

Brian's corner

The FTDI EVE graphics controller (1)



ELEKTRONIKASvet




We at Svet elektronike are proud on what we do since 1994!

Slovenian website

www.svet.el.si

English website

www.svet-el.si/english



www.svet-el.si/english

[Home](#)
[Blog](#)
[Inputs & Outputs](#)
[Data Displays](#)
[Data Measurement](#)
[Development Tools](#)
[Projects](#)
[Download](#)
[Brian's corner](#)

About the Book

- Table of Contents
- Authors
- Contact us
- Have your PCB
- My profile
- Shop
- Terms & conditions

Programming Blog

Brian is a well known author who uses different programming platforms and programs for his projects and articles. He regularly writes for different electronics magazines, including Svet elektronike (published in the Slovenian language).

Because Brian's articles are really great we have decided to publish them in English free of charge for all visitors of our web page.

Using NeoPixel LED's to Build a Unique Wall Clock

Written by Brian Miller
Friday, 27 May 2016 00:00



Like everyone interested in electronics, I'm hearing daily about the "Internet of Things" as well as "wearables" which I think is driven by an industry desperate to find the next "Great New Product" to sell us. I'm not sure how useful these new ideas will be.

[More...](#)


Last Updated on Tuesday, 31 May 2016 09:04

About the Book

- Table of Contents
- Authors
- Contact us
- Have your PCB
- My profile
- Shop
- Terms & conditions

Using NeoPixel LED's to Build a Unique Wall Clock

Written by Brian Miller
Friday, 27 May 2016 00:00



Like everyone interested in electronics, I'm hearing daily about the "Internet of Things" as well as "wearables" which I think is driven by an industry desperate to find the next "Great New Product" to sell us. I'm not sure how useful these new ideas will be.

However, I do feel that LEDs are products that will see an exponential growth in the future. There is no question that their efficiency and versatility make them the perfect choice for almost all general lighting needs. Although cost is an issue today, that will soon disappear because the cost of LEDs is on a steady downward trajectory.

You have likely seen large outdoor displays used for sporting events and concerts. These are very expensive, highly specialized displays, but a similar idea is now being found in smaller commercial displays used for promotional and advertising purposes. Such displays are usually a square meter and upwards in size, and are made up of thousands of individually-driven LEDs. Although displays like this used for traffic notifications are generally only one color, most of the units used for any form of advertising use RGB LEDs and can reproduce the whole color spectrum.

While the cost of such a large number of RGB LEDs is not insignificant, what is more of an issue is the problem of driving them in such a way that they can be individually controlled. You have probably built projects using multiplexed 7-segment LED displays, so you are familiar with that method of reducing the number of wires/drivers needed to handle multiple LEDs. Multiplexing of LEDs works because the human eye has a persistence of vision, which means it can't respond to changes in a light source at a frequency much beyond about 40 Hz. So, when we look at a 4-digit LED multiplexed display, our brain will make out the fact that, at any given moment, only one of the 4 LED digits is actually lit. While you can certainly use multiplexing techniques for large panels of LEDs, as you increase the multiplexing ratio, the "perceived" brightness will decrease as this ratio increases. Since brightness is a key factor for outdoor displays, this is definitely a limitation.

To facilitate the design of large LED display panels (and other related display products), the concept of the NeoPixel LED was invented. Initially, the idea was to develop a small IC that could control 3 LEDs (Red, Green, Blue) utilizing 8-bit PWM drive for each LED. Theoretically this gives "24-bit" color resolution: in practice the LEDs don't really yield this high a color resolution, but it is nonetheless very good. To further simplify the wiring, these LED driver ICs use a single-wire control protocol, and each IC device contains both a Data in and a Data out pin, allowing many such devices to be strung together in a "daisy-chain" configuration.

In practice, it's possible to string up to about 100 RGB LED driver ICs in this fashion, using only three lines (Vcc, ground and data). This is a tremendous reduction in the amount of wiring and circuitry needed to drive 300 completely independent LEDs (100 x Red, Green and Blue). While the original idea was to have this IC used with separate Red, Green and Blue LEDs (or a single RGB LED unit), it soon made sense to manufacture RGB LEDs with this driver IC built in. Such LED/driver assemblies are called NeoPixel LEDs. These driver ICs were developed by World-Semi in Shenzhen, China, and, as far as I can tell, all of the NeoPixel modules come from the far-east. Let's look at some technical details of the most common NeoPixel IC driver chip, the WS2812.

Parameter	Information	Maximum
VCC	5.0V to 5.5V	5.5V
I _{LED}	10.0mA to 15.0mA	15.0mA
I _{DC}	10.0mA to 15.0mA	15.0mA
I _{PK}	10.0mA to 15.0mA	15.0mA
R _{LED}	10Ω to 15Ω	15Ω

[Download program](#)
[Download article](#)

Using NeoPixel LED's to Build a Unique Wall Clock
2015_0226_31

Article on the web site

Download programs and
Download PDF of the article

WWW.SVET-EL.SI






Medio KIT - Svet elektronike magazine

www.svet-el.si/Download/Medio_KIT_08_16.pdf

AX elektronika d.o.o.
Špruha 33
SI-1236 Trzin
Slovenia / Europe

00386 1 549 14 00
www.svet-el.si
stik@svet-el.si

WWW.SVET-ME.SI






Presentation of Svet mehatronike magazine

www.svet-el.si/Download/Medio_KIT_08_16.pdf

The FTDI EVE graphics controller (1)

Most electronics enthusiasts strive to make their projects as user-friendly and commercial in appearance as possible. I'm no exception, and lately I have been trying to use TFT colour displays with touch-screen capability whenever practical. Although they are still somewhat expensive, you should also consider the savings that you can obtain by eliminating many of the switches, potentiometers, etc. that the touch-screen can replace.

Sophisticated graphics using only a modest 8-bit MCU

In late 2013, FTDI (the company that makes the USB-serial interface chips we all use) started advertising their new EVE display controller chips. Soon afterward, Mikroelektronika started selling 4.3" TFT display boards based upon this new controller. I got quite intrigued at this point, and started to look into the EVE controllers more closely.

Basically, the EVE controller chip is a very intelligent TFT display controller which can handle TFT panels up to 512 x 512 pixels, in up to 18-bit colour depth, or resolution. They will interface to any MCU with an SPI port, which covers most all MCUs apart from a few low pin-count ones. The EVE SPI interface is high speed (up to 30 Mb/s). This, coupled with the fact that the EVE controller is an intelligent one, executing high-level graphics commands, means that you can achieve very impressive graphics displays, even if you are hosting it on a modest 8-bit MCU, as FTDI's advertisements claim. In my personal experience, it is possible to implement a very nice GUI using the EVE controller driven with the Atmel AVR Atmega328 (as found on the Arduino Uno board, for example.)

The EVE controller provides a comprehensive variety of low-level graphics commands such as those needed to clear all/part of the screen, draw lines, rectangles, circles and other basic block figures. In addition, it also contains a co-processor engine, which adds a whole series of widgets such as buttons, sliders, rotary controls, clocks, switches, progress bars, etc. These are generated quite easily by sending out the proper widget command, along with the parameters needed to customize it to your needs: i.e. size, orientation, full scale value, etc.

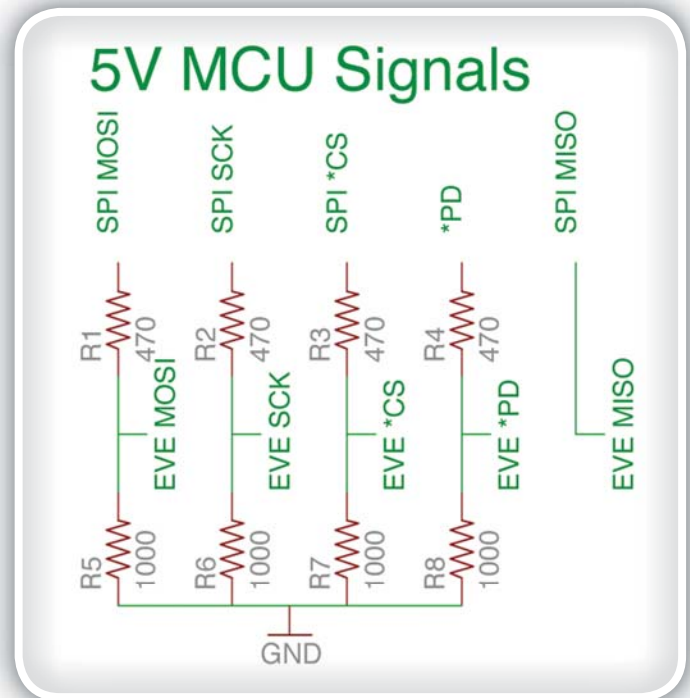


Figure 1: If you are using the Mikroelektronika Connect EVE module (3.3V Logic and power supply) with a 5V MCU, you can match signal levels using this resistive voltage divider.

The EVE controller also handles the resistive touch-screen functionality. In addition to the normal touch screen routines, where the controller returns the X-Y value of the spot being pressed, the widgets mentioned earlier can be “tagged” with an ID number, and when the user touches those widgets, this distinctive “tag” ID is returned to your program. This makes a touch-enabled GUI quite easy to implement, even when using only a modest 8-bit MCU.

Finally, the EVE controller provides an audio output. I'll discuss this further, but at this point, let's just say that the EVE can “play” sound files of various compressed formats. Also, it implements a sound synthesizer function, which allows it to play musical notes, melodies, or provide sound effects.

Now that you have a basic idea of what EVE will do, let's step back a bit, and take a comparative look at the various other methods that are available to provide a colour TFT touch-screen capability to your MCU project. In an article back in SE issue #xxx, I outlined two basic approaches to adding a TFT display to your project. The first one involves the use of a “dumb” TFT display which employs an 8-bit (or 16-bit) parallel interface to your MCU of choice. These



Photo 1: This is a touch-screen multi-function IR remote control I built a while back, using 4D Systems uLCD-43PT intelligent TFT touchscreen module. I am now designing a similar device using the much less expensive EVE display modules.

displays are very inexpensive on eBay (10-20 EUR) but they do have some disadvantages:

- They require up to 26 digital I/O lines on your MCU, forcing you to use a higher pin-count MCU than you might otherwise employ. All TFT displays operate on 3.3V, so all of these I/O lines must be at 3.3V levels.
- You must find the proper driver firmware for your chosen MCU, and this driver often uses up a lot of Flash memory space that you could otherwise use for your own program. You should also be aware that while these displays come in a limited number of physical sizes, there are many different LCD driver chips used on the different panels, which makes finding the proper driver somewhat more difficult.

I should mention that some of these “dumb” TFT displays are now being designed with direct Arduino compatibility. That is,

they are mounted on a PCB which will plug directly into an Arduino Mega 2560 (or they include a “transition” PCB that goes between the TFT display itself and the Arduino Mega 2560). The Mega2560 MCU has plenty of I/O capacity, which addresses concern #1 above, and some of these display modules come with Arduino drivers, covering concern #2. Such Arduino-targeted boards contain level-shifting chips to handle the 5V levels present on the Arduino Mega 2560.

I tried one of these “dumb” TFT display/Arduino combinations, and while it may not be typical, I found that while the display module that I received worked, it was so dim that it was unusable. I suspect that these Chinese vendors are selling “seconds”: TFT displays that don't pass the normal QC standards of the TFT panel manufacturer. I have seen feedback from customers, on various web-sites, regarding this shortcoming. You may want to think twice about going this route.

The second approach involves a serially-interfaced TFT display which contains its own intelligent display controller MCU. Such displays contain a whole library of high-level graphics routines and touch-screen handling, all of which you can access by sending the appropriate commands to the display, over a high-speed serial data link. Such commands are pretty compact in relation to the complexity of the graphics objects that they generate, so a standard serial data link, at a high baud rate (i.e. 115,200) is adequate to produce quite useable graphics display. I've had excellent results on several projects using the uLCD series of displays from 4D Systems in Australia. They come in many sizes from small mobile-phone sizes up to large 4.3” displays. The SE article that I referred to earlier, covered these displays in detail, as well as including lots of hints and examples of Bascom/AVR code to use with them. I recently finished a personal project using 4D Systems 4.3” uLCD display: an IR remote control which controlled

Feature	FTDI VM800B	Mikroelektronika Connect-EVE	4D Systems 4DLCD-FT843
Power supply	3.3V or 5V	3.3V	3.3V
Logic levels	3.3V or 5V	3.3V (not 5V Tolerant)	3.3V (not 5V tolerant)
Ease of Panel mount	Easy (bezel included)	Mounting holes but no bezel	Difficult without Bezel and Breakout board kit (available separately)
Interface connector	10 pin 0.1” header	10 pin 0.1” header & 2X5 0.1” header	10 pin 0.5mm FPC flex. ribbon
Audio capability	Amplifier & speaker	Audio output pin	Audio output pin
Price	64 EUR	50 EUR	43 EUR (includes Bezel and breakout board)

Table 1: Comparison of unique features amongst three currently-available 4.3” EVE-based display modules. All modules feature a QVGA resolution of 480 X 272 pixels, and 16-bit colour depth.



my flat-screen TV and the three peripheral units associated with it. In place of the myriad of small buttons present on four separate remote controls, this unit features a clear, easy to use graphics display containing only the commonly-used buttons on each of the individual IR remotes. The user can quickly switch amongst the four “screens” (one per remote unit) using a small push-button. An added advantage to this approach is that this unit is easy to see in the dark, which is not true of a standard commercial IR remote. Photo 1 shows one of the two such units that I built recently.

Photo 1. This is a touch-screen multi-function IR remote control I built a while back, using 4D Systems uLCD-43PT intelligent TFT touchscreen module. I am now designing a similar device using the much less expensive EVE display modules.

4D System's new “Workshop” IDE program contains a very high-level method of designing the various graphics screens needed for such a project. If you are familiar with Visual Basic, you would find 4D Systems “Workshop” IDE very easy to use in designing a nice GUI for your application. The disadvantage of the 4D Systems μ LCD display is that they are relatively expensive. The 4.3” model, with a resistive touch-screen, cost about 100 EUR, when I bought them last year. They also need a μ SD card to be inserted into an on-board socket- to hold the files containing the graphics images for the various “widgets” that are a part of the user's GUI design. This adds about another 5 EUR to the price of the display.

Photo 2. This is the control PCB side of the Connect-EVE 4.3” display module. Note that it contains both a 10 pin interface and a 2X5 header, with all necessary signals available on both. The “150 MA OP” label is one that I added to remind me that the module draws 150 milliamp when



operating, which is important since I am using it in battery-powered projects .

I happened to have 2 extra 4.3” μ LCD displays left over after finishing a commercial project I designed/built recently, so it cost me basically nothing to use my “spares” for the IR remote controller project. But, since I felt that the average user would not likely be interested in such an expensive IR remote, I decided against writing an article about this project, at least in its present form (but I am working on an EVE version/article).

So, with this quick comparison of the various TFT display options out of the way, let's look in more detail at the EVE controller and the display panels that are currently available.

Whats' So Great About EVE?

Since I am a fan of the 4D Systems intelligent μ LCD modules, you might wonder how I became interested in the EVE display controller chips. Well, to start with, there was the issue of price. Right from the start, it was clear that EVE-based TFT display modules were going to be a lot less expensive than the 4D systems μ LCD display modules. For example, the 4.3” size (which I find ideal for many of my projects) was available in Mikroelektronika's Connect-EVE module, costing about 50 EUR. This is about $\frac{1}{2}$ the cost of a comparable 4D Systems μ LCD module.

Another consideration concerned the interface method. While I generally like using the serial port method used by μ LCD modules, there can be some disadvantages to it. For any graphics applications requiring fast motion, or complex graphics operations, the speed of the serial port can be a limiting factor. Also, many AVR MCUs contain only a single serial port, so a problem exists if you have an additional peripheral device that also needs a serial port. Indeed, if your project needs a USB port capability, you will often use an FTDI USB-serial interface chip for this purpose, so two serial ports would be needed if you also use a μ LCD display.

The EVE graphics controller chip instead uses an SPI interface, along with a couple of other control lines (*PD and interrupt). An SPI port is much faster than a serial port. In the case of the EVE chip, it is capable of running at up to 30 Mb/s. You won't be able to achieve this high a rate with common AVR chips, as their highest SPI rate is $\text{SYSCLK}/2$ (8 Mb/s when using the 16 MHz crystal common on Arduino boards). Still, this is 69 times faster than the 115,200 baud rate that you could use between an AVR MCU and a μ LCD display.

The other advantage to the SPI protocol is that you can have many different devices



Photo 2: This is the control PCB side of the Connect-EVE 4.3" display module. Note that it contains both a 10 pin interface and a 2X5 header, with all necessary signals available on both. The "150 MA OP" label is one that I added to remind me that the module draws 150 milliamp when operating, which is important since I am using it in battery-powered projects.

sharing a single SPI port, so long as they all don't need to communicate simultaneously. So, even with a modest Atmega328 MCU, you can drive an EVE display along with several other SPI peripherals, as well as any other peripheral that needs a serial port.

A second advantage to the EVE display controller chip, is that it uses an advanced method of generating all of the "Widgets" or graphics objects which you might need. These are all generated and stored within the EVE controller chip itself. In contrast, the μ LCD displays form their widgets using bit-map images, which must be stored in the μ SD card mounted on the μ LCD board. This μ SD card also holds any other images that you need to display, as well as any sound files. While the cost of a μ SD card is low, it's important to note that one must download these bitmap files to the μ SD card using the PC computer that is running the 4D Systems "workshop" IDE program. This usually requires the use of a USB card-reader module on most PCs (apart from laptops). To put this in another way: All of the code needed to generate a GUI on an EVE display is contained in the firmware you write for your chosen MCU, which can be easily distributed.

The μ LCD displays will require that you write firmware for your MCU, which you can easily distribute, but you (or the end-user) must also have access to the 4D Systems "Workshop" IDE (which runs only on a PC) to generate the necessary GUI bitmaps. These must then be downloaded to a μ SD card, which is then inserted into the μ LCD display's on-board card socket.

As you can see, the EVE method is more straight-forward, particularly if others need to duplicate your project.

As I mentioned earlier, both EVE-based and μ LCD modules handle all interaction with the touch screen using internal, high-level routines. That is to say, your program is relieved of the task of constantly scanning the resistive touch screen

display for presses, and then doing a lot of calculations with X,Y co-ordinates, to determine which of the buttons, widgets, etc. was actually touched. (or adjusted). On both of these display, your program merely polls the display controller periodically, and it returns a code which identifies which widget on the screen was touched. This is really nice!

The last major feature of both of these display modules involves sound generation capabilities. The μ LCD display's sound capability involves playing various types of compressed sound files, at a fairly low bit-rate and resolution. This file(s) must be downloaded to a μ SD card, which is then mounted into the display's on-board socket. When I used μ LCD displays for my most recent project, this was the only way to generate sound. In other words, if you needed something as simple as a "beep" or a click to indicate a screen touch, you had to download a compressed audio file to the μ SD card. I think they should have allowed for a simple routine which generates a square-wave tone, the frequency and duration of which you could pass to the display as command parameters. Something along the lines of Bascom/AVR's SOUND statement would have been fine. The larger μ LCD displays contain an audio amplifier and a very small, 12 mm speaker.

In the case of the EVE display controller, the sound functions are somewhat more versatile. It can play sound files in various formats (8-bit PCM, μ LAW, 4-bit ADPCM) like the μ LCD displays. However it also contains the equivalent of a MIDI synthesizer (something like the one I described in my SE article #xxx) which can either play simple musical melodies, or be used to provide simple beeps and clicks as needed for user interaction with the GUI. Playing a note consists of just a few short commands specifying the MIDI instrument, musical note and note duration. The Mikroelektronika Connect-EVE modules that I am using have an audio output pin, but no amplifier or speaker on-board. I haven't needed or tried out the sound capability of the EVE controller yet.

What's your preference: 5V or 3.3V Displays?

If you are using an AVR MCU, you are probably running it at a V_{cc} 5 volts. You get full speed operation that way, and many of the common peripheral IC devices operate on 5V. The most commonly-used Arduino boards also operate on 5V, although this is gradually changing with the advent of the Due, as well as numerous "clones" from other manufacturers that operate on either 3.3V, or both 3.3V and 5V.

Regardless of what TFT display module you choose, they all operate internally at 3.3V. However, the power supply voltage that you must provide to the module will vary from one manufacturer to the next. Also the necessary logic levels that are required will also vary. It's critical to note that supplying a module that requires a 3.3V power

PROGRAMMING

source with 5V will likely destroy it, as will the application of 5V logic level signals to a module that calls for 3.3V (maximum) logic levels.

Table 1 shows some important features of the three 4.3" EVE-based display modules that were available when I wrote this article. Here you can see that only FTDI's own modules are capable of operating with either a 3.3 or 5 volt power supply. Related to this, you can see that they are also the only modules that will interface to either 3.3 or 5V logic-level signals. If you are using an MCU running on 5V, like an Arduino Uno for example, its probably best to choose the FTDI module rather than worrying about adding your own level-shifting circuitry. Incidentally, FTDI also sell similar EVE modules that contain a smaller board (for the EVE) that connects up to the TFT display with a flex cable. These models don't come with a mounting bezel for the display however, so you might find them harder to use.

I started working with EVE-based displays after purchasing a few Mikroelektronika Connect-EVE modules, which I chose because they were the first EVE display modules available. These displays operate on 3.3V only, and the first few projects that I had in mind for them called for a 5V AVR MCU, both for speed considerations, and because most of the other peripheral devices needed were 5V devices.

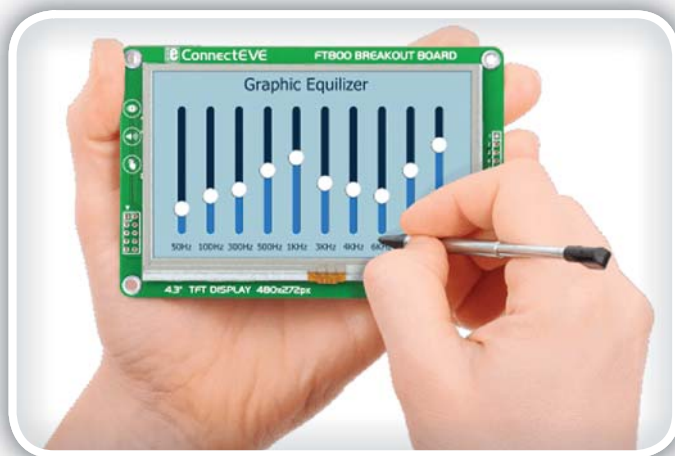
Although I powered the EVE display itself with a separate 3.3V regulated supply, I also found that the required logic-level conversion was easily accomplished using only a simple resistive divider network (1K and 470 ohm resistors) on the MCU's MOSI, SCK and -PD output pins. The Connect-EVE's 3.3V logic-level MISO output signal was sufficient to drive the ATmega328's MISO pin directly.

The Connect-EVE's -INT pin would also have interfaced to the ATmega328 directly, but I did not need that pin for my design. That being said, I should add that I was using an SPI clock rate of only 4 MHz, due to limitations of some of the other SPI devices used in the project. But I am doubtful that the EVE display module would operate at much higher SPI rates (it's rated up to 30 Mb/s maximum), using this simple resistive level-shifting method. (shown in Figure 1).

Figure 1. If you are using the Mikroelektronika Connect EVE module (3.3V Logic and power supply) with a 5V MCU, you can match signal levels using this resistive voltage divider.

I didn't make use of the audio capabilities of the EVE controller, but it should be noted that the FTDI VM800B modules contain both an amplifier/filter and a small speaker, which the other two modules do not.

When studying such exciting new TFT display modules, it is quite easy to overlook the matter of mounting it in a cabinet or on a panel. Like most readers, I am an electronics enthusiast without access to customized plastic enclosures with all the cut-outs and mounting tabs/brackets already present. Unlike most readers, I do have a home-built CNC milling machine available, so I can easily cut out



rectangular holes in aluminum or plastic enclosures. So, personally, it was fairly easy to mount the Mikroelektronika Connect-EVE modules by merely cutting the proper-sized rectangular hole in an aluminum enclosure, and fastening it in place using 4 spacers and screws/nuts. If you don't have an easy way of cutting out a "clean" rectangular hole, the FTDI VM800B modules may be your best choice, as they come with a bezel which can hide any rough edges of the hole that you cut out. Alternately, if you like the low price of 4D System's 4DLCD-FT843 module, you can order the optional Bezel/Breakout board kit (which I included in the price shown in Table 1). This kit takes care of any mounting complications, as well as providing a small breakout PCB which mates up with the 10-way 0.5mm FPC ribbon cable, and provides a 0.1" header connection. I should add that using the Mikroelektronika Connect-EVE modules has the advantage that it takes up a bit less space on your front panel than using either of the other two models with their respective bezels.

EVE's Basic Architecture

Before discussing the graphics features of the EVE controller, I want to mention a couple of basic architectural details in which EVE differs from most other graphics controllers. Generally, graphics display controllers contain a fairly large RAM memory device, which is used to store whatever image is being displayed at the time. For the 4.3" TFT displays that we're talking about, the pixel resolution is 480 x 272, and the colour bit depth or resolution is 18-bits. This would theoretically require a RAM chip capacity of

480 X 272 X 3 bytes/pixel or 391,680 bytes.

If you want to allow for smooth screen updates (like when quick motion must be displayed). you generally have to double the size of this RAM. This allows for two screen buffers: one acting as the active display buffer, with the other one being the one that the host MCU "fills up" with the content of the next image "frame". Then, you can just swap the pointers to the buffers, to provide an instantly updated screen, i.e. preventing the display of a screen which visually "morphs" as its screen buffer is updated by the host MCU. So, you can see that close to 800K bytes of RAM is needed in this scenario.

The EVE controller doesn't work in this manner at all. Instead, it keeps track of all of the visual items needed on the active screen (lines, text, widgets, etc.) in a “display list”. Then, custom-designed, high-speed logic examines all of the aspects of the various objects in the display list, and, on a line-by-line basis, determines what pixel data has to be sent to the screen. So, while EVE still has to store the display list in RAM, you don't need to design in a RAM array that is capable of storing 2X a whole screen's worth of information at a time. Like the traditional graphic controller that contains double the amount of RAM needed for a given size screen and swaps it between consecutive frames, so too does the EVE controller maintain two separate display object lists, which it swaps between, for instant display updating.

While I don't pretend to understand the fine points of EVE's design, it must be a lot more efficient for the EVE controller to manipulate compact display lists than it is to try and manipulate bits in a large display RAM array. This would account for the fact that EVE-based displays are much less expensive than competing modules using conventional controller technology.

I should mention that if you are using an intelligent display module such as the 4D Systems μ LCD displays or EVE, which of the above two methods is actually used in your display module, is not overly important to you, the programmer of the Host MCU. In either case you are basically sending high-level graphics commands from the host MCU to the display controller serially, and how the display controller actually renders the video display, is somewhat transparent to the host MCU programmer.

In my opinion, it's quite a bit easier for the MCU programmer to learn the high-level graphics commands needed to successfully use the 4D Systems μ LCD displays, than it is to use EVE-based modules. With only a few simple commands (and parameters), you can clear the screen and put up some text or a box very easily on a μ LCD display. There are a lot more commands and parameters needed to use EVE-based display modules even for modest applications. However, once you advance to more complex GUI applications, I believe that the code complexity of either of these display options are comparable.

I think it's fair to say that if the average MCU programmer was not provided with a comprehensive EVE graphics library (written for his/her chosen MCU family), along with some reasonable demo programs, he/she would likely give up in despair if they had to “start from scratch”. Once you get going however, the low price of the EVE displays, along with its very powerful architecture, makes it well worth the effort.

My Introduction to EVE-based Displays

At the start, I carefully read the “preliminary” datasheet

[1] that FTDI published at the same time as their early advertisements. This datasheet made everything look very simple. However, all of their examples appeared to be written in a “pseudo-code” of sorts: i.e. all commands seemed to be simple “English” phrases. They also referred to various function calls which seemed to be taken from a C driver library, which was not yet available to the public. So, I delayed purchasing any actual display modules for a few months until the more comprehensive Programmer's Guide became available [1]. At the same time that this guide (with a “draft” watermark) was published, FTDI also released a software package for the Arduino, with drivers and example programs. While I am much more comfortable using Bascom/AVR for Atmel AVR MCUs, I did have some experience writing/understanding Arduino “sketches” (which the Arduino IDE basically surrounds with a “wrapper” to simplify things for newbie users, and then passes on to a C/C++ compiler). With at least some software available, I decided it was time to order some actual hardware. At that time, I chose the only modules available: Mikroelektronika's 4.3” Connect-EVE displays. Once the display modules arrived, I had a few choices on how to use them with the AVR MCUs that I customarily use in my projects. One obvious choice was to further investigate the Arduino software package created by FTDI themselves, as referred to in their datasheet and application notes.

Another choice was to use Mikroelektronika's own software which supports EVE-based displays. They sell compilers of various types (C, Basic and Pascal) targeted at several different MCU families (AVR, PIC and ARM). Associated with all of these compilers is their Visual TFT software package, which provides a high-level interface to many types of TFT displays, including their own Connect-EVE display module. Mikroelektronika had been very generous in providing me (being an electronics author) with a compiler of my choosing, at no charge. I had chosen their AVR Basic compiler as well as the Visual TFT package. So, I decided to try this first.

The basic concept behind the Visual TFT package is that you select the MCU development board that you have, as well as the type of TFT display module that you wish to use. Then you start out with a “clean slate”, so to speak, and drag/drop the various graphics elements you need onto a “virtual screen” contained within Visual TFT's IDE. If you have used Microsoft Visual Basic or Visual C++, then this GUI-based drag-drop method of programming will be familiar to you. Once you have done this, Visual TFT will generate the source code needed to implement your screen (or multiple screens if needed). Next you will be transferred to whatever Mikroelektronika compiler you are using. Here you would add your own code to handle all non-display aspects of your program. Also, the code needed to handle the various touch screen actions can be done either in Visual TFT, or later in the compiler itself. Then you “build” the program, and upload it to your MCU.

Although I did not own any Mikroelektronika MCU

development boards, I picked their XMEGA development board from the list, since I had an Atmel XMEGA Xplained board on hand. Having had prior experience with the Xmega Xplained board, I knew which of the 4 SPI ports available on the XMEGA devices, was accessible on the Xplained board's header socket. It turns out that Mikroelektronika's Xmega board used a different SPI port. After quite a bit of searching, I found the spot in the Visual TFT source code where the SPI port was defined and initialized, and made the necessary changes to some global definitions. After doing this, I was able to upload a very simple program to the Xmega Xplained board/Connect-EVE display, and everything worked fine. So far so good!

The next thing I did was to examine the source code generated by Visual TFT, to see if I could understand it enough to be able to integrate their code into whatever code I would be writing myself. Also, I had to determine how to make Visual TFT/Mikroelektronika AVR Basic compiler generate code that would work on the actual AVR devices that I commonly use (Mega 88, 328, 644, 1284 etc.). This is where I hit “a brick wall”.

It seems like Mikroelektronika first developed C compilers for the various MCU families and then went on to expand into Pascal and Basic. However, being so familiar with both Bascom/AVR and Visual Basic (PC), I found the syntax and conventions used by Mikroelektronika's Basic compiler to be quite different and confusing. To me, the code generated by Visual TFT looked more like C++ than Basic, and I struggled to follow it.

The Visual TFT program has to be able to generate code for both “intelligent” TFT displays (i.e. EVE) and many different types of “dumb” displays (as mentioned at the start of the article). As such, I believe it is generating non-optimal (and hard-to decipher) code. I think this is particularly true in the case of EVE-based displays. The fact that it is called upon to generate code in three different compiler languages, for a large number of MCUs in the PIC, AVR and ARM families contributes to making the code much more complex than it need be, in my opinion. To back up this observation, I found that a very simple Visual TFT demo program containing only a few buttons on the screen (with NO code in the handling routines for those button presses), generated a 40 Kilobyte AVR program. This is far too big to fit into the flash memory of an Arduino Uno's Mega328. In contrast, I have since written a pretty complicated Arduino sketch, containing a fairly sophisticated GUI (keypads, buttons & X-Y graphs etc.) and lots of code to handle several other peripheral chips. This sketch takes up only about 26K of Flash memory on a Mega328.

I don't mean to paint an unflattering picture of Mikroelektronika's software products, based solely on my own personal experience. If C++ is your main MCU development language, then you would probably be happy with Visual TFT and Mikroelektronika's C compilers. Since most of the EVE demo programs supplied by

Mikroelektronika were written in C for the PIC MCU family, I did not find it very helpful personally.

FTDI Drivers and Demos

Given the problems described in the previous section, I next decided to try FTDI's own Arduino driver/demo code for the Arduino [3]. I was able to compile FTDI's Arduino demo program easily enough using V1.5 of the Arduino IDE that I use (I'm sure V1.05 would work as well). It would have been nice had I been able to load the code directly into either the Arduino Uno or Mega2560 boards that I had on hand. However, since the Connect-EVE display module will only work on a 3.3V power supply, using 3.3V logic levels, neither of these 5V logic-level Arduino boards would work. So I wired the Connect-EVE up to a home-built circuit board containing a Atmega328. While this board ran the 'Mega328 at 5V, there was plenty of space on the board to add both a 3.3V regulator, and the necessary level-shifting circuitry. The simple resistive level-shifter that I used is shown in Figure 1.

After I loaded the FTDI Arduino demo program into the Mega328, I did not initially see anything appearing on the display. I was a bit surprised, as I had connected the Connect-EVE module up exactly as shown in FTDI AN 246 [3], which is the application note which accompanies this demo program. A lesson: don't stop reading this application note after you get to the wiring diagram, like I did! It turns out that later on in the app. note, it instructs you to define your screen size in the program. The Connect-EVE's 4.3” screen was the default, so I was OK there. But more importantly it tells you to un-comment one of the five lines which define which set of demo routines that you want to run. This demo program contains a lot of different demo functions, and an Arduino Uno (with a 'mega328) would not have nearly enough flash memory to hold this program if more than one of these sets of routines was included. By default, all of these 5 lines are commented-out, so, until you choose one line to un-comment, your program will compile OK, but nothing will appear on the screen! After my oversight was corrected, the program ran as designed, and the Connect-EVE display went through a set of demo routines. It was quite impressive. Success at last.

Of course, I had already gotten the Connect-EVE to work using a simple program compiled by the Mikroelektronika compiler and Visual TFT, only to find I couldn't follow the code it generated. To be honest, when I first examined the code in the FTDI Arduino sample program, I was similarly unable to make much sense of it. To begin with, it wasn't written using the normal syntax of an Arduino sketch, but rather as a C++ program. Also, where an Arduino sketch is generally quite easy-to-follow, with all of the complexities of the hardware driver hidden in an Arduino “class” library, this demo program did not define an “EVE” class at all.

To make things even more confusing (to a newbie), the C++ demo program was written to work with either an Arduino board or a PC computer interfaced to the EVE's SPI interface

via an FTDI USB interface chip programmed in the MPSSE mode (Multi-protocol Synchronous Serial Engine)

As a result, the program was full of compiler directives meant to instruct the compiler to generate code for whichever of the above options was chosen. This basically made the listing at least twice as long as it would have been for just the Arduino alone. That, combined with the fact that the demo program contained such a large number of different demo functions, made it hard for me (with C++ being my “third language”) to follow.

What I have done is go through all of the code and remove all of the conditionally-compiled code specific to the PC environment (#2 above). Then I further removed all of the “fancy” screen demos that were not necessary for a simple application. The resulting code is easier to follow, and makes it somewhat easier for a newbie to get started. For reference, this simple program, which initializes the screen, puts up a few buttons, text, etc takes up about 6K of AVR code. As I mentioned earlier, a pretty complex GUI program that I've subsequently written, takes only about 26K of Flash memory on a 'mega328. You can certainly do some complex programs with this EVE display, even with an MCU as modest as that used in an Arduino Uno. This

basic program will be included with Part 2 of this multi-part EVE article series.

In the next part of the article, I'll be covering some of the basic EVE functions that you will want to use, such as Text, lines, boxes, etc. I'll also cover some of the most useful widgets, such as the buttons, and how to draw graphs, etc. I'll also look at the Mikroelektronika Connect-EVE display module in more depth.

References

- [1] EVE FT800 Datasheet
 - ◇ http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT800.pdf
- [2] EVE FT800 Programmer's Guide
 - ◇ <http://www.ftdichip.com/Support/Documents/ProgramGuides/FT800%20Programmers%20Guide.pdf>
- [3] AN_246 VM800CB SampleApp_Arduino_Introduction
 - ◇ http://www.ftdichip.com/Support/Documents/AppNotes/AN_246%20VM800CB_SampleApp_Arduino_Introduction.pdf

www.svet-el.si/english