

Конвертирование текста для ЖКИ на контроллере HD44780

Кодировка символов индикатора на контроллере HD44780 не соответствует кодировке исходных текстов проекта (у нас, как правило, это CP1251). Поэтому, для корректного отображения текста на индикаторе, его необходимо преобразовать. Существует два основных способа преобразования текста:

- преобразование на уровне исполнения;
- преобразование на уровне компиляции (для чего, собственно, и писалась данная статья).

Далее все примеры будут демонстрироваться для компилятора IAR Embedded Workbench (www.iar.com). Язык программирования – Си.

Способ 1. Преобразование на уровне исполнения

Данный способ позволяет, при задании значений строковых переменных в исходных файлах проекта, использовать национальный шрифт. Текст, требующий вывод на индикатор, также хранится в памяти микроконтроллера в кодировке исходных файлов проекта, что не соответствует кодировке индикатора. В связи с этим, текст, перед каждым выводом на индикатор, необходимо конвертировать. Такое преобразование требует наличия во Flash таблицы перекодировки, а также использования ресурсов микроконтроллера. Для современных микроконтроллеров это уже не является проблемой, но всё же это бесполезная работа для микроконтроллера. Данный способ упрощает жизнь разработчику, но усложняет микроконтроллеру.

Реализовать преобразование можно определив массив на 256 байт, где каждому коду символа в кодировке CP1251 (порядковый номер элемента в массиве) соответствует код символа в кодировке индикатора (значение):

```
const uint8_t __flash TableLCDDecode[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0xD9, 0xDA, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0xA2, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0xB5, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x41, 0xA0, 0x42, 0xA1, 0xE0, 0x45, 0xA3, 0xA4, 0xA5, 0xA6, 0x4B, 0xA7, 0x4D, 0x48, 0x4F, 0xA8,
    0x50, 0x43, 0x54, 0xA9, 0xAA, 0x58, 0xE1, 0xAB, 0xAC, 0xE2, 0xAD, 0xAE, 0x62, 0xAF, 0xB0, 0xB1,
    0x61, 0xB2, 0xB3, 0xB4, 0xE3, 0x65, 0xB6, 0xB7, 0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0x6F, 0xBE,
    0x70, 0x63, 0xBF, 0x79, 0xE4, 0x78, 0xE5, 0xC0, 0xC1, 0xE6, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7
};
```

Выводимый на индикатор текст определяется удобным для разработчика способом:

```
const uint8_t __flash MsgAttention[] = "Внимание";
```

Далее, где то в исходных файлах (у меня, например, в прерывании), каждый раз перед записью символа в индикатор, подменяем его символом из таблицы:

```
data = TableLCDDecode[data];
LCDWriteData(data);
```

Способ 2. Преобразование на уровне компиляции

Единственный способ заставить микроконтроллер выводить текст на индикатор без накладных расходов, это хранить текст во Flash уже в кодировке индикатора. Т.е. необходимо произвести преобразование до формирования исполняемого файла. Это можно сделать либо вручную (например, сторонними программами), либо попытаться доверить эту работу препроцессору языка Си.

Наличие в исходных файлах проекта выводимого на индикатор текста в виде последовательности кодов не совсем наглядно, нужно было придумать другой способ преобразования. В итоге, используя препроцессор языка Си, была создана серия макросов (файл `lcd_conv.h`).

Макрос конвертирует текст из кодировки CP1251 в кодировку индикатора. На выходе макроса получается строка соответствующая массиву байт с завершающим кодом 0x00.

Макрос начинается с префикса LCD_TEXT_XX, где XX это количество символов в строке (отсчёт начинается с единицы). Макросы поддерживают преобразование до 80 символов, что соответствует количеству символов индикатора 2004 (4 строки по 20 символов). Избавиться от количества символов в названии макроса не получается в связи с неработоспособностью функции sizeof() в препроцессоре.

Если нам необходимо вывести на индикатор слово «Внимание», то макрос будет выглядеть следующим образом:

```
LCD_TEXT_08("Внимание")
```

где «08» это количество символов в слове «Внимание».

После выполнения данного макроса на выходе получаем строку:

```
{0x42, 0xBD, 0xB8, 0xBC, 0x61, 0xBD, 0xB8, 0x65, 0x00}
```

Если в макросе ошибочно будет указано меньшее количество символов, то на индикатор будет отображена только часть строки. Если в макросе будет указано большее количество символов, то компилятор выдаст ошибку.

Пример использования:

```
// подключаем заголовочный файл с макросом
#include "lcd_conv.h"

// определяем глобальные переменные с текстом
const uint8_t __flash MsgAttention[] = LCD_TEXT_08("Внимание");
const uint8_t __flash MsgWarningT[] = LCD_TEXT_12("Т > 100 град");
const uint8_t __flash MsgWarningP[] = LCD_TEXT_10("P > 10 бар");

//=====
// Функция вывода текста с заданной позиции на ЖКИ
//=====
void LCDShowText(uint8_t ypos, uint8_t xpos, const uint8_t __flash *s)
{
    ...
}

//=====
// Функция вывода сообщения о достижении уставки по температуре
//=====
void LCDShowMsgWarningT(void)
{
    ...
    LCDShowText(0, 0, MsgAttention);
    LCDShowText(1, 0, MsgWarningT);
    ...
}

//=====
// Функция вывода сообщения о достижении уставки по давлению
//=====
void LCDShowMsgWarningP(void)
{
    ...
    LCDShowText(0, 0, MsgAttention);
    LCDShowText(1, 0, MsgWarningP);
    ...
}
```

Таким образом, применение данного макроса, позволяет в исходных файлах проекта использовать национальный язык без дополнительных затрат со стороны микроконтроллера.

Данный макрос без изменений может использоваться с любым компилятором языка Си.