Графические контроллеры FT8xx. Работа с пользовательскими шрифтами и растровыми изображениями

Сергей ДОЛГУШИН
dsa@efo.ru

Пользовательские шрифты и изображения — неотъемлемая часть любого графического приложения. Работа с ними является одним из основных вопросов на этапе освоения микросхем FT8xx, поэтому мы рассмотрим программные средства, предлагаемые производителем, и основные приемы их применения. О работе с пользовательскими шрифтами и изображениями мы уже рассказывали ранее в [1–3]. В предлагаемой статье объединим всю эту информацию, а также ответим на некоторые часто встречающиеся вопросы.

Работа с пользовательскими шрифтами

Для вывода текстовой информации на экран дисплея в графических контроллерах FTDI предусмотрен ряд специализированных команд. Эти команды (например, CMD_Text или CMD_Number) используют для своей работы растровые шрифты. Графические контроллеры имеют предустановленный набор таких шрифтов. В микросхемах серии FT80x доступно 16 типоразмеров одного шрифта, в серии FT81x — 17. Каждый набор состоит из 127 символов, порядок которых соответствует стандартной кодировке ASCII.

Кроме встроенных шрифтов, графические контроллеры FTDI могут работать с пользовательскими шрифтами. В качестве пользовательского шрифта может выступать любой растровый шрифт, например, из состава шрифтов Windows (условия его применения определяются лицензионными соглашениями для каждого конкретного шрифта). Пользовательский шрифт тоже ограничен 127 символами. Перед использованием такого шрифта в программе он должен быть загружен в графическую память контроллера FT8xx. Теоретически в приложении одновременно может быть до 16 шрифтов. На практике мы ограничены размером графической памяти контроллера: 256 кбайт для FT80x и 1 Мбайт для FT81x.

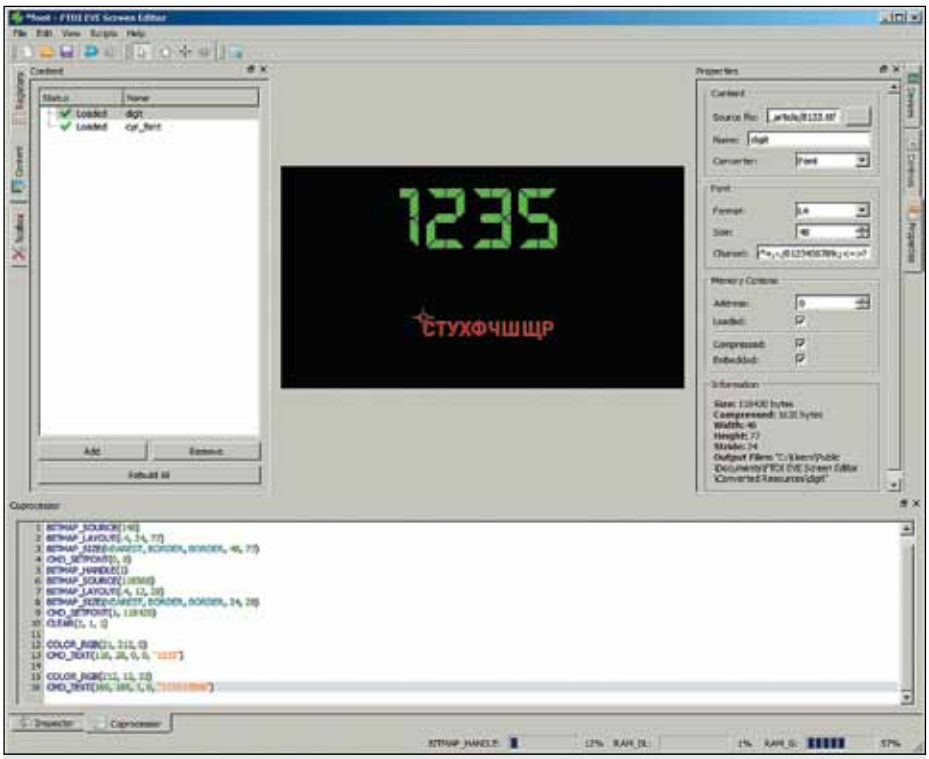

Рис. 1. Конвертирование шрифта в приложении Screen Editor

Для конвертации шрифта производитель предлагает специализированные утилиты, самой удобной из которых является Screen Editor v.1.17 (рис. 1) [4]. С ее помощью мы можем сразу оценить, как будет выглядеть шрифт на экране дисплея, какие объемы памяти микроконтроллера (МК) и графической памяти FT8xx потребуются для хранения шрифта и работы с ним. Данная утилита подходит для обоих семейств графических контроллеров, а методика работы со шрифтами для обоих семейств полностью одинакова. Сконвертированный шрифт имеет ряд параметров (ширина, высота и т. д.), описанных в статье [1]. При использовании Screen Editor все параметры рассчитываются автоматически и при генерации проекта в Arduino или MS Visual Studio (MSVC) подставляются в функции загрузки шрифта.

Процесс конвертации начинается с добавления выбранного шрифта в Screen Editor, это делается во вкладке **Content** -> **Add**. Для примера были выбраны два шрифта с сайта

Рис. 2. Свойства шрифта

Рис. 3. TFT-дисплей 7″ RVT70UQFNWC00

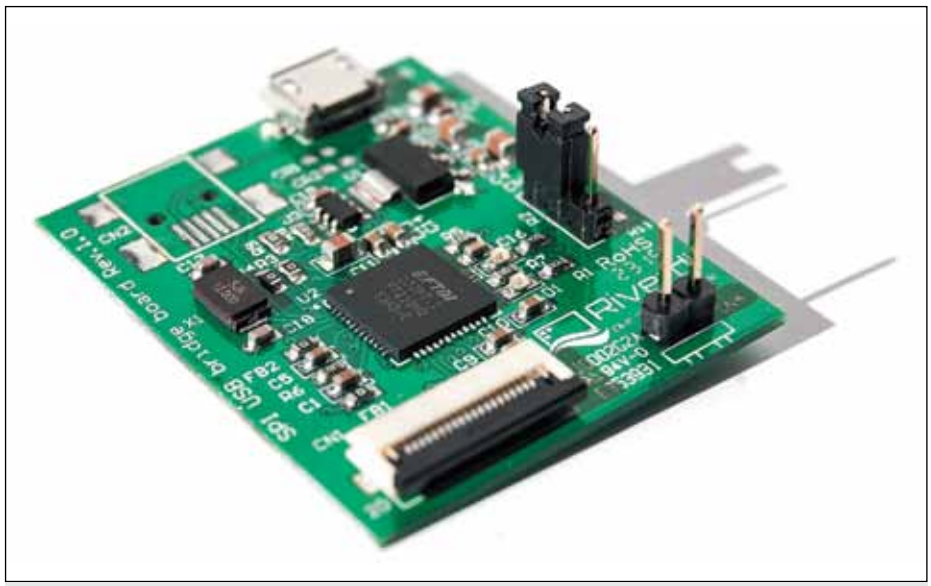

Рис. 4. Отладочная плата Hermes Board

http://www.fonts-online.ru/, чьи исходные файлы можно найти в этом архиве [5]. Как только шрифт добавлен, для него становятся доступными настройки параметров во вкладке **Properties** (рис. 2). Одновременно можно сразу перетащить добавленный шрифт в центральное окно утилиты. При этом мы сразу увидим, как оно будет выглядеть на экране дисплея (рис. 1), а в нижнем текстовом окне утилиты будет приведен набор команд, необходимых для его загрузки в графический контроллер и вывода на экран дисплея. Все манипуляции с параметрами нового шрифта синхронно отображаются в центральном окне утилиты. Это позволяет сразу оценить влияние того или иного параметра на качество отображения символов на экране дисплея.

В области **Content** для нас интересно только поле Name, в нем задается имя шрифта, которое потом будет использоваться при создании проекта для MSVC или Arduino. Остальные поля, чье назначение очевидно из названия, не трогаем и оставляем по умолчанию.

Область **Font** включает три поля, которые отвечают за то, как будет выглядеть шрифт на экране и какой набор символов будет в него включен. Поле Format задает степень сглаживания шрифта на экране. Для выбора доступны три варианта: L1 — минимальное сглаживание, L4 — среднее и L8 — максимальное. Поле Size отвечает за высоту шрифта, его выбирают, исходя из требований приложения. Поле Charset задает набор символов, которые будут включены в шрифт. По умолчанию в этом поле задан стандартный набор символов ASCII. Любой из этих символов мы можем заменить на свой при условии, что нужный символ есть в импортируемом шрифте.

Область **Memory Options** содержит поле Address для указания начального адреса, по которому шрифт будет загружаться в графическую память контроллера FT8xx. Одно из преимуществ утилиты Screen Editor — автоматический расчет начального адреса для каждого нового добавляемого графического объекта (шрифта или растрового изображения). Эта функция полезна в том случае, когда все пользовательские объекты могут быть помещены в графическую память RAM_G одновременно. Галочки Loaded, Compressed и Embedded остаются включенными по умолчанию.

Поле **Information** содержит информацию о физическом размере шрифта и параметрах символа. Подробнее с параметрами шрифта Width, Height и Stride можно ознакомиться в статье [1]. Физический размер шрифта дается для сжатого и распакованного видов. Первый показывает, какой объем памяти МК потребуется для хранения шрифта, второй — сколько графической памяти займет шрифт при загрузке его в графический контроллер. Эти размеры автоматически пересчитываются при любом изменении параметров шрифта (качество сглаживания, количество символов и высота), что позволяет сразу оценить внешний вид шрифта и объем памяти, необходимый для его хранения.

Теперь, когда мы подготовили шрифты для работы, можно приступить к генерации проекта. В данном примере для управления графическим контроллером предлагается персональный компьютер и MSVC. В качестве целевой платформы возьмем 7″-дисплей Riverdi серии uxTouch RVT70UQFNWC00 с разрешением 800×480 точек (рис. 3). Для подключения дисплея у нас есть специальная отладочная плата Hermes Board (рис. 4), изготовленная компанией Riverdi и представляющая собой конвертер USB-SPI на базе моста FTDI FT232H. На этапе освоения плата станет полезным дополнением — с ее помощью можно быстро протестировать работоспособность дисплея и разрабатываемого кода на персо-

нальном компьютере. Производитель предлагает примеры для MSVC, которые можно использовать в качестве заготовки для своих проектов. Они ничем не отличаются от примеров FTDI для MSVC, кроме параметров инициализации контроллеров FT8xx для работы с дисплеями Riverdi.

Итак, в качестве основы для нашего примера возьмем проект, предлагаемый компанией Riverdi для их дисплеев на базе микросхем FT81x [6]. Почему он нам нужен в качестве заготовки? К сожалению, на текущий момент инженеры FTDI не реализовали в новой версии утилиты Screen Editor 2.28 (с поддержкой новых контроллеров FT81x) функцию генерации кода для MSVC, как было описано в статье [4]. Поэтому, если мы используем дисплеи с новыми контроллерами, нам приходится работать с предыдущей версией утилиты Screen Editor v.1.17 с последующим копированием рабочего кода программы в вышеприведенный пример. Несмотря на кажущуюся сложность данной процедуры, такой путь наиболее прост на этапе освоения возможностей графических контроллеров нового семейства FT81x.

Начнем с подготовки примера Riverdi для дальнейшей работы с ним. Уберем из файла *SampleApp.c* лишний для нашей задачи код, оставив только необходимые функции. Получившийся проект можно скачать по ссылке [7].

Следующий этап — генерация проекта для MSVC с помощью Screen Editor. Выполняется эта процедура через меню **Scripts -> Export HAL (FTDI) Project**. Результатом станет готовый проект MSVC для графических контроллеров серии FT80x. Откроем файл *font_FTEVE_HAL.c* этого сгенерированного проекта и скопируем нужный нам код. Это адрес RAM_DIGIT с массивом ***digit[]*** первого шрифта и адрес RAM_CYR_FONT с массивом второго шрифта ***cyr_font[]***. Также скопируем основной код, в котором осуществляется загрузка обоих шрифтов в RAM_G и вывод на экран двух текстовых строк в подготовленный ранее проект:

```
Ft_Gpu_CoCmd_Dlstart(phost);
//Загрузка шрифтов в память контроллера
Ft_Gpu_Hal_WrCmd32(phost, CMD_INFLATE);
Ft_Gpu_Hal_WrCmd32(phost, RAM_DIGIT);
Ft_Gpu_Hal_WrCmdBuf(phost, digit, sizeof(digit));
Ft_Gpu_Hal_WrCmd32(phost, CMD_INFLATE);
Ft_Gpu_Hal_WrCmd32(phost, RAM_CYR_FONT);
Ft_Gpu_Hal_WrCmdBuf(phost, cyr_font, sizeof(cyr_font));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(0));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(148));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(L4, 24, 77));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, 48, 77));
Ft_Gpu_CoCmd_SetFont(phost, 0, 0);
Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(1));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(118568));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(L4, 12, 20));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, 24, 20));
Ft_Gpu_CoCmd_SetFont(phost, 1, 118420);
Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));

//Вывод текста на экран дисплея
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(21, 212, 0));
Ft_Gpu_CoCmd_Text(phost, 50, 4, 0, 0, "1235678");

Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(212, 12, 32));
Ft_Gpu_CoCmd_Text(phost, 69, 136, 1, 0, " 01292364");

Ft_App_WrCoCmd_Buffer(phost, DISPLAY());
Ft_Gpu_CoCmd_Swap(phost);
Ft_App_Flush_Co_Buffer(phost);
```

Этот код отвечает за загрузку шрифтов в графическую память контроллера FT8xx и вывод текстовой строки на экран. Проект Screen Editor, шрифты и сгенерированный утилитой проект доступны в архиве [5]. Откомпилируем и запустим проект — на экране дисплея появятся две текстовые строки (рис. 5).

При работе с пользовательскими шрифтами есть одно неудобство: символы пользовательского шрифта заменяют собой символы стандартной таблицы ASCII. Это не позволяет использовать в программе русский текст в текстовых командах Ft_Gpu_CoCmd_Text в прямом виде. Русские символы необходимо заменять соответствующими кодами ASCII или стандартными символами, как показано в примере выше. В MSVC это можно обойти следующим образом:

Рис. 5. Результат работы нашей программы на экране дисплея RVT70UQFNWC00

```
unsigned char mytext[] = "ПРИВЕТ";
int t;

for (t = 0; t<strlen(mytext); t++)
{
    mytext[t] = mytext[t] - 160;
}

while (1)
{

    Ft_Gpu_CoCmd_Dlstart(phost);
    Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));

    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(21, 212, 0));
    Ft_Gpu_CoCmd_Text(phost, 50, 4, 0, 0, "1235");

    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(212, 12, 32));
    Ft_Gpu_CoCmd_Text(phost, 69, 136, 1, 0, mytext);

    Ft_App_WrCoCmd_Buffer(phost, DISPLAY());
    Ft_Gpu_CoCmd_Swap(phost);
    Ft_App_Flush_Co_Buffer(phost);
}
```

Разница между кодами русских символов в таблице Win-1251 и в нашем шрифте cyr_font равняется 160. Эту разницу мы компенсируем в цикле для всех элементов текстовой переменной mytext в коде выше. Приведенный пример будет работать в MSVC, но не сработает, например, в Arduino, где используется другая кодировка символов, отличная от Win-1251. Соответственно, в зависимости от используемого компилятора пересчет придется выполнять по-разному. Все то же самое справедливо и для случаев, когда мы обращаемся к данным из внешней среды, полученным по тому или иному каналу

Рис. 6. Вывод на экран символов шрифта digit с размером 70

обмена, и пытаемся их отобразить на экране дисплея. Мы должны учитывать, в какой кодировке принимаются данные, и пересчитывать коды символов в соответствии с ней.

Как уже упоминалось, каждый шрифт может содержать до 127 символов. Мы можем задавать любое количество символов в пределах данной границы. Например, нам необходим специальный шрифт большого размера и содержащий только цифровые значения, как шрифт digit в примере на рис. 6. Мы можем оставить весь стандартный набор ASCII-символов, включая служебные: !"#$%&' ()*+,—./0123456789:;<=>? @ABCDEFGHIJKLMNOPQRSTUVWXYZ [\]^_`abcdefghijklmnopqrstuvwxyz{|}~. Такой подход приемлем до тех пор, пока для работы нашего приложения достаточно графической памяти микросхем FT8xx. Объем памяти, необходимый для хранения шрифта (МК) и работы с ним (FT8xx), пропорционально зависит от трех параметров — количества символов, размера шрифта (size) и сглаживания (форматы L1, L4 или L8). Служебные символы в диапазоне кодов с 0x00 по 0x31 таблицы ASCII хотя и не отображаются на экране, но тоже занимают память. В целях экономии памяти мы можем убрать из шрифта все элементы, которые не будут использованы в нашем приложении, например, оставив только цифры: 0123456789. Пробел в начале строки предназначен для сохранения служебного символа «конец строки» с кодом 0x00. Этот символ используется компилятором при работе со строковыми переменными. Если мы заменим его пользовательским элементом, функции работы со строками будут функционировать некорректно.

Работа с изображениями

Графические контроллеры FTDI могут работать с изображениями, сжатыми по алгоритму DEFLATE или представленными в формате JPEG или PNG. Перед выводом изображения на экран дисплея оно должно быть загружено в контроллер. В процессе загрузки картинка распаковывается и помещается по заданному адресу в графическую память микросхемы RAM_G. Тип изображения определяется командой загрузки изображения. Команда CMD_INFLATE сообщает контроллеру, что загружаемое изображение будет сжатым по алгоритму DEFLATE, CMD_LOADIMAGE — изображение в формате JPG или PNG.

Поддержка форматов JPG или PNG может быть полезной, если мы работаем с внешним источником, например с USB-накопителем [3]. Если же храним изображения во внутренней памяти МК, то сжатое по алгоритму DEFLATE изображение будет более удобно в работе.

На текущий момент для работы с изображениями, в отличие от работы со шрифтами, мы не сможем воспользоваться утилитой

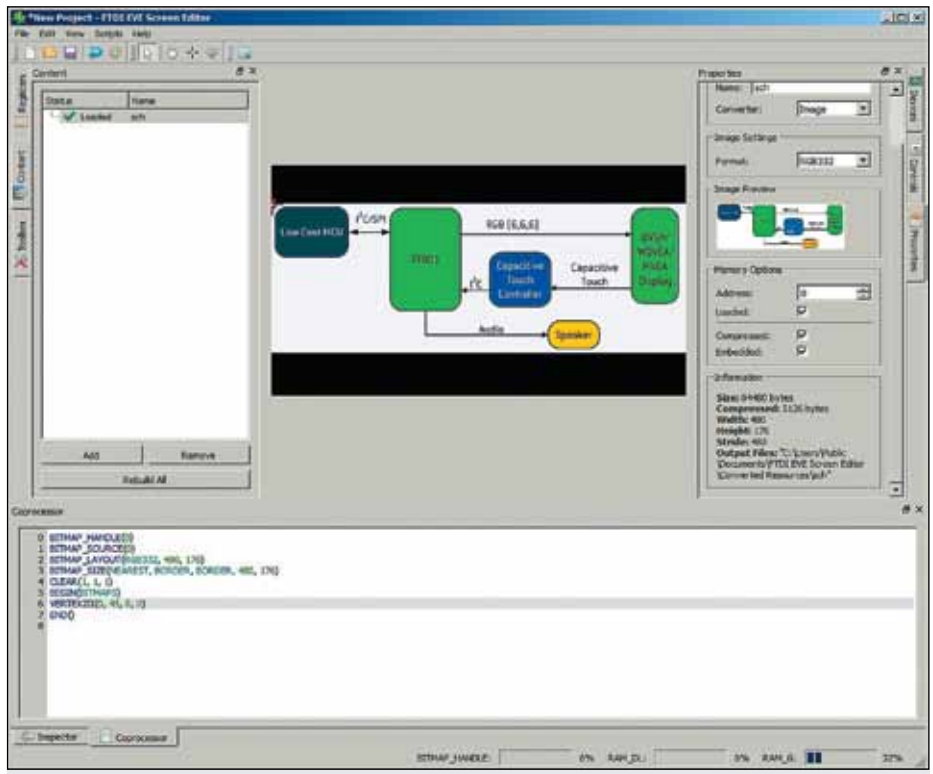

Рис. 7. Добавление изображения в Screen Editor

Screen Editor 1.17, если размеры изображения превышают 512×512 точек. Новая версия Screen Editor 2.2.8, в которую включена поддержка контроллеров FT81x, пока не экспортирует код в MSVC. Без такой функции утилита полезна только для оценки результата конверсии изображения и его размеров в сжатом и распакованном видах. Поэтому, если необходимы изображения большего размера, придется прибегнуть к конвертеру изображения *img_cvt.exe* [9], работающему из командной строки. Далее мы покажем порядок работы с изображениями для обоих случаев.

Начнем с самого простого и удобного на первых порах варианта, где для конвертации изображения будем использовать утилиту Screen Editor 1.17. В качестве тестовой платформы возьмем MSVC, плату Hermes Board и 7″ TFT-дисплей RVT70UQFNWC00. Данный дисплей на базе FT813 имеет разрешение 800×480 точек и с его помощью можно показать работу с изображениями 512×512 точек и более.

Утилита Screen Editor позволяет работать с изображениями в форматах JPG, PNG или BMP. После добавления изображений в программу они автоматически сжимаются по алгоритму DEFLATE. Добавление картинок осуществляется аналогично загрузке шрифта. Во вкладке Content нажимаем кнопку **Add** и выбираем требуемое изображение. Перетащив его из вкладки Content в центральное окно, добавляем изображение в наш проект (рис. 7). В свойствах изображения Properties можно выбрать формат вывода на экран. Заданный формат определяет качество представления картинки на экране, количество цветов и размер. Самое лучшее качество изображения в списке доступных форматов обеспечивает формат RGB565 (5 бит на красный канал, 6 — на зеленый, 5 — на синий, в общей сложности это составляет 65536 цветов). Для нашего изображения 480×176 точек размеры в сжатом и распакованном виде для этого формата составят 12 и 169 кбайт. Если мы поменяем формат на RGB332 (256 цветов) размеры будут 5 и 84 кбайт сответственно. Screen Editor позволяет сразу увидеть изменения качества отображения картинки на экране в зависимости от выбранного формата, а также оценить требуемый объем памяти. В поле Address задается начальный адрес размещения изображения в памяти RAM_G графических контроллеров. Адрес рассчитывается утилитой автоматически, что удобно при добавлении в проект следующего изображения и/или шрифта.

Экспорт проекта полностью соответствует тому, как это было описано для примера со шрифтами. В качестве заготовки проекта воспользуемся ранее подготовленным проектом [8]. Для сравнения вывода изображений в разных форматах добавим в проект на рис. 7 вторую копию блок-схемы и выберем для нее формат RGB565. Суммарный объем двух копий изображений не превысит 256 кбайт, поэтому Screen Editor позволит нам создать заготовку. Если изображения, добавленные в проект, превысят допустимый объем RAM_G, то утилита выдаст предупрежде-

ние о превышении размера. Перенесем получившийся код в проект MSVC:

```
#define RAM_SCH 0
static ft_uchar8_t sch[] = {120, … 156}; // изображение в формате RGB332

//Начало нашего примера
   Ft_Gpu_CoCmd_Dlstart(phost);
   Ft_Gpu_Hal_WrCmd32(phost, CMD_INFLATE); // загрузка первого изображения в RGB332
   Ft_Gpu_Hal_WrCmd32(phost, RAM_SCH);
   Ft_Gpu_Hal_WrCmdBuf(phost, sch, sizeof(sch));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(0));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB332, 480, 176));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, 480, 176));
   Ft_Gpu_Hal_WrCmd32(phost, CMD_INFLATE); // загрузка второго изображения в RGB565
   Ft_Gpu_Hal_WrCmd32(phost, RAM_SCH_565);
   Ft_Gpu_Hal_WrCmdBuf(phost, sch_565, sizeof(sch_565));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(1));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(84480));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565, 960, 176));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, 480, 176));
   Ft_Gpu_CoCmd_Swap(phost);
   Ft_App_Flush_Co_Buffer(phost);

while (1)
   {

      Ft_Gpu_CoCmd_Dlstart(phost);
      Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(255, 255, 255));
      Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));

      Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
      Ft_App_WrCoCmd_Buffer(phost, VERTEX2II(0, 0, 0, 0));
      Ft_App_WrCoCmd_Buffer(phost, END());

      Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
      Ft_App_WrCoCmd_Buffer(phost, VERTEX2II(0, 181, 1, 0));
      Ft_App_WrCoCmd_Buffer(phost, END());

      Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(156, 0, 2));
      Ft_Gpu_CoCmd_Text(phost, 14, 75, 28, 0, "RGB332");
      Ft_Gpu_CoCmd_Text(phost, 14, 258, 28, 0, "RGB565");

      Ft_App_WrCoCmd_Buffer(phost, DISPLAY());
      Ft_Gpu_CoCmd_Swap(phost);
      Ft_App_Flush_Co_Buffer(phost);
   }
```

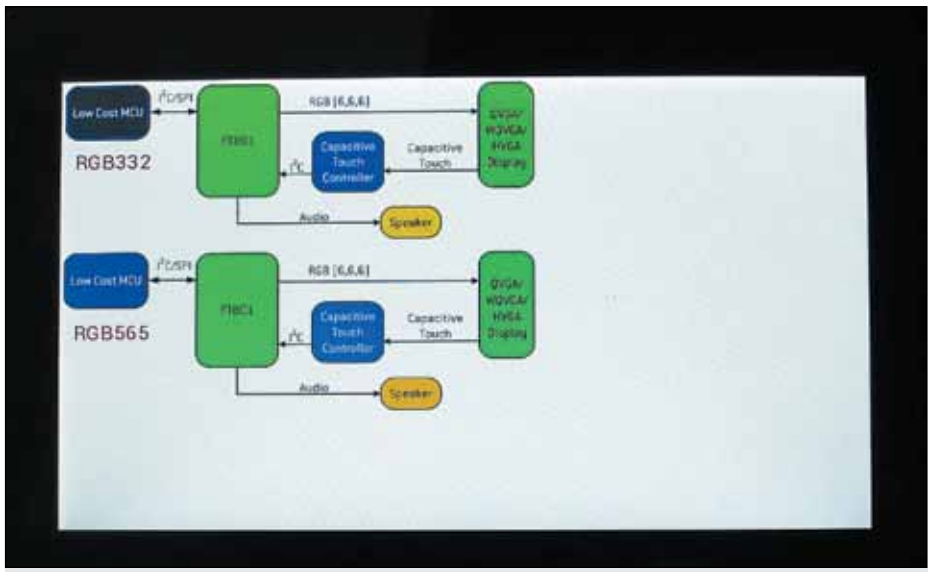

Рис. 8. Вывод двух изображений на экран дисплея

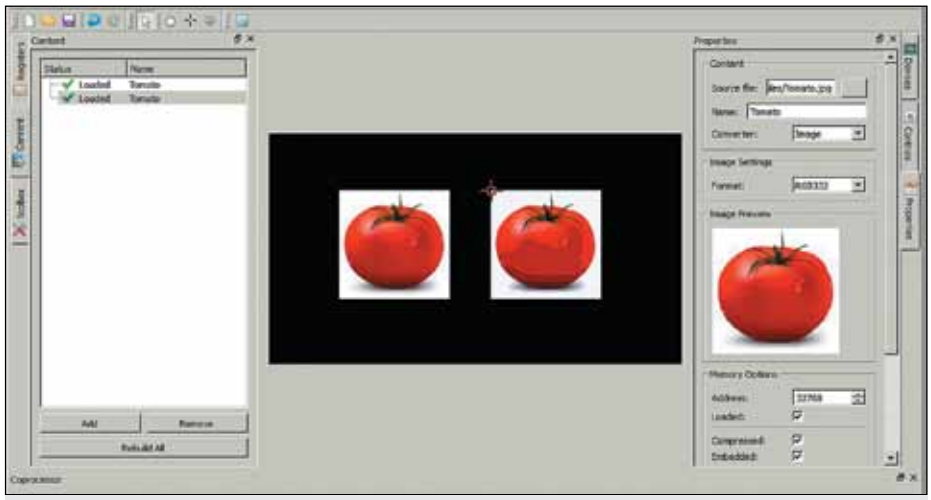

Рис. 9. Вывод двух изображений в окне Screen Editor

В результате выполнения этого кода на экране дисплея появится картинка, представленная на рис. 8. Как видно из рисунка, для такого изображения принципиальной разницы в использовании разных форматов нет. Небольшое отличие в оттенках цветов не влияет на восприятие изображения на экране дисплея.

Совсем другое мы будем наблюдать, если используем в качестве заготовки фотографию или изображение со сложными цветовыми переходами. На рис. 9 проиллюстрирована разница двух копий одного изображения: левое — в формате RGB565; правое — в RGB332.

Таким образом, правильно подобранное и подготовленное изображение может сэкономить память МК, требуемую для его хранения, а также память графического контроллера, куда это изображение будет распаковано. Чем больше используемых в нашем приложении изображений мы разместим в RAM_G, тем меньше операций по их загрузке понадобится в процессе выполнения приложения. Пусть в нашем приложении будет две картинки, чей суммарный размер не превышает объема памяти RAM_G FT8xx. В этом случае мы можем загрузить их в память контроллера один раз, например, в процессе инициализации приложения. Дальнейший вызов изображений на экран происходит из внутренней памяти контроллера, что не потребует передачи данных от МК в FT8xx. Если суммарный размер обоих изображений превышает объем RAM_G, то перед каждой сменой одного из этих изображений на экране дисплея нам придется выполнять загрузку текущей картинки в память графического контроллера.

В некоторых случаях утилитой Screen Editor 1.17 можно пользоваться для конвертации изображения и с большими размерами, чем 512×512 точек. Единственное условие для генерации кода в этом случае — только размер изображения. Он не должен превышать в распакованном виде объем памяти контроллеров FT80x, равный 256 кбайт. Для примера возьмем блок-схему с рис. 7, увеличим ее размер до 800×294 точек и заменим ею исходное изображение в предыдущем примере. В формате RGB332 размеры этого изображения в сжатом и распакованном виде будут следующие: 9,7 кбайт и 235 кбайт. То есть ее размер не превысит допустимого, и Screen Editor сгенерирует проект. В самой утилите мы увидим только часть изображения, ограниченного размерами 512×512 (рис. 10). Если мы без изменений скопируем код из сгенерированного проекта, на экране дисплея также отобразится только эта часть изображения, поскольку функции BITMAP_LAYOUT и BITMAP_SIZE из общего набора команд для FT80x и FT81x поддерживают только такие разрешения. Поэтому после экспорта проекта в MSVC необходимо добавить две дополнительные команды контроллеров FT81x — BITMAP_LAYOUT_H и BITMAP_SIZE_H. Эти команды добавляют по 2 старших бита каждому из параметров команд BITMAP_LAYOUT и BITMAP_SIZE, отвечающих за размеры изображения. Итоговый код будет выглядеть следующим образом:

```
//Начало нашего примера
  Ft_Gpu_CoCmd_Dlstart(phost);
  Ft_Gpu_Hal_WrCmd32(phost, CMD_INFLATE); // загрузка первого изображения в RGB332
  Ft_Gpu_Hal_WrCmd32(phost, RAM_SCH_800);
  Ft_Gpu_Hal_WrCmdBuf(phost, sch_800, sizeof(sch_800));
  Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(0));
  Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
  Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT_H(800, 294));


  Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB332, 800, 294));
  Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE_H(800, 294));
  Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, 0, 294));

  Ft_Gpu_CoCmd_Swap(phost);
  Ft_App_Flush_Co_Buffer(phost);


  while (1)
  {

    Ft_Gpu_CoCmd_Dlstart(phost);

    Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(255, 255, 255));
    Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));

    Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
    Ft_App_WrCoCmd_Buffer(phost, VERTEX2II(0, 0, 0, 0));
    Ft_App_WrCoCmd_Buffer(phost, END());

    Ft_App_WrCoCmd_Buffer(phost, DISPLAY());
    Ft_Gpu_CoCmd_Swap(phost);
    Ft_App_Flush_Co_Buffer(phost);
  }
```

Результат выполнения данного примера приведен на рис. 11.

В случае, когда изображение превышает 512×512 точек и его размер в распакованном виде больше объема памяти FT80x, нам придется пользоваться конвертером изображений *img_cvt.exe*, работающим из командной строки. Для конвертирования изображения надо выполнить следующие действия: скопировать исходное изображение в папку с утилитой *img_cvt.exe*; выбрать формат представления (RGB332, RGB565 и т. п., полный перечень поддерживаемых форматов описан в файле *ReadMe.txt*); запустить процесс преобразования командой ***img_cvt -i inputfilename -f format*** (где ***inputfilename*** — имя исходного изображения, ***format*** — формат сконвертированного изображения). В результате преобразования появится папка **«Имя исходного файла_формат»** с файлами со следующими расширениями: *.bin*, *.binh*, *.raw* и *.rawh* (рис. 12).

Рис. 12. Процесс преобразования изображения sch_800 в формат RGB332

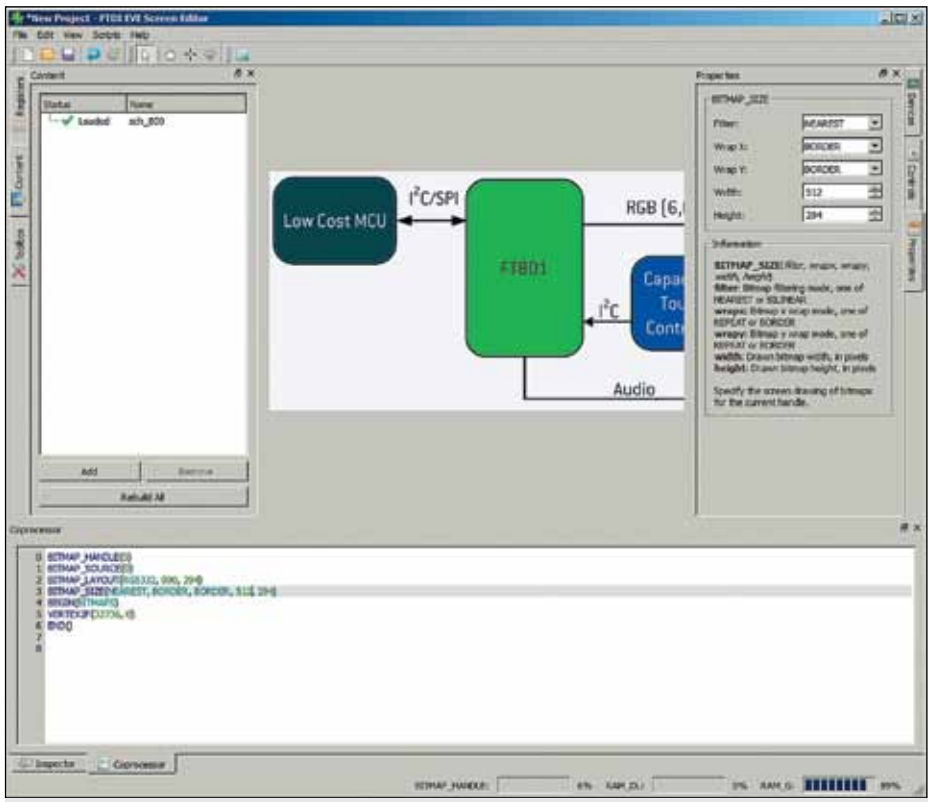

Рис. 10. Конвертация изображения с размерами 800×294 точки

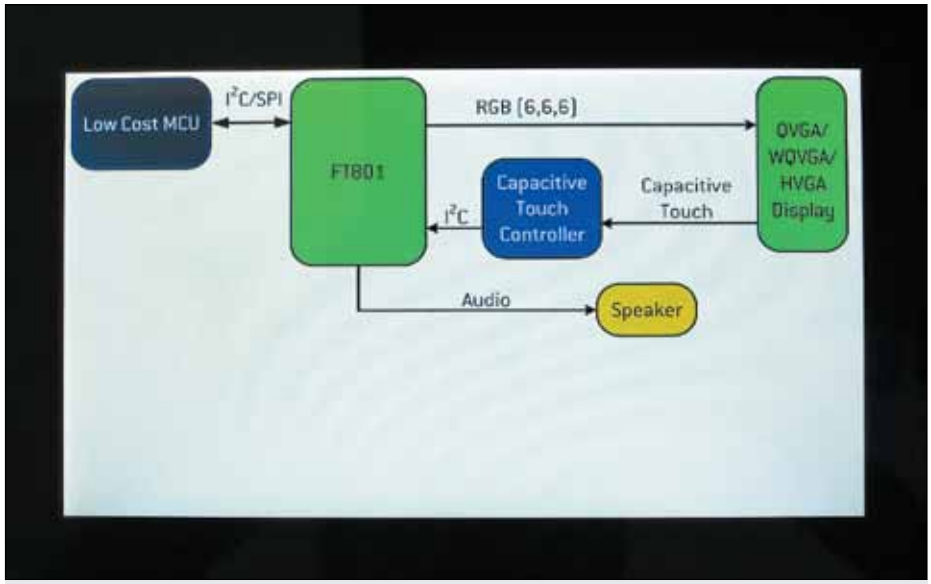

Рис. 11. Вывод изображения 800×294 точки на экран дисплея

Файлы raw и rawh содержат несжатое изображение, первый — в бинарном формате, второй — в текстовом. В общем случае для работы нам понадобится только rawh. В заголовке этого файла приводится информация о параметрах изображения, которые используются при его загрузке в графический контроллер в функциях BITMAP_SOURCE, BITMAP_LAYOUT и BITMAP_SIZE.

Файлы bin и binh содержат сжатое изображение: первый — в бинарном формате, второй — в текстовом. Формат bin может быть использован, если изображение предполагается хранить во внешней памяти. Формат binh применяется, если изображение хранится во внутренней памяти МК. Файл может быть добавлен в проект как заголовочный файл или в виде массива, как в приведенных примерах. Код для загрузки и вызова изображения

будет полностью идентичен приведенному в листинге выше. Архив с проектом MSVC и картинками, использованными во второй части статьи, доступен по ссылке [10].

Итак, в статье мы постарались ответить на наиболее часто задаваемые вопросы, которые возникают при первом знакомстве с графическими контроллерами FTDI, работе со шрифтами и изображениями. Надеемся, этот материал будет полезен и тем, кто только задумывается о возможности использования TFT-дисплеев в своих проектах и подбирает подходящее решение.

Литература

1. Долгушин С. Графический контроллер EVE FT800 FTDI. Работа с пользовательскими шрифтами, кнопками и сенсорным экраном // Компоненты и технологии. 2014. № 6.
2. Долгушин С. Графический контроллер EVE FT800 FTDI и микроконтроллер SAMD21 Atmel. Работаем с графическими изображениями // Компоненты и технологии. 2014. № 8.
3. Долгушин С. Графический контроллер EVE FT800 FTDI и микроконтроллер SAMD21 Atmel. Работаем с графическими изображениями // Компоненты и технологии. 2014. № 10.
4. Долгушин С. Графический контроллер FT800. Программные средства разработки и отладки // Компоненты и технологии. 2015. № 6.
5. **http://mymcu.ru/content/files/ftdi/fonts.rar**
6. **http://riverdi.com/wp-content/uploads/downloads/Hermes%20Board%20FT81X%20Demo%20source%20code_Rev.1.0.zip**
7. **http://mymcu.ru/content/files/ftdi/ft81x_demo_source_code.rar**
8. **http://mymcu.ru/content/files/ftdi/fonts.rar**
9. **http://www.ftdichip.com/Support/Utilities/EVE/Image_Convert_Tool_V0.7.zip**
10. **http://mymcu.ru/storage/content/articles/FTDI/fonts_bitmaps.rar**