Вадим Елисеев @VadimEliseev
Пользователь

7 октября 2013 в 13:31

Работа с TFT дисплеем на ARDUINO DUE

из песочницы tutorial

DIY или Сделай сам*

Вторая часть здесь: http://habrahabr.ru/post/196864/
Эта статья открывает небольшой цикл статей, посвященных работе с многоцветными TFT дисплеями на Arduino DUE. В этой и следующих статьях будут рассмотрены основные возможности TFT дисплеев, приведено описание библиотек, рассмотрены примеры типичных задач, возникающих при работе с такими дисплеями.

В настоящее время на рынке Arduino-комплектующих присутствует множество разнообразных TFT дисплеев. С точки зрения пользователя они отличаются друг от друга, главным образом, размерами, разрешающей способностью, способами подключения и дополнительным функционалом. Большинство таких дисплеев оборудовано сенсорным экраном, делающим управление системой более удобным и позволяющим избавиться от традиционных кнопок, джойстиков, энкодеров и других механических приспособлений.

Работа с графическим дисплеем с разрешением порядка 320x240 и выше предполагает наличие солидного объема памяти и достаточно высокое быстродействие самого микроконтроллера. Кроме того подключение часто требует большого количества пинов, поэтому в качестве базы был выбран контроллер Arduino DUE.
В качестве «экспериментальной установки» при написании этой статьи было использовано следующее оборудование:

- Контроллер Arduino DUE — Arduino-совместимый клон Iduino от Getech.
- TFT дисплей: после долгих раздумий был выбран «продвинутый» вариант дисплея Coldtears Electronics 3,2'', отличающийся повышенным разрешением (480x320 вместо 320x240 у большинства аналогичных дисплеев) и дополнительным функционалом (у этого дисплея имеется встроенная Flash-память, в которую вшиты дополнительные шрифты и иконки, что позволяет отказаться от загрузки дополнительных шрифтов, сэкономив тем самым драгоценную память контроллера). Дисплей оборудован контроллером ILI948. На плате также установлен разъем для SD-карты.
- Специальный шильд для подключения дисплея к контроллеру Arduino DUE. Этот шильд позволяет подключить дисплей через 16-битный параллельный интерфейс, а Flash-память через ISP разъем Arduino, что, по уверению разработчиков, обеспечивает максимальную производительность. Шильд также оборудован разъемом для SD-карты (по умолчанию он отключен, для включения его необходимо замкнуть перемычку на плате, но об этом – чуть позже).

Обратите внимание, шильд разработан именно для Arduino DUE. Для использования дисплея с Arduino MEGA нужна другая версия шильда, оборудованная преобразователем уровня 5в–3.3в.
Все материалы, касающиеся распиновки дисплея и CTE шильда можно скачать на сайте производителя:
Дисплей: http://coldtears.lin3.siteonlinetest.com/files/3.2b.zip
Шильд: http://coldtears.lin3.siteonlinetest.com/files/CTE_DUE_shield.zip
Для работы с TFT дисплеями используется набор библиотек UTFT. Наиболее свежие версии этих библиотек можно найти на сайте разработчика: http://www.henningkarlsen.com/electronics/
После подключения дисплея остаются свободными следующие пины контроллера: D0, D1, D8, D11-D13, D14-D24, D29-D31, D41-D43, A0-A11, DAC0, DAC1, SCL1, SDA1, которые можно использовать по своему усмотрению.

1. Базовая библиотека UTFT. Общие сведения

Библиотека UTFT предназначена для работы с TFT дисплеями различных типов, подключаемых как по параллельному (8бит или 16бит), так и по последовательному интерфейсу. Первоначально она разрабатывалась для поддержки дисплеев производства ITead Studio и NKG Electronics, однако потом в нее была включена поддержка нескольких других производителей. Оригинал библиотеки можно скачать здесь: http://www.henningkarlsen.com/electronics/library.php?id=51
Библиотека поставляется с довольно неплохой документацией на английском языке. Данная статья основывается, главным образом, на документации разработчика с добавлением некоторого количества практического опыта и дополнительных сведений, которые были получены в процессе работы.

Список поддерживаемых библиотекой дисплеев и их основные особенности приводятся в сопроводительном документе “UTFT Supported display modules & controllers”, который поставляется вместе с библиотекой.
Эта библиотека является базовой и содержит только основной функционал вывода на дисплей. К ней существует несколько дополнений, в которых реализованы дополнительный функции, такие как управление при помощи сенсорного экрана (U_Touch), работа с экранными кнопками (UTFT_Buttons), расширенный набор функций для рисования графических примитивов (UTFT_Geometry), использование встроенных в некоторые дисплеи шрифтов и иконок (UTFT_CTE) и т. д.
Базовая библиотека позволяет работать с подгружаемыми шрифтами. Шрифты хранятся в виде массивов данных, которые размещаются в отдельных файлах и подключаются к тексту программы.

2. Начало работы с библиотекой

Библиотека может быть использована как с контроллерами, основанными на AVR платформе (Arduino MEGA), так и с контроллерами на ARM платформе (Arduino DUE). Прежде чем начать использовать библиотеку с контроллером Arduino DUE необходимо зайти в папку, в которую установлена библиотека: _ARDUINO_FOLDER_/libraries/UTFT, найти там папку /hardware/arm и в ней файл HW_ARM_defines. В этом файле необходимо раскомментировать строку:

```
#define CTE_DUE_SHIELD 1
```

У библиотеки UTFT есть также полезная «фишка», которая позволяет отключить коды инициализации тех моделей дисплеев, которые в данный момент не используются. Это позволяет сэкономить некоторое количество памяти (например, у меня объем скомпилированной программы уменьшился почти на 12Кбайт). Для этого необходимо открыть в редакторе файл memorysaver.h, расположенный в корневой папке библиотеки и раскомментировать строки с названиями тех контроллеров, которые Вы не планируете использовать в своем проекте. При этом закомментированным должен остаться только один контроллер – тот, что Вы используете. Наш дисплей, как уже говорилось выше, оборудован контроллером ILI9481. Поэтому строчку:

```
//#define DISABLE_ILI9481 1 // CTE32HR
```

оставляем закомментированной, а в остальных строках удаляем все символы // в начале строк. Теперь можно работать.

Использование библиотеки в проекте начинается с ее подгрузки и инициализации. Параметры инициализации зависят от типа используемого дисплея. Подгружаем базовую библиотеку UTFT и создаем объект – дисплей с именем myGLCD. Параметры – идентификатор модели дисплея и номера пинов, к которым подключаются линии RS, WR, CS, RST и ALE. Эти параметры следует выяснить у производителя дисплея или подобрать по списку поддерживаемых дисплеев, который поставляется вместе с библиотекой.
В нашем варианте – дисплей 3,2’’ с разрешением 480x320 и контроллером ILI9481 производства Coldtears electronics обозначается как CTE32HR. При этом, согласно схеме шильда, управляющие линии подключаются к пинам 25-28.

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
```

Далее, необходимо подключить внешние шрифты (использовать встроенные шрифты дисплея пока не будем – это тема для отдельного разговора). Библиотека поставляется с тремя базовыми шрифтами. Вот так выглядят строки объявления для этих шрифтов:

```
extern uint8_t SmallFont[];
extern uint8_t BigFont[];
extern uint8_t SevenSegNumFont[];
```

Итак, мы подключили и инициализировали библиотеку, присоединили шрифты, теперь мы должны выполнить инициализацию нашего дисплея в процедуре void setup():

```
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
}
```

Команда InitLCD() позволяет задать вертикальную или горизонтальную ориентацию дисплея. По умолчанию (если команда вызывается без параметра) устанавливается горизонтальная ориентация. Команда clrScr(); просто очищает дисплей. Обратите внимание, в библиотеке нет понятия фонового цвета всего дисплея. После очистки дисплей всегда становится черным.

Ну и наконец, прежде чем мы перейдем к подробному рассмотрению команд библиотеки, закончим наш самый простой пример — выберем шрифт BigFont и напечатаем традиционное «Hello, world!» в центре верхней строки дисплея:

```
void loop() {
myGLCD.setFont(BigFont);
myGLCD.print("Hello, world!",CENTER,0);
}
```

3. Команды библиотеки

При работе с библиотекой необходимо установить ее в папку с библиотеками Arduino и подключить при помощи директивы:
`#include <UTFT.h>`
Описание команд взято, главным образом, из англоязычного мануала, поставляемого вместе с библиотекой, и значительно дополнено практическими наблюдениями и примерами. Итак…

UTFT – создает базовый класс дисплея с заданным именем, в качесте параметров указывается идентификатор модели и способ подключения.
Возможны два варианта: для моделей с параллельным интерфейсом команда имеет вид:
`UTFT _NAME_ (model, RS, WR, CS, RST, ALE)` где _NAME_ — произвольное имя объекта «дисплей», которое будет использоваться в качестве префикса при любом обращении к нему, model – идентификатор модели конкретного дисплея (см. список поддерживаемых моделей), а RS, WR, CS, RST и ALE – номера пинов, к которым посредством шильда подключены соответствующие управляющие сигналы дисплейного модуля. Сигнал ALE есть не у всех поддерживаемых моделей, если в Вашей модели его нет – просто пропустите этот параметр.
Для моделей с последовательным интерфейсом команда имеет вид:
`UTFT _NAME_ (model, SDA, SCL, CS, RST, RS)` где SDA, SCL, CS, RST и RS – номера пинов, к которым подключены соответствующие сигналы последовательного интерфейса. Параметр RS – опциональный, для некоторых моделей дисплеев он не задается.
Строка с этой командой размещается в области определений и должна предшествовать любым другим командам библиотеки. Заданное имя дисплея должно использоваться в качестве префикса ко всем последующим командам библиотеки.

InitLCD – инициализирует дисплей и задает горизонтальную или вертикальную ориентацию. В качестве параметра указывается идентификатор ориентации.
Будучи заданной без параметров команда устанавливает горизонтальную ориентацию. Если указать параметр PORTRAIT или 0 – будет выбрана вертикальная ориентация, если указать LANDSCAPE или 1 – горизонтальная.
При горизонтальной ориентации разъемы контроллера располагаются слева, при вертикальной – внизу. Остальные варианты не предусмотрены. Задание в качестве параметра других чисел, кроме 0 и 1 приводит к искаженному выводу информации на дисплей.
Эта команда обычно используется в разделе void setup() но ее можно использовать и в основном цикле программы, как, например, здесь:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {  }
void loop() {
myGLCD.InitLCD(1);
myGLCD.setFont(BigFont);
myGLCD.print("Hello, world!",CENTER,0);
delay(1000);
myGLCD.InitLCD(0);
myGLCD.setFont(BigFont);
myGLCD.print("Hello, world!",CENTER,0);
delay(1000);
}
```

В принципе, присоединив к дисплею датчик положения, можно было бы организовать переворот экрана как в планшете ;)) правда при инициализации дисплей на короткое время вспыхивает белым цветом, что несколько портит впечатление. ;)
Эта команда устанавливает черный цвет фона, белый цвет текста, сбрасывает имя используемого шрифта в «none», при этом результат работы всех команд печати становится непредсказуемым до тех пор, пока не будет явно задан шрифт (см. команду setFont). После инициализации рекомендуется выполнить очистку дисплея (см. команду clrScr), при этом экран будет залит черным цветом.

clrScr – очищает дисплей, стирая всю отображаемую на дисплее информацию и заливая дисплей черным цветом. Параметров не имеет.

При очистке дисплея заданный цвет фона (см. команду setBackColor) не сбрасывается и остается прежним, но дисплей все равно заливается черным цветом. Для того, чтобы очистить дисплей с другим цветом фона необходимо использовать команду fillScr. Работу команды clrScr иллюстрирует следующий пример:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
delay(1000);
myGLCD.setBackColor(0,255,0);
myGLCD.clrScr();
delay(1000);
myGLCD.setFont(BigFont);
myGLCD.print("FFFFFFFF", CENTER,0);
}
void loop() {  }
```

После сброса контроллера происходит инициализация дисплея, затем через 1 секунду очистка (дисплей становится черным) и еще через секунду выводятся символы, при этом мы видим, что заданный перед очисткой цвет фона не изменился.
Прежде чем рассматривать следующие несколько команд необходимо рассмотреть особенности задания цветов в командах библиотеки.

Система кодирования цветов
Цвета в библиотеке задаются несколькими способами. Внутренним форматом представления данных о цвете является формат RGB565. В этом формате цвет кодируется 16-битным значением, в котором уровень красного и синего кодируется пятью битами, а зеленого – шестью. Большинство команд библиотеки, работающих с цветом, воспринимает значения, заданные в виде трех чисел, разделенных запятой. Каждое из этих чисел отвечает за уровень соответствующего цвета (R, G, B). Допустимые значения каждого числа – от 0 до 255.

Таким образом, цвета задаются пользователем в формате RGB888, а внутри библиотеки используется RGB565. Преобразование форматов выполняется внутри библиотеки по следующей формуле:
`word color = ((r&248)<<8 | (g&252)<<3 | (b&248)>>3);`
где color – значение цвета в формате RGB565, а r, g, и b – значения цветов, задаваемые пользователем. Именно в таком формате возвращают значения и все функции определения текущего цвета.
При задании цвета необходимо учитывать, что не все сочетания задаваемых значений r, g, и b будут давать разные цвета. Поэтому, например, значения уровня красного 255 и 253 дают один и тот же уровень красного. Приведенная ниже программа выводит на дисплей максимальные и минимальные значения r, g и b, допустимые для получения цвета, заданного в программе в командах установки цвета. Для выполнения расчета подставьте в параметры команды myGLCD.setBackColor десятичные значения r, g и b для выбранного цвета, скомпилируйте и запустите программу. На дисплее отобразится код цвета в системе RGB565, возвращенный функцией myGLCD.getBackColor, а также минимальные и максимальные значения r, g и b, позволяющие получить данный цвет.

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
myGLCD.setBackColor(245,34,112);
myGLCD.setColor(VGA_NAVY);
word CurrentColor=myGLCD.getBackColor();
byte R_1 = highByte(CurrentColor)>>3;
byte R_min = R_1<<3;
byte R_max = R_min|7;
byte G_1 = highByte(CurrentColor)<<5;
G_1 = G_1>>2|lowByte(CurrentColor)>>5;
byte G_min = G_1<<2;
```

```
byte G_max = G_min|3;
byte B_1 = lowByte(CurrentColor)<<3;
B_1 = B_1>>3;
byte B_min = B_1<<3;
byte B_max = B_min|7;
myGLCD.print("GetColor", 0,0);
myGLCD.print("RGB888 Min", 0,36);
myGLCD.print("RGB888 Max", 0,54);
myGLCD.printNumI(CurrentColor, 162,0);
myGLCD.printNumI(R_min, 162,36);
myGLCD.printNumI(R_max, 162,54);
myGLCD.printNumI(G_min, 212,36);
myGLCD.printNumI(G_max, 212,54);
myGLCD.printNumI(B_min, 262,36);
myGLCD.printNumI(B_max, 262,54);
}
```

Некоторые стандартные цвета можно задавать при помощи идентификаторов:
VGA_BLACK – черный,
VGA_SILVER – серебряный
VGA_GRAY – серый
VGA_WHITE – белый
VGA_MAROON – красно-коричневый
VGA_RED – красный
VGA_PURPLE – пурпурный
VGA_FUCHSIA – фуксия
VGA_GREEN – зеленый
VGA_LIME – лайм
VGA_NAVY – темно-синий
VGA_BLUE – синий
VGA_TEAL – сине-зеленый
VGA_AQUA – морская волна
Если вы хотите узнать RGB коды для этих цветов – подставьте идентификатор цвета вместо трех чисел в команду myGLCD.setBackColor в приведенной выше программе, после компиляции и запуска на дисплее отобразятся нужные коды.

Итак, продолжим рассмотрение команд библиотеки:

fillScr – очищает дисплей, стирая всю отображаемую на дисплее информацию и заливая его указанным в качестве параметра цветом фона.
Цвет фона задается тремя числами или идентификатором цвета, например, так:

```
fillScr(0,0,0); // черный цвет
fillScr(255,255,255); // белый цвет
fillScr(255,128,0); // оранжевый цвет
fillScr(VGA_RED); // стандартный красный цвет
```

Эта команда, так же как и clrScr не изменяет заданный цвет фона для команд печати, поэтому при печати на дисплее, залитом заданным цветом необходимо дополнительно указывать цвет фона (см. команду setBackColor).

setColor – устанавливает цвет «чернил» для всех команд печати и рисования графических примитивов. В качестве параметра указывается необходимый цвет.
Цвета задаются так же, как и для команды fillScr – либо числами, либо идентификаторами стандартных цветов. Пример:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
myGLCD.setColor(VGA_RED);
```

```
myGLCD.print("Hello, World!",CENTER,0);
myGLCD.setColor(VGA_NAVY);
myGLCD.print("Hello, World!",CENTER,18);
myGLCD.setColor(VGA_TEAL);
myGLCD.print("Hello, World!",CENTER,36);
myGLCD.setColor(VGA_LIME);
myGLCD.print("Hello, World!",CENTER,54);
while(1);
}
```

setBackColor – устанавливает цвет фона для команд печати. В качестве параметра указывается необходимый цвет.
Цвета задаются обычным способом – числами или идентификаторами (см. команду fillScr) однако для этой команды существует еще один идентификатор – VGA_TRANSPARENT, позволяющий печатать символы на «прозрачном» фоне.
При необходимости печати на дисплее, залитом любым цветом, отличным от черного, необходимо задавать точно такой же цвет фона командой setBackColor, или использовать команду setBackColor(VGA_TRANSPARENT).

getColor – возвращает текущее значение цвета «чернил». Параметров не имеет.

getBackColor –возвращает текущее значение цвета фона. Параметров не имеет.

Эти две функции возвращают значение типа word, соответствующее текущему заданному цвету в формате RGB565. Это значение можно передавать в качестве параметра в командах setColor, SetBackColor и fillScr например так:

```
setColor(32586);
```

эта команда дает тот же самый цвет, что и setColor(120,232,80) или setColor(125,233,84), в чем можно убедиться при помощи рассмотренной выше программы декодирования цветов RGB565.

getDisplayXSize – возвращает значение ширины дисплея в пикселях при выбранной ориентации. Параметров не имеет.

getDisplayYSize – возвращает значение высоты дисплея в пикселях при выбранной ориентации. Параметров не имеет.

Эти две функции можно проиллюстрировать следующим примером:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {  }
void loop() {
myGLCD.InitLCD(PORTRAIT);
myGLCD.setFont(BigFont);
myGLCD.clrScr();
myGLCD.print("Screen Width=",0,0);
myGLCD.printNumI(myGLCD.getDisplayXSize(),250,0);
myGLCD.print("Screen Height=",0,18);
myGLCD.printNumI(myGLCD.getDisplayYSize(),250,18);
delay(2000);
myGLCD.InitLCD(LANDSCAPE);
myGLCD.setFont(BigFont);
myGLCD.clrScr();
myGLCD.print("Screen Width=",0,0);
myGLCD.printNumI(myGLCD.getDisplayXSize(),250,0);
myGLCD.print("Screen Height=",0,18);
myGLCD.printNumI(myGLCD.getDisplayYSize(),250,18);
delay(2000);
}
```

При запуске программы (устанавливается вертикальная ориентация) ширина дисплея будет равна 320 пикселей, а высота 480, через 2 секунды ориентация меняется на горизонтальную и ширина становится равной 480 пикселей, а высота – 320.
Прежде чем рассматривать команды печати и работу со шрифтами следует более подробно остановиться на методах

использования шрифтов в библиотеке UTFT.

Внешние шрифты для команд печати.
Внешние шрифты хранятся в отдельных файлах с расширением.c и представляют собой массивы данных, содержащие информацию о параметрах шрифта и закодированные графические изображения символов.
Шрифт подключается строкой «extern uint8_t _ИМЯ_ШРИФТА_;». Спецификатор extern указывает, что массив с данными шрифта находится в другом файле, а имя шрифта берется из файла со шрифтом. Обратите внимание, это не имя файла, а имя самого шрифта (они могут различаться). Используемые шрифты должны быть объявлены в Вашей программе при помощи спецификатора extern. Например так:

```
extern uint8_t BigFont[];
```

Библиотека поставляется с тремя основными шрифтами:

- SmallFont – 95 символов 8x12
- BigFont – 95 символов 16x16
- SevenSegNumFont – 10 цифровых символов 32x50

Шрифт SevenSegNumFont имитирует 7-сегментный цифровой индикатор.
Дополнительные шрифты можно взять здесь:
http://www.henningkarlsen.com/electronics/r_fonts.php

А здесь находится онлайновый сервис, позволяющий сформировать собственные шрифты из специальным образом подготовленных графических файлов:
http://www.henningkarlsen.com/electronics/t_make_font_file.php

Шрифты, входящие в комплект библиотеки, хранятся в файле DefaultFonts.c в корневой папке библиотеки. Дополнительные шрифты (нарисованные самостоятельно или скачанные по вышеуказанной ссылке) рекомендуется располагать в папке Вашего проекта, иначе компилятор может их не найти.

Файлы шрифтов представляют собой определения массивов, содержащих служебную информацию и закодированное графическое изображение шрифта. Помимо названия в комментариях внутри файла шрифта находится также информация о размере символа в пикселях (эта информация будет полезна при задании координат вывода текста на дисплей), объеме памяти, занимаемом массивом данных шрифта и количестве символов, заданных в этом шрифте. Различают «полные» шрифты из 95 символов, «цифровые» (10 символов) и «подмножества» (другое количество символов).
Первые 4 байта массива содержат служебную информацию:

Байт 0 – размер шрифта по горизонтали в пикселях
Байт 1 – размер шрифта по вертикали в пикселях
Байт 2 – код первого символа шрифта (для «полных»шрифтов это 0x20, т. е. код пробела, для «цифровых» – 0x30 – код символа «ноль»).
Байт 3 – количество символов шрифта (для «полных» шрифтов 0x5F, для «цифровых» — 0x0A).

Подробнее о тонкостях формата и создании пользовательских шрифтов можно почитать здесь:
http://www.henningkarlsen.com/electronics/h_utft_fonts_101.php

Если Вы планируете работать с крупными стилизованными под 7-сегментный индикатор цифрами, рекомендуем сразу же заменить шрифт SevenSegNumFont на альтернативный шрифт SevenSegNumFontPlus, который можно скачать здесь:
http://www.henningkarlsen.com/electronics/dlfont.php?id=21&t=c
он отличается наличием весьма полезного символа «:», которого нет в стандартном шрифте. Файл шрифта следует поместить в папку с Вашей программой, заменить объявление extern uint8_t SevenSegNumFont[]; на extern uint8_t SevenSegNumFontPlus[]; после чего закрыть и снова открыть проект. Файл шрифта должен открыться вместе с проектом на соседней вкладке.

setFont – устанавливает шрифт для команд печати. В качестве параметра передается имя шрифта.
Шрифт должен быть предварительно объявлен в области определений программы. Пример:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t SmallFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.setFont(SmallFont);
myGLCD.clrScr();
myGLCD.print("Hello, World!",0,0);
```

```
}
void loop() {}
```

getFont – возвращает указатель на адрес текущего шрифта в памяти контроллера.
Рассмотрим пример. Чуть выше мы уже говорили о том, что первые четыре байта в массиве данных шрифта содержат информацию о свойствах шрифта. Эту информацию для текущего шрифта можно получить непосредственно из памяти контроллера и использовать в программе. Следующая программа помещает в переменные и выводит на дисплей размеры символа в пикселях, код первого символа и количество символов текущего шрифта, заданного командой setFont.

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
SerialUSB.begin(9600);
}
void loop() {
uint8_t* FontAddr = myGLCD.getFont();
byte CurrentFontX = *FontAddr;
byte CurrentFontY = *(FontAddr+1);
byte CurrentFontStartCode = *(FontAddr+2);
byte CurrentFontSymNumber = *(FontAddr+3);
myGLCD.printNumI(CurrentFontX, 0,0);
myGLCD.printNumI(CurrentFontY, 0,18);
myGLCD.printNumI(CurrentFontStartCode, 0,36);
myGLCD.printNumI(CurrentFontSymNumber, 0,54);
```

Аналогичным образом можно получить доступ к любому байту текущего шрифта.

getFontXsize – возвращает горизонтальный размер (ширину) символа текущего шрифта в пикселях. Параметров не имеет.

getFontYXsize – возвращает вертикальный размер (высоту) символа текущего шрифта в пикселях. Параметров не имеет.

Эти функции могут быть полезны при вычислении координат для команд печати. Примеры использования этих функций для расчета координат приведены в описании команд печати.
Далее мы рассмотрим несколько команд, предназначенных для вывода на дисплей символьной информации, т. е. команды печати.

print – выводит на дисплей текст, содержимое символьной переменной или объекта типа String. В качестве параметров передаются выводимый текст, координаты верхнего левого угла области печати. Еще один опциональный параметр позволяет располагать печатаемую строку с заданным наклоном.
Эта команда предназначена для вывода текстовой информации. Координаты печати X и Y задаются в пикселях и могут быть переданы как явно, так и через целочисленные переменные или выражения. Существуют также три предопределенных идентификатора, предназначенные для использования в качестве координаты X:

- LEFT –текст выравнивается по левой границе дисплея
- CENTER – текст выравнивается по центру дисплея
- RIGHT – текст выравнивается по правой границе дисплея

Текстовая информация может быть представлена в виде строки, заключенной в двойные кавычки:

```
myGLCD.print("Hello, World!", 0,0);
```

или в виде переменной типа String:

```
String text = "Hello, World!";
myGLCD.print(text, 0,0);
```

или в виде результата функции, возвращающей строковое значение:

```
int valueInt = 12345;
myGLCD.print(String(valueInt), 0,0);
```

```
float valueFloat = 12345.67;
myGLCD.print(String(valueFloat), 0,18);
```

Координаты задаются индивидуально для каждого вызова команды, текущая позиция печати командой не возвращается. Поэтому Вам необходимо самостоятельно рассчитывать позицию для следующей команды печати. Сделать это можно довольно простым способом. Следующий пример печатает два строковых значения подряд в одной строке (без пробела):

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
int X=0;
int Y=0;
int valueInt1 = 12345;
myGLCD.print(String(valueInt1), X,Y);
X=X+myGLCD.getFontXsize()*String(valueInt1).length();
int valueInt2 = 67890;
myGLCD.print(String(valueInt2), X,Y);
}
```

Для разделения выводимых значений пробелом расчетное значение X должно быть увеличено на ширину одного символа. А здесь печать осуществляется в две строки:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
int X=0;
int Y=0;
int valueInt1 = 12345;
myGLCD.print(String(valueInt1), X,Y);
Y=Y+myGLCD.getFontYsize();
int valueInt2 = 67890;
myGLCD.print(String(valueInt2), X,Y);
}
```

Использование предопределенных идентификаторов LEFT, CENTER и RIGHT для выравнивания текста показано в следующем примере:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
int Y=60;
myGLCD.print("Hello, World!", LEFT, Y);
Y=Y+myGLCD.getFontYsize();
myGLCD.print("Goodbye, World!", LEFT,Y);
Y=Y+myGLCD.getFontYsize()*4;
myGLCD.print("Hello, World!", CENTER, Y);
```

```
Y=Y+myGLCD.getFontYsize();
myGLCD.print("Goodbye, World!", CENTER,Y);
Y=Y+myGLCD.getFontYsize()*4;
myGLCD.print("Hello, World!", RIGHT, Y);
Y=Y+myGLCD.getFontYsize();
myGLCD.print("Goodbye, World!", RIGHT,Y);
}
```

Еще один опциональный параметр позволяет печатать строки, наклоненные под углом от 0 до 359 градусов. Вращение задается относительно координат печати (левый верхний угол). Нулевое значение угла приводит к обычной горизонтальной печати, далее по мере увеличения угла происходит вращение текста по часовой стрелке на заданный угол. Приведенный ниже пример позволяет получить забавный графический эффект:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
int X=240;
int Y=160;
for (int DEG=0; DEG<360; DEG+=20) {
String text = "Hello, World!";
myGLCD.print(text, X,Y, DEG);
}
}
```

При печати текста под углом использование предопределенных идентификаторов LEFT, CENTER и RIGHT не рекомендуется, т. к. они не могут быть адекватно вычислены библиотекой. Т. е. выравнивание все равно рассчитывается так, как будто строка печатается горизонтально, а поворот осуществляется уже после выравнивания и все равно относительно верхнего левого угла области печати.
Кстати, команды печати не умеют определять выход за пределы дисплея. Так что за максимальной длиной строки придется следить самостоятельно. Если строка окажется слишком длинной, то ее «хвост» может быть выведен поверх уже напечатанного текста. Если же вылезти за нижнюю границу дисплея – результат вообще может оказаться непредсказуемым.

printNumI – выводит на дисплей целое число или содержимое целочисленной переменной. В качестве параметров передаются выводимое значение и координаты верхнего левого угла области печати. Опциональные параметры позволяют управлять форматом вывода.
Координаты печати задаются так же, как и для команды print. При печати чисел со знаком в позиции X выводится знак числа, а затем – первая цифра. При печати беззнаковых чисел или положительных значений в позиции X выводится первая цифра числа.

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
void setup() {
myGLCD.InitLCD();
myGLCD.clrScr();
myGLCD.setFont(BigFont);
}
void loop() {
byte X=100;
byte Y=10;
int Num1 = 1234;
int Num2 = -1234;
unsigned int Num3= 12345;
myGLCD.printNumI(Num1,X,Y);
Y=Y+myGLCD.getFontYsize();
```

```
myGLCD.printNumI(Num2,X,Y);
Y=Y+myGLCD.getFontYsize();
myGLCD.printNumI(Num3,X,Y);
}
```

Выводимое на печать значение может быть передано в виде целого числа:

```
myGLCD.printNumI(1250,0,0);
```

или переменной одного из целочисленных типов:

```
int Num = 1324;
myGLCD.printNumI(Num,0,0);
```

Также возможно вывести результат любой функции или выражения, представляющий собой целочисленное значение:

```
myGLCD.printNumI(myGLCD.getFontYsize()*2,0,0);
```

При работе с целочисленными значениями необходимо помнить, что в Arduino DUE тип int хранится как и тип long в виде 4-х байтового числа с диапазоном допустимых значений от −2,147,483,648 до 2,147,483,647.
Типы unsigned int и unsigned long командой printNumI не поддерживаются, т. е. значение может быть передано команде, но будет выведено на дисплей как число со знаком. Тип char наоборот трактуется как беззнаковый.

Дополнительные опциональные параметры этой команды позволяют задать формат вывода чисел. Параметр length определяет минимальное количество знакомест (включая знак числа), занимаемых выводимым числом на дисплее. Если количество разрядов числа меньше, чем заданное значение length – недостающее количество знакомест дополняется слева нужным количеством символов. Параметр filler позволяет задать символ (по умолчанию — пробел), которым будет дополняться число. Комбинация этих символов позволяет, в частности, организовать вывод чисел, выровненных по правой границе или дополнять незначащими нулями значения при выводе времени или даты. Т. е. вместо привычной конструкции:

```
byte Day = 2;
byte Month = 9;
int Year = 2013;
if (Day<10){
myGLCD.print("0",0,0);
myGLCD.printNumI(Day,16,0);
}else{
myGLCD.printNumI(Day,0,0);}
myGLCD.print(".", 32,0);
if (Month<10){
myGLCD.print("0",48,0);
myGLCD.printNumI(Month,64,0);
}else{
myGLCD.printNumI(Month,48,0,2,'0');}
myGLCD.print(".", 80,0);
myGLCD.printNumI(Year,96,0);
```

можно просто написать:

```
byte Day = 2;
byte Month = 9;
int Year = 2013;
myGLCD.printNumI(Day,0,0,2,'0');
myGLCD.print(".", 32,0);
myGLCD.printNumI(Month,48,0,2,'0');
myGLCD.print(".", 80,0);
myGLCD.printNumI(Year,96,0);
```

не правда ли так гораздо лучше?

printNumF – выводит на дисплей вещественное число или содержимое переменной вещественного типа. В качестве параметров передаются выводимое значение, количество знаков после десятичной точки и координаты верхнего левого угла области печати. Опциональные параметры позволяют управлять форматом вывода.
Координаты печати задаются так же, как и в других командах печати. Выводимое вещественное значение может быть задано в виде числа, в виде переменной вещественного типа (float), а также в виде функции, возвращающей вещественный результат или выражения, результатом которого является вещественное число.
Количество знаков после десятичной точки может быть задано от 1 до 5. Нулевое значение не допускается (в этом случае рекомендуется привести число к целому типу и использовать команду printNumI). Например, команда:

```
myGLCD.printNumF(-234.3442, 2, 0, 0);
```

позволяет вывести на дисплей число −234.34
Число может быть задано как в экспоненциальной форме, так и в формате с «плавающей точкой», при этом вывод будет осуществлен в формате с «плавающей точкой». Например две команды:

```
myGLCD.printNumF(-2.343442E2, 2, 0, 0);
myGLCD.printNumF(-234.3442, 2, 0, 0);
```

Выведут на дисплей одно и то же значение −234.34
Опциональный параметр divider позволяет переопределить символ, выполняющий роль десятичной точки (по умолчанию – '.'). Например, команда:

```
myGLCD.printNumF(-234.3442, 2, 0, 0, ',');
```

позволяет заменить символ десятичной точки на запятую. При этом, естественно, вводить числа в программе нужно по-прежнему с использованием точки. Замена влияет только на вывод.
Опциональные параметры length и filler работают аналогично команде printNumF, позволяя управлять минимальным количеством символов при отображении числа. Параметр length учитывает все знакоместа, занятые отображаемым числом, включая знак, целую часть, десятичную точку и дробную часть. Например, команда:

```
myGLCD.printNumF(-234.3442, 2, 0, 0, '.', 10, '0');
```

выводит на дисплей число −000234.34 (исходное число, дополненное тремя незначащими нулями так, чтобы общее количество знакомест было равно 10).
ВНИМАНИЕ! При использовании опционального параметра length параметр divider обязательно должен быть задан, иначе система попытается интерпретировать значение length как код символа-разделителя, что может привести к непредсказуемым результатам.
Теперь перейдем к группе команд, позволяющих отображать на дисплее графические примитивы – точки, линии, прямоугольники и окружности.

drawPixel – выводит на дисплей точку. Цвет точки определяется текущим значением цвета, устанавливаемым командой setColor().
Координаты X и Y передаются в качестве параметров.
Параметры могут быть заданы числами, переменными, результатами функций или выражений. Параметры могут представлять собой вещественное число, но при этом дробная часть будет отбрасываться. Необходимо также следить, чтобы координаты точки не выходили за пределы дисплея, в противном случае точка может оказаться совсем не там, где вы ожидаете ее увидеть ;) Для контроля выхода за пределы дисплея можно воспользоваться рассмотренными ранее функциями getDisplayXsize() и getDisplayYsize().
Пример использования команды drawPixel – следующая конструкция выведет на дисплей синусоиду, построенную из отдельных точек:

```
for (int x=0; x<48; x++) {
int y=(sin(x)*10)+100;
myGLCD.drawPixel(x*10,320-y);
```

Следующая группа команд требует в качестве параметров координаты двух точек, определяющих размер изображаемой фигуры.

drawLine – выводит на дисплей линию заданную координатами начальной и конечной точек.
Направление рисования линии зависит от расположения начальной и конечной точек. Например, строка:

```
myGLCD.drawLine(10,20,100,200);
```

рисует текущим цветом линию между точками с координатами X=10, Y=20 для первой точки и X=100, Y=200 для второй. Строка:

```
myGLCD.drawLine(100,200,10,20);
```

рисует точно такую же линию между такими же точками, но направление рисования будет противоположным.

drawRect – выводит на дисплей прямоугольник, заданный координатами двух противоположных углов.
Координаты задаются так же, как и для команды drawLine.

drawRoundRect – выводит на дисплей прямоугольник со скругленными углами, заданный координатами двух противоположных углов.
Минимальный размер сторон прямоугольника ограничен 5 пикселями. При задании сторон меньшего размера прямоугольник не может быть выведен на дисплей.

fillRect – выводит на дисплей закрашенный прямоугольник, заданный координатами двух противоположных углов.
Прямоугольник рисуется и закрашивается текущим цветом.

fillRoundRect – выводит на дисплей закрашенный прямоугольник со скругленными углами, заданный координатами двух противоположных углов.
Эта команда функционирует так же, как и команда drawRoundRect.

Две следующих команды предназначены для рисования окружностей и кругов и требуют трех параметров: X и Y координат центра окружности (или круга) и радиуса.

drawCircle – выводит на дисплей окружность, определяемую координатами центра и радиусом.
Радиус окружности не должен принимать отрицательные значения, так как в этом случае она будет отображаться неправильно.
Следующая команда выводит на дисплей окружность с радиусом 50 пикселей и центром в точке с координатами X=100 и Y=120:

```
drawCircle(100,120,50);
```

fillCircle – выводит на дисплей закрашенный текущим цветом круг, определяемый координатами центра и радиусом.
При отрицательных значениях радиуса круг не отображается.

Итак, мы рассмотрели команды рисования графических примитивов. Кстати, для библиотеки UTFT существует дополнение UTFT_Geometry, которое позволяет выводить на дисплей треугольники (контурные и заполненные), дуги окружностей и сектора кругов. Работу с этим дополнением мы рассмотрим в одной из следующих статей.

А сейчас осталось совсем немного: две команды, позволяющие отобразить на дисплее специально подготовленное растровое изображение:

drawBitmap – выводит на дисплей специально подготовленное растровое графическое изображение.
В качестве параметров задаются координаты верхнего левого угла изображения, его размеры и имя массива, в котором хранится закодированное изображение.
Опциональный параметр scale позволяет управлять масштабированием изображения при выводе на дисплей.
Изображение для этой команды подготавливается при помощи специальной утилиты ImageConverter565.exe, которая поставляется вместе с библиотекой и располагается в папке Tools. А вот здесь можно воспользоваться онлайновым сервисом по конвертации изображений:
http://www.henningkarlsen.com/electronics/t_imageconverter565.php
Конвертер может работать с файлами форматов jpg, png и gif.

В качестве примера рассмотрим задачу вывода на дисплей растрового изображения и текстового сообщения на его фоне. Конечно, фоновое изображение размером 480x320 пикселей займет непозволительно много места в памяти контроллера, поэтому мы используем изображение, уменьшенное в 2 раза (240x160 пикселей) и воспользуемся масштабированием.
Для подготовки изображения необходимо запустить файл ImageConverter565.exe, и открыть в нем файл с Вашим изображением. Далее мы установим флаг «Reduce size to» и зададим требуемые размеры картинки (240x160). В переключателе «Save As» выберем «.c», а в переключателе «Target Board» выберем «chipKit/Arduino Due». В поле «Array Name» зададим имя картинки и нажмем «Save».

В результате работы конвертера мы получим файл с расширением .c, в котором будет храниться информация о картинке и закодированное изображение. Этот файл необходимо поместить в папку Вашего проекта и объявить в программе массив при помощи спецификатора extern так же, как мы это делали для шрифтов. Только в квадратных скобках обязательно нужно указать размер массива в 16-ричном формате (это значение находится в первом элементе массива, его можно посмотреть открыв полученный в результате конвертирования файл в любом текстовом редакторе).
Не забудьте после копирования и подключения массива закрыть и снова открыть файл программы. При этом файл массива откроется на соседней вкладке рядом с текстом программы. Текст нашей программы будет выглядеть так:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
extern unsigned short background[0x9600];
void setup() {
  myGLCD.InitLCD();
  myGLCD.setColor(0,255,0);
  myGLCD.setFont(BigFont);
  myGLCD.setBackColor(VGA_TRANSPARENT);
}
void loop() {
myGLCD.drawBitmap (0,0, 240, 160, background,2);
myGLCD.print("Hello, World!", CENTER,50);
while(1);
}
```

Таким образом, мы задали вывод в левый верхний угол дисплея картинки, размещенной в массиве с именем background размером

240 на 160 пикселей, но поскольку мы использовали параметр scale, равный двум – картинка будет растянута на весь дисплей. Далее поверх картинки с использованием параметра «VGA_TRANSPARENT», обеспечивающего «прозрачность» фона при печати, мы выводим текстовое сообщение. Довольно расточительно – около 100кб всего лишь на одну картинку, зато красиво ;)

Возможна еще одна форма вызова этой команды с дополнительными параметрами, позволяющая поворачивать изображение на заданный угол. Параметров в этом случае должно быть три: deg – угол поворота картинки rox – X координата центра вращения, roy – Y-координата центра вращения. Угол поворота задается в градусах (от 0 до 359), а координаты центра вращения rox и roy отсчитываются от левого верхнего угла картинки. Рассмотрим пример. В качестве изображения используем файл info.c, поставляемый вместе с библиотекой (он находится в подпапке /examples/Arduino (ARM)/UTFT_Bitmap корневой папки библиотеки). Размер картинки 32x32 пикселя, объем массива 0x400. Следующая программа выведет на дисплей вращающуюся вокруг своей оси картинку:

```
#include <UTFT.h>
UTFT myGLCD(CTE32HR,25,26,27,28);
extern uint8_t BigFont[];
extern unsigned short info[0x400];
void setup() {
  myGLCD.InitLCD();
  myGLCD.fillScr(VGA_WHITE);
}
void loop() {
for (int R=0; R<360; R+=10){
myGLCD.drawBitmap (100,100, 32, 32, info,R,16,16);
}
}
```

К сожалению, одновременно использовать вращение и масштабирование невозможно.

И в завершение еще несколько команд:

lcdOff – отключает дисплей. Параметров не имеет.

lcdOn – включает дисплей. Параметров не имеет.

После выполнения команды lcdOff() дисплей отключается и не реагирует ни на какие команды до тех пор, пока не будет выполнена команда lcdOn().
Эти две команды, согласно утверждению разработчика, реализованы только для дисплеев с контроллером PCF8833, поэтому на нашем дисплее они просто игнорируются.

setContrast – устанавливает контраст дисплея. Параметр – величина контраста.
В качестве параметра задается условная величина, определяющая контраст дисплея: от 0 (минимальный контраст) до 64 (максимальный контраст).
Эта команда также реализована только для дисплеев с контроллером PCF8833, поэтому на нашем дисплее изменения контраста не произойдет и команда будет проигнорирована.

Заключение

Итак, мы научились пользоваться всеми командами базовой библиотеки UTFT. В следующих статьях мы поговорим о дополнениях к этой библиотеке, позволяющих расширить ее возможности. На очереди – рисование треугольников, дуг и секторов, работа с сенсорным экраном, а также работа с текстовыми и графическими кнопками на дисплее, управляемыми при помощи сенсорного экрана. В дальнейшем мы планируем рассмотреть и встроенные возможности нашего дисплея – работу со внутренней памятью дисплейного модуля, зашитыми в его ПЗУ шрифтами и иконками.

arduino due, TFT, UTFT, дисплей

Вадим Елисеев @VadimEliseev карма 6,0 рейтинг 0,0