

STM32 Journal

Volume 1, Issue 2



In this Issue:

- » Bringing 32-bit Performance to 8- and 16-bit Applications
- » Developing High-Quality Audio for Consumer Electronics Applications
- » Bringing Floating-Point Performance and Precision to Embedded Applications
- » Achieving Ultra-Low-Power Efficiency for Portable Medical Devices
- » Accelerating Time-to-Market Through the ARM Cortex-M Ecosystem
- » Introducing a Graphical User Interface to Your Embedded Application



Table of Contents

- 2 Editorial
- 3 Bringing 32-bit Performance to 8- and 16-bit Applications
- 11 Developing High-Quality Audio for Consumer Electronics Applications
- 19 Bringing Floating-Point Performance and Precision to Embedded Applications
- 26 Achieving Ultra-Low-Power Efficiency for Portable Medical Devices
- 34 Accelerating Time-to-Market Through the ARM Cortex-M Ecosystem
- 42 Introducing a Graphical User Interface to Your Embedded Application

We Are Not Alone

By Nicholas Cravotta, Technical Editor

When I got my first paycheck as an engineer nearly three decades ago, coding and layout weren't exactly social activities. While there was a certain amount of team collaboration to decide what I would work on, the majority of what I did was by myself. When I decided to shed the ten pounds I had gained as a freshman, it was a similar story. I never found someone willing to go consistently to the gym with me, so I pressed those weights alone as well.

It's quite a different world today. Take a look at the Nike+ FuelBand on the cover of this issue's STM32 Journal. Worn on your wrist, it records your every activity, not just when you're on the treadmill.

What makes the FuelBand such a ground-breaking product is how it brings people together. It doesn't matter whether you work out at 2am or are in a strange city on travel, with this next-generation exercise monitor, you are never alone. Connected to your phone via Bluetooth, you can be in touch


with exercise buddies all around the world through the Nike+ online community.

The Nike+ FuelBand is quite a feat of engineering. To differentiate between simple gestures and active motions requires complex signal processing capabilities. The device must also be constantly on since even you don't know when you might jump into action. 120 LEDs comprise the display and "Fuel" indicator, and the device can operate for up to four full days without recharging. It also weighs less than one ounce, including the batteries. Now that's an efficient design.

At the heart of the FuelBand is ST's ultra-low power STM32 L1 microcontroller. In addition to providing the 32-bit performance and processing capacity required for advanced signal processing, the STM32 architecture offers the real-time responsiveness, power efficiency, and highly integrated peripherals and memory required for even the most demanding embedded applications.

With innovations like FuelBand and Nike+ technology, Nike has leveraged social networking to change the way we live together. Exercise, as a result, is no longer a solo endeavor.

Neither, it turns out, is engineering. The network supporting the STM32 architecture enables a whole new level of collaboration. Design tools from companies like Keil, IAR Systems, and Micrium are like having a team of experts sitting right next you. Need to extend a design by adding audio or a capacitive touch GUI-based interface? Just call upon partners like DSP Concepts and GeeseWare. And with the STM32 architecture based on the ARM Cortex-M0, M3, and M4 cores, you have access to a global ecosystem second to none. You can even ask questions of your fellow engineers at 2am or share your own hard-won experience through forums, blogs, and tweets.

It truly is a different world we live, play, exercise, and work in. 

Bringing 32-bit Performance to 8- and 16-bit Applications

By Reinhard Keil, Director of MCU Tools, ARM Germany GmbH
Shawn Prestridge, Senior Field Applications Engineer, IAR Systems
Sean Newton, Field Applications Engineering Manager, STMicroelectronics

Today's embedded applications are being called upon to provide an increasing number of capabilities. More and more devices need to be connected, require greater precision, must offer a graphics-based interface with touch capabilities, utilize sophisticated signal processing, and support multimedia playback.

In the past, developers were compelled by cost constraints to base their designs on 8- and 16-bit architectures that limited performance. Now, with the availability of next-generation MCUs like the STM32 F0 that provide 32-bit performance at 8-bit budget pricing, OEMs can bring substantial value to end-users without having to compromise functionality. In addition, powerful development tools like Keil's MDK-ARM and IAR Embedded Workbench enable developers new to 32-bit programming to immediately

exploit the full capabilities of the STM32 F0 architecture.

The 32-bit Advantage

There are several ways in which the STM32 F0 lowers product cost compared to 8- and 16-bit-based designs. Specifically, because these MCUs tend to be based on legacy architectures, they have many limitations that slow development by forcing designers to work around the architecture, so to speak. For example, to complete a 16 x 16 multiplication for a processing algorithm, a 16-bit CPU requires four multiplies and several additions, depending upon the implementation. An 8-bit CPU would require significantly more cycles. With the STM32 F0, this takes a single instruction.

The result is code that makes better utilization of MCU resources, leading to faster operation, more performance per

MHz, higher code density, and greater power efficiency. Since each instruction does more per clock cycle, applications can be written using less code. In addition to accelerating development, shorter code is easier to debug as well. Together, all of these benefits lead to lower system cost.

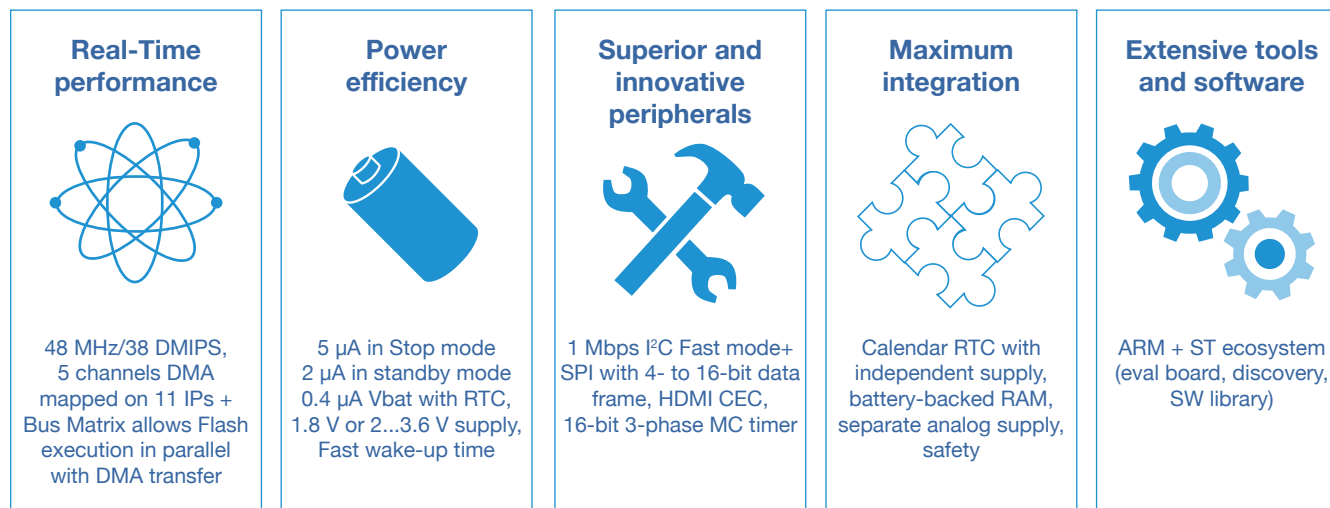
Cost, however, is only one of the numerous advantages the STM32 F0 has over 8- and 16-bit architectures. The STM32 F0 is a full embedded MCU built using the same STM32 DNA that the rest of the STM32 family has, including excellent real-time performance, DMA, high-resolution ADC and DAC peripherals, motor control timers, and connectivity interfaces. These integrated capabilities bring tremendous efficiency to cost-sensitive designs in a way that limited 8- and 16-bit MCU architectures cannot (see Figure 1).

For example, the availability of a 32-bit bus not only speeds data transfers and increases computing performance, it improves system reliability. Consider the challenge of reading a 12-bit DAC using an 8-bit bus where the CPU has to read the DAC twice to capture the entire sample. If an interrupt occurs between these reads, the DAC data may be overwritten by the next sample before the interrupt is completed and the second read can be executed. To prevent this, developers have to manually disable interrupts for every such "atomic" operation in an application. If even one instance is missed, this creates a potential for an intermittent error that will be extremely difficult to resolve.

DMA: Moving Data Efficiently

The STM32 F0 is a modern architecture integrating the

STM32 F0 Series: Key Features



STM32 F0 series

L1, F0, F1, F2, F4 series: seamless migration amongst 300 pin-to-pin compatible part #s

Figure 1 The STM32 F0 is a full embedded MCU built using the same STM32 DNA that the rest of the STM32 family has and offers tremendous efficiency to cost-sensitive designs in a way that limited 8- and 16-bit MCU architectures cannot.

latest in processing, power, and debugging technology. For example, multiple low power modes extend greater control over power consumption to achieve longer operating life for battery-operated and portable devices. In addition, the STM32 F0 offers advanced features, including full Direct Memory Access (DMA) and the ability to shut down the ADC between samples to further increase performance while lowering power consumption.

In general, 8-bit MCUs don't have the powerful peripherals that higher performance MCUs tend to have. For example, DMA has become an essential peripheral for applications that need to move a great deal of data, whether as part of a processing algorithm, receiving data from an interface, playing back audio, or transferring graphics to the display. In a traditional 8-bit architecture, each word of data has to be moved by the CPU. In addition,

pointers need to be updated and a loop managed. Thus, every 8-bits of data takes several cycles of CPU time to move.

With the DMA in the STM32 F0, an entire block of data can be moved without involving the CPU. After the program configures the transfer, the DMA manages moving the data in the background. In fact, the CPU can drop into a low power sleep mode while it waits for the transfer to complete. As

a result, data transfers do not consume unnecessary CPU cycles and require less power to complete than for 8- and 16-bit architectures.

The availability of a DMA controller can also greatly simplify and accelerate product development. Consider reading data off of a high-speed data interface such as I2C. Because of the load on the CPU, 8-bit developers have to work around the MCU's architecture, using many interrupts to utilize the time between data reads. With the STM32 F0, the CPU operates independently of the interface, allowing developers to program the CPU for other tasks without having to worry about missing an interrupt or losing data.

Because the STM32 architecture uses an internal bus matrix, the DMA can be used in conjunction with each of the different on-chip memories as well as many of the peripherals. For example, the DMA can be configured to sample the ADC regularly over a period of time: a timer triggers the DMA to read the ADC and store the result in memory without involving the CPU. Once the operation is complete, the ADC shuts down until the



life.augmented



IAR Embedded Workbench®



IAR Embedded Workbench® supports ARM® processors with features like Stack usage analysis, Eclipse/Subversion integration and advancements in power debugging technology.

www.iar.com

next sample time. In fact, the bus matrix combined with a 5-channel DMA enables the STM32 F0 to support execution of code from Flash in parallel with other memory-memory, peripheral-memory, or memory-peripheral DMA transfers.

There are many tools to assist developers in taking advantage of the STM32 F0's DMA capabilities without requiring them to become DMA experts. The ARM DSP Cortex Microcontroller Software Interface Standard (CMSIS) library, for example, provides signal processing functionality that has been optimized for the STM32 F0 and takes full advantage of the DMA.

An intelligent compiler can also help developers exploit DMA technology to its fullest advantage. IAR Embedded Workbench, for example, offers a feature that will automatically rearrange program data to maximize the use of the DMA. This enables developers to achieve high efficiency without having to put much forethought into how to layout the data space. The compiler achieves this by analyzing how data is used by the application.

Consider a program that copies two different data structures using DMA. Each copy operation requires a separate DMA operation. However, after the compiler collocates the data structures in memory, they can be copied with a single DMA transfer.

Note that each MCU may use the DMA in a slightly different manner. Keil's MDK-ARM, for example, abstracts how the DMA is used from the application through an API that prevents code from being tied to a particular processor. This enables developers to migrate applications to other STM32 devices and know that code utilizing the DMA will still perform optimally.

Writing 32-bit Code

Moving from 8-bit to 32-bit assembly is not trivial, given the vastly different instructions 32-bit architectures offer; i.e., single-instruction, multiple data (SIMD) instructions work on multiple data to vastly accelerate processing. Even moving between 16-bit architectures is challenging given that the peripherals can differ and impact how application code is written.

Benchmark Application		8-bit Math		8-bit Matrix		8-bit Switch		16-bit Math		16-bit Matrix		16-bit Switch		32-bit Math		Floating-Point Math		Matrix Multiplication		FIR filter	
16-bit	MSP430	178		86		198		126		90		198		222		1102		136		980	
	dsPIC	236		420		424		224		552		424		424		2020		464		2256	
	PIC24	236		420		416		224		552		416		424		2020		464		2256	
	H8/300H	344		412		444		352		482		478		574		1104		482		1392	
	MaxQ20	230		252		192		204		328		184		288		1172		398		1478	
	HCS12	83		188		162		76		262		174		323		2082		219		1917	
	ATxmega64A1	118		398		338		174		490		350		300		1080		584		1362	
	ARM7TDMI	636		392		452		636		396		452		620		1832		428		1528	
8-bit	8051	233		398		305		452		504		493		909		2190		536		2056	
	PIC18F242	170		324		208		286		692		282		542		1400		676		2006	
	ATmega8	134		354		350		198		434		382		342		1088		490		1358	

STM32 F0 Compiler Option		Speed		Size		Speed		Size		Speed		Size		Speed		Size		Speed		Size		Speed		Size	
Code Size		110	94	68	52	98	120	114	106	68	52	98	120	112	108	640	636	268	84	550	550				

Figure 2 The STM32 F0 delivers substantial code size reduction when comparing to other 8/16 architectures. This grid show the code size in bytes for various benchmark applications. Source: Benchmark applications and results for 8/16-bit: TI MSP430 Competitive Benchmarking (www.ti.com/lit/an/slaa205c/slaa205c.pdf). STM32 F0 code generated with MDK V4.23, MicroLib, and compiler optimization for execution speed or code size.

The STM32 F0 architecture facilitates a smooth migration to 32-bits. The ability to develop in embedded C reduces the learning curve of moving to a new architecture. In many cases, engineers are already familiar with the ARM Cortex-M

architecture. Developers can further ease migration by using a tool chain they are already familiar with, such as IAR Embedded Workbench and Keil's MDK-ARM. Finally, developing for the STM32 F0 is simplified through the use of the ARM

CMSIS libraries that abstract much of the underlying hardware from the application.

Moving to the STM32 F0 will result in a substantial reduction in code size because of the density possible with 32-bit

instructions, on the order of 30% (see Figure 2). With its 32-bit address space, the STM32 F0 also eliminates addressing and paging limitations that complicate memory management in 8-bit designs. For example, data

sets can be larger than a single page and there are no longer “far” addressing penalties. The use of object-oriented constructs, as is common with modern programming and modeling tools, can also be implemented without disruptive fragmentation.

Without question, the best compiler is the human brain. Given enough time, a person can create a highly optimized program that no compiler can beat. Programming in assembly can also be more efficient than a C version of the same program. Time, however, is one of the resources of which developers don’t have a surplus. In addition, hand-written code can be extremely fragile; if the product specs change in a material way, many of a programmer’s optimizations will need to be completely reevaluated.

The reality is that Keil’s MDK-ARM and IAR Embedded Workbench are smart enough to make excellent coding choices that might take a person weeks to evaluate. For example, how data is laid out impacts performance. There’s also the challenge of balancing optimization techniques like

loop unrolling to memory footprint. A compiler can make these decisions for an entire program in just minutes. Each of these tools offers numerous optimization options it can perform automatically for the STM32 F0 architecture that are significantly different than those typical with 8- and 16-bit MCUs. These options include data-flow optimizations such as common sub-expression elimination and loop optimizations such as loop combining and distribution. They also include advanced techniques like branch speculation and executing code out of sequence.

These development tools for the STM32 F0 give excellent results. Compiler efficiency compared to human coding has been estimated at 97%. Put another way, the cost of achieving that last 3% is on the order of weeks to months of development time. In addition, if a major design change is required, the compiler can complete a new set of optimizations with just a simple recompile.

As a modern architecture, the STM32 F0 is supported by similarly modern tools that

utilize the latest advancements in compiler, debugger, and middleware technology to reduce development time and effort considerably. Being based on the Cortex-M architecture, the STM32 F0 is backed by a larger ecosystem of tools and production-ready software than any other MCU architecture on the market. In addition, for many applications where the code base is small, the tools may be effectively free. For example, both IAR Embedded Workbench and Keil’s MDK-ARM are free when used for programs under 32 KB, thus enabling 32-bit design with a low initial investment.

Advanced Debugging

While the ability to design demanding applications quickly is important, developers need debugging capabilities that can abstract the complexity of applications while still providing full visibility and control during run-time operation. In addition, many embedded markets, including medical and industrial, require that application software be certified as well.

The integrated debug capabilities of the STM32 F0 provide many advanced

capabilities that offer a superior debug experience compared to old-fashioned 8- and 16-bit architectures. For example, the STM32 F0 architecture features ARM’s Coresight technology to help developers analyze, optimize, and verify program execution with minimal effort and cost.

Coresight represents the latest in advanced debugging technology. Traditional MCUs offer only limited run/stop debug capabilities. To achieve greater visibility, an in-circuit emulator on the order of \$1000s may be required, and a different pod will be required for each MCU in use. A few of the benefits Coresight provides which other MCU architectures do not include on-the-fly read/write access and trace capabilities at the instruction, data, and application level. As implemented in the STM32 F0, Coresight also supports up to 4 hardware breakpoints and 2 watchpoints without requiring the use of intrusive monitoring techniques that can skew performance.

Developers also have a choice of many low-cost debug adapters for the STM32 F0. For example, the STLink in-circuit



STM32[®] F0

**STM32[®] DNA at a
budget price with
superior analog and
control peripherals**



STM32[®] Releasing your **creativity**

For further information, visit www.st.com/stm32f0

debugger and programmer, which links the STM32 F0 target board to a PC via USB, is \$25. For more advanced debugging, IAR Systems has the I-Jet debugger while Keil offers developers its ULINK2 and ULINKpro debuggers.

These debuggers offer powerful capabilities that are often not available for 8- and 16-bit designs. Keil MDK-ARM tools, for example, enable comprehensive code coverage, execution profiling, and performance analysis to ensure maximum performance efficiency. With the I-jet debugger, IAR Systems is able to offer non-intrusive power consumption monitoring at the board- and chip-level. Such “power debugging” enables developers to uncover opportunities to utilize and tune hardware to achieve the highest power efficiency.

STM32 F0 Features

STM32 F0 MCUs have been designed with real-time operating system (RTOS) and kernel support in mind to enable much tighter integration with RTOSes like Keil’s royalty-free RTX. In a typical 8- or 16-bit MCU, for example, the

RTOS and application share the stack, and complex nesting problems can arise that overflow the stack and crash the system. The only way to avoid such issues is to overprovision the stack. The STM32 F0, in contrast, has two stacks: one for the application and one for the RTOS. This prevents applications from compromising RTOS integrity. In addition, RAM overhead is much lower.

Other companies basing MCUs on the Cortex-M0 architecture integrate only the minimum capabilities an MCU requires. ST is the only company to offer Cortex-M0-based MCUs with:

» **Easy Communication:** Using the integrated DMA controller, the STM32 F0 can support continuous I²C at a rate of 1 Mbps without bogging down the CPU. This data rate isn’t possible to achieve on an 8- or 16-bit MCU that does not support DMA.

» **Advanced Digital and Analog Capabilities:** The STM32 F0 integrates a wide range of IP to facilitate the design of sensing and control systems. For example, advanced timers enable the accurate output

of complex AC waveforms. On-chip comparators simplify the design of sensors. The 12-bit, multi-channel ADC operating at up to 1 MSample/s allows for fast and precise data acquisition, as well as improves system responsiveness to external events. Advanced timing control is enabled using the 32-bit and 16-bit PWM timers with 17 capture/compare I/O mapped onto up to 28 pins.

» **Safety Ready:** With shrinking process technologies and larger memories combined with frequently changing data, bit errors from cosmic rays can occur. For systems that must meet stringent safety compliance standards, the STM32 F0 performs real-time, hardware-based RAM parity checking and 16-bit CRC verification for Flash to ensure the integrity of memory. RAM checks are performed automatically whenever memory is accessed. Flash verification is self-managed, enabling developers to confirm program integrity upon startup and when updating firmware to verify that no bits have been flipped since they were written.

Double Sourcing

In recent years, the industry has seen shortages when devices are manufactured in a single location. To ensure its customers will always have uninterrupted access to product, ST employs a double sourcing strategy in which all STM32 devices are manufactured in at least two fabs in different parts of the world. This prevents product supply from being vulnerable to environmental factors that shut down a particular production fab. It also enables ST to meet any unanticipated rise in demand more easily by shifting production among multiple fabs.

» **Reliability:** The STM32 F0 integrates two watchdog timers, one of which is a windowed watchdog timer. These timers, which can operate in low power modes as well, provide a higher level

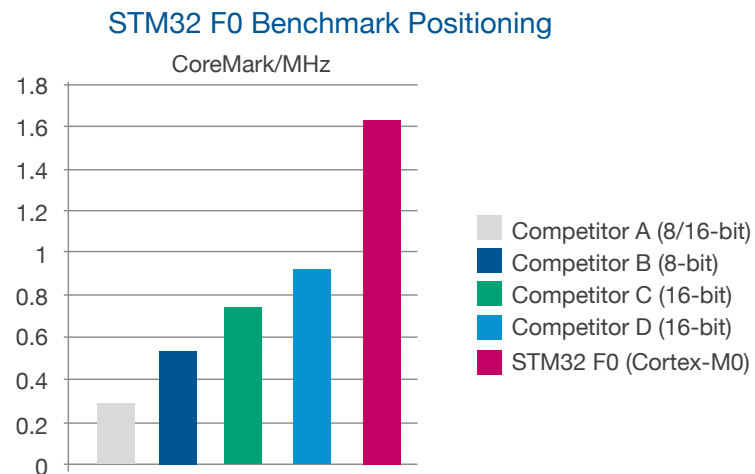


Figure 3 The STM32 F0 provides an optimal balance of cost, performance, and peripherals for embedded applications.

of reliability not available in most 8- and 16-bit MCUs. A Clock Security System (CSS) enables systems to switch to internal RC-based clocking in case of external clock failure to ensure systems can shut down gracefully rather than catastrophically.

» **Optimized Communications:** The STM32 F0 supports the HDMI Consumer Electronics Communication (CEC) protocol. Important for devices targeted for consumer markets, this peripheral enables devices to have smart control over multiple HDMI lines. For devices needing

remote control capabilities, ST provides a full infrared firmware library.

- » **Memory:** Memory capacity ranges from 16 KB to 128 KB Flash
- » **1.8V Ready:** The STM32 F0 can interface directly to 1.8 to 3.6 V-based devices. This eliminates the need for additional conditioning circuitry 8- and 16-bit MCUs require.
- » **Capacitive Touch Sensing:** To add touch to 8- and 16-bit MCU-based designs, a second processor is typically required. With the STM32

F0, developers can easily introduce capacitive touch sensing to applications, with up to 18 keys and slider/wheel configurations, all with a single chip. In addition, touch sensing can be implemented with zero CPU loading when using the charge transfer method.

Overall, the STM32 F0 provides an optimal balance of cost, performance, and peripherals for embedded applications (see Figure 3). Rather than tie developers to a proprietary architecture with limited tools and support, ST offers the industry's widest Cortex-M portfolio with more than 300 compatible devices across the entire STM32 family.

With code-, pin-, and peripheral-compatibility across the STM32 family, developers can leverage Cortex-M0-based designs to M3- and M4-based MCUs with unparalleled flexibility. For example, applications designed using the STM32 F0 are easily migrated to the STM32 F2 and STM32 F4. With Keil's MDK-ARM and IAR Embedded Workbench, developers just need to change the MCU selection and the compiler handles all of

the details by recompiling the code. This enables developers to easily migrate to an MCU with more performance, memory, and peripherals without rewriting the application. As a result, developers can leverage the same application and tool chain across an entire product line and a variety of MCUs.

Similarly, developers have the option of designing code on the STM32 F2 or F4 with the intention of later downsizing to the STM32 F0. This enables design to take place on a platform with the highest performance and memory to accelerate proof-of-concept design. Once the design has settled, developers can optimize it for the STM32 F0.

With the STM32 F0, ST offers a compelling alternative to 8- and 16-bit devices. For the same price, developers get more performance, higher resolution peripherals, better tools, wider support, accelerated development, and faster time-to-market. To explore how the new STM32 F0 can bring the benefits of 32-bit technology to your designs, the STM32 F0 Discovery Kit is available now for less than \$10. 🦋

Developing High-Quality Audio for Consumer Electronics Applications

By Paul Beckmann, CEO/CTO, DSP Concepts
Dragos Davidescu, Chief System Architect, STMicroelectronics
John Knab, Application Engineer, STMicroelectronics

With the falling cost of high-performance MCUs, manufacturers are considering adding digital audio functionality to more and more consumer devices and other embedded applications. Their goal is to support the wide variety of media sources users want to access such as an iPhone, Internet radio, external USB devices, and SD cards.

Achieving high-quality sound output, however, is non-trivial. Sound quality depends greatly upon the final system configuration, making it difficult to design even when prototype hardware is available. In addition, implementing real-time digital signal processing algorithms introduces a whole new set of concerns for developers used to MCU-based design. These include implementing advanced filters and processing algorithms,

handling fixed-point issues, using DSP-like instructions, and optimizing complex algorithms for speed, MIPS, memory, and power.

In this article, we'll show how developers can leverage MCU-based digital signal processing (DSP) and floating-point unit (FPU) capabilities to enable real-time audio playback, implement enhanced algorithms, convert between multiple clock domains, manage high-speed communications without impacting audio quality, optimize designs to balance quality and cost, and manage other system tasks such as a graphical user interface, all with a single microcontroller.

Consumer Audio

Traditionally, introducing audio to an embedded application requires digital signal-processing capabilities beyond the capabilities

of most MCUs. Even a “simple” product like an iPod speaker dock requires a significant number of advanced audio algorithms to achieve full performance:

Spatial enhancement: In an iPod docking station, the speakers may be only 12-18 inches apart. To create a more spacious, rich sound, spatial enhancement is required to compensate for the close proximity of the speakers.

Multi-channel audio: For systems supporting more than two speakers, the stereo input signal requires processing to create the additional audio channels.

Equalization: Speakers need to be equalized to achieve better sound quality. If the speakers in use change, the equalization needs to be adjusted as well. Developers can employ a variety of equalization methods,

including graphic and parametric equalization. For higher-end applications, developers may even want to design their own equalization algorithms using a tool like MATLAB.

Peak limiting: Speakers exhibit nonlinearity at louder sound levels. By applying a time-varying gain and carefully controlling the peak levels, the system can play louder with a minimum amount of distortion.

Boost: When listening to music at low volume levels, much detail, and therefore depth, can be lost. Boosting of the bass and certain other frequencies at low volume levels using loudness compensation or perceptual volume-control techniques can significantly improve perceived sound quality.

Level matching: Level matching eliminates the need for users to adjust the volume for each song

when shuffling through a large library of albums.

Digital audio has commonly been implemented in consumer electronics and embedded applications using a second processor dedicated to this task. To meet market cost pressures, however, manufacturers need to be able to process audio on the host CPU.

In general, it is easier to implement audio on an MCU than it is to implement real-time responsiveness and connectivity on a DSP. DSPs, while excellent at processing audio, don't have the peripherals or interrupt responsiveness required for real-time systems. DSP architectures are also typically designed for high-end signal processing and massive parallelism that exceeds the requirements of the typical consumer application. In addition, DSPs are not designed to support communication interfaces like USB, SD cards, or Wi-Fi, so a DSP-based docking station would still require a second processor to handle connectivity.

With the introduction of DSP capabilities to MCU instruction sets, MCUs now have the advanced math processing

capabilities required to handle not only basic audio processing but the advanced algorithms required to improve quality as well. In addition, rather than requiring developers to hand-code assembly as is typical for DSP-based designs, MCUs offer ease-of-use and faster time-to-market through C programming and application libraries. MCUs are also specifically

architected to provide short and deterministic interrupt latency as well as ultra low-power operation for battery-powered applications.

The STM32 MCU + Audio Architecture

The STM32 architecture from ST has been designed to bring 32-bit MCU capabilities to a wide range of consumer audio applications, including

multimedia speakers, docking stations, and headphones. The STM32 F4, based on an ARM Cortex-M4 core operating at up to 168 MHz, also integrates capabilities such as DSP instructions and a floating-point unit to allow manufacturers to produce consumer audio applications offering quality playback at the lowest cost (see Figure 1).

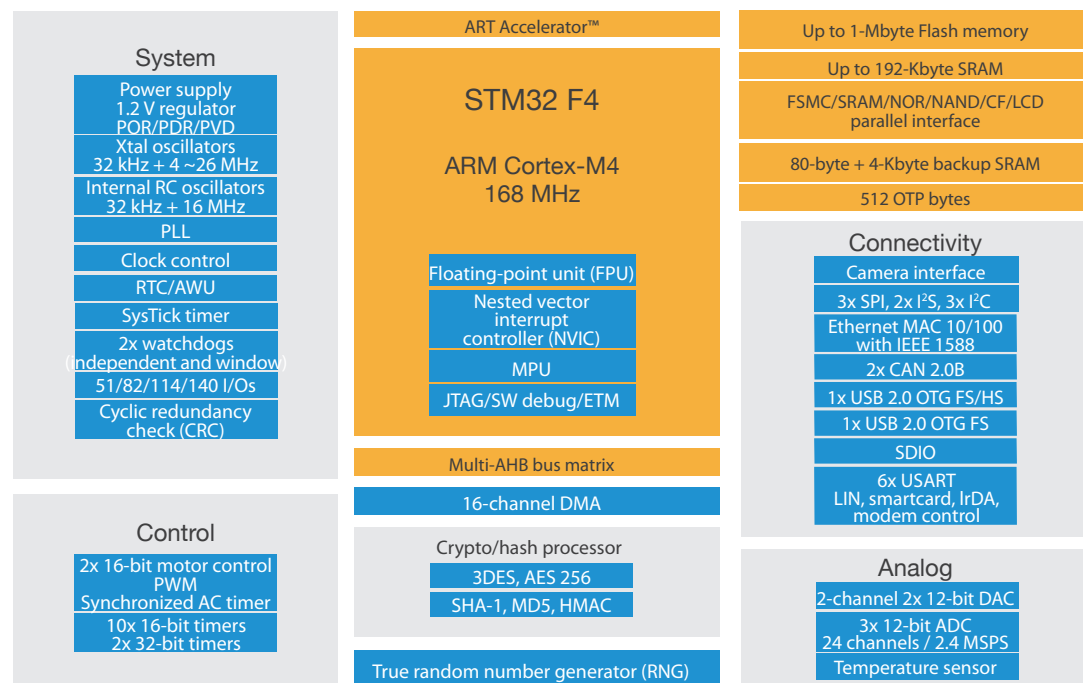
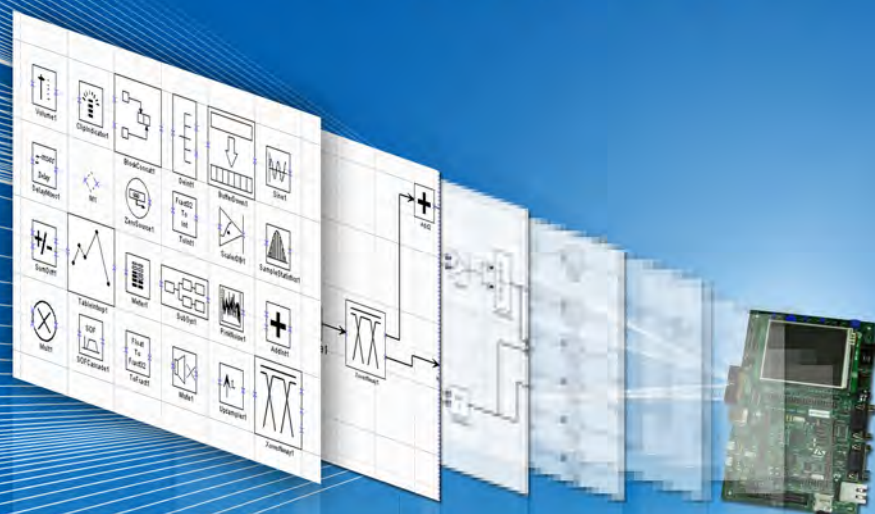


Figure 1 ST has expanded its STM32 MCUs beyond the base Cortex-M architecture with a variety of integrated peripherals to create a wide range of MCUs that optimize performance, memory, and cost for nearly every embedded application.

Accelerating Audio Product Development



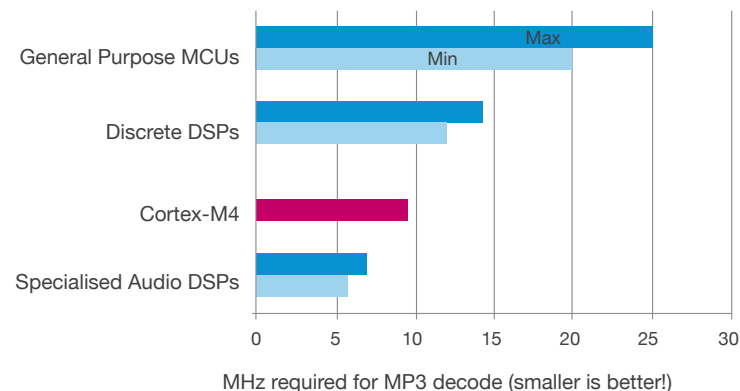
Announcing Audio Weaver support for the STM32 F4

Reference designs for USB audio, multimedia speakers, headphones, and docking stations

www.dspsconcepts.com



DSP application example: MP3 audio playback



DSP Concept

Figure 2 With its Cortex-M4 core, the STM32 F4 offers excellent audio processing capabilities that exceed the performance of many general-purpose MCUs and discrete DSPs.

The STM32 F4 offers excellent audio processing capabilities (see Figure 2). With its rich peripheral integration, a single STM32 F4 can provide a cost-effective, single-chip solution for implementing embedded audio that combines performance, ease-of-use, connectivity, and signal processing to achieve quality audio playback. Key capabilities of the STM32 F4 for accelerating audio design, enhancing performance, and lowering system cost include:

Digital Signal Processing Instructions: With the STM32 F4, developers have access to up to 105 DSP-specific instructions. These instructions include single-cycle multiply-and-accumulate (MAC), saturated arithmetic, and both 8- and 16-bit SIMD integer operations. Its architecture is designed to enable high-quality audio in consumer electronics and embedded applications in a more cost-effective manner than is possible with DSPs.

Floating-Point Unit: All STM32 F4 devices also have an integrated floating-point unit (FPU). While signal-processing algorithms can be implemented using fixed-point arithmetic, this approach adds complexity in that underflow and overflow need to be manually managed. In addition, fixed-point processing offers less dynamic range than floating-point, which impacts many audio functions. With the integrated FPU, there is no penalty for retaining this precision. Code based on floating-point can also be substantially faster and requires less memory than fixed-point code.

32-bit Efficiency: The bus size of the processor has a tremendous impact on both performance and power efficiency. Even if audio samples are streaming at 16 bits, the system still needs 32-bits to store intermediate computations. A 16-bit MCU or DSP, for example, requires seven operations (4 multiplies and 3 additions) just to complete a single 32 x 32 multiplication. The STM32 F4 can execute a 32-bit MAC (multiply and accumulate) with only one single-cycle operation.

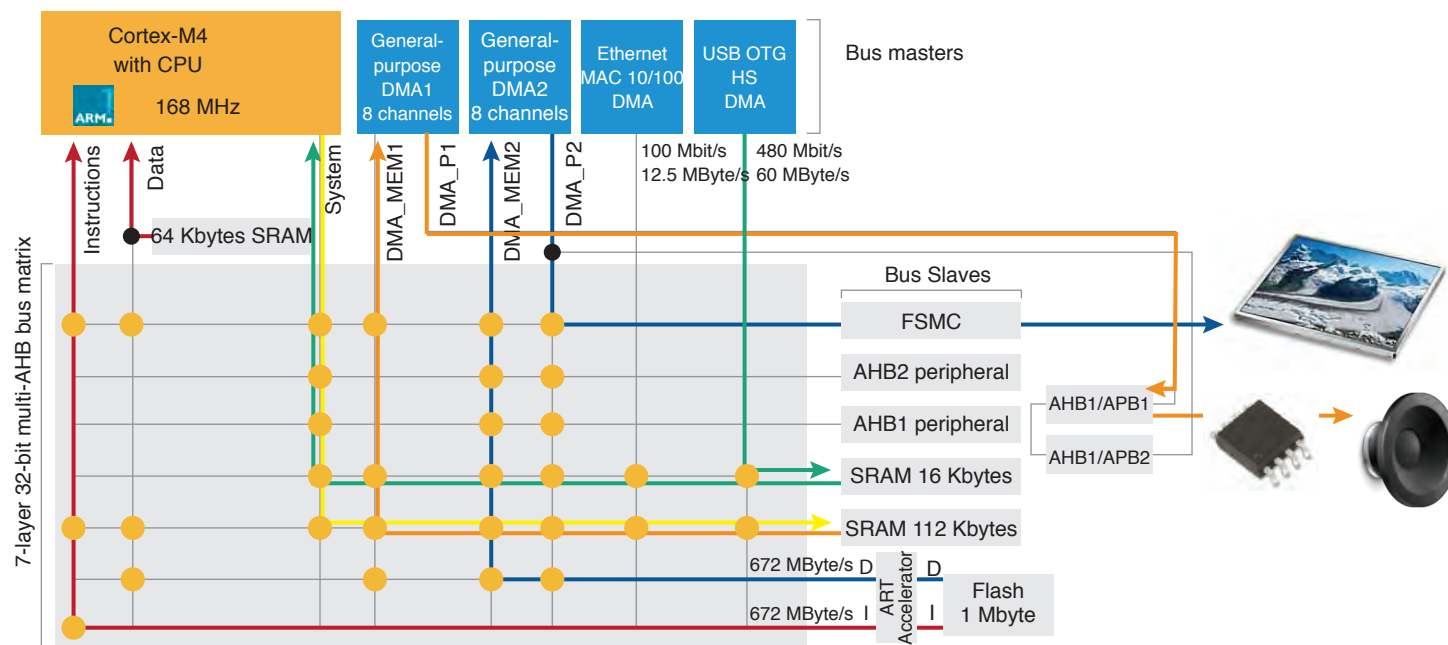


Figure 3 The multi-layer matrix that interconnects STM32 MCUs with peripherals and memory enables simultaneous transfer between multiple masters and slaves without requiring involvement from the CPU. This provides STM32 MCUs with a tremendous interconnect capacity that eliminates peripheral and memory access bottlenecks for the highest operating performance.

Multi-Layer Bus Fabric:

The key to real-time signal processing is maintaining efficient data flow. In a consumer audio device, however, the MCU must move not only signal data but manage program memory, communication ports, and other system tasks. The complexity and real-time nature of audio algorithms also requires them to be integrated with application code to ensure that

neither core application tasks nor audio playback compromise each other.

The STM32 architecture is designed to minimize this problem so that developers do not need to spend time resolving potential conflicts. This is achieved through the low interrupt service overhead of STM32 devices combined with the multi-layer bus fabric that allows multiple DMA transactions to occur simultaneously without

burdening the CPU. Figure 3 shows the high level of parallelism that can be achieved through simultaneous transfers over a multi-layer fabric:

- » Program code is executed from Flash with data stored in SRAM (red)
- » The compressed audio stream is received over USB and stored in SRAM (green).
- » CPU with DSP and FPU functionality accesses the

compressed audio stream for decompression and signal processing (green)

- » Decompressed MP3 data is sent from the CPU to SRAM (yellow)
- » Audio data is output to I2S through DMA (orange)
- » Graphical icons are transferred from Flash to the display through DMA (blue)

Communications Interfaces:

Users want to be able to access audio data from different sources and over different interfaces. With the right mix of interfaces—including USB (host and device), Ethernet for Internet Radio, SDIO, and external memory—developers can create flexible devices that support a wide range of usage models.

In addition to being able to receive data without loading the CPU, developers need to be able to address the many issues related to streaming audio, including lost packets and lack of feedback controls. For example, USB feedback controls to prevent under and overflow of the audio buffer are not always used or well implemented. This can result in lost or dropped packets that impact audio

quality. To overcome this limitation, developers can utilize sample-rate conversion (SRC). SRC is also useful for converting between audio speeds (i.e., clock domains) while maintaining audio fidelity, compensating for slight mismatches in clock speed, or for mixing audio from different sources. For applications that need SRC, the STM32 F4 requires only 10% utilization, leaving plenty of headroom for other signal-processing tasks.

Multiple Clock Sources:

Consumer audio systems require a number of different clock domains—including the CPU, USB, and I2S—that have fixed frequencies and need to be accurate as well as free of jitter. Trying to use the same clock for each of these can impact precision. For example, it is straightforward to achieve a clean clock at 168 MHz for the CPU, 44.1 KHz for an I2S interface or 48 MHz for USB but not for all three using a single clock source.

The STM32 F4 integrates two PLLs for increased clocking flexibility. The main PLL is used to generate the system clock and the second PLL is available to generate the accurate clocks needed for high-quality audio.

Complete audio system

	STM32 F2 CPU load	STM32 F4 CPU load	Flash footprint	RAM footprint
MP3 decoder	17%	6%	23k	12344
MP3 encoder	22.5%	9%	25k	16060
WMA decoder	17.5%	6%	45k	36076
AAC+ v2 decoder	25%	11%	54k	87000
Channel mixer	2.5%	2%	0.6k	16
Parametric Equalizer	16%	12%	2k	300
Loudness Control	4.5%	3.5%	3.25k	632
SRC	22.5%	10%	17.5k	1880

Figure 4 When computing 16- and 32-bit DSP functions, the STM32 F4 offers a 25–70% improvement. As a result, systems can drop into sleep mode faster to conserve power or run more algorithms to further improve audio quality.

In addition to being able to receive data without loading the CPU, developers need to be able to address the many issues related to streaming audio, including lost packets and lack of feedback controls. For example, USB feedback controls to prevent under and overflow of the audio are not always used or well implemented.

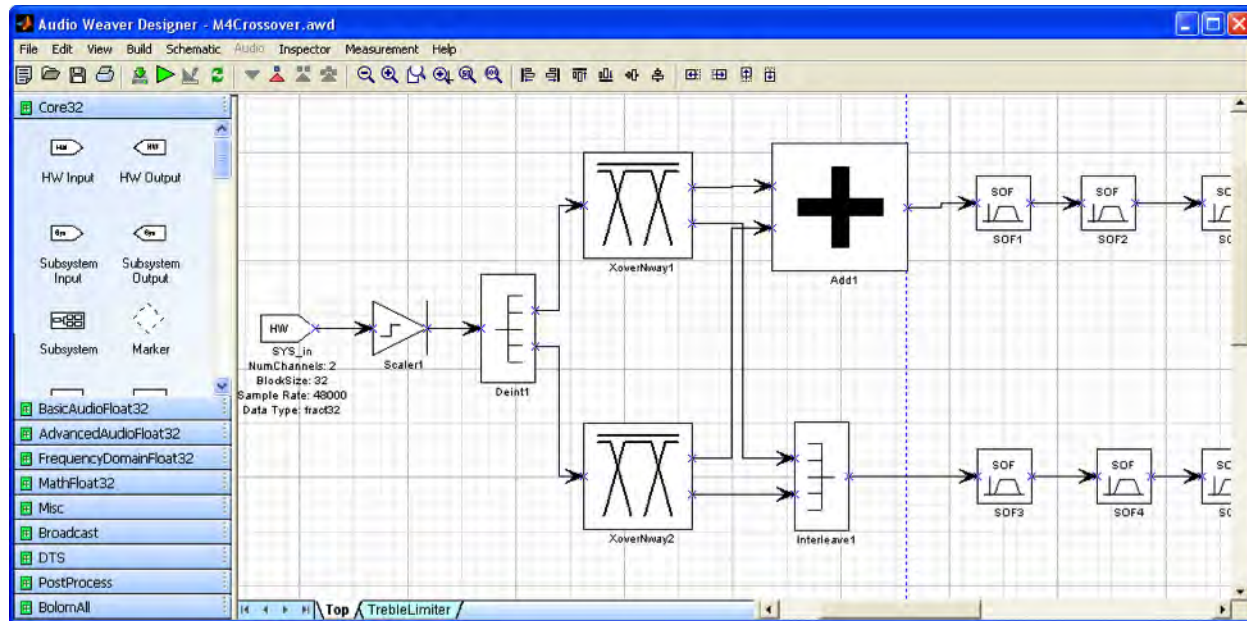


Figure 5 Audio Weaver from DSP Concepts offers a GUI-based development environment that enables developers to design the signal flow for their application by selecting processing blocks and connecting them using a drag-and-drop editor.

The ability to source different clock domains enables designs based on the STM32 architecture to maintain a permanent USB connection and avoid audio synchronization issues.

Integrated Audio Interfaces:

The STM32 F4 has two full-duplex I2S standard stereo interfaces offering less than 0.5% sampling frequency error. There is also an external clock input to the I2S peripheral if an external high-quality audio PLL is preferred. In addition to

simplifying design, integrating the I2S interfaces reduces component count, board size, and system cost.

MCU Peripherals: The STM32 architecture includes all of the real-time peripherals required for even the most demanding MCU-based application.

The combination of the STM32 F4's capabilities brings a new level of performance to audio applications. Performing a long 32-bit multiply or multiply-accumulate (MAC)

operation on an STM32 F2, for example, takes 3-7 cycles. With the STM32 F4, this operation is performed in a single cycle. When computing 16- and 32-bit DSP functions, the STM32 F4 offers a 25-70% (see Figure 4) improvement. As a result, systems can drop into sleep mode faster to conserve power or run more algorithms to further improve audio quality.

In addition to the integrated DSP capabilities of the STM32 F4, developers have access to the

CMSIS DSP library to accelerate development. The CMSIS DSP library includes a large number of DSP and floating-point functions optimized for the algorithms commonly used in audio applications. This library is supplied by ARM for processors built around the Cortex-M4 processor. DSP Concepts is the company that wrote the CMSIS DSP library. They have leveraged their intimate knowledge of the library to create the audio blocks that make up their signal processing design tool, Audio Weaver.

Audio Algorithm Design

Audio Weaver enables developers to quickly design the audio processing portion of their system; i.e., everything that goes on between receiving an audio signal and outputting it. Audio Weaver offers a GUI-based development environment that enables developers to design the signal flow for their application by selecting processing blocks and connecting them using a drag-and-drop editor (see Figure 5). Each block has hand-optimized code behind it, and the tool automatically creates the required data structures.

Because complex functions are built from base audio functions, the final code executes with no performance or efficiency losses compared to hand-coding from scratch.

When algorithm code is written by hand, each design iteration requires substantial time investment since the code must be optimized and tuned to see what its actual impact on sound quality and processing load are. With Audio Weaver, the design cycle is much faster, giving developers the ability to explore more configurations in their efforts to increase sound quality while reducing system cost. Code is highly optimized for MIPS and memory usage, supports floating-point processors such as the STM32 F4, offers flexible deployment modules, and does not require an RTOS to operate. The library includes over 150 different audio blocks, including third-party IP.

With tools like Audio Weaver, it has become possible to create highly tuned audio applications without engineers needing to have a deep knowledge of audio processing. For companies new to audio,

complete reference designs are available, with assistance from DSP Concepts to tune them for the final production system. Companies that are comfortable with audio processing can work with individual audio blocks that provide basic functionality and build them into higher-level processing

algorithms. Even sophisticated companies can accelerate design using Audio Weaver as it provides a framework with core components that not only jumpstarts design with highly optimized code but provides a development environment that facilitates fast prototyping and tuning. For these companies the

value-add of Audio Weaver is faster time-to-market.

Accelerating Optimization

To speed design, Audio Weaver supports cross-platform development. The ability to run the same algorithms on a PC as on the STM32 F4 gives engineers a powerful environment in

MIPS and memory required by each audio processing stage. This enables engineers to measure how much a particular improvement in sound quality will cost in terms of CPU utilization to determine the most efficient use of processing resources when many functions have to operate simultaneously.

When algorithm code is written by hand, each design iteration requires substantial time investment since the code must be optimized and tuned to see what its actual impact on sound quality and processing load are. With Audio Weaver, the design cycle is much faster, giving developers the ability to explore more configurations in their efforts to increase sound quality while reducing system cost.

which to design and tune the software in parallel with hardware development. Once target hardware is available, the code can be retargeted for the STM32 F4 and final optimizations made, resulting in significant time-to-market savings.

During final optimization, developers can profile the

Consider the use of different order filters to equalize the speakers. A lower-order filter, for example, may provide a frequency response that is 3 dB off of the ideal response while a higher order filter is off by only 1 dB. The relative difference in CPU utilization between these two filters can be used

to determine where to allocate CPU resources to maximize sound quality.

At the end of the day, however, audio quality is not about response graphs but how it actually sounds to people. With many development systems, engineers have to make adjustments to code, recompile, and download code before they can hear a new configuration. However, to assess the impact of a lower-order filter on quality, for example, developers need to be able to hear both configurations right after each other.

Audio Weaver solves this problem by supporting a tuning interface that can change filter characteristics in real-time. With the ability to configure and switch between multiple settings with the click of a button, developers can compare two sets of speaker equalizations or different spatial processing. Note that the tuning interface is seamless and transparent, compared to instrumenting code that can impact quality because of extra loading on the CPU.

The ability to tune quickly and easily without recompiling can substantially shorten the time

it takes to optimize a system. Flexible tuning also simplifies the optimization process for developers new to audio.

Note that audio applications are not comprised solely of audio processing. To accelerate system design, DSP Concepts also provides an extensive range of software functionality beyond its extensive audio module library, including:

- » Real-time kernel
- » Audio I/O management
- » PC/host control interface
- » Boot loader
- » Update manager
- » Flash file system

System-level Design

One of the challenges to adding audio to embedded designs is that while many MCU manufacturers offer reference designs, audio is typically not one of the applications supported.

To address this shortcoming, ST has invested significantly in creating digital audio resources for its customers in order to offer complete audio reference designs as well as tools that enable the design of quality

audio optimized for the STM32 architecture. For example, ST and its partners offer a variety of evaluation boards with audio capabilities. ST also offers several docking station reference designs that provide a representative design that can be used in a wide range of embedded applications.

For Apple Made for iPod (MFI) licensees, ST offers the Apple iAP application, a complete solution based on STM32 F2 and STM32 F4 devices to deliver a high-quality music experience. The Apple iAP application support both simple accessory and audio streaming accessory for iPod, iPhone, and iPad devices. Components include:

- » Either the STM322xG-EVAL or STM324xG-EVAL board to which developers connect their Apple Authentication Coprocessor (ACP) circuit
- » Free Apple “iPod Accessory Protocol” (iAP) firmware with Lingo for authentication and control/information data
- » Free USB Host Library with USB Host HID class for control and information data

For audio streaming accessories, the Apple iAP application also supports:

- » Free USB Host Library USB Host Audio classes
- » Remote iPod/iPhone/iPAD control
- » Digital audio streaming
- » Music tag extraction
- » Flash card reader capabilities, such as using an SD card or MMC, that can decode audio files from this media. Optimized decoders are provided for this purpose free of charge

Today’s consumer audio devices are complex systems that require both high performance and flexibility to meet rapidly changing market expectations. With its high performance core, efficient multi-layer bus fabric enabling simultaneous data transactions, and the right mix of MCU peripherals and connectivity, the STM32 F4 is an ideal architecture for many embedded and consumer audio applications. Developers can now design systems offering synchronized digital audio playback of the highest quality using a single MCU. 

Bringing Floating-Point Performance and Precision to Embedded Applications

By Olivier Ferrand, Application Engineer, STMicroelectronics
Stephane Rainsard, Application Engineer, STMicroelectronics
Abdelhamid Ghith, Application Engineer, STMicroelectronics

Hardware-based floating-point capabilities have long been an option on high-end MPUs and DSPs designed to serve as computational workhorses. Embedded systems based on MCUs, however, have classically used a fixed-point implementation.

There are many reasons for this. The types of simple calculations embedded systems needed to make could be handled sufficiently using fixed-point math. The resulting code was not only well-suited to an MCU's native mathematical capabilities, it resulted in faster execution and was more memory efficient than an equivalent floating-point implementation would be. The only disadvantage of using fixed-point was the tradeoff between range and precision, which could be tolerated in most cases.

Some of today's embedded systems are completely different from their early predecessors. They operate at high clock speeds and need to perform more complex calculations than ever before. While processing and memory efficiency are still primary design considerations, a more useful range and higher precision have become essential in many applications.

For example, to output high quality audio, MCUs need to support a wide dynamic range so that the system sounds good at both low and high frequencies. Medical applications, such as heart-rate monitors and glucose meters, need to measure more subtle signals and quantities. For industrial applications, higher precision in the MCU enables developers to use lower quality components in other parts of the system, reducing overall

system cost. And with the increasing use of capacitive-sensing technology, nearly every embedded application can benefit from detecting user inputs with greater accuracy, especially on displays with limited surface area.

Floating-Point vs Fixed-Point

The balanced combination of range and precision in a floating-point number comes from its separation of the exponent and mantissa. When a number is stored in a fixed-point format, the position of the exponent is assumed and all of the bits are reserved for the mantissa (i.e., the actual digits used to represent the number). The issues that arise with fixed-point formats are similar to those associated with reading an ADC: the wider the range of values to be represented, the less resolution

there is at the lower end of the range. If very small changes in value need to be captured, the range must be more narrow. Developers, then, must choose between range and resolution.

Floating-point addresses these issues by dedicating a few bits to an exponent to track the decimal point. For example, the single-precision format supported by the STM32 F4 uses one bit for the sign and 8 bits for the exponent, leaving 23+1 bits for the mantissa (the normalized format used adds an implicit bit to the 23 bits stored in the floating-point number). As such, a single-precision floating-point number gives a more useful range and precision combination compared to 8-, 16-, or even 32-bit fixed-point numbers which either give a wide range with greatly reduced precision or higher precision with a much smaller range.

The 32-bit single-precision format is part of the IEEE 754 Floating-Point Arithmetic standard. This standard represents decades of experience and provides a common approach for supporting floating-point arithmetic that unifies processors, coding tools, and high-level design tools. Specifically, the 754 standard is the basis for the floating-point data types used in C. In turn, these floating-point C formats are used in code generated by high-level modeling/Meta language tools like MATLAB and Scilab.

One of the key benefits of using floating-point is that it enables developers to make more efficient use of C and powerful algorithm development tools that have previously been reserved for DSP- and MPU-based design. Generally speaking, floating-point numbers are easy to manipulate in C. High-level tools further accelerate and simplify development by enabling developers to describe complex algorithms in equation form and then quickly generate efficient C code rather than requiring them to hand-code algorithms in assembly. In addition, these high-level tools offer tremendous flexibility in allowing developers to rapidly implement changes

to algorithms without having to completely rewrite and re-optimize algorithm code. The end result is significant savings in development time and cost savings. Such tools also offer a powerful means for developers to test and validate applications.

Software-based Floating-Point

In the past, developers had a limited number of ways in which they could utilize floating-point technology in MCU-based designs to increase computational precision. Introducing a second processor to handle calculations was often too expensive to consider. Alternatively, while the design could be migrated to a powerful MPU or DSP, neither type of architecture is as well-suited for the real-time demands of embedded systems as an MCU.

The availability of high-performance MCUs made it possible for developers to bring in a floating-point library to perform calculations in software. While such libraries enabled developers to use floating-point algorithms, they came at the expense of significantly reducing system throughput.

Such libraries also tended to be quite large and introduced significant processing overhead. For example, every operation between two numbers had to first align the numbers to have the same exponent and then, after performing the operation, round the result and code it back into the fixed-point format. While all of these actions were performed by the floating-point library and were not directly visible to developers, they still resulted in significantly degraded performance, greater processing latency, and an increased memory footprint.

the C code generated by the tool into a fixed-point implementation. The downside of this approach is the significant time investment required to adapt the code combined with the inflexibility to easily modify the algorithm later in the design cycle.

The STM32 F4 Advantage: Integrated Floating-Point Technology

To stay competitive, developers of precision and high-performance embedded systems need to have access to floating-point functionality without sacrificing performance or memory efficiency

To stay competitive, developers of precision and high-performance embedded systems need to have access to floating-point functionality without sacrificing performance or memory efficiency.

If none of these alternatives was feasible and developers still needed the flexibility that high-level modeling tools bring to the design process, they had the alternative to manually convert the floating-point operations in

through the use of a software-based floating-point library or having to adapt the code by hand to a fixed-point implementation.

With the STM32 F4 MCU architecture, developers have



STM32[®] F4

**World's highest
performance
Cortex[™]-M MCU
168 MHz/210 DMIPS**



STM32[™] Releasing your **creativity**

For further information, visit www.st.com/stm32f4

the option of bringing floating-point efficiency to an extensive range of low-cost embedded applications. The STM32 F4 integrates a floating-point unit (FPU) to execute these operations natively in hardware. The FPU is fully compliant with the IEEE.754 standard and has its own 32-bit single-precision registers to handle operands and results. These registers can be viewed as double-word registers to enable more efficient load and store operations. The context of the FPU can be saved to the CPU stack using several methods based on the application architecture and whether registers need to be preserved or not.

The FPU supports the five different classes of numbers defined by the 754 standard—normalized, denormalized, zeros, infinities, and NaNs (Not-a-Number). It also supports the five exceptions of the standard—overflow, underflow, inexact, divide by zero and invalid operation—allowing applications to handle operations such as trying to compute the square root of a negative number (i.e., resulting in NaN + invalid operation exception). Exceptions are “untrapped”, meaning that the FPU will return the result as specified by the 754 standard and raise an exception

FPU assembly code generation

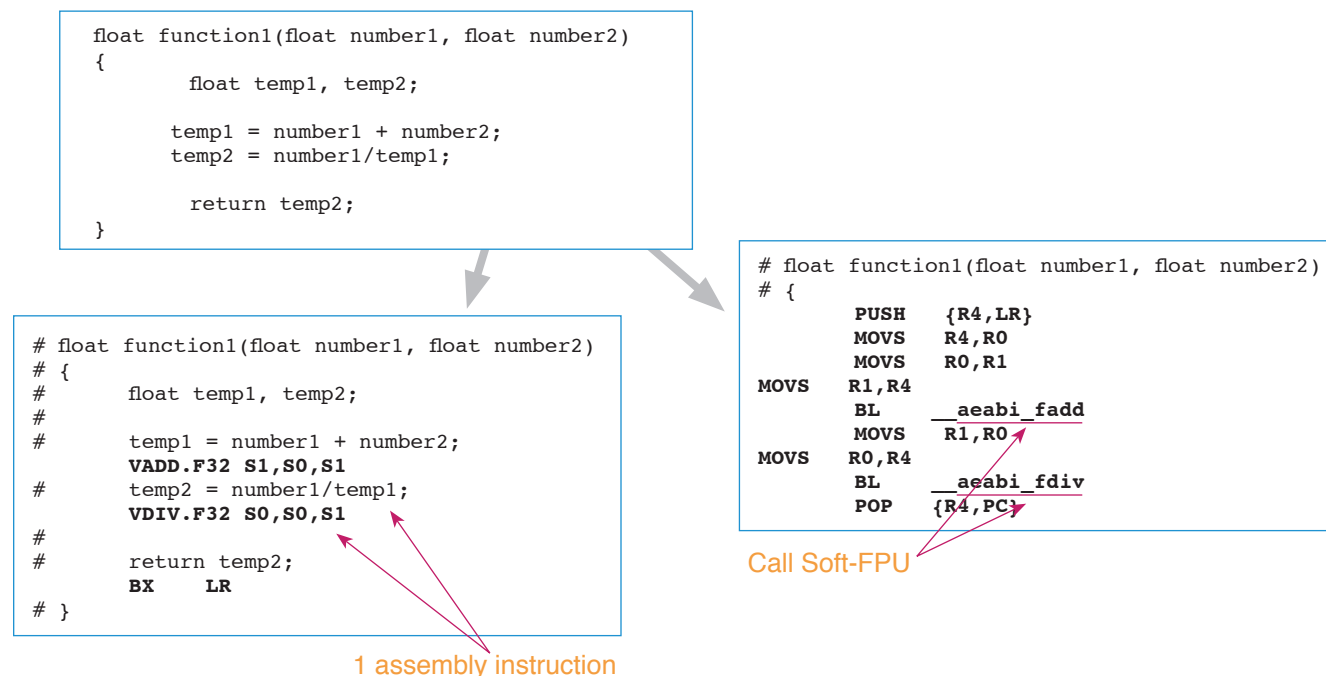


Figure 1 There is a significant reduction in code size when an integrated FPU is available (code on left) than when one is not (code on right).

flag. If needed, developers can also use the STM32 F4 floating-point global interrupt to address the issue.

The integrated FPU of the STM32 F4 offers a number of advantages to embedded designers:

- » Access to the more useful range and precision that floating-point brings
- » Reduced coding complexity by being able to work with numbers in a more natural format

- » Greater throughput compared to software floating-point libraries.
- » Accelerated application development as C code generated by high-level tools can be used without modification or wrappers
- » Smaller code footprint since instructions that used to be multiple lines of code in software libraries are now implemented with a single instruction

- » Simplified debugging as macro calls in floating-point libraries are eliminated

Effectively, the STM32 F4’s FPU reverses the value proposition between fixed- and floating-point for many MCU-based designs.

Seamless Integration

Figure 1 shows the difference between the assembly code generated when an FPU is available on an MCU as

```

void GenerateJulia_fpu(uint16_t size_x, uint16_t
size_y, uint16_t offset_x, uint16_t offset_y, uint16_t
zoom, uint8_t * buffer)
{
float tmp1, tmp2;
float num_real, num_img;
float radius;

uint8_t i;
uint16_t x,y;

for (y=0; y<size_y; y++)
{
for (x=0; x<size_x; x++)
{
num_real = y - offset_y;
num_real = num_real / zoom;
num_img = x - offset_x;
num_img = num_img / zoom;
i=0;
radius = 0;
while ((i<ITERATION-1) && (radius < 4))
{
tmp1 = num_real * num_real;
tmp2 = num_img * num_img;
num_img = 2*num_real*num_img + IMG_CONSTANT;
num_real = tmp1 - tmp2 + REAL_CONSTANT;
radius = tmp1 + tmp2;
i++;
}
/* Store the value in the buffer */
buffer[x+y*size_x] = i;
}
}
}

```

Figure 2 Algorithmic code, such as for the Julia set as shown here, requires no modification to take advantage of the STM32 F4's hardware-based floating-point capabilities.

compared to when one is not. The example is a simple one, adding number1 and number2 together and storing the result in temp1. Number1 is then divided by temp1 and stored in temp2.

When an FPU is available, the compiler directly uses native FPU instructions (see add and divide instructions in the code on the left). When an FPU is not available, however, the compiler inserts macro calls to the software floating-point library to perform the function (see code on the right). These functions comprise multiple instructions that take many more cycles to complete the calculation. Although the difference is only a few instructions, the extra overhead represents a high percentage of the algorithm's overall load on the CPU. When considered over the whole of an application, the impact on processing efficiency is tremendous.

Performance by the Numbers

The following example clearly illustrates the benefit of having an integrated FPU. Figure 2 shows the code for a simple mathematical fractal known as the

Julia set, given by the equation:

$$Z_{n+1} = Z_n^2 + c$$

where the sequence for each $x+ i.y$ point is computed with:

$$c = c_x + i.c_y$$

This algorithm provides an effective way to show the impact on performance of the STM32 F4's FPU since no code modification is required to utilize it. In fact, the only difference is whether the FPU is activated during compilation.

Figure 3 shows the time spent for the calculation of the Julia set using different zooming factors. As can be seen, the presence of an FPU yields an increase in performance on the order of 11.5 to 17 times faster. Again, no modification to code is required, just selecting the FPU in the compiler options.

For applications that need to implement signal processing capabilities or operate on multiple data in parallel, STM32 F4 MCUs have a versatile architecture that also implements hardware-based DSP capabilities. The availability of both an FPU and DSP instructions in the STM32 F4 provides developers with a full range of capabilities to

Frame	Zoom	Duration with FPU	Duration without FPU (microlib)	Ratio	Duration without FPU (no microlib)	Ratio
0	120	222	3783	17.04	2597	11.70
1	110	194	3276	16.89	2243	11.56
2	100	167	2794	16.73	1906	11.41
3	150	298	5156	17.30	3558	11.94
4	200	312	5412	17.35	3740	11.99
5	275	296	5124	17.31	3540	11.96
6	350	284	4905	17.27	3389	11.93
7	450	289	4989	17.26	3448	11.93
8	600	273	4705	17.23	3251	11.91
9	800	267	4592	17.20	3173	11.88
10	1000	261	4485	17.18	3100	11.88
11	1200	255	4374	17.15	3023	11.85
12	1500	242	4138	17.10	2860	11.82
13	2000	210	3555	16.93	2455	11.69
14	1500	242	4138	17.10	2860	11.82
15	1200	255	4374	17.15	3023	11.85
16	1000	261	4485	17.18	3100	11.88
17	800	267	4592	17.20	3173	11.88
18	600	273	4705	17.23	3251	11.91
19	450	289	4989	17.26	3448	11.93
20	350	284	4905	17.27	3389	11.93
21	275	296	5123	17.31	3540	11.96
22	200	312	5412	17.35	3740	11.99
23	150	298	5156	17.30	3558	11.94
24	100	167	2794	16.73	1906	11.41
25	110	194	3276	16.89	2243	11.56

Figure 3 This table shows the time spent for the calculation of the Julia set using different zooming factors. The presence of an FPU yields an increase in performance on the order of 11.5 to 17 faster with no modification to code required.

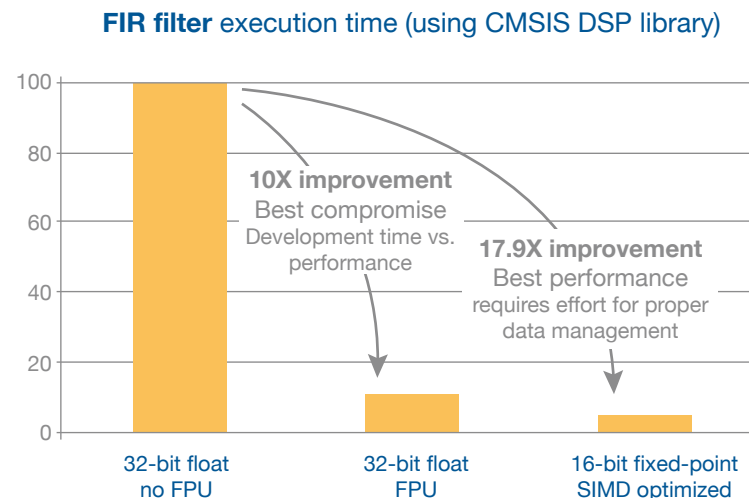


Figure 4 The relative time it takes to execute a FIR filter is 10X better when utilizing the STM32 F4's FPU. If more headroom is needed, the use of the 16-bit fixed-point SIMD (single instruction, multiple data) optimized instructions that are part of the DSP capabilities of the STM32 F4 increase performance by 17.9X.

implement the wide variety of algorithms embedded applications require. Each accelerates different types of processing and together they complement each other to enable the most optimized implementation based on performance, memory, and ease of programming.

The ease of migrating an existing application to seamlessly take advantage of the STM32 F4's integrated FPU capabilities is important

as well. Consider an STM32 F2 application executing a floating-point FIR filter based on the CMSIS DSP library available through ARM. Figure 4 shows the relative time it takes to perform the FIR filter 100 times on an STM32 F2 with no FPU or DSP capabilities using a purely software implementation to make the necessary floating-point calculations.

When the same code is compiled for the STM32 F4 to leverage the FPU in hardware, there is

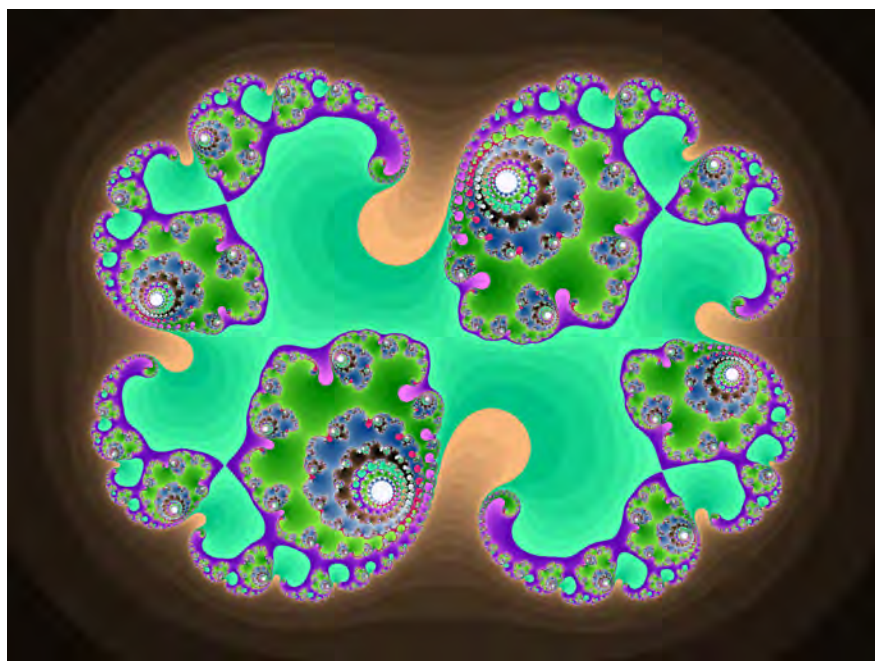


Figure 5 A top-level view of the Julia set.

an immediate 10X improvement in relative performance (i.e., factoring out the impact of clock speed). Just by changing processors and activating the FPU, there is a significant performance advantage. This one change to the design is enough to yield a tremendous amount of headroom for additional tasks, depending upon the application.

If more headroom is needed, the performance of the FIR filter can be further improved

through the use of the 16-bit fixed-point SIMD (single instruction, multiple data) optimized instructions that are part of the DSP capabilities of the STM32 F4. For example, when using the SIMD optimized FIR algorithm of the CMSIS library from ARM, the performance improvement is 17.9X, again not considering clock speed.

Unparalleled Flexibility

Traditionally, when designing a DSP-based system, developers have to choose between fixed- and floating-point architectures. Floating-point comes at a price premium, and manufacturers have had to balance this added cost against the extra complexity in application development that comes when using fixed-point. While the FPU is an optional component of the ARM Cortex-M4 architecture, ST has designed its STM32 F4 family so that every device offers an FPU. In this way, developers always have access to floating-point performance and precision without compromise.

The bottom line is that the STM32 F4 provides developers with a flexible, high-performance architecture that offers the real-time responsiveness of an MCU with the precise floating-point and digital signal processing capabilities that today's embedded designs need. The ability of the STM32 F4's FPU to perform fast mathematical computations on C-based float data is a key benefit for many application tasks that require precision, including loop control,

audio processing, sensor signal conditioning, digital media decoding, and digital filtering, to name just a few.

The availability of an FPU speeds complex algorithm development, all the way from high-level design tools down to software generation. Hardware native support for floating-point operations simplifies coding and substantially accelerates product development by enabling the most efficient implementation for any mathematical calculation. Code that has been generated by such tools to be used directly by the FPU will offer the highest level of performance.

The faster processing an FPU brings to applications offers either more headroom to support new functionality or faster time-to-sleep for power-sensitive applications. It also enables developers to introduce more complex processing and functionality to applications than was previously possible with traditional MCUs. As a result, when implementing a mathematical algorithm on an STM32 F4, developers never have to choose between performance and development time. 🦋

Achieving Ultra-Low-Power Efficiency for Portable Medical Devices

By Jean J. Labrosse, Founder, CEO and President, Micrium
Jim Lombard, Application Engineer, STMicroelectronics

The availability of low-cost, full-featured microcontrollers has created a revolution in the health industry leading to equipment migrating from the hospital and into patients' homes. Designing these portable medical devices presents new challenges for engineers, such as implementing precision analog processing requiring complex calculations and creating a system that is simple and comfortable to operate even for physically-challenged users. In addition, these devices have to be able to run as long as possible without having to change or recharge batteries, even after sitting on a warehouse shelf for up to 18 months. To meet the unique requirements of portable medical devices, developers need an advanced processor architecture that combines performance, ultra-low-power process technology, low-cost, and efficient power management and communications capabilities.

Designing for Medical Applications

Next-generation medical equipment is evolving along two primary paths: sports and personal health. Both markets require innovation that enables portable devices to collect more run-time data and complete advanced calculations to create a comprehensive profile of a person's current condition. Within personal health, developers also need to understand the special requirements of emerging disposable devices.

The portable medical device market is extremely strong. Consider that with 8.3% of the US population suffering from diabetes, a portable glucose meter is an essential tool for individuals who need to monitor their glucose for conceivably the rest of their lives. Similarly, an electrocardiogram (ECG) monitor enables people at risk for any

number of conditions to record their ECG waveform at home. In addition to eliminating the need for multiple office visits, home-based testing gives doctors a more comprehensive patient history to work with.

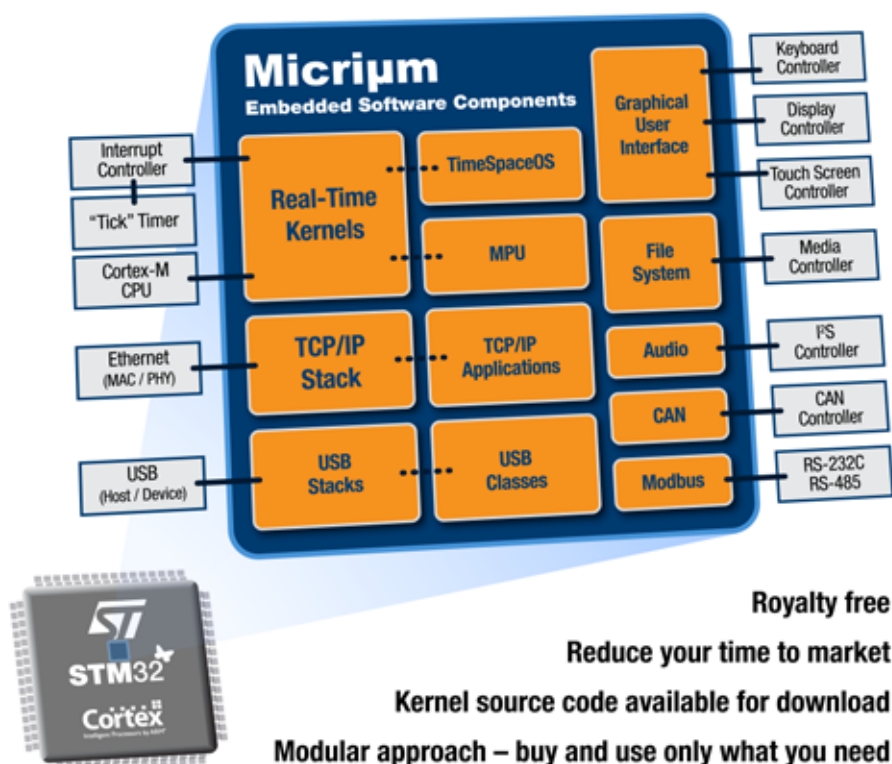
Portable handheld medical devices have a stringent set of requirements (see Figure 1.) Given that prospective users can be from all walks of life, devices must be as simple to use as possible, with limited or no setup required. They also need to be comfortable to use and operable even by physically-challenged users. In general, the display must be large for ease of reading and utilize a minimum number of buttons to avoid confusing users. Ideally, as many functions as possible need to be automated so that users don't have to be trained how to use the device. If the device has a touchscreen, the GUI must be intuitive and have a limited number of

- » Easy and simple to operate
- » Large display
- » Minimal number of large operator buttons
- » Batteries are easy to change, easy to recharge, or are sufficient to last for the operating life of the device
- » Safe and accurate operation
- » Low cost
- » Low power
- » Simple connectivity
- » Audio feedback

Figure 1 Common Product Requirements for Portable Medical Devices

operating modes. Both low cost and low power consumption are critical as well, and devices need to be able to run as long as possible without having to change or recharge the battery.

Take your STM32-Based Products to the Next Level

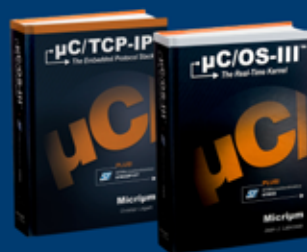


Micrium’s industry leading books take you far beyond product training. They give you in-depth knowledge of the inner working of Micrium’s high-quality software components, with a focus on STMicroelectronics devices.

- *µC/OS-III: The Real-Time Kernel*
- *µC/TCP-IP: The Embedded Protocol Stack*

Micrium products are created for engineers by engineers.

Contact us to discuss your project requirements, challenges, and goals.



www.Micrium.com

An important trend in medical applications is the use of disposable devices. High-volume devices such as heart-rate monitors can provide medical professionals with important data that is useful in identifying issues before they become full-blown problems; e.g., by having a patient track his or her heart rate for a few weeks after an operation, a doctor can verify the patient’s successful recovery.

The use of disposable devices offers many benefits. Rather than require patients to purchase a monitoring device designed to operate for years, the hospital can provide a “disposable” version that can perform the task reliably for weeks at a significantly lower cost. This strategy also enables hospitals to leverage innovations in technology faster.

Designing a medical device for limited use, however, substantially shifts the design mindset. For example, device cost becomes significantly more important when the revenue stream from consumables is weeks rather than years. Devices also have to be ready to operate out-of-the-box, so parameters

such as which test strips are going to be used need to be preprogrammed into devices by manufacturers. Power efficiency becomes more critical as well. Even though they will only be used for a handful of weeks, devices may first sit on the shelf for up to 18 months. During this time, the device is in a low power mode with the real-time clock running. With an ultra-low-power MCU, it is possible to achieve this without requiring the user to change batteries.

The STM32 Architecture

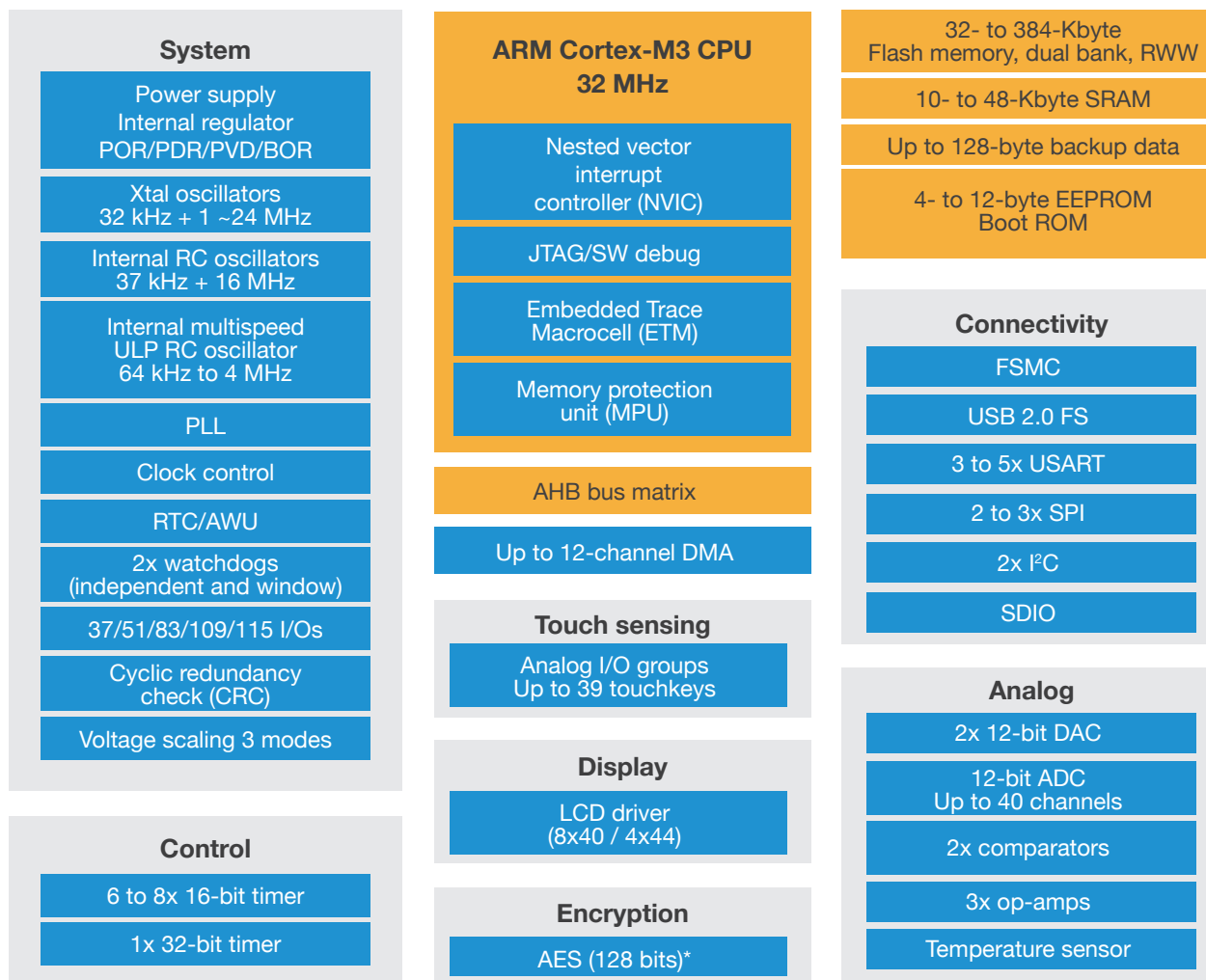
With the STM32 architecture, ST offers developers a variety of options for balancing cost, power, and performance. With its smaller die, reduced instruction set, and smaller memory footprint, the STM32 F0 provides excellent power efficiency at a very low cost. This is an ideal MCU for applications that don’t need to operate longer than six months or that use rechargeable batteries. For applications where power is tantamount, such as devices operating on a coin cell, the STM32 L1 is optimized for ultra-low power performance. For devices needing more processing capabilities and

connectivity, the STM32 F1, STM32 F2, and STM32 F4 offer an increasing range of capabilities.

The STM32 F0 is based on the ARM Cortex-M0 core. While other manufacturers also offer MCUs based on this core, ST has integrated its “STM32 DNA” into the STM32 F0 to create an MCU that provides efficient data processing with the key peripherals MCU-based applications require. The STM32 F0 also offers DMA capabilities to accelerate data processing and enable the lowest power operation even when sampling the ADC at a high data rate. With MCUs that have a lower level of integration, developers have to make compromises, such as settling for an 8-bit ADC rather than having access to the 12-bit ADC of the STM32 F0.

For complex medical systems that involve extensive computations, power consumption can be reduced by introducing the STM32 F0 as a second processor. When a system is in a low power mode, for example, it must still manage the UI and incoming data over its interfaces. Using

STM32 L15x



Note: * STM32L16x only

Abbreviations:

AWU: Auto wakeup from halt
BOR: Brown out reset
I²C: Inter integrated circuit

PDR: Power down reset
POR: Power on reset
PVD: Programmable voltage detector

RTC: Real time clock
SPI: Serial peripheral interface
USART: Universal sync/async receiver transmitter

Figure 2 The STM32 L1 family is built on ST’s 130 nm ultra-low leakage process technology that minimizes node capacitance for ultra-low-power operation and efficiency.

the primary host processor to perform these operations will use substantially more power than having a more power-efficient STM32 F0 dedicated to these tasks.

For the highest power efficiency, ST offers its STM32 L1 family (see Figure 2). Based on the ARM Cortex-M3, the STM32 L1 is built on ST's 130 nm ultra-low leakage process technology that minimizes node capacitance. This, combined with how the ARM Cortex RISC architecture reduces the overall number of active nodes, creates a powerful combination for ultra-low-power operation. Providing 32-bit performance at up to 32 MHz, STM32 L1 MCUs offer up to 384 K Flash and 48 K SRAM, as well as from 48 to 144 pins to support high I/O applications.

With the STM32 F1, STM32 F2, and STM32 F4 families, developers have access to a tremendous range of processing capacity and connectivity. The STM32 F1 is based on a Cortex-M3 core running at up to 72 MHz. For higher-end applications, the STM32 F4 offers a 168 MHz Cortex-M4 core with both integrated

floating-point (FPU) and digital signal processing (DSP) capabilities. In addition, these MCUs have multiple DMAs and a multi-layer bus matrix to offload the CPU and maximize data throughput.

Power Efficiency

There are many ways the STM32 architecture optimizes power consumption. For example, the STM32 L1 has an internal LDO regulator that can be programmed to three discrete voltage levels. Since energy consumed is proportional to the square of the core voltage, even a very small change in voltage can produce dramatic results. This allows developers to keep the STM32 L1 running at 1.2V and only switch to a higher performance range when a specific task needs to run faster.

Integrated DSP capabilities in the STM32 L1 also speed complex algorithm processing. This is in addition to the power/performance advantage 32-bit architectures have compared to an 8- or 16-bit MCUs. With its wider bus, the STM32 can perform tasks such as moving memory or complex mathematics much faster and more efficiently as well.

The STM32 L1's high level of integration enables it to provide a single-chip solution, with the exception of a few analog signal conditioning circuits, for many portable medical applications, including glucose meters, heart-rate monitors, and pulse oximeters.

The STM32 L1 also provides multiple, industry-leading low-power modes (see Figure 3), dynamic voltage scaling, and the ability to run out of RAM and disable the Flash controller

to conserve power. It also integrates a programmable voltage detector that can monitor battery voltage using less current than an ADC.

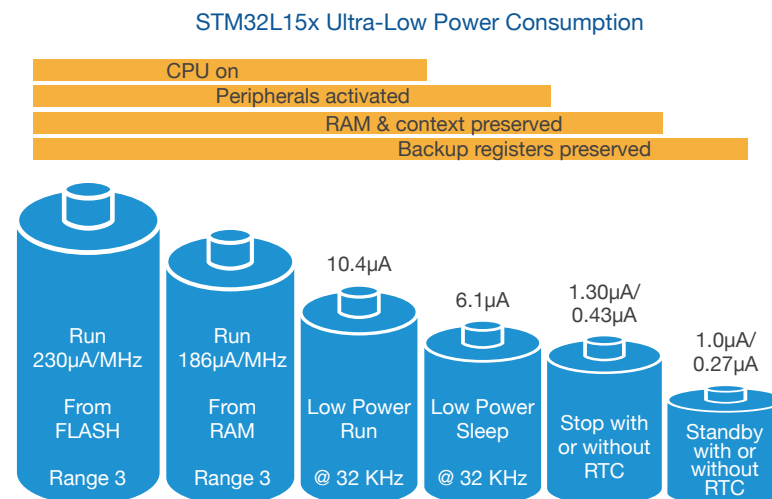


Figure 3 The STM32 L1 provides multiple, industry-leading low-power modes to maximize device operating power efficiency.

The STM32 L1's high level of integration enables it to provide a single-chip solution, with the exception of a few analog signal conditioning circuits, for many portable medical applications, including glucose meters, heart-rate monitors, and pulse oximeters. For example, in a glucose meter (see Figure 4), the STM32 L1 can automatically wake from sleep when a test strip is inserted into the device. Its 2-channel DAC can be used to generate a strip bias, enable strip calibration, and output audible instructions and test results. Timers accurately control the ADC trigger for sample measurement, onboard comparators measure correct sample staging, and the temperature sensor logs the ambient temperature for use in calculating results. Developers can also use the integrated comparators to create a power-efficient analog watchdog that monitors an input and wakes the STM32 L1 when either the upper or lower threshold is exceeded (see Figure 5).

A key manner in which the STM32 architecture conserves power is through the ability of the CPU to sleep during ADC

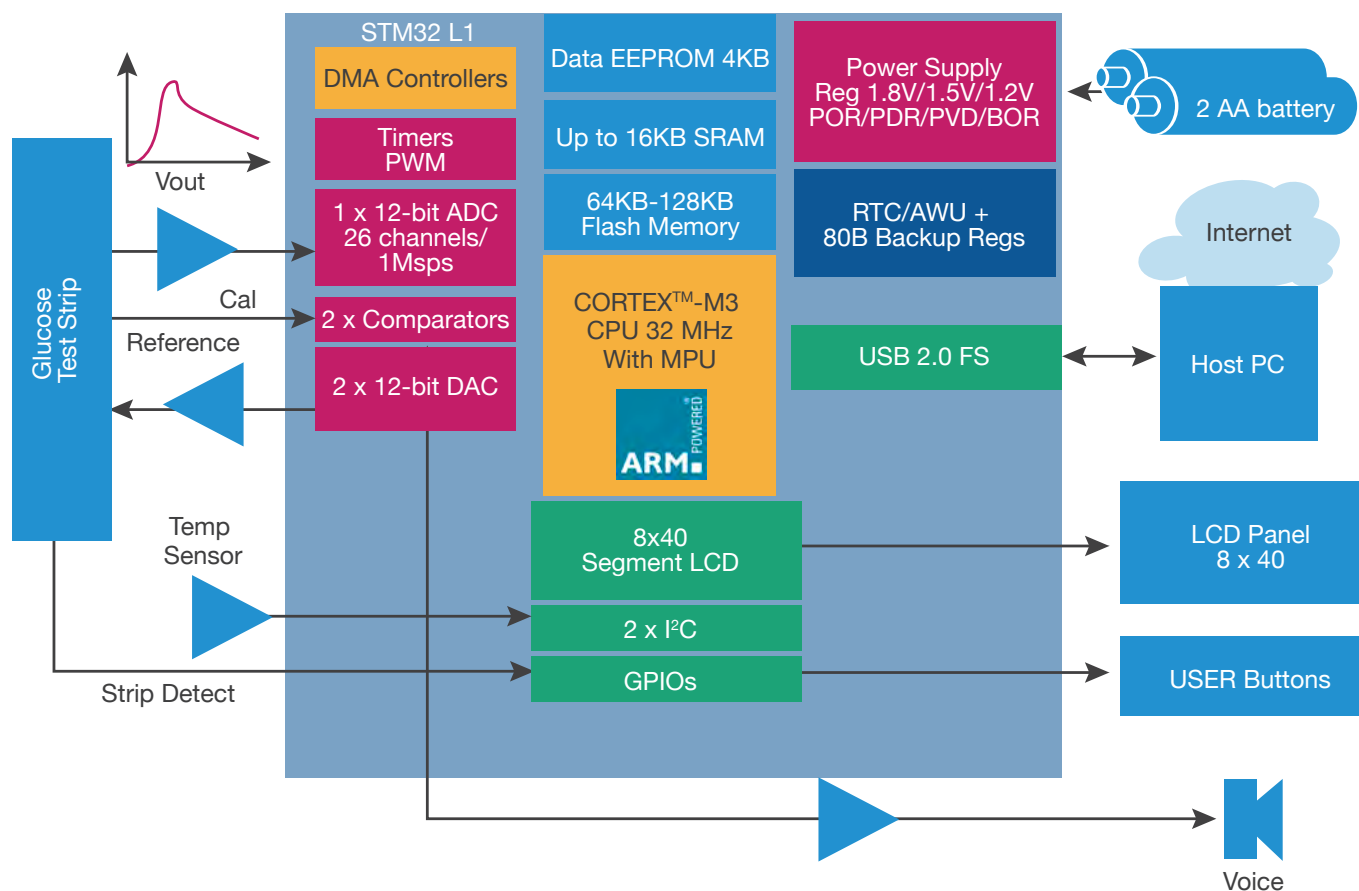


Figure 4 The STM32 L1's high level of integration enables it to provide a single-chip solution, with the exception of a few analog signal conditioning circuits, for many portable medical applications such as blood glucose meters.

sample capture. To achieve this, the entire analog data capture chain needs to be completely automated, with no need for CPU intervention at any point after the chain is initiated. Specifically, after the CPU configures and starts the auto

sample capture, it enters sleep mode. Between samples, the ADC enters an automatic shut down mode until a timer triggers it. The ADC captures the current sample, using the DMA to store the data in SRAM, and then shuts down again. This process

repeats until the entire capture sequence is complete. The DMA will then wake the CPU to process the samples that have been stored in SRAM. The power savings can be significant using the unique ability of the STM32 L1 to

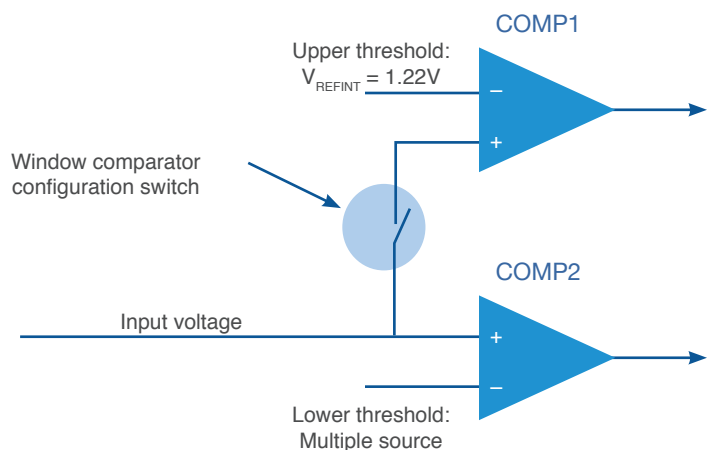


Figure 5 Developers can use the integrated comparators to create a power-efficient analog watchdog that monitors an input and wakes the STM32 L1 when either the upper or lower threshold is exceeded.

CPU running at	ADC current consumption (in μA) (ADC in continuous mode, delay of 15 cycles between channel conversions, conversions last $1\mu\text{s}$ (16 cycles at 16MHz))	
	ADC is running in Normal Mode **	ADC is On in Power Saving Mode***
16MHz (from HSI)	1453 μA	630 μA
4MHz (from MSI)*	1453 μA	445 μA
1MHz (from MSI)*	1000 μA	258 μA
32kHz (from MSI)*	900 μA	150 μA

* : HSI is On ** : PDI=PDD=0 *** : PDI=PDD=1

Figure 6 The automatic shut down mode of the STM32 L1 turns off the ADC for the majority of time between samples to provide dramatic savings in power consumption compared to when the ADC must be left on continuously.

power down the ADC between samples. Consider an ECG monitor where the sampling rate is 1 KHz or one sample every 1 ms. The ADC of the STM32 L1 consumes at most 900 μA and has a capture time of only 1 μs for a power duty cycle of 0.1%. If the STM32 L1 were a typical MCU, the ADC would consume full power even when it is not capturing samples. Thus, if the system were using a 1 ms measurement window, the ADC would draw 900 μA during a power duty cycle of 100%.

With the automatic power down control mode of the STM32 L1, however, the ADC is able to power down for the majority of time between samples. Figure 6 shows actual power consumption numbers for various operating conditions. The difference in power consumption is dramatic between when the ADC is left on continuously versus using automatic power down control saving mode, dropping in the lowest power consumption example from 900 μA to 150 μA at 32 kHz.

Note that the STM32 L1 has the flexibility to scale the CPU clock while keeping a constant 16 MHz clock available for ADC

conversion. This means that the CPU can run at a frequency of 32 kHz while the ADC maintains a 1 MSPS sampling rate while only drawing 150 μA .

Precision Analog

Many medical applications require the ability to read very small signals. For example, an ECG monitor has to capture the very small electrical signal generated by the heart muscle while removing the 50 or 60 Hz noise signals that are common in electrodes attached to the human body.

MCUs typically offer an 8- or 10-bit ADC. With its 12-bit ADCs, STM32 MCUs enable developers to achieve greater precision when measuring the weak signals typical of the human body. In addition, with sample rates up to 1 Msample/second, developers have the headroom to access even more accuracy through oversampling. This enables devices to operate in noisy environments and applications that could not be served by an MCU with only an 8-bit ADC.

Power-Efficient Communications

The STM32 architecture offers a wide range of interfaces,

including Ethernet (available on the STM32 F1, F2, and F4), USB (available on the STM32 L1, F1, F2, and F4), SPI, SDIO, and CAN. For applications that only transmit data a few times a day, interface current has minimal impact on operating life. However, for an application like a heart-rate monitor which may be constantly transmitting data, the efficiency of the STM32 L1's peripherals can provide substantial benefit. With integrated USB and/or Ethernet, developers can support connectivity at the lowest cost.

Some devices may need to support both USB and Ethernet. By abstracting the communications interface, the application won't need to know which type of link it is using. Developers can achieve this by creating a data in/out function that determines the actual data interface to be used in any particular operating circumstance and applies the appropriate communications APIs. Developers can also take steps at the application level to ensure power efficient transmissions. For example, collecting data for a short time and then bursting it will conserve power compared to continuous transmission.

For devices that will connect directly to the Internet and transmit personal health data, security is essential. This will impact power efficiency since security handshaking will increase the time the communications link must be active. The STM32 F2 and STM32 F4 minimize transmit processing latency with an integrated security engine that accelerates AES 256, 3DES, SHA-1, MD-5, and HMAC processing.

Using an off-the-shelf stack substantially speeds time-to-market. For example, the 40 standard APIs in the Micrium μ C/TCP-IP stack encapsulate more than 150,000 lines of code. This stack has been designed to minimize memory usage and so further reduce power consumption. Extensions to the stack are also available, such as SSL support.

Micrium is also introducing IPv6 over IPv4 support to its μ C/TCP-IP stack. IPv6 is an important enabling technology for medical applications. With the virtually unlimited number of IPv6 addresses available, individual devices can be assigned a unique IP address. This enables

devices and patients to be identified automatically and without interaction from users who may not be tech-savvy enough to register a device online. It also future-proofs devices to be able to be used in the years to come as IPv4 is phased out.

To further simplify device connectivity, ST offers the ST Healthcare Library that is certified for the USB Personal Health Device specification. This spec supports different subclasses of common medical devices and enables doctors and patients to connect seamlessly over the Internet. The library brings remote monitoring capabilities to every STM32-based design, as well as accelerates time-to-market. In addition, the library components have been tested and certified, enabling developers to bypass several certification tests with confidence that the stack will perform as required. The stack footprint is very small at just 9.2 K Flash and 2.9 K RAM for a typical thermometer application. A thermometer reference demo is available on the STM32 L1 evaluation board.

Real-time Integration

To enable a portable device to support a wireless interface with only 3AAA batteries, it is critical to be able to intelligently manage power. Using an OS like Linux, for example, significantly increases startup time from low power modes. In contrast, tight integration between the stack and a real-time kernel ensures that the radio can be turned on and off as quickly as possible.

Running a real-time kernel also helps simplify power mode management. Without a kernel, the typical embedded application manages tasks using a main loop. Effectively, the loop polls a series of tasks. Even if the device is not doing anything, the CPU still needs to poll tasks continuously to see if any of them need attention.

Because developers never know which tasks will be executed during any loop iteration, it becomes difficult to determine the optimal power saving mode in which to place the system after any particular task has completed since each mode offers different responsiveness. In addition, as the main loop increases in

size, managing power becomes more convoluted.

With a real-time kernel, the CPU returns to the IDLE task whenever none of the tasks need attention. Because no tasks are currently running, it is much more straightforward to manage power. When the IDLE task is interrupted, the system can quickly be powered up. This gives developers the ability to take the system context into account when selecting a power mode and clock speed that provides the appropriate level of responsiveness (i.e., how fast the system can wake to an active state).

Using the IDLE function to manage power also simplifies any migration of applications between STM32 processors. For example, tuning an application based on the STM32 L1 for power may require a different approach compared to an STM32 F0 or STM32 F4. By consolidating power management in a single routine, developers can be sure to be able to optimize efficiency without having to completely rewrite the application.

Designing your own real-time kernel is an option. However,

the use of a commercial real-time kernel like $\mu\text{C}/\text{OS-III}$ from Micrium provides a wide range of multitasking capabilities that make it easier to add new functionality to a system without impacting system reliability. For example, when a new feature like a GUI is introduced, it can be given a low priority. Thus, a critical system task such as driving the pump on a blood pressure monitor is always executed when it needs to be and not disrupted because a USB packet arrived at a critical moment. In this way, developers can extend the UI of a device, as well as its connectivity options, without negatively impacting reliability.

A real-time kernel also simplifies design and power management as more complex functions are added to a system. For example, while executing a time-intensive function without a kernel, developers need to manually check within the function if shorter, higher priority tasks need attention. With $\mu\text{C}/\text{OS-III}$, multitasking and priority management is automatically managed, and developers can break their system into separate


and distinct pieces without having to manually manage how these pieces interact. This also enables developers to more accurately measure power consumption during early design stages as well as verify where in their applications power spikes may be occurring.

Certification is a concern for many manufacturers of medical devices as well. Given that it is the complete system that must be certified, the Micrium $\mu\text{C}/\text{OS-III}$ kernel and $\mu\text{C}/\text{TCP-IP}$ stack are provided with source code and are fully documented so as to facilitate the certification process. Using the Micrium kernel and stack can simplify certification compared to an in-house kernel implementation that does not have the maturity and years of industry testing commercial code offers.

Rather than base a design on a specialized MCU that can serve only one price point, the STM32 F0 and STM32 L1, along with the rest of the STM32 families, provide a flexible architecture that can enable developers to leverage their code and other IP across an entire product line. In addition, code- and pin-compatibility between devices

allows for a nearly seamless migration between processors.

Designing with the STM32 architecture is also faster compared to 8- or 16-bit MCUs. Specifically, 8- and 16-bit MCUs tend to have a limited tool chain, especially when the MCU vendor is the only source for tools. In contrast, the STM32 architecture is based on the ARM Cortex-M architecture. As such, it has the largest ecosystem available for any MCU. These tools provide more run-time information and advanced debugging capabilities to ensure system robustness and power efficiency, as well as verify device operation to speed certification.

Creating portable medical devices requires an MCU architecture that provides both performance and power efficiency. With its tightly integrated architecture, STM32 MCUs offer a single-chip solution that balances cost and ultra-low-power operation. Combined with the powerful capabilities of production-ready software such as Micrium's $\mu\text{C}/\text{OS-III}$ kernel and $\mu\text{C}/\text{TCP-IP}$ stack, developers can bring reliable medical products to market faster. 

Accelerating Time-to-Market Through the ARM Cortex-M Ecosystem

By Reinhard Keil, Director of MCU Tools, ARM Germany GmbH
 Shawn Prestridge, Senior Field Applications Engineer, IAR Systems
 Laurent Desseignes, Embedded Software Marketing Manager, STMicroelectronics

Before ARM, the MCU market was fragmented with a multitude of vendors offering proprietary architectures and tool chains. Now, with open architectures like the Cortex-M, developers are able to base products on an entire architecture rather than have to tie designs to a specific MCU. The result is that designs won't be tied to an outdated architecture in just a few years. Rather, designs will always be current with the evolving Cortex-M architecture and extensible across a wide range of devices that can leverage the same application code. For example, ARM has recently shown its commitment to keeping the Cortex-M architecture relevant to embedded design as well as on the leading edge of technology by introducing the Cortex-M0 core. This core extends the Cortex-M architecture into

traditionally 8- and 16-bit applications while offering 32-bit performance.

The STM32 Ecosystem

ST is the MCU industry leader with the broadest portfolio of Cortex-M-based devices from a single company. The STM32 product line offers an extraordinary variety of options with more than 300 devices all based on the Cortex-M cores (M0, M3, and M4) to give developers unparalleled flexibility in finding the optimal MCU for their application (see Figure 1).

Part of ST's strength can be found in the software and tools available to developers for the STM32 architecture. ST provides a range of evaluation boards, including the STM32 Discovery Kits, which enable developers to evaluate each of the STM32 MCU product lines. The Discovery kits are

the cheapest and quickest way to discover the STM32 family while the evaluation boards are complete development platforms providing access to all peripherals in each STM32 product line and extension

headers to make it easy to connect a daughterboard or wrapping board for the target specific application. Development tools include in-circuit debuggers and eprogrammers, reference

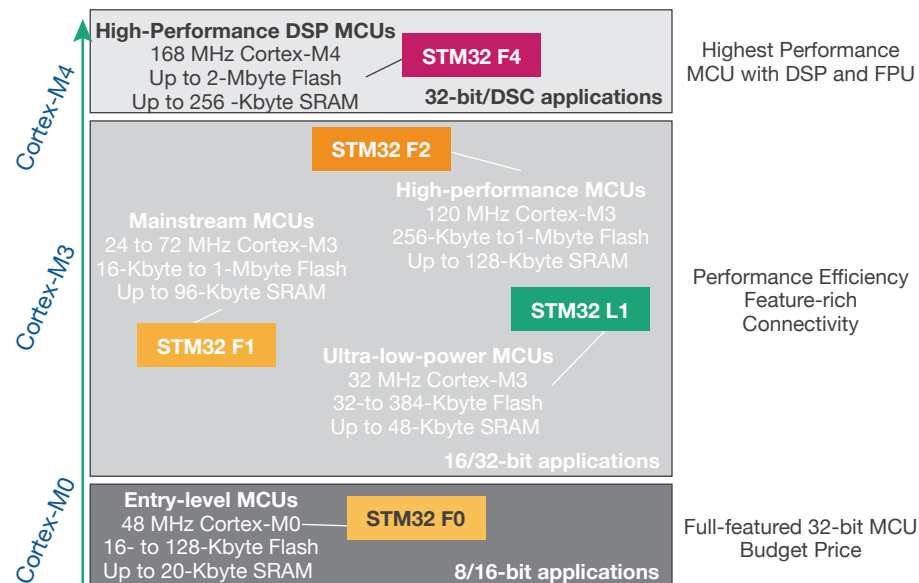


Figure 1 The STM32 product line offers more than 300 devices all based on the Cortex-M to give developers unparalleled flexibility in finding the optimal MCU for their application.

Leading Embedded Development Tools



The complete development environment
for ARM® processor-based
microcontroller designs



1-800-348-8051
www.keil.com



designs, and application notes. ST also offers a wide variety of firmware libraries, many of which are available at no cost, for implementing motor control, DMA, timers, interfaces, audio, LCDs, communications, GUIs, all standard peripherals, touch sensing, and in-application programming, to name a few.

ST also recognizes the benefit of working with other leaders in the industry to expand the tools and software available to developers. The STM32 architecture is based on the ARM Cortex-M architecture which has the most comprehensive ecosystem of tools, software, and engineering resources of any microcontroller in the world. ST has worked directly with a wide range of partners to create offerings that specifically accelerate the development of applications based on the STM32 architecture, thereby minimizing product development investment and speeding time-to-market.

The global ecosystem available for the STM32 architecture also offers many cost savings for developers:

- » The open architecture of the ARM Cortex-M cores gives developers access to the largest MCU ecosystem in the world

- » The vast diversity of the Cortex-M ecosystem ensures that developers can find the tools, software, and middleware they need to speed the design of nearly any application

- » With so many OEMs standardizing on ARM, the pool of engineers already experienced with the Cortex-M architecture is large, making it easier to find developers. Many colleges are teaching to the ARM architecture as well

- » Architectural and tool chain familiarity reduce the developer learning curve and speed time-to-market

The ecosystem for the STM32 consists of a complete range of tools and software specifically designed to speed design. The comprehensive tool chain is just the beginning. Developers also have access to the ARM CMSIS libraries, ST peripheral drivers, extensive middleware, and production-ready application software.

Developer's Foundation: The Tool Chain

There are several key components comprising an efficient development tool chain:

Compiler: Each compiler offers different strengths. Key factors to consider are code size and performance (typically measured in Coremarks). In addition, compilers like IAR Embedded Workbench automatically use the STM32 architecture to its best advantage, both in terms of leveraging silicon and middleware.

Assembler: Assemblers have become more of a specialty tool over the years given the performance and time-to-market advantages of coding in C. Today, if an assembler is used at all, it will likely be to code an application’s critical control loop.

Linker: The linker pulls together all the code required to create the final application. It needs to supply program and data information to the debugger.

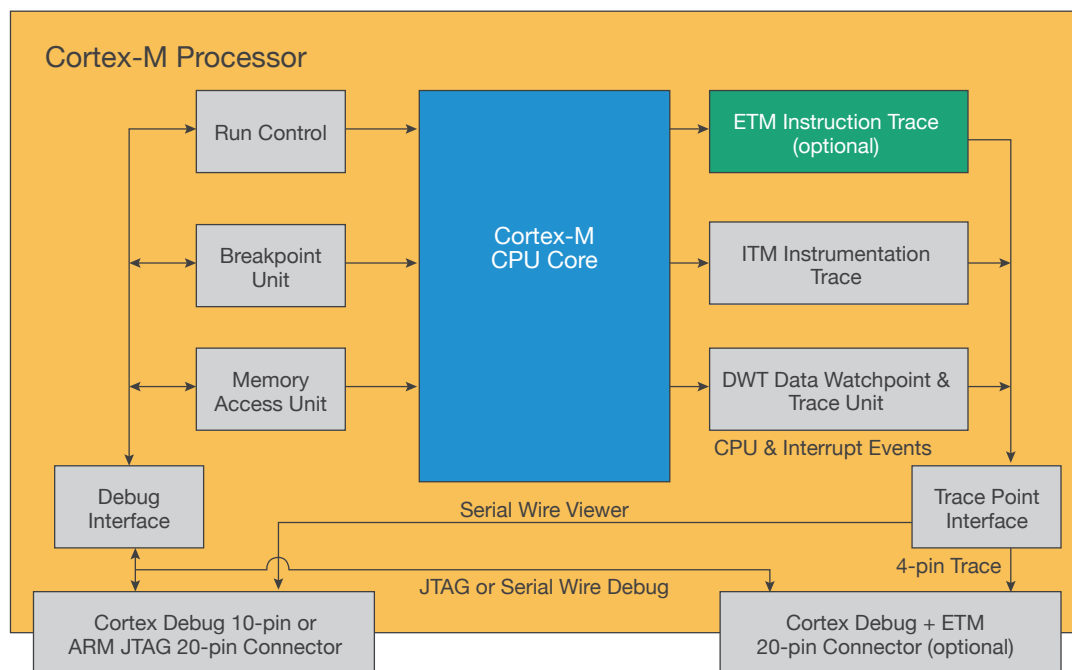
Debugger: The debugger is the tool most frequently used by developers. It provides visibility into systems to verify program operation and troubleshoot issues.

IDE: The Integrated Development Environment (IDE) defines the workspace and design flow. Some tool

chains support the option of replacing the IDE with one of a developer’s own choice. Given how much time a developer will spend with the debugger, using a debugger that is well-integrated with the compiler and middleware can substantially simplify troubleshooting. For example, a debugger that is RTOS-aware, USB-aware, TCP/IP-aware, etc. will facilitate faster problem

identification and resolution. The usefulness of the debugger, however, is limited by the ability of the MCU to provide visibility into its real-time operation. MCUs without integrated debugging capabilities will require developers to rely upon outdated and intrusive debugging techniques such as instrumenting code that can interfere materially with run-time execution.

The STM32 family of MCUs utilizes ARM’s Coresight Debug Technology to provide leading-edge troubleshooting capabilities (see Figure 2). For example, with streaming trace, developers can capture trace data limited only by the size of a PC’s hard drive. This enables full code coverage testing, timing analysis, and performance profiling. Because Coresight is implemented in hardware, it



Trace (ETM, ITM, DWT) not available on Cortex-M0

Figure 2 The STM32 family of MCUs utilizes ARM’s Coresight Debug Technology to provide leading-edge troubleshooting capabilities including streaming trace, full code coverage testing, timing analysis, and performance profiling.



Figure 3 The I-jet in-circuit debugging probe from IAR Systems offers seamless integration with IAR Embedded Workbench and provides a wide range of real-time debugging capabilities, including power debugging with $\sim 200 \mu\text{A}$ resolution at 200 kHz, for fine-tuning performance while achieving the highest power efficiency.

imposes no software overhead on the CPU and requires no external emulation hardware. In addition, breakpoints can be set and variable values changed while the system is running. Extensive trace records capture the program counter and read/write accesses while noting time delays and lost cycles. Statistical information about exceptions, interrupts, and events is also collected. Signals the MCU is processing can be monitored graphically, and Coresight provides instrumented trace capabilities enabling developers to write data to specific memory locations at run-time.

With proprietary architectures, typically only one or two debug platforms are available. With STM32 MCUs, developers can choose from more than a dozen vendors. In addition, the options include low-end tools for those with simple applications and tight budgets as well as high-end tools that provide sophisticated capabilities for accelerating development and simplifying verification and certification processes.

For example, IAR Systems offers its new I-jet in-circuit debugging probe for use with the STM32 architecture (see Figure 3). It offers seamless integration with

CMSIS provides a consistent software interface that simplifies software reuse across the entire STM32 product line of 300+ devices. Each of the CMSIS libraries reduces the learning curve behind using new software and tools to significantly accelerate design and lower development costs.

IAR Embedded Workbench and provides a wide range of real-time debugging capabilities. The I-jet is also capable of measuring target power consumption with $\sim 200 \mu\text{A}$ resolution at 200 kHz. This enables developers to debug applications in terms of power to fine-tune performance while achieving the highest power efficiency.

CMSIS: Faster Time-to-Market

Part of the value of the STM32 architecture is the Cortex Microcontroller Software Interface Standard (CMSIS). Created by ARM, CMSIS provides a consistent software interface that simplifies software reuse across the entire STM32 product line of 300+ devices. Each of the

CMSIS libraries reduces the learning curve behind using new software and tools to significantly accelerate design and lower development costs.

Development tools and middleware that are CMSIS-compliant can accelerate product design in multiple ways. For example, developers can create a proof-of-concept using a high-performance STM32 MCU with the maximum memory. This provides full flexibility during early design when the core product is still being defined. Once the design is proven, CMSIS makes it easier for designs to be transparently implemented on a different, more optimal STM32 MCU, even one based on a different Cortex-M core.

CMSIS comprises several parts (see Figure 4):

- » CMSIS-CORE is a standard API for all Cortex-M-based devices and abstracts key functionality
- » CMSIS-RTOS provides an API for RTOS integration with other tools and middleware
- » CMSIS-DSP is a collection of 61 digital signal processing functions that take advantage of the STM32's integrated floating-point and DSP capabilities. The library is designed for block processing to reduce interrupt overhead, optimize DMA utilization, and facilitate easy integration with an RTOS or kernel
- » CMSIS-SVD provides a System View Description for all peripherals to abstract actual peripheral implementations from application code and enable peripheral-awareness for debuggers

With the release of CMSIS 3.0, ARM is standardizing on an RTOS API that makes it easier for middleware components from different vendors to interoperate. This will facilitate the propagation of application-specific components, such as a TCP/IP stack, as well as enable the

development tool chain to be RTOS-aware. The CMSIS DSP library gives developers a substantial head start on the development of complex algorithms. It also allows developers to create complex application code that can be carried between MCUs

that may not have hardware-based DSP instructions. This means that developers can write code for the STM32 F2 and, when they compile it for the STM32 F4, automatically get the full advantage of the STM32 F4's DSP capabilities.

Accelerated Peripheral Configuration and Driver Design

One time-consuming part of product development is configuring an MCU's peripherals and then writing drivers for the application to utilize them. To facilitate faster application

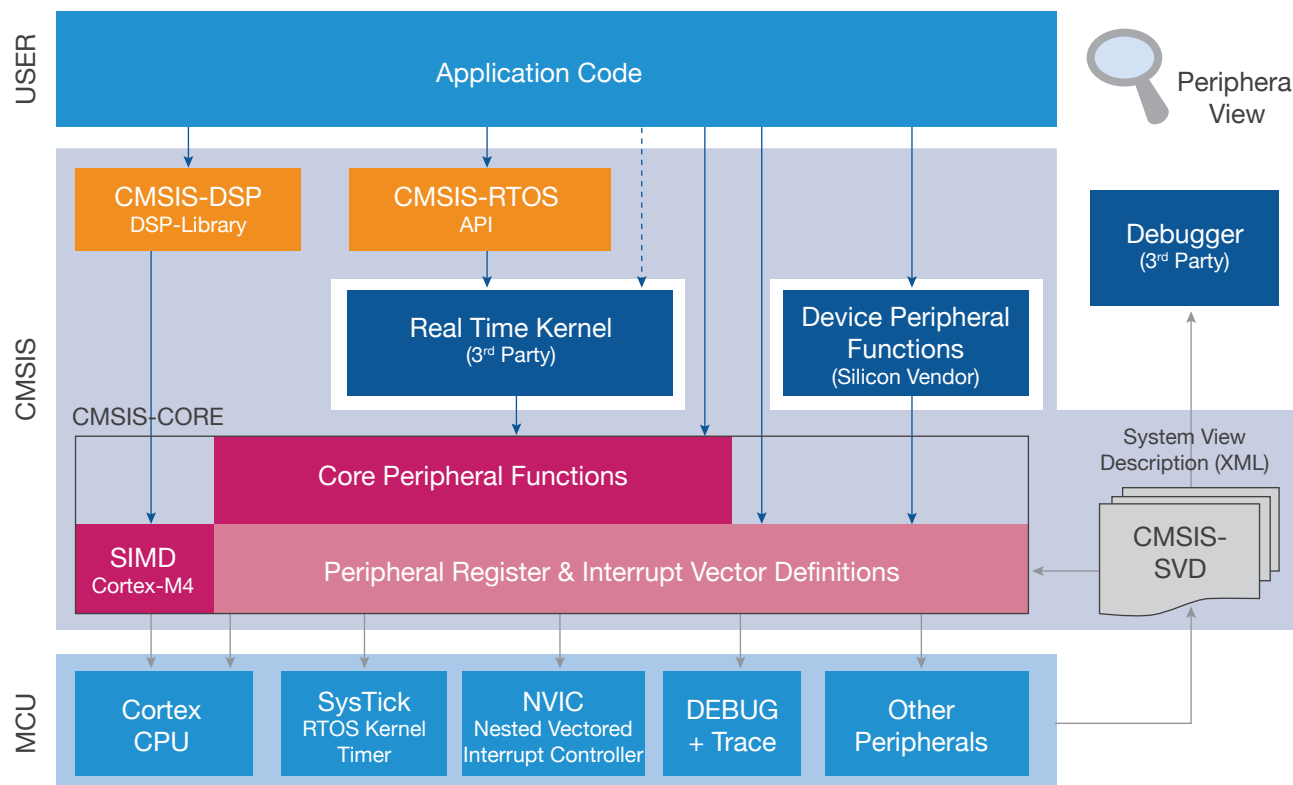


Figure 4 The ARM Cortex Microcontroller Software Interface Standard (CMSIS) provides a consistent software interface that accelerates product development as well as simplifies software reuse across the entire STM32 product line of 300+ devices.

design, ST supplies a complete peripheral library for use with all STM32 MCUs. Instead of writing to peripherals directly and locking code to a particular device, developers use APIs that abstract the use of peripherals. This approach offers several benefits:

- » **Faster time-to-market** as developers do not need to write peripherals drivers or debug them
- » **More reliable code** since the peripheral drivers supplied are mature and industry-proven
- » **Easy migration between STM32 devices**, even ones based on different Cortex-M cores. Because the peripherals, pin-outs, and code are the same or similar among the STM32 family, migrating between devices really is just a recompile
- » **Simple customization** as C source code is provided for all peripherals

Middleware: Production-Ready Software Solutions

One of the factors that substantially accelerates product development and time-to-market is the extensive range of middleware available for the

STM32 architecture, including:

- » Real-time kernels and operating systems (RTOS)
- » Stacks for all major communications protocols, including USB, TCP/IP, and Bluetooth
- » Advanced design and modeling tools such as those used for signal processing
- » Display/GUI solutions
- » Touch sensing
- » Java

The STM32 architecture has wide industry support, with ST's partners providing a multitude of optimized solutions for a diversity of applications, including audio, industrial, and motor control, to name a few. For example, developers have the option of selecting a full-featured RTOS for applications needing high reliability or a more simple kernel for a low-cost consumer device.

To accelerate design, however, the development tool chain needs to be able to integrate well with middleware offerings. IAR Embedded Workbench, for example, has hooks into the CSpy software debugger that

enables RTOS vendors to create drivers that make the debugger kernel-aware.

Effectively, this approach maximizes the flow of information available to developers to speed problem resolution. Since the kernel/RTOS is the primary interface between the application and middleware such

debugger can interpret packets and present them in a format that allows developers to debug at the link level. Complete visibility into the system also enables developers to more thoroughly verify program execution for both ensuring reliable operation and for certification processes.

One of the factors that substantially accelerates product development and time-to-market is the extensive range of middleware available for the STM32 architecture.

as communications stacks, it is important that the debugger understand how middleware is being managed by the system. When a debugger is RTOS-aware and USB-aware, for example, it can show developers directly into frame buffers and display protocol information in its native format.

Without visibility into the RTOS, developers would have to manually collect this information themselves, introducing potential error and delay to the design process. Rather than having to manually resolve headers to find the payload, the

IAR Embedded Workbench directly supports many middleware companies and nearly every hardware debugger. This enables developers not only to seamlessly take advantage of components from different companies but also select best-in-class components for their applications. IAR Systems is the world's oldest embedded C compiler company and understands the needs of developers to be able to choose the elements of the ARM ecosystem that are best for their application.

Alternatively, Keil’s MDK-ARM development system offers a comprehensive approach to embedded design by bringing together an integrated set of development tools, middleware, and debug hardware.

MDK-ARM is available in four offerings, from Lite to Professional, to match the varying needs of development teams (see Figure 5). Some of its components include:

- » Version 5 of the MDK compiler has recently been released and offers best-in-class performance with full support for the CMSIS libraries
- » The Keil µVision IDE provides a wide range of features from trace view with source code synchronization and editing to detailed peripheral debugging to comprehensive project management
- » RTX is a full-featured RTOS supporting multiple scheduling options, low interrupt latency, unlimited tasks, and a memory footprint of less than 5 KB. Full source code is available
- » Extensive middleware libraries optimized for the Cortex-M architecture enable developers to quickly add support for

CAN, USB Host/Device, TCP/IP, GUI development, and file system management to embedded applications. Intuitive configuration wizards simplify the use of middleware

- » The ULINK2 and ULINKpro debuggers support high-speed streaming trace for non-intrusive, real-time visibility into embedded systems
- » ETM Trace enables full code coverage testing and performance analysis

Frameworks and Libraries

Part of the STM32 value proposition is the availability of a tremendous amount of off-the-shelf software. Because of the huge volumes of ARM processors shipped, even niche markets are large enough to incent third parties to create specialized tools. Since the market can support software that is highly application-specific, developers have a greater chance of finding exactly what they need already available. Some of the application-specific libraries offered for the STM32 include audio, motor control, and industrial control.

Using a library with an STM32 MCU is a straightforward process:

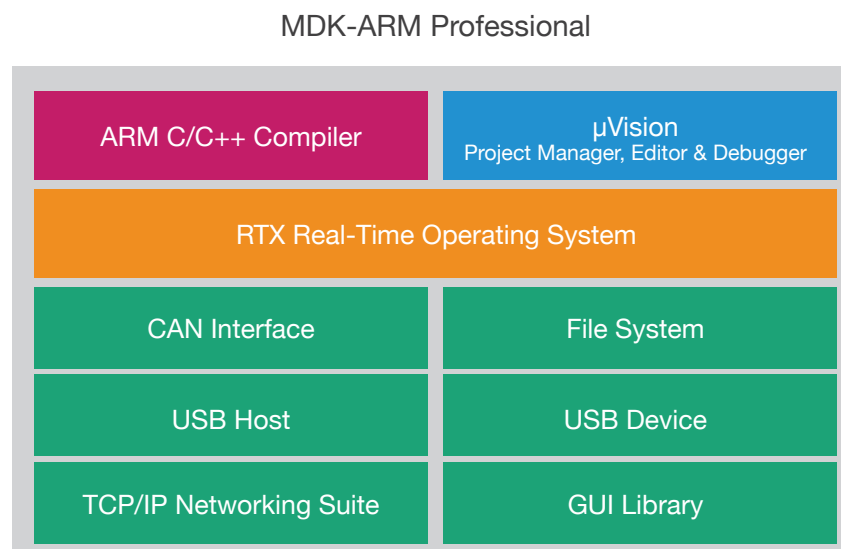


Figure 5 Keil’s MDK-ARM development system offers a comprehensive approach to embedded design by bringing together an integrated set of development tools, middleware, and debug hardware to match the varying needs of development teams.

just include it into the application workspace. Depending upon the compiler, each function will be recognized as a keyword and right-clicking will provide fast access to the function’s definition and parameters. Many compilers and linkers will also automatically pare out those parts of the library you are not using to conserve code space. Many libraries are provided as binary files created for a specific STM32 MCU and tool chain. Others are offered with source code to enable developers to customize code.

The availability of so much production-ready software accelerates design by enabling developers to work with code that provides the base functionality for their application. Rather than having to learn the STM32 architecture from scratch, developers can let the tool chain abstract low-level implementation details.

For example, IAR Embedded Workbench offers over 3000 example projects. With these, developers can load a project onto an evaluation board such

as the STM32 F4 evaluation board and see how a complete system has been implemented. These projects provide a working framework for a wide variety of applications ranging from simply blinking an LED to operating the LCD screen to creating a lightweight IP stack to implement a web server. Effectively, these projects give developers a template upon which to base their own design. The startup code gets developers into their design immediately, rather than forcing them to spend time figuring out how to initialize the MCU.

Combined with the CMSIS peripheral library, many of the projects can be transferred between devices with minimal reworking. Thus, developers can take projects built for another MCU altogether and quickly pull the core of the application over to the STM32 device of their choice.

The 80%+ Advantage

With the extensive variety of software, middleware, and reference designs available, developers can expect to substantially jumpstart their designs. Depending upon the application, it is not uncommon to find a combination of off-the-shelf software and middleware

within the STM32 ecosystem that provides 80% or more of the code required for a design. This enables developers to focus their design efforts on the last 20%, leading to faster time-to-market as well as a highly differentiated product.

This 80% figure is not unrealistic. For example, with the availability of off-the-shelf USB and TCP/IP stacks, a communications link truly becomes a component that can easily be added to a system rather than a subsystem that has to be designed and fine-tuned. In addition, there are many support resources available to assist developers in customizing code for applications that have a specific need. ST's support engineers and Field Application Engineers (FAE), for example, can provide valuable insight in overcoming a variety of design challenges.

With the rise of social networking, peer support has become another important element of an MCU architecture. The size of the ARM community is tremendous, and there are a variety of forums where engineers can provide peer-to-peer support. This support can be product-based, such as the support communities hosted by Keil, IAR Systems, and ST

for their own products. Free software, such as FreeRTOS, tends to have its own support network as well.

Many independent forums have also come into being in the last decade to assist engineers by hosting discussions about silicon manufacturers and their middleware partners. Engineers can search for specific MCUs and tools to hear how useful others in the industry find them. They can also post issues and share solutions.


Software the Way You Want It

ST embraces the benefits that an extensive ecosystem offers its customers. By giving developers the choice of software developed by ST, by a commercial third-party, or through open source, ST ensures that developers have the broadest variety of options available to match their application needs.

For example, with its cryptographic library, ST provides developers with security capabilities free-of-charge that have been 100% optimized for the STM32 architecture. ST also offers a number of other application-

specific libraries, including USB, motor control, and HDMI Consumer Electronics Communication (CEC).

ST has focused on working with numerous partners to ensure that there are many solutions available for the STM32 architecture. Their goal is that developers should be able to find whatever software and tools they need and, in many cases, have a choice between several vendors as well as between open and commercial versions. For example, developers can select from a range of MP3 codecs ranging from the open source Helix, ST's codec, and commercial codecs with extensions including mixers, equalization, etc. This allows developers to determine for themselves the balance between cost, support, efficiency, and time-to-market.

The STM32 family of 32-bit MCUs offers developers the most extensive and complete portfolio of Cortex-M-based devices. Combined with the comprehensive ecosystem around the STM32 architecture, developers are able to simplify development while bringing products to market more quickly and at the lowest cost. 

Introducing a Graphical User Interface to Your Embedded Application

By Gérard Bouvet, Marketing and Sales Manager, GeeseWare
Dominique Jugnon, Microcontrollers Development Tools Manager, STMicroelectronics

Smart phones and similar devices have redefined the way we interact with technology and created a new set of consumer expectations as to how a Graphical User Interface (GUI) should appear. The complex layout of Windows that was thought to define how a GUI should be has given way to streamlined interfaces that can give users access to all of a system's functionality on the smaller displays often used with handheld devices.

Manufacturers have recognized that touch-based GUIs can bring value to a wide range of embedded applications beyond consumer electronics, including industrial automation, appliances, meters, HVAC, security, access control, military, automotive, and infotainment devices. For example, traditional push-button interfaces have mechanical parts which can fail. Moving to a capacitive touch sensing display

enables not only a more robust interface but one that offers more flexibility and extensibility.

32-bit Processing

Adding a GUI to a system, however, is not like adding a few more buttons or controls to a device's front panel. With the nearly ubiquitous availability of touchscreens in mobile handsets, consumers have come to expect electronic devices of all types to have a sophisticated user interface utilizing 3D objects, perceived depth, animated transitions, textures, and complex background lighting. To create an intuitive interface that adds value as well as flair to an application, GUIs also need to support gestures such as tap, drag, fling, and slide that consumers are quickly learning to consider as a natural aspect of any touch-based interface.

Applications based on 8- and 16-bit processors simply don't have the horsepower to handle

even simple graphics. The availability of high-performance MCUs like the STM32 family that can provide complete GUI functionality with capacitive touch sensing on top of the primary application has been a key enabling factor in the proliferation of advanced user interfaces. For example, the STM32-F0 provides 32-bit processing at 8- and 16-bit pricing. For more graphics-intensive applications, the STM32-F2 and STM32-F4 provide sufficient Flash to store large graphic images. With devices up to 168 MHz/210 DMIPs, the STM32 family is also fast enough to provide the responsiveness consumers are used to from GUI-based devices.

However, as the cost of hardware has dropped, software complexity has continued to increase. In fact, application software has become the leading development cost in embedded

systems, both in terms of money and time-to-market. To maintain their competitive edge, companies need to be able to introduce complex GUI functionality while tightly controlling software development cost. Achieving this goal requires access to a GUI framework, the ability to define the look and feel in Java rather than C, rapid prototyping capabilities to enable consumers to provide feedback while target hardware is still being designed, and tools optimized for the tight memory and processing constraints of the typical embedded application.

GUI Framework

There are two primary phases to designing a GUI. The first is the creation of the underlying software code that provides basic UI functionality. Once this GUI framework is in place, the look-and-feel of the GUI must be designed. To keep system cost

down, developers must minimize the expense for both of these processes as well as eliminate any unnecessary design delays.

In the past, UIs for embedded systems were designed specifically for the hardware on which they were going to be run. With increasing pressure to shorten product design cycles, IP

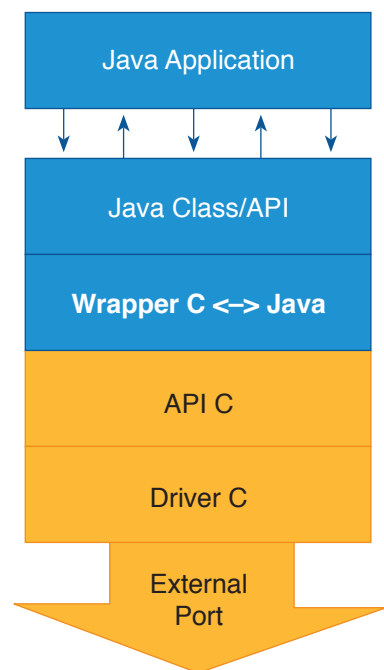


Figure 1 Abstracting the hardware through a Hardware Abstraction Layer (HAL) frees the GUI framework from handling specific low-level details, leading to faster code development and greater reusability.

reuse has become an important consideration in UI design. Ideally, developers need to be able to carry a UI across different products using MCUs that may be from different families as well.

To achieve this, GUI application code is abstracted above the hardware. A Hardware Abstraction Layer (HAL) handles specific low-level details such as how graphical data is stored in memory and transferred to the display (see Figure 1). By interacting with the HAL using APIs, the GUI application code becomes a framework that can be ported across an MCU family with minimal rewriting required.

Creating an extensible GUI framework is an involved process. The HAL enables developers to build the framework using a language other than assembly, leading to faster code development and greater reusability. Designing the framework using C, however, can still require substantial development resources.

Ideally, rather than design a framework from scratch, developers can use off-the-shelf software to minimize development investment. With the right tools, the GUI design cycle can be

shortened from months to weeks. GWStudio™ from GeeseWare, for example, is a Java Framework with a wide array of preexisting GUI libraries providing a complete human-machine interface (HMI) development environment. Its Java engine based on IS2T MicroEJ® technology is specifically optimized for embedded applications that have limited memory, constrained peripherals, restricted network connectivity, and low power consumption requirements.

Java brings many advantages to GUI-based design compared to working in C. Java was designed to facilitate GUI creation with an emphasis on reuse. In addition, its flexibility ensures a simple revision process that enables developers to quickly implement changes to existing designs. With the availability of Java for embedded applications, developers can leverage the benefits of Java in many applications:

- » Improved code portability and reuse
- » Accelerated development up to 3-5 times faster than working in C
- » Equivalent performance to C-based designs; i.e., less than 1 ms responsiveness for machine-to-machine (M2M)

processes (i.e., Ethernet) or touchscreen latency

- » Greater functionality in a smaller footprint
- » Higher reliability and robustness by eliminating manual management of memory and exceptions
- » Operating system independence
- » Large development community

Note that even though the GUI is written in Java, the main application can be based on C. This enables developers to introduce a GUI to an existing design without having to rewrite the application.

Even with the framework in place, however, only half the job is done. Now the look and feel of the GUI needs to be designed.

Intuitive Look and Feel

Developing an effective GUI can be one of the most challenging aspects of system design. GUI design involves much more than simply arranging icons on a screen. To be intuitive, a user interface has to anticipate how a variety of different types of people are going to use the device. However, it can be very difficult to know how an end-customer is going to use a device from

GeeseWare™

**Straightforward
TouchScreen
Applications
Development**

GWStudio

LSI
Ultimate Solutions, Inc.
<http://www.ultsol.com>

<http://www.geeseware.com>

IS2T **ST** **ARM**
CONNECTED

**a One-Stop Shop for
Java Embedded Applications**

a developer's isolated position behind his or her lab bench.

For example, while it may be logical to a developer to group functions based on what part of the system they impact, users will interact with devices based on what they want it to do. If the function a person wants to use most frequently is buried under a series of icons, the overall experience will be frustrating. The UI has become the core factor determining the user experience. In today's market where consumers have become quite sophisticated, a poorly designed GUI can mean failure for a product regardless of its other qualities.

The reality is that developers don't always know how perspective users will try to interact with a system. Ideally, the hardware should not come between users and how they want to use the device. In addition, to keep the interface from being cluttered and difficult to navigate, the screen needs to display as little information as possible while still displaying enough data to allow the user to easily and quickly make decisions. Tappable UI objects/elements must also be of a minimum size but yet also comfortable to select.

The placement of icons and ordering of GUI elements is, at least at first, a fairly arbitrary process. However, a slider may end up being in an inconvenient location or be improperly sized for reasons that couldn't be predicted during initial design stages. This won't be clear until users are actually given a chance to use the interface.

Designing an effective GUI involves many such intangible considerations that require direct feedback. With limited screen real estate, the UI must be selective and display only content that is relevant to the choices a user is currently considering. The application's main function should be accessible quickly upon start-up and always simple to return to. The final test for an intuitive GUI is that it must be obvious to users how to use it efficiently without a steep learning curve or more than a few minutes of training.

It may take extensive testing with users for developers to understand how the GUI should be laid out. This likely won't happen after bringing in just a single focus group but rather will involve many rounds that iteratively improve the ease-of-use of the interface. The design schedule needs to take into

consideration that the GUI may need to be redesigned several times. The sooner accurate user feedback can be incorporated into the GUI design, the more confidence developers can have that major changes will not be required after significant engineering resources have been invested in implementing the design.

GUI Testing

The iterative aspect of GUI design is an important consideration when selecting a GUI toolset. The speed and ease with which developers can modify an existing GUI will determine how many design iterations the schedule will allow and, consequently, how well the GUI will capture actual user behaviors.

Any testing process needs to enable stakeholders and end-users to provide timely input into GUI design, preferably as early in the design cycle as possible. This means that developers need access to rapid prototyping capabilities before target hardware is available. The testing process should facilitate the capture of user behaviors and generation of use cases. To achieve this, GUI tools must accelerate design to shorten the time between test iterations.

Traditionally, developers have created simulated environments for users to test. Often these “wireframe” simulators are independent tools that allow developers to put together a GUI but not necessarily one that accurately reflects how the final product will look or operate. For example, because the simulator is running on a high-speed PC, screen updates can be near instantaneous. Unless the simulator is able to emulate the MCU that will be used in the actual product, developers will be unable to verify whether the system is responsive enough to satisfy users. In fact, feedback from such testing may actually mislead developers and result in launch delays.

To ensure that the simulator matches how the interface will operate in production hardware as accurately as possible, the simulator needs to emulate the operation of the target MCU. Developing an embedded GUI on a PC capable of accurate emulation offers several benefits to developers (see Figure 2). In addition to speeding testing by eliminating the need to download new firmware to a target, the simulator provides several analysis capabilities to facilitate optimization and debugging:

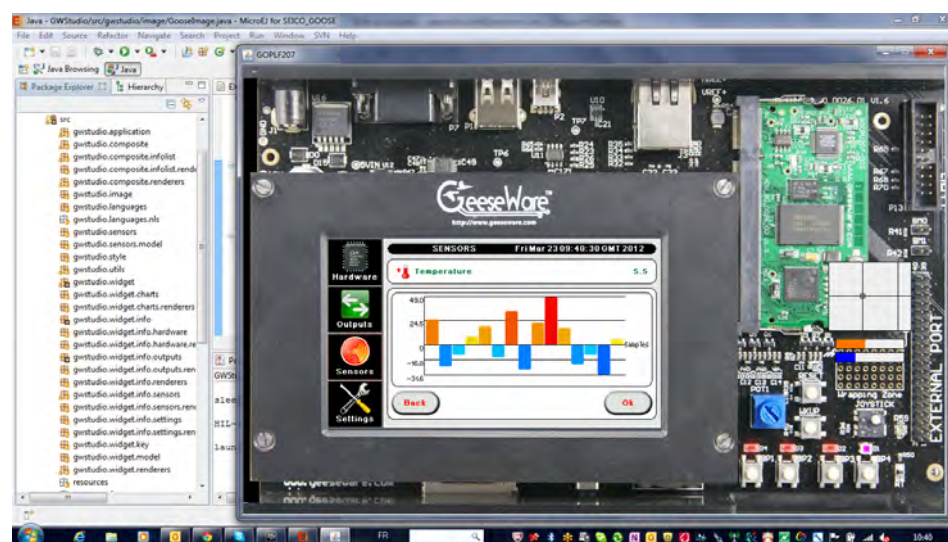


Figure 2 GUI design tools like the IS2T MicroEJ® simulator that is part of GWPack™ from GeeseWare emulate the target MCU to ensure consumer testing matches the operation of the production GUI as closely as possible.

- » Static and run-time analysis of timing and memory footprint
- » Functional code coverage
- » Task profiling and scheduling

Ideally, testers need to have as realistic an experience as possible. This means working on a small screen the same size as the one that will be used for production. When running on a PC, the user may need to use a mouse rather than be able to touch the device screen. Even if a touchscreen is available, it is likely the wrong size or a monitor that the user cannot hold in his or her hand.

To enable realistic testing, GeeseWare offers its GWPack™ development system. The GWPack™ includes a standalone and small form factor board based on either the STM32-F2 or STM32-F4 that is prequalified and production-ready. With a 4.3” resistive or capacitive touchscreen, 10 ms response time, and access in Java to all of the STM32 architectures peripherals and interfaces of the pack, the GWPack gives developers a fully operational Java platform upon which to carry applications from proof-of-concept to production faster than has been possible before.

A complete framework is provided as part of GWPack™ that allows developers to dive immediately into interface development rather than writing low-level drivers and managing system resources to develop the Board Support Package (BSP). The IS2T MicroEJ® simulator for GeeseWare GWPack further allows developers to simulate their designs on a PC while emulating the operation of an STM32 MCU. Key features include:

- » Built-in display tree and events management
- » Window and clickable area definition and management
- » Support for single and two-finger gestures
- » Driving of low-level events to closest tree nodes
- » Sub-level display and event management at the node level
- » Fully customizable
- » Support for TCP/IP, UDP, and HTTP via the Java framework
- » The ability to reference all STM32 peripherals directly in Java

Because events are object dependent, developers can easily

define—and redefine—interface operation, thus accelerating the ability to implement feedback into designs.

With the GWPack™, developers can begin GUI development immediately with a low initial investment. The standalone board is also available in preproduction volumes to enable manufacturers to quickly mock up a large number of systems that can be put out into the field to capture customer feedback while target hardware is still being defined and built. Finally, for applications with volumes under 10,000 production units, the board is a cost-effective, off-the-shelf alternative to designing your own.

For volume applications, GWPack™ enables manufacturers to complete a proof-of-design before investing in implementing a design in hardware and software. This ability to design the GUI in parallel with hardware can substantially reduce time-to-market.

Critical Design Constraints

Applications like industrial control, metering, home automation, medical, and smart energy don't have the memory or processing

capabilities of smart phones. As a consequence, they need a GUI that is specifically designed to minimize processing requirements and memory footprint. For example, to implement a GUI in an embedded Linux environment can be quite expensive: 32 MB RAM + 8 MB Flash for the Linux OS plus graphics libraries push Flash requirements up to 50-60 MB, adding on the order of \$100 to system cost.

The framework supplied with GWPack™ keeps system memory requirements under tight control: a full GUI can fit in just 128 KB RAM and 1 MB Flash. In addition, the framework does not require a real-time operating system (RTOS) or kernel to operate. For applications with more intensive graphical requirements, more Flash may be required.

The STM32 architecture, with its 32-bit bus, DSP and FPU capabilities, and multi-layer bus fabric supporting simultaneous data transfers provides more than enough processing capacity to handle the GUI, application, drivers, touchscreen, and communication ports all with a single chip. With its broad portfolio of MCUs based on the ARM Cortex-M0, M3, and M4

cores, the STM32 architecture enables developers to select a processor with the optimal blend of performance, memory, and peripherals for their application. ST also provides a variety of tools to accelerate the development of general-purpose Java-based systems with the ST Evaluation Kit.

Touch-based GUIs can bring substantial value to a wide range of applications by bringing the look and feel of products up to date while also by capturing more value through the user interface. In addition, they can improve system robustness by eliminating mechanical components while increasing device flexibility to modify the UI over time to introduce new features without redesigning the enclosure.

With the STM32 family of MCUs and GUI design tools such as those from GeeseWare, manufacturers are able to cost-effectively introduce next-generation GUIs to new designs as well as to legacy products built on 8-, 16-, and 32-bit MCUs. With the ability to design a fully-functional system in weeks instead of months, developers can accelerate GUI design to enable a dramatic reduction in time-to-market. 