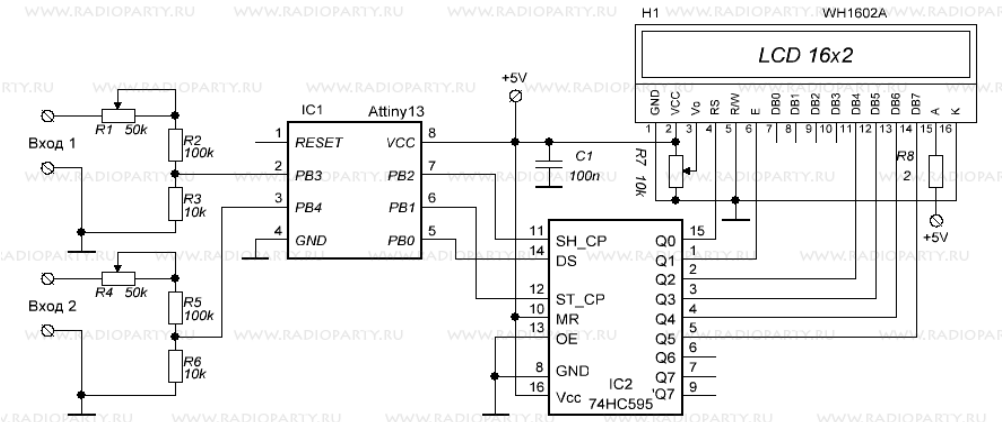
К параллельным выходным линиям регистра сдвига подключен ЖК индикатор: выводы данных D4-D7 и выводы E и RS (4-битный режим работы). Такое решение потребует от микроконтроллера лишь трех линий ввода/вывода:

- **SH_CP** для передачи тактового сигнала
- **DS** для передачи данных
- **ST_CP** для защелкивания данных регистра.

Так как используется 4-битный режим работы, любые восемь бит (команда или данные) передаются в два этапа: сначала старший полубайт, затем передается младший полубайт. Стоит отметить также, что вывод управления индикатора R/W (чтение/запись) подключается к общему проводнику, вследствие чего чтение данных или состояния ЖК модуля при таком подключении невозможно.

Практическим примером такого решения является двухканальный вольтметр (0 - 65V) на микроконтроллере Attiny13 с выводом данных на ЖК дисплей. При подключении дисплея через регистр у контроллера остается как раз две свободных линии, воспользуемся этим и подключим к ним два канала АЦП. Микроконтроллер работает от внутреннего тактового генератора частотой 9,6MHz. Этот вольтметр не является образцовым, но обладает

достаточно хорошей точностью измерения напряжения. Схема вольтметра показана на рисунке ниже:



В управляющей программе используются две функции для общения микроконтроллера и дисплея:

- функция **registr(unsigned char data, unsigned char WriteOrErase)** для передачи данных регистру 74HC595, где **data** - данные, в зависимости от состояния переменной **WriteOrErase**(0 или 1) можно устанавливать отдельный бит передаваемых данных(не трогая другие) в лог. 0 или лог. 1 .

- функция **write_to_lcd(char p, unsigned char rs)** для передачи данных или команд в ЖК дисплей, где **p** - данные, в зависимости от состояния переменной **rs**(0 или 1) передаем команду или передаем данные.

Чтение АЦП производится с помощью функции **readADC(unsigned char ch)**, переменная **ch** определяет номер используемого канала АЦП. В бесконечном цикле уже преобразованное значение АЦП раскладываем на целое значение и значение после запятой.

Полный текст программы:

```
// Подключение LCD HD44780 к AVR через регистр сдвига
// Делаем двухканальный вольтметр на Attiny13
#include <avr/io.h>
#include <util/delay.h>

// Команды для управления портами LCD
/* RS */
#define RS1 registr(0x01, 1)
#define RS0 registr(0x01, 0)

/* E */
#define E1 registr(0x02, 1)
#define E0 registr(0x02, 0)

/* D4 */
#define D41 registr(0x04, 1)
#define D40 registr(0x04, 0)

/* D5 */
#define D51 registr(0x08, 1)
#define D50 registr(0x08, 0)

/* D6 */
#define D61 registr(0x10, 1)
#define D60 registr(0x10, 0)

/* D7 */
#define D71 registr(0x20, 1)
#define D70 registr(0x20, 0)

// Функция установки курсора в указанную точку
#define lcd_gotoxy(x, y) write_to_lcd(0x80|((x)+((y)*0x40)), 0)

volatile static unsigned char tempdata = 0;
unsigned int volt_1, volt_2;

// Функция передачи данных в регистр
void registr(unsigned char data, unsigned char WriteOrErase)
{
if(WriteOrErase == 1)
tempdata = (tempdata|data);
else
```

```
tempdata &= ~(data);
PORTB &= ~(1 << PB1); // ST_CP 0

PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x80)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x40)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x20)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x10)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x08)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x04)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x02)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB &= ~(1 << PB2); // SH_CP 0
if(tempdata & 0x01)PORTB |= (1 << PB0);
else PORTB &= ~(1 << PB0);
PORTB |= (1 << PB2); // SH_CP 1
PORTB |= (1 << PB1); // ST_CP 1
}

// Функция передачи данных или команды в LCD
void write_to_lcd(char p, unsigned char rs)
{
if(rs == 1) RS1;
else RS0;

E1;
```

```
if(p&0x10) D41; else D40;
if(p&0x20) D51; else D50;
if(p&0x40) D61; else D60;
if(p&0x80) D71; else D70;
E0;

_delay_ms(2);

E1;
if(p&0x01) D41; else D40;
if(p&0x02) D51; else D50;
if(p&0x04) D61; else D60;
if(p&0x08) D71; else D70;
E0;

_delay_ms(2);
}

// Функция инициализации LCD
void lcd_init(void)
{
write_to_lcd(0x02, 0); // Курсор в верхней левой позиции
write_to_lcd(0x28, 0); // Шина 4 бит, LCD - 2 строки
write_to_lcd(0x0C, 0); // Разрешаем вывод изображения, курсор не виден
write_to_lcd(0x01, 0); // Очищаем дисплей
}

// Функция вывода строки
void lcd_puts(char *str)
{
unsigned char i = 0;

while(str[i])
write_to_lcd(str[i++], 1);
}

// Функция чтения АЦП
unsigned int read_adc(unsigned char ch)
{
ADMUX = ch; // Выбираем канал АЦП

ADCSRA |= (1 << ADSC); // Запускаем преобразование
while((ADCSRA & (1 << ADSC))); // Ждем окончания преобразования

return ADC; // Возвращаем значение АЦП
```

```
}

int main(void)
{
// Настройка портов ввода/вывода
DDRB = 0b00000111;
// Настройка АЦП
ADCSRA |= (1 << ADEN) // Разрешение АЦП
      |(1 << ADPS2)|(1 << ADPS1); // Предделитель на 64

ACSR |= (1 << ACD); // Выключаем аналоговый компаратор
DIDR0 |= (1 << ADC3D)|(1 << ADC2D); // Отключаем неиспользуемые
цифровые входы

lcd_init(); // Инициализация дисплея

write_to_lcd(0x01, 0); // Очищаем дисплей

lcd_gotoxy(0, 0); // Выводим строки на LCD
lcd_puts("U1 =   .  V");
lcd_gotoxy(0, 1);
lcd_puts("U2 =   .  V");

while(1)
{
// Рассчитаем максимальное входное напряжение на делителе
// Umax = Uin*(R1+R2)/R2
// Umax = 5*(120k+10k)/10k = 65V
// Рассчитаем коэффициент делителя напряжения
// K = (R1+R2)/R2
// K = (120k + 10k)/10k = 13
// Рассчитаем результат преобразования в мВ
// U = (ADC*Uref*K*100)/1024

volt_1 = ((unsigned long)read_adc(2)*5*13*100)/1024;
lcd_gotoxy(5, 0);
write_to_lcd(volt_1/1000   + '0', 1);
write_to_lcd(volt_1/100%10 + '0', 1);
lcd_gotoxy(8, 0);
write_to_lcd(volt_1/10%10  + '0', 1);
_delay_ms(100);

volt_2 = ((unsigned long)read_adc(3)*5*13*100)/1024;
lcd_gotoxy(5, 1);
write_to_lcd(volt_2/1000   + '0', 1);
```

```
write_to_lcd(volt_2/100%10 + '0', 1);
lcd_gotoxy(8, 1);
write_to_lcd(volt_2/10%10  + '0', 1);
_delay_ms(100);
}

}
```

Задания для выполнения:

1. Изучить работу контроллера **HD44780**
2. Реализовать приложенную программу в среде Proteus.

Результаты работы отправить на e-mail: rasov@rambler.ru с темой **HD44780_ФИО**