Графический контроллер FT800. Вывод на экран изображения стрелочного индикатора и оптимальное использование ресурсов управляющего микроконтроллера

Сергей ДОЛГУШИН
dsa@efo.ru

В предыдущих материалах [1–5] мы познакомили читателей с функциями графического контроллера FTDI FT800, предназначенными для работы со шрифтами, графическими изображениями и базовыми графическими примитивами. В данной статье рассмотрены методы вывода на экран TFT-дисплея изображений стрелочных индикаторов, а также специальные функции микросхемы FT800, которые позволяют снизить нагрузку на управляющий микроконтроллер при выводе статических изображений на примере шкалы индикатора.

Одним из главных преимуществ графического контроллера FT800 перед конкурентными решениями является то, что он самостоятельно формирует изображение для вывода на экран. Управляющий микроконтроллер (МК) готовит набор команд (дисплей-лист) и передает их в FT800. В соответствии с этим дисплей-листом FT800 формирует изображение и выводит его на экран. При использовании конкурентных решений управляющий МК должен создавать картинку самостоятельно.

Другое существенное преимущество FT800 — встроенная графическая память RAM_G, хранящая пользовательские шрифты и графические изображения, в том числе в формате JPEG [3–5]. За счет этой возможности также снижаются требования к ресурсам управляющего МК.

Но возможности графической памяти не ограничены хранением растровых изображений. Она может быть использована и для хранения команд. Какой выигрыш это нам дает, покажем на примере формирования изображения стрелочного индикатора (рис. 1).

Рис. 1. Изображение шкалы индикатора на экране TFT-дисплея

Выбор стрелочного индикатора для демонстрации возможностей FT800 обусловлен тем, что на таком изображении максимально полно проявляется весь потенциал графического контроллера FTDI по экономии ресурсов управляющего МК. Это обусловлено тем, что изображение индикатора условно можно разделить на две части: статическую (изображение шкалы) и динамическую (изображение стрелки). При изменении контролируемого параметра нам необходимо перерисовать только стрелку, но из-за особенностей вывода на дисплей перерисовывать придется и шкалу, которая не меняется. С помощью функции FT800, позволяющей хранить команды в области памяти RAM_G, управляющему МК не придется обновлять шкалу полностью, пересылая полный набор команд для ее формирования. Достаточно дать FT800 указание воспользоваться набором команд, ранее записанным в его графическую память. Управляющий МК будет заниматься только расчетом нового местоположения стрелки и передачей обновленного списка команд для нее. В итоге обмен между FT800 и МК будет сокращен в десятки раз, что далее мы покажем на примерах.

Для работы с памятью производитель предусмотрел набор специальных команд, в нашем случае будут использоваться две из них.

Команда **MEMCOPY**, которая используется для копирования загруженного дисплей-листа из памяти RAM_DL в указанную область памяти, в данном примере в графическую. Так же как при сохранении растровых изображений, команды будут храниться в графической памяти RAM_G до ее очистки или сброса FT800.

Вторая команда — это команда **APPEND**. Она указывает графическому контроллеру, что в текущий дисплей-лист требуется добавить набор команд из указанного места. Таким образом, в основном цикле программы набор команд для шкалы будет заменен одной командой **APPEND**.

Настало время показать, как эти функции работают в программе. За основу проекта возьмем готовый пример производителя для среды

Рис. 2. Изображение шкал индикаторов на экране TFT-дисплея модуля VM800P

Рис. 3. Вид шкалы в зависимости от значения переменной Corner: а) 50; б) 10

разработки Arduino [6], демонстрирующий вывод на экран двух стрелочных индикаторов (рис. 2). Но для данного примера мы будем использовать свою библиотеку для работы с FT800 для MK SAMD21, описанную нами ранее [4, 5]. На наш взгляд, с помощью этой библиотеки удается более подробно раскрыть алгоритм работы FT800, тогда как в библиотеке для Arduino многие моменты скрыты внутри готовых функций и не всегда очевиден порядок вызова тех или иных команд.

Для формирования изображения стрелочного индикатора на экране дисплея потребуется в основном только функция для рисования линий. С помощью этой команды будет формироваться шкала индикатора и стрелка (рис. 3). Для удобства работы с указанной функцией предварительно сделаем преобразование прямоугольной системы координат, которая используется в FT800, в полярную систему. Следующие функции выполняют данную задачу:

```
static void polarxy(int32_t r, uint16_t th, int32_t×x, int32_t×y)
{
  ×x = (16×(FT_DispWidth/(2×noofch)) + (((long)r×qsin(th)) >> 11)
+ 16×ox); // ox — координата x центра шкалы
  ×y = (16×Centr – (((long)r×qcos(th)) >> 11)); // Centr — централь-
ная точка окружности. Коэффициент 16 требуется для функции
Vertex2F, в которой координаты x, y задаются, как 1/16 пикселя.
}
static void polar(int32_t r, uint16_t th) // функция задает координаты
точки в полярных координатах
{
  int32_t x, y;
  polarxy(r, th, &x, &y);
  vertex(x, y);
}
uint16_t da(int32_t i) // расчет угла луча
{
  return ((i – Corner)×32768L/360); // переменная Corner — за-
дает угол между первым делением шкалы и центральной осью.
На рис. 3а — Corner = 50, на рис. 3б — Corner = 10. Делитель 360
задает шаг между делениями равным 1°.
}
```

Для работы с цветами введем вспомогательную функцию, с помощью которой удобно задавать цветовую градацию шкалы индикатора:

```
static void cs(uint8_t i) // цвет делений, до 60 — зеленый,
до 80 — желтый, до 90 — красный
{
  switch (i)
  {
    case 0: Ft_App_WrCoCmd_Buffer(phost,COLOR_
RGB(29,170,76)); break;
    case 60: Ft_App_WrCoCmd_Buffer(phost,COLOR_
RGB(255,255,0)); break;
    case 80: Ft_App_WrCoCmd_Buffer(phost,COLOR_
RGB(255,0,0)); break;
  }
}
```

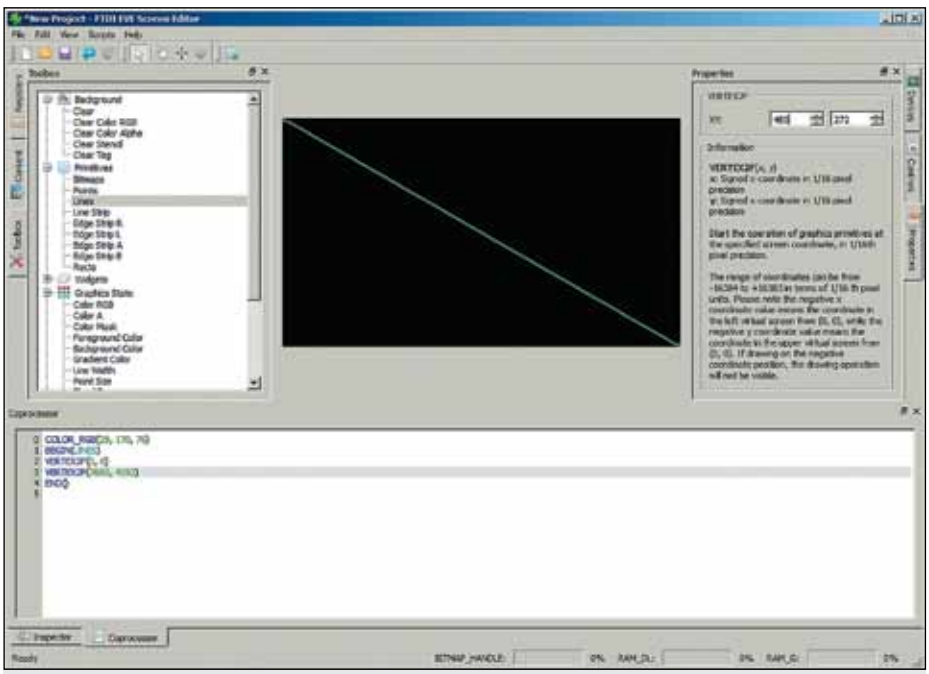

Рис. 4. Утилита EVE Screen Editor

Подготовив необходимые функции для работы с полярными координатами, можно переходить к основной задаче — формированию изображения шкалы. Для отображения делений шкалы воспользуемся графической функцией FT800 для рисования линий. Для того чтобы изобразить на экране дисплея линию, нужно передать в графический контроллер следующий набор команд: ***BEGIN(LINES)*** — сообщает FT800, с каким графическим примитивом будем сейчас работать; ***VERTEX2F(0, 0)*** — задаем координаты начальной точки линии; ***VERTEX2F(7680, 4352)*** — координаты конечной точки. В результате выполнения данных команд на экране будет отображена линия с начальными координатами 0,0 и конечными координатами 480,272 (рис. 4). Результат работы функции рисования линии приведен на рис. 4. Это окно утилиты EVE Screen Editor, предназначенной для изучения набора команд FT800 и их синтаксиса. Сама программа выложена в общий доступ на сайте производителя [8].

Вывод на экран дисплея делений шкалы будет выглядеть следующим образом:

Рис. 5. Вывод делений на экран

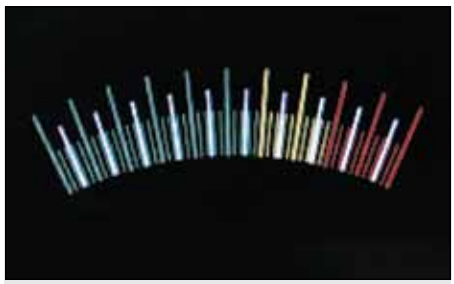

Рис. 6. Результат добавления делений

Рис. 7. Итоговый вид шкалы индикатора

```
Ft_App_WrCoCmd_Buffer(phost,CLEAR(255,1,1));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINES));

// единицы
Ft_App_WrCoCmd_Buffer(phost, LINE_WIDTH(10));
for (bi = 0; bi < 91; bi += 10)
{
   cs(bi); // выбор цвета линий
   for ( i = 2; i < 10; i += 2) // количество десятичных делений
{

a = da(bi + i); // угол поворота нового деления
polar(180, a); // начальная точка деления
polar(200, a); // конечная точка деления
}
}
// десятки
Ft_App_WrCoCmd_Buffer(phost, LINE_WIDTH(16));
for (i = 0; i < 101; i += 10)
{
   cs(i);
   a = da(i);
   polar(180, a);
   polar(220, a);
}
```

В результате выполнения этого кода на экране мы увидим следующее изображение (рис. 5). Количество делений задается в параметрах цикла. Количество циклов зависит от того, какое представление делений требуется. Например, добавив еще один цикл следующего вида, мы получаем еще один набор делений (рис. 6):

```
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
   Ft_App_WrCoCmd_Buffer(phost, LINE_WIDTH(12));
   // рисуем десятичные деления
   for (i = 5; i < 101; i += 10)
   {
      a = da(i);
      polar(180, a);
      polar(210, a);
   }
```

Последними элементами шкалы индикатора будут обозначения делений:

```
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255
   for (i = 0; i < 101; i += 10)
   {
   a = da(i);
   polarxy(230, a, &tx, &ty);
   Ft_Gpu_CoCmd_Number(phost,tx >> 4, ty >> 4,26,OPT_CENTER, i);

   }
```

Итоговый результат на экране дисплея иллюстрирует рис. 7.

Стрелка прибора формируется аналогично, с помощью команды для вывода линии. В простейшем случае можно обойтись одной линией — красная стрелка на рис. 8:

```
Ft_App_WrCoCmd_Buffer(phost, LINE_WIDTH(20));
   th = da(VAL); // расчет угла поворота стрелки в зависимости
от текущего значения VAL

   {
      polar(100, th);
      polar(200, th);
   }
```

Или более сложный вариант из нескольких линий — белая стрелка на рис. 8:

```
th = da(val);

for (o = –10; o < 11; o++)
   {
      polar(100, th + (o << 5));
      polar(200, th);
   }
```

Рис. 8. Варианты стрелок

На базе представленных выше функций могут быть реализованы различные варианты индикаторов. Как видно из представленных листингов, основной объем кода отводится на изображение статичной шкалы. Для шкалы в нашем примере мы должны передать в FT800 приблизительно 300×32-разрядных слов, или 1,2 кбайт. Далее рассмотрим, каким образом можно сократить объем данных, передаваемых от МК в FT800.

Как было сказано в начале статьи, одним из вариантов решения такой задачи является создание растровой картинки шкалы прибора. Картинка помещается в графическую память RAM_G [4,5] FT800 и вызывается в процессе выполнения программы. Программным путем формируется только стрелка. Этот метод позволит снизить нагрузку на управляющий МК в плане обмена по SPI, но потребует большего объема ПЗУ.

В данном примере наиболее эффективным путем решения задачи минимизации использования ресурсов управляющего МК будет метод, основанный на специальных командах FT800, предназначенных для работы с памятью. Такой подход напоминает по своему алгоритму работу с готовым растровым изображением, но в графическую память RAM_G мы помещаем не созданную картинку, а код дисплей-листа, который на порядок меньше, чем размер растрового изображения, даже если изображение хранится в сжатом виде.

Применение команд для работы с памятью позволит минимизировать обмен по SPI между управляющим МК и FT800. Статическая часть изображения загружается в FT800 один раз и далее используется по мере необходимости. Кроме того, с точки зрения использования памяти гораздо эффективнее хранить изображение в виде команд дисплей-листа, а не растрового изображения. Возвращаясь к нашему примеру с индикатором, следует отметить, что объем памяти, нужный для хранения одного растрового изображения шкалы индикатора, будет достаточен для хранения нескольких вариантов кода дисплей-листа для отображения разных шкал (рис. 9). В частности, размер шкалы не влияет на размер кода, тогда как увеличение растрового изображения потребует больше памяти для хранения.

Итак, на практике работа с памятью будет выглядеть следующим образом:

```
void Gauges()
{
…
Ft_Gpu_CoCmd_Dlstart(phost);
… // здесь рисуем шкалу индикатора
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

dloffset = Ft_Gpu_Hal_Rd16(phost,REG_CMD_DL); // размер копируемого дисплей-листа
Ft_Gpu_Hal_WrCmd32(phost,CMD_MEMCPY); // указание FT800 скопировать области памяти
Ft_Gpu_Hal_WrCmd32(phost,100000L); // адрес, куда будем копировать в области графической памяти RAM_G
Ft_Gpu_Hal_WrCmd32(phost,RAM_DL); // адрес, откуда копируем
Ft_Gpu_Hal_WrCmd32(phost,dloffset); // количество байт
}
```

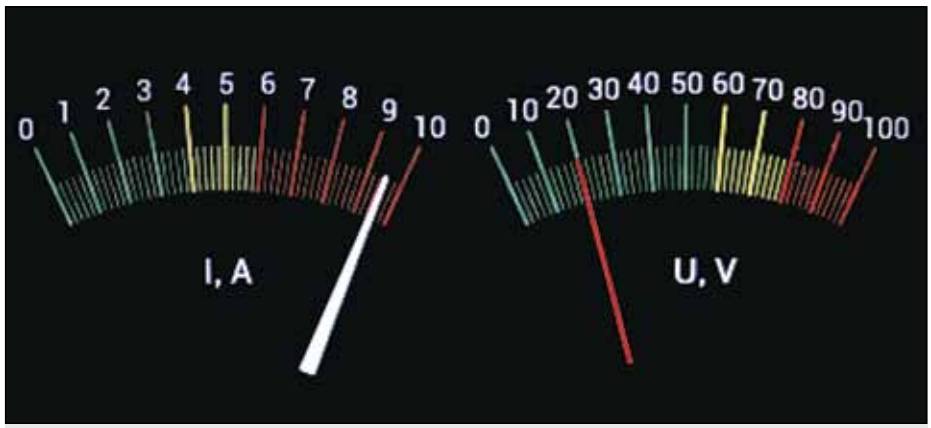

Рис. 9. Два индикатора на экране

Функция ***Gauges()*** содержит набор команд для отображения шкалы индикатора (рис. 7). Этот набор начинается со стандартной команды FT800 DLSTART, обозначающей начало нового дисплей-листа. Далее идет набор команд, необходимых для формирования нашей шкалы, и все заканчивается вызовом команды **MEMCOPY**. По этой команде происходит копирование команд дисплей-листа из памяти RAM_DL в графическую область RAM_G. Перед копированием следует получить от FT800 размер копируемого кода, что осуществляется чтением регистра REG_CMD_DL. Далее передаем в FT800 команду **MEMCOPY** с указанием адреса в области памяти RAM_G и адреса, откуда будем копировать код, а также размера этого кода. Теперь статическая часть экрана в виде соответствующего кода загружена в память FT800 и доступна для дальнейшего использования.

Обращаем внимание, что в приведенном листинге в конце дисплей-листа нет команд **DISPLAY** и **SWAP**. Это важный момент, обусловленный принципом работы FT800. Область памяти RAM_DL имеет два независимых массива, которые переключаются командой **SWAP**. В один массив записывается новый дисплей-лист, а из второго в этот момент происходит выполнение предыдущего. То есть по любому из адресов 0x10000–0x101FFF управляющему МК в текущий момент времени доступен только один массив памяти RAM_DL, а другой, из которого происходит выполнение программы, нет. После подачи команды **SWAP** массивы меняются местами. В документации производителя и во всех наших статьях под термином RAM_DL подразумевается массив, доступный для записи и чтения. Массив, из которого происходит выполнение предыдущего кода, нам не доступен до окончания вывода этого кода контроллером FT800 на экран дисплея и подачи команды **SWAP**.

Кроме этого, внимательный читатель должен был обратить внимание, что в примере мы работаем с командами сопроцессора и, следовательно, наш МК записывает команды дисплей-листа в область памяти RAM_CMD (FIFO), а копирование выполняется из памяти RAM_DL. Это связано с тем, что сопроцессор самостоятельно не осуществляет вывод изображения на экран. Он обрабатывает сложные команды и в виде простого набора команд записывает в память RAM_DL, из которой уже и происходит дальнейший вывод изображения на экран после команды **SWAP**. Поэтому, записав требуемый набор команд в RAM_CMD сопроцессора, копирование мы осуществляем из RAM_DL. Все внутренние операции работы с памятью скрыты от программиста и реализуются контроллером FT800 автоматически.

После записи нужного набора команд в графическую память RAM_G этот набор доступен для добавления в любом месте программы. Вызов кода, формирующего статическую часть изображения, осуществляется командой **APPEND** следующим образом:

```
Ft_Gpu_CoCmd_Dlstart(phost); // начало нового дисплей-листа
Ft_Gpu_CoCmd_Append(phost,100000L,dloffset); // добавление к текущему набору команд, команд скопированных MEMCPY

ox = 5;

Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(255,255,255));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINES));
Ft_App_WrCoCmd_Buffer(phost, LINE_WIDTH(10));

// рисование стрелки
th = da(val);
for (o = –10; o < 11; o++)
  {
    polar(100, th + (o << 5));
    polar(200, th);
  }

Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
Ft_Gpu_CoCmd_Swap(phost); // начало вывода изображения на экран
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

В основном цикле программы перед началом вывода стрелки в новое положение FT800 добавляет по команде **APPEND** код из графической памяти. В итоге дисплей-лист, на основе которого генерируется новое изображение, будет содержать шкалу и стрелку. При этом управляющий МК должен будет передать лишь одну команду **APPEND** размером 4 байт, вместо набора команд для шкалы размером около 1 кбайт.

Итак, в серии статей [1–5], посвященных графическому контроллеру FTDI FT800, мы постарались рассказать об основных моментах, принципиально отличающих его от существующих на текущий момент аналогов. В частности, наличие графической памяти RAM_G, в которой могут храниться пользовательские шрифты, сжатые растровые изображения и статические изображения в виде набора команд, позволяют в несколько раз снизить нагрузку на управляющий МК. Такими возможностями не может похвастаться ни одно из конкурентных решений аналогичного класса. Именно эти преимущества FT800 позволяют использовать TFT-дисплеи в полном объеме даже с 8-разрядными МК.

Литература

1. Долгушин С. Графический контроллер EVE FT800 компании FTDI // Компоненты и технологии. 2013. № 11.
2. Долгушин С. Начинаем работать с графическим контроллером FT800 FTDI // Компоненты и технологии. 2014. № 5.
3. Долгушин С. Графический контроллер EVE FT800 FTDI. Работа с пользовательскими шрифтами, кнопками и сенсорным экраном // Компоненты и технологии. 2014. № 6.
4. Долгушин С. Графический контроллер EVE FT800 FTDI и микроконтроллер SAMD21 Atmel. Работаем с графическими изображениями // Компоненты и технологии. 2014. № 8.
5. Долгушин С. Графический контроллер EVE FT800 FTDI и микроконтроллер SAMD21 Atmel. Работаем с графическими изображениями // Компоненты и технологии. 2014. № 10.
6. http://www.ftdichip.com/Support/SoftwareExamples/EVE/FT_App_Gauges.zip
7. http://www.mymcu.ru/support/otrisovka_strelochnogo_indikatora_s_pomoschyu/
8. http://www.ftdichip.com/Support/Utilities.htm