




# STM32 Journal

Volume 1, Issue 1



## In this Issue:

- » Selecting the Optimal MCU for Your Embedded Application
- » Maximizing Performance for Real-Time Systems
- » Designing for Low Power
- » Simplifying Design for Accelerated Time-to-Market
- » Designing Efficient Connectivity
- » Accelerating Next-Generation Design Through IP Reuse



# Navigating Next-Generation Design

By Nicholas Cravotta, Technical Editor

## Table of Contents

- 2 Editorial
- 3 [Selecting the Optimal MCU for Your Embedded Application](#)
- 12 [Maximizing Performance for Real-Time Embedded Systems](#)
- 20 [Designing for Low Power Applications](#)
- 27 [Simplifying Embedded Design for Accelerated Time-to-Market](#)
- 35 [Designing Efficient Connectivity](#)
- 42 [Accelerating Next-Generation Design Through IP Reuse](#)

A great deal has changed in the thirty years I've designed and written about embedded systems. When I started, writing code was a straightforward, linear process. You told the CPU to do something, and then you waited until it was done.

Over the years, I've watched as the MCU has evolved from a simple CPU to an efficient network of integrated processors working in parallel. Each accelerator or coprocessor operates independently of the CPU, enabling the simultaneous processing of an incredible amount of data. In an MCU like the new STM32 F4 from ST, these integrated processors are combined with a multi-layer bus interconnect and multiple DMA engines to provide tremendous processing capacity.

Development tools have evolved as well. Today developers can profile application code non-intrusively to focus their optimization efforts. Similarly, they can accurately measure power consumption while an MCU is switching between active and low power modes, even to the point of showing developers

which line of code is responsible for a spike in power consumption.

You can see the results of this evolution in designs like the Stanford Xenith solar vehicle shown on the cover of this issue. Stanford selected the STM32 for several of its more complex subsystems because of the architecture's extensive peripheral set and performance. In addition, the team found that STM32 MCUs are so energy efficient that there is virtually no operating cost in terms of power to using them. The integrated development environment also makes it easy for new team members to immediately begin contributing.

For example, a single STM32107RB monitors the voltage of the Xenith's 35 cell groups, measures the temperature and current, and performs critical operations such as controlling the flow of power through the vehicle. Four STM32F205RBT6s track the maximum power point of the solar array to optimize power density and efficiency. This process involves synchronous rectification which requires accurate interrupt

handling and FET switching at very specific times. Two STM32107RBs manage communication between the driver, the vehicle, the motor controller, and the rear wheel steering system; they also accurately read encoders with minimal overhead. Several STM32F103C8T6s are used to manage ancillary systems such as lighting, telemetry, and tire-pressure monitoring. You can see the Xenith in action [here](#).

We live in a fast-changing world, and the only way to keep up is to find silicon and tools that evolve as quickly. As you can see with the Stanford solar vehicle, a single architecture like the STM32 is versatile enough to serve across many diverse applications using the same toolset. This is no accident.

In this issue, we'll explore the current state-of-art in designing embedded systems. Whether you're interested in optimizing performance, minimizing power, connecting to the network, or designing for reuse, you'll find it here.

Enjoy! 

# Selecting the Optimal MCU for Your Embedded Application

By Alec Bath, Applications Engineer, STMicroelectronics

Innovation in silicon technology has escalated over the past decade with the availability of MCUs which combine a powerful processor with architectural enhancements, advanced acceleration engines and specialized peripherals. As a result, embedded applications have become more complex with each new generation. For example, many MCUs now have DSP-type instructions to support high-performance signal processing, USB and Ethernet data rates are magnitudes of order faster, and low-power operation is no longer just an option but rather an essential design factor that must be considered early in the design process.

The only way to stay competitive is to build upon an MCU architecture that is not just flexible but continues to evolve over time by integrating new functionality that increases application performance while

lowering overall system cost. Selecting an MCU that offers only limited performance, memory, and peripheral options will result in a system that needs to be completely redesigned in only a few years. By choosing an MCU with a broad roadmap of software- and pin-compatible devices with a mature tool chain and an extensive support ecosystem, developers can be sure that their designs will not only be cost-effective but be able to carry them well into the future.

## So Many Choices

When an engineer begins looking for a new architecture, there's usually a strong need driving the change. Perhaps there isn't enough performance with the current architecture, even with the highest-end family member, or the memory and peripheral options don't offer the right mix. Once the design is opened for review, it's worth looking beyond just what is

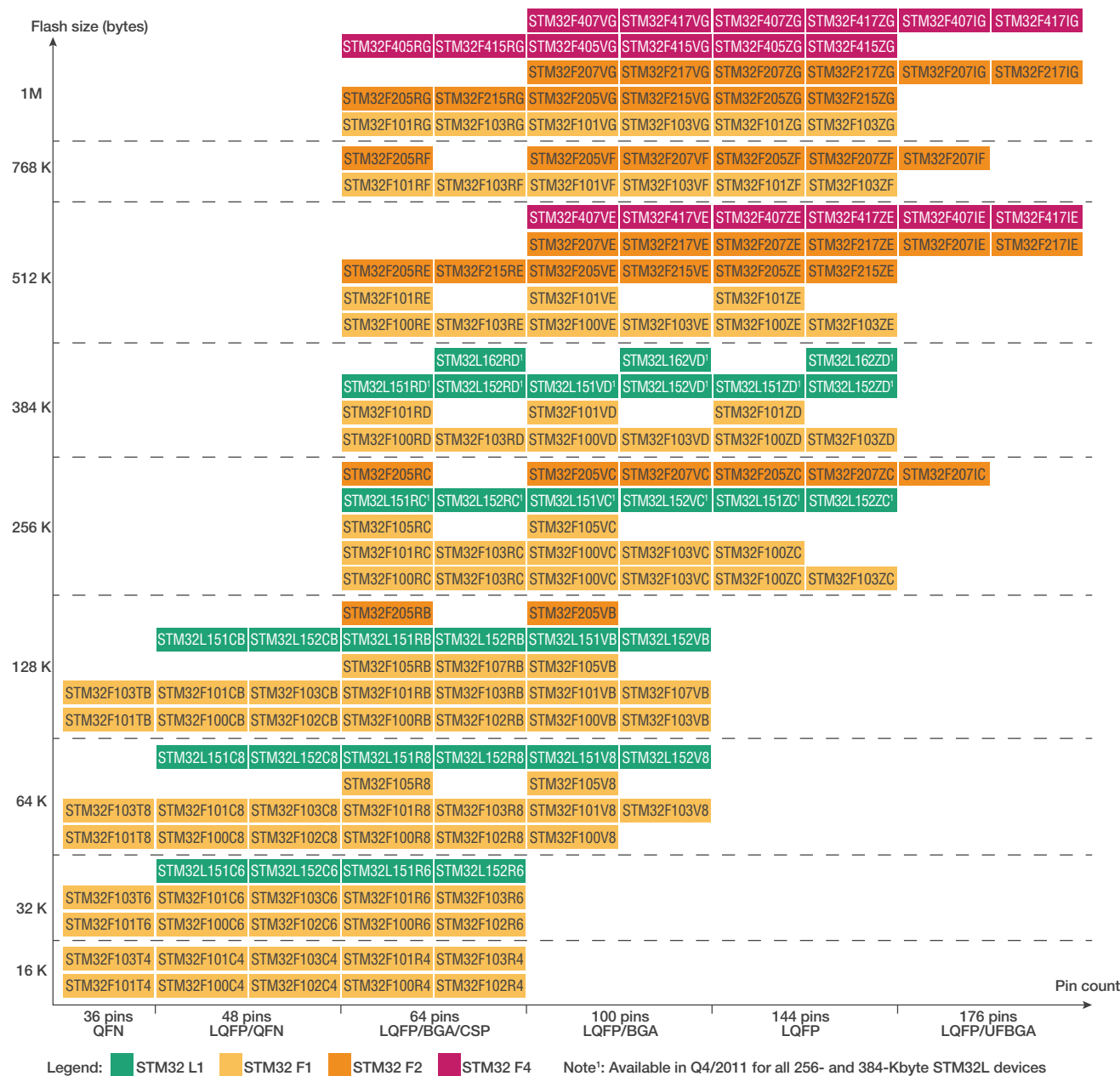
needed at the moment to see what else a new MCU family can offer to improve the value a product provides customers. Introducing an advanced communications peripheral such as USB, for example, may open new markets without negatively impacting system cost. As a consequence, the selection of a new MCU platform directly determines what future products can be designed as well.

For the engineer starting a new design or needing a new architecture with more capacity for an existing design, the extraordinary variety of MCU options can seem overwhelming. ST, for instance, offers more than 250 different devices in its STM32 product line alone. Perhaps surprisingly, selecting the optimal processor for an application is actually easier today than it ever has been.

In the early days of electronics design, developers were

limited to a short list of MCU architectures, each offering a variety of Flash, RAM, I/O, and UART options. Right-sizing an MCU to an application could be a difficult estimation. For example, if the estimation of the amount of Flash required to hold program code was too low, this could force a developer to switch to another processor family late in a product's design cycle and require a redesign that delayed time-to-market and increased cost substantially.

Today's MCU architectures seem to provide an even more difficult challenge to estimate code size and performance. Specialized DSP-specific instructions, for example, both accelerate signal processing tasks and reduce the amount of code required to complete these tasks. Advanced DMA controllers, for example, can significantly impact performance by managing the data flow of an application in



**Figure 1** The STM32 family offers more than 250 code-compatible MCUs, giving developers unparalleled flexibility in right-sizing an MCU to their application.

the background. Similarly, a wide range of dedicated accelerators implemented in hardware are available to speed processing of all kinds of data, including cryptographic security, CRC calculations, and communications protocols, to name just a few. The availability of application-specific peripherals also increases performance through intelligent management of peripheral data and functions to further offload the main CPU.

However, the ability to accurately estimate application performance and memory requirements before any code has been written is no longer as critical an exercise as it once was. All 250+ STM32 MCUs are not only code compatible, more importantly the integrated peripherals and system functions are compatible across the entire line. This gives developers unparalleled flexibility in right-sizing an MCU to an application (see Figure 1). If more performance is required, there are numerous choices available further up the product line. Likewise, if an application can be implemented using fewer cycles, another MCU further down the product line can be

used instead without any need to recode the application. From this perspective, the extended variety of options within the STM32 family actually makes it easier to select the ideal MCU for an application.

This variety of options also enables manufacturers to leverage the same architecture and code base across an entire product line. The STM32 F1 Value Line, for example, provides the right mix of peripherals, memory, and performance for low-end applications where price is tantamount. For a high-end version of the same system, the STM32 F4 provides the range of peripherals needed for a full-featured device backed with the processing capacity and memory to support these features. With complete compatibility between devices—code, pin, peripheral, and system functions—manufacturers can reuse their existing code base and hardware designs thereby reducing the development cost of next-generation devices as well as significantly accelerating time-to-market.

## The ARM Advantage

When evaluating an MCU architecture, it is useful to look at the entire design platform being offered. Recently, EETimes conducted a survey asking engineers what they considered the most important factors in selecting an MCU. The cost of the device was third. Performance ranked second. The most important factor: development tools.

Given the software complexity of today's embedded systems, software development represents a significant portion of the cost of a system. Software also impacts time-to-market more than any other design factor. From this perspective, it is not surprising that development tools are the most important consideration to developers.

The various ARM cores have the broadest support and tools ecosystem compared to any other MCU architecture. Tools are available from many different suppliers, ensuring a good range of tools that are easy to use. With so much competition, these tools have to be superior to survive. In addition, tool suppliers further differentiate themselves by

Given the software complexity of today's embedded systems, software development represents a significant portion of the cost of a system...from this perspective, it is not surprising that development tools are the most important consideration to developers.

offering tools designed for specific market segments. As a result, these tools provide greater functionality than a proprietary MCU supplier can supply on its own.

The ARM Cortex-M has also become the standard MCU architecture in many application markets. Its architecture has been field-proven and is well-established with an extensive tools ecosystem and widespread support from industry players. Designed from the ground up to meet the real-time performance and memory requirements of embedded applications, the Cortex-M core offers:

### **Integrated Interrupt Controller:**

The Cortex-M architecture integrates the interrupt controller rather than requiring silicon manufacturers to add their own as they have to with the ARM7 and ARM9 cores. This results in faster interrupt handling and more deterministic application response.

**Single Instruction Set:** The ARM7 and ARM9 cores tried to address conflicting performance and code density requirements by interworking the architecture's original 32-bit instruction set with the 16-bit Thumb instruction set. This approach required developers



**STM32<sup>®</sup> F4**

**World's highest  
performance  
Cortex<sup>™</sup>-M MCU  
168 MHz/210 DMIPS**



**STM32<sup>™</sup>** Releasing your **creativity**

For further information, visit [www.st.com/stm32f4](http://www.st.com/stm32f4)

to manually use a special subroutine branch instruction to switch between instruction sets based on whether they needed performance or code density at the moment. It also introduced design complexity and forced developers wanting to work in C to have to involve themselves with lower level implementation details. The Cortex-M architecture automatically blends the benefits

of performance and code density with the Thumb2 instruction set, freeing developers to focus on the application.

**Better computational capabilities:** The Cortex-M introduces advanced computational capabilities— including a single-cycle 32-bit multiply, hardware divide instructions, DSP functions, and saturated math, to name a few—to support the various

types of complex signal processing and computational analysis many embedded applications require today.

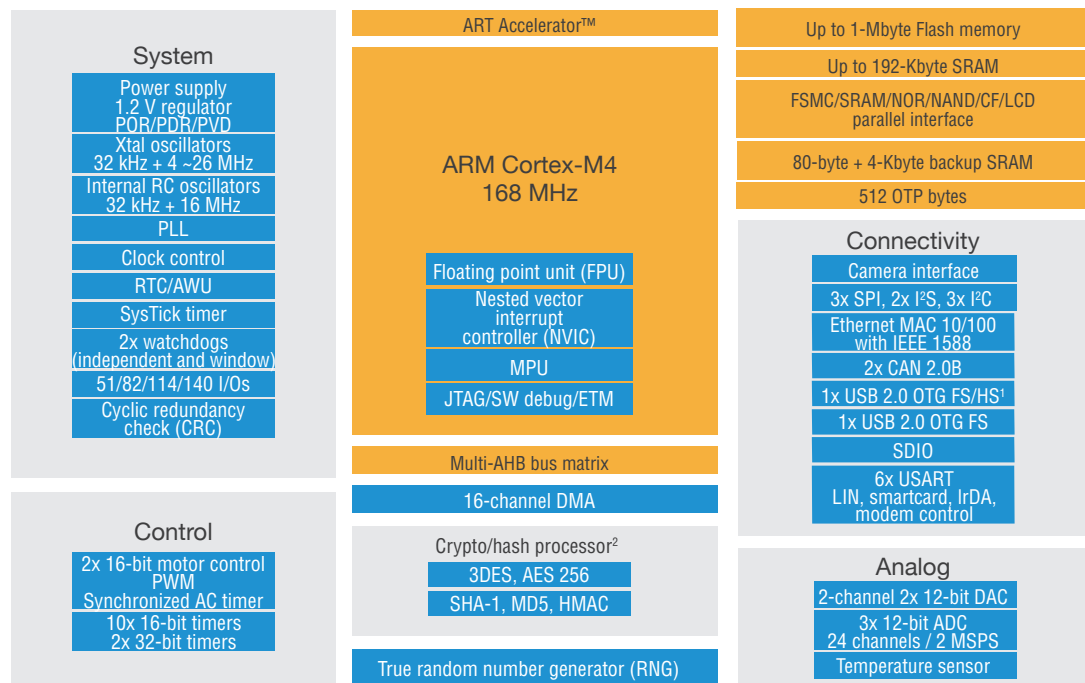
**RTOS:** The Cortex-M architecture has been designed to support real-time operating systems. For example, privilege modes enable kernel-level schedulers to guarantee real-time responsiveness.

## The STM32 Architecture

The ARM Cortex-M architecture provides an excellent foundation for embedded design. However, performance and reliability are not determined by the CPU architecture alone. For this reason, ST has expanded its STM32 MCUs beyond the base Cortex-M architecture with a variety of integrated peripherals to create a wide range of devices that optimize performance, memory, and cost for nearly every embedded application (see Figure 2). Combined together, these peripherals provide significant benefits to manufacturers:

**Reduced BOM:** Some of the STM32’s integrated peripherals are core components that embedded systems require, such as a real-time clock, internal oscillators, and supervisor functions. While individually these components are relatively inexpensive when implemented externally, their combined integration within the MCU results in a substantially lower bill of materials (BOM), smaller form factor, and simplified board layout.

**Increased Performance:** Multiple DMAs, integrated connectivity for USB OTG and Ethernet, and application-specific peripherals



**Figure 2** 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.

like hardware-based encryption all offload CPU processing to increase overall system efficiency.

**Complete Compatibility:** Developers can easily migrate designs between different STM32 MCUs as all STM32 MCUs are code-compatible. Most STM32

devices are also pin-compatible, simplifying hardware redesign as well. Furthermore, STM32 MCUs are also peripheral and system function compatible, meaning that even migrating low-level code is a seamless and transparent process.

**Unparalleled Accuracy:** By integrating the analog signal chain – including high-precision 12-bit ADCs and advanced cascadable control timers that run at full core speed – precise sampling, control, and timing functions are possible.

**Enhanced Safety:** Peripherals such a windowed watchdog and automatic clock switchover circuit are essential for the design of products that must meet high-reliability standards, including consumer electronics, white goods, and industrial applications.

**Simplified Design Through Flexible Voltage:** STM32 MCUs support a range of supply voltages, as wide as 1.65 to 3.6 V for some devices. This allows the system to operate from a single battery without an external regulator. In addition, most GPIOs are 5 V tolerate for compatibility with industry standards.

*(For a detailed exploration of how the high-level of integration of the STM32 architecture simplifies application design, see page 27.)*

## Optimized for Your Application

ST was the first major semiconductor company to bring the Cortex-M architecture to market and has the largest portfolio of devices available from any company with core speeds from 24 to 168 MHz and memory options ranging from 16 KB to 1 MB of Flash. ST has developed four separate series within the STM32 family that focus on different applications

### 4 product series

Common core peripherals and architecture:

Communication peripherals: USART, SPI, I <sup>2</sup> C
Multiple general-purpose timers
Integrated reset and brown-out warning
Multiple DMA
2x watchdogs Real-time clock
Integrated regulator PLL and clock circuit
External memory interface (FSMC)
Dual 12-bit DAC
Up to 3x 12-bit ADC (up to 0.41 μs)
Main oscillator and 32 kHz oscillator
Low-speed and high-speed internal RC oscillators
-40 to +85°C and up to 105°C operating temperature range
Low voltage 2.0 to 3.6 V or 1.65/1.7 to 3.6 V (depending on series) 5.0 V tolerant I/Os
Temperature sensor

STM32 F4 series—High performance with DSP (STM32F405/415/407/417)

168 MHz Cortex-M4 with DSP and FPU	Up to 192-Kbyte SRAM	Up to 1-Mbyte Flash	2x USB 2.0 OTG FS/HS	3-phase MC timer	2x CAN 2.0B	SDIO 2x I <sup>2</sup> S audio Camera IF	Ethernet IEEE 1588	Crypto/hash processor and RNG
------------------------------------	----------------------	---------------------	----------------------	------------------	-------------	--	--------------------	-------------------------------

STM32 F2 series—High performance (STM32F205/215/207/217)

120 MHz Cortex-M3 CPU	Up to 128-Kbyte SRAM	Up to 1-Mbyte Flash	2x USB 2.0 OTG FS/HS	3-phase MC timer	2x CAN 2.0B	SDIO 2x I <sup>2</sup> S audio Camera IF	Ethernet IEEE 1588	Crypto/hash processor and RNG
-----------------------	----------------------	---------------------	----------------------	------------------	-------------	--	--------------------	-------------------------------

STM32 F1 series—Connectivity line (STM32F105/107)

72 MHz Cortex-M3 CPU	Up to 64-Kbyte SRAM	Up to 256-Kbyte Flash	USB 2.0 OTG FS	3-phase MC timer	2x CAN 2.0B	2x I <sup>2</sup> S audio	Ethernet IEEE 1588
----------------------	---------------------	-----------------------	----------------	------------------	-------------	---------------------------	--------------------

STM32 F1 series—Performance line (STM32F103)

72 MHz Cortex-M3 CPU	Up to 96-Kbyte SRAM	Up to 1-Mbyte Flash	USB FS device	3-phase MC timer	CAN 2.0B	SDIO 2x I <sup>2</sup> S
----------------------	---------------------	---------------------	---------------	------------------	----------	--------------------------

STM32 F1 series—USB Access line (STM32F102)

48 MHz Cortex-M3 CPU	Up to 16-Kbyte SRAM	Up to 128-Kbyte Flash	USB FS device
----------------------	---------------------	-----------------------	---------------

STM32 F1 series—Access line (STM32F101)

36 MHz Cortex-M3 CPU	Up to 80-Kbyte SRAM	Up to 1-Mbyte Flash
----------------------	---------------------	---------------------

STM32 F1 series—Value line (STM32F100)

24 MHz Cortex-M3 CPU	Up to 32-Kbyte SRAM	Up to 512-Kbyte Flash	3-phase MC timer	CEC
----------------------	---------------------	-----------------------	------------------	-----

STM32 L1 series—Ultra-low-power (STM32F151/152)

32 MHz Cortex-M3 CPU	Up to 48-Kbyte SRAM	Up to 384-Kbyte Flash	USB FS device	Data EEPROM up to 12 Kbytes	LCD 8x40 4x44	Comparator	BOR MSI VScal
----------------------	---------------------	-----------------------	---------------	-----------------------------	---------------	------------	---------------------

**Figure 3** The STM32 family is comprised of four separate series of MCUs focusing on different applications with core speeds from 24 to 168 MHz and Flash memory options ranging from 16 KB to 1 MB to provide the ideal mix of performance and peripherals at the lowest cost.



to provide the ideal mix of performance and peripherals at the lowest cost (see Figure 3):

**High Performance:** The entire architecture of the STM32 F4 and STM32 F2 MCU series has been tuned to provide high-performance without compromising flexibility (see sidebar, Introducing the STM32 F4). Built upon a 90 nm process, ST's innovative Advanced Real-Time (ART) memory accelerator technology, and a zero-wait execution path, STM32 F4 and STM32 F2 series MCUs provide outstanding performance (up to 168 MHz/210 DMIPs for the STM32 F4) and power efficiency (only 22.5 mA at 120 MHz for the STM32 F2). Its multilayer bus matrix enables the highest

levels of multiprocessing by supporting simultaneous transfers to multiple peripherals and memory while the CPU continues to execute code. In addition, its Flash memory has been designed to remove access bottlenecks typical of other MCUs. This enables the CPU to operate at its full speed when executing from Flash. *(For more information on developing real-time, high-performance applications, see page 12.)*

**General-Purpose Applications:** ST offers three general-purpose MCU series based on the STM32 F-1 architecture with varying performance, memory, and peripheral capabilities. For simple applications, Access Line MCUs provide

36 MHz performance with the STM32 architecture's extensive selection of advanced peripherals. For applications needing to support USB, USB Access Line MCUs operate at 48 MHz and have an integrated USB port. For higher end devices, Performance Line MCUs offer 72 MHz performance, USB, and application-specific peripherals like ST's unique 3-phase Motor Control Timer.

**Superior Connectivity:** The STM32 Connectivity Line makes networking economical by integrating an embedded Ethernet MAC with its own dedicated DMA and IEEE 1588 precision time protocol hardware support. Turnkey consumer

device connectivity with full USB functionality is enabled through the USB 2.0 OTG (On-The-Go) peripheral, and dual CAN interfaces make this the MCU of choice for CAN gateways. The STM32 Connectivity Line also offers two audio-class I2S interfaces to meet the needs of most consumer audio applications. *(To see how to design systems with more efficient connectivity, see page 35.)*

**Cost-Sensitive Design:** STM32 Value Line microcontrollers provide the least expensive path to market. Ideal for cost-sensitive consumer, appliance, and industrial applications, STM32 Value Line MCUs offer the performance of a 32-bit

## Introducing the STM32 F4

As the world's highest performance Cortex-M microcontroller, the STM32 F4 is ST's flagship MCU operating at 168 MHz and providing 210 DMIPs. Based on the ARM Cortex-M4 core, it provides advanced floating point and digital signal processing capabilities to enable compute-intensive processing across a broad range of applications ranging from point of sale, industrial automation, solar, transportation, medical, security, consumer, communications, and test and measurement. The ultra-low power STM32 F4 is built on ST's 90 nm process which allows

the CPU core to run at only 1.2 V. In addition, with dynamic voltage scaling capabilities and ST's innovative Adaptive Real-Time (ART) memory accelerator which maximizes processing performance equivalent to 0-wait state execution, the STM32 F4 offers outstanding power efficiency of just 230  $\mu$ A/MHz or 38.6mA at 168 MHz executing Coremark benchmark from Flash memory. The STM32 F4 also integrates a wide range of application-specific peripherals and interfaces, providing advanced processing capabilities and turnkey communications with a single chip.

4 starter kits, numerous boards



STM32 promotion kits



13 different RTOS and stack solution providers



More than 15 different development IDE solutions



Figure 4 Development tools for the STM32 come from the world's largest vendor ecosystem, providing the evaluation boards, compilers, and development software designers need to accelerate their development.

core at 16-bit pricing, giving developers the headroom to implement enhanced features and provide superior products compared to the competition. These MCUs also have a 3-Phase Motor Control Timer to improve performance for motor control applications, flexible static memory controller, LCD parallel interface, and a CEC (Consumer Electronics Control) interface for communicating

with other devices over HDMI. Running at up to 24 MHz, STM32 Value Line MCUs offer an excellent balance of cost, performance, and peripherals, making it the ideal choice for developing cost-effective applications traditionally addressed by 16-bit MCUs.

**Ultra-Low Power and Portable Devices:** The feature-rich STM32 L1, based on ST's industry-leading EnergyLite™

technology implemented in ST's 130 nm ultra-low leakage process technology, is the industry's only MCU offering ultra-low power (down to 185  $\mu$ A/DMIPS) with high performance (up to 33 DMIPS) for maximum energy efficiency. Multiple innovative low power operating modes enable developers to further minimize power consumption depending upon the current

operating requirements. For example, a typical low-power MCU can conserve power by dropping the clock frequency; however, the core still runs at its full supply voltage. The STM32 L1's integrated regulator gives developers the option to also reduce the core operating voltage as the operating frequency drops, enabling even higher power efficiency. Stop Mode with

Everything you need to discover STM32 F4 32-bit ARM Cortex™-M4 based MCUs featuring:

- » Evaluation board
- » Embedded ST-LINK/V2
- » USB interface for debugging and programming
- » Numerous examples available on [www.st.com](http://www.st.com)



**Figure 5** ST's STM32 Discovery kits provide everything developers need for a quick start to a production design with minimal tools investment.

RTC and full RAM retention requires only 1.3  $\mu\text{A}$ , and low-power run mode clocked at 32 kHz provides access to the full capabilities of the CPU while bringing power consumption down to 10.4  $\mu\text{A}$ . Supporting a wide supply voltage range from 1.65 to 3.6 V, flexible 1.8 to 3.6 V operation for all digital and analog functions, optional LCD controller, integrated AES, and a variety of enhanced security and safety features, the STM32 L1 is ideal for a wide range of applications, including portable medical, alarm systems, factory automation, mobile devices, metering, and sensors.

*(To learn more about achieving ultra-low power consumption, see page 20.)*

### Unparalleled Flexibility

Software- and pin-compatibility between the various STM32 series makes for a broad range of devices that allow developers to easily step up or step down performance across a product line. Developers have the ability to develop first prototypes using a high-performance STM32 with large memory to speed development. Once the application code is in place, the MCU can be scaled down to a slower device with less memory that also eliminates peripherals which aren't needed to cost reduce the system.

*(To learn more about extending a product line by scaling the MCU and reusing IP, see page 42.)*

ST also offers **MicroXplorer**, a graphical tool which simplifies pinout configuration of STM32 MCUs. Since the same pin can be used for different peripherals and functions (GPIO, USART Tx, ADC input channel, etc.), MicroXplorer assists developers by defining a pinout that maps the pins needed for a peripheral based on the current operating mode. This not only speeds initial MCU configuration but supports reconfiguration of a system based on new application requirements.

Development tools for the STM32 come from the world's largest vendor ecosystem, offering the evaluation boards, compilers, and development software designers need to accelerate their development (see Figure 4). In addition, everything needed for a quick start to a production design with minimal tools investment is available in ST's Discovery Kits for the STM32 F2/F4, STM32 Value Line, and STM32 L1. Providing the ideal development environment for rapidly evaluating, learning, and prototyping, each kit comes with an in-circuit ST-LINK debugger/programmer for non-intrusive debugging, full development

tool chain, ready-to-run example applications, and schematics for reference designs (see Figure 5). Kits also have an extension connector providing access to all of the STM32 pins and can be used to debug prototype boards as well.

To keep up with changing application requirements, developers need a flexible MCU architecture that offers flexible performance. The breadth of the STM32 family allows developers to select devices with the highest performance, the lowest power, the best price, and the right mix of peripherals. Because STM32 MCUs are software- and pin-compatible, developers can use the same tool chain and libraries to accelerate and simplify design across an entire product line. And with the broadest ARM Cortex-M family on the market, the STM32 architecture offers an ever-expanding line of MCUs that meet the evolving needs of developers both today and tomorrow. 🦋

# Maximizing Performance for Real-Time Embedded Systems

By Reinhard Keil, Director of MCU Tools, ARM Germany GmbH  
 Ian Johnson, Product and Third Party Relations Manager, ARM Ltd  
 Shawn Prestridge, Sr. Field Applications Engineer, IAR Systems  
 Alec Bath, Applications Engineer, STMicroelectronics

Many embedded applications—including control systems, digital audio/video devices, industrial automation, portable medical instruments, and diagnostic test equipment—require deterministic system behavior to meet real-time deadlines and process data within a meaningful time frame. With increased interest in greater operating efficiency, network capabilities, and higher performance, these systems also require more advanced signal processing capabilities.

In the past, embedded applications that require real-time control and signal processing capabilities have had to compromise by either using an MCU or DSP but not being able to use both. Using a separate MCU and DSP will introduce complex multi-processor issues into the design flow as well as the need to

handwrite assembly code for critical-loop algorithms.

Rather than complicate or compromise design, ST's STM32 F4 MCU architecture intelligently blends the capabilities of an MCU with a DSP to provide a powerful yet easy to program platform. Based on the ARM Cortex-M4 core, the STM32 F4 is supported by an extensive ecosystem that enables systems to deliver high performance and advanced signal processing using the CMSIS DSP Library.

## The STM32 F4: The Industry's Highest Performance Cortex-M-based MCU

Maximizing performance is an art. And given that the majority of an embedded application's functionality is implemented in software, the efficiency of a system will depend primarily

upon how well it is coded. The STM32 architecture offers a powerful platform with hardware-based accelerators and application-specific peripherals that simplify application design while offloading processing and memory management from the CPU.

The STM32 architecture is built upon the ARM Cortex-M3 and Cortex-M4 cores using a Harvard architecture with separate instruction and data buses for

parallel instruction fetching and data loads and stores. All STM32 MCUs are programmed using the rich, unified Thumb-2 instruction set that provides 32-bit performance while supporting 16-bit Thumb instructions for the smallest code density and memory requirements; up to ten lines of code can be replaced with a single 32-bit instruction (see Table 1). In addition, 16- and 32-bit instructions can be used without having to switch modes.

Feature	Cortex-M3 cycles	Cortex-M4 cycles
High-precision MAC	3–7	1
DSP instructions	1–2	1
Saturated arithmetic	1	1
Bitwise operations	1	1
Mixed bit-width capabilities	Not available	1
Packed data processing	Not available	1
SIMD capabilities	Not available	1

**Table 1** Cycle execution times for advanced instructions

For deterministic, low-latency responsiveness, a Nested Vectored Interrupt Controller (NVIC) provides a worst-case interrupt response time of 12 cycles. This includes automatic saving of corruptible registers as well as handling of exception prioritization and nesting. In addition, if another interrupt occurs before the current one has completed, the MCU recognizes this and can respond to the next interrupt within just 6 cycles. Called tail-chaining, this approach to interrupts increases performance and responsiveness for interrupt-driven applications. This enables the system to be maintained in a low-power state, to rapidly wake to service an interrupt, and then re-enter sleep state on exit from the interrupt handler.

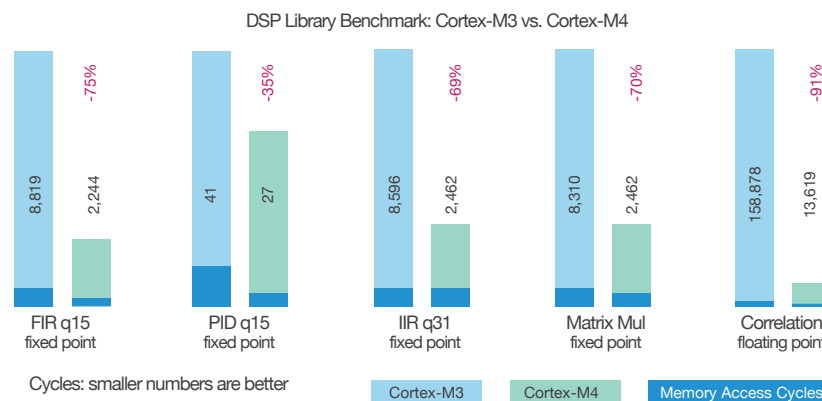
With ST’s flagship MCU—the STM32 F4—developers can give designs based on the STM32 F2 a substantial increase in performance without having to rewrite their application. At a raw performance level, fixed-point DSP functions execute on the Cortex-M4 core-based STM32 F4 up to twice as fast as they do on a Cortex-M3 core-based STM32 F2 while floating-point functions execute

up to 10X faster (see Figure 1). Compared to the leading 16- and 32-bit MCUs with DSP extensions, the Cortex-M4 core is twice as efficient when performing the fundamental operations—FIR, IIR, and FFT—that make up a significant portion of communications, audio, and motor control processing (see Figure 2). This means that low-cost consumer devices can perform real-time digital content processing like MP3 while leaving sufficient bandwidth for other application tasks (see Figure 3).

Even though the STM32 F4 is based on the Cortex-M4 core, STM32 F4 MCUs support all of the features of STM32 F2 MCUs, making it straightforward to migrate designs which need more performance. The tremendous performance gains of the STM32 F4 come from its higher clock rate and several powerful new capabilities:

**Single-cycle MAC:** The ability to perform a multiply-and-accumulate (MAC) in a single cycle provides substantial performance improvements for critical-loop algorithms.

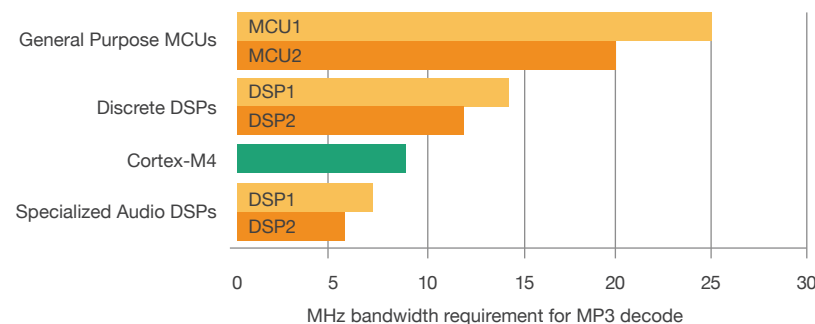
**DSP extensions:** With SIMD instructions (single instruction,



**Figure 1** Fixed-point DSP functions execute on the STM32 F4’s Cortex-M4 core up to twice as fast as they do on an STM32 F2’s Cortex-M3 core while floating-point functions execute up to 10X faster.



**Figure 2** Compared to the leading 16- and 32-bit MCUs with DSP extensions, the STM32 F4’s Cortex-M4 core is twice as efficient when performing the fundamental operations—FIR, IIR, and FFT—that make up a significant portion of communications, audio, and motor control processing.



**Figure 3** The STM32 F4 is able to perform real-time digital content processing like MP3 decoding while leaving sufficient bandwidth for other application tasks.

multiple data operations per cycle), saturating arithmetic capabilities, and a packed data format to increase processing efficiency, the STM32 F4 is capable of accelerating performance in a wide range of applications.

**FPU:** The Floating Point Unit is a single-precision engine available on all STM32 F4 MCUs to provide hardware-assisted addition, subtraction, multiplication, division, fused MAC, and square root. The FPU can be individually powered down when it is not in use.

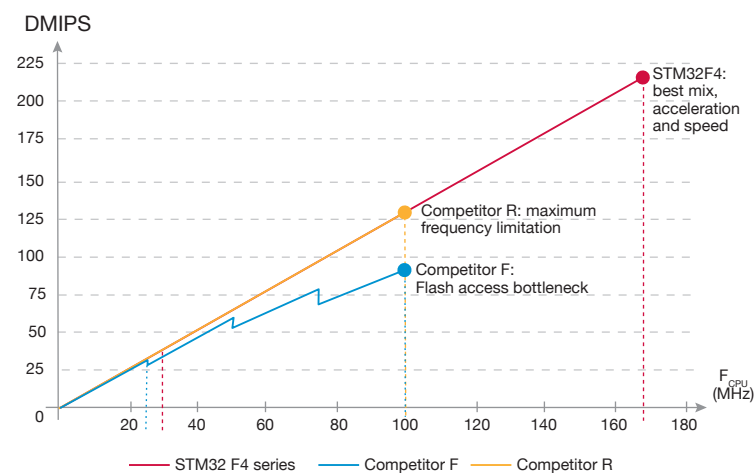
**Larger SRAM:** With a large 192 KBytes of SRAM, developers have more memory available for optimizing application performance.

### The STM32 Advantage

The STM32 architecture builds upon the foundation of the ARM Cortex-M3 and Cortex-M4 cores. However, ST has integrated numerous innovative technologies to enable the STM32 architecture to achieve the highest performance of any Cortex-M processor-based MCU in the industry with ultra-low power consumption:

**ART Accelerator:** ST's Adaptive Real-Time (ART) memory accelerator technology is a key performance differentiator for STM32 F2 and STM32 F4 MCUs when executing from Flash. At high CPU speeds, Flash access can become a bottleneck and significantly reduce performance by introducing undesirable wait states. The ART Accelerator uses a prefetch queue and branch queue to store first instructions and constants associated with branches in code. With its deep caches, 128-bit wide memory interface, and background operation, wait states can be effectively eliminated when executing from Flash. Rather than have performance degrade as clock speed increases, STM32 F2 and STM32 F4 MCUs are able to consistently provide their full performance across all clock speeds (see Figure 4).

**Multi-layer Bus Interconnect:** With the great number of peripherals in the STM32 architecture, performance is highly dependent upon the efficiency with which the MCU can move data internally between the CPU, peripherals, and memory. The 7-layer bus



**Figure 4** ST's Adaptive Real-Time (ART) accelerator technology utilize a deep branch cache, 128-bit wide memory interface, and background operation to effectively eliminate wait states when executing from Flash, enabling STM32 F2 and STM32 F4 MCUs to consistently provide their full performance across all clock speeds.

matrix that interconnects STM32 MCUs (see Figure 5) enables simultaneous transfers between multiple masters and slaves without requiring CPU involvement. This provides STM32 MCUs with a tremendous interconnect capacity that eliminates peripheral and memory access bottlenecks for the highest operating performance.

### Dedicated DMA engines:

Embedded systems must be able to support multiple real-time data streams, including one or more communications links, high-frequency data from multiple ADC channels,

and accesses to different memory blocks. To facilitate the greatest throughput, the STM32 F4 integrates multiple DMA engines, including dedicated DMA engines for its Ethernet and USB interfaces to support true zero-copy functionality. This enables the system to support various data streams with no contention between high-speed interfaces and without loading the CPU.

### Application-Specific Accelerators:

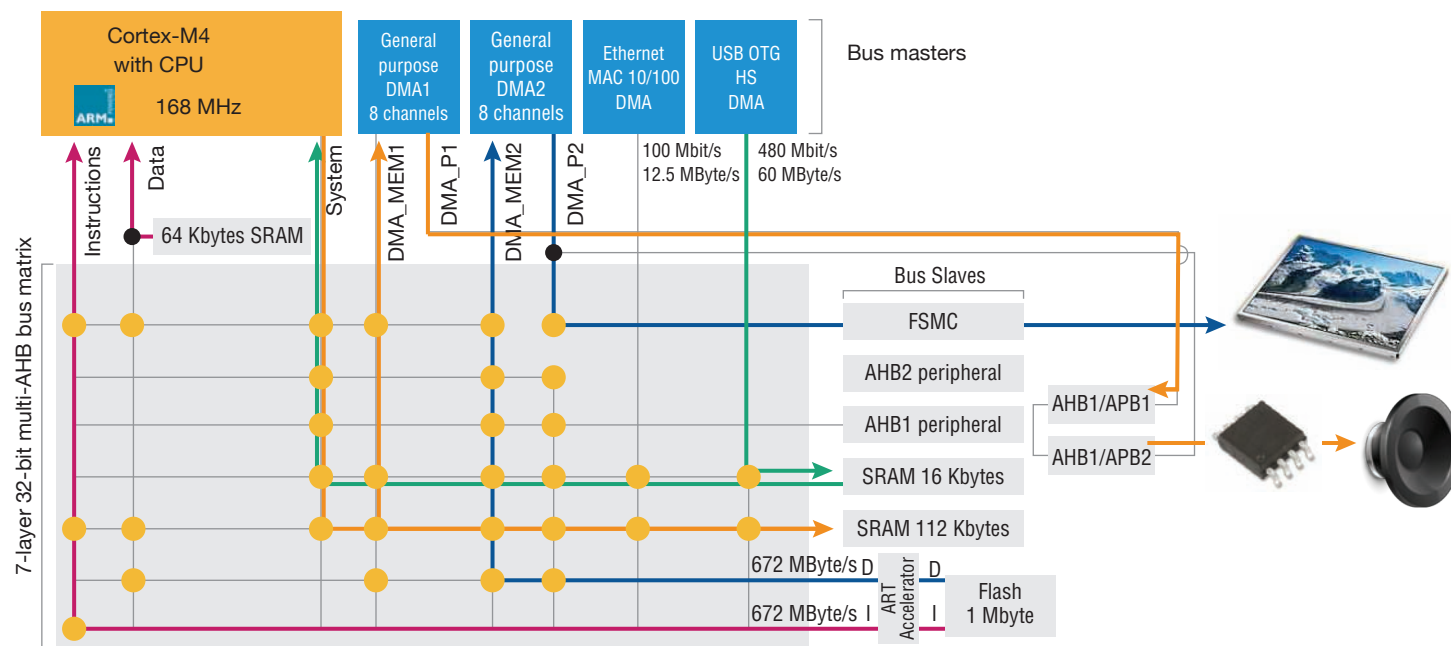
The STM32 architecture offers a variety of hardware-based engines to accelerate processing for various applications. For example,

efficient security can be easily added to communications using the integrated cryptographic/hash engine available on many STM32 MCUs. Packets are sent to the engine for processing and a flag is set or interrupt triggered when the result is available. Likewise, the hardware-based CRC engine offloads communications overhead from the CPU.

Together, these enhancements—ART Accelerator, multi-layer bus interconnect, dedicated DMAs, and application-specific accelerators—make for an architecture that is highly optimized for real-time embedded processing. In addition, the ability to support simultaneous memory and peripheral transactions enables a single STM32 F4 MCU to maintain several high-speed interfaces, perform compute-intensive signal processing, and manage a GUI-based display with advanced HMI functions.

### Accelerated Development and Optimized Performance

To make the complexity of the STM32 architecture transparent during development, ST and its partners, including IAR and Keil,



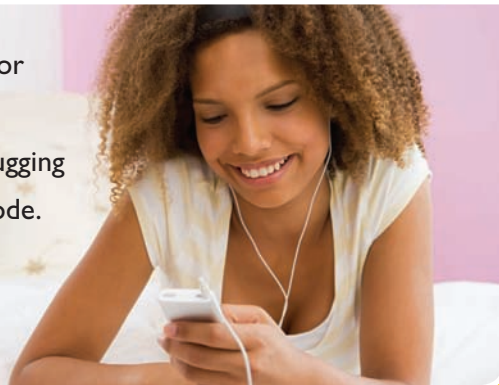
**Figure 5** The 7-layer matrix that interconnects STM32 MCUs with peripherals and memory enables simultaneous transfer between multiple masters and slaves without requiring involvement from the CPU. This provides STM32 MCUs with a tremendous interconnect capacity that eliminates peripheral and memory access bottlenecks for the highest operating performance.

The ability to support simultaneous memory and peripheral transactions enables a single STM32 F4 MCU to maintain several high-speed interfaces, perform compute intensive signal processing, and manage a GUI-based display with advanced HMI functions.

Software tools  
for your next  
embedded design  
with STM32 MCUs.



IAR Embedded Workbench® for  
ARM® with Power Debugging  
technology simplifies the debugging  
and testing process of your code.



 **IAR**  
SYSTEMS  
the code to success™

offer a great variety of innovative tools that go beyond the standard compiler and debugger so that developers can maximize performance without having to create low-level code which cannot easily be ported when the

describing how to use them most efficiently under different operating conditions. In addition, if major specification changes are subsequently made to the system, so long as developers work within the framework, it

The multitasking capabilities of an RTOS also eliminate the need for polling interfaces or holding up the CPU while waiting for a critical task to complete.

design needs to move to a lower cost or higher performance MCU. These tools not only accelerate time-to-market by simplifying design, they make it easier to focus on optimized performance.

To accelerate development, ST provides a configuration wizard for creating a base framework upon which developers can quickly build applications. Configuration of peripherals is non-trivial and failing to take full advantage of each of the MCU's application-specific peripherals can negatively impact performance. The framework provides optimized drivers for all of the STM32 peripherals with documentation

will be flexible enough to allow developers to quickly adapt to the new requirements.

Real-time operating systems (RTOS) like RTX from Keil™ provide a reliable framework that efficiently manages real-time processes and allows applications to fully utilize the STM32 architecture. Task management can quickly become extremely complex, and choosing an embedded RTOS that has been specifically designed to exploit the integrated capabilities of the STM32 simplifies the process of managing multiple real-time tasks while optimizing allocation of processor resources to



ensure real-time deadlines are met. For example, interrupts never need to be turned off for the RTOS. The multitasking capabilities of an RTOS also eliminate the need for polling interfaces or holding up the CPU while waiting for a critical task to complete. Instead, available CPU cycles can be allocated to other tasks until an interrupt signals the completion of the task.

Developers can also rely upon the extensive ARM development ecosystem to speed design with off-the-shelf code which has been specifically adapted for the STM32 architecture and optimized for performance. Networking applications, for example, can make use of middleware supplied by ARM and other companies for quickly creating efficient Flash file systems, TCP/IP stacks, and CAN controllers, among other capabilities. Application-specific software is available as well: ST and its partners, for example, offer a variety of solutions for consumer audio. ST's audio library is designed to ensure that audio processing is completed in time so that users will not hear any pops or clicks.

Middleware and libraries are easy to integrate into STM32-based systems because of the Cortex Microcontroller Software Interface Standard (CMSIS) which provides a rich collection of building blocks for accelerating embedded system design. The CMSIS CORE library offers a standardized interface for all Cortex-M-based MCUs while the SVD library provides a System View Description for peripherals to speed design and facilitate code compatibility among processors. The RTOS library comprises a standard API for RTOSes to enable interoperability with an extensive variety of software templates, middleware, and libraries.

For applications requiring efficient signal processing capabilities, the CMSIS DSP library offers more than 80 algorithms including vector operations, matrix computing, complex arithmetic, filtering, and PID and Fourier transforms. Designed to make DSP programming easy for developers used to working with MCUs, the CMSIS DSP library enables engineers to quickly develop a wide range of complex and reconfigurable systems across industrial,

## Working With Your Compiler

*By Shawn Prestridge, Senior Field Applications Engineer, IAR Systems*

Today's compilers can generate application code from C source code that provides nearly the same or better efficiency than hand-coded assembly. However, since code can be optimized in terms of performance, size, and/or power, compilers need to be guided to achieve the optimal balance for a given application. Compilers do their best to optimize code, but ultimately programmers who assist the compiler by writing "compiler-friendly" code will achieve higher efficiency and better performance.

- » Be careful not to start the optimization process before the functionality of code has been verified. Optimized code can look bizarre, making it difficult to debug. In addition, it will be much easier to determine whether a problem is the result of a bug or an unintended consequence from some element being optimized out of the system.
- » Only call a function once. Compilers cannot predict the side effects of functions at compile time. Therefore, if an identical call is made twice, the compiler cannot assume they will have the same result and will have to make the call twice as well. Calling the function once and putting the result in a variable allows the compiler to store the result in an easily accessed register.
- » Pass by reference rather than copy. Passing by reference saves significant code space, memory, and execution cycles by avoiding copying data during a function call, especially if large arrays are involved. However, passing by copying may be necessary if the function must be prevented from having direct access to data.
- » Inline functions. Inlining generates function code rather than a function call. This eliminates the overhead of a function call to improve performance but can increase code size. IAR's Embedded Workbench, for example, makes intelligent decisions for when to inline code to balance performance and code size to ease this optimization technique. In addition, developers always the option to override these decisions.
- » Use appropriate data sizes. The STM32 is a 32-bit architecture. Using a different data size can force the compiler to shift, mask, and sign-extend operations, leading to lower performance and issues that arise from signedness and casting.
- » Use signedness with care. Using signedness when it is not required can increase code size (i.e., certain operations may require an extra

automotive, medical, and military applications. Available free of charge, the CMSIS DSP library is provided as C source code so the compiler can optimize the code to the application (i.e., performance versus code and data size). Code is portable across all Cortex-M-based MCUs, enabling simple migration of DSP functionality across the STM32 families. In addition, the CMSIS DSP library has been optimized to take advantage of the optional floating point unit (FPU) integrated into the Cortex-M4 architecture.

## Debugging to Optimize Performance

To optimize a system, developers need run-time visibility into operations to verify how various subsystems interact with each other and to identify, locate, and resolve errors quickly. However, many embedded systems, including systems maintaining a communications link or providing control data, cannot be stopped and then restarted. Developers, however, need more than just a “snapshot” of the moment when code is halted so they can analyze system

behavior and identify real-time performance bottlenecks.

STM32 MCUs have superior debugging capabilities integrated into the MCU, including hardware breakpoints, on-the-fly read/write access to variables and memory contents, and instruction stream tracing for advanced code execution analysis, all without having to go through the processor or halt its execution. Rather than requiring code to be instrumented (and thereby impacting code execution), developers can use the integrated Embedded Trace Macrocell (ETM) or serial wire tracing to non-intrusively access the system, including capabilities such as monitoring the RTOS or tracking the different threshold levels of an ADC.

Companies like IAR Systems and Keil offer an array of tools which utilize the trace technology implemented within the STM32 MCUs to assist developers in maximizing application performance and reliability by giving them complete visibility into program execution and MCU operations. For example, embedded trace data can be streamed to a PC hard drive to collect long-term run-time

test-and-jump condition to handle negative numbers) or have intended consequences (i.e., shifting or masking data that has been sign-extended).

- » Avoid explicit casting. Casting is not a free operation. Casting to a larger type can introduce unnecessary sign-extended overhead, invoke the floating point library, or corrupt a pointer (i.e., if developers use ints and pointers interchangeably).
- » Invoke libraries intentionally. Using `print()` where you can use `printf()` may cause a larger library to be included in your application, needlessly consuming code space for functions that are never called.
- » Baseline your code. A dramatic increase in code size from simple changes may mean a previously-unused library has been invoked through an unintentional cast.
- » Avoid using global variables. A function that accesses a global variable multiple times will have to repeatedly read that value from memory. Rather, read the global variable into a temporary variable that is local to the function and the compiler may be able to hold it in a register for better performance.
- » Group function calls. When function calls are separated by other operations, each call forces the compiler to store any register-allocated variable to memory. When calls are consecutive, these values only need to be saved once.
- » Avoid inlining assembly. Since the compiler knows nothing about inlined assembly, it cannot optimize the code around it. Putting assembly code in a separate file frees the compiler to optimize the rest of the code.
- » Don't write “clever” code. Writing an expression in the fewest lines of C code with conditional values often results in difficult-to-read code that can actually require more instructions and take longer to execute because of the need to store temporary values or perform a function multiple times.
- » Access structures in order. Rather than bouncing through a structure and requiring complex pointer manipulation, accessing elements in order enables the compiler to use a fast increment instead.

Powerful search and filtering capabilities enable developers to focus their optimization efforts where they will yield the most gains.

data for offline analysis. Having the entire execution history of an application synchronized to C source code complete with timing information is especially useful for analyzing a wide range of issues, including sporadic problems that arise from data

that impacts throughput and increases the load on the CPU. Another benefit of using performance analysis is that it shows developers the actual impact certain coding techniques have on performance. Many times code can be written

including how many times an exception has been entered and the min/max time spent in each exception. Display of event counters can be used to reveal system behavior such as when extra cycles are taken to execute instructions due to

intrusive instruction stream so that testing is done on final, optimized code running at full speed. Data is color-coded and summarized by function or module and can be saved for documentation. To view a video showing how to perform code coverage, click [here](#).

## The STM32's ETM provides a complete, non-intrusive instruction stream so that testing is done on final, optimized code running at full speed.

corruption or incorrect timing. Powerful search and filtering capabilities combined with different reporting and graphing options enable developers first to quickly sort trace data to determine which areas of code the application spends most of its time executing and then focus their optimization efforts where they will yield the most gains.

Performance can be analyzed from a timing perspective for an entire module or as narrow as a single line of code to expose spikes in performance that might otherwise pass unnoticed. For example, a change in task priority may create a conflict with a communications link, resulting in lost data and the need to request a retransmission

different ways, such as a function outputting a single character or a whole line. Often, one of these approaches will yield better performance. Over time, developers will learn which approaches are more efficient so they can change how they program and write optimized code from the very start.

A system can also be analyzed based on changes to variables. Signals/variables can be monitored graphically with accurate timing information showing change across time as well as any instructions that have modified a variable. The STM32 debugging capabilities can also be extended to capture detailed statistical information about exceptions and interrupts,

memory contention, overhead from handling exceptions, cycles spent in sleep mode, cycles spent accessing memory, and number of folded branch instructions.

For mission-critical applications, software validation requires code coverage. Code coverage tracks which lines of code in a program are executed so OEMs can verify that all parts of a system are operational and reliable. Traditionally, this process has been tedious, requiring developers to set a trigger and capture trace data until the internal buffer is full, then reconfigure and run the system again until coverage has been completed. The STM32's ETM provides a complete, non-

With devices ranging from the ultra-low power EnergyLite™ STM32 L1 to the industry's highest performance Cortex-M processor-based MCU, the STM32 F4, developers can find the ideal MCU to meet the processing, power, and cost requirements of nearly every embedded application. The combination of the powerful Cortex-M3 and Cortex-M4 cores with ST's innovative ART Accelerator, multi-layer bus interconnect, dedicated DMAs, and application-specific peripherals provides developers with an unbeatable platform for high-performance applications. In addition, STM32 MCUs are supported by a wide range of advanced tools that speed design, development, and debugging of even the most complex applications. 🦋

# Designing for Low Power Applications

By Wolfgang Schmitt, Director Sales North America, Hitex  
 Shawn Prestridge, Senior Field Applications Engineer, IAR Systems®  
 John Knab, Applications Engineer, STMicroelectronics

The increasing complexity of portable and mobile electronics has made power efficiency a primary design constraint which developers need to consider from the very beginning of the design process. In the past, power efficiency was a factor only hardware developers were able to influence using tools like a multimeter and oscilloscope. So much of a system's functionality is now implemented in software, however, that how an application is architected will have a substantial impact on its power efficiency.

ST's STM32 L1 EnergyLite™ architecture provides many features that allow developers to optimize applications for power consumption. In addition to a wide array of advanced peripherals which accelerate performance and offload the main CPU so that devices can spend more time in sleep mode, the STM32 L1 platform offers low-power operation, dynamic

voltage scaling, intelligent peripheral management, and partial sleep modes. STM32 L1 MCUs are built upon ST's 130 nm ultra-low-leakage process technology and provide outstanding power efficiency for a wide range of applications.

With these integrated technologies, developers can maximize utilization of MCU resources while minimizing power consumption so that energy efficiency for all applications can be significantly improved. In addition, by taking advantage of power debugging techniques that correlate power consumption to application code, developers can gain visibility into how their design decisions impact overall power efficiency. Armed with this information, developers can make more informed decisions about how to configure MCU resources and structure application code.

## Power Modes: More Than Just Sleep

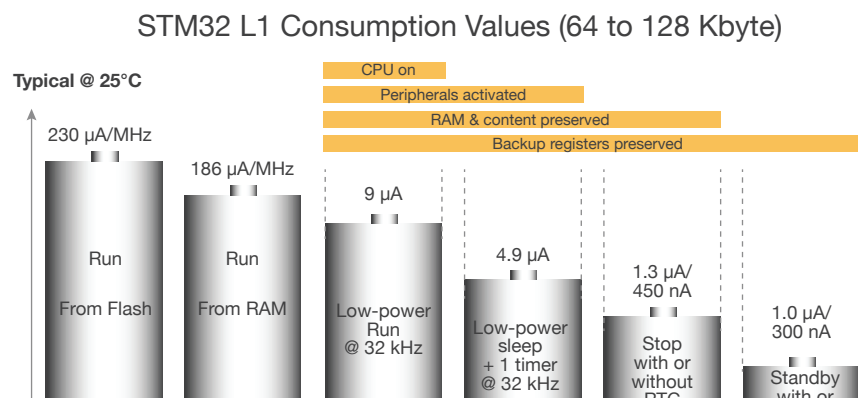
Estimating power used to be a relatively straightforward calculation. Devices were either on or off, and developers could instrument code to measure active and sleep times for an application (i.e. Total Power = Active Current \* Active Time +

By taking advantage of power debugging techniques that correlate power consumption to application code, developers can gain visibility into how their design decisions impact overall power efficiency.

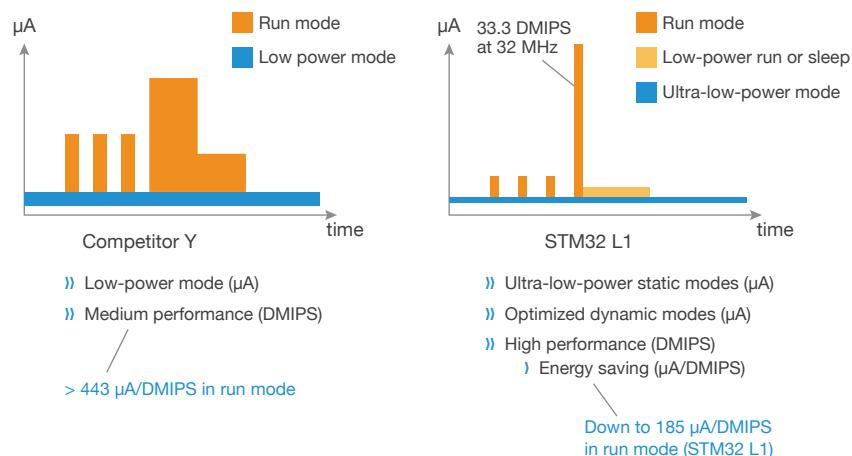
Sleep Current \* Sleep Time). With the availability of multiple power modes, however, estimating actual power consumption has become more difficult. Depending upon what

background tasks are active and whether a user is currently interacting with the device, the system will switch among several different operating modes potentially thousands of times per second. There are multiple factors that affect dynamic power which can be modified to improve efficiency, including supply voltage,

operating frequency, peripheral clock gating, and running from RAM. How fast the system can wake up also affects dynamic power since no instructions are executed while the CPU wakes.



**Figure 1** The STM32 architecture enables efficient power management by providing developers with several operating modes. The figures listed here are typical power consumption at 25° C for an STM32 L1 MCU.



**Figure 2** The STM32 L1 offers superior power consumption of down to 185 µA/DMIPS for those applications which need ultra-low power consumption.

The optimal power strategy for an application depends a great deal upon how the system is going to be used. A system that stays on continuously, for example, needs to be architected quite differently compared to a system that spends most of its time asleep. Similarly, a system that must frequently wake the CPU to perform background tasks, such as servicing a communications interface, will need to offer fast responsiveness as well as power efficiency. Power can also be managed by turning off peripherals that are not in use. Each of these considerations results in a distinctly different power profile, and balancing performance, system responsiveness, and power requires that developers select the most efficient operating modes.

The STM32 architecture offers many options for optimizing ultra-low power consumption, including several operating modes (see Figure 1):

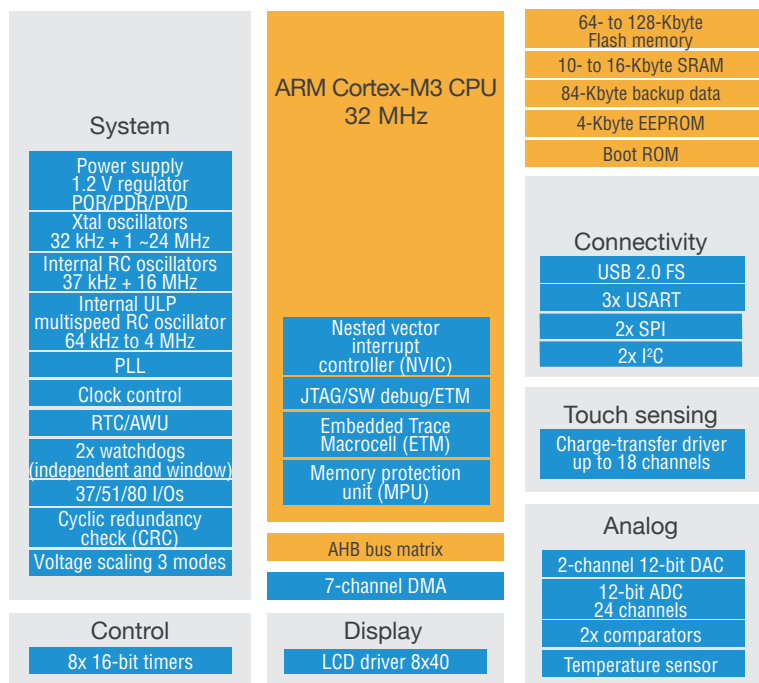
**Run from Flash:** When executing code from Flash, the STM32 L1 offers an outstanding combination of performance and low power with power consumption down to 230 µA/MHz.

**Run from RAM:** Powering down the Flash and executing code from RAM further lowers power consumption down to 186 µA/MHz. An additional benefit is that code can be executed at full speed from the internal RAM.

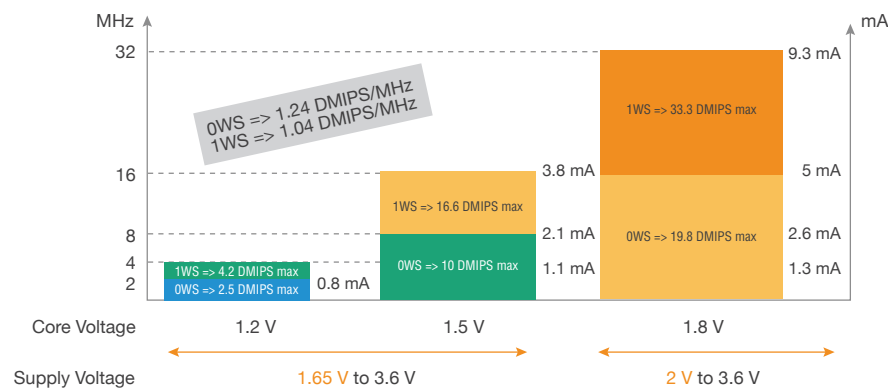
**Low-Power Run:** In this mode, the CPU is active but operating at 32 kHz. Power consumption drops to 9 µA while still enabling the system to be completely available, including all MCU peripherals. This mode is useful for systems that must always be on to monitor system operations but do not always require high performance processing capabilities (see Figure 2). In addition, this mode provides excellent time-to-wake-up.

**Low-Power Sleep:** This mode puts the CPU to sleep but still keeps some peripherals active for both receiving incoming data and waking the system quickly. Operating at 32 kHz with one timer available, this mode consumes only 4.9 µA and provides a power-effective way to intelligently monitor and wake the system without having to be continuously active.

**Stop Mode:** The CPU and peripherals are shut down in Stop mode. The system can run



**Figure 3** The STM32 L1 (and the STM32 architecture in general) offers a variety of features specifically designed to further decrease power consumption.



**Figure 4** The STM32 architecture supports three core voltage levels—1.8, 1.5, and 1.2 V—to dynamically scale down performance and power consumption when the full processing capabilities of the CPU are not needed. The dynamic voltage scaling figures shown are based on an STM32 L1 in run mode.

at 450 nA without the real-time clock or 1.3  $\mu$ A with the real-time clock active. This mode allows the system to wake more quickly than standby mode.

**Standby Mode:** This mode provides the lowest power operating mode: only 1.0  $\mu$ A without the real-time clock or 300 nA with the real-time clock active.

In addition to its numerous operating modes, the STM32 architecture offers several other options for further decreasing power consumption (see Figure 3):

» **Dynamic Voltage Scaling:**

The STM32 L1 supports three core voltage levels – 1.8, 1.5, and 1.2 V – to scale down power consumption when the full processing capabilities of the CPU are not necessary. Selectable using an on-chip programmable LDO voltage regulator, each level gives another incremental reduction in power consumption (see Figure 4). Developers also have the option to scale frequency to balance power and clock rate to match dynamic processing requirements.

» **Flexible Clock Tree:** Three clock sources can be configured as the main system

clock to ensure that the MCU is not running any faster than is needed to conserve power. For example, for a low power mode, a multi-speed internal clock can serve as the system clock. When extra performance is needed, the system can switch to either the High-Speed External clock (HSE) or the High-Speed Internal oscillator (HSI).

» **Multispeed Internal RC Oscillator (MSI):**

The MSI is one of the STM32 L1’s five clock options and is an ultra-low power clock that is able to generate multiple frequencies from 64 kHz to 4 MHz with power consumption proportional to speed. This is the clock the MCU uses when powering up after a reset.

» **Automatic Clock Gating:**

Clock gating turns on and off downstream buses to lower the power consumption of the bus and peripherals. In addition, each peripheral can be disabled when not in use.

» **Power Down Flash:**

With this feature, an application can run code out of RAM and disable the Flash controller, resulting in power savings on the order of 8.5%.

CPU running at	ADC current consumption measure (in $\mu\text{A}$ )	
	ADC is running in Normal Mode**	ADC is On in Power Saving Mode***
16 MHz (from HSI)	1453 $\mu\text{A}$	630 $\mu\text{A}$
4 MHz (from MSI)*	1453 $\mu\text{A}$	445 $\mu\text{A}$
1 MHz (from MSI)*	1000 $\mu\text{A}$	258 $\mu\text{A}$
32 kHz (from MSI)*	900 $\mu\text{A}$	150 $\mu\text{A}$

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

**Figure 5** Rather than consume ~900  $\mu\text{A}$  continuously when active, ADCs can be configured to automatically shut down after conversion for significant power savings. In this example, the ADC is in continuous mode with a delay of 15 cycles between each channel conversion.

#### » ADC Automatic Shutdown:

Rather than consume ~900  $\mu\text{A}$  continuously when active, ADCs can be configured to automatically shut down after conversion. Figure 5 shows the impact of this feature. In fact, with the CPU clocked at 32 kHz, the STM32 L1 can still support a 1 MSPS sampling rate while only drawing an average current of 150  $\mu\text{A}$ .

#### » Integrated RTC with Wake-Up from Low Power Modes:

Essential for applications that need to wake periodically, this RTC runs out of standby circuitry separate from the main core to provide maximum standby power savings.

Which operating modes and capabilities are best to use

depends upon what CPU performance is required, which peripherals need to be active, and how fast the system has to be able to wake up. For example, it may be the case that the CPU operating at a low frequency can complete the required task before the oscillator/PLL is able to lock. For calculations with a long duration, power consumption will be better with the PLL active.

Wake time is important because all of the power consumed while waking is effectively wasted since no work is being done. For systems waking frequently, it may make more power sense to use a low power mode that wakes faster. To minimize overall wake losses, turn on system peripherals starting with those with the lowest current consumption and enable

peripherals only when they need to be used.

### Power Profiling

Ideally, a low power application spends long periods in low power modes and short periods in active modes. With all of the low power features of the STM32 architecture, power consumption can be substantially improved by dynamically switching among the various power modes. However, it can be difficult to accurately determine the actual power consumption for each use case. To get the most out of a system requires a clear understanding of how the different power modes

meets its power budget will require power profiling. Power profiling, or power debugging as it is sometimes called, is not about locating explicit flaws in source code but rather uncovering opportunities to tune how the hardware is utilized. Because each use case imposes different requirements upon the MCU, developers can use profiling tools to determine which low power modes are best for each of the different operating conditions. In addition, developers can try a variety of different test cases using immediate feedback to see which approach or configuration minimizes the power profile,

Power profiling, or power debugging as it is sometimes called, is not about locating explicit flaws in source code but rather uncovering opportunities to tune how the hardware is utilized.

and other power management capabilities of the STM32 can be used. Without knowing how the system is consuming power, optimization will be difficult.

To determine whether a system

as well as to determine the maximum, minimum, and average power consumption of the final application. The power data available will also enable developers to select the optimal



## Go beyond Power Debugging!

- **PowerScale measures the real power consumption – current and voltage at the same time**
- **ACM technology provides a wide dynamic range from 200nA to 500mA**
- **Easy integration to Keil ULINK**

[www.hitex.com/powerscale](http://www.hitex.com/powerscale)

**hitex**    
DEVELOPMENT TOOLS

operating frequency and voltage for each use case.

Depending upon its state, an STM32 MCU can consume currents from mAs down to a few hundred nAs. This is a huge range, and without wide monitoring resolution, power profiling will have limited accuracy and may become unusable. In addition, as soon as a debugger is connected to the system, the overall power analysis is changed due to the additional signals being used for debugging. This is why debugging is always a system intrusion in terms of power measurement.

To address this need, Hitex has implemented the energy profiling tool PowerScale which can be used independently of any debugger/compiler combination and allows for non-intrusive power profiling. PowerScale can also be integrated with 3<sup>rd</sup> party debug solutions to address the need for correlating instruction flow with energy profiling. Such an implementation has been implemented in cooperation with ARM/Keil and their ULINK<sup>®</sup>pro debug adapter. When used with the  $\mu$ Vision Debugger, profiling can be combined with trace

events to record sleep statistics, analyze ISR execution, and perform code coverage with time execution analysis. PowerScale offers non-intrusive monitoring using up to four probes, giving developers the versatility to profile up to four different power domains. The system measures current and voltage simultaneously on all power domains and accurately correlates these measurements. This allows for a whole-system power profile that includes external components such as a Bluetooth radio, LCD, or memory IC. Supporting a large measurement range (Active Current Measurement (ACM) Probe from 200 nA to 500 mA; standard probe from 1 mA to 1 A), a time resolution of up to 100kHz, and easy adaptation, PowerScale analyzes the effects of MCU-specific power features so developers can understand the power impact of every design decision.

Similarly, developers can optimize power consumption using the J-Link debug probe from IAR Systems<sup>®</sup> which can be used to measure both board-level and chip-level power consumption (see Figure 6). For non-intrusive board-level



measurements which include the power usage of all system components, the probe is easily connected to JTAG pin 19. For MCU-level measurements, the probe can be connected to the MCU's  $V_{dd}$  pins with minimal intrusiveness. The J-Link debug probe operates by sampling the program counter at a frequency up to 20 kHz and collects time-stamped event information. The current power is also sampled using an ADC with a resolution of 1 mA.

These tools play an important role in quantifying the differences between different low power modes and allow developers to quantify how each system and program modification affects overall power consumption.

Common tasks to profile include the energy consumption and execution time of an interrupt handler, the actual power consumption of an ADC in each of the different modes, and which voltage scaling option provides the best power efficiency.

For more accurate results, measure power consumption over a long period of time—at least one complete cycle of the device's tasks. Since power is measured statistically, the more measurements taken and the higher the time resolution, the greater the precision. By uploading data streams over USB to a hard drive, long term measurements for statistical analysis can be made without any loss of data granularity.

Effectively unlimited recording of profile data to a hard drive enables long-term analysis of system performance. For example, a system's power profile may change if the system is hot or has been on for a long time, and profiling can expose potential problems before products are already in the field.

Both the PowerScale and J-Link tools also correlate current consumption with application code, and clicking on an event in a power graph will open the corresponding source code. This allows developers to focus their optimization effects on those code segments which consume the most power. These tools also provide valuable comparisons between the different power

saving modes to assist developers in making sure the different elements of the MCU are being used as efficiently as they can. Each tool supports multiple ways of displaying measurement data—from simple current/voltage/power graphs to more complex analysis statistics—and can be integrated with debuggers and other test tools.

This ability to quickly perform root cause power analysis by identifying any line of code responsible for a major change in power consumption enables developers to efficiently optimize power consumption by focusing on system hot spots to maximize their effort. Baseline power consumption also provides a fast means for determining if a



**Figure 6** The J-Link debug probe from IAR Systems® can be used to non-intrusively measure both board-level and chip-level power consumption as well as synchronize power sampling with the MCU program counter.

These tools play an important role in quantifying the differences between different low power modes and allow developers to quantify how each system and program modification affects overall power consumption.

change in code has unintended power consequences. In addition, developers can switch out different hardware components to determine which provides better power performance.

consumption. While this problem can be identified using breakpoints, the process is time-consuming and tedious. With power profiling, developers simply click on a power spike to see the cause.

The STM32 architecture utilizes a flexible, multi-channel DMA engine capable of transferring data between peripherals and/or memory with minimal involvement of the CPU. The CPU can even be put into sleep mode during DMA transfers to further improve power efficiency.

Power profiling can also help identify mismanagement of peripherals. For example, a peripheral may be accidentally left on after being used by a specific function. Such an oversight can be quickly recognized and resolved by noting that the idle power is several mA higher than the expected baseline. Alternatively, an event such as an external signal occurring more frequently than originally specified may wake the system prematurely and adversely increase power

### Power Efficient Design

A key component of power efficiency is processor performance given that the faster the MCU can perform a task, the sooner it can return to sleep. The Cortex-M foundation of the STM32 architecture was specifically designed for the real-time processing needs of embedded systems. Its Thumb2 instruction set brings 32-bit performance with 16-bit code density, and combined with other

enhancements like an integrated interrupt controller, single-cycle multiply functionality, and dedicated instruction and data buses, the Cortex-M architecture provides a 30% performance improvement over the ARM7TDMI architecture, allowing applications to reduce active time and improve power efficiency.

For example, consider an application requiring 20 DMIPS. An ARM7TDMI running at 22.2 MHz will consume 8.7 W while a Cortex-M3 running at 16.7 MHz will consume only 2.5 W. The result is that the Cortex-M3 requires 25% less speed and consumes 70% less power.

For additional CPU offloading, the STM32 architecture utilizes a flexible, multi-channel DMA engine capable of transferring data between peripherals and/or memory with minimal involvement of the CPU. The CPU can even be put into sleep mode during DMA transfers to further improve power efficiency. With its dual APB architecture, peripherals can also be clocked independently. To speed memory transactions, the Cortex-M core supports Atomic Bit Manipulation (ABM). Traditionally, adjusting

a bit in RAM or a peripheral requires a read/modify/write sequence. ABM uses aliases to enable single-instruction reading and writing, resulting in performance and power savings.

With much of a system's functionality implemented in software, application developers can improve power efficiency beyond optimizing how fast processing is completed and the MCU can return to sleep. The STM32 architecture, and EnergyLite STM32 L1 MCUs in particular, provide a variety of low power modes and integrated circuitry such as dynamic voltage scaling, flexible clocking system with automatic clock gating, the ability to turn off Flash, and fast time-to-wake. Through the use of power profiling tools like J-Link and PowerScale, developers can focus their optimization efforts on power hot spots within the system as well as quickly identify and resolve unexpected power spikes. In this way, developers can meet the changing performance and responsiveness requirements of their application while achieving the optimal operating configuration for every case. 🦋

# Simplifying Embedded Design for Accelerated Time-to-Market

By Alec Bath, Applications Engineer, STMicroelectronics

Designing an efficient, real-time embedded system requires that each component has been optimized to operate in conjunction with the rest of the system. Today's MCUs accelerate system design by integrating many of the components an application requires and managing their interactions with an efficiency not possible when using external components.

The high level of intelligent integration in the STM32 family, for example, speeds time-to-market by enabling:

- » Greater performance through coordination of all system components
- » Elimination of throughput bottlenecks with a multi-layer bus interconnect
- » Increased responsiveness and determinism by offloading tasks from the CPU to reduce interrupts

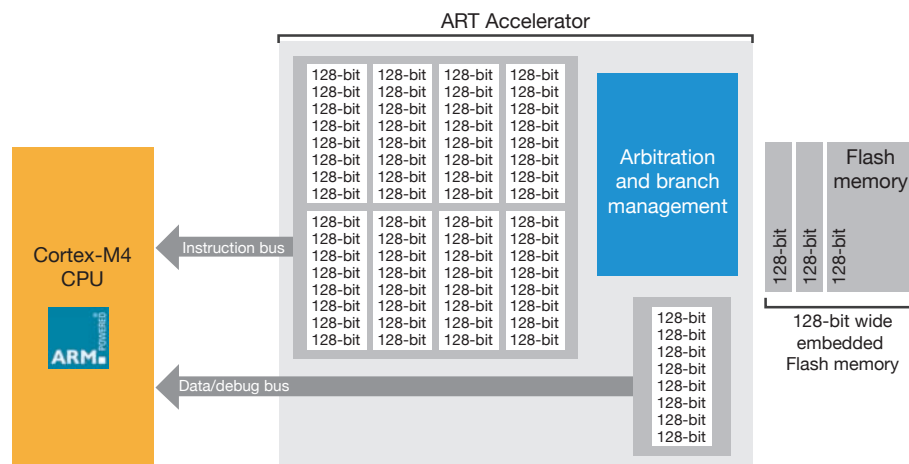
- » Simplification of design through flexible clocking and interface options
- » Increased system safety through self-monitoring
- » Lower system cost by reducing component cost and system complexity

## Greater Performance through Coordination

As embedded MCUs become more complex, their internal architectures can give rise to bottlenecks which can impede system performance. For example, executing code from embedded Flash can reduce an MCU's maximum performance because of wait states imposed by the Flash controller. Similarly, embedded accelerators may be unable to keep up with the MCU core, thus slowing system performance when their capacity is tapped. The result is that performance often degrades at higher core frequencies.

Architectures designed to compensate for known system bottlenecks can eliminate their negative impact on performance. For example, the STM32 architecture avoids losses in performance as core frequency increases through the use of Adaptive Real-Time (ART) memory accelerator technology.

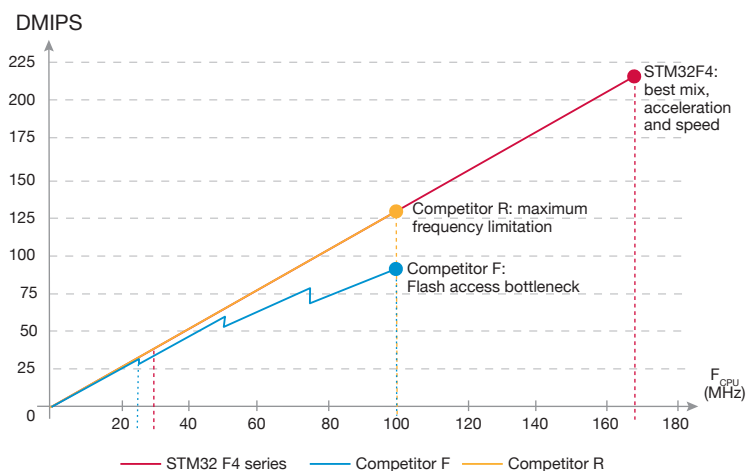
The ART Accelerator introduces a prefetch queue and branch cache between the embedded Flash and CPU core. Each time an event such as a subroutine call, interrupt, or conditional branch occurs and breaks the linear execution of the code, the ART Accelerator checks if this event has already been



**Figure 1** Integrated architectures can eliminate system bottlenecks by coordinating interactions between system components. For example, the ART Accelerator of the STM32 architectures buffers first instructions and constants so they can be placed into the prefetch queue to eliminate any performance losses associated with branching when executing from Flash.

stored in the cache (see Figure 1). If so, the first instructions and constants associated with this branch can be immediately pushed from the branch cache to the prefetch queue, thus eliminating any performance losses associated with branching. If the event has not yet been stored and its first instructions are not available, they are now stored in the buffer to prevent any delay the next time this event occurs. With a deep branch cache, most applications achieve performance equivalent to zero wait states when executing from Flash.

Figure 2 shows the impact of ART Accelerator technology on MCU performance when the MCU operating frequency reaches a point where the Flash becomes a bottleneck and imposes losses which can represent a significant percentage of overall performance. By eliminating these losses, the ART Accelerator provides consistent performance that scales linearly with core frequency. As can be seen, the STM32 F4 is able to provide its full 210 DMIPS performance at 168 MHz when executing from Flash.



**Figure 2** Flash bottlenecks that arise as operating frequency increases can represent a significant percentage of overall performance. By eliminating these losses, the ART Accelerator provides consistent performance that scales linearly with core frequency, allowing the STM32 F4 to provide its full 210 DMIPS performance at 168 MHz when executing from Flash.

With the increase of digital processing in embedded systems, whether for motor control, decoding of digital content, or high speed interfaces, MCUs need to move significantly more data than they have in the past.

The ART Accelerator is an excellent example of an architectural enhancement which is only possible when an MCU is highly integrated. Eliminating the wait states associated with branching requires intimate coordination between the Flash controller and the instruction pipeline. When multiple architectural enhancements like the ART Accelerator and those described below are combined in an MCU, the improvements in performance are tremendous.

### Multi-layer Bus Interconnect: Data Transfers without Involving the CPU

An essential element of a highly integrated MCU is its internal interconnect bus. With the increase of digital processing in embedded systems, whether for motor control, decoding of digital content, or high speed interfaces, MCUs need to move significantly more data than they have in the past. The interactions between the CPU, numerous accelerators, peripherals, and high-speed data interfaces can severely stress an MCU's

bus. Developers must be able to account for worst-case operating scenarios when multiple peripherals and interfaces are active at the same time, and how the bus is architected determines the overall level of contention and latency in the system.

With the myriad peripherals in a highly integrated MCU, a simple interconnect will not suffice; rather, a bus matrix which enables independent

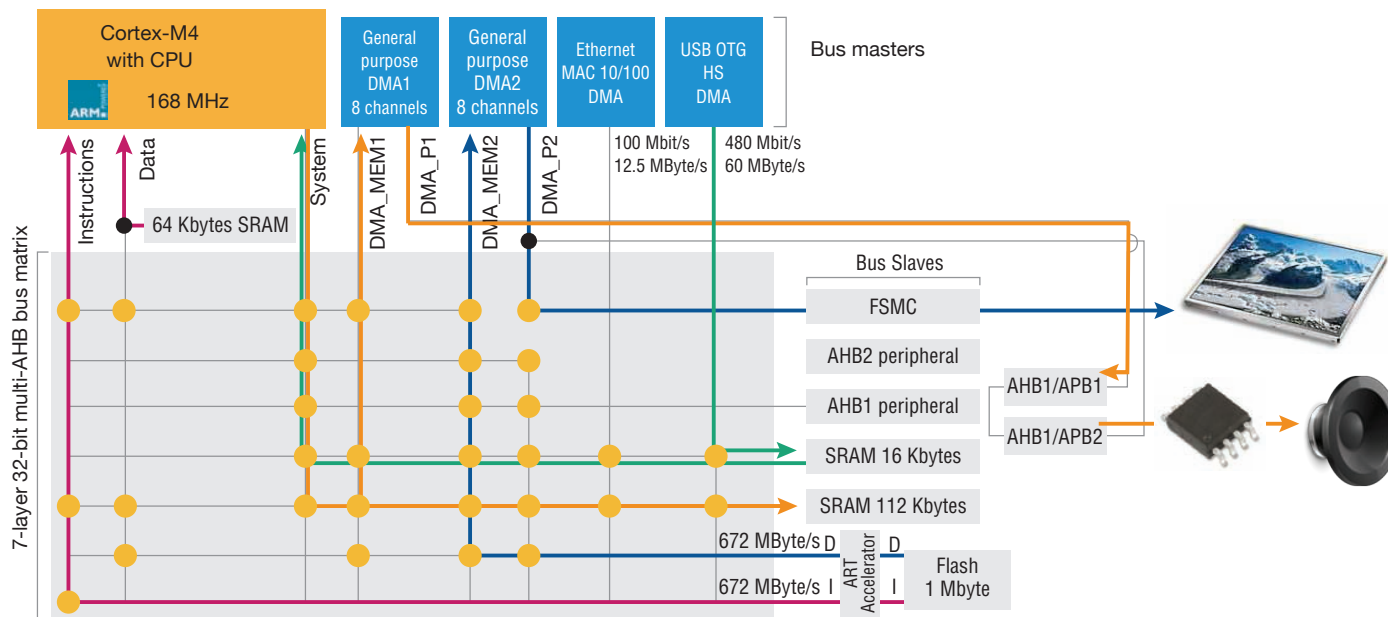
connections between masters and slaves is required. Figure 3 shows the 7-layer matrix that interconnects the STM32 F4 to enable seamless and efficient operation of the core and multiple peripherals simultaneously. Bus masters (shown along the top) include the CPU, the 2 DMA controllers, and the Ethernet and USB interfaces. Slaves, shown on the right, include the Flash memory through the ART Accelerator, 2

blocks of SRAM, the Flexible Static Memory Controller (FSMC), and the peripherals through the AHB bus interfaces. The nodes on the matrix represent the actual connections between the masters and slaves.

The flexibility and efficiency of this approach is illustrated in the figure with five simultaneous operations:

- » The core accesses Flash through the ART Accelerator

- » The core accesses the 112-Kbyte SRAM
- » The DMA2 controller transfers data from the camera interface located on the AHB2 peripheral bus
- » The DMA2 controller transfers camera data to an LCD connected through the FSMC
- » The USB OTG High Speed interface stores received data in the 16-Kbyte SRAM block



**Figure 3** The 7-layer matrix that interconnects the STM32 F4 enables independent connections between multiple masters and slaves. For example, five distinct data transfers are taking place simultaneously in this example. Note that the CPU is only involved in those transfers where it is a master.

The matrix interconnect offers a powerful improvement on performance. For example, each DMA in the system can operate, once configured, with no CPU overhead. In addition, because the Ethernet and USB OTG peripherals have their own dedicated DMA, they can operate simultaneously as long as they are not trying to access the same slave. As a result, multiple transactions can take place simultaneously and without interfering with each other, significantly increasing the internal throughput of the MCU and potentially eliminating data access as a system bottleneck.

It is important to note that the CPU does not need to be involved in transfers initiated by other masters. For example, rather than



**STM32<sup>®</sup> F4**

**World's highest  
performance  
Cortex<sup>™</sup>-M MCU  
168 MHz/210 DMIPS**



**STM32<sup>™</sup>** Releasing your **creativity**

For further information, visit [www.st.com/stm32f4](http://www.st.com/stm32f4)

a high-speed interface consuming most of a CPU's cycles with interrupts to receive or send the next byte of data, data can be received/stored or retrieved/sent in a way that does not involve the CPU. This provides the MCU with tremendous interconnect and data transfer capabilities to eliminate many traditional peripheral and memory access bottlenecks.

enhanced filtering, and power factor correction. In addition, application-specific peripherals further offload the CPU by accelerating processing of specific tasks in hardware:

**Cryptographic Accelerator:**

With the increasing number of embedded devices interconnected over the

I2C interfaces to provide robust communications for even the noisiest industrial environments.

**Nested Vector Interrupt Controller (NVIC):**

The NVIC allows the Cortex-M core to enter and exit an interrupt in just 12 cycles to enable highly deterministic MCU behavior. The process is completely

For example, Serial-Wire Debug supports up to 8 breakpoints and provides limited real-time data trace capabilities without needing to halt the CPU core.

An embedded application may also have a variety of sensors to monitor system health. If the temperature gets too high, for example, the system may need to turn on a fan or adjust a control algorithm (such as when managing LED lights) to compensate for the extra heat. Portable systems often monitor battery voltage so they can alert users to pending system shutdown. A motor control application may measure current or voltage as part of the feedback loop or as a check that the motor is not being overdriven. In some cases, such as when measuring temperature or battery voltage, sensors only need to be checked infrequently. With other sensors, such as when measuring current, the duty cycle of the sensor can be quite high.

Traditionally, sensor management has been a task for the CPU to handle. This means the CPU has to interrupt the current task, read the sensor, convert the captured reading, compare this value to an event threshold, and then take

Today, a single MCU can drive a motor with enough overhead to perform advanced processing, including precision control, enhanced filtering, and power factor correction. In addition, application-specific peripherals further offload the CPU by accelerating processing of specific tasks in hardware.

**Low to Zero Overhead**

Common operations which are executed frequently can be integrated into an MCU in a variety of ways. For example, in the early days of digital motor control, a separate DSP and MCU were required to provide basic control functionality.

Today, a single MCU can drive a motor with enough overhead to perform advanced processing, including precision control,

network, hardware-based cryptographic capabilities are required to ensure secure transactions. The STM32's integrated crypto/hash processor supports DES, 3DES, AES128, and AES 256 (up to 106 Mbytes/s) as well as SHA-1, MD5, and HMAC hashing.

**Hardware-based CRC:**

Developers can add CRC checking in hardware to SPI and

handled in hardware, allowing developers to write application code in C without the need for any assembly wrappers, thus simplifying software design.

**Serial-Wire Debug:** The Cortex-M architecture supports Serial-Wire Debug mode, which provides more functionality than a standard JTAG port while using fewer wires to accelerate system troubleshooting and verification.

appropriate action. Even with a low duty cycle, these operations can consume a significant number of processor cycles. The use of an analog watchdog takes advantage of the fact that for most sensors, the default response is to take no action unless a high or low threshold is exceeded. Developers can set the analog watchdog with high and low thresholds which trigger an interrupt if the sensor ever exceeds one of the thresholds. The only time the CPU is involved is if either threshold is exceeded. The result is that reliable sensor monitoring can be implemented with zero CPU overhead, regardless of the duty cycle of the sensor.

### Flexible Clocking

Depending upon the MCU family, numerous clocking options are available. For example, multiple integrated PLLs allow developers to easily run the core and peripherals at different frequencies as well as generate specific frequencies such as those required for audio codecs or a USB link. In addition, frequencies can be quickly scaled down without incurring latency to unlock a PLL when dropping into low power modes.

The integration of industry standard interfaces reduces system cost by allowing developers to quickly and easily develop products that connect to other devices or components with minimal design effort.

For applications which don't need a crystal for precision clocking (i.e., the system does not need to support a high-speed USB interface), an internal RC oscillator that can be trimmed to 1% accuracy is sufficient for most applications except those that require an even more precise external crystal (i.e., USB), thereby reducing component count and system cost. The STM32 family supports an 8 MHz (16 MHz for the STM32 F4 family) internal RC oscillator. In addition to this high speed clock source, STM32 L1 devices have a multi-speed internal oscillator that allows ultra-low power operation from 64 KHz to 1 MHz with fast wake up. Also, all STM32 devices have a low-speed oscillator that runs a watchdog timer independently of other system clocks and drives the real-time clock (RTC) peripheral.

The STM32 architecture also offers multiple general-purpose timers. Because timers can be linked and cascaded with each other, they can be used to manage low frequency tasks without requiring the MCU to handle timer rollover in software. This can result in a significant savings in processor cycles and increased determinism through a reduction of the total number of interrupts the system handles. Timers can be used to trigger peripherals together, such as is required to synchronize output stereo audio signals.

Dedicated Timer Functions are also available to support a variety of applications, including quadrature encoders, Hall sensors, and 3-phase motor control. For example, 3-phase motor control enables finer control of motor torque and speed by synchronizing three pairs of PWM outputs. Less

software overhead is incurred because part of the function is performed in hardware by the dedicated timer function.

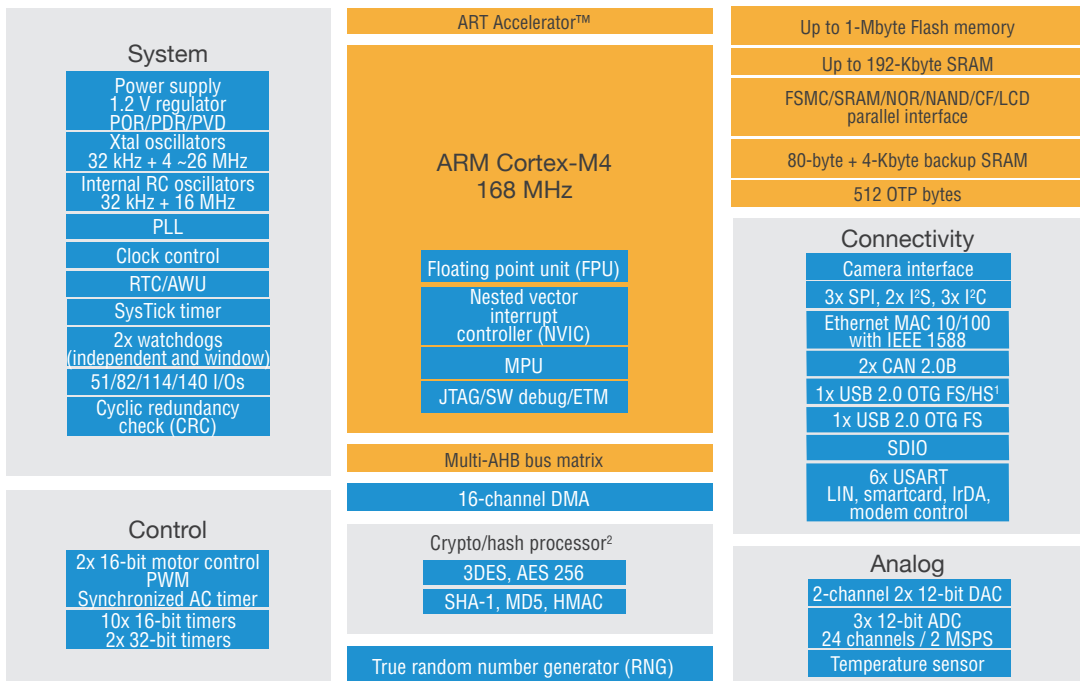
### Extensive Interfaces

Today's embedded systems also need to be able to interconnect with a wide range of off-chip components as well as network with other devices. The integration of industry standard interfaces reduces system cost by allowing developers to quickly and easily develop products that connect to other devices or components with minimal design effort:

**Audio Applications:** I<sup>2</sup>S and USB peripherals with advanced dual PLL and data synchronization schemes provide digital audio support.

**High Speed USB OTG** (external PHY required): The STM32 F4 offers two USB peripherals, both supporting On-The-Go (OTG) so a device can be a host or





**Figure 4** STM32 F4 MCUs offer a rich variety of performance, peripherals, and memory options to ensure that manufacturers pay only for the capabilities they need.

peripheral. Both support Full Speed USB and have integrated Full Speed PHYs. Additionally, one of the peripherals also supports High Speed USB. Full host and device drivers are available to speed design.

**Ethernet MAC 10/100, with IEEE 1588 V2 hardware support:**

IEEE 1588 is an important standard for time-synchronizing different Ethernet devices on the network. This is

required for applications such as industrial control where each device must act in concert within a few nanoseconds. The MAC supports either standard MII or RMII, giving developers the flexibility to connect to the PHY of their choice. Performance is further enhanced through the use of a dedicated DMA controller.

**Camera interface:** STM32 MCUs have a flexible 8- to 14-bit parallel interface which supports a

number of different color formats and allows devices to easily capture images or moving video at up to 48 Mbytes/s at 48 MHz.

**Flexible Static Memory Controller (FSMC):**

Rather than locking developers into using a limited type of memory, the FSMC is capable of driving different types of static memory or supporting an external display, depending upon the needs of the application. The FSMC runs

at up to 60 MHz and supports different topologies so that it can interface with SRAM, pSRAM, Flash, NAND Flash, and others.

**USART:** MCUs that offer a UART typically only support asynchronous communications, which limits the types of interfaces the MCU can implement. By supporting both synchronous and asynchronous communications, the STM32’s Universal Synchronous/Asynchronous Receiver-Transmitter (USART) is able to support interfaces like LINBUS or IrDA without the need for any other specialized hardware. In addition, interface driver libraries are available, including a fully-configurable I2C library, giving developers the flexibility to implement whichever interfaces they need for a particular application. Finally, the STM32 F4’s USARTs are fast, operating up to 10.5 Mbps.

**Consumer Electronics Control (CEC):**

The CEC interface is defined within the HDMI spec and allows different consumer electronics devices to control each other over HDMI.

**Up to 140 GPIO:** STM32 MCUs provide enough I/O for nearly embedded application. In

addition, these I/Os are high performance and can toggle at up to 60 MHz.

The extensive variety of the STM32 family, with more than 250 different devices in its portfolio, ensures that the optimal mix of performance, memory, and peripherals is available so manufacturers pay only for the capabilities they need. Devices range from the minimum level of integration required for real-time embedded applications all the way up to the STM32 F4 with many optional peripherals and/or multiple peripherals (i.e., STM32 F4 MCUs offer as many as 3 SPI, 3 I2C, and 6 USARTs) for applications which need them (see Figure 4). Extensive peripheral libraries provide driver code for all MCU peripherals and ensure compatibility of application code across the various STM32 product families. In addition, ST offers its MicroXplorer development tool which assists developer in defining and mapping peripherals to the MCU's pins.

### Increased Safety

Integrating protective functionality into an MCU can increase system robustness and reliability by alerting the MCU of

pending system failure so that the system can be shut down safely before personnel injury or property damages can occur:

**Reset supervisor:** This function keeps the MCU in a safe state when the input voltage is out of range, such as when the system is turned on/off. It does so by asserting the reset line until the voltage is back in range. This keeps the MCU from incorrectly fetching instructions or transitioning I/O pins.

**Brown-out detection:** This is an early warning that triggers an interrupt when the voltage drops below a certain software-configurable range. Brown-out detection is critical for applications such as medical or industrial which must perform a graceful shutdown upon pending power failure.

**Windowed watchdog timer:** A watchdog timer ensures that the system is not locked in a loop by requiring the system to periodically reset the watchdog to prove it is still active. A windowed watchdog timer also requires that the reset happen within a certain time frame to prevent the system from being locked in a loop which resets the watchdog timer.

**Dual watchdog timer:** A second watchdog timer runs off the internal low-speed oscillator. In this way, regardless of what happens with the other system clocks, the system is guaranteed to have an operational and reliable watchdog timer.

### Lower Cost, Component Count, and Complexity:

Many integrated peripherals not only reduce system cost by eliminating the need for specific external components, they also shrink the overall device size while simplifying design. For example, the STM32 architecture offers:

**Voltage Regulator:** By integrating the voltage regulator, STM32 MCUs can operate off a single voltage without requiring an external component. This also gives developers the flexibility, in the case of the STM32 F4, to supply from 1.8 to 3.6 V to the MCU.

**Multiple PLLs:** An integrated phase-locked loop (PLL) allows the MCU to be clocked at high frequencies using a low frequency crystal. Having multiple on-chip PLLs allows every frequency an MCU needs to be generated from a single clock source. For example, the

core could be operated at 168 MHz while the USB peripheral receives a 48 MHz clock while the Ethernet peripheral receives either 25 or 50 MHz.

**Real-Time Clock:** Integrating the real-time clock not only lowers system cost, it enables the MCU to keep time when the system bus voltage is removed (i.e., the system is powered by a coin cell battery).

The STM32 architecture has been designed to provide the performance and capabilities required by today's real-time embedded systems. By offering the highest level of integration technology available today, STM32 MCUs achieve optimal performance by minimizing CPU loading, allowing simultaneous data transfers, and eliminating system bottlenecks. The flexible configuration of clocking and interface options, combined with an extensive range of application-specific peripherals, simplifies design while reducing component count and system cost. Together, these factors enable developers to create reliable and efficient systems based on the STM32 architecture that can be brought to market quickly and easily. 🦋

# Designing Efficient Connectivity

By Christian Légaré, Vice President, Micrium  
Bob Waskiewicz, Application Engineer, STMicroelectronics

Many embedded systems, from sensors to handheld medical units to energy meters, can perform more efficiently and offer more intelligent operation when they are interconnected. Network connectivity extends a wide range of functionality to applications, allowing performance information, control data, and other real-time traffic to be exchanged between devices to improve system performance, reliability, and versatility.

In the past, introducing high-speed communications to a system could be challenging given that embedded systems tend to be fairly constrained in terms of available memory and processing resources. With new MCUs like ST's STM32 F1 Connectivity Line, STM32 F2, and STM32 F4 that have been designed to provide turnkey communications capabilities, developers can implement interfaces in a reliable way that provides sufficient bandwidth with low latency while leaving enough headroom on the MCU

to support the main system application.

## Connectivity versus Throughput

The first question to consider when adding connectivity to a system is whether an application requires high throughput or just simple connectivity. Consider the data needs of a smart meter. Relatively little bandwidth is required between the smart meter and a washing machine to collect power usage information or to enable time-of-day/pricing-based controls. What matters most in this case is basic connectivity. In contrast, a home appliance could be connected to the Internet to serve up video showing the user how to operate or service the appliance. Because of the large size of video streams and the low latency requirements associated with real-time data, throughput becomes a critical design constraint.

The difference between connectivity and throughput

is substantial. Connectivity is relatively simple to implement, especially if the application can tolerate high latency. When performance does not matter, the system can implement fewer and smaller buffers, thus conserving memory resources. It is when throughput matters

a time, the longer it takes to transmit data, the lower the responsiveness and precision the system will have. For such an application, implementing a 100 Mbps interface will reduce latency significantly compared to an interface operating at only 10 Mbps.

Once basic connectivity is in place, it is relatively straightforward to expand functionality by introducing new services.

that the design of an interface becomes more complex and potentially expensive.

Note that the amount of data to exchange is not the only factor determining the need for high throughput since latency impacts the operation and reliability of many real-time embedded systems. For example, even though an industrial controller distributing control data via Ethernet may only need to transfer a handful of bytes at

Once basic connectivity is in place, it is relatively straightforward to expand functionality by introducing new services. When a system already has a TCP/IP stack, then email, FTP, web browsing, and web serving functionality can easily be added to the stack. For example, Micrium's  $\mu$ C/TCP-IP stack enables developers to include these services, as well as custom functions, and access them through an intuitive

Bytes/Packet	10 Mbps packets/sec ( $\mu$ sec/packet)	100 Mbps packets/sec ( $\mu$ sec/packet)
64	19,531 (51.2)	195,312 (5.12)
256	4,883 (204.8)	48,828 (20.4)
1024	1,221 (819.2)	12,207 (81.9)
1518	823 (1,214)	8,234 (121.4)

**Table 1** Rate at which an MCU must be able to process packets

API. These services can run concurrently and be tuned for a specific application, thereby simplifying stack implementation and management. The primary design challenge is balancing data throughput to match the services offered so that the CPU's capacity is not exceeded.

For high data rate applications, the processor has to have the capacity to create/transmit and receive/use packets at the desired rate. Table 1 shows the rate at which the MCU must be able to process packets across a 10/100 Mbps Ethernet connection. Note that the total amount of data transferred does not change these figures; i.e., whether the system is receiving 10 packets or 1000 packets, each packet must be processed before the next packet arrives to prevent loss of data.

Factors which affect the ability to process and use packets

include CPU operating frequency, memory allocated for the TCP/IP stack, CPU bandwidth available to the stack, stack efficiency, and how data is moved within the MCU. The priority of network communications compared to the application's primary function also comes into play. For some applications, communications and application processing are intimately tied together and so have correspondingly similar priority: without any data, the application has nothing to operate upon. For example, an industrial control system will need to balance update frequency with the complexity of calculations it needs to perform during each update. For other applications, such as a smart meter, communications is a secondary function which can be delayed if the application requires the CPU's full attention. For these types of applications, assuming

the presence of a real-time kernel such as  $\mu$ C/OS-III, developers will need to lower the priority of TCP/IP stack operations so that communications does not negatively impact application performance.

### Turnkey Connectivity on a Chip

While the entire STM32 platform from ST integrates a variety of communications interfaces, three of the families have been optimized to provide advanced communication peripherals—including an Ethernet MAC and USB Host/OTG controller—which speed development, increase link performance, and reduce system cost. Providing turnkey Ethernet and USB connectivity with support for consumer audio, these MCUs are also the controller of choice for CAN gateways. Each family offers a range of Flash and RAM sizes and interfaces to enable developers to balance memory and performance for every application:

#### STM32 Connectivity Line:

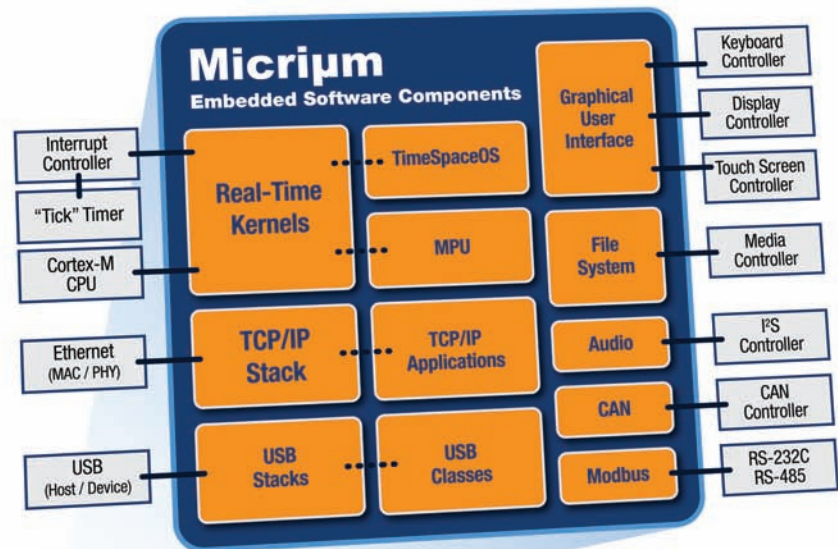
Providing economical networking capabilities for consumer and industrial applications, STM32 Connectivity Line MCUs are based on the ARM Cortex-M3 and offer up to 72 MHz with

256 KB Flash and 64 KB SRAM. Integrated interfaces include an embedded Ethernet MAC with dedicated DMA and IEEE 1588 precision time protocol hardware support, USB 2.0 OTG Full Speed controller, 2 CAN ports, and two audio-class I<sup>2</sup>S interfaces.

**STM32 F2:** The STM32 F2 brings more performance to connected applications, offering a Cortex-M3 core operating at up to 120 MHz with 1 MB Flash and 128 KB SRAM. In addition to an integrated cryptographic accelerator, on-chip interfaces include an embedded Ethernet MAC with dedicated DMA and IEEE 1588 precision time protocol hardware support, two USB 2.0 OTG controllers (Full Speed/High Speed), 2 CAN ports, two audio-class I<sup>2</sup>S interfaces, and a camera interface. The STM32 F2 also has 528 bytes of one-time programmable (OTP) memory for reliably storing critical user data such as Ethernet MAC addresses or cryptographic keys.

**STM32 F4:** ST's flagship MCU, the STM32 F4, is based on the Cortex-M4 core and provides 168 MHz of performance (210 DMIPS) with up to 1 MB Flash and 192 KB SRAM for the

# Take your STM32-Based Products to the Next Level



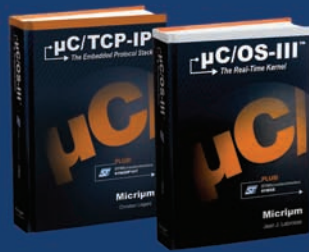
**Royalty free**  
**Reduce your time to market**  
**Kernel source code available for download**  
**Modular approach – buy and use only what you need**

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](http://www.Micrium.com)

most demanding connected applications. In addition to introducing hardware-based FPU, DSP, and cryptographic capabilities to increase the performance of compute-intensive applications, the STM32 F4 integrates an embedded Ethernet MAC with dedicated DMA and IEEE 1588 precision time protocol hardware support, two USB 2.0 OTG controllers (Full Speed and High Speed), two CAN ports, two audio-class I<sup>2</sup>S interfaces, and a camera interface. The STM32 F4 also has 528 bytes of one-time programmable (OTP) memory for reliably storing critical user data such as Ethernet MAC addresses or cryptographic keys.

STM32 MCUs are capable of higher data rates because of their streamlined architecture. Using a multi-layer bus interconnect and dedicated DMA, the integrated Ethernet controller can simultaneously move data directly to and from main memory without requiring any cycles from the CPU. Integrating the Ethernet MAC into the STM32 architecture also lowers processing latency since data does not have to then be transferred between ICs while at the same time reducing system

size and cost. While some MCUs integrate the PHY as well, this limits overall flexibility by fixing the type of interface the system can support. STM32 MCUs do not integrate the PHY so that developers have complete freedom to utilize different Ethernet technologies using the same base product design.

The flexible clocking architecture of STM32 MCUs further reduces system cost while improving power efficiency. Depending upon the application, a single clock source can be distributed to supply each of the different frequencies required to drive the system, USB, and Ethernet clocks. Through the use of PLLs and prescalers, the system clock also drives the MCU’s various peripherals.

The STM32 Connectivity Line, STM32 F2, and STM32 F4 MCUs also integrate dual I<sup>2</sup>S interfaces with a separate PLL for generating precise frequencies to support high-quality audio playback. Combined with an integrated USB port and ST’s STM32 audio software supporting MP3 and WMA formats with channel mixing, standalone 3-band parametric equalizer, and loudness

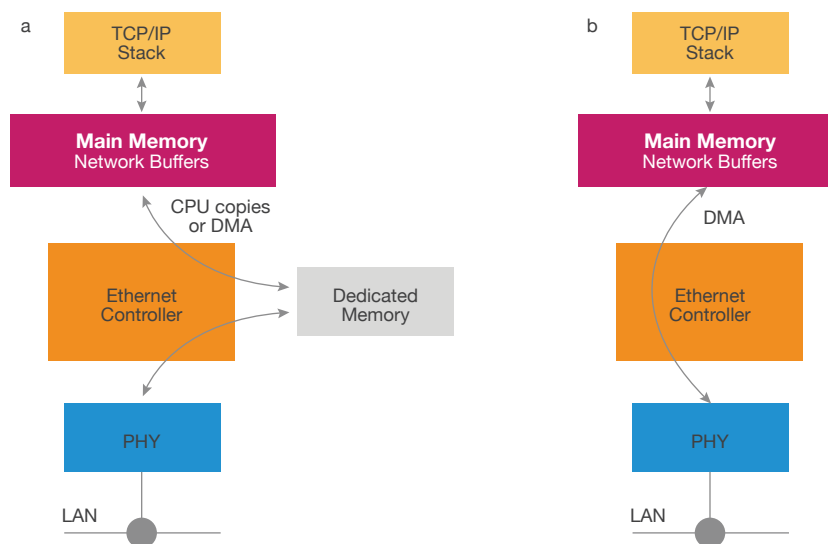
control, these MCUs meet the requirements for most embedded audio applications.

Finally, STM32 MCUs give developers the flexibility to remap peripherals and interfaces within the device. The ability to define pin outs not only simplifies initial design and layout, it facilitates migration between each of the various STM32 families to allow developers to quickly spinoff high-end or low-cost products from the same base design.

## Efficient Communications Stacks

Because the stack forms the heart of the communications interface, it may be in constant use, meaning it will have a significant impact on how the overall system operates. However, simply knowing the theory behind a protocol like TCP/IP will not improve link efficiency. What matters is the manner in which a particular stack operates and how its use of system resources and interrupts will impact application performance. Rather than making function calls blindly, by understanding the implications of each stack call, the stack can be used in the most efficiency way.

For example, for a lightweight stack to have a smaller footprint than a full stack, certain functionality must be dropped or implemented less efficiently. If an important feature like congestion control has been eliminated, performance may suffer since the available bandwidth will be consumed with more retransmissions. Similarly, when stack execution speed is sacrificed for footprint, latency will be negatively affected. If the stack is optimized for IPv4, this may create issues when the system is connected to IPv6-based devices. For example, the network infrastructure will require the use of a dual stack router so that the IPv4-only device will need to be connected to this special router to tunnel the IPv4 packets into IPv6 packets onto the IPv6 networks, adding unnecessary processing to the IP packets and affecting performance. Stacks that are based on a polling mechanism will also consume more CPU cycles than interrupt-based stacks. Depending upon the application, short cuts taken in a stack's design may result in greater contention for an MCU's processing or memory resources, effectively resulting in higher system cost.



**Figure 1** a) An Ethernet controller which stores received data in dedicated memory requires CPU involvement to transfer data to the network buffers of the TCP/IP stack. b) Micrium's  $\mu$ C/TCP-IP stack supports the advanced performance capabilities of the STM32 architecture to implement a true zero-copy architecture where data can be transferred directly to and from application memory, eliminating the need for the CPU to copy data to and from buffers in the TCP/IP stack.

For high performance applications, the stack needs to be ported to the MCU. Micrium's  $\mu$ C/TCP-IP, for example, supports the advanced performance capabilities of the STM32 architecture, making full use of not only base Cortex-M features such as the Count Leading Zero (CLZ) instruction and new Cortex-M4 FPU and DSP instructions but also the STM32 architecture itself, including the integrated Ethernet MAC, multiple DMAs, multi-layer bus interconnect, and integrated CRC engine.

$\mu$ C/TCP-IP also utilizes a true zero-copy architecture – data can be transferred directly to and from application memory, eliminating the need to copy data to and from buffers in the operating system (see Figure 1). In addition, developers have full control of stack task priority so they can balance link performance with real-time application requirements.

For applications where performance and cost are critical, a full stack with source code enables developers to customize

the stack configuration. The  $\mu$ C/TCP-IP stack is fully documented and implemented using a consistent coding style. Components can be individually selected, and code can be optimized by the compiler to match the system's performance and memory needs. Reference designs provide a robust foundation to enable developers to quickly build a stack optimized for their application. The kernel has also been certified in many safety-critical applications.

For Ethernet-based applications, developers can select between TCP and UDP. For reliable connectivity, TCP guarantees packet delivery through a connection-oriented protocol where the receiver acknowledges each transfer. Connections can remain active for long periods of time. Developers can also run a variety of high-level application layers on top of TCP, including HTTP for serving web pages, FTP for transferring files, and SMTP and POP3 for mail services. Designed for reliable stream transfers, TCP is ideal for non-time sensitive data transfers. The types of devices and services the system is going to connect to will dictate which optional features need to be included in a stack.

Some applications do not require the robustness or reliability of TCP, especially directly connected devices which do not have to contend with other data sources (i.e., controller to servo) and those sending real-time data which is obsolete before it can be resent. In these cases, UDP offers a faster alternative to TCP where packets are sent and then forgotten. Providing quick and simple single-block transfer capabilities, data can be sent without first having to establish a connection. Applications include those handling time-sensitive data such as audio and video as well basic network services such as DHCP (Dynamic Host Configuration Protocol), DNS (Domain Name Service), TFTP (Trivial File Transfer Protocol), and SNTP (Simple Network Time Protocol). See Table 2 for a comparison of TCP and UDP.

### Tuning Communications Performance through RAM

With up to 1 MByte of onboard Flash (STM32F2 and STM32F4), STM32's MCUs provide more than enough code space to enable developers to optimize stack performance for speed as well as include higher level

	UDP	TCP
Service	Connectionless	Connection-oriented
Data Verification	Yes	Yes
Rejection of erroneous datagrams/segments	Yes	Yes
Sequence Control	No	Yes
Retransmission of erroneous and lost segments	No	Yes
Reliability	Low	High
Delay Generated	Low	High
Total Throughput	Higher	Lower
Service Type	Quick and simple single block transfers	Reliable stream transfers
Applications	<ul style="list-style-type: none"> <li>» Network services with short queries and answers</li> <li>» DHCP and DNS</li> <li>» Time-sensitive data that can cope with minimal packet loss, including voice, video, audio, repetitive sensor data, etc.</li> </ul>	Non-time sensitive data transfers, including file transfers, web pages, email, etc.

**Table 2** Comparison between UDP and TCP

connectivity features in their application. For example, Micrium's TCP/IP stack with all options requires less than 60 KBytes even when optimized for speed. For another 30 KBytes or so, the following popular application-layer services DHCP, SMTP, and POP3 can be included as well.

Where developers may be required to begin making tradeoffs is when allocating RAM. RAM is an essential factor in determining the overall communications throughput of a system. In general, the more RAM that is available for buffers within the communications stack, the faster protocol processing

will be. However, the same holds true for application code, making RAM an important resource to allocate with care.

One of the most important factors in fine-tuning TCP performance is providing enough buffers for the TCP Receive Window size. In general, if the receive window is lower than the product of latency and available bandwidth, the system will not be able to fill the connection at its capacity since the client cannot send acknowledgements back fast enough. Effectively, there needs to be at least a certain number of packets in transit on the network to make sure the TCP stack will have enough packets to process while accommodating packet latency. The system will also need 3 or 4 additional buffers to allow the system to send ACK message for TCP packets received. The more RAM that is available, the more efficient stack processes can be. Systems can be configured with less buffers but connection speed will drop because of flow-control effects or because of the increased number of retransmissions generated.

STM32 F4 MCUs provide up to 192 KBytes SRAM, thereby

simplifying performance tuning and giving developers substantial flexibility in how they allocate available memory. For example, configuring the TCP/IP stack to have ten large transmit and receive buffers consumes only approximately a fifth of the available SRAM. More or less buffers can be allocated to match an application's specific performance requirements.

To offload the CPU and increase link performance, the STM32 multi-layer bus interconnect allows for transfers to each of the separate SRAM blocks simultaneously using DMA. This enables, for example, TCP/IP transmit and receive data to be transferred concurrently. To accelerate processing when multiple interfaces are active, the STM32 architecture has a separate DMA for both the USB and Ethernet peripherals. The result is that data can be received from several interfaces and stored in buffers without involving the CPU.

Developers can tune TCP/IP performance by configuring different buffer sizes based on the type of data the system will be working with. If the network is open and any compatible

device can be connected, then the input buffers will have to be able to handle the largest possible incoming packet (i.e., 1518 bytes) to avoid exceeding the buffer limit and either losing data or overwriting other application data. If the network is closed and attached devices can be guaranteed to send smaller packets, then smaller buffers can be used.

On the transmit side, buffer size can be optimized for small or large data packets since the system knows how much data it will be sending. To conserve RAM, the system can be designed to produce smaller packets. Note that this is different than segmenting larger data blocks into smaller ones, as segmentation potentially requires its own buffer and that data be copied an additional time as well.

Developers can measure actual stack performance using unbiased benchmarking tools like iPerf which provide transmit and receive throughput for both UDP and TCP. Using iPerf during development and testing can provide valuable insight into how the TCP/IP stack and application impact each other's performance. Developers new

to TCP/IP should remember that running iPerf with just the stack active can create unreasonable throughput expectations since the performance figures do not reflect system-level performance.

Balancing RAM should be done from a system-level perspective early in the design process to ensure that RAM is being used in the most efficient manner. A system may support several interfaces—i.e., Ethernet, USB, and SPI for an industrial application—which may need to run concurrently and thus cooperatively share the same processing resources. Each application-layer service running on top of the TCP/IP requires RAM as well, as does a graphical user interface (GUI). At some point, it may make sense to give the system some breathing room by adding external RAM. This decision needs to be considered well before going to hardware.

### Leveraging a Real-Time Kernel

Depending upon the application, using a real-time kernel can significantly improve overall performance by enabling more efficient use of the CPU. For example, the most common method for communicating using



TCP/IP is using a BSD (Berkeley Software Distribution) Socket. Sockets allow for bidirectional communication between a source and destination. A system can have many sockets, with some carrying UDP data and others carrying TCP streams. Developers need to be careful when using sockets because they can block the system until an operation is complete (i.e., a socket cannot complete a *recv()* function until data is actually sent by a remote host). For this reason, blocking sockets are only used when a real-time kernel that allows other threads to run is part of the product architecture.

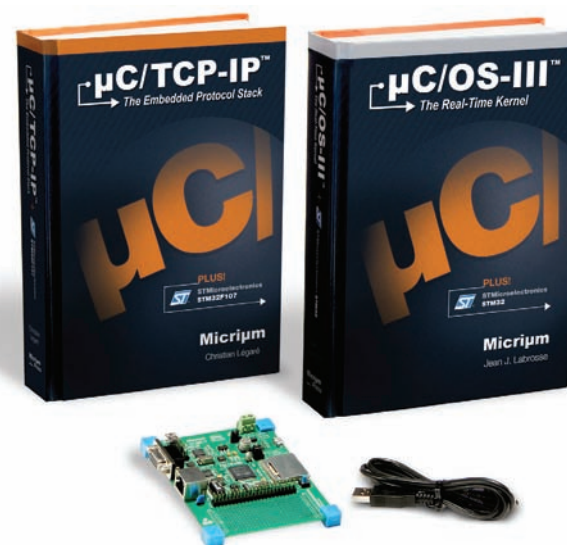
For systems without a real-time kernel, non-blocking sockets must be used and the socket API returns an error value as to whether the call was successful or not (i.e., data was/was not received). The disadvantage of this approach is that it effectively requires the system to poll the socket, forcing low power systems to wake more frequently than might otherwise be necessary.

When using sockets with a real-time kernel, care should be taken not to use non-blocking sockets because of the tendency of TCP/IP

services to “starve” low priority applications tasks. Micrium’s  $\mu$ C/TCP-IP stack allows developers to adjust the priority of the various stack components so that mission-critical or real-time application tasks can be placed ahead of or behind communications processing based on the functional needs of the application. In addition, developers have the option of using Micrium’s  $\mu$ C/OS-II or  $\mu$ C/OS-III which were the very first kernels to be ported to the ARM architecture.

Micrium also offers  $\mu$ C/Probe, a Microsoft Windows-based product that displays run-time data using a variety of GUI-based objects (gauges, meters, bar graphs, numeric indicators, plots and more) and provides full run-time visibility into the  $\mu$ C/OS-II/III kernels and  $\mu$ C/TCP-IP stack.  $\mu$ C/Probe connects to a target using an IAR J-Link, RS-232C, or TCP/IP connection without requiring developers to write any code.

Developers interested in evaluating the STM32 Connectivity Line MCU and Micrium’s  $\mu$ C/OS-II/III and  $\mu$ C/TCP-IP stack can readily do so by obtaining the  $\mu$ C/OS-III and  $\mu$ C/TCP-IP books targeted for



**Figure 2** The  $\mu$ C/Eval-STM32F107 evaluation board and  $\mu$ C/OS-III and  $\mu$ C/TCP-IP books available from Micrium are ideal for evaluating the STM32 Connectivity Line of MCUs.

the STM32 as well as the  $\mu$ C/Eval-STM32F107 evaluation board (available from Micrium) which has built-in J-Link SWD debugging capabilities (see Figure 2). The book also provides access to free evaluation tools from IAR allowing developers to experience the full power of STM32 MCUs,  $\mu$ C/OS-III, and  $\mu$ C/TCP-IP.

Today’s embedded systems can provide significant value when interconnected to other devices, and STM32 MCUs provide the ideal platform for efficiently implementing real-time interfaces over Ethernet, USB, CAN, and I<sup>2</sup>S in a cost-effective manner.

Built upon the powerful Cortex-M architecture with integrated interface peripherals, separate DMAs, and a multi-layer bus interconnect that enables data transfers without involving the CPU, developers can select the optimal connected MCU from a variety of performance, peripheral, and memory options among the STM32 Connectivity Line, STM32 F2, and STM32 F4 families. In addition, with the availability of real-time kernels like  $\mu$ C/OS-II/III from Micrium and their  $\mu$ C/TCP-IP stack, developers can introduce turnkey connectivity to a wide range of embedded applications. 🦋

# Accelerating Next-Generation Design Through IP Reuse

By Reinhard Keil, Director of MCU Tools, ARM Germany GmbH  
Andrew Frame, Senior Product Manager, ARM Ltd  
James Lombard, Applications Engineer, STMicroelectronics

Given ever-changing application requirements, it is critical for developers to be able to easily migrate hardware and software designs between different MCUs. For example, a feature added to a system late in the design cycle may push processing requirements beyond the capabilities of the current MCU. Without a flexible architecture and broad selection of devices, developers may find themselves unable to complete the system.

Conversely, developers need to be able to scale down systems as well. From a design standpoint, it is much easier to work out the initial design of a system using a higher performance processor with more memory than may be necessary for the final production application. Once the scope of the design has been determined, the processor can be cost-reduced to an MCU that has

a more optimal balance of performance and memory.

Flexibility of the MCU architecture as well as the ability to reuse IP is also essential for bringing next-generation designs to market. Developers need to be able to not only scale the performance and memory of a system's MCU but introduce new functionality or power efficiency where it is needed to meet different price points. For example, systems built around the STM32 F2 architecture can be migrated to the STM32 F4 family—the industry's highest performance Cortex-M device to date—to build a high-end version of a system with more advanced features, including greater precision, faster responsiveness, a GUI-based interface, and other compute-intensive features. Similarly, the same design targeted for portable applications can be migrated to

the EnergyLite™ STM32 L1 ultra-low power line of MCUs.

The difficulty of migrating a system to a different MCU is determined by how much of the system will need to be redesigned. If the degree of changes required is small, typically an MCU within the same family offering can be used. However, at some point developers will need to move to a different MCU family. Low-end MCUs simply aren't designed to handle high-performance tasks like audio or video processing. Similarly, a high-performance MCU cannot achieve the power efficiency of an architecture designed specifically for portable applications.

Consistency without sacrificing efficiency or flexibility is essential for ease of migration. This consistency must encompass every aspect of the MCU architecture, from its low-level

hardware including pin layout and peripheral compatibility up through to application code, drivers, and development tools.

To meet this need, ST offers the STM32 architecture which provides a high level of consistency between its more than 250 devices. Through technologies such as the ST Standard Peripheral Library and the Cortex Microcontroller Standard (CMSIS), low-level implementation details can be kept transparent to enable code to be reused on a different MCU as simply as reconfiguring the compiler. In this way, developers can easily migrate designs across the four STM32 series to quickly bring product line extensions to market without a redesign.

## Code Compatibility

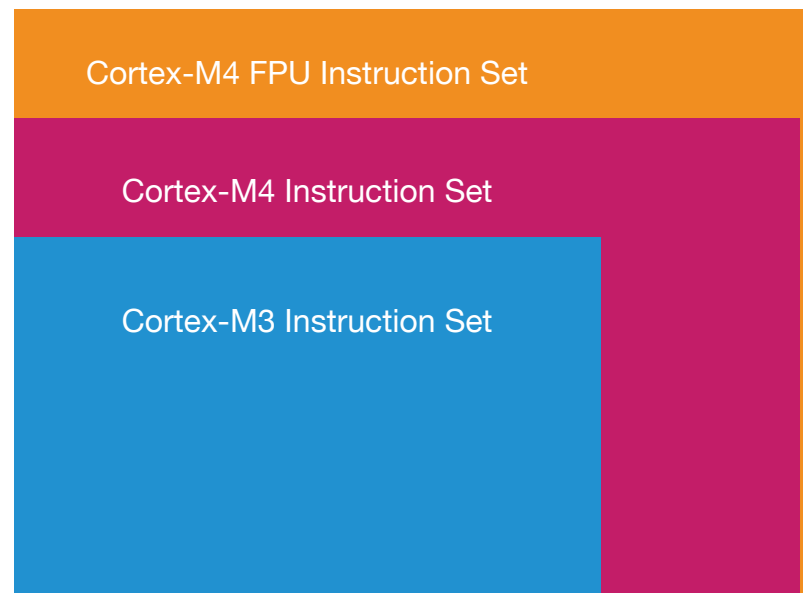
Key to the portability of application code is ability to work above the specific

implementation details of the MCU using a higher-order language like C. MCU and DSP architectures which require critical-loop code to be written in assembly to achieve the necessary levels of performance impose upon developers a plethora of details that need to be managed. Such code also tends to be extremely processor-specific, making it difficult to migrate between families.

The ability to reuse code, especially complex algorithmic code, is essential for fast time-to-market. Because each STM32 MCU is based on the ARM Cortex-M architecture, code is fully upwards compatible across all STM32 families. Specifically, the Thumb-2 instruction set provides a consistent instruction set among STM32 devices and improves performance and memory efficiency through an optimized blend of 16- and 32-bit instructions. This means that an STM32 F4 MCU based on the Cortex-M4 core supports all of the features of a Cortex-M3-based STM32 F2 MCU while introducing powerful new capabilities that increase program efficiency and reduce code size (see Figure 1).

The ability to work in C and take full advantage of the architectural enhancements of each MCU is an important aspect of the STM32 architecture. In addition to offering a more simple learning curve for faster application development, C is easier to maintain as well as reuse when migrating a design. The compilers available for STM32-based development, including MDK-ARM from Keil, have been specifically optimized for the architecture, providing highly efficient code that utilizes the capabilities of each MCU to its fullest. In this way, developers are able to exploit each aspect of the optimized STM32 architecture—including its Adaptive Real-Time (ART) memory accelerator, multi-layer bus interconnect, and dedicated DMAs—without having to lock code to a specific processor.

Low-level implementation details are taken care of at all levels of an application. At the driver level, compatibility is achieved using the ST Standard Peripheral Library. For advanced DSP algorithms, the CMSIS DSP Library automatically handles the migration of existing application code to take advantage of the advanced DSP and FPU features



**Figure 1** Because each STM32 MCU is based on the ARM Cortex-M architecture, code is fully upwards compatible across all STM32 families. This means that an STM32 F4 MCU based on the Cortex-M4 core supports all of the features of an STM32 F2 MCU based on the Cortex-M3 core while introducing powerful new capabilities that increase program efficiency and reduce code size.

of the Cortex-M4 architecture. In addition, the STM32 architecture itself takes care of many low-level implementation details. For example, when an interrupt is triggered, the MCU will automatically push vulnerable registers to the stack, resolve interrupt priority, and prepare to execute the first line of C code for the interrupt.

## Compatibility Beyond Code

An essential element for ease of portability of code is consistency through the use of programming standards. Code compatibility, while important, is only one part aspect of migration. Compatibility across tools and libraries is required as well if migration is to be seamless. This is achieved using the Cortex Microcontroller Standard (CMSIS).

Developed in conjunction with silicon, tools, and middleware vendors, CMSIS is an abstraction layer that ensures off-the-shelf code has consistent software interfaces throughout the tool chain so that code can interoperate with an RTOS, other libraries, and the development environment. By serving as a standard interface between the Cortex core and C language, CMSIS provides a consistent structure for systems (see Figure 2).

This consistency is the key to enabling the future migration of code between MCUs. Core-specific details—such as interfacing to internal peripherals like the NVIC, internal core-based control and status registers, and special assembly language instructions translated to a C macro—have been abstracted to make using an MCU’s peripherals transparent to developers. For example, in the case of the CMSIS DSP library header files, a common C function interface is defined. This allows developers

to use the same named function calls to execute DSP instructions regardless of whether they are using a Cortex-M4-based STM32 F4 with hardware FPU and SIMD capabilities or a Cortex-M3-based STM32 F1 where DSP functions are emulated in software.

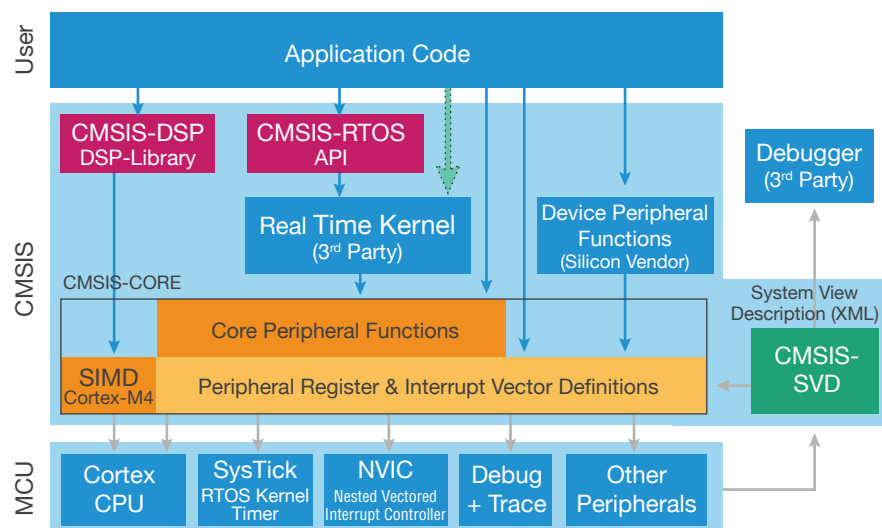
The ST Standard Peripheral Library performs the same function as CMSIS with the peripherals that are unique to the STM32 architecture by maintaining a common function call interface across all STM32 MCUs. For example, developers don’t have to worry about the register differences between the UART Status register on a STM32 L1 and the equivalent register on a STM32 F4. The ST Peripheral Library uses an abstraction layer that separates the programmer from the register details by handling these details at the lower device-specific driver level. Together, the CMSIS Core library and ST Standard Peripheral Library (see below) enable portability up and down the STM32 product families.

Access to the RTOS is simplified using the CMSIS RTOS library, which comprises a standard API for use by RTOSes to enable interoperability among

a wide range of software frameworks, middleware, and libraries. In effect, developers can be assured that CMSIS-compliant tools and middleware will interoperate seamlessly with the RTOS. For example, CMSIS communicates using the message/mail passing system of RTOSes like Keil’s RTX real-time operating system to enable the debugger to offer RTOS-aware visibility into the system.

For simplified debugging, the CMSIS SVD Library provides a System View Description (SVD) layer that provides consistent visibility between the debugger, peripherals, and CMSIS CORE layer. For example, when debuggers have peripheral awareness, developers can debug peripherals from a functional level. Similarly, there are differences between the Flash architectures of the STM32 F2 and STM32 F1. With the various CMSIS layers in place, developers need not concern themselves with the individual differences between the Flash architectures when debugging their application.

The final piece of CMSIS is the CMSIS DSP Library.



**Figure 2** The Cortex Microcontroller Standard (CMSIS) is an abstraction layer that ensures off-the-shelf code has consistent software interfaces throughout the tool chain to make low-level implementation details transparent to developers as well as simplify future migration of code between MCUs.

# Leading Embedded Development Tools



The complete development environment and middleware solution for STM32 Cortex™-M processor-based devices

**ARM**

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

**KEIL**  
Tools by ARM

## Digital Signal Processing

Advanced digital signal processing capabilities are required for many consumer electronics, industrial automation, medical, and military applications. The extended DSP and FPU capabilities of the Cortex-M4 within the STM32 F4 make the need for a separate DSP unnecessary. Not only is a single-processor system more cost effective, it avoids all of the partitioning and synchronization issues associated with multi-core designs.

The CMSIS DSP Library provides the underlying building blocks required to support applications ranging from motor control to low-power handheld medical instruments to consumer audio. With more than 80 algorithms, from complex arithmetic and vector operations to various filters and transforms, the CMSIS DSP Library accelerates initial product design by enabling developers used to programming an MCU to immediately utilize DSP capabilities. Implemented using both fixed and floating point functions, developers can also easily tradeoff between precision and performance

based on the application requirements of a system at different price points using different MCUs. Provided as C source code so that algorithms can be optimized by the compiler for performance or size, the CMSIS DSP library is available free of charge.

Among the many reasons to use the CMSIS DSP library is its portability. CMSIS-based code is portable across all STM32 families, enabling simple migration of DSP functionality between MCUs without developers having to rewrite any complex algorithmic code. This means that migration of even higher-level algorithms, such as various audio and video decode functions, is easily managed when these algorithms are CMSIS-compliant.

## Peripheral Compatibility

The ST Standard Peripheral Library offers a complete software interface and firmware to keep user code independent from underlying hardware details. Providing developers with an initial framework upon which to build their application speeds overall time-to-market for new and existing designs and simplifies future migration.

To facilitate migration, peripherals are effectively the same across the different STM32 families. This consistency is at both the hardware and software level. From a hardware perspective, peripheral pin-outs are typically the same between MCUs with the same package type, even those from different families. In terms of software, peripherals have been given a common software interface regardless of which MCU is in use. Effectively, code designed to work with a specific peripheral will be compatible across the various STM32 MCUs.

The peripheral library achieves this by generating code that is CMSIS-compliant. This also means that various tools, such as the MDK-ARM development environment from Keil, directly support the libraries enabling, for example, the debugger to be peripheral-aware.

The ST Peripheral Configuration tool also supports a project having multiple targets. The fact that each STM32 MCU has a common framework means that developers can easily carry a design from one MCU to another. For example, if a

developer wants to create a low-power version of an existing design using the STM32 L1, changing out the framework takes care of a significant portion of the migration process. The same application code base can be used for both designs by using different peripheral configurations. This approach greatly simplifies design by managing configuration differences as well as eliminating the need to maintain distinct versions of code which can quickly diverge and create additional code management issues.

### Migrating Between Families

Each of the four STM32 MCU series offers a range of performance, peripherals, and memory configurations to provide the optimal mix of capabilities and power consumption for an application at the lowest cost. As a consequence, each family has distinct differences in how they are architected. For example, the STM32 F4 family of MCUs is designed to provide the highest performance with excellent power consumption. The STM32 L1 family, in contrast, provides

ultra-low power efficiency with excellent performance. To achieve these different goals, however, there are internal differences between the MCU architectures.

minimizing the number of pins that may need to be rerouted. With the various STM32 F2 and STM32 F4 families, GPIO are mapped to the internal AHB bus for better performance. To speed

In terms of software, peripherals have been given a common software interface regardless of which MCU is in use. Effectively, code designed to work with a specific peripheral will be compatible across the various STM32 MCUs.

ST has designed the STM32 architecture to minimize the effort required by developers to migrate between the different STM32 MCU families while still achieving maximum performance and power efficiency. In general, each of the four families of STM32 MCUs maintains close compatibility with the others.

At a hardware-level, the power and functionality of each device have been designed to be pin-to-pin compatible with other devices in same package size,

layout and keep board size down, I/O pins can be mapped to different peripherals using a multiplexing mechanism which prevents conflicts between peripherals sharing the same pin. This gives developers flexibility in placing interfaces as well as simplifies remapping of peripherals when migrating between devices.

Using the ST Standard Peripherals Library, peripheral firmware can be updated by reconfiguring the peripheral library for the new MCU.

For example, a new Flash architecture was implemented between the STM32 F1 and STM32 F2 to improve system performance by using a more efficient interface, employing sectors instead of pages, and offering three read protection levels with JTAG fuse. The peripheral driver libraries capture these differences to make the transition seamless from a system perspective. Application code can be quickly migrated by updating the appropriate Flash function calls with those for the new MCU.

For an in-depth description of the migration process between the various STM32 MCU families, individual application notes are available describing specific migration details such as moving between the STM32 F1 and STM32 L1 architectures. Note that reviewing the migration process is well-worth the time for developers who are working on their first STM32 design and not yet concerned with migration. Developers considering migrating between processors will want to begin with application note AN3364 which describes the general migration and compatibility guidelines and lists the most

important aspects that need to be addressed during design migration. By understanding the key issues now that will need to be addressed when migration becomes important, developers can anticipate and design for compatibility from the start, thus enabling them to take full advantage of the migration capabilities of the STM32 architecture and its development tools.

### Tool Compatibility

Because the CMSIS libraries are not vendor specific, they enable developers to use their development tools of choice. An extensive selection of application-specific software libraries and middleware components have been designed for Cortex-based MCUs, giving developers a broad range of software options to choose from. In addition, many of these libraries and components have been optimized for the STM32 architecture, specifically taking advantage of its DMA architecture and application-specific peripherals. Combined with advanced development tools to manage code and facilitate reuse, developers have access to a strong software

ecosystem to accelerate development of both new and existing designs.

Just as MCU architectures offload processing from the CPU through integrated application-specific capabilities implemented in hardware, embedded development tools offload part of the design burden from engineers through enhanced capabilities. The embedded developer's primary tools—the compiler and debugger—are so efficient that code can be written in C rather than an MCU's unique assembly language. These tools can also automatically optimize code. In addition, access to performance and power profiling tools greatly simplify system optimization of complex systems. This enables developers to fully exploit the many real-time capabilities of the STM32 microcontroller architecture, including its advanced timing mechanisms, dynamic frequency and voltage scaling, multiple DMAs, 3-phase motor control timer, and cryptographic engine.

Each STM32 MCU family is also supported by a complete range of high-end and low-cost evaluation, software, debugging,

and programming tools. These tools include integrated development environments such as MDK-ARM from Keil that easily integrate with middleware such as IP stacks and libraries from other vendors to provide developers with everything they need to design, manage, and migrate applications between different MCUs.

The ability to transparently migrate code between processors is an important aspect of bringing next-generation systems to market quickly and easily by reusing existing hardware and software IP. Consistency across the entire STM32 portfolio—from the ultra-low power STM32 L1 through the STM32 Value Line for cost-sensitive applications to the high-performance STM32 F4—gives developers a broad portfolio of compatible devices that enable product line extensions without having to substantially rewrite code or redesign hardware. By abstracting low-level implementation details using CMSIS-compliant libraries, middleware, and tools, migrating between different STM32 MCUs can be as simple as recompiling the system. 🦋