Графический контроллер EVE FT800 FTDI и микроконтроллер SAMD21 Atmel.

Работаем с графическими изображениями

Сергей ДОЛГУШИН
dsa@efo.ru

Данная статья продолжает тему [1—3], посвященную возможностям графического контроллера FT800 компании FTDI. Материал познакомит с функциями FT800, предназначенными для работы с растровыми изображениями. В качестве управляющего микроконтроллера в описываемых примерах фигурирует новая микросхема компании Atmel — SAMD21 ARM Cortex-M0+.

Работа с растровыми изображениями является одной из базовых задач при использовании цветного TFT-дисплея, наличие которого в приборе предполагает определенный уровень оформления информации. Однако графическая информация, нормально воспринимаемая на экране монохромного ЖК-дисплея, может оказаться неприемлемой при переходе на TFT-дисплей. Ведь конечный пользователь ожидает увидеть на таком дисплее именно то, что он видит, скажем, на экранах современных мобильных устройств. Таким образом, если мы хотим сделать красивый интерфейс, нам понадобятся растровые изображения, в свою очередь предназначенные для заставок, воспроизведения шкал приборов и т. д. А для экономии памяти и скорости загрузки предпочтительны сжатые изображения.

Рис. 1. Отладочная плата XPlained Pro ATSAMD21-XPRO и графический модуль VM800B43A

Такую возможность и предоставляет нам графический контроллер FT800, способный работать с изображениями, сжатыми по алгоритму Deflate и в формате JPEG (должны соответствовать спецификации JFIF). Декодирование изображения контроллер FT800 выполняет самостоятельно, без участия управляющего микроконтроллера (МК).

Для работы с растровыми изображениями, картинками и шрифтами предназначена графическая область памяти RAM_G, объем которой составляет 256 кбайт. После загрузки изображения в графическую область памяти FT800 оно будет доступно для вызова на экран до тех пор, пока в этой области памяти не появятся новые данные или не будет произведен сброс FT800. Такой подход позволяет уменьшить объем передаваемых данных между управляющим МК и FT800, например, при выводе на экран нескольких одинаковых картинок приборных шкал. Кроме того, несколько изображений могут быть загружены в память графического контроллера одновременно и выводиться на экран по мере необходимости.

Помимо вывода на экран исходного загруженного в память FT800 изображения, графический контроллер осуществляет и его трансформацию — изменяет физические размеры, вращает или перемещает картинку по экрану дисплея, как заставку (функция screen saver). Эти функции тоже выполняются без участия управляющего МК.

Для демонстрации работы FT800 с графическими объектами в качестве управляющего МК была выбрана новая микросхема Atmel SAMD21J18A на базе ядра Cortex-M0+ [4]. Наличие 256 кбайт флеш-памяти и USB-интерфейса с поддержкой режима хоста очень полезны при работе с графическими элементами. В частности, хост USB позволит продемонстрировать чтение JPEG-изображения с внешнего носителя и его вывод на экран TFT-дисплея.

В качестве среды разработки возьмем Atmel Studio 6, поскольку в нее входит Atmel Software Framework (ASF), применение которой существенно упростит выбор и настройку необходимых для проекта драйверов периферийных узлов. Это поможет сконцентрироваться

Таблица. Подключение ATSAMD21-XPRO и VM800B43A

VM800B43A	GND	3,3 В	SCK	MOSI	MISO	CS	PD
ATSAMD21-XPRO	PWR 2	PWR 4	EXT1 18	EXT1 16	EXT1 17	EXT1 15	EXT1 7

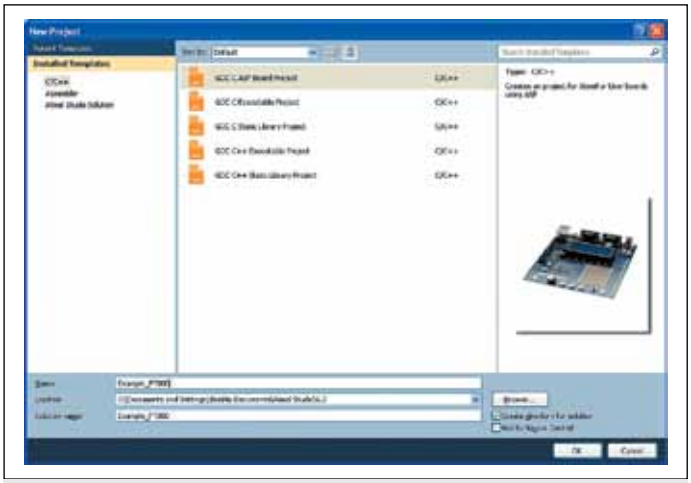

Рис. 2. Окно выбора шаблона проекта

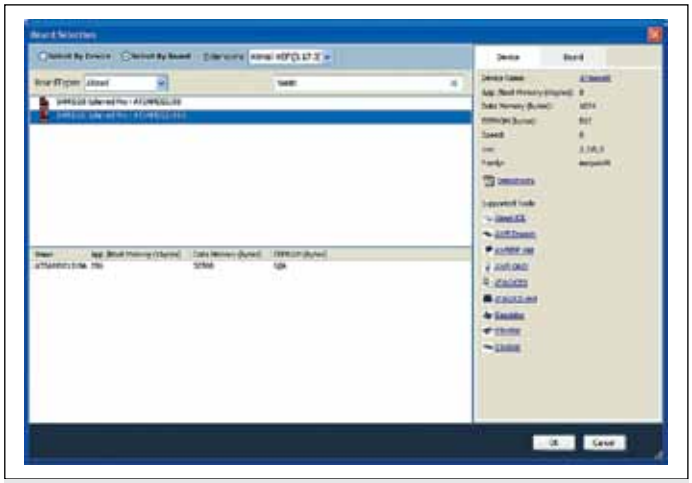

Рис. 3. Окно выбора целевой платы

на главном вопросе, то есть на работе с графическим контроллером FT800, не углубляясь в тонкости инициализации SAMD21.

Будем использовать отладочную плату XPlained Pro ATSAMD21-XPRO и графический модуль VM800B43A (рис. 1). Отладочная плата подключается к графическому модулю в соответствии с таблицей.

В качестве библиотеки для работы с FT800 воспользуемся примером производителя [5]. Процесс переноса библиотеки на платформу SAMD20/21 ничем не отличается от описанного ранее для МК Cypress [2]. В статье покажем только основные шаги по созданию проекта в Atmel Studio 6, которые необходимы для реализации управления графическим контроллером FT800 с помощью МК SAMD20/21.

На этом же примере продемонстрируем и методику работы с графическими объектами, преобразованными в бинарный формат по алгоритму Deflate, и изображениями в формате JPEG. Бинарный формат представления изображения удобен, если графические объекты будут храниться во встроенной памяти МК или на внешней микросхеме памяти. Для преобразования изображения в данный формат производитель предоставляет специальную утилиту Image Converter [6]. Если предполагается использовать внешние носители, например USB флеш-память или карту памяти SD, то формат JPEG будет более удобен. В данной статье опишем применение изображений, сжатых по алгоритму Deflate, а работу с JPEG — в следующей публикации.

Начнем с примера, в котором рассмотрим процесс формирования проекта, перенос библиотеки FTDI на SAMD20/21 и работу с растровыми изображениями. Откроем Atmel Studio и создадим новый проект через меню **File → New → Project**, в результате откроется окно выбора шаблона проекта (рис. 2). Для чего используем готовые шаблоны для нужной нам отладочной платы. Поэтому в появившемся окне выбираем первый пункт GCC ASF Board Project, вводим имя проекта и указываем место его хранения. В результате на следующем шаге ASF Wizard предложит выбрать целевую плату (рис. 3) и создаст для нее шаблон проекта. На данном этапе в проект будут включены все необходимые узлы для работы ядра и портов ввода/вывода.

Нам остается выбрать драйверы периферийных устройств и дополнительные функции, которые понадобятся для работы с FT800. Снова прибегнем к помощи ASF Wizard и вызовем его через меню **ASF → ASF Wizard**. В итоге будет открыто окно, в правой части которого отображен список драйверов, уже используемых в проекте, а в левой — всех поддерживаемых выбранной микросхемой драйверов (рис. 4). С помощью ASF добавим в проект драйверы SPI-интерфейса и управления вводом/выводом, функцию

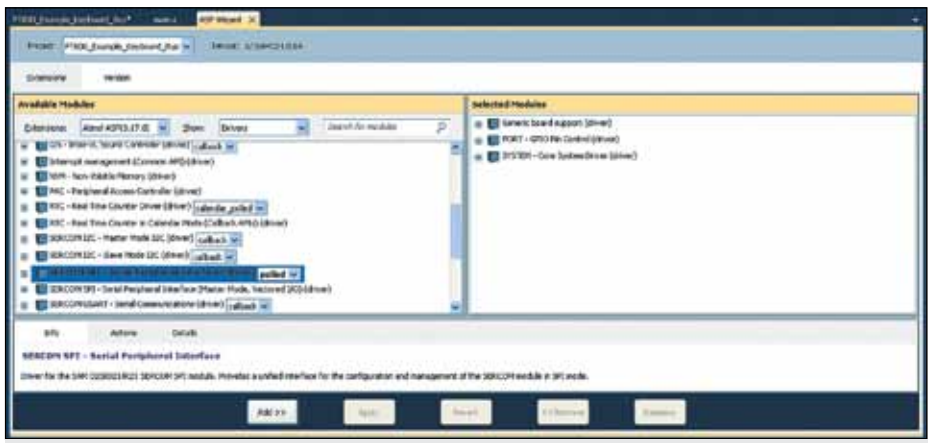

Рис. 4. Окно выбора драйверов

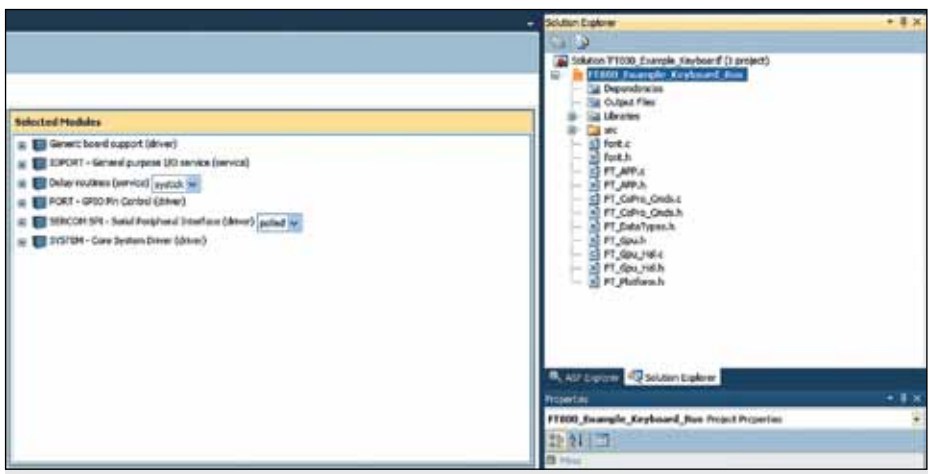

Рис. 5. Итоговый состав проекта

формирования задержек, а также файлы библиотеки FTDI. В результате проект будет выглядеть следующим образом (рис. 5).

Внесем соответствующие изменения в команды низкого уровня, описанные в файле *FT_GPU_HAL.c*, который был подробно рассмотрен в [2]. Для примера приведем модифицированную базовую функцию обмена между управляющим МК и FT800, предназначенную для чтения и записи одного байта в графический контроллер FT800:

```
ft_uint8_t Ft_Gpu_Hal_Transfer8(Ft_Gpu_Hal_Context_t *host,ft_
uint8_t value)
{

   ft_uint8_t Status;
   if (host->status == FT_GPU_HAL_WRITING)
      {
      Status=spi_write_buffer_wait (&spi_master_instance, &value, 1);
      }
      else
      {
      Status=spi_read_buffer_wait (&spi_master_instance, &value, 1, 0);
      }

   if (Status != STATUS_OK)
   host->status = FT_GPU_HAL_STATUS_ERROR;
   return value;
}
```

Архив с описываемым примером проекта доступен по ссылке [7], поэтому не станем повторно рассказывать об инициализации периферийных блоков МК SAMD21. Практически все нужные действия выполняет ASF Wizard, разработчику остается не забыть вызвать функции инициализации в начале программы. Также не будем останавливаться на процессе описанной ранее инициализации графического контроллера [1–3].

Итак, шаблон программы подготовлен и можно приступать к написанию кода для вывода изображения на экран графического модуля VM800B. Начнем с подготовки бинарного файла изображения с помощью утилиты Image Converter.

Утилита работает с графическими форматами **.PNG* и **.JPEG*. Взаимодействие с утилитой осуществляется из командной строки. Исходный файл изображения должен быть помещен в одну папку с утилитой, процесс преобразования запускается следующей командой:

```
img_cvt -i inputfilename -f format
```

где inputfilename — имя конвертируемого файла; format — формат бинарного файла: 0 : ARGB1555, 1 : L1, 2 : L4, 3 : L8, 4 : RGB332, 5 : ARGB2, 6 : ARGB4, 7 : RGB565, 8 : PALETTEED [8].

В результате работы конвертера создается папка, чье название включает имя конвертируемого файла и формат его представления. В папке находятся бинарные файлы в нескольких вариантах представления. Для нашего примера понадобится файл с расширением **.binh*, который является текстовым представлением бинарного файла. Добавим данные из этого файла в виде массива в программу в следующем виде:

```
ft_uchar8_t logo[] = {120,156,93,89…};
```

Подготовительный этап завершен, можно приступать к основной задаче — выводу изображения на экран. Прежде всего изображение должно быть загружено в графическую область памяти RAM-G контроллера FT800. Создадим следующую функцию для данной операции:

```
ft_void_t LoadToRam()
{
   Ft_Gpu_Hal_WrCmd32(phost, CMD_INFLATE);
   Ft_Gpu_Hal_WrCmd32(phost, 0);//destination address if inflate
   Ft_Gpu_Hal_WrCmdBufFromFlash(phost,logo,7200);
}
```

Функция начинается командой INFLATE, сообщающей FT800, что в память будет загружено изображение, сжатое по алгоритму Deflate. Следующая команда задает начальный адрес памяти, по которому будет доступно загружаемое изображение. Последняя команда загружает из массива с изображением данные в память FT800. Затем изображение будет доступно для использования до момента очистки памяти или сброса графического контроллера.

С помощью следующей функции осуществляется вывод изображения на экран, изменение прозрачности и цвета изображения:

```
ft_void_t SAMAPP_CoPro_Inflate()
{

   ft_int16_t xoffset,yoffset;

   xoffset = ((FT_DispWidth - 60)/2);
   yoffset = ((FT_DispHeight - 60)/2);

   Ft_App_WrCoCmd_Buffer(phost, CMD_DLSTART);
   Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(255,255,255));
// установка фона экрана в белый цвет
   Ft_App_WrCoCmd_Buffer(phost, CLEAR(1,1,1));
   Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255,255,255));
   Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS)); // начало
работы с графическими примитивами
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0)); // адрес
в графической памяти, где находится изображение
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565,120,60));
// указываются параметры изображения
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR,
BORDER,BORDER, 60,60)); // указываем размеры изображения,
метод сглаживания
   Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(xoffset*16,yoffs
et*16)); // вывод изображения с центром в точке x, y
   Ft_App_WrCoCmd_Buffer(phost, END()); // конец работы с изо-
бражением

   // вывод изображения в синем цвете
   Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565,
120,60));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR,
BORDER,BORDER, 60,60));
   Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(0,0,255)); // вывод
в синем цвете
   Ft_App_WrCoCmd_Buffer(phost, VERTEX2F((xoffset+30)*16,
(yoffset+30)*16));
   Ft_App_WrCoCmd_Buffer(phost, END());
   // вывод изображения в зеленом цвете
   Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565,120,60));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR,
BORDER,BORDER, 60,60));
   Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(0,255,0)); // вывод
в зеленом цвете
   Ft_App_WrCoCmd_Buffer(phost, VERTEX2F((xoffset+60)*16,
(yoffset+60)*16));
   Ft_App_WrCoCmd_Buffer(phost, END());
   // вывод изображения в красном цвете с прозрачностью равной 125
   Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565,
120,60));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR,
BORDER,BORDER, 60,60));
   Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255,0,0)); // вывод
в красном цвете
   Ft_App_WrCoCmd_Buffer(phost, COLOR_A(125); // установка
прозрачности, равной половине диапазона
   Ft_App_WrCoCmd_Buffer(phost, VERTEX2F((xoffset+90)*16,
(yoffset+90)*16));
   Ft_App_WrCoCmd_Buffer(phost, END());

   Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
   Ft_Gpu_CoCmd_Swap(phost);
   Ft_App_Flush_Co_Buffer(phost);
   Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
   delay_s(5);

   }

}
```

Обратим особое внимание на команду ***BITMAP_LAYOUT(RGB565,120,60)***. Данная команда указывает контроллеру FT800 на формат вызываемого изображения, его размер в памяти и его физические размеры. Для формата RGB565 [8] информация о цвете пикселя хранится в двух байтах, количество бит на цвет указано в названии формата. Второй параметр равен произведению ширины на количество байтов, требуемое для отображения одного пикселя. Третий параметр представляет собой высоту изображения. Остальные функции не нуждаются в особых комментариях, кроме тех, что указаны в листинге.

Добавим вызов описанных функций в программу следующим образом:

```
LoadToRam();

while (true)
{
   SAMAPP_CoPro_Inflate();
}
```

Такой порядок вызова наглядно показывает, что изображение загружается в память FT800 один раз. Для повторного вывода этого графического элемента на экран дисплея FT800 обращается к своей памяти. Повторная передача изображения от управляющего МК в графический контроллер не требуется. В конкурентных решениях каждый вывод изображения на экран предполагает новый цикл обмена.

Результатом работы программы будет вывод на экран графического модуля следующего изображения (рис. 6).

Изображение, хранящееся в памяти графического контроллера, можно выводить на экран, изменяя его размеры и/или угол поворота. Для этого предусмотрен набор из шести параметров TRANSFORM A-F, изменяя которые мы можем манипулировать изображением [8].

Рис. 6. Вывод изображения на экран

Рис. 7. Вывод увеличенного изображения на экран

Продемонстрируем влияние этих параметров на масштаб изображения, увеличив исходное изображение в два раза. Добавим в функцию SAMAPP_CoPro_Inflate() следующий код:

```
Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565,120,60));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_TRANSFORM_A(60*2)); // увеличиваем высоту в 2 раза
Ft_App_WrCoCmd_Buffer(phost, BITMAP_TRANSFORM_E(60*2)); // увеличиваем ширину в 2 раза
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR,BORDER,BORDER, 120,120));
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255,255,255));
Ft_App_WrCoCmd_Buffer(phost, COLOR_A(0);
Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(120,120));
Ft_App_WrCoCmd_Buffer(phost, END());
```

Итоговое изображение на дисплее примет следующий вид (рис. 7). В верхнем левом углу экрана дисплея отображается увеличенное изображение исходной картинки.

Изображения, хранящиеся в графической памяти микросхемы FT800, могут быть использованы для реализации анимированной заставки. Специализированная функция CMD_SCREENSAVER осуществляет перемещение выбранного изображения по экрану дисплея без участия управляющего МК. Следующая функция продемонстрирует эту возможность:

```
ft_void_t ScreenSaver()
{
   ft_int16_t xoffset,yoffset;
   xoffset = ((FT_DispWidth - 60)/2);
   yoffset = ((FT_DispHeight - 60)/2);

   Ft_App_WrCoCmd_Buffer(phost, CMD_DLSTART);
   Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(0,0,255));
   Ft_App_WrCoCmd_Buffer(phost, CLEAR(1,1,1));
   Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255,255,255));

   Ft_App_WrCoCmd_Buffer(phost, CMD_SCREENSAVER); // выбор автономного режима заставки
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(0));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565,120,60));
   Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR,BORDER,BORDER, 60,60));
   Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
   Ft_App_WrCoCmd_Buffer(phost, MACRO(0));
   Ft_App_WrCoCmd_Buffer(phost, END());

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

Команда CMD_SCREENSAVER указывает графическому контроллеру, что следующая за ней картинка будет предназначена для экранной заставки. Вместо команды VERTEX, определяющей координаты вывода изображения, используется команда MACRO, обеспечивающая автоматическое обновление координат и вызов команды VERTEX. В результате работы данной функции изображение будет плавно перемещаться по экрану дисплея. Для завершения работы экранной заставки управляющий МК должен передать команду CMD_STOP.

Рис. 8. Пример экранной клавиатуры

В завершение статьи необходимо сказать то, о чем не было упомянуто в [3]. Учитывая, что пользовательские шрифты также являются растровыми изображениями, методика работы с ними идентична работе с картинками. То есть перед их применением в приложении они должны быть загружены в графическую область памяти. Так же, как и картинки, они будут доступны в приложении в любое время до момента очистки памяти или сброса контроллера FT800. В продолжение статьи [3] приведем модифицированный пример производителя, реализующий экранную клавиатуру. Встроенный шрифт заменен кириллицей, а сам проект портирован на МК SAMD21 (рис. 8). Архив с данным проектом доступен по ссылке [9].

В следующей статье будет рассмотрена работа с изображениями в формате *.*JPEG*. Основное внимание будет уделено работе SAMD21 с USB флеш-диском, чтению с него JPEG-файлов и отображению их на экране. Немного коснемся формата *.*JPEG* и методики считывания из файла изображения данных о его размерах.

В заключение отметим, что интерес к FT800 в мире побудил компанию FTDI продолжать развитие данной линейки микросхем. В конце 2014 года можно ожидать появления новой микросхемы FT801 с поддержкой контроллеров емкостных экранов, а также готовых модулей на ее базе. В начале 2015 года в планы производителя также входит выпуск микросхем с поддержкой дисплеев с большим разрешением экрана. Это подтверждает, что функциональные возможности микросхемы FT800 из новой для FTDI линейки графических контроллеров TFT-дисплеев являются востребованными. Благодаря им FT800 не имеет аналогов в своем классе на текущий момент.

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. Сазанов Д. SAMD — новая линейка микроконтроллеров с ядром ARM Cortex-M0+ компании Atmel // Компоненты и технологии. 2014. № 5.
5. Application Note AN 245. FT800 Sample Application Introduction for VM800B and VM800C Development Kits and Windows PC.
6. **http://www.mymcu.ru/support/eve_image_converter/**
7. **http://www.mymcu.ru/content/devtools/FTDI/Example_FT800_SAMD.rar**
8. FT800 Programmer Guide.
9. **http://www.mymcu.ru/content/devtools/FTDI/FT800_Example_Keyboard_rus.rar**