History shows that disruptions in the mobile market can happen when least expected. Is the current OS duopoly created by Apple and Google a prime target for disruption? Is an industry with six to eight competing operating systems (or variants) preferable to an industry with only two or three dominant platforms? Here is a look at some of the forces that may be creating the next mobile phone disruption.

Brewing to Create a Mobile Market Disruption?
By Guy Agin
Director of Strategy & Business Development, Red Bend Software
In the Spotlight: Rob Gray
Freelance Embedded Electronics Designer

Did you always know you wanted to be an engineer?
I was living in London in the months spanning Christmas 1978/9, at the time I was a photographer but I’ve had an interest in electronics since I was a lad. I remember wanting to control the direction of a motor when I was about 10 years old. I made what I now recognize as an H-bridge from switches and wires nailed to a square of chip board.

I’ve always had an interest in model railways, so moving on 15 years and several continents I find myself sitting in my London flat wondering how I would control the boom gates on a model train layout such that when a train approached a crossing the gates would lower and cars would stop.

I started drawing circuits using the technology that I was familiar with, namely magnets, reed switches etc. and while the results weren’t very clever the seed was sown. I then bought some books, most notably one called “The CMOS Cookbook,” and was amazed at the possibilities these little black multi-legged devices offered.

On my return to Australia I started door knocking to all the engineering firms I could find in the phone book. I had nothing but my enthusiasm and some rolled up circuit diagrams under my arm and while I’m sure my circuits didn’t impress anyone my enthusiasm obviously did, and I found myself changing vacations after talking to the owner of the first firm I approached.

What are your favorite hardware tools?
Without doubt my logic analyzer. How anyone can design microprocessor/controller systems without one I have no idea and I’ve used them since the 80s although the requirements have changed over the years. In those days most peripherals were external to the processor and we were dealing with both data flow and timing issues so a wide and fast analyzer was needed.

These days most high speed peripherals are internal to the controller, the hard stuff has been done by the IC manufacturer so while there are still some timing issues it’s more a case of looking at data flow on serial links of various persuasions. For this you can get a really good analyser with serial decoders for $150 and a fantastic one for about twice that.

Do you have any tricks up your sleeve?
With modern SMD components it’s no longer practical or even possible to connect test equipment directly to the component for debugging prototypes. And trying to hold a probe on a TQFP leg is an accident waiting to happen. This means a method of connecting test gear should be implemented on the PCB and the most obvious method is to add a header. However these same SMDs also mean smaller and tighter boards, even a 6-way standard-pitch header consumes a huge amount of board real estate.

My solution is distributed test points. I identify the signals that will be needed for debugging and ensure that there is an access point somewhere on the board for those signals. This can be as simple as not having solder mask tinfoil on a via, but normally I place an over-sized via on the signal, either replacing an existing via or just dropped on the signal trace in a convenient location.

This test via is hexagonal in shape to help identify it and I often add an overlay label as well. It also has a larger hole to make it easier to securely hold a probe on the test point or even solder in a flying wire.

Do you have any noteworthy engineering experiences?
While working in the R&D section of a large multi-national computer firm I was asked to cycle the power on one of the mainframes. The computer in question had been shut down with all apps terminated, disks spun down, databases closed etc. I had to do was press the power button.

The button in question was a type that did nothing electrically when pressed but went open circuit when released which was fortunate because the instant I pressed it I realised I was about to pull the rug from under the wrong mainframe. Now killing the power of a mainframe without warning is a very bad thing but I had not released the button so no harm done, but like a soldier standing on a landmine I couldn’t move and therefore couldn’t tell anyone of my predicament.

Eventually someone came into the room, they informed all concerned parties and an orderly mainframe shutdown was started. However I had to stand there for some time providing the chief source of office entertainment until I could release the button.

On a more serious note I was electrocuted once (well several times really but once of particular note). With digital clocks in every home appliance from TVs to toothbrushes it can be a real pain to reset them all after disconnecting power from a house, so given that I’m a lazy fellow I tend to work on simple jobs with the wiring “hot”. Normally the first time I touch a bare wire I use the back of my hand to feel it’s live my muscles will pull the hand away, but on this occasion I was not concentrating and grabbed the bare wires between my thumb and first finger.

The experience is not something I would recommend and my hand would not let go of the wires. It all happened very fast but I think what saved me was the fact that my bicep also contracted and pulled my arm away from the wire, as the wire was short and mechanically fixed at the other end my arm muscles overrode my finger muscles and pulled my hand off the wire.

For some time I had burn holes in my skin, and I no longer have any mains-driven digital clocks.

And in the “famous last words” category, I remember looking at a microprocessor datasheet for the first time in the early 80s and thinking “These things are useless.”

What are you currently working on?
My major field of interest is serial monitoring and control networks and I’m currently spending a large part of my time designing two projects in this domain.

One is a dual processor board featuring an LPC1227 and an ATMega1284, it’s ostensibly an Arduino with an LPC co-processor that handles RS-485 networking, but can be used the other way around, that is an LPC board with an AVR co-processor for IO.

The other project is a quite complex Arduino Due work alike with a SAM3X8E. This board enables all the good stuff that the Arduino Due napples, for example 4-bit microSD, Ethernet, external RAM and two CAN interfaces. This board also has hardening for all IO and is designed to be a usable controller in its own right, often with no extra components required.
We are witnessing a duopoly in the mobile market, where two device platforms, iOS and Android, make up 90 percent of the smartphone operating system market. Apple and Samsung are the two dominant manufacturers that capture most of the volume and profits. Nokia with Windows Phone and BlackBerry are far behind. In the current market no analyst is betting against Apple, Google or Samsung to foresee any dramatic shifts. However, history shows that market disruptions can happen when least expected.

Five years ago the Symbian smartphones from Nokia were by far the market leaders, and while the iPhone had already launched and Android was announced, no analyst was forecasting a major sea change in the market within three years. Is there some unnoticed force or initiative currently under the radar that will create a similar disruption?

One emerging trend that could potentially change the status quo is the Android ecosystem fragmentation. Increasingly there are non-Google certified Android devices. Typically these come from Chinese and other regional manufacturers that are using Android without Google services. The new wave of regional manufacturers, which operate at a lower cost, is already challenging the leading brands in Asia and other emerging markets such as Latin America and Eastern Europe. Once smartphone penetration peaks in developed markets, growth will slow and the focus will shift to emerging markets. At that point, the growth of those alternative Android ecosystems might threaten the dominance of Samsung, Apple and Google.

The other new development is the emergence of several new device platforms, all based on Linux OS at the core. There are various new platform initiatives, each with its own heritage, uniqueness and supporters but the following four are the most advanced:

- **Tizen** – Is an open source platform backed by Samsung, Intel and several mobile operators. It is an evolution of past attempted industry collaborations such as LiMO and MeeGo. While Samsung seems committed to Tizen, the market is waiting for commercial devices.

- **Firefox OS** – Is developed by Mozilla, backed by Telefonica, and several OEMs are committed to launch devices including ZTE, Huawei and Sony. Low-cost Firefox OS smartphones are already available in Latin America and Europe.

- **Ubuntu Touch OS** – Is developed by Canonical, the Linux vendor. The open source platform has the backing of several important operators who created the Carrier Advisory Group to influence the Ubuntu roadmap. To date, no major OEM has announced a commitment to create a device, and only a concept hybrid Ubuntu-Android device called Edge is planned to be produced in part by a recent crowd funding campaign.

- **Sailfish OS** – Is developed by Jolla, a company started by ex-Nokia employees, who built a new platform on top of Nokia’s discontinued MeeGo OS. Jolla is planning to build its own smartphone (the Jolla phone) and potentially license the platform to OEMs, probably in China.

History shows that disruptions in the mobile market can happen when least expected. Is the current OS duopoly created by Apple and Google a prime target for disruption? Is an industry with six to eight competing operating systems (or variants) preferable to an industry with only two or three dominant platforms? Here is a look at some of the forces that may be creating the next mobile phone disruption.

By Guy Agin
Director of Strategy & Business Development, Red Bend Software

**UNNOTICED FORCES**

Brewing to Create a Mobile Market Disruption?
Will these new initiatives fail and be relegated to the history dustbin, or will they create a market disruption? At this point it is too early to tell. What can be assessed are the factors that are essential to the success of a device platform in the mainstream smartphone market:

**Availability of applications and developer mindshare** – In today’s ecosystem-driven market, a limited third party application catalog might kill a platform. The new Linux platforms, which are naturally disadvantaged, appear to be tackling this issue by adding virtualization or compatibility layers that enable Android apps to run on the platform (Tizen, Ubuntu) or via promoting broad Web/HTML5 development (Firefox OS, Ubuntu).

**Consumer mindshare and positioning** – To capture consumer attention and overall industry mindshare the new platforms need to find unique differentiation and position the device for target segments where they can compete effectively. For example, low-cost smartphones for emerging markets (Firefox OS) or unique, enterprise PC / desktop duplication (Ubuntu).

**Dominant OEM support** – The new Linux platforms are not brought to market by deep pocketed industry giants such as Apple or Google. Therefore without at least one dominant OEM to create compelling devices in volumes they will struggle. Tizen has the strong backing of Samsung, while Firefox OS is supported by brands like ZTE and Sony. The others are lacking such OEM support.

**Operator support** – While operator support does not ensure a platform’s success, as seen with past failed operator group initiatives like LiMo and WAC, the lack of operator support can be fatal for a new platform. Firefox OS, Ubuntu and Tizen all have supporting operator groups. Telefonica is leading the charge by strongly pushing Firefox OS to its customers.

**Software and device management capabilities** – As proven by iOS and Android, to succeed, a modern smartphone platform must equip itself with the infrastructure for software updating and device management capabilities so that OEM and operator partners can keep the platform constantly updated via over-the-air services.

In conclusion, is an industry with six to eight competing operating systems (or variants) preferable to an industry with only two or three dominant platforms? Application developers might prefer fewer platforms to minimize their porting efforts. However, with more platforms, consumers might enjoy more choice and a variety of user experiences in devices, and other industry players such as challenger OEMs, cross-platform solution providers and mobile operators may benefit from the decreased dominance of giants such as Apple and Google. If broad application availability can be ensured for various platforms via compatibility layers or by standardizing on HTML5, it may tilt the balance in favor of a multiple platform industry.

---

**The new wave of regional manufacturers, which operate at a lower cost, is already challenging the leading brands in Asia and other emerging markets such as Latin America and Eastern Europe.**

---

---

---
Setting Up Your SD Card Image

Installing Arch Linux ARM starts off the same way as installing Raspbian. We must begin by downloading and preparing the Arch Image, which you can grab here.

Just to recap the instructions from Part 2, we’ll quickly go over how to prepare your SD Card.

First, mount the SD Card on your computer, if you’re using Windows, you merely have to plug it in, if you’re using Linux, you’ll have to get the right drive names using “df -h”. Once you’ve done that, write the Arch Linux ARM image to your SD Card using the Win32 Disk Imager Tool on Windows or the “dd” command on Linux, as described in the previous article.

Remember, be careful with these commands and make sure you select the correct drives, making a mistake could cause you to lose data that you want to keep! Writing these images erases whatever is written on the SD Cards.

When the write is complete, eject your card, load it up on your Raspberry Pi, and start it up.

Setting up Arch Linux ARM

Insert the formatted SD Card into your Raspberry Pi and turn it on. You should now see the system boot up and, provided nothing went wrong, prompt for login to the device, which by default is named ‘alampi.’

The default login information is:

Username: root
Password: root

At this point you should have a bash shell prompt waiting and be logged in as root. If you’re connected to a network, you can test your connectivity by using the ping command (e.g., ping google.com). If you don’t get any response you should check your network connectivity and fix any problems down the line.

Assuming you’ve gotten this far, you now should perform first-time setup of the device. This includes changing the root password, device name, setting the appropriate region and timezone, and installing basic software.

You can (and should) change the root password of your device from the default ‘root’ to something

Assuming you’ve checked out the first two installments of this series, you may have already gotten a Raspberry Pi and have it running on Raspbian Wheezy. Hopefully, you’ve even started tinkering with the platform and have created some projects of your own.

While Raspbian is an excellent choice for an operating system, with extensive support and numerous packages, you may want to use a system that is a bit more customized. Using Raspbian, or some other similar operating system, means that there will probably be resources on your Raspberry Pi that won’t be utilized. For example, Raspbian comes with software, packages, and environments that are unnecessary for specialized applications, and only result in wasted memory and CPU usage. You don’t need a graphical desktop environment when you’re only setting up a web-server, after all.

There are many alternative operating systems to the Raspberry Pi. Each of them has their own strengths and weaknesses. While some operating systems are developed with a specialized use case in mind (for example, XBian is a Raspberry Pi flavoured version of XBMC, an operating system that turns your Raspberry Pi into a media-center), others are more general and allow you to decide what you want to do with them. In this article, we will go through the processes of setting up a bare-bones Arch Linux ARM on your Raspberry Pi, a lightweight and flexible operating system which is inspired by the Arch Linux operating system for x86 architectures.
else. Don’t forget this password or else you’ll have trouble setting up further software on the device. Enter “passwd” into the terminal and it should prompt you to enter the new root password.

Unlike Raspbian, which has a graphical tool used to perform first time configuration, on Arch Linux ARM configuration settings are done using a text editor or command line tools. Vi is installed by default and can be used to edit files.

To set your timezone use the command: “timedatectl set-timezone <Region>/<SubRegion>” where <Region> and <SubRegion> should be replaced with the appropriate region and subregion strings. If you’re not sure what these strings are, enter the “timedatectl list-timezones” command to get a list. You can use the arrow keys to browse the list or enter a string preceded by “/” to search and highlight results.

Change your hostname (the name that identifies your Raspberry Pi on the network) by editing the file /etc/hostname using vi. If you don’t know how to use vi, I would recommend you perform a quick google search and get familiar with the text editor.

A restart of your Pi should reflect these changes.

Finally, you will want to initialize your package manager, pacman, and update the system. First, you have to generate a keyring for pacman. In order to ensure a secure key, you must also generate entropy on the system by running commands in a second window. Start a second terminal by hitting ALT + F2 (you can hit ALT + F1 to switch back to the first window, and ALT + F2 to switch to the second).

In one terminal, run the command: “pacman-key —init.” Switch to the other window and generate entropy by entering random commands such as “ls -R /.” Generating a key can take a long time if you don’t actively generate entropy. By running commands in a separate terminal it can take much less time.

Once pacman-key —init has completed, you can run “pacman -Syu” to perform a system update. This command will update the software repositories for pacman, and attempt to update any out of date software that is already installed. This can take some time.

When the system update is completed, your Arch Linux install is ready to be customized for your specific use case. Whether you want to install a web server stack, a graphical desktop interface, or lightweight application, you simply need to install and setup the required software components for your needs. The beauty of Arch Linux is its minimalist simplicity – it will have only what you need and want, and none of the extra stuff that can get in the way.

To get you started, you may want to look into setting up secondary users and common software packages such as:

- sudo
- XFCE / LXDE
- Vim
- GCC

The software packages you decide to install is up to you, and you should do the relevant research to figure out what you need, and how to configure it. In the next installment of this series, we’ll look at how to set up your newly configured Raspberry Pi running Arch Linux ARM as an internal server, including serving web pages and as a file server.
rett and its proven dependability helps keep your support costs down as well. ThreadX licensing model helps keep your BOM low, simply must succeed. Its royalty-free licensing model helps keep your ROM low, and its proven dependability helps keep your support costs down as well. ThreadX repeatedly tops the time-to-market results reported by embedded developers like you. All the while, Express Logic is there to assist you with enhancements, training and responsive telephone support.

Join leading organizations like HP, Apple, Marvel, Philips, NASA, and many more who have chosen ThreadX for use in over one billion of their products — because their products are too important to rely on anything but the best. Rely on ThreadX, when it really counts!

Contact Express Logic to find out more about our ThreadX RTOS, FileX®, file system, NetX™ Dual IPv4/IPv6 TCP/IP stack, USBX™ USB Host/Device/OTG stack, and PEG™ graphics toolkit for embedded GUI development. Also ask about our TraceX® real-time event trace and analysis tool, and StackX™, our stack size analysis tool that makes stack overflows a thing of the past. And if you’re developing safety-critical products for aviation, industrial or medical applications, ask about our new Certification Pack® for ThreadX.

expresslogic
For a free evaluation copy, visit www.rtos.com • 1-888-THREADX
Copyright © 2012, Express Logic, Inc.
ThreadX, FileX, and TraceX are registered trademarks, and NetX, USBX, PEG, StackX, and Certification Pack are trademarks of Express Logic, Inc. All other trademarks are the property of their respective owners.

Intel Galileo Development Board

Galileo is a microcontroller board based on the Intel® Quark™ SoC X1000 Application Processor, a 32-bit Intel Pentium-class system on a chip (datasheet). It’s the first board based on Intel architecture designed to be hardware and software pin-compatible with Arduino shields designed for the Uno R3. Digital pins 0 to 13 (and the adjacent AREF and GND pins), Analog inputs 0 to 5, the power header, ICSP header, and the UART port pins (0 and 1), are all in the same locations as on the Arduino Uno R3. This is also known as the Arduino 1.0 pinout.

Galileo is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to 5V at the VIO pins. This provides support for 5V Uno shields and is the default behavior. By switching the jumper position, the voltage translation can be disabled to provide 3.3V operation at the VIO pins.

SAMA5D31-EK Development Kit

The SAMA5D31-EK is a full-featured evaluation platform for the Atmel SAMA5D31 embedded microprocessor. The kit consists of a main board, a computer module equipped with a SAMA5D31 processor, and a display module. It is supported by the SAMA5D31 Softpack, a free set of bare metal C examples, and is delivered with on-board demos for Linux and Qt-based GUI. The SAMA5D31-EK allows users to extensively evaluate, prototype and create applications that require graphics, audio, mass storage, networking, connectivity and more.

Source files and binaries for the on-board demo are available on Linux4SAM, the entry point to open source solutions for Atmel ARM® core-based embedded microprocessors.
Express Logic provides advanced, high quality software solutions for deeply embedded applications. The company was founded in 1996 by Bill Lamie, who did all of the initial products, including the company’s flagship ThreadX operating system. From there, Express Logic has been broadening its product portfolio as well as increasing its market share, due in part to its unique, royalty-free business model.

We spoke with Bill Lamie, CEO, President, and Founder of the company, about the royalty-free business model, how the embedded industry is changing, and their brand-new product: The GUIX.
With over 1.5 billion devices running it, ThreadX from Express Logic is one of the most popular real-time operating systems.

Your company’s core product is called ThreadX. What was the biggest challenge in bringing these RTOS products to market?

I would say that keeping up with all of the new hardware that comes out is the biggest challenge. Our day to day is getting ports to new devices and making sure we have good coverage to our customers. The next biggest challenge is probably the middleware that goes around it. When we first started Express Logic in 1996, maybe 20% of our customers had networking needs. Now, probably 80% of our customers have networking needs. The reliance on middleware and the capabilities of small processors in deeply embedded situations is increasing, so it creates more work and opportunities for us.

What are some of the typical devices that are running this technology?

The most obvious products running ThreadX are space products. The cameras on Deep Impact, both the imager and the satellite to watch the impact, use ThreadX in the camera system. The Mars Reconnaissance Orbiter uses ThreadX in the camera system as well. Those are two applications pretty well uses ThreadX in the camera system as well. Before that, the sheer number of devices wasn’t so great, but now with cellphones and all these other gadgets, the volumes are too great for a royalty model to actually work in the operating system world.

How is the embedded development space changing?

The biggest thing is the capability of the parts. We were saying before that only 20% of our customers used to have networking needs as opposed to the 80% that do now. The same part 20 years ago might have been an 8-bit part that could do really little. Today, that’s a 32-bit part with lots of memory and even less power than it needed 20 years ago. It’s really just about the capabilities and when you have the capabilities now you just need to find software support for it. There’s a huge increase in what can be done with a very small part, especially things like the Cortex-M series parts. What could be done on those parts used to be a real high-end processor. Even going with the thing we are promoting now, the GUI X product, these small Cortex-M parts are putting LCD controllers on chip, so that opens up a whole new market of displays where displays weren’t even possible previously.

Could you talk a little bit more about the new GUI X product?

That goes back to the Accelerated days, actually. Before co-founding Accelerated Technology, I was working as a consultant for a company building another military device that was a fiber optic network switch. This company was using a commercial operating system at the time, which was around the late 1980s. There weren’t too many options either. These were royalty bearing and had no source code. These operating systems were just terrible to work with, so it got me motivated to do something better.

That’s how Nucleus was born, and it was born to be exactly the opposite of all the available operating systems in every way. It had open systems so you could get all the source codes done. If you had any problems the company had the opportunity to fix it as well as the support people at the vendor site, and there were no royalties because they thought it was just not fair or practical. That’s how the no-royalty thing got started and it turns out that is the only model that works, especially as you start building lots and lots of devices. Before that, the sheer numbers of devices wasn’t so great, but now with cellphones and all these other gadgets, the volumes are too great for a royalty model to actually work in the operating system world.

What are some of your largest customers, and what types of problems are you helping them solve?

Probably the largest customer we have is HP. They are a pretty typical customer in the sense that we help them solve a cost problem—they were previously using a royalty variant operating system, so we made life a lot easier for them. On the hardware side, our operating system is much easier to use, so it really enhances their development and also returns cycles back to their applications because it is much faster than the predecessor that is also royalty variant. Most companies are looking at a good business model and they are looking at improving their development time.

What prompted you to use the royalty-free business model?

That goes back to the Accelerated days, actually. Before co-founding Accelerated Technology, I was working as a consultant for a company building another military device that was a fiber optic network switch. This company was using a commercial operating system at the time, which was around the late 1980s. There weren’t too many options either. These were royalty bearing and had no source code. These operating systems were just terrible to work with, so it got me motivated to do something better.

That’s how Nucleus was born, and it was born to be exactly the opposite of all the available operating systems in every way. It had open systems so you could get all the source codes done. If you had any problems the company had the opportunity to fix it as well as the support people at the vendor site, and there were no royalties because they thought it was just not fair or practical. That’s how the no-royalty thing got started and it turns out that is the only model that works, especially as you start building lots and lots of devices. Before that, the sheer numbers of devices wasn’t so great, but now with cellphones and all these other gadgets, the volumes are too great for a royalty model to actually work in the operating system world.

How does this product compare to Android, iOS, or other operating systems on the market?

That’s a great question. On the real high-end graphics, we’re not really looking at that market—those things take tons of memory and lots of CPU power. We’re looking at filling the need of 2D graphics, typically on the smaller parts. We’ll move into the higher parts too where the core processor also has to do a lot of real-time processing. There will be a little bit of overlap between what the high-end graphics can do and what GUI X will do, but most of our business will be on the Cortex-M 2D type of graphic displays.

We’ll also have decent play in the certification space too. Some of the medical apps that have displays where it’s all self-contained and running on one processor—the other higher end graphics packages are not going to be able to certified, so GUI X will at some point be certified as well. That’ll be another area where we can meet the needs of high demand applications.

"We’re really excited about the potential opportunities—we have people beating down our door now and they don’t even know that we are about to introduce this, so it already looks like a winner for us."
How do you stay competitive in this market? Why do they choose Express Logic over other companies?

I guess it’s a couple of things. The first being that we make software that we would want to use ourselves—that’s been my philosophy since I was started out working on operating systems. It’s easy; if I can make something that I would want to use, there’s a good chance a lot of other people will want to use it too. The other thing is that we listen to customers. A lot of Nucleus and a lot of Thread X is developed by feedback. A lot of problems were resolved over time based on what the customer’s needs were.

How would you describe the culture at Express Logic?

We’re an engineering-heavy company, so over half of the people here are engineers. We have a typical engineering type of culture, where it’s not a rigid 7 in the morning schedule where people are wearing suits and ties—it’s a casual engineering-type company. The only other thing I’d say is that we’re really good about making sure that everyone has a voice and that their voice is listened to. That’s the only thing that we try to do that might be different than some companies.

Is it a challenge to find competent software engineers to work on your complex products?

There are two parts to our company that are tough—the R&D part in the beginning, and the support part. The rest of our business is pretty easy, really. The other pieces are tough because you can’t just have kids that don’t have any experience doing R&D or don’t have any experience doing support. We have full, high-experienced engineers on both sides of that equation and that is a big challenge.

The real benefit of writing a user guide first is that it allows you to really think about what the product is. If you have to explain it to someone and as you are explaining it, it gets convoluted, you start realizing that it’s not really the way to go.
GUIX™, the new graphical user interface (GUI) development framework from Express Logic, is a small-footprint, low-overhead runtime engine and development tool featuring automatic code generation for embedded systems capable of graphical display. GUIX simplifies GUI development and targets the ARM 32-bit MCU and MPU architectures, including Cortex-M3, M4, A8, and A9, in medical devices, consumer electronics, and industrial control equipment.

GUIX includes a full-featured runtime graphical library fully integrated with ThreadX, Express Logic’s popular RTOS that is deployed in over 1.5 billion devices. Like ThreadX, GUIX achieves high performance in small footprint, deeply embedded applications. Designed to meet the growing need for dynamic user interfaces with limited hardware resources, GUIX uses the same optimized design and coding methods of ThreadX and the other Express Logic embedded system software products, is distributed with full ANSI C source code, and has no run-time royalties.
Thanks to the explosion of opportunities that started in smartphones, GUIs have become commonplace in medical, consumer, and industrial applications, prompting the need for advanced tools to simplify their development. Most embedded system programmers are not LCD specialists and do not want to program these displays at the lowest level, which involves constructing individual graphical shapes and objects (“widgets”). Instead, to speed time to market, most developers use a library of routines that manage the GUI details. By describing the widgets at a high level, such libraries help developers to enhance programming productivity and avoid many errors.

GUIX offers developers an advanced UI framework and rich library of unique widgets tailored to help them construct whatever GUI they envision. Programmers can call GUIX functions from their C application programs, and GUIX performs all the necessary drawing functions to produce a clear, interactive GUI on LCD screens of various sizes and resolutions.

GUIX Studio
GUI designers can create GUIs using GUIX Studio™, the companion PC-based application that enables WYSIWYG rapid prototyping of GUI designs. With GUIX Studio, the designer can select, drag-and-drop, and resize images, backgrounds, widgets, and other elements of a powerful GUI without having to write a single line of code. GUIX Studio generates the code necessary to implement the exact GUI design constructed on the PC. The generated code can be dropped into the application and executed on the target system.

A complete GUIX UI application can be executed on a PC desktop within the GUIX Studio environment, enabling developers or designers to quickly and easily generate and demonstrate UI concepts and test screen flows as well as observe screen transitions and animations. When completed, the design can be exported to target-ready C data structures that are ready to be compiled and linked with the GUIX and ThreadX libraries as part of a project.

Developers can produce pre-rendered fonts for their applications using integrated font generation in GUIX Studio. Fonts can be generated in monochrome or antialiased formats that are compressed to save space on the target. Fonts can include any set of characters, including Unicode characters for multilingual applications. Importing graphics from PNG, JPG, or BMP files and converting them to compressed GUIX pixelmaps for the target system is another integrated feature of GUIX Studio, and many of the GUIX widget types are designed to incorporate developers’ proprietary graphics for a custom look and feel. In addition, developers can customize default colors and drawing styles used by the stock GUIX widgets, allowing them to customize the appearance of GUIX very easily. GUIX Studio also generates and maintains application strings for any number of target languages.

“Thanks to the explosion of opportunities that started in smartphones, GUIs have become commonplace in medical, consumer, and industrial applications, prompting the need for advanced tools to simplify their development. Most embedded system programmers are not LCD specialists and do not want to program these displays at the lowest level, which involves constructing individual graphical shapes and objects (“widgets”). Instead, to speed time to market, most developers use a library of routines that manage the GUI details. By describing the widgets at a high level, such libraries help developers to enhance programming productivity and avoid many errors. GUIX offers developers an advanced UI framework and rich library of unique widgets tailored to help them construct whatever GUI they envision.”

GUIX Studio offers developers an advanced UI framework and rich library of unique widgets tailored to help them construct whatever GUI they envision.
Embedded Modules

Speed Development & Decrease Costs in Medical Equipment Designs

by Wolfgang Heinz-Fischer
Head of Marketing & PR, TQ-Group

Developments and trends on the electronics market are generally quickly reflected on the electrical medical equipment market. Cost pressure and time constraints are also heightened here, forcing companies to focus on their core competencies.

This does not usually include processor cores, which are generally “common” technology which can be purchased easier and cheaper. In the past, there were many arguments in favor of using an embedded module, but today it is becoming increasingly more important to use the same module in different devices, as was the case previously for processors. In real applications, this platform concept is becoming more important than the ability to scale a system. Which factors play a role when selecting a platform, and what does the market offer?

Electrical medical equipment has a wide range of requirements for the processor used. This starts with computing power and includes the necessary interfaces. Almost every device today has a display, and therefore needs graphics, communicated via LAN or WLAN, and the interfaces for a series of sensors. Depending on whether it is a mobile battery-operated device or a stationary device, power dissipation is a crucial criterion. As the focus of costs and development time is today all about software, it is important to consistently find the same software environment. It is thus not always easy to find the right processor for a number of devices, which is why it is sometimes necessary to select several platforms.

There are many reasons to employ a platform solution. Technology is becoming increasingly complex, increasing development times and the need for development resources. And this is precisely what conflicts with the market requirements. In some cases, the technical requirements for implementation are simply lacking. The use of a module with complex technology, e.g. a processor with 1.2 GHz clocking and DDR3 memory, requires a layer structure with Microvia and at least 10 or 12 layers.

If the processor were to be integrated, the entire application board would have to have this layered structure. In cases of normal use, the additional costs often incurred for the module’s plug are calculated based on the layers saved for the application board.

The average price of a PCB with dimensions of 175 mm x 120 mm and 12 layers is around 20 to 25 Euros ($26-$33) for medium-sized quantities. The layered structure can be reduced to 8 or less layers by using one module. For 8 layers, the PCB price is around 15 to 20 Euros ($20-$26) for the same quantities.

A module’s counter-plug set is around 5 or 6 Euros ($6.50 to $8) in higher quantities, meaning no additional material costs are incurred by using a module. Along with this, all the known arguments in favor of using a module of course also apply: Time is clearly saved for users during development due to the fact that the processor module already exists as ready-made hardware.

As the application board has a much simpler structure, it can be developed quicker and produced later on. Software development can begin immediately, as the reference platform is already available, usually with the relevant BSPs and drivers. Initial performance tests can thus commence very early on.

One clear advantage for users is the fact that the “time-to-market” is significantly shortened. But risk minimization is also crucial. And risk means time and money. As the application board’s design is much simpler, the risk of any redesign is greatly reduced.

During the product’s lifecycle, a redesign is generally only necessary for the embedded module. Less risk and prompt completion of a development can significantly contribute to a product’s market success.

Attempts have previously been made to always use the same processor wherever possible in order to utilize the infrastructure investments made, e.g. for development tools, software drivers, BSPs, test tools etc. including for new developments, but the vast number of processors and relevant embedded modules drew people to always select the optimum processor the each development and device.

As the software portion of an overall development and in a device’s added value is constantly increasing, further considerations have deemed to more beneficial to focus on a processor or platform in the form of an embedded module. So if the right processor has been found, we have to also find the right module to use as a platform.

The first question is, of course, whether or not the module provides the necessary output and interfaces, not just for current use but also for further planned applications for which the processor would be suitable. If the module does not possess all the processor’s functions, there is a risk that, although the processor may be suitable for the application, it may not provide the necessary signals. The same applies for board size. The smaller a module, the more likely it is to mechanically fit into all devices for which the processor is suitable.
When it comes to electrical medical devices, long-term availability is also important, since they must be supplied over a long period of time. As already mentioned, software is playing an increasingly important role, so the relevant drivers should be provided for the platform(s). In doing so, the application software should be able to run on as many platforms as possible. Choosing the right module which will be successful over the long-term is thus always a question of whether or not the module restricts the processor in any way.

Thus in most cases the use of modules constitutes a significant advantage for the developer. They allow him to fully concentrate on his core competence thus saving design resources and enabling several devices to be developed within a short space of time. The probability of developing a device within the prescribed time limit also increases since the device’s complex parts are already available as a tested and ready-to-use solution.

And time-to-market, i.e. the possibility of being the first on the market, may provide a decisive competitive advantage. Even lifecycle management becomes simpler since the probability of a redesign less predominantly on the module side, and thus is not a problem for the developer but must be carried out by the module provider. Moreover, the use of a module is also advantageous with regard to costs.

The development and production costs are considerably lower and easier to calculate. The entire investment for a product development may be up to 200,000 Euros lower than the cost of a fully integrated development. And these lower investment costs also imply lower interest costs since the development, as a rule, must be pre-financed.

The strong points of a platform concept become evident in any case upon deployment in a second device. Here, the savings are even higher since the developer is able to draw on many pre-existing elements.

An example of this is the optimum use of investments that have already been made on one occasion. In addition, the higher number of units of the module deployed arising from several projects also offers the buyer the possibility of benefitting from quantity discounts. Thus as a rule, the successful route to the fast development of electro-medical devices passes through a processor platform.

All TQ modules consistently provide these requirements as an optimum platform module: compact design, all processor signals available, robust, suitable for industrial applications, and available for at least 10 years.

Save development time and money with a product designed with an embedded module from TQ-Systems.

TQ-Systems has decades of experience in developing and manufacturing System on Modules (SOMs) and complete devices for medical technology companies – ranging from pocket-format end-user devices to large equipment for clinics and hospitals. We provide solutions that meet the full spectrum of your requirements, including critical care, home care, general medicine and dentistry.

Our products are medical product standard DIN EN ISO 13485 and software development process EN 62304 certified.

With offices to serve you in California and Massachusetts, we can provide you with sales, technical assistance, product distribution and support.

To order a Starter Kit or for more information: www.TQ-USA.com

TQ-USA is the brand for a module product line represented in North America by Convergence Promotions, LLC.