Графический контроллер EVE FT800 FTDI.

Работа с пользовательскими шрифтами, кнопками и сенсорным экраном

Сергей ДОЛГУШИН
dsa@efo.ru

Графический контроллер FT800 позиционируется производителем как идеальное решение для создания пользовательских графических интерфейсов. Его специализированные графические функции для создания кнопок, слайдеров, библиотека звуковых эффектов и контроллер сенсорного экрана ориентированы на данную область применения.
Пользовательский графический интерфейс как минимум должен включать в себя элементы управления и информационные сообщения. Для вывода любой текстовой информации или надписей на элементах управления на русском языке разработчику потребуются загружаемые шрифты. Мы рассмотрим работу с такими шрифтами, а также базовые принципы работы с кнопками и сенсорным экраном.

Статья продолжает тему, начатую с описания в [1, 2] общих функциональных возможностей новой микросхемы FT800 и адаптации примеров производителя для Microsoft Visual Studio под компилятор Image Craft для МК PSoC Cypress. А теперь мы покажем, как использовать пользовательские шрифты с графическим контроллером FT800.

Он имеет 16 встроенных шрифтов, содержащих стандартный набор ASCII символов. На экран могут выводиться символы в диапазоне кодов от 32 до 127. Встроенные шрифты отличаются друг от друга только размером.

Выбор шрифта, который будет использован API-функцией для вывода текста, осуществляется указанием специального индекса шрифта. Все шрифты, встроенные и пользовательские, индексируются в диапазоне от 0 до 31. Встроенные шрифты имеют индексы от 16 до 31 включительно. Если мы хотим использовать в приложении шрифты другого размера, другого начертания или отличные от английских, то мы можем применить собственные. Пользовательские шрифты перед их использованием с API-функциями должны быть загружены в графическое ОЗУ FT800 (RAM_G), объем которого составляет 256 кбайт. Загружаемые шрифты могут иметь индексы от 0 до 14.

Шрифты, встроенные и загружаемые, хранятся в памяти контроллера в виде растровых изображений. Пользовательские шрифты могут загружаться в трех растровых форматах: L1, L4 и L8. Структура форматов приведена в таблицах 1–3.

Таблица 1. Структура формата L1

Пиксель 0	Бит 7	Байт 0
Пиксель 1	Бит 6	
...	...	
Пиксель 7	Бит 0	

Таблица 2. Структура формата L4

Пиксель 0	Бит 7...4	Байт 0
Пиксель 1	Бит 3...0	

Таблица 3. Структура формата L8

Пиксель 0	Бит 7...0	Байт 0
Пиксель 1	Бит 15...8	Байт 1
Пиксель 7	Бит 23...16	Байт 2

В зависимости от наличия свободной памяти в управляющем МК разработчик может выбрать, какой из представленных форматов использовать в своем приложении. Для сравнения на рис. 1 приведен снимок экрана дисплея модуля VM800C43A-D, на который выводится текстовая информация в указанных форматах. Размер кода для показанных на снимке символов для форматов L1, L4 и L8 равен соответственно 540, 1890 и 3780 байт. (Приведен размер кода, содержащего изображение трех символов — «АБВ».) Для получения полного объема кода для одного загружаемого шрифта к каждому из этих размеров необходимо добавить служебную информацию (метрики шрифта). Метрика для каждого шрифта и формата всегда своя, ее размер фиксирован и равен 148 байтам.

В качестве загружаемых шрифтов можно использовать, например, True Type. Производитель предоставляет специальную утилиту Font converter [3] для их конвертации в поддерживаемые графическим контроллером форматы (L1, L4 и L8). Работа с утилитой осуществляется из командной

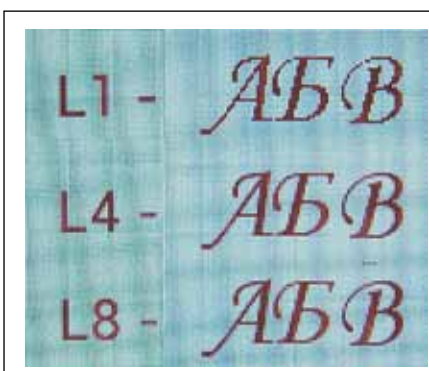

Рис. 1. Вид экрана, демонстрирующий разницу между форматами хранения растровых изображений

Рис. 2. Создание и сохранение текстового файла с требуемым набором символов

строки, в результате преобразования формируются три папки с исходными кодами шрифтов во всех трех форматах: L1, L4 и L8.

Опишем процесс конвертации шрифта. В отдельную папку, например *D:\fonts*, копируем утилиту *fnt_cnv.exe*. В эту же папку помещаем выбранный шрифт, например Monotype Corsiva, использованный в примере на рис. 1. В программе «Блокнот» (notepad) набираем требуемые нам символы — АБВ. Символы набираются без пробела, форматирование (размер и тип шрифта, установленные в «Блокноте») не влияет на результат преобразования. Созданный файл сохраняем в папке *D:\fonts* как текстовый документ (имя.*txt*) в формате UTF-8 (рис. 2).

В командной строке вводим команду следующего вида:

```
fnt_cvt.exe -i MTCORSVA.TTF -u cyr_34.txt -s 30 -d 1000
```

Здесь

-i — обязательный аргумент, за которым указывается исходный файл шрифта в формате *.*ttf*, *.*ttc*. Утилита ищет указанный шрифт в папке, где она расположена, или в системной папке Windows. (В примере используется шрифт Monotype Corsiva (*MTCORSVA.TTF*).)

-u — обязательный аргумент, за которым указывается имя текстового файла с требуемым набором символов в формате UTF-8. (Количество символов в файле не должно превышать 127.)

-s — аргумент, определяющий ширину символов в пикселях. Если этот аргумент не указан, ширина устанавливается равной 12 пикселям. (В примере ширина символов равна 30 пикселям.)

-d — аргумент, определяющий начальный адрес размещения шрифта в графическом ОЗУ FT800. (Если он не указан, начальный адрес равен начальному адресу области графической памяти.)

В данном примере начальный адрес для размещения нашего шрифта равен RAM_G + 1000. В результате выполнения команды утилитой будут созданы три подпапки: L1, L4 и L8. Каждая из них включает в себя три файла: *.*raw*, *.*rawh* и *.*rtf*. Текстовый файл содержит таблицу символов нашего шрифта, *.*raw* — образ шрифта, готовый к загрузке в память FT800, *.*rawh* — заголовочный файл.

Ниже приведены изменения, внесенные в пример производителя [4] и демонстрирующие использование сконвертированного выше шрифта. В файл *SampleApp_RawData.c* добавим описание нашего шрифта, воспользовавшись данными из файла *.*rawh*:

```
    FT_PROGMEM ft_prog_uchar8_t SAMApp_Metric_L1[148] =
{
/* Widths */
0,25,25,26,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,
/* Format */
1,0,0,0,
/* Stride */
4,0,0,0,
/* Max Width */
28,0,0,0,
/* Max Height */
45,0,0,0,
/* Raw Data Address in Decimal: <968> */
200,3,0,0,
/* 148 Metric Block End --- */
};

FT_PROGMEM ft_prog_uchar8_t SAMApp_L1[540]=
{
/*Bitmap Raw Data begin +++*/
/*The expected raw bitmap size is 540 Bytes */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,64,0,0,63,192,0,1,225,192,0,3,3,192,0,6,7,128,0,4,15,128,0,12,11,128,0,12,19,128,
0,12,51,128,0,12,99,128,0,0,67,128,0,0,199,0,0,1,135,0,0,1,7,0,0,2,7,0,0,6,7,0,
0,15,255,0,0,8,7,0,0,24,6,0,0,48,14,0,0,32,14,0,0,64,14,0,0,192,14,0,1,128,31,224,
3,0,0,0,6,0,0,0,12,0,0,0,248,0,0,0,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,63,255,128,1,142,1,128,
2,14,1,0,4,30,1,0,8,28,0,0,24,28,0,0,16,28,0,0,48,56,0,0,48,56,0,0,48,56,0,0,
48,63,240,0,56,120,124,0,56,112,28,0,24,112,14,0,0,112,14,0,0,240,14,0,0,224,14,0,0,224,28,0,
0,224,28,0,1,192,56,0,1,192,112,0,3,193,192,0,31,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,3,252,0,0,12,7,0,0,32,3,128,0,192,65,192,1,7,193,192,
2,3,193,192,4,3,129,192,4,3,131,128,8,3,131,128,8,7,135,0,16,7,28,0,16,7,255,0,16,7,15,128,
16,14,3,192,16,14,1,192,24,14,1,192,12,30,1,192,14,28,1,192,6,28,1,128,0,28,3,128,0,56,7,0,
0,56,14,0,0,248,56,0,3,255,224,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
/*Bitmap Raw Data end ---*/
};
```

В заголовочный файл *SampleApp.h* добавим определение этих переменных:

```
extern FT_PROGMEM ft_prog_uchar8_t SAMApp_Metric_L1[];
extern FT_PROGMEM ft_prog_uchar8_t SAMApp_L1[];
```

В файл *SampleApp.c* добавим функцию, с помощью которой будет осуществляться загрузка нашего шрифта в память FT800 и вывод символов «АБВ» в центре экрана дисплея (рис. 1):

```
ft_void_t SAMAPP_Russian()
{

  Ft_Gpu_Hal_WrMemFromFlash(phost, RAM_G + 1000, SAMApp_Metric_L1, SAMApp_Metric_L1_
SIZE); //загружаем метрику шрифта
  Ft_Gpu_Hal_WrMemFromFlash(phost, RAM_G + 1000 + SAMApp_Metric_L1_SIZE,SAMApp_
L1,SAMApp_L1_SIZE); //загружаем шрифт

  Ft_Gpu_CoCmd_Dlstart(phost); // команда начала дисплей-листа
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(0xff,0xff,0xff)); //установка цвета фона
  Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1)); //установка цвета фона
  Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,0,0)); // установка цвета текста
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(7)); //назначение нашему шрифту указателя
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(968)); //указание адреса RAM_G, где помещается
наш шрифт. Значение приведено в файле *.rawh, где оно указано в следующем виде: "/* Raw Data
Address in Decimal: <968> */ "

  Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L1,4,45)); // указание формата растровой кар-
тинки (в данном случае шрифта) — L1, параметра stride указывающего, сколько байт требуется для
линии 4, и высоты изображения в линиях. Эти параметры можно взять из файла *.rawh, где они
приведены в следующем виде: "/*('file properties ', 'format ', 'L1', ' stride ', 4, ' width ', 28, 'height', 45)*/ "
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,28,45)); // установка
размера выводимого изображения, NEAREST — без сглаживания, BORDER — без повторения, 28 —
ширина, 45 — высота.

  Ft_Gpu_CoCmd_SetFont(phost, 7, RAM_G + 1000); // регистрация шрифта в сопроцессоре FT800
для дальнейшего использования с его набором команд
  Ft_Gpu_CoCmd_Text(phost,(FT_DispWidth/2), 80, 7, OPT_CENTER, "\x01\x02\x03"); // вывод первых
трех символов на экран

  Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
  Ft_Gpu_CoCmd_Swap(phost);
  Ft_App_Flush_Co_Buffer(phost);
  Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
  Ft_Gpu_Hal_Sleep(3000);
}
```

Итак, мы рассмотрели процесс создания и добавления нового шрифта и вывод его на экран дисплея в виде текста. Возможности по применению пользовательских шрифтов не ограничиваются выводом текста на экран. Их, как и встроенные шрифты, можно использовать совместно с кнопками.

Кнопки могут выводиться на экран отдельными элементами командой ***Cmd_Button***:

```
Ft_Gpu_CoCmd_Button(phost, x,y, w, h, z,s, "Надпись");
```

Здесь
x и y — координаты верхнего левого угла кнопки;
w и h — ширина и высота в пикселях;
z — указатель на шрифт (установленный командой BITMAP_HANDLE()), который будет использован в надписи на кнопке;
s — вид кнопки: OPT_FLAT — плоское изображение или OPT_3D — объемное;
«Надпись» — выводимая надпись.

Чтобы вывести изображение кнопки с надписью шрифтом из ранее приведенного примера, создадим команду, которая будет выглядеть следующим образом:

```
Ft_Gpu_CoCmd_Button(phost, x,y, w, h, 7,s, "\01\02\03");
```

В результате на экране будет отображена кнопка с надписью «АБВ».

Если нам требуется ряд кнопок с одним символом в надписи, то удобнее воспользоваться командой ***Cmd_Keys***:

```
Ft_Gpu_CoCmd_Keys(phost,x,y,w,h, z,s,"12345");
```

Здесь
x и y — координаты верхнего левого угла кнопки;
w и h — ширина и высота в пикселях набора кнопок;
z — указатель на шрифт, который будет использован в надписи на кнопке;
s — вид кнопки: OPT_FLAT — плоское изображение или OPT_3D — объемное;
"12345" — выводимая надпись, количество символов в ней определяет количество кнопок в линии, в данном случае будет нарисовано пять кнопок.

По этой команде FT800 выводит на экран в ряд несколько кнопок. На первой кнопке будет символ «1», на второй «2» и т. д. Количество кнопок в ряду определяется числом символов в строковой переменной. Такой формат команды удобен, если нам требуется нарисовать на экране клавиатуру.

Для того чтобы изображение кнопок выполняло их функцию, необходимо добавить обработку касания сенсорного экрана в требуемой области. Для этого микросхема FT800 имеет встроенный контроллер резистивного 4-проводного сенсорного экрана. Набор команд для работы с ним позволяет получать координаты точки касания, отслеживать не только касание в области кнопок или других графических объектов, но и изменение координат касания для слайдеров.

Рассмотрим алгоритм работы с контроллером сенсорного экрана для определения касания экранных кнопок. Каждому графическому элементу, выводимому на экран дисплея, может быть назначена своя уникальная метка (TAG) в диапазоне номеров от 1 до 255. При использовании такой метки контроллер FT800 без участия управляющего МК определяет, попадают ли координаты текущего касания сенсорного экрана в область графического объекта с данной меткой. Если касание произошло в области графического объекта, то в специальный регистр REG_TOUCH_TAG заносится значение метки объекта, в области которого это касание зафиксировано. Периодически опрашивая этот регистр, управляющий МК узнает, какой из элементов графического интерфейса был активирован.

Если в разрабатываемом изделии применяются только кнопки и/или не требуются координаты точки касания, то использование меток является очень удобным и простым способом для контроля элементов интерфейса. Причем благодаря команде ***Cmd_Keys*** (см. выше) назначение меток происходит автоматически. Каждой кнопке из набора будет соответствовать своя метка в зависимости от номера символа на ней. Для встроенных шрифтов номер метки будет совпадать с номером символа в ASCII таблице. Это очень удобно, если по нажатию кнопки необходимо выводить информацию на экран. Для пользовательского шрифта номер метки будет соответствовать номеру символа из таблицы в файле **.rtf*, созданного при конвертировании этого шрифта. В листинге показаны вывод ряда из трех кнопок по команде ***Cmd_Keys***, обработка их касания и вывод соответствующих символов в виде текста (рис. 3).

Рис. 3. Пример вывода на экран ряда кнопок командой Cmd_Keys и вывод текстовой строки с использованием пользовательского шрифта

```
StringArray[0] = '\0';
count = 0;
while(1)
{
   Ft_Gpu_CoCmd_Dlstart(phost);
   Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(64,64,64));
   Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
   Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xff,0xff));
   ReadWord = Ft_Gpu_Hal_Rd8(phost, REG_TOUCH_TAG); // чтение регистра контроллера сенсорного
экрана

   if (ReadWord!=0)  // цикл для формирования текстовой строки, которая будет выводиться
при нажатии кнопок
   {
   StringArray2[count] = (char)ReadWord;
   StringArray2[count+1] = '\0';
   count++;
   if (count >=18){count=0;}
   }

   Ft_Gpu_CoCmd_Text(phost,5, 80, 7, OPT_CENTERY, StringArray);  // вывод символов на экран,
в соответствии с нажатой кнопкой
   Ft_Gpu_CoCmd_FgColor(phost,0x008000);
   Ft_Gpu_CoCmd_Keys(phost,(115),(212),250,50,7,0,"\x01\x02\x03");  // вывод трех кнопок в ряд
с именами "А", "Б" и "В"

   Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
   Ft_Gpu_CoCmd_Swap(phost);
   Ft_App_Flush_Co_Buffer(phost);
   Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
   Ft_Gpu_Hal_Sleep(30);

}
```

Используя команду ***Cmd_Keys***, можно не задумываться о присваивании каждому элементу своей метки. Это делается автоматически.

Если мы используем команду ***Cmd_Button*** и выводим этой командой на экран несколько кнопок, мы должны самостоятельно присвоить каждой из них свою метку. Это осуществляется командой ***TAG(x)***:

```
Ft_App_WrCoCmd_Buffer(phost,TAG(x))
```

где x — номер метки от 1 до 255.

Данная команда присвоения метки действует глобально, то есть этот номер будет присвоен всем объектам, вызываемым в программе за ней. Действие команды может быть отменено ее новым вызовом с новым аргументом или специальной командой ***TAG_MASK (0)***, где

аргумент «0» говорит о запрете присваивания номера следующим за командой объектам:

```
Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(0);
```

Отменой запрета на присваивание метки служит эта же команда с аргументом «1» — ***TAG_MASK (1)***.

Добавим к предыдущему листингу после вызова функции ***Cmd_Keys*** следующие команды:

```
//Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(0));
Ft_App_WrCoCmd_Buffer(phost,TAG(1));
Ft_Gpu_CoCmd_Button(phost,(115),(152),100,50,7,0,"\01");
//Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(1));
Ft_App_WrCoCmd_Buffer(phost,TAG(2));
Ft_Gpu_CoCmd_Button(phost,(220),(152),100,50,7,0,"\02");
```

На экране теперь будут отображаться пять кнопок: три — по команде ***Cmd_Keys*** и две — по командам ***Cmd_Button*** (рис. 4). Метки новым кнопкам присвоены в соответствии с таблицей символов (**.rtf*), и новые кнопки фактически дублируют кнопки «**А**» и «**Б**» из нижнего ряда.

Снимем комментарий с команды ***TAG_MASK***. В результате метки новым кнопкам не будут назначены и не будет происходить обновления регистра REG_TOUCH_TAG при их касании. То есть кнопки будут не активны. Уберем второй комментарий, и вторая кнопка вновь будет активирована.

Мы привели базовые примеры работы с загружаемыми шрифтами, с выводом на экран кнопок и обработки их касания. Эти моменты являются основой для разработки любого графического пользовательского интерфейса. Набор команд микросхемы FT800 специально приспособлен для таких приложений и прост для освоения. А возможность управления графическим контроллером

Рис. 4. Вывод на экран двух кнопок командой Cmd_Button, ряда из трех кнопок командой Cmd_Keys и текстовой строки

с помощью 8-разрядного микроконтроллера [2] позволяет использовать TFT-дисплеи в существующих проектах без перехода на более мощные процессоры.

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. **http://www.mymcu.ru/support/eve_font_converter/**
4. Application Note AN 245. FT800 Sample Application Introduction for VM800B and VM800C Development Kits and Windows PC.
5. FT800 Programmer Guide.