Графический контроллер EVE FT800 компании FTDI

В начале 2013 года компания FTDI анонсировала новую микросхему — графический контроллер EVE FT800. Поводом для ее создания послужил возрастающий интерес производителей конечного оборудования к использованию TFT-дисплеев во встраиваемых приложениях.
С помощью графического дисплея с сенсорным экраном можно реализовать удобный, интуитивно понятный и привлекательный интерфейс. Новый графический контроллер EVE компании FTDI позволит использовать возможности TFT-дисплеев даже с 8-разрядными микроконтроллерами. Встроенные графические функции для отображения простых графических примитивов (точек, линий и т. д.), а также сложных объектов (кнопок с эффектом объема, слайдеров и т. д.) положительно отличают контроллер FT800 от существующих на данный момент аналогов и снижают требования к вычислительным возможностям и объему памяти хост-контроллера. А встроенные контроллер сенсорного экрана и аудиоконтроллер дают возможность реализовать все функции пользовательского интерфейса на одной микросхеме FT800.

Сергей ДОЛГУШИН
dsa@efo.ru

Введение

Новый графический контроллер EVE (Embedded Video Engine) FT800 смело можно назвать уникальным решением на данный момент. Ни один из существующих графических контроллеров не обладает такими возможностями, которые заложены в FT800. Компания FTDI позиционирует его как законченное решение для создания графического пользовательского интерфейса. Микросхема FT800 позволяет реализовать графический интерфейс пользователя на базе TFT-дисплея. Взаимодействие с интерфейсом обеспечивается при помощи резистивного сенсорного экрана и звуковых эффектов. Все эти возможности аппаратно поддерживаются графическим контроллером EVE FT800 (рис. 1).

Такой функциональный набор позволяет сравнивать FT800 с программно-аппаратной реализацией компании 4D Systems или программным вариантом компании MikroElektronika, а никак не с используемыми в стандартных TFT-дисплеях графическими контроллерами Solomon, Himax и другими.

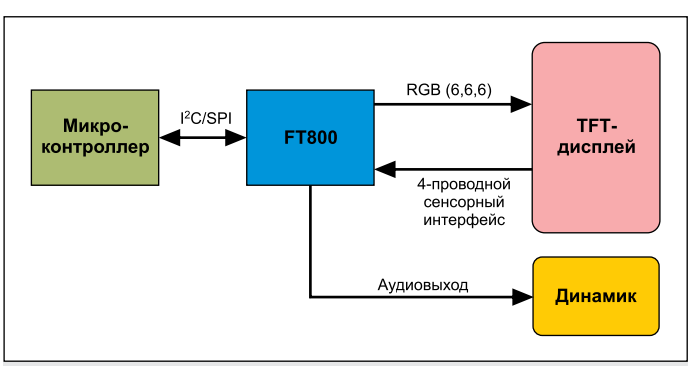


Рис. 1. Графический контроллер FT800

С продукцией компании 4D Systems и функциональными особенностями ее графических контроллеров и модулей можно познакомиться в [1]. Главным плюсом решения 4D Systems является специализированная и бесплатная среда разработки “4D Workshop IDE” для графических контроллеров собственной разработки. Эта среда предоставляет простые в использовании инструменты для создания графического интерфейса; разработка приложения занимает короткое время и интуитивно понятна. Но сами графические модули этого производителя дороги, что ограничивает их применение. Их целесообразно использовать в мелкосерийных изделиях, где скорость разработки важнее стоимости конечного изделия. Одной из причин высокой стоимости является то, что микросхемы 4D Systems (Goldelox, Picaso и Diablo) работают с TFT-дисплеями через стандартные графические контроллеры типа Solomon SSD1963. А TFT-дисплеи со встроенными контроллерами такого типа стоят на 30–50% больше, чем дисплеи с интерфейсом RGB.

Программной альтернативой для работы с TFT-дисплеями являются библиотеки и специализированный софт компании MikroElektronika. Эта компания занимается разработкой отладочных и программных средств разработки, в том числе и для работы с дисплеями. Ее графическая среда разработки Visual TFT, схожая по методологии с 4D Workshop IDE, предназначена для работы с графическими контроллерами разных производителей, например Solomon SSD1963 или Himax HX8347. Эти контроллеры широко используются производителями дисплеев, в частности, компания Winstar часто применяет в своих TFT-модулях контроллер SSD1963. Отметим также, что в последнюю версию Visual TFT включена поддержка и графического контроллера FTDI FT800. Таким образом, MikroElektronika предоставляет удобные библиотеки и графическую среду для работы со стандартными TFT-модулями разных производителей. Главный минус такого решения — эти библиотеки можно использовать только с компиляторами, предлагаемыми компанией MikroElektronika, а сама среда разработки Visual TFT и компиляторы не бесплатные.

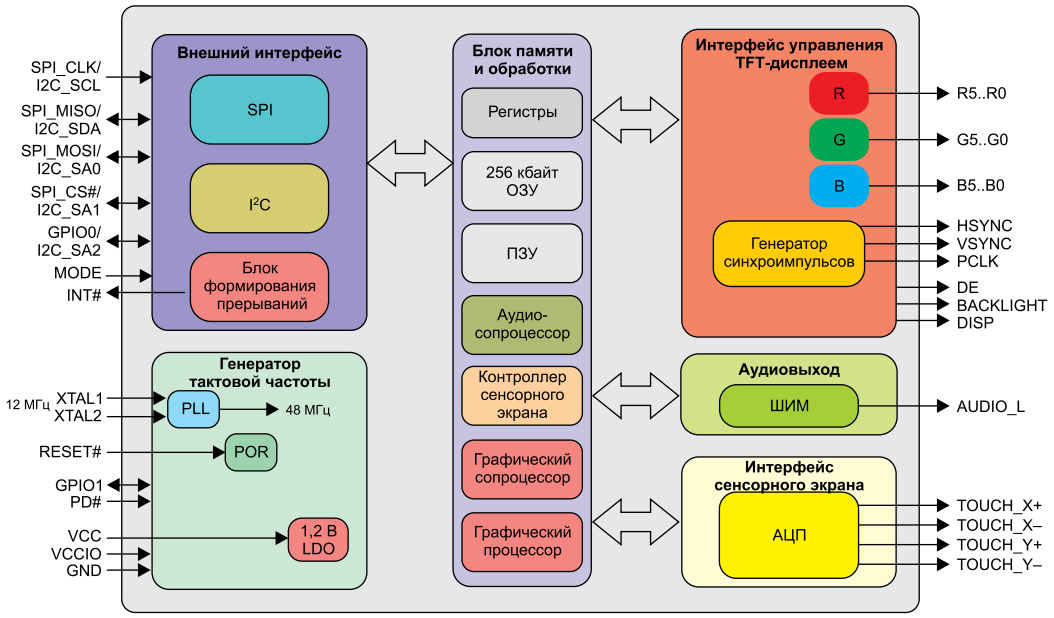


Рис. 2. Функциональная схема графического контроллера FT800

Функциональные возможности FT800

Что же представляет собой новая микросхема FTDI и в чем заключаются ее особенности?

Начнем с того, что контроллер EVE объединяет в себе возможности типовых графических контроллеров, таких как Solomon или Himax. Сюда входят формирование сигналов управления дисплеем и работа с базовыми графическими функциями (точки, линии, растровые картинки и т. п.). В дополнение к этим возможностям FT800 имеет встроенные функции для отображения сложных графических объектов (кнопки, шкалы, анимированные заставки и т. п.). При работе со стандартными графическими контроллерами такие функции реализуются программными средствами, а следовательно — на управляющем микроконтроллере. При реализации можно использовать готовые растровые изображения или базовые графические функции, но в любом случае это повышает требования к управляющему микроконтроллеру. Нужна или большая память для хранения готовых элементов в виде картинок, или высокая производительность для формирования изображения с помощью точек, линий и т. п. При использовании FT800 эти требования значительно снижаются за счет аппаратной поддержки: рисование кнопки на экране будет заключаться в передаче в контроллер только параметров изображения (размера, координат, стиля).

Кроме графических функций, контроллер FT800 также поддерживает работу с резистивным сенсорным экраном и имеет встроенный аудиоконтроллер. Таким образом, нет необходимости использовать внешние элементы и/или ресурсы управляющего контроллера для работы с сенсорным экраном и звуком.

Управление микросхемой FT800 осуществляется по простому последовательному интерфейсу IIC или SPI (рис. 1). Это также выгодно отличает FT800 от стандартного контроллера SSD1963 с 8/16-разрядной шиной данных и набором сигналов управления. Меньше корпус микросхемы хост-контроллера — меньше и его стоимость.

На рис. 2 представлена функциональная схема контроллера FTDI FT800. Основным узлом микросхемы является блок Memory and Processing. В этот блок входят: память (регистры, ОЗУ и ПЗУ), графический процессор и три сопроцессора (графический, аудио и сенсорный).

Управление контроллером FT800 и получение информации о выполнении той или иной команды, состоянии сенсорного экрана и других внутренних функциях микросхемы происходит посредством записи/чтения определенных областей памяти (рис. 3). Области памяти Coprocessor RAM и Display list RAM предназначены для записи графических команд, на основании которых графический процессор или сопроцессор формируют изображение. Графический процессор предназначен для отображения графических примитивов (точек, линий, прямоугольников, растровых картинок, текста и т. п.). Сопроцессор отвечает за сложные графические объекты (widget) — кнопки, шкалы, слайдеры и т. п. Кроме того, сопроцессор поддерживает простую анимацию, которая

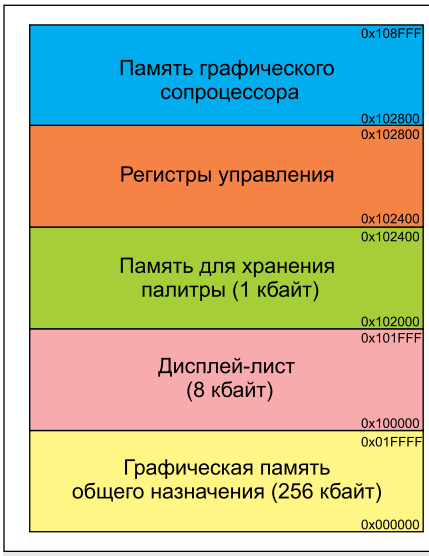


Рис. 3. Области ОЗУ

может быть использована для перемещения загруженной картинки по экрану, формирования анимированного изображения, иллюстрирующего процесс ожидания (spinner) или рисования (sketch). Эти функции после загрузки в память контроллера FT800 выполняются без участия управляющего микроконтроллера. Их выполнение длится до тех пор, пока не будет получена команда на завершение от управляющего процессора.

Аудиопроцессор предназначен для проигрывания встроенного набора звуковых эффектов, а также может воспроизводить звуковой поток в форматах PCM, ADPCM и u-Law с частотой дискретизации от 8 до 48 кГц. Аудиопроцессор может проигрывать также загружаемые извне

звуковые файлы указанных форматов и воспроизводить различные звуковые эффекты, содержащиеся в его ПЗУ [2].

Микросхема FT800 поддерживает работу с дисплеями с максимальным разрешением 512×512 точек. Если говорить о стандартных дисплеях, то это дисплеи формата WQVGA (480×272 точки) или QVGA (320×240 точек).

Микросхема FT800 выпускается в корпусе типа VQFN-48 и рассчитана на работу в температурном диапазоне от –40 до +85 °C. Напряжение питания — 3,3 В, потребление в активном режиме — 24 мА, в спящем режиме контроллер потребляет ток в пределах от 10 до 25 мкА. Все линии ввода/вывода микросхемы могут работать в диапазоне от 1,8 до 3,3 В и не совместимы с 5-В уровнями. В последнем случае для сопряжения необходимо использовать конвертер уровней, например буферную микросхему семейства 74LCx125 [3].

Средства отладки

Для тестирования возможностей новых микросхем компания FTDI предлагает несколько вариантов отладочных модулей.

Модули серии VM800B (рис. 4) выполнены в виде готового дисплейного блока, включающего плату с контроллером FT800, TFT-дисплей и декоративную рамку. На плате контроллера дисплея установлена сама микросхема FT800 и все необходимые для ее работы компоненты. Все линии ввода/вывода, которые могут потребоваться для подключения управляющего процессора, выведены на разъем через конвертер уровней 3,3–5 В. Сам графический модуль может питаться от источника с напряжением 5 или 3,3 В. Для вывода звука на плате предусмотрен аудиоусилитель и динамик.

Модули VM800B поставляются с дисплеями 3,5″, 4,3″ и 5″, которые имеют резистивный сенсорный экран.

Отладочная плата VM800 Cxx-N по функциональным возможностям не отличается от платы модуля VM800B, за исключением топологии и форм-фактора (рис. 5). Плата поставляется в двух модификациях: с разъемом FPC/FFC для подключения TFT-дисплея с количеством выводов 54 или 40. На базе этой платы выпускаются комплекты VM800 Cxx-D (рис. 6), в которые входит плата контроллера и дисплей (3,5″, 4,3″ и 5″).

Кроме отладочных средств компании FTDI, можно упомянуть два модуля компании MikroElektronika: ConnectEVE (рис. 7а) и EVE Click (рис. 7б).

ConnectEVE представляет собой графический модуль с платой контроллера на FT800 и дисплеем 4,3″. Модуль EVE Click выполнен в виде мезонинной платы с разъемом FPC/FFC для подключения TFT-дисплея с 40-выводным шлейфом. В отличие от модулей FTDI, ConnectEVE и EVE Click рассчитаны на питание от 3,3 В и не имеют кон-

Рис. 4. Отладочный модуль VM800Bxx-xx

Рис. 5. Отладочная плата VM800 Cxx-N

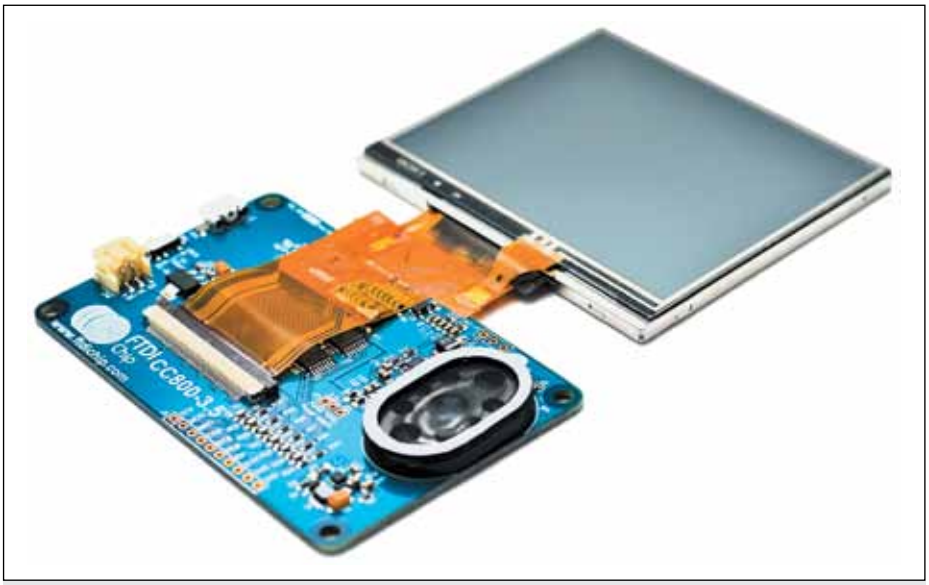
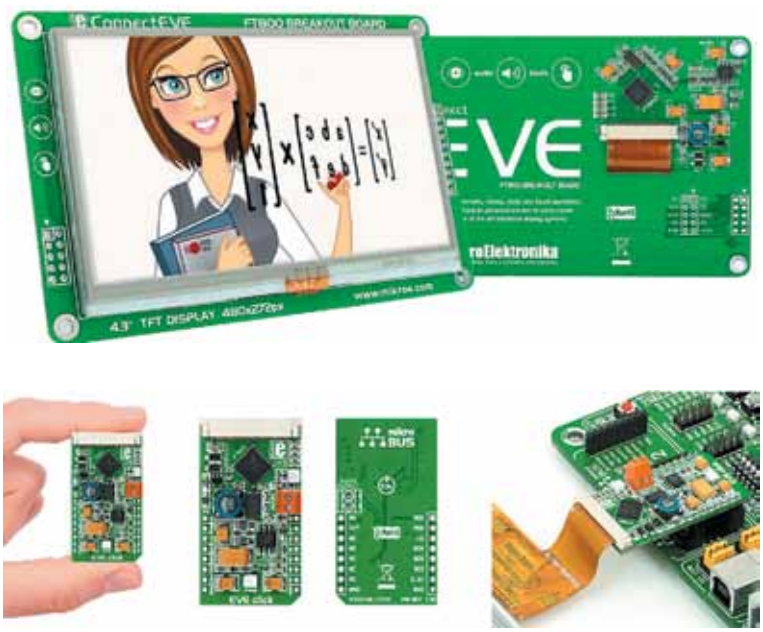


Рис. 6. Отладочный комплект VM800 Cxx-D



а

б

Рис. 7. Модули компании MikroElektronika: а) ConnectEVE; б) EVE Click

а

	7	6	5	4	3	2	1	0
1 байт	0	1	Код команды [5:0]					
2 байт	0	0	0	0	0	0	0	0
3 байт	0	0	0	0	0	0	0	0

б

	7	6	5	4	3	2	1	0
1 байт	1	0	Адрес [21:16]					
2 байт	Адрес [15:8]							
3 байт	Адрес [7:0]							
4 байт	1 байт данных							
...	...							
n байт	n байт данных							

в

	7	6	5	4	3	2	1	0
1 байт	0	0	Адрес [21:16]					
2 байт	Адрес [15:8]							
3 байт	Адрес [7:0]							
4 байт	«Пустой» (0x00) байт							
1 байт	1 байт данных от FT800							
...	...							
n байт	n байт данных от FT800							

Рис. 8. Формат команды:
а) управляющей; б) записи; в) чтения

вертера уровней по линиям связи с управляющим процессором. На обоих модулях реализован аудиоусилитель, но отсутствует встроенный динамик.

По нашему мнению, EVE Click за счет своего форм-фактора могут представлять некоторый интерес. В остальном модули, предлагаемые FTDI, обладают большей универсальностью, а также имеют более низкую цену.

Для освоения программирования контроллера FT800 компания FTDI предлагает готовые примеры, демонстрирующие все основные функции микросхемы и методы управления ими. Знакомство с графическим контроллером рекомендуем начинать с двух базовых примеров. Первый пример программы реализован для платформы Arduino [7], второй — на базе Visual Studio C [6]. Для работы с примером для ПК дополнительно понадобится конвертер USB-SPI на базе микросхемы FT232H. Им могут служить, например, готовые кабели C232HM-EDHSL-0 (5 В) или C232HM-DDHSL-0 (3,3 В). Для работы с модулями MikroElektronika или платами собственной разработки без конвертера уровней подойдет только второй вариант. С модулями FTDI можно использовать обе версии конвертера USB-SPI.

Общие принципы и примеры работы с FT800

Управление контроллером FT800 осуществляется с помощью интерфейса SPI или IIC. В общем виде управляющий процессор взаимодействует с FT800 путем записи и чтения определенных регистров и разделов памяти контроллера EVE. Для взаимодействия хоста с FT800 предусмотрено три типа команд:

- Host command — управляющая команда (рис. 8а);
- Host Memory Write — команда записи (рис. 8б);
- Host Memory Read — команда чтения (рис. 8в).

Управляющие команды предназначены для выбора как режима работы FT800 (активное состояние, режим готовности, спящий режим и выключение), так и источника тактовой частоты, а также установки тактовой частоты и сброса контроллера FT800. Управляющие команды всегда состоят из трех байтов, первый из которых несет информацию о типе команды, а два последующих байта — нулевые (рис. 8а).

Следует обратить внимание на то, что в документации и примерах значения битов 7 и 6 включены в код команды. Например, команда выбора частоты равна GPU_PLL_48M = = 0x62 (b**01**100010). Поэтому никаких дополнительных битовых операций с кодом команды проводить не требуется. Это правило применимо только к управляющим командам.

С помощью команды записи хост-процессор передает в FT800 наборы команд на выполнение тех или иных графических операций, воспроизведение звука и т. п. Команда записи всегда состоит из 22-битного адреса и набора данных (рис. 8б).

Команда чтения позволяет хост-процессору получать информацию от FT800 (состояние регистров, результаты обработки сигналов сенсорного экрана и т. п.). Команда чтения состоит из 22-битного адреса, за которым следует пустой байт. Получив этот последний байт, FT800 начинает передачу данных (рис. 8в).

Алгоритм включения и работы контроллера FT800 приведен на рис. 9.

Первым этапом начала работы будет конфигурация интерфейса SPI управляющего процессора. Интерфейс SPI хост-процессора всегда является «мастером» и должен быть установлен в режим mode 0 с передачей старшего бита первым (MSB first). Рабочая частота последовательного интерфейса хост-процессора может составлять до 30 МГц. До окончания этапа инициализации контроллера EVE рабочая частота SPI должна быть не выше 11 МГц.

После подачи питания контроллер FT800 требуется перевести в активный режим. Это делается переводом сигнала PD в низкий уровень на 20 мс с последующим переводом его в высокий уровень. Через 20 мс после установки высокого уровня на линии PD контроллер FT800 перейдет в режим готовности (Standby). Переход в рабочий режим из режима готовности осуществляется записью команды ACTIVE (0x00) по адресу 0x00.

После перехода в активный режим рекомендуется сбросить значения всех регистров и логических цепей. Сброс осуществляется командой CORERST (0x68).

Завершающим шагом будет переключение графического контроллера на работу от внешнего источника тактовой частоты. Микросхема может тактироваться от кварцевого резонатора или генератора с частотой 12 МГц. Для перехода на внешний источник тактирования необходимо подать управляющую команду CLKEXT (0x44). После выбора источника тактирования нужно выбрать рабочую частоту графического контроллера (48 или 36 МГц) подачей соответствующих команд: CLK48M (0x62) или CLK36M (0x61).

Проверка корректного завершения всех начальных процедур выполняется чтением идентификатора микросхемы.

Этап инициализации FT800 с помощью микроконтроллера FTDI VNC2 может быть реализован следующим образом:

```
// формирование импульса длительностью 20 мс на линии PD
state = 0xFF;
   vos_dev_write(hGPIO_PORT_A, &state, 1, NULL);
   vos_delay_msecs(20);

   state = 0x00;
   vos_dev_write(hGPIO_PORT_A, &state, 1, NULL);
   vos_delay_msecs(20);

   state = 0xFF;
   vos_dev_write(hGPIO_PORT_A, &state, 1, NULL);
   vos_delay_msecs(20);
   // в активном режиме на линии PD всегда высокий уровень

   // перевод FT800 в активное состояние из режима Standby

   Transfer_Array[0] = FT_GPU_ACTIVE_M; // FT_GPU_ACTIVE_M
= 0x00
   Transfer_Array[1] = 0; // первый нулевой байт управляющей
команды
   Transfer_Array[2] = 0; // второй нулевой байт управляющей
команды
   vos_dev_write(hSPIm, Transfer_Array, 3, NULL); // функция
передачи по SPI, где: hSPIm — обозначение блока SPI, к кото-
рому идет обращение; Transfer_Array — команда + два нулевых
байта в соответствии с рис. 9; 3 — количество передаваемых байт;
NULL — данный параметр не используется

   Transfer_Array[0] = FT_GPU_EXTERNAL_OSC;
   Transfer_Array[1] = 0;
   Transfer_Array[2] = 0;
   vos_dev_write(hSPIm, Transfer_Array, 3, NULL);
   vos_delay_msecs(10);

   //Выбор рабочей частоты

Transfer_Array[0] = FT_GPU_PLL_48M; // FT_GPU_PLL_48M = 0x62
   Transfer_Array[1] = 0;
   Transfer_Array[2] = 0;
   vos_dev_write(hSPIm, Transfer_Array, 3, NULL);
   vos_delay_msecs(10); // ожидание для установки PLL

// Логический сброс

   Transfer_Array[0] = FT_GPU_CORE_RESET; // FT_GPU_CORE_
RESET = 0x68
   Transfer_Array[1] = 0;
   Transfer_Array[2] = 0;
   vos_dev_write(hSPIm, Transfer_Array, 3, NULL);

   // Инициализация контроллера FT800 завершена

   // чтение идентификатора REG ID для проверки корректного
завершения этапа инициализации
   while(1)
   {
      Transfer_Array[0] = REG_ID>>16;
      Transfer_Array[1] = REG_ID>>8;
      Transfer_Array[2] = REG_ID;
      Transfer_Array[3] = 0;

      // REG_ID = 1057792, в итоге: Transfer_Array[0] = 0x10,
Transfer_Array[1] = 0x24, Transfer_Array[1] = 0x00, Transfer_
Array[4] = 0x00;

      vos_dev_write(hSPIm, Transfer_Array, 4, NULL); // передача
сформированной команды в FT800

      // ожидание ответа и выход из цикла чтения идентификатора,
если идентификатор получен
      spim_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_
QUEUE_STATUS;
      vos_dev_ioctl(hSPIm, &spim_iocb);
      dataAvail = spim_iocb.get.queue_stat;
         if (dataAvail > 0)
         {
         vos_dev_read(hSPIm, in_buf2, dataAvail, NULL);
            if (in_buf2[0] == 0x7C)
            {
             break;
            }
         }
   }
```

Конечно, функции работы с SPI для каждого микроконтроллера могут различаться, но сам принцип формирования команд остается одинаковым. Данный код, на наш взгляд, наглядно иллюстрирует принципы

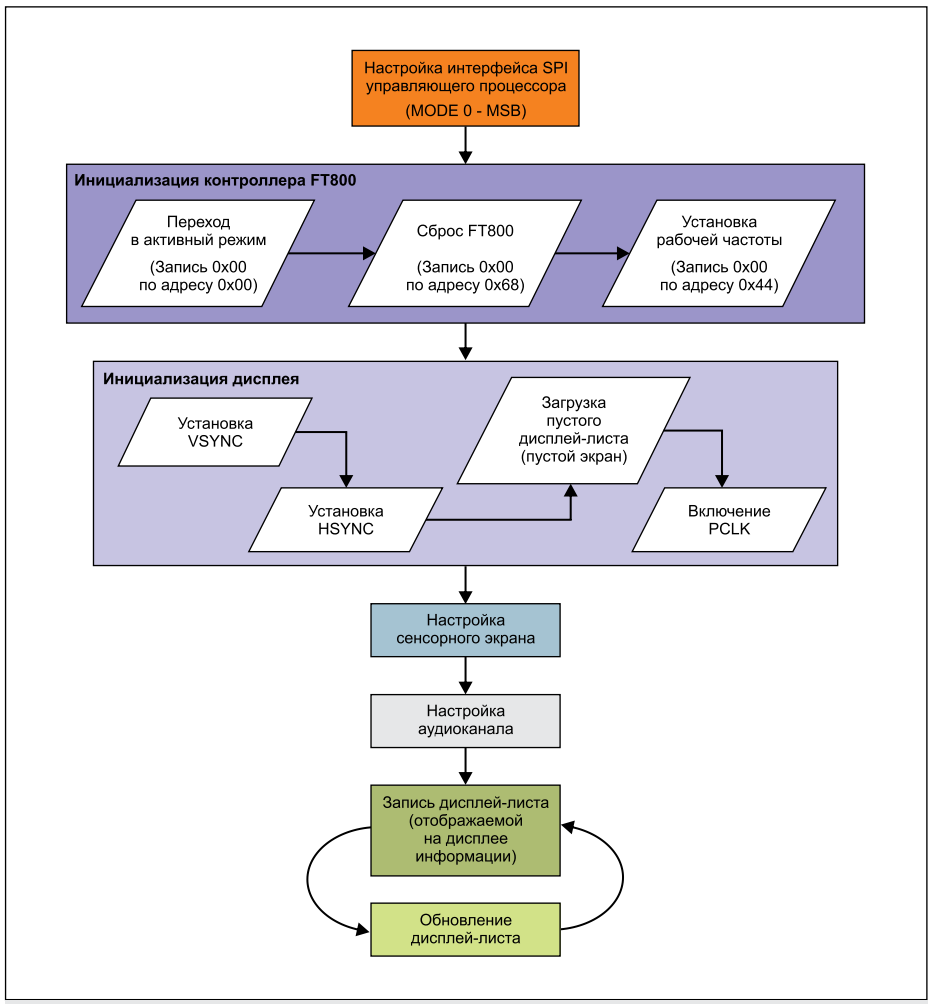


Рис. 9. Алгоритм работы контроллера EVE FT800

формирования команд для работы с FT800. Все дальнейшие примеры в этой статье будут даны в аналогичном виде.

По окончании этапа инициализации графического контроллера EVE следует этап настройки параметров для работы TFT-дисплея. На этом этапе в контроллер записываются параметры, необходимые для формирования сигналов управления TFT-дисплеем. Расчет их параметров и подробное описание приведены в [4, 5]. Ниже приведен пример формирования одной из команд конфигурации дисплея:

```
// Запись в контроллер FT800 значения ширины дисплея в пикселях
Transfer_Array[0] = (0x80 | (REG_HSIZE >> 16)); // Формируем три
байта, состоящие из признака команды записи и адреса регистра
REG_HSIZE = 0x102430
Transfer_Array[1] = REG_HSIZE >> 8;
Transfer_Array[2] = REG_HSIZE;
Transfer_Array[3] = FT_DispWidth & 0xFF; // Формируем два байта
со значением ширины дисплея
Transfer_Array[4] = ((FT_DispWidth >> 8) & 0xFF);
vos_dev_write(hSPIm, Transfer_Array, 5, NULL); // Передаем в FT800
итоговые 5 байт
```

После установки параметров дисплея загружаем в контроллер список команд для очистки дисплея и вывода пустого экрана. Список команд, в результате исполнения которых FT800 выведет некое изображение, называется дисплей-листом. Количество инструкций в дисплей-листе не должно превышать 2048. Память дисплей-листа (RAM_DL) занимает область памяти контроллера, начиная с адреса 0x100000 и заканчивая адресом 0x101FFF (рис. 3). Запись команд в дисплей-лист всегда начинается со значения 0x100000, каждая команда занимает не менее четырех байтов. Дисплей-лист всегда заканчивается командой DISPLAY ().

Отображение нового дисплей-листа начинается по окончании вывода предыдущего кадра (записываем в регистр REG_DLSWAP значение SWAP FRAME = 0x2, это рекомендуемый параметр) или по окончании вывода текущей строки (записываем в регистр REG_DLSWAP значение SWAP FRAME = 0x1). Второй функцией регистра REG_DLSWAP является отображение статуса буфера дисплей-листа. Буфер готов для записи новых команд, если его значение равно 0x00. Во всех остальных случаях запись запрещена, так как если записывать в этот момент новые команды в дисплей-лист, текущее изображение может быть испорчено, а новое не будет отображаться вовсе. Примеры формирования изображения или дисплей-листа будут рассмотрены далее.

Команда, которой мы заканчиваем конфигурацию дисплея, будет выглядеть так:

```
Transfer_Array[0] = (0x80 | (REG_PCLK >> 16));
    Transfer_Array[1] = REG_PCLK >> 8;
    Transfer_Array[2] = REG_PCLK;
    Transfer_Array[3] = FT_DispPCLK;
    vos_dev_write(hSPIm, Transfer_Array, 4, NULL);
```

Этой командой задается коэффициент деления внутренней частоты FT800 для формирования тактовой частоты дисплея и ее включения. Если значение FT_DispPCLK равно 0, тактирование дисплея выключено. После подачи этой команды на экран будет выведено изображение, заданное нами в предыдущем шаге в дисплей-листе, то есть чистый экран с заданным фоном.

Процесс конфигурации FT800 заканчивается настройкой параметров аудиопроцессора и контроллера сенсорного экрана. По окончании этих этапов контроллер полностью готов к работе.

Как уже говорилось выше, графическая система контроллера FT800 содержит два процессора. Основной процессор предназначен для работы со стандартными примитивами и растровыми картинками, а сопроцессор отвечает за работу с предустановленным набором графических объектов — виджетов (widget) (рис. 10).

Основной графический процессор формирует изображение на основе команд (дисплей-листа), загруженных в память RAM DL. Например, требуется вывести в центр дисплея изображение точки размером в пять пикселей. Для этого в дисплей-лист необходимо будет загрузить следующий набор команд:

Рис. 10. Готовые графические объекты

- Предустановка цвета (формат RGB — в данном случае фон будет темно-зеленый) для команды CLEAR — dl(CLEAR_COLOR_RGB(0, 200, 0)).
- Установка цвета фона экрана. Цвет определяется значениями, заданными командой CLEAR_COLOR_RGB — dl(CLEAR(1, 1, 1)).
- Установка цвета объекта — dl(COLOR_RGB(255, 255, 255)).
- Параметры объекта. Для точки — размер в пикселях (в данном случае размер будет равен пяти пикселям) — dl(POINT_SIZE(5*16)).
- Команда BEGIN определяет тип выводимого объекта, в данном случае точки (POINTS) — dl(BEGIN(POINTS)).
- Координаты экрана, где будет изображен объект, — dl(VERTEX2F(480/2*16,272/2*16)).
- Служебная команда, которая определяет конец текущего дисплей-листа, — dl(DISPLAY()).

Для того чтобы изображение, заданное этим набором команд, было отображено на экране, необходимо в регистр REG_DLSWAP записать значение SWAP FRAME = 0x2. Итогом выполнения описанных выше операций будет изображение белой точки размером пять пикселей в центре экрана на зеленом фоне.

Для микроконтроллера Vinculum II код для реализации этого дисплей-листа будет формироваться следующим образом:

```
// формирование и запись команды CLEAR_COLOR_RGB
    temp = CLEAR_COLOR_RGB(0,200,0);

//формируем начальный адрес дисплей-листа RAM_DL = 0x10000
    TRANSFER_ARRAY[0] = (0x80 | (RAM_DL >> 16));
    TRANSFER_ARRAY[1] = RAM_DL >> 8;
    TRANSFER_ARRAY[2] = RAM_DL;

// формируем команду CLEAR_COLOR_RGB(red,green,blue)
((2<<24)|(((red)&255)<<16)|(((green)&255)<<8)|(((blue)&255)<<0))
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
// записываем массив в память FT800
vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

temp = CLEAR(1,1,1); //CLEAR(c,s,t) ((38<<24)|(((c)&1)<<2)|(((s)
&1)<<1)|(((t)&1)<<0))
//формируем начальный следующий адрес дисплей-листа RAM_DL +4
    TRANSFER_ARRAY[0] = (0x80 | (RAM_DL+4 >> 16));
    TRANSFER_ARRAY[1] = RAM_DL+4 >> 8;
    TRANSFER_ARRAY[2] = RAM_DL+4;
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

temp = COLOR_RGB(255, 255, 255); // COLOR_RGB(red,green,blue)
((4<<24)|(((red)&255)<<16)|(((green)&255)<<8)|(((blue)&255)<<0))
    TRANSFER_ARRAY[0] = (0x80 | (RAM_DL+8 >> 16));
    TRANSFER_ARRAY[1] = RAM_DL+8 >> 8;
    TRANSFER_ARRAY[2] = RAM_DL+8;
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

temp = POINT_SIZE(5 * 16); // POINT_SIZE(size)
((13<<24)|(((size)&8191)<<0))
    TRANSFER_ARRAY[0] = (0x80 | (RAM_DL+12 >> 16));
    TRANSFER_ARRAY[1] = RAM_DL+12 >> 8;
    TRANSFER_ARRAY[2] = RAM_DL+12;
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

temp = BEGIN(FTPOINTS); // BEGIN(prim) ((31<<24)|(((prim)&15)<<0))
    TRANSFER_ARRAY[0] = (0x80 | (RAM_DL+16 >> 16));
    TRANSFER_ARRAY[1] = RAM_DL+16 >> 8;
    TRANSFER_ARRAY[2] = RAM_DL+16;
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

temp = VERTEX2F((FT_DispWidth/2) * 16, (FT_DispHeight/2) * 16);
//VERTEX2F(x,y) ((1<<30)|(((x)&32767)<<15)|(((y)&32767)<<0))
    TRANSFER_ARRAY[0] = (0x80 | (RAM_DL+20 >> 16));
    TRANSFER_ARRAY[1] = RAM_DL+20 >> 8;
    TRANSFER_ARRAY[2] = RAM_DL+20;
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

// запись в дисплей-лист команды DISPLAY, определяющей конец
текущего дисплей-листа
    temp = DISPLAY(); // DISPLAY() ((0<<24))
    TRANSFER_ARRAY[0] = (0x80 | ((RAM_DL+24) >> 16));
    TRANSFER_ARRAY[1] = RAM_DL+24 >> 8;
    TRANSFER_ARRAY[2] = RAM_DL+24;
    TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
    TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
    TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

// Запись команды на отображение текущего дисплей-листа
    TRANSFER_ARRAY[0] = (0x80 | (REG_DLSWAP >> 16)); // адрес
регистра REG_DLSWAP = 0x102450
    TRANSFER_ARRAY[1] = REG_DLSWAP >> 8;
    TRANSFER_ARRAY[2] = REG_DLSWAP;
    TRANSFER_ARRAY[3] = DLSWAP_FRAME; //записываемое
значение DLSWAP_FRAME = 0x02
    vos_dev_write(hSPIm, TRANSFER_ARRAY, 4, NULL);
```

Другие графические примитивы, поддерживаемые контроллером FT800, программируются аналогично. Отличаться они будут только настройкой параметров. В описанном выше примере таким параметром был размер точки POINT_SIZE. Со всеми графическими примитивами и их параметрами можно ознакомиться в [4].

Помимо стандартных функций, доступных и в контроллерах других производителей, графический контроллер FT800 может работать с более сложными объектами, часть из которых изображена на рис. 10. За формирование и отображение таких объектов отвечает графический сопроцессор FT800. Команды для сопроцессора записываются в специальную область памяти RAM_CMD (0x108000) размером четыре кбайта, организованную в виде кольцевого буфера (рис. 11). Два регистра, REG_CMD_READ и REG_CMD_WRITE, показывают размещение текущего набора ко-

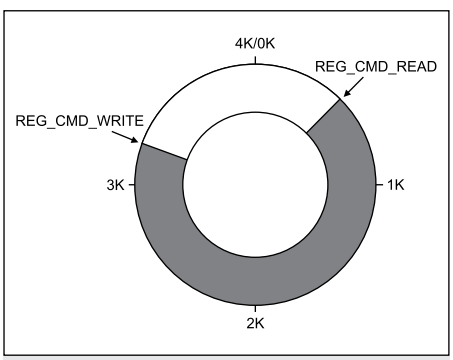


Рис. 11. Кольцевой буфер сопроцессора

манд в памяти. Если значения регистров REG_CMD_WRITE и REG_CMD_READ равны, сопроцессор находится в режиме ожидания. Как только управляющий процессор закончил передачу набора команд на выполнение, он должен установить регистр REG_CMD_WRITE в соответствии с переданным количеством команд. Сопроцессор, обнаружив изменение этого регистра, начинает выполнение загруженных в кольцевой буфер команд.

Кроме специализированных функций, в набор команд для графического сопроцессора могут быть включены стандартные графические функции, описанные выше. Стандартные команды всегда начинаются с записи служебной команды CMD_DLSTART. По этой команде сопроцессор передает следующий за ней набор графических команд в дисплей-лист основного процессора. Формирование изображения и вывод его на экран начинается при обнаружении сопроцессором команды CMD_SWAP.

Приведем примеры работы с графическим сопроцессором. Первый листинг показывает последовательность служебных и графических команд, результатом выполнения которых будет вывод на экран анимированной заставки с логотипом FTDI:

```
temp = 0xffffff31; // код команды CMD_LOGO для вывода анимированного логотипа

// установка адреса памяти команд RAM_CMD (0x108000) сопроцессора
   TRANSFER_ARRAY[0] = (0x80 | (RAM_CMD >> 16));
   TRANSFER_ARRAY[1] = RAM_CMD >> 8;
   TRANSFER_ARRAY[2] = RAM_CMD;
   // преобразование кода команды для передачи по SPI
   TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
// запись команды в память FT800
   vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

   temp = 0x0004; // количество байт в данной команде — 4 байта
TRANSFER_ARRAY [3..6] в предыдущей команде
   // установка адреса регистра REG_CMD_WRITE
   TRANSFER_ARRAY[0] = (0x80 | (REG_CMD_WRITE >> 16));
   TRANSFER_ARRAY[1] = REG_CMD_WRITE >> 8;
   TRANSFER_ARRAY[2] = REG_CMD_WRITE;
   // преобразование количества байт temp для передачи по SPI
   TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
   // записываем в регистр REG_CMD_WRITE значение temp
   vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);

   vos_delay_msecs(5000); // задержка на вывод анимированной картинки
```

Этот пример демонстрирует последовательность передачи команд для работы с сопроцессором. Записываем команды в кольцевой буфер сопроцессора (RAM_DL) и указываем размер записанного набора через регистр REG_CMD_WRITE.

Покажем работу еще одной команды сопроцессора — CMD_SPINNER. Это анимированный элемент, который может быть использован для отображения, например, режима ожидания.

Команда имеет следующий формат:

```
void cmd_spinner( int16_t x, int16_t y, uint16_t style, uint16_t scale )
```

здесь x и y — координаты верхнего левого угла объекта; style — тип объекта; scale — размер. В примере будем использовать style = 0, в данном случае на экран будет выводиться анимированное движение точки по окружности.

Код будет выглядеть следующим образом:

```
// установка адреса памяти команд RAM_CMD (0x108000) сопроцессора
TRANSFER_ARRAY[0] = (0x80 | (RAM_CMD >> 16));
   TRANSFER_ARRAY[1] = RAM_CMD >> 8;
   TRANSFER_ARRAY[2] = RAM_CMD;

// начало команд для дисплей-листа основного процессора
   temp = CMD_DLSTART;
   TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);

   //установка цвета по умолчанию
   temp = CLEAR_COLOR_RGB(2,2,2);
   //Ft_Gpu_Hal_Transfer32(host,v);
   TRANSFER_ARRAY[7] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[8] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[9] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[10] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);

   // установка цвета фона в значения, заданные выше
   temp = CLEAR(1,1,1);
   //Ft_Gpu_Hal_Transfer32(host,v);
   TRANSFER_ARRAY[11] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[12] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[13] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[14] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);

   // цвет объекта
   temp = COLOR_RGB(0, 255, 0);
   //Ft_Gpu_Hal_Transfer32(host,v);
   TRANSFER_ARRAY[15] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[16] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[17] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[18] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);

   temp = 0xffffff16; //CMD_SPINNER
   // преобразование кода команды CMD_SPINNER
TRANSFER_ARRAY[19] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[20] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[21] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[22] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);

// координаты x=240 и y=130
   TRANSFER_ARRAY[23] = 240; // x-мл. байт
   TRANSFER_ARRAY[24] = 0; // x-ст. байт
   TRANSFER_ARRAY[25] = 130; // y-мл. байт
   TRANSFER_ARRAY[26] = 0; // y-ст. байт
   // значения style=0 и size = 0
   TRANSFER_ARRAY[27] = 0; // style-мл. байт
   TRANSFER_ARRAY[28] = 0; // style-ст. байт
   TRANSFER_ARRAY[29] = 0; // size-мл. байт
   TRANSFER_ARRAY[30] = 0; // size-ст. байт

// запись в дисплей-лист команды DISPLAY, определяющей конец текущего дисплей-листа

   temp = DISPLAY();
   TRANSFER_ARRAY[31] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[32] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[33] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[34] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);

// Запись команды на отображение текущего дисплей-листа
   temp = CMD_SWAP;
   TRANSFER_ARRAY[35] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[36] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[37] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[38] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
   // передача набора команд
   vos_dev_write(hSPIm, TRANSFER_ARRAY, 39, NULL);

   temp = 36; // количество байт в данной команде
   // обновление регистра REG_CMD_WRITE
   TRANSFER_ARRAY[1] = REG_CMD_WRITE >> 8;
   TRANSFER_ARRAY[2] = REG_CMD_WRITE;

   TRANSFER_ARRAY[3] = ((temp & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[4] = (((temp & 0xFFFF)>> 8)& 0xFF);
   TRANSFER_ARRAY[5] = (((temp >> 16) & 0xFFFF)& 0xFF);
   TRANSFER_ARRAY[6] = ((((temp >> 16) & 0xFFFF)>> 8)& 0xFF);
   vos_dev_write(hSPIm, TRANSFER_ARRAY, 7, NULL);
```

Результатом выполнения этих инструкций будет отображение движения по окружности зеленой точки по черному фону.

Приведенные примеры показывают основные принципы работы с графическим контроллером FT800. Для дальнейшего освоения возможностей микросхемы можно воспользоваться примерами производителя [6, 7].

Заключение

Завершая первое знакомство с новой микросхемой компании FTDI, можно отметить, что FT800 обладает рядом преимуществ, выгодно отличающих ее от конкурентов. По соотношению стоимости, функциональных возможностей и доступности FT800 имеет все шансы, чтобы ее популярность у разработчиков была не меньше, чем у USB-мостов этого же производителя.

Литература

1. Долгушин С. Графические контроллеры и дисплейные модули компании 4D Systems // Компоненты и технологии. 2013. № 2.
2. Application Note AN 252 FT800 Audio Primer.
3. FT800 Embedded Video Engine Datasheet Version 1.0.
4. FT800 Programmer Guide.
5. Application Note AN 240 FT800 From the Ground Up.
6. Application Note AN 245 FT800 Sample Application Introduction For VM800B and VM800C Development Kits and Windows PC.
7. Application Note AN 246 FT800 Sample Application Introduction For VM800B or VM800C Development Kits and Arduino Pro.