Приемы программирования в среде Arduino и avr gcc

МпУА Колкер А.Б. Ф-т автоматики
НГТУ a.kolker@corp.nstu.ru

Типы данных

```
boolean a= false;
if(a==true) a=!a;
```

1 байт в памяти

Типы данных

char a='a'; // signed -128 127, 1 byte

byte b= B10010011; (B binary) (== unsigned char)

unsigned char a=254; 0..254

a |=0x01;

a &=0x01;

a ^= 0x01;

Целочисленные

int

AT TINY,MEGA, а значит построенные на них Arduino Uno (и все Arduino Mega) 16 bit!!!!

(привычная длина int 4 байта!! 32 бит)

ArduinoDue 4 байта (Cortex CPU).

word w=1000; // unsigned 2 bytes anywhere

short s=100; // signed 2 байта

long /unsigned long l=186000L; //4 байта, требуется пометка L

С плавающей точкой

float 32 бит с плавающей точкой. 6-7 десятичных цифр точности 3.4028235E+38 and as low as -3.4028235E+38

double для ATMEGA **не дает ничего** и в точности повторяет float

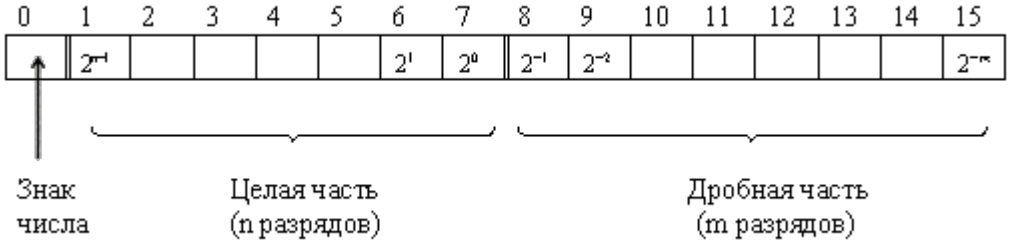
Особенности

- Микроконтроллер не поддерживает на аппаратном уровне арифметику с плавающей точкой.
- Если вы используете плавающую точку, сборщик добавляет код математики, что значительно увеличивает размер (память программ не резиновая), код выполняется медленно.
- Что делать? Не использовать float вовсе!

Решение: фиксированная точка

- Число с фиксированной запятой (обычно байт или слово из двух байт) состоит из целой части (находится в старших битах 8 или 16-разрядного числа) и дробной части (находится в младших битах). Пример числа с фиксированной запятой указан на рисунке, с разрядностью в 8 бит (1 байт).

Числа с фиксированной и плавающей запятой



Фиксированная и плавающая запятая

- Числа с фиксированной запятой:
  - Точность фиксирована
  - Диапазон значений не может меняться и определяется разрядностью целой части
- Числа с плавающей запятой
  - Диапазон определяется разрядностью порядка
  - Точность меняется (от значения порядка)
  - Используется нормализация числа (первая цифра значащая

Плавающая запятая

- Округление до четного (IEEE754) 12.5->12 13.5->14
- Вычитание близких чисел может привести к потере значащих разрядов.
- Многие формулы приходится оптимизировать x^2 -y^2 лучше (x-y)(x+y)
- Не асоциативность: (a*b)*c = a*(b*c) не выполняется.

  И др.

Фиксированная запятая

При всех недостатках такой системы представления чисел, результат абсолютно предсказуем.

Фиксированная запятая

- Перед использованием чисел с фиксированной запятой главное - выбрать разрядность числа (байт или слово), и также выбрать положение запятой.
- Для слова (16 бит) вычисления будут точнее, но скорость упадет и объем кода вырастет (и то и другое не меньше чем в 2 раза), для байта (8 бит) максимальное быстродействие и самый маленький код, но ухудшится точность.
- Обычно вычисляют, сколько разрядов надо выделить на целую часть, и оставшуюся часть достается дробной части.

Пример

- 8-битное число хранит напряжение, которое будет меняться от 0 до 5 вольт. Число от 0 до 5 может кодироваться минимум тремя разрядами, поэтому разряды 7, 6 и 5 будут хранить целую часть (3 разряда), а разряды 4, 3, 2, 1 и 0 - остаются под дробную часть (5 разрядов). Число в дробной части будет показывать, сколько 1/32 (0,03125) от вольта будет в дробной части. Например, если в битах 7, 6 и 5 будет число 4, а в битах 4, 3, 2, 1 и 0 - число 30, то это будет кодировать напряжение 4.9375 вольта (0.9375 = 30/2^5 = 30/32). Значение байта при этом будет 100.11110b или 0x9E.

Правила:сложение,умножение

- В результате сложения двух чисел возможно появление дополнительного разряда. Это происходит, если произошло переполнение. Если возможность переполнения нужно учитывать, то дополнительный 1 бит числа надо где-то хранить.
- Результат умножения двух 8-битных чисел хранить в 16-разрядном числе, двух 16-битных в 32-разрядном, и т. п. (разрядность при умножении складывается).

Деление

- При делении (малого числа на большое особенно) нужно предварительно делимое умножить на константу. Самое простое - сдвинуть число влево на нужное число раз (каждый сдвиг умножает на 2), поместить сдвинутый результат в число вдвое большей разрядности, и потом уже делить. В результате получим число с фиксированной запятой. Например, если делим 8-разрядное целое делимое, сдвинутое влево на 5 разрядов (получили 16-разрядное делимое), на целый делитель, то получаем дробное 16-битное число с фиксированной запятой, где запятая находится между 5 и 4 разрядами.
- Лучше как можно больше пользоваться предварительно вычисленными на этапе компилирования константами, чтобы убрать код, который будет их генерировать

  (Например, если мы должны сравнить напряжение на аккумуляторе с напряжением 1.05 вольт, то это напряжение 1.05 вольт лучше сразу представить в нужном формате и определить директивой #define.)

Отображение

- - сначала берут целую часть, и преобразуют её в символьный вид обычным образом.
- - за целой частью рисуют запятую (или точку).
- - берут дробную часть, приводят её к десятичной дроби, умножая и числитель, и знаменатель дробной части на дробное число (при этом значение дроби, как мы знаем, не изменится) - константу.
- Эта константа выбирается так, чтобы знаменатель стал числом - степенью десятки, а не двойки - при этом получится десятичная дробь. Фраза "домножая на дробное число" означает набор целочисленных операций (сначала умножить на целую константу, а потом разделить на целую константу), результат которых и будет это умножение на дробное число. При операциях умножения и деления либо множитель будет четным, либо делитель, либо они оба - и множитель, и делитель, будут нечетными (мы ведь формируем таким образом умножение на нецелое число). В качестве четной удобно использовать константу, являющуюся степенью двойки (2, 4, 8 и т. д.), потому что умножение и деление на эту константу заменяется простым сдвигом влево и вправо соответственно.
- - после этого полученное значение числителя переводим в набор десятичных цифр и приписываем их после запятой.

Пример: 3 корректно отобразить X.XXX

100.11110b (0x9E) — 4,9375 В

- 100 => 100b => 4.
- 11110b=>30/2^5=>30/32 нам надо делитель 1000
- 1000/32=31.25 , но как умножить на дробное, только целыми числами?
- 31.25=125/4 (делить на 4 можно сдвигом вправо на 2 разряда)
- ИТОГ: 3-31.25=(30*125)р-т в 16 бит/4=3750/4=938=>
- 4.938

Массивы/строки

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino"

char myString[] = "This is the first line"
" this is the second line"
" etcetera";

char* myStrings[]={"This is string 1", "This is string 2", "This is string 3",
"This is string 4", "This is string 5","This is string 6"};
```

-
-

Объект строка (wiring)

```
String stringOne = "Hello String";                 // using a constant String
String stringOne =  String('a');                   // converting a constant char into a String
String stringTwo =  String("This is a string");    // converting a constant string into a String object
String stringOne =  String(stringTwo + " with more");// concatenating two strings
String stringOne =  String(13);                    // using a constant integer
String stringOne =  String(analogRead(0), DEC);    // using an int and a base
String stringOne =  String(45, HEX);               // using an int and a base (hexadecimal)
String stringOne =  String(255, BIN);              // using an int and a base (binary)
String stringOne =  String(millis(), DEC);         // using a long and a base
```

Память и ее использование

- Сколько займет откомпилированный код во Flash всегда видно в процессе компиляции и этот потолок нельзя превысить.
- Сколько памяти использовала программа в RAM не видно явно и, если использовать память бездумно, это довольно опасно.

  В оперативной памяти (RAM), есть три области:
  - статических данных, в ней хранятся глобальные переменные и массивы… и строки!
  - «куча»(heap), используется, если вы вызываете malloc() и free() (растет «вверх»)
  - «стек»(stack), используется, когда одна функция вызывает другую и во время прерываний

Куча (heap)

Куча растёт вверх, и используется довольно непредсказуемым образом. Если вы освобождаете области памяти, то они становятся неиспользованными пробелами в куче, которые могут повторно использоваться новым вызовом malloc(), если запрошенный блок вписывается в эти пробелы.

«Вершина» кучи лежит в переменной _brkval

нематематика: 400+300-400+401 =?=> 1101

stack

Стек расположен в конце оперативной памяти(RAM), и расширяется вниз, по направлению к области кучи. Область стека расширяется и освобождается при необходимости вызова других функций. Там же внутри хранятся локальные переменные.

```
void sub(int a)
    {
        int b; // в этой подпрограмме стек вырастет на 2
        переменных!
    }
```

Дилемма!

Стек растёт навстречу данным! Если он разрастётся, то может достигнуть области данных и записать свои данные вместо ваших значений переменных. Возможна и обратная ситуация — данные разрослись и переписали данные стека. Оба случая крайне неприятны и сложно детектируемы.

Сколько памяти используется?

```
void setup () {
    Serial.begin(57600);
    Serial.println("\n[memCheck]");
    Serial.println(freeRam());
}

void loop () {}

int freeRam () {
 extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

Ужасы нашего городка

Скомпилируем его для ArduiniUnoAtmega328 (2048 RAM) и ужаснемся

- 1846 байт...

  (Кстати, 128 байт съел порт для буфера)

  Добавим Serial.println("\n[memCheck2]");

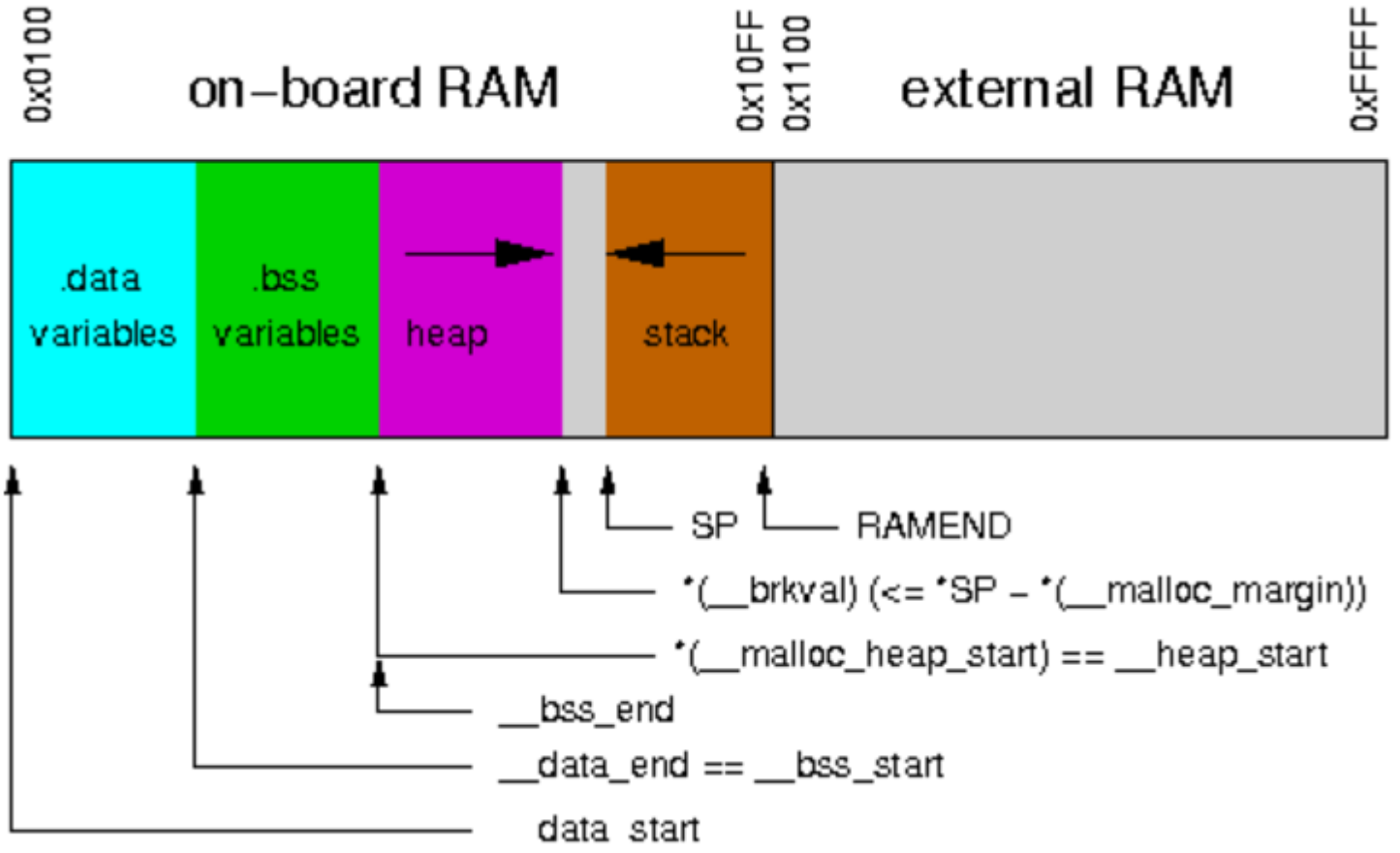
  [memCheck2]

  1844

  Почему?

  - А) все строки хранятся в RAM и копируются туда при старте (не «айс»
  - Б) Флеш организован в слова и не всегда 1 байт== 1 байту в результате

http://www.nongnu.org/avr-libc/user-manual/malloc.html (ATMega128)



Что делать. Девиз экономить!

- Хранить строки и константы во флеше

  Serial.println(F("START")); (wiring)
- Передавать параметры в функции указателями

  void fun(char *var1,int *var2) {}
- Аккуратно действовать с malloc/free
- Размещать неизменяемые данные во flash памяти(PROGMEM)

PROGMEM

- Размещает во flash памяти программ, а ее МНОГО(256к MEGA2560). Директива говорит компилятору: «положи константы в память программ,вместо SRAM»
- Поддерживаются только определенные и только целочисленные типы

prog_char - a signed char (1 byte) -127 to 128
prog_uchar - an unsigned char (1 byte) 0 to 255
prog_int16_t - a signed int (2 bytes) -32,767 to 32,768
prog_uint16_t - an unsigned int (2 bytes) 0 to 65,535
prog_int32_t - a signed long (4 bytes) -2,147,483,648 to * 2,147,483,647.
prog_uint32_t - an unsigned long (4 bytes) 0 to 4,294,967,295

- Требует специальных методов (функций) для доступа к данным (pgmspace.h)

Пример работы с такими данными

```
#include <avr/pgmspace.h>


// save some unsigned ints
PROGMEM  prog_uint16_t charSet[]  = { 65000, 32796, 16843, 10, 11234};

// save some chars
prog_uchar signMessage[] PROGMEM  = {"THE TEXT SHOULD BE PLACED TO FLASH"};

unsigned int displayInt;
int k;    // counter variable
char myChar;

// read back a 2-byte int
 displayInt = pgm_read_word_near(charSet + k)

// read back a char
myChar =  pgm_read_byte_near(signMessage + k);
```

```
#include <avr/pgmspace.h>
prog_char string_0[] PROGMEM = "String 0";   // "String 0" etc are strings to store - change to suit.
prog_char string_1[] PROGMEM = "String 1";
prog_char string_2[] PROGMEM = "String 2";
prog_char string_3[] PROGMEM = "String 3";
prog_char string_4[] PROGMEM = "String 4";
prog_char string_5[] PROGMEM = "String 5";

// Then set up a table to refer to your strings.
PROGMEM const char *string_table[] =        // change "string_table" name to suit
{
  string_0,
  string_1,
  string_2,
  string_3,
  string_4,
  string_5 };
char buffer[30];    // make sure this is large enough for the largest string it must hold

void loop()
{
  /* Using the string table in program memory requires the use of special functions to retrieve the data.
     The strcpy_P function copies a string from program space to a string in RAM ("buffer").
     Make sure your receiving string in RAM  is large enough to hold whatever
     you are retrieving from program space. */


  for (int i = 0; i < 6; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i]))); // Necessary casts and dereferencing, just copy.
    Serial.println( buffer );
    delay( 500 );
  }
}
```

МпУА Колкер А.Б. Ф-т автоматики
НГТУ a.kolker@corp.nstu.ru

PROGMEM SUMMARY

- Выгодно использовать для больших объемов
- Для одиночных переменных используйте ключ F«»
- Иногда выгоднее(и быстрее) использовать таблицу значений,чем использовать математику. Например, разместить таблицу синуса в PROGMEM с нужной дискретностью и достать ее за несколько тактов, чем считать тригонометрию.
- Изменять программно данные PROGMEM нельзя.

EEPROM

	ATMega168	ATMega328P	ATmega1280	ATmega2560
FLASH (-1KB bootloader)	16 KBytes	32 KBytes	128 KBytes	256 KBytes
SRAM	1024	2048	8K	8K
EEPROM	512	1024	4K	4K

EEPROM хранит данные при выключении питания, но в отличие от PROGMEM Данные могут быть изменены.
Удобно хранить данные,которые должны быть доступны после перезагрузки.
НО: Ресурс перепрограммирования EEPROM ограничен 100,000 записываний (читать можно столько,сколько влезет)

EEPROM

EEPROM.write(address, value)

adress (int) 0-TOPEEPROM

```
#include <EEPROM.h>
```

….

```
for (int i = 0; i < 255; i++) EEPROM.write(i, i);
value = EEPROM.read(a);
```

EEPROM- трюки

Задача: делаем автомобильный одометр на ATMEGA168 (512 EEPROM)

{ код который читает датчик оборотов колеса и делает из него километры в час тривиален и опущен}

В чем хранить пробег:

- Int 2^16 65535 км = моторесурс ЗИС5
- Long 2^32 4`294`967`296 км — для полетов на Марс-ОК, для авто пока не актуально. Но в RAM других удобных вариантов нет — берем, но реально используем 20 бит (традиционно считаем 999999км, хотя реально некоторые японские цифровые одометры не «мотают» больше 300000км.) => в EEPROM храним 3 байт

МпУА Колкер А.Б. Ф-т автоматики
НГТУ a.kolker@corp.nstu.ru

Как записывать:

Ресурс EEPROM, если записывать каждый километр, позволит проехать только 100000км, что мало. Как быть?

Вариант 1) В лоб. 30 байт,пишем «по кругу», на дисплей выводим максимальное значение. Едем 1 млн км. => не оптимально!

Вариант 2) Кратно 100 км в 2, ячейки, 100км добавку в остальные «по кругу» по специальному правилу

D1 D2 [X1 X2] ~200 000 км (например, все четные кратные 100км в X1, все нечетные X2)

D1 D2 [X1 X2 X3 X4 X5 X6 X7 X8 X9 X10] ~1`000`000 км

второй вариант= 12 байт 1 миллион ресурса.

В самом дохлом варианте 512 байт. Если расходовать «с умом»хватит надолго!

Биты и байты

- byte lowByte(x) - Достает младший байт
- byte highByte(x) — достает старший байт
- {0;1} bitRead(x,n) читает n й бит из числа X
- bitWrite(x, n, b) устанавливает nй бит числа X в состояние b
- bitSet(x,n) bitClrear(x,n) установить/сбросить

print(),println (wiring)

Метод предназначен для вывода в последовательный порт строк и чисел.

- Целые числа напечатаются «как есть»
- Float будет «укушен» до 2х позиций после точки
- Символы и строки будут напечатаны как есть
- println в обязательном порядке завершит строку '\n'

Вывод простой и форматированный (wiring)

Serial.print(78) gives "78"

Serial.print(1.23456) gives "1.23"

Serial.print('N') gives "N"

Serial.print("Hello world.") gives "Hello world."

Serial.print(78, BIN) gives "1001110"

Serial.print(78, OCT) gives "116"

Serial.print(78, DEC) gives "78"

Serial.print(78, HEX) gives "4E"

Serial.println(1.23456, 0) gives "1"

Serial.println(1.23456, 2) gives "1.23"

Serial.println(1.23456, 4) gives "1.2346"

Поток (stream)

Stream это базовый класс для символьных и битовых потоков.

Многие библиотеки наследуют данный класс, в частности:

Serial,

Write

Ethernet Client

Ethernet Server

SD

И др.

Метод available(),write(),read()

Syntax:

stream.available()

Parameters:

stream : an instance of a class that inherits from Stream.

Returns:

int : the number of bytes available to read

Возвращает количество байт в потоке.

While(Serial.available()) {Serial2.write(Serial.read());}

Методы работы с потоками

.flush() - очищаем буфер

.find(target) читаем пока не найдем строку, возврат boolean

.findUntil(target,terminator)

- target : the string to search for (char)
- terminator : the terminal string in the search (char)

.peek() читаем байт, но не сдвигаем указатель чтения

.readBytes(buffer, length) читаем в буфер строку нужной длины, или таймаут.

Методы работы с потоками

.stream.readBytesUntil(character, buffer, length) + символ терминатора

.readString() читаем строку

.readString(terminator)

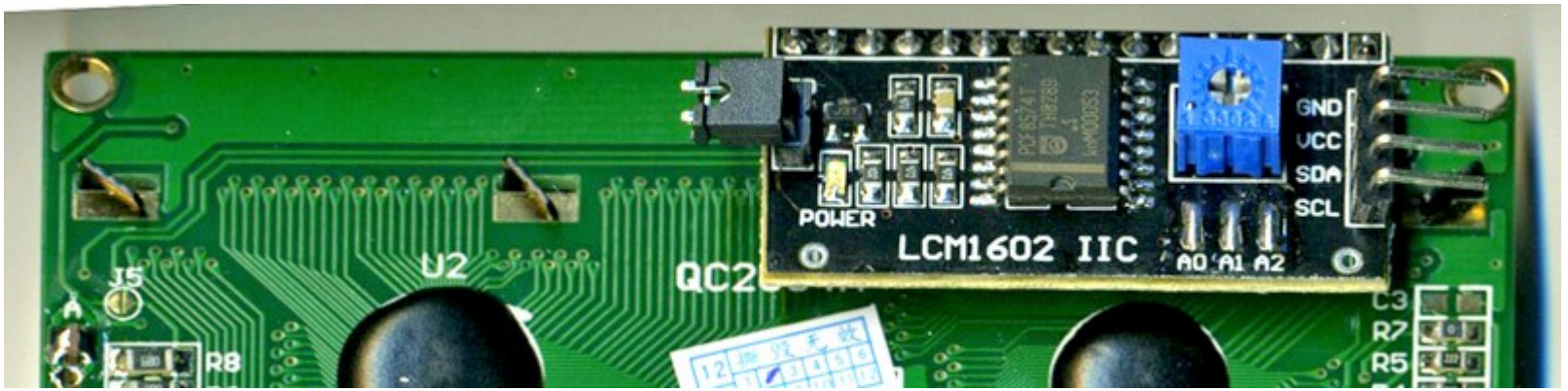
.parseInt() - считывает первый корректный int и присваивает его переменной

.stream.parseFloat()

.setTimeout() = устанавливает макс время ожидания (1000ms def)

Текстовый дисплей

- Дисплей: Символьный 16x02 либо 20x04
- Подсветка: зависит от типа дисплея
- Контраст: Настраивается потенциометром
- Напряжение питания: 5В
- Интерфейс: I2C / прямой /
- I2C адрес: 0x27
- Размеры: 82мм x 35мм x 18мм



https://arduino-info.wikispaces.com/LCD-Blue-I2C

МпУА Колкер А.Б. Ф-т автоматики
НГТУ a.kolker@corp.nstu.ru

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,20,4);  // set the LCD address to 0x20  chars and 4 line display
int i=0;

void setup()
{
  lcd.init();                    // initialize the lcd
//  lcd.init();
  // Print a message to the LCD.
  lcd.backlight();
  lcd.setCursor(3,0);
  lcd.print("FIRST STRUING");
  lcd.setCursor(2,1);
  lcd.print("SECOND STRING");
   lcd.setCursor(0,2);
  lcd.print("THIRD STRING");
   lcd.setCursor(2,3);
}


void loop()
{
  lcd.setCursor(2,3);
  lcd.print(i++);
  delay(1000);
  lcd.clear();
}
```

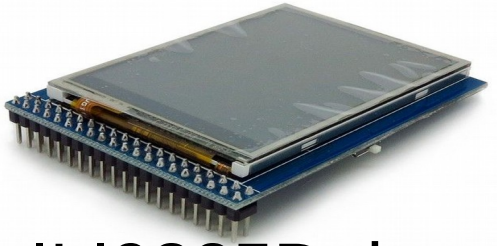
- lcd.LiquidCrystal() - первичная инициализация дисплея без контроллера i2c
- lcd.LiquidCrystal_I2C lcd(0x27,20,4); первичная инициализация дисплея с контроллером I2c
- lcd.begin(cols, rows) - установка типа дисплея
- lcd.clear() - очищает дисплей
- lcd.home() - возвращает курсор в левую верхнюю позицию
- lcd.setCursor(col, row) - курсор в положение
- - lcd.write(data) — пишет символ в позицию курсора
- - print()
- - cursor() показывает курсор
- - NoCursor() скрывает курсор
- - blink() моргать курсором
- - noBlink() не моргать курсором
- - display() разбудить дисплей
- - noDisplay() усыпить дисплей
- - scrollDisplayLeft() - скроллинг дисплея на одну позицию влево
- - scrollDisplayRight() скроллинг дисплея на одну позицию вправо
- - autoscroll() автоскроллинг дисплея.
- - noAutoscroll()
- - leftToRight() справа налево
- - rightToLeft() слева направо

lcd.createChar(num, data)

Создает кастомный символ. Разрешается иметь только 8 символов.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte smiley[8] = {
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
};

void setup() {
  lcd.createChar(0, smiley);
  lcd.begin(16, 2);
  lcd.write(byte(0));
}

void loop() {}

```

TFT 2.4" LCD Touch Screen (ITDB02-2.4E)



- 2,4"
- 320x240
- 65K цветов
- The controller of this LCD module is ILI9325D, it supports 8bit / 16bit data interface with 4 wires control interface. Moreover, this module includes the touch screen and SD card socket.
- Сложное устройство,управлять которым на низком уровне (командами) весьма неудобно.
- Все управление команды по последовательному интерфейсу.

Как с ним работать на низком уровне? (

```
void Write_Command(unsigned int c)
{
    digitalWrite(RS,LOW);//LCD_RS=0;
digitalWrite(CS,LOW);//LCD_CS =0;
PORTD = c>>8; //LCD_DataPortH=DH>>8;
digitalWrite(WR,LOW);//LCD_WR=0;
digitalWrite(WR,HIGH);//LCD_WR=1;
PORTD = c;//LCD_DataPortH=DH;
digitalWrite(WR,LOW);//LCD_WR=0;
digitalWrite(WR,HIGH);//LCD_WR=1;
digitalWrite(CS,HIGH);//LCD_CS =0;
}
```

```
void Write_Data(unsigned int c)
{
   digitalWrite(RS,HIGH);//LCD_RS=0;
digitalWrite(CS,LOW);//LCD_CS =0;
PORTD = c>>8; //LCD_DataPortH=DH>>8;
digitalWrite(WR,LOW);//LCD_WR=0;
digitalWrite(WR,HIGH);//LCD_WR=1;
PORTD = c;//LCD_DataPortH=DH;
digitalWrite(WR,LOW);//LCD_WR=0;
digitalWrite(WR,HIGH);//LCD_WR=1;
digitalWrite(CS,HIGH);//LCD_CS =0;
}
void Write_Command_Data(unsigned int cmd,unsigned int dat)
{
Write_Command(cmd);
Write_Data(dat);
}
```

```
void LCD_clear()
{
   unsigned int i,j;
SetXY(0,239,0,319);
for(i=0;i<X_CONST;i++)
{
   for(j=0;j<Y_CONST;j++)
{
        Write_Data(0x0000);
}
}
}

void SetXY(unsigned int x0,unsigned int x1,unsigned int y0,unsigned int y1)
{
  Write_Command_Data(0x0046,(x1 << 8)| x0);
  //Write_Command_Data(0x0047,x1);
  Write_Command_Data(0x0047,y1);
  Write_Command_Data(0x0048,y0);
  Write_Command_Data(0x0020,x0);
  Write_Command_Data(0x0021,y0);
  Write_Command (0x0022);//LCD_WriteCMD(GRAMWR);
}
```

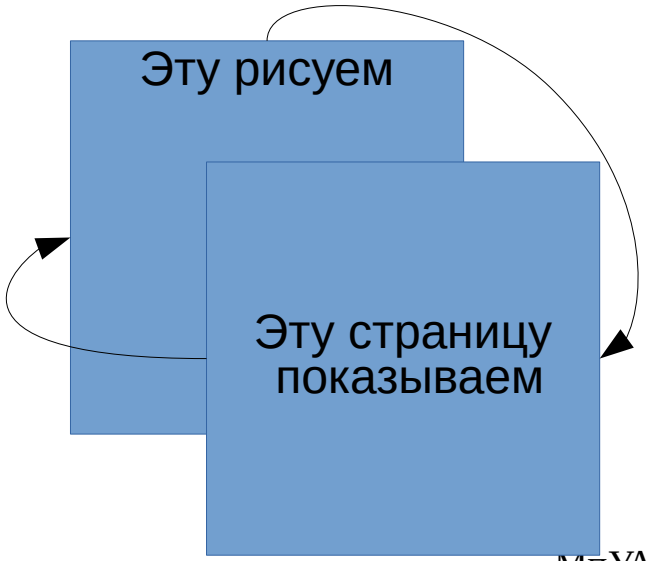
Но есть библиотека UTFT!

```
#define TOUCH_ORIENTATION  PORTRAIT
#define TOUCH_IRQ 9

#include <UTFT.h>
#include <UTouch.h>

// Declare which fonts we will be using
extern uint8_t SmallFont[];
extern uint8_t BigFont[];
extern uint8_t SevenSegNumFont[];
UTFT myGLCD(ITDB24E_8,A5,A4,A3,A2);
```

```
void setup()
{
  pinMode(LED,OUTPUT);
  myGLCD.InitLCD();
  myGLCD.clrScr();
  attachInterrupt(0, blink_func,RISING);
}
void loop()
{
char buffer[50];

myGLCD.setColor(255,255,255);
myGLCD.setFont(SevenSegNumFont);
sprintf(buffer,"  %d  ",analogRead(15));
myGLCD.print(buffer, CENTER, 20);
```

Трюки с дисплеями

- Дисплей управляется по последовательной линии. Прорисовка и перерисовка данных происходит медленно и печально,что раздражает пользователей.
- Выход: Многостраничность!!!!(там где это поддерживается)



- MyGLCD.setDisplayPage(pg);// pg 0-7 какую страничку сейчас показывать
- MyGLCD.setWritePage(pg);// pg 0-7 а на какой страничке рисовать!

Ложка дегтя в сторону ардуинщиков

Дисплеи в части контроллера сенсорной панели для UNO и MEGA несовместимы, поскольку у UNO и MEGA порты, поддерживающие прерываения разнесены на разные пины разьемов.

МпУА Колкер А.Б. Ф-т автоматики
НГТУ a.kolker@corp.nstu.ru

Что почитать

- Arduino.cc
- http://microsin.ru/content/category/5/26/44/
- https://habrahabr.ru/post/131171/
-